UTM
UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING

SESSION 2023/2024 SEMESTER 1

SECB3203 PROGRAMMING FOR BIOINFORMATICS

RECURSIVE FEATURE ELIMINATION METHODS FOR GENE SELECTION ON OVARIAN CANCER CLASSIFICATION

PROJECT PROGRESS 5

SECTION : 01

LECTURER'S NAME : DR. NIES HUI WEN

DATE OF SUBMISSION : 30 DECEMBER 2023

GROUP NAME : GROUP 10

NAME	MATRIC NO
NURATHIRAH BINTI MUHAMAD ZAKI	A21EC3025
YUSRA NADATUL ALYEEA BINTI YUSRAMIZAL	A21EC0151

Table of Content

1.0 Introduction

Ovarian cancer, an effective conduct in the field of cancer, is getting attention due to the late stage at which it is typically diagnosed and the few treatment options that are available for it. This condition, which affects a significant number of people all over the world, caused efforts to improve the accuracy of diagnosis and to find different options for treatment. Because of the nature of ovarian cancer, which can be identified by its unclear symptoms and multiple subtypes, there is a greater requirement to explore new methods to better understand and diagnose the disease.

Gene expression data, which captures the activity levels of numerous genes within cells, has shown to be an extremely helpful resource for acquiring insights into the molecular pathways that are causing cancer. When it comes to ovarian cancer, doing an analysis of gene expression patterns presents the possibility of developing robust classification models that are able to differentiate between malignant and non-cancerous samples. However, gene expression datasets often include a very large number of genes, which means that selecting features is an essential stage in the process of simplifying the data and enhancing the performance of classification models.

Recursive Feature Elimination (RFE), a method for selecting features that has become increasingly popular in this sector, is one example of this type of technique. The RFE method is an iterative process that systematically discovers and retains the genes that contain the most useful information while rejecting the genes that have the least relevant information. This strategy is essential for the construction of accurate and effective ovarian cancer classification models, for lowering the dimensionality of the data, and for contributing to improved model generalization. RFE assists in the selection of a group of genes that have the most discriminative potential for accurate classification. This is accomplished by the iterative process of ranking gene characteristics and deleting them.

The identification of and diagnosis of ovarian cancer, which is a tough opponent in the field of medicine, still has challenges. It is a disease that is often found in later phases, which makes treatments less effective. Despite these issues, the development of advanced biological

techniques and the availability of large amounts of gene expression data bring up new ways to learn more about ovarian cancer and make it easier to identify.

This project stands out because it uses two effective methods together in a smart way. Recursive Feature Elimination (RFE) and Support Vector Machines (SVM). As a feature selection method, RFE helps us quickly find our way around the huge amount of gene expression data that we have. The information helps us find the genes that are most important for classifying ovarian cancer. SVM is an effective machine learning method that we use to build our classification model.

One of the best things about this project is that it uses ideas from many different fields. Working with experts in the biological process and cancer makes sure that our results are useful for both medical and scientific research. Combining ideas from data with a scientific understanding is what can help us come up with better ways to diagnose and treat diseases.

We are excited about the chance to make a real difference in the fight against ovarian cancer as we start this research project. Our goal is to make the lives of people who have this disease better by finding it earlier, handling it more effectively, and learning more about how it works biologically. It is a project that brings together science, health, and creative thinking. The main goal is to make the struggle against ovarian cancer better in the future.

1.1 Problem Background

In recent years, ovarian cancer has become known as a serious and challenging health concern, affecting a huge number of people all over the world. Ovarian cancer is a challenge because it affects a large number of people. Due to the fact that it is typically diagnosed at a late stage and offers few treatment choices, there is an urgent requirement for the development of new diagnostic and categorization procedures. Because of the evasive nature of ovarian cancer, which is characterized by its growth and progression being closely connected to genetic variables, there has been an increased emphasis placed on comprehending the molecular roots of this illness.

The propensity of ovarian cancer to avoid early identification presents one of the most major challenges in the treatment of the disease. In contrast to other types of cancer, ovarian cancer's early stages are frequently uncharacteristic in terms of the symptoms they present. As a consequence of this, many cases are not detected until they have progressed to an advanced level, which makes it more difficult to successfully treat the condition. The need for more advanced diagnostic technologies, which have the potential to detect the disease at earlier stages when it is easier to treat, is of the highest priority.

