Capstone Project: Dry Bean Dataset

Author :
Angel Enrique MEDINA PEREZ

Year : 2024

Introduction

This report aims to put into practice the knowledge acquired during the course of the MLOps bootcamp, through a small but enriching exercise that emulates the deployment of an ML pipeline in a professional environment. First, a dataset (Dry Bean Dataset) will be analyzed through an Exploratory Data Analysis. Subsequently, we will explore the challenges and questions that can be addressed and solved by leveraging ML models. Python will act as the main technological conduit to perform our analysis and execution tasks.

Contents

Chapter 1

Exploratory Data Analysis

In this first phase we will focus on knowing the characteristics of our data, such as its type, the completeness of each category, finding possible outliers and finally exploring possible correlations.

1.1 Loading the data

The data will be loaded from the $xlsx$ file found in the dataset repository.

```
import pandas as pd
import numpy as np
import seaborn as sb
import matplotlib.pyplot as plt
import math
%matplotlib inline

file_path = "data/DryBeanDataset/Dry_Bean_Dataset.xlsx"
data = pd.read_excel(file_path)
```

1.2 Dataset characteristics

Some of the characteristics of the dataset that were analyzed will be summarized below. The goal is to find possible records with missing columns or null values and verify that the data types are appropriate for the next steps.

1.2.1 Shape

Rows: 13611, Columns: 17

1.2.2 Missing values

No missing values were found in this dataset after executing:

```
data.isnull().sum()
```

1.2.3 Data types

The following data types were obtained by executing:

```
data.dtypes
```

There are 17 features, of which 16 are numerical and 1 is a string, representing the class of the row.

Area	int64
Perimeter	float64
MajorAxisLength	float64
MinorAxisLength	float64
AspectRation	float64
Eccentricity	float64
ConvexArea	int64
EquivDiameter	float64
Extent	float64
Solidity	float64
roundness	float64
Compactness	float64
ShapeFactor1	float64
ShapeFactor2	float64
ShapeFactor3	float64
ShapeFactor4	float64
Class	object

Table 1.1: Data types obtained with $describe$ method.

1.2.4 Summary

This dataset is composed by 13,611 entries and 17 features, of which 16 are numerical and 1 is categorical (string). It also does not include any Null value. Therefore, no transformation to the dataset is required at this point of the project.

1.3 Distribution

As can be seen in the histogram below, some of the numerical distributions have long tails, which implies that the data are loaded on one of the extremes of the range. In some cases, it is even observed that there are two modes, which probably indicates that some bean classes are different from the rest in that particular feature.

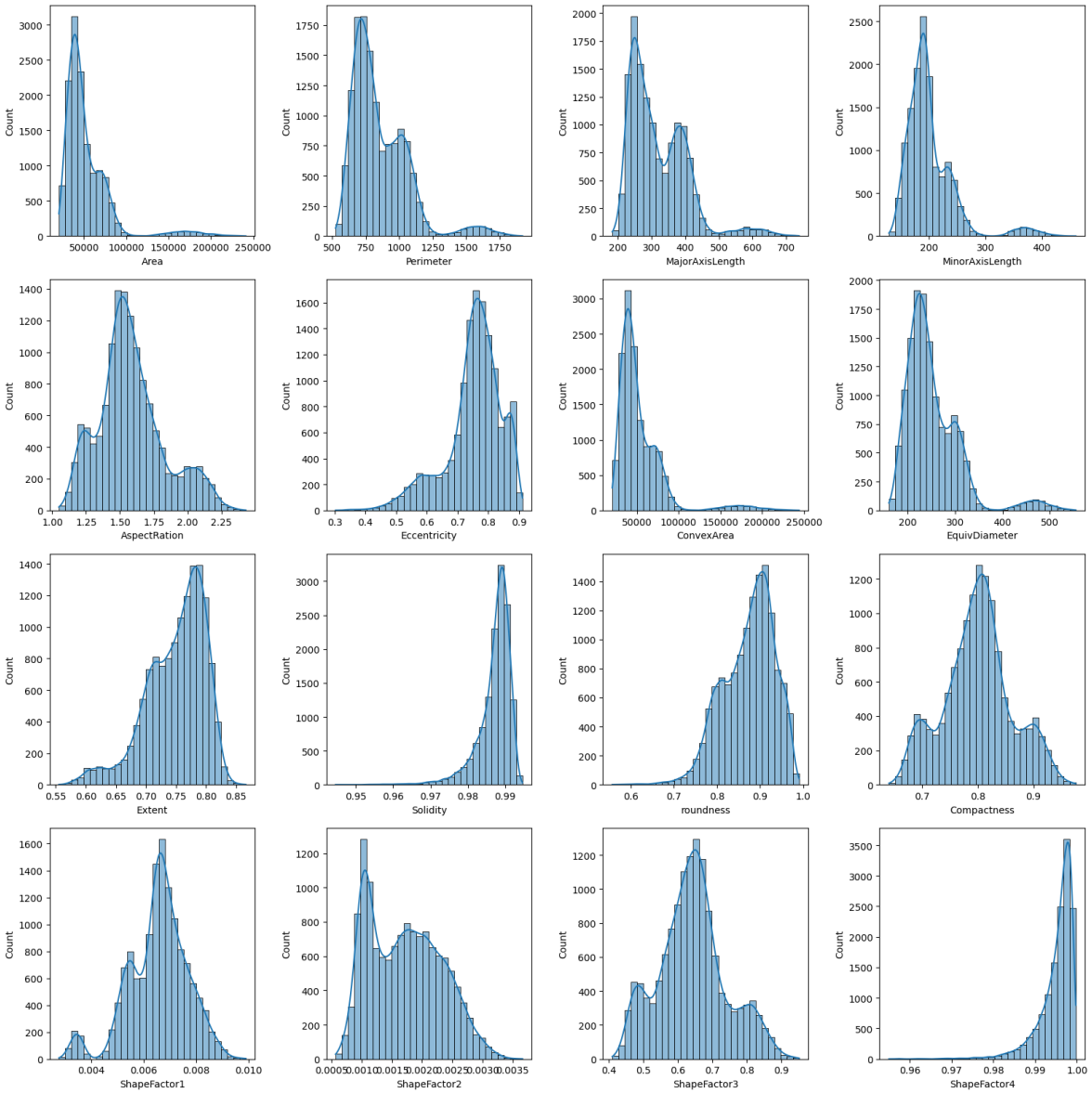


Figure 1.1: Histogram per feature.

1.4 Possible Outliers

In this stage, we will try to identify possible outliers in each of the features whose data type is $int64$ or $float64$. Those observation points distant from other observations can be visually identified using a box plot, where the minimum, first quartile, median, third quartile, and maximum values are drawn.

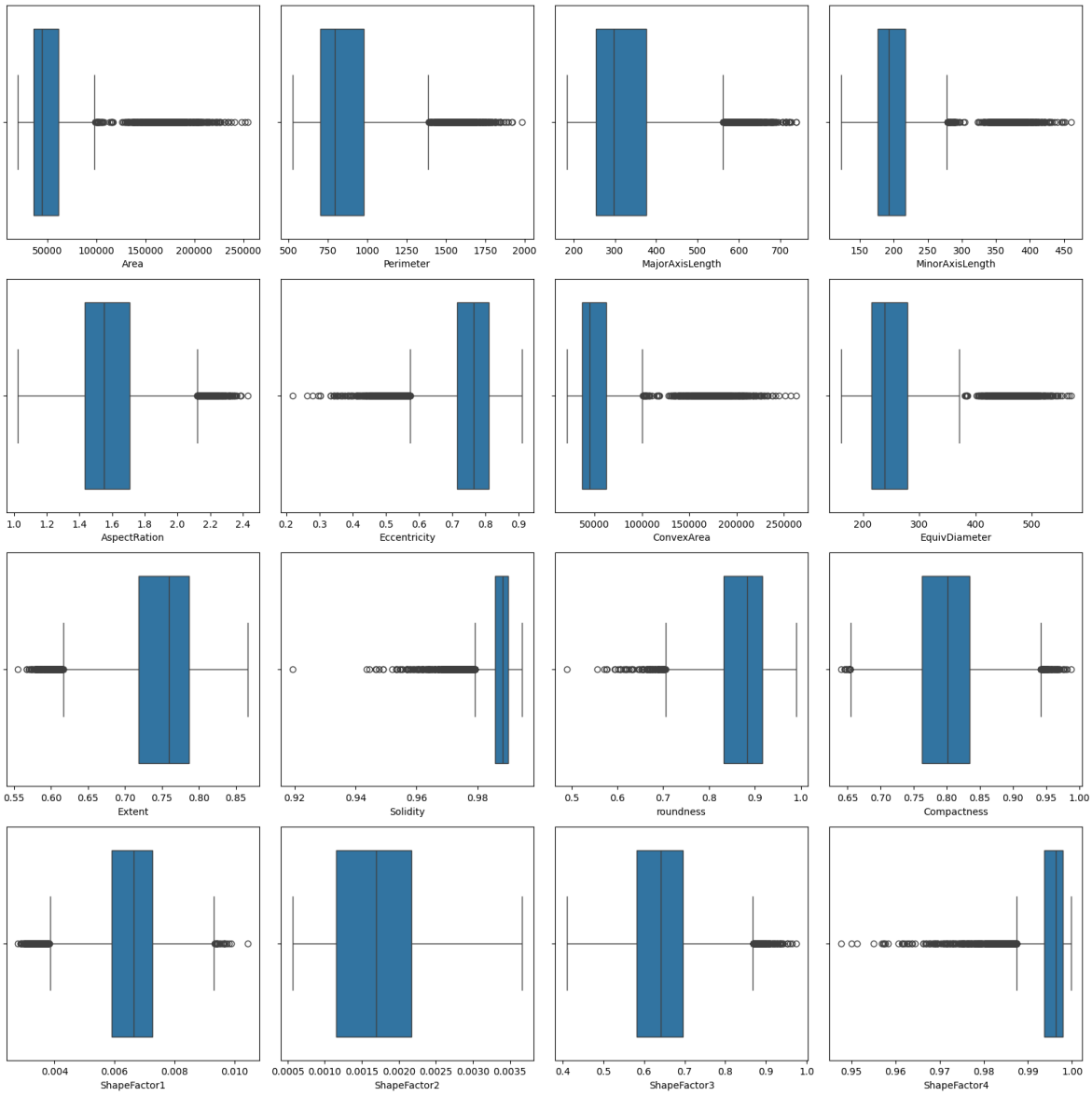


Figure 1.2: Box plot per category.

In order not to affect the size of the dataset or cause the loss of potentially valuable information, only those observations that are furthest from the set of the rest of the observations will be eliminated, even if this set exceeds the value $max = Q3 + 1.5IQR$. Based on the box plots and the dataset, the limits to consider for filtering outliers are:

- Eccentricity >= 0.3
- ConvexArea <= 250000
- Solidity >= 0.94
- Roundness >= 0.51
- ShapeFactor1 <= 0.0104
- ShapeFactor4 >= 0.954

1.5 Possible correlations

The correlations between features were analyzed thanks to a correlation matrix. Visually, a heatmap was used to denote the pairs of features with the highest correlations.

```
corr_matrix = data.corr(numeric_only=True)
sb.heatmap(corr_matrix, annot=True, cmap='YlOrBr')
plt.show()
```

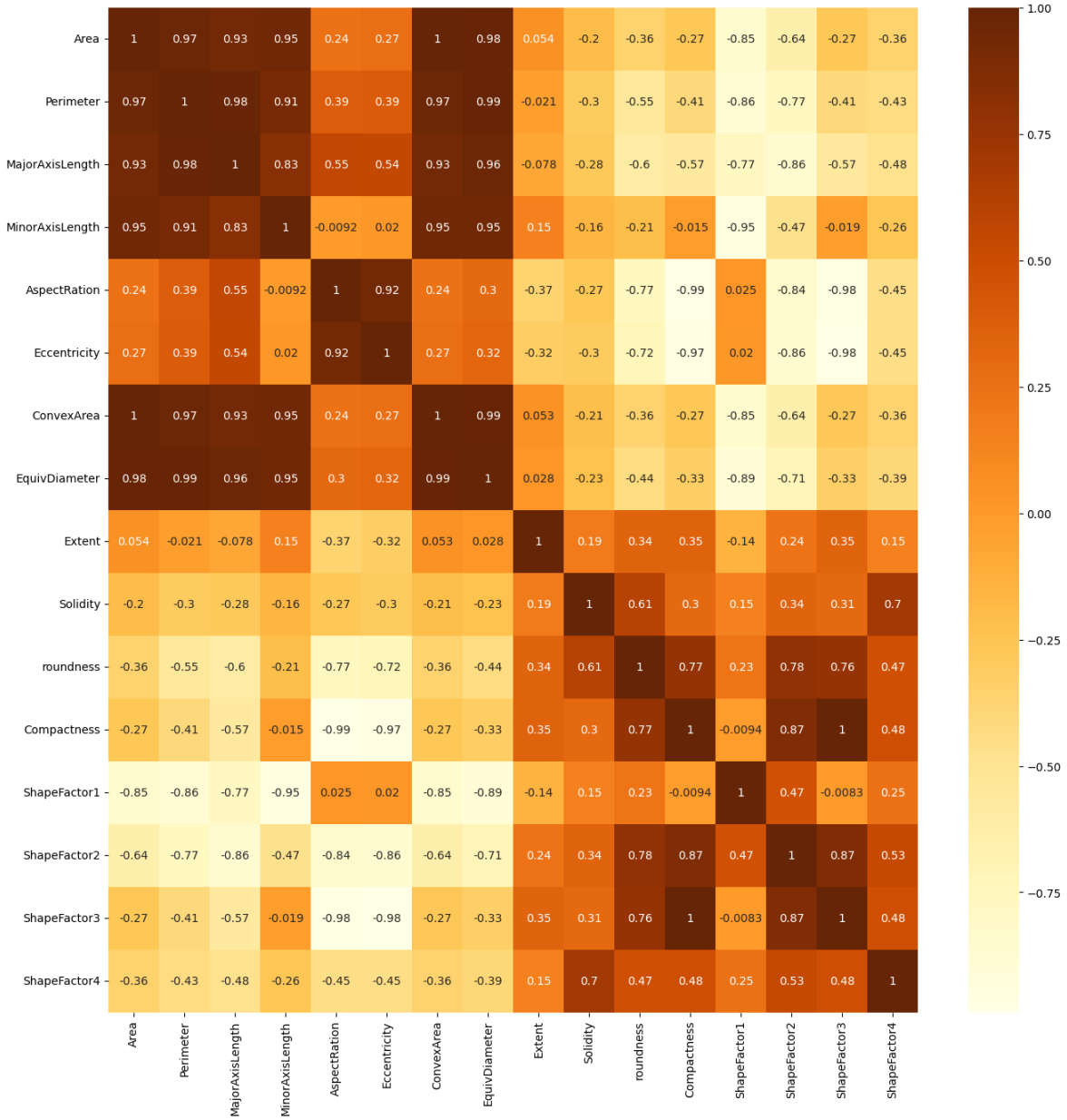


Figure 1.3: Correlation heatmap.