1.2 Problem Statement

The diagnosis of ovarian cancer tends to happen at a late stage, and there are few treatment options offered. Ovarian cancer presents a substantial burden to public health. Gene expression data, which offers helpful insights into the molecular pathways underlying cancer, shows potential for improving the classification of ovarian cancer and providing a more accurate diagnosis of the disease. However, there is a challenge that is caused by the huge amount of genes found in gene expression databases. The challenge is to identify a subset of genes that are most relevant for discriminating between samples of ovarian cancer and samples of non-cancerous tissue, to construct an appropriate classification model based on these selected genes, and to gain insights into the biological processes linked with ovarian cancer.

1.3 Objectives

1. To evaluate the selected genes on an ovarian dataset by using a Support Vector Machine (SVM) classifier for the classification matrix.
2. To improve the accuracy of current works and research related to the classification of ovarian cancer using Recursive Feature Elimination (RFE) and Support Vector Machine (SVM).
3. To select an informative gene by using the Recursive Feature Elimination (RFE) method.

1.4 Scopes

Domain of the Data:

This research project is based on a gene expression dataset where patient samples are represented as rows, and gene expression levels as columns. The primary objective is to identify significant gene patterns associated with ovarian cancer and employ them as features in a classification model.

Techniques to Be Used:

The research will employ advanced techniques for gene selection and feature extraction, with a specific emphasis on Recursive Feature Elimination (RFE) methods. These techniques will be crucial for systematically identifying genes that are strongly correlated with ovarian cancer. Additionally, Support Vector Machines (SVM) will be used as the classification algorithm to predict whether a patient has ovarian cancer based on their gene expression data.

Methodology:

RFE methods will be applied iteratively to select and rank genes based on their relevance to ovarian cancer. The selected genes will be employed as features in an SVM-based classification model. Model training, validation, and evaluation will be carried out to assess its predictive performance and generalizability.

Limits of the Research:

Certain limitations are inherent to this study. The findings are contingent upon the quality, completeness, and representativeness of the selected gene expression dataset. While the research aims to identify gene patterns related to ovarian cancer and build a predictive classification model, it does not encompass clinical validation. Ethical considerations related to patient data usage will not be explicitly addressed, as the focus is on the technical aspects of model development and gene selection.

1.5 Conclusion

In conclusion, this study project proposal is an important and inventive step toward better understanding ovarian cancer and making diagnosis accuracy better. The use of gene expression data along with the strong RFE feature selection method shows possibility for a more accurate and dependable way to define ovarian cancer.

We discussed the problem and how important it is to improve ovarian cancer diagnostics right away. We have also talked about how important gene expression data is and how important RFE is as a method for gene selection. We have made clear that the objectives of our study consist of identifying the set of genes that are most important for accurate classification, creating an accurate model for ovarian cancer classification, and looking for biological insights into the disease.

As we begin this journey, we recognize that even though the main focus of our project is on classification, it could have a much bigger effect than that. The study may not only help specialists make more accurate diagnosis of ovarian cancer, but it may also help us learn more about how this disease works at the molecular level. Working with oncology and biology experts makes our dedication to translational study and putting our findings to use even stronger.

The scope of the project has been carefully laid out to assist us stay focused on goals that can be fulfilled while also being aware of what the project is unable to do. Our goal is to create an accurate classification model. This project is making progress toward a time when ovarian cancer can be found faster and better. As a part of the ongoing fight against this hard disease, the present study shows how data-driven methods can help.

1.6 Software and Hardware Requirements

Hardware:

1. **Computer**

   A reasonably powerful computer with a multi-core processor and sufficient RAM is essential for handling the data preprocessing, feature selection, model training, and analysis tasks efficiently.

2. **Storage**

   As gene expression datasets can be large, we need to have ample storage space. Enough storage is needed to store the datasets and the various versions of data during preprocessing.

3. **Graphics Processing Unit (GPU)**

   While not strictly necessary, using a computer with a dedicated GPU can significantly speed up the training of machine learning models like SVM, particularly for large datasets. Many libraries and software tools support GPU acceleration.

Software:

1. **Visual Studio Code (VSCode)**

   A code editor that's particularly useful for writing, debugging, and managing code, including Python scripts.