As can be seen in the previous results, there are several highly related features, mainly due to the fact that these are characteristics that describe the physical appearance of the bean, such as area and perimeter. Likewise, those physical characteristics in which bean dimensions are

involved present a high correlation, such as the duples $(Area|Perimeter, MajorAxisLength)$, $(Area|Perimeter, MinorAxisLength)$, $(Area|Perimeter, EquivDiameter)$, $(Area|Perimeter, ConvexArea)$, etc.

Said the above, $Area$ and $Perimeter$ will be dropped from the train dataset as removal of correlated features helps improve a model's ability to generalize to new data, which reduces the risk of overfitting and makes the model more robust. Similarly, $Compactness$ and $ShapeFactor3$ present a high correlation between them ($correlation = 1$), so both features will be removed from the train dataset.

1.6 Data Cleaning

At this stage, the data will be cleaned acoording to the results of the EDA.

```
# Removing outliers

initial_rows = data.shape[0]

data = data[data['Eccentricity'] >= 0.3]
data = data[data['ConvexArea'] <= 250000]
data = data[data['Solidity'] >= 0.94]
data = data[data['roundness'] >= 0.51]
data = data[data['ShapeFactor1'] <= 0.0104]
data = data[data['ShapeFactor4'] >= 0.954]

final_rows = data.shape[0]

print(f'Initial rows = {initial_rows}, final rows = {final_rows},
    difference = {initial_rows - final_rows}')

>> Result: Initial rows = 13611, final rows = 13599, difference = 12.
```

A total of 12 rows were removed from the dataset, which does not represent a number of observations that could be detrimental to the model, considering the initial size of the dataset.

Chapter 2

Problem definition

In Mexico, common beans are the second most produced and consumed crop after maize[1]. During recent years, attempts have been made to develop policies and methods that improve the competitiveness and productivity of beans in the Mexican national territory[2]. However, it is interesting to explore the possibility of using machine learning techniques to improve some quality criteria in the cultivation and harvesting of beans. The classification of bean types thanks to the extraction of their physical characteristics and the use of computer vision systems becomes relevant in the context of quality control.

Bean classification allows bean seeds to be separated based on quality, size, shape and other attributes. This ensures that the seeds used for sowing are in optimal condition and have a high probability of germination and healthy plant development. High quality seeds tend to produce more uniform crops with higher yields.

Bean classification can also help identify and remove weed seeds or other impurities that might be present in the batch. This is crucial to prevent the spread of weeds and diseases that could compete with crops or reduce their yield.

In summary, dry bean classification is essential to ensure optimal crop quality and yield, as well as to minimize problems associated with weeds, diseases and uneven planting.

Having said the above, the problem to be addressed will consist of implementing a dry bean classifier that can be deployed in an agricultural production context, aimed at increasing the quality metrics of the bean crop and harvest.

2.1 Why MLOps for this scenario?

MLOps ensures consistency, availability and standardization of data throughout the entire design, implementation, testing, monitoring and life cycle management [3]. On the one hand, ML models depend on data quality, availability and relevance. After performing the EDA, it is clear that our dry bean dataset is a good candidate to be implemented in an MLOps pipeline; it is sufficient and quality data to be subjected to an ML process. On the other hand, although it is true that the success of the model will depend largely on the reliability and robustness of the computer vision system that reports the observations of the beans, it will be assumed that it works adequately for this exercise.

As established in the previous section, the main objective of this ML model implementation exercise is to improve the quality and effectiveness with which beans are classified, and of course to automate the classification process, which in an agricultural production context , can be laborious and subject to human error.

Chapter 3

Architecture

The architecture to follow for this problem will be online and divided into 5 stages, EDA, Data Preparation, Training, Source and version control and Running Environment. The idea will be to do the EDA with the provided dataset and in the same way feed the Data Preparation process with this same data source (in case a re-training is required without going through the EDA stage). On the other hand, when training a model, there is the possibility of performing iterative hyper parameter tuning until the model meets the required validation. Subsequently, the model is packaged, versioned and saved in the common repository and then deployed together with the service that will host the model. Once the prediction service has been deployed in a running environment, client applications can consume this service and obtain predictions. In parallel, a monitoring service will be responsible for collecting the necessary metrics to determine if the model needs retraining.

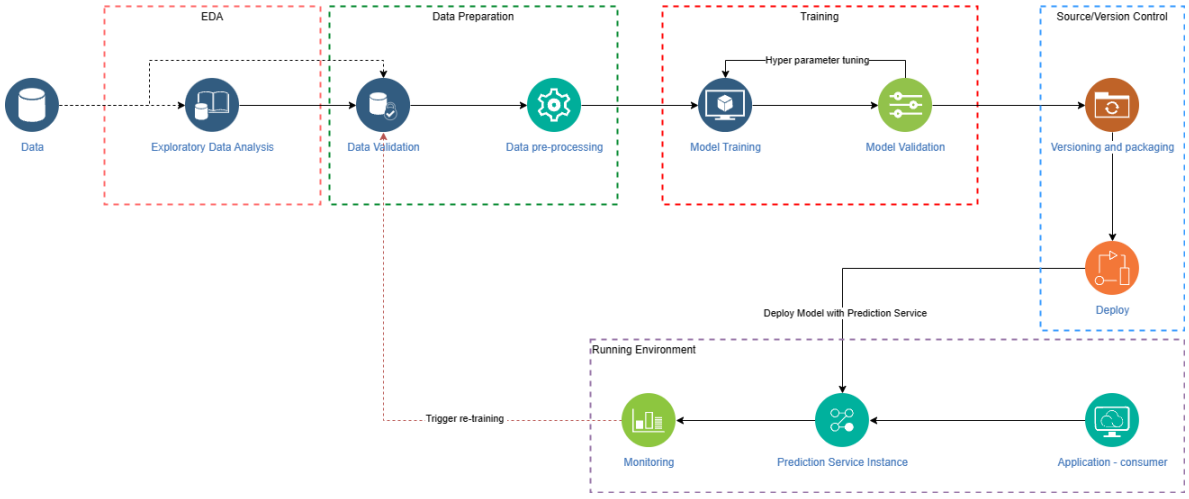


Figure 3.1: Proposed pipeline architecture.

Chapter 4

Choosing a model

4.1 Data Pre-processing

At this stage, the data will be cleaned and prepared according to the results of the EDA.

4.1.1 Removing Outliers

In this step, the objective is to remove those values that, according to the EDA, are outside the normal range of values of the sample set. The following code fragment shows the ranges of values that will be removed from the general dataset for each feature.

```
# Removing outliers

initial_rows = data.shape[0]

data = data[data['Eccentricity'] >= 0.3]
data = data[data['ConvexArea'] <= 250000]
data = data[data['Solidity'] >= 0.94]
data = data[data['roundness'] >= 0.51]
data = data[data['ShapeFactor1'] <= 0.0104]
data = data[data['ShapeFactor4'] >= 0.954]

final_rows = data.shape[0]

print(f'Initial rows = {initial_rows}, final rows = {final_rows},
    difference = {initial_rows - final_rows}')

>> Result: Initial rows = 13611, final rows = 13599, difference = 12.
```

A total of 12 rows were removed from the dataset, which does not represent a number of observations that could be detrimental to the model, considering the initial size of the dataset.

4.1.2 Data split

Dataset will be split into random train and test subsets. A 80-20 ratio is used in this dataset split. By using 80% of the data to train the model and the remaining 20% to evaluate it, a balance can be achieved between having enough data for the model to learn relevant patterns (training) and having enough data to evaluate the performance of the model on unseen data (test).

```
features = data.drop(columns=['Class']).columns

X = data[features]
y = data['Class']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    train_size=0.8, random_state=None, shuffle=True)
```

4.1.3 Scaling

Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data [4].

The StandardScaler from sklearn is implemented in this exercise in order to standardize features by removing the mean and scaling to unit variance.

The standard score of a sample x is calculated as $z = (x - u)/s$, where u is the mean of the training samples, and s is the standard deviation of the training samples.

Since StandardScaler is sensitive to outliers, and the features may scale differently from each other in the presence of outliers, these where removed in the previous section.

```
columns: list[str] = X_train.columns.to_list()

standard_scaler = StandardScaler()
X_train[columns] =
    pd.DataFrame(standard_scaler.fit_transform(X_train[columns]),
    index=X_train.index)
X_test[columns] =
    pd.DataFrame(standard_scaler.fit_transform(X_test[columns]),
    index=X_test.index)
```

4.1.4 Removing correlated features

As stated in the EDA, $Area$ and $Perimeter$ as well as $Compactness$ and $ShapeFactor3$ as these features present a high correlation between them, so the 4 features will be removed from the train dataset.

```
# Removing correlated features
dropped_columns = ['Area', 'Perimeter', 'Compactness', 'ShapeFactor3']
X_train = X_train.drop(columns=dropped_columns)
X_test = X_test.drop(columns=dropped_columns)
```

4.1.5 Models

The list of models to be tested after the data pre-processing can be found here below. These classification models where chosen according to the following criteria: the dry bean dataset is a small dataset with around 13K records and there is a relatively low number of features.

```
models = {
```

```
  "LogisticRegression": LogisticRegression(max_iter=1000),
  "DecisionTreeClassifier": DecisionTreeClassifier(),
  "SVC": SVC(max_iter=1000),
  "LinearSVC": LinearSVC(max_iter=1000, dual='auto'),
  "KNeighborsClassifier": KNeighborsClassifier(),
  "RandomForestClassifier": RandomForestClassifier()
}
```

The next step is to train each of the selected models. For this, we iterate over our list of models and train them using the split training datasets (X_train and y_train).

```
for name, model in models.items():
    model.fit(X_train, y_train)
```

The last step is to measure the performance of each trained model. For this, the classification of our X_test set will be predicted and the accuracy of each model will be calculated, comparing the predicted class and the expected class (y_pred and y_test). The model to be chosen will be the one with the best accuracy.

```
# Model Performance

print("Model Performance:")
for name, model in models.items():
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    print(f"{name}:")
    print(classification_report(y_test, y_pred))
    disp = ConfusionMatrixDisplay.from_predictions(y_true=y_test,
        y_pred=y_pred)
    disp.ax_.set_title(name)

plt.show()
```

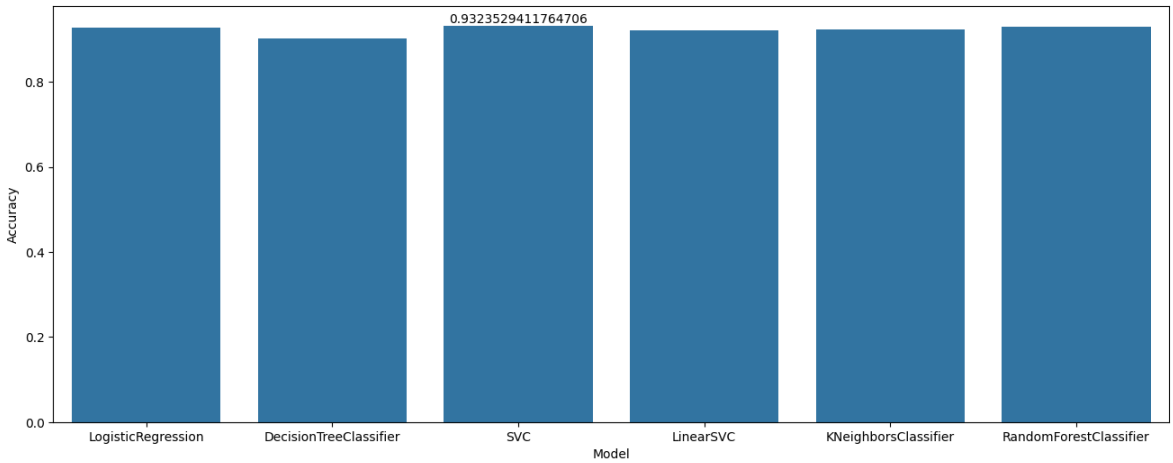


Figure 4.1: Model performance.

According to the results, the model that obtained the best score was SVC, so it will be the

model chosen to carry out the next step of this exercise. Below is the confusion matrix for the SVC model.