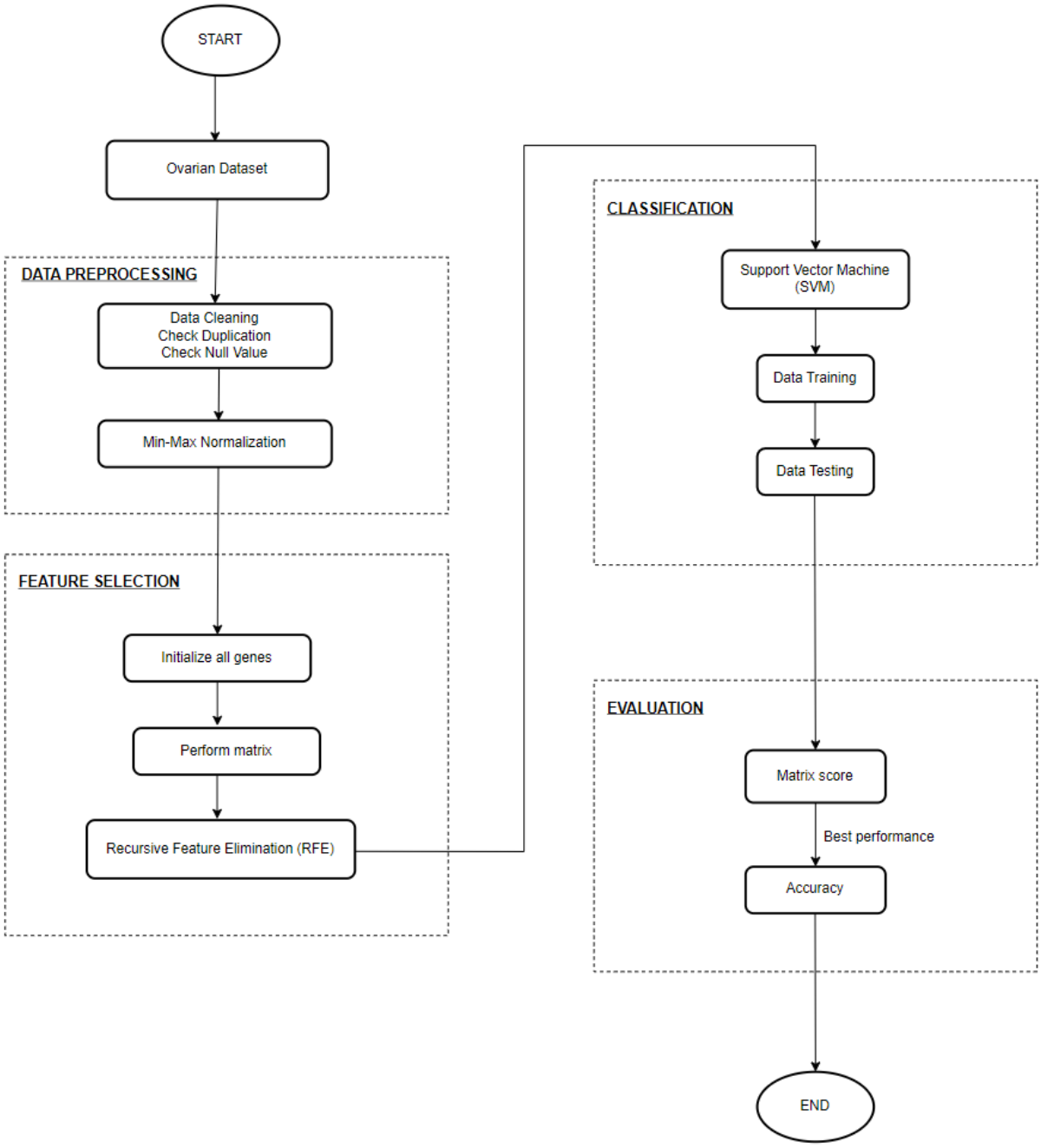
2. **Scikit-learn (sklearn)**

   A powerful machine learning library that provides SVM implementations and tools for model development and evaluation.

3. **WEKA**

   A software for machine learning and data analysis, offering a user-friendly interface and a wide range of algorithms and tools for research in bioinformatics and data science.

1.7 Flowchart of The Proposed Approach



2.0 Data Collection and Preprocessing

2.1 Importing Dataset

The dataset was obtained from Zexuan Zhu, Y. S. Ong, and M. Dash's work titled “Markov Blanket-Embedded Genetic Algorithm for Gene Selection,” published in Pattern Recognition, Vol. 49, No. 11, 3236-3248, 2007. It comprises 253 rows and 15,155 columns, representing 253 samples and 15,154 features. The study aims to enhance the accuracy of existing research on ovarian cancer classification by employing Recursive Feature Elimination (RFE) and Support Vector Machine (SVM) methodologies.

2.1.1 Importing and Exporting Data in Python

The dataset is in ARFF file format. However, to import it into Google Colab, we need to convert it to a CSV file. Then, upload the "csv_result-Ovarian.csv" file to the Google Colab environment.

```
[5]  from google.colab import files
     uploaded = files.upload()
```

```
Choose Files  csv_result-Ovarian.csv
• csv_result-Ovarian.csv(text/csv) - 33914743 bytes, last modified: 11/19/2023 - 100% done
Saving csv_result-Ovarian.csv to csv_result-Ovarian (2).csv
```

```
[11] # Read the CSV file into a Pandas DataFrame
     data = pd.read_csv('csv_result-Ovarian.csv')
```

2.1.2 Understanding The Data

After uploading the file, we started by gaining an understanding of the data. We displayed a few rows of the dataset, obtained information about it, and computed summary statistics. This included showcasing the table with attributes and their respective values from our dataset.

```
[12] # Display the first few rows of the dataset
     print(data.head())

     # Display information about the dataset
     print(data.info())

     # Summary statistics of the dataset
     print(data.describe())
```

```
   id  MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  MZ0.000810195  \
0   1     0.494626    0.263735    0.321841       0.220934       0.297622
1   2     0.258063    0.406593    0.321841       0.069771       0.333335
2   3     0.537636    0.032966    0.321841       0.209307       0.404762
3   4     0.000000    0.395605    0.310347       0.197673       0.404762
4   5     0.526884    0.395605    0.367817       0.383719       0.488099

   MZ0.001428564  MZ0.002221123  MZ0.003187869  MZ0.004328805  ...  \
0       0.316458       0.154763       0.223685       0.304346  ...
1       0.354432       0.321431       0.144740       0.260869  ...
2       0.113927       0.369049       0.223685       0.536231  ...
3       0.455701       0.416666       0.210527       0.420292  ...
4       0.392405       0.238094       0.500000       0.362316  ...

   MZ19974.404  MZ19977.042  MZ19979.68  MZ19982.319  MZ19984.957  \
0     0.483622     0.449296    0.449296     0.449296     0.449296
1     0.631765     0.619718    0.619718     0.619718     0.619718
2     0.038462     0.035918    0.035918     0.035918     0.035918
3     0.497864     0.486621    0.486621     0.486621     0.486621
4     0.267096     0.251408    0.251408     0.251408     0.251408

   MZ19987.596  MZ19990.235  MZ19992.874  MZ19995.513   Class
0     0.449296     0.449296     0.449296     0.449296  Normal
1     0.619718     0.619718     0.619718     0.619718  Normal
2     0.035918     0.035918     0.035918     0.035918  Normal
3     0.486621     0.486621     0.486621     0.486621  Normal
4     0.251408     0.251408     0.251408     0.251408  Normal
```

```
[5 rows x 15156 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 253 entries, 0 to 252
Columns: 15156 entries, id to Class
dtypes: float64(15151), int64(4), object(1)
memory usage: 29.3+ MB
None
                id  MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  \
count  253.000000   253.000000  253.000000  253.000000     253.000000
mean   127.000000     0.532450    0.462623    0.465859       0.421363
std     73.179004     0.183136    0.196834    0.196418       0.212991
min      1.000000     0.000000    0.000000    0.000000       0.000000
25%     64.000000     0.397850    0.329667    0.321841       0.255817
50%    127.000000     0.537636    0.461537    0.459768       0.430235
75%    190.000000     0.655912    0.593407    0.597701       0.569765
max    253.000000     1.000000    1.000000    1.000000       1.000000

       MZ0.000810195  MZ0.001428564  MZ0.002221123  MZ0.003187869  \
count     253.000000     253.000000     253.000000     253.000000
mean        0.481133       0.489570       0.466263       0.512587
std         0.196406       0.185289       0.173548       0.214232
min         0.000000       0.000000       0.000000       0.000000
25%         0.345240       0.354432       0.345240       0.342109
50%         0.476194       0.468359       0.464290       0.500000
75%         0.619047       0.620253       0.595238       0.684212
max         1.000000       1.000000       1.000000       1.000000

       MZ0.004328805  ...  MZ19971.766  MZ19974.404  MZ19977.042  MZ19979.68  \
count     253.000000  ...   253.000000   253.000000   253.000000  253.000000
mean        0.531018  ...     0.435985     0.443944     0.430548    0.430548
std         0.215837  ...     0.160151     0.156482     0.155192    0.155192
min         0.000000  ...     0.000000     0.000000     0.000000    0.000000
25%         0.362316  ...     0.324327     0.334049     0.330282    0.330282
50%         0.536231  ...     0.441929     0.451566     0.433102    0.433102
75%         0.710146  ...     0.544194     0.547010     0.541554    0.541554
max         1.000000  ...     1.000000     1.000000     1.000000    1.000000
```

```
       MZ19982.319  MZ19984.957  MZ19987.596  MZ19990.235  MZ19992.874  \
count   253.000000   253.000000   253.000000   253.000000   253.000000
mean      0.430548     0.430548     0.430548     0.430548     0.430548
std       0.155192     0.155192     0.155192     0.155192     0.155192
min       0.000000     0.000000     0.000000     0.000000     0.000000
25%       0.330282     0.330282     0.330282     0.330282     0.330282
50%       0.433102     0.433102     0.433102     0.433102     0.433102
75%       0.541554     0.541554     0.541554     0.541554     0.541554
max       1.000000     1.000000     1.000000     1.000000     1.000000

       MZ19995.513
count   253.000000
mean      0.430548
std       0.155192
min       0.000000
25%       0.330282
50%       0.433102
75%       0.541554
max       1.000000

[8 rows x 15155 columns]
```