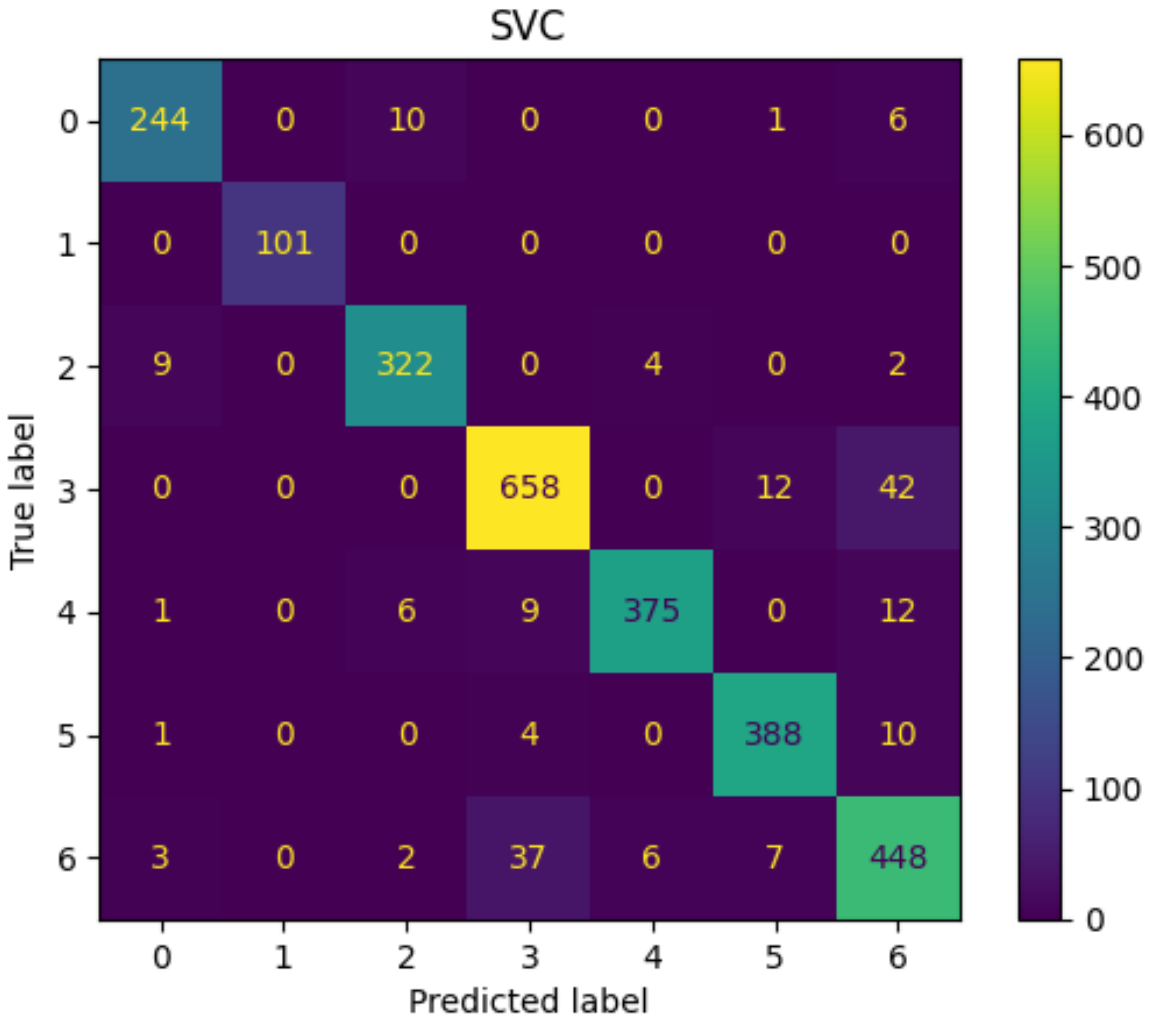


Figure 4.2: SVC Confusion Matrix.

Chapter 5

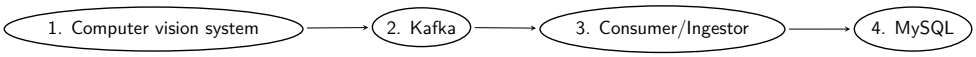
Developing the pipeline

This project involves the development of a machine learning pipeline designed for real-time data ingestion and retrieval using a database. The chosen architecture enables efficient data management and supports the training and evaluation of machine learning models. The development process followed object-oriented programming (OOP) principles to modularize the system's main components, including database connection, data reporting, and model training and evaluation.

5.1 System architecture

5.1.1 MySQL Database

For real-time data ingestion and retrieval, MySQL was selected as the database management system (DBMS). MySQL offers robust performance and scalability, which are essential for handling the large volumes of data in machine learning projects, rather than using the provided CSV files in the public repositories of the datasets. The MySQL database will be responsible for storing raw data ingested in real-time from the computer vision system as stated in the problem definition, ideally through a Kafka topic.For the scope of this project, it is assumed that initial data is already stored in the database and that some Kafka consumer is responsible for new data intake.



For the purposes of the project, two $.sql$ files were generated (under */mysql-dumps* in the final project structure) and are responsible for creating the database and inserting the data (extracted from the Dry Beans repository) during the docker-compose process. This data will be subsequently used for training and evaluating machine learning models.
The created database **dry_bean** will contain an unique table **bean**. The definition of the **bean** table is the following:

```
CREATE TABLE `bean` (
    `id`                BIGINT auto_increment NOT NULL,
    `load_timestamp`    DATETIME NOT NULL,
    `area`              INT NOT NULL,
    `perimeter`         DECIMAL(7, 3) NOT NULL,
    `major_axis_length` DECIMAL(10, 7) NOT NULL,
    `minor_axis_length` DECIMAL(10, 7) NOT NULL,
    `aspect_ration`     DECIMAL(10, 9) NOT NULL,
```

```
    `eccentricity`      DECIMAL(10, 9) NOT NULL,
    `convex_area`       INT NOT NULL,
    `equiv_diameter`    DECIMAL(10, 7) NOT NULL,
    `extent`            DECIMAL(10, 9) NOT NULL,
    `solidity`          DECIMAL(10, 9) NOT NULL,
    `roundness`         DECIMAL(10, 9) NOT NULL,
    `compactness`       DECIMAL(10, 9) NOT NULL,
    `shape_factor_1`    DECIMAL(10, 9) NOT NULL,
    `shape_factor_2`    DECIMAL(10, 9) NOT NULL,
    `shape_factor_3`    DECIMAL(10, 9) NOT NULL,
    `shape_factor_4`    DECIMAL(10, 9) NOT NULL,
    `class`             VARCHAR(10),
    CONSTRAINT `pk_bean` PRIMARY KEY (`id`)
);
```

5.1.2 Python main components

An object-oriented programming (OOP) approach was adopted to develop the main components of the pipeline. This approach encapsulates functionality within classes and methods, facilitating code maintainability and extensibility. The main components can be found in **src/components**, and provide the core functionality of the project.

1. **MySQLQueryEngine** *metaclass* **Singleton**: The *MySQLQueryEngine* class handles the connection to the MySQL database. This class includes methods for establishing and closing connections, as well as executing SQL queries. This class implements the Singleton design pattern to provide a single instance of the database connection across the application.

2. **DataReporter**: The *DataReporter* class generates reports based on the data stored in the database. This class includes methods for retrieving specific data and generating descriptive projections.

3. **SVCModelTrainer** *extends* **AbstractModelTrainer**: The *SVCModelTrainer* class provides an implementation of the C-Support Vector Classification from sklearn.svm by managing the training and evaluation of models. This class includes methods for preparing data, training the model, and evaluating its performance. This class inherits its behavior from the abstract base class *AbstractModelTrainer*.

Code Documentation

To enhance code readability and maintainability, detailed comments have been added within the main classes and methods. Some of these comments are located within the abstract classes from which concrete implementations inherit, providing a clear explanation of the expected behavior of each method.

5.1.3 Additional components

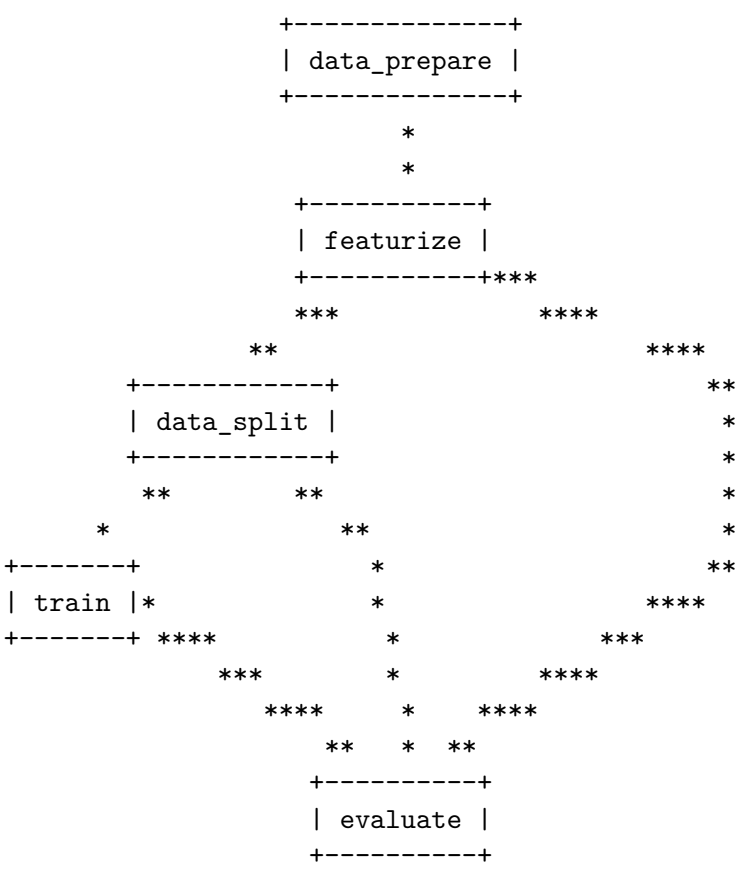
The project also includes auxiliary components designed to handle custom exceptions, configure logging, encapsulate properties, and define data types. These components enhance the robustness and maintainability of the system by providing clear error handling, consistent logging, and well-defined data structures.

5.2 Pipeline definition

A folder named pipeline has been added to the project, containing five files that outline each step required to execute an automated pipeline. The steps are *data_prepare*, *featurize*, *data_split*, *train*, and *evaluate*. Each step in the pipeline utilizes the main components mentioned earlier, such as database connection, data reporting, and model training and evaluation. This structure ensures that the pipeline is modular and maintainable. By organizing the pipeline in this manner, it is possible to effectively use a tool like DVC for data versioning, enhancing the reproducibility and traceability of the machine learning workflow.

The required parameters for each step where defined in the **params.yml** file, like *random_state*, *test_size*, *train_size*, *max_iter*, and the related paths to store intermediate data.

Additionally, the **dvc.yaml** file defines the steps and its dependencies and outputs. The complete stage dependency graph is as follows:



5.2.1 Integration with GDrive

The integration of Google Drive with DVC was chosen to create a repository for storing the models generated by the pipeline. Google Drive offers reliable and accessible cloud storage, making it an ideal choice for remote data and model storage. To configure Google Drive as a remote repository in DVC, the following steps were required:

DVC GDrive dependency installation with *pip install dvc-gdrive*.

Then, configure Google Drive as Remote with *dvc remote add -d myremote gdrive://<folder-id>*, where the *folder-id* is the identifier of the root folder created in Google Drive, ex. *1H34fOdxDOQeVa9F5db6_XIhkLrkjfIBg*.

The authentication was performed using a service account from Google Cloud. This service account provides a JSON key which can be assigned to a environment variable *GDRIVE CREDENTIALS DATA*. It was necessary to activate the *gdrive_use_service_account* flag with *dvc remote modify myremote gdrive_use_service_account true*.

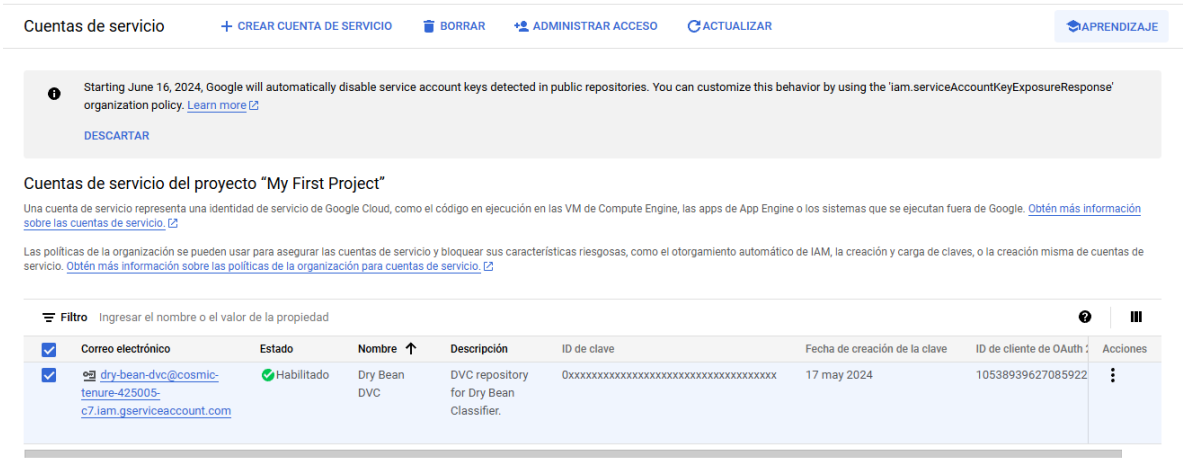


Figure 5.1: Service account configuration for the project.

At this point, we are able to push and pull data from DVC as we do locally.

with this, Google Drive was configured as a remote repository for DVC, ensuring that models and datasets are stored securely and can be easily accessed or shared, enhancing collaboration and reproducibility within the machine learning pipeline.

Figure 5.2: Google Drive folder with uploaded data from DVC.

5.3 Containerization

The entire project was containerized using Docker. A **docker-compose.yml** file was added, describing the two main services for the model training process: the MySQL service and the app service. In addition, a **Dockerfile** was included that performs the initialization of the DVC repository and installs the necessary dependencies thanks to the **requirements.txt** file. On the

other hand, during the startup of the MySQL service, the database creation and data load files are executed, which allows the service to be ready to respond to the requests performed. The following is an excerpt from the docker compose file.

```
services:
  db:
    image: mysql:8.0
    container_name: mysql_container
    restart: always
    environment:
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      ...
    ports:
      - "3306:3306"
    volumes:
      - ./mysql-dump:/docker-entrypoint-initdb.d

  app:
    build:
      context: ./
      args:
        APP_DATA_FOLDER: ${APP_DATA_FOLDER}
        ...

    container_name: ml-beans
    environment:
      - GDRIVE_CREDENTIALS_DATA=${GDRIVE_CREDENTIALS_DATA}
      ...
    ports:
      - "8088:8088"
    depends_on:
      - db
```

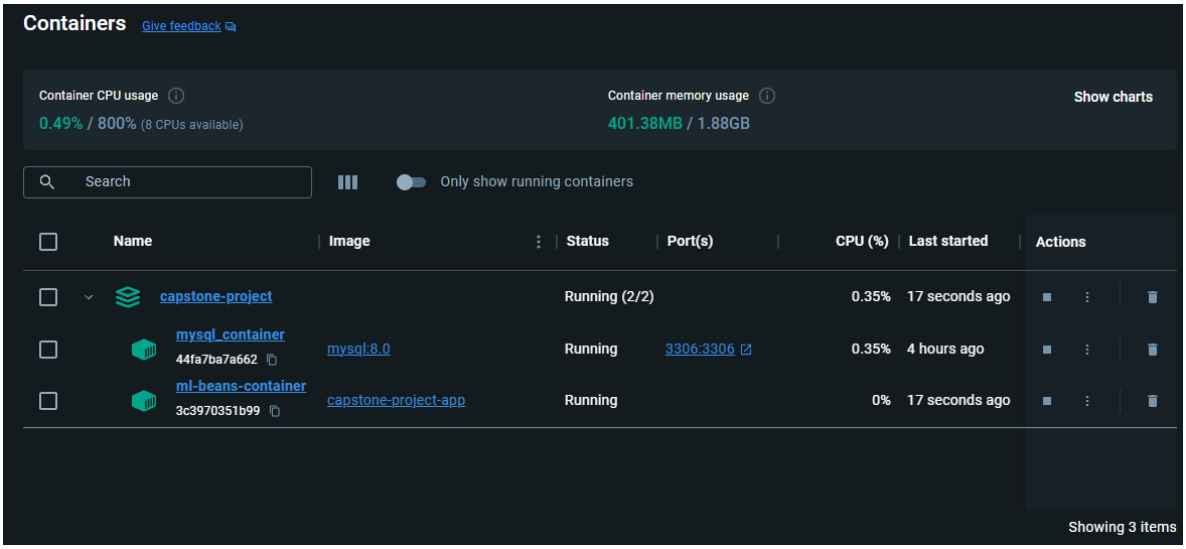


Figure 5.3: Docker containers in execution.

A partial parameterization was also performed to pass some values needed for the connection to the database and the credentials to establish the connection to the Google Drive repository. The values are taken from an **.env** file in the root folder.

Finally, the commands to start the services and access the containers were encapsulated inside a **makefile** for ease of retention and convenience, these include:

- make compose
- make run

An example of the logs obtained during the execution of the pipeline can be found in the annexes.

5.3.1 Link to the GitHub repository

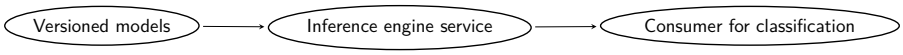
The complete code base can be found in the following GitHub repository: `https://github.com/angmedin/capstone-project`.

Chapter 6

Inference engine service

6.1 Service description

Thanks to the execution of the pipeline and the models generated and versioned with DVC in the previous chapter, it is possible to generate a service that works as an API for consumers of the prediction models we generated. Under the main idea of this project, some consumer in the bean production lines would make use of this inference service to classify beans in real time.



6.1.1 Technologies implemented

- **FastAPI**: A fast web framework for building APIs in Python.
- **DVC**: versioning and control of machine learning models generated by the pipeline.
- **Docker**: packaging, distributing, and running the service in any environment.

6.2 Development process

The inference service is developed using FastAPI to create a RESTful API that can receive requests and return inference results. The necessary endpoints are defined to perform the inferences, and the corresponding functions are implemented to process the requests and execute the inferences using the machine learning models.

The complete code base can be found in the following GitHub repository: `https://github.com/angmedin/inference-engine`.

6.2.1 Integration with DVC for Model Management

DVC is used to manage and version the machine learning models generated by another service. During the startup of the inference service, DVC is configured to load the necessary models from the remote repository and ensure they are available for making inferences.

6.2.2 Packaging into a Docker Container

A Dockerfile is created specifying how to build the runtime environment of the inference service. All necessary dependencies, including FastAPI, DVC, and any other required libraries for loading and executing the machine learning models, are included. The Docker image is built, and a container is created that can be deployed in any Docker-compatible environment.

6.2.3 Endpoints

A single endpoint was enabled to perform predictions, **/bean/predict/class**. This endpoints receives an object with the characteristics of the bean and returns the prediction.

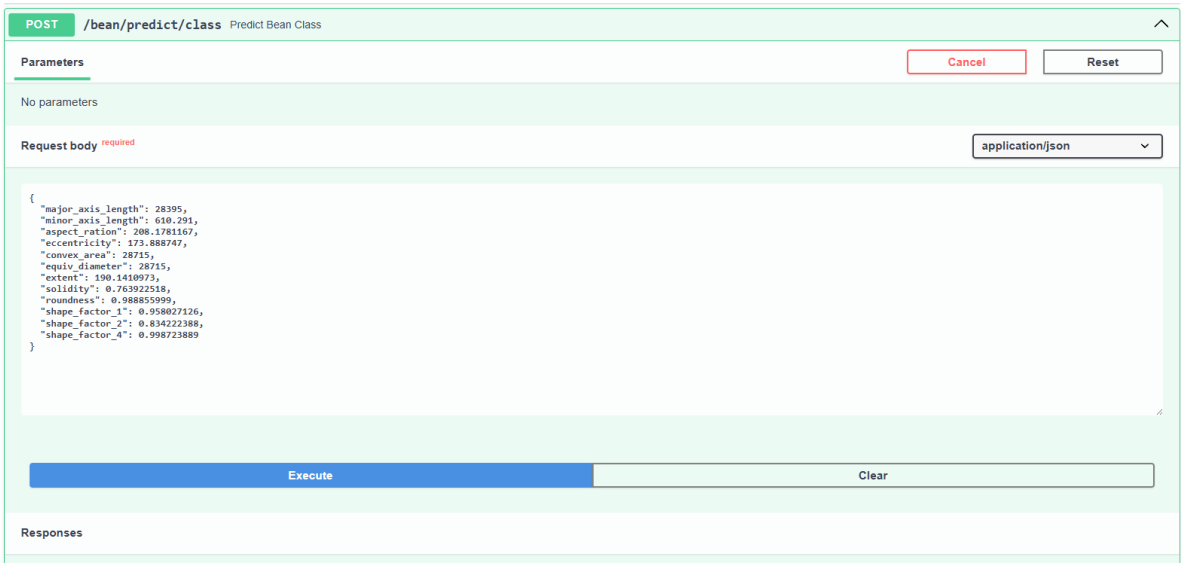


Figure 6.1: Endpoint enabled for predictions.

Figure 6.2: Response from the application.

6.2.4 Conclusion

The combination of FastAPI, DVC, and Docker provides a robust and scalable solution for creating an inference service in Python that utilizes machine learning models. This architecture allows for easy model management, a fast and secure API, and straightforward deployment in production environments. With this approach, efficient and flexible inference services can be developed to meet the needs of various applications and use cases.

Chapter 7

Final comments

In the current technology landscape, the implementation and use of machine learning pipelines are crucial to develop robust, scalable and maintainable ML solutions. The use of technologies such as Python, DVC and Docker significantly improves the efficiency and reliability of these pipelines.

Python is the backbone of ML development due to its extensive libraries and frameworks that support various stages of the ML lifecycle, from data preprocessing and model formation to deployment and monitoring. Its simplicity and readability make it an ideal choice for beginners and experienced professionals alike.

DVC plays a key role in managing the complexity of ML projects by enabling version control of data and models. It ensures reproducibility and transparency, allowing teams to track changes, collaborate effectively and revert to previous versions if necessary. This version control is essential for maintaining the integrity of the ML process, especially when working with large data sets and multiple iterations of model training.

For its part, Docker provides a consistent environment for development, testing and deployment. This containerization simplifies the deployment process and facilitates scalability, making it easier to manage variable workloads and integrate with cloud services.

Moreover, data monitoring and model drift is another critical aspect of maintaining the efficiency of ML processes. To ensure that ML models remain accurate and relevant, it is essential to continuously measure and address these deviations. Implementing drift detection mechanisms helps to identify and mitigate performance degradation early.

Throughout this work, it was possible to briefly appreciate the use of the aforementioned technologies applied to solve a problem that may arise considering the current context of the industry. While it is true that there are still many other technologies and tools that are useful in the field of MLOps, this introduction represents a big step towards the implementation and improvement of current processes in the personal and business contexts in which the people participating in this course find themselves.

Appendix A

Model performance

A.1 Performance results

LogisticRegression:

precision	recall	f1-score	support	
0	0.94	0.93	0.94	261
1	1.00	1.00	1.00	101
2	0.93	0.96	0.95	337
3	0.93	0.92	0.92	712
4	0.97	0.92	0.95	403
5	0.95	0.96	0.95	403
6	0.86	0.88	0.87	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

DecisionTreeClassifier:

precision	recall	f1-score	support	
0	0.86	0.92	0.89	261
1	1.00	0.99	1.00	101
2	0.92	0.93	0.92	337
3	0.91	0.89	0.90	712
4	0.96	0.89	0.93	403
5	0.94	0.95	0.95	403
6	0.82	0.85	0.83	503
accuracy	0.90	2720		
macro avg	0.92	0.92	0.92	2720
weighted avg	0.90	0.90	0.90	2720

SVC:

precision	recall	f1-score	support	
0	0.95	0.93	0.94	261
1	1.00	1.00	1.00	101
2	0.95	0.96	0.95	337
3	0.93	0.92	0.93	712
4	0.97	0.93	0.95	403
5	0.95	0.96	0.96	403
6	0.86	0.89	0.88	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

LinearSVC:

precision	recall	f1-score	support	
0	0.95	0.91	0.93	261
1	1.00	1.00	1.00	101
2	0.94	0.96	0.95	337
3	0.93	0.90	0.91	712
4	0.97	0.92	0.95	403
5	0.96	0.95	0.96	403
6	0.81	0.89	0.85	503
accuracy	0.92	2720		
macro avg	0.94	0.93	0.93	2720
weighted avg	0.92	0.92	0.92	2720

KNeighborsClassifier:

precision	recall	f1-score	support	
0	0.93	0.93	0.93	261
1	1.00	1.00	1.00	101
2	0.94	0.95	0.94	337
3	0.91	0.92	0.92	712
4	0.97	0.92	0.94	403
5	0.95	0.95	0.95	403
6	0.85	0.87	0.86	503
accuracy	0.92	2720		
macro avg	0.94	0.93	0.93	2720
weighted avg	0.92	0.92	0.92	2720

RandomForestClassifier:

precision	recall	f1-score	support	
0	0.94	0.91	0.93	261
1	1.00	1.00	1.00	101
2	0.93	0.96	0.94	337
3	0.92	0.94	0.93	712
4	0.97	0.93	0.95	403
5	0.95	0.96	0.95	403
6	0.88	0.88	0.88	503
accuracy	0.93	2720		
macro avg	0.94	0.94	0.94	2720
weighted avg	0.93	0.93	0.93	2720

Appendix B

DVC Pipeline

B.1 DVC run exp logs

```
Reproducing experiment 'muggy-rant'
Building workspace index

    |10.0 [00:00, 269entry/s]
Comparing indexes

    |11.0 [00:00, 1.57kentry/s]
Applying changes

    |0.00 [00:00,   ?file/s]
Running stage 'data_prepare':
> python -m src.pipeline.p1_data_prepare --config=params.yaml
31-05-2024 08:58:31 - INFO - p1_data_prepare:18 [MainThread] : Stage 1: data
    preparation.
31-05-2024 08:58:32 - INFO - p1_data_prepare:26 [MainThread] : Fetching data.
31-05-2024 08:58:33 - INFO - p1_data_prepare:35 [MainThread] : CSV file
    created successfully in data/processed/prepare_beans.csv.

Running stage 'featurize':
> python -m src.pipeline.p2_featurize --config=params.yaml
31-05-2024 08:58:47 - INFO - p2_featurize:18 [MainThread] : Stage 2:
    featurize.
31-05-2024 08:58:47 - INFO - p2_featurize:23 [MainThread] : Reading prepared
    dataset.
31-05-2024 08:58:47 - INFO - p2_featurize:28 [MainThread] : Dropping unused
    columns.
31-05-2024 08:58:47 - INFO - p2_featurize:31 [MainThread] : Encoding
    categorical feature.
31-05-2024 08:58:48 - INFO - p2_featurize:38 [MainThread] : CSV file created
    successfully in data/processed/featured_beans.csv.
31-05-2024 08:58:48 - INFO - p2_featurize:43 [MainThread] : Encoder file
    created successfully in encoder/encoder.joblib.

Running stage 'data_split':
```

```
> python -m src.pipeline.p3_data_split --config=params.yaml
31-05-2024 08:58:50 - INFO - p3_data_split:17 [MainThread] : Stage 3: data
    split.
31-05-2024 08:58:50 - INFO - p3_data_split:22 [MainThread] : Reading
    featurized dataset.
31-05-2024 08:58:50 - INFO - p3_data_split:28 [MainThread] : Data split.
31-05-2024 08:58:50 - INFO - p3_data_split:33 [MainThread] : Feature scaling.
31-05-2024 08:58:50 - INFO - p3_data_split:38 [MainThread] : Trying to save
    train and test datasets.
31-05-2024 08:58:51 - INFO - p3_data_split:48 [MainThread] : CSV files
    created successfully in data/processed/train_beans.csv and
    data/processed/test_beans.csv.
Updating lock file 'dvc.lock'

Running stage 'train':
> python -m src.pipeline.p4_train --config=params.yaml
31-05-2024 08:58:53 - INFO - p4_train:19 [MainThread] : Stage 4: train.
31-05-2024 08:58:53 - INFO - p4_train:24 [MainThread] : Load train dataset.
31-05-2024 08:58:53 - INFO - p4_train:29 [MainThread] : Training model.
31-05-2024 08:58:54 - INFO - p4_train:32 [MainThread] : Trying to save model.
31-05-2024 08:58:54 - INFO - p4_train:38 [MainThread] : Model saved
    successfully in models/model.joblib.
31-05-2024 08:58:55 - INFO - p4_train:50 [MainThread] : Model tracked with
    DVC.
Updating lock file 'dvc.lock'

Running stage 'evaluate':
> python -m src.pipeline.p5_evaluate --config=params.yaml
31-05-2024 08:58:58 - INFO - p5_evaluate:19 [MainThread] : Stage 5: evaluate.
31-05-2024 08:58:58 - INFO - p5_evaluate:24 [MainThread] : Loading test
    dataset.
31-05-2024 08:58:58 - INFO - p5_evaluate:28 [MainThread] : Evaluating model.
31-05-2024 08:58:59 - INFO - p5_evaluate:39 [MainThread] : Saving metrics.
31-05-2024 08:58:59 - INFO - p5_evaluate:48 [MainThread] : CSV file created
    successfully in reports/metrics.json.
31-05-2024 08:58:59 - INFO - p5_evaluate:52 [MainThread] : Logging metrics.
Updating lock file 'dvc.lock'

Ran experiment(s): muggy-rant
Experiment results have been applied to your workspace.
```

B.1.1 Obtained metrics


```
{
    "accuracy": 0.9264705882352942,
    "macro": {
        "precision": 0.9395945585461837,
        "recall": 0.936281511660934,
        "f1-score": 0.93779613021304,
```

```
        "support": 2720.0
    }
}
```

Appendix C

References

1 Ramírez-Jaspeado, R., Palacios-Rojas, N., Nutti, M. R., & Pérez, S. (2020). ESTADOS POTENCIALES EN MÉXICO PARA LA PRODUCCIÓN y CONSUMO DE FRIJOL BIOFORTIFICADO CON HIERRO y ZINC.
Revista Fitotecnia Mexicana, 43(1), 11. https://doi.org/10.35196/rfm.2020.1.11

2 Salazar, J. A. G., Licea, G. R., Torres, A. S., & Rebollar, S. R. (2006). Políticas para mejorar la competitividad de la producción de maíz y frijol en México. Revista Fitotecnia Mexicana, 29(Es2), 115-121.

3 MLOps & Quality Data: The Path to AI Transformation - Spiceworks. (2023, 9 agosto). Spiceworks. https://www.spiceworks.com/tech/artificial-intelligence/guest-article/mlops-best-practices-optimizing-ai-with-quality-data/

4 sklearn.preprocessing.StandardScaler. (s.f.). Scikit-learn.
https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html