2.1.3 Getting Started Analyzing Data in Python

From the dataset, we analyze that it contains 253 samples and an extensive set of 15,155 features, encompassing various data types such as floating-point, integer, and categorical values within its columns. Notably, the 'id' column consists of integers, while the 'Class' column holds categorical data signifying class labels. Delving into the summary statistics, we uncover valuable insights into the distribution of data, unveiling key metrics like mean values, standard deviations, and ranges across the features. These statistics offer a glimpse into the dataset's central tendencies, spread, and potential outliers. Moreover, examining quartile values aids in comprehending data consistency and variations present within the dataset. Understanding the distribution of the 'Class' column becomes pivotal for classification tasks, facilitating an assessment of class balance or imbalance.

2.1.4 Python Package for Data Science

We begin by importing the necessary libraries: Pandas for data handling and any other libraries required for analysis.

```
[10] import pandas as pd
     import numpy as np
```

```
[ ]  #  Install scikit-learn
     !pip install scikit-learn
```

2.2 Data Wrangling (Pandas/Numpy)

2.2.1 Identify and Handling Missing Values

To identify and handle missing values in the dataset, we initially checked for missing values in the 'id' column using the code `data.isnull().sum()`. The result indicated no missing values for the 'id' column. Subsequently, we handled missing values across the dataset by executing `data.dropna(inplace=True)`. The resulting output of '0' confirmed the absence of missing values in the dataset.

```
[13] # Check for missing values
     print(data.isnull().sum())

     # Handle missing values by dropping or filling them
     data.dropna(inplace=True)
```

```
id               0
MZ-7.86E-05      0
MZ2.18E-07       0
MZ9.60E-05       0
MZ0.000366014    0
                ..
MZ19987.596      0
MZ19990.235      0
MZ19992.874      0
MZ19995.513      0
Class            0
Length: 15156, dtype: int64
```

2.2.2 Data Formatting

Load the file into a Pandas DataFrame using `pd.read_csv()`. Then, to display the first few rows of the original DataFrame, utilize `df.head()`.

```
[13]
     # Display the first few rows of the DataFrame
     print("Original DataFrame:")
     print(df.head())
```

```
Original DataFrame:
   id  MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  MZ0.000810195  \
0   1     0.494626    0.263735    0.321841       0.220934       0.297622
1   2     0.258063    0.406593    0.321841       0.069771       0.333335
2   3     0.537636    0.032966    0.321841       0.209307       0.404762
3   4     0.000000    0.395605    0.310347       0.197673       0.404762
4   5     0.526884    0.395605    0.367817       0.383719       0.488099

   MZ0.001428564  MZ0.002221123  MZ0.003187869  MZ0.004328805  ...  \
0       0.316458       0.154763       0.223685       0.304346  ...
1       0.354432       0.321431       0.144740       0.260869  ...
2       0.113927       0.369049       0.223685       0.536231  ...
3       0.455701       0.416666       0.210527       0.420292  ...
4       0.392405       0.238094       0.500000       0.362316  ...

   MZ19974.404  MZ19977.042  MZ19979.68  MZ19982.319  MZ19984.957  \
0     0.483622     0.449296    0.449296     0.449296     0.449296
1     0.631765     0.619718    0.619718     0.619718     0.619718
2     0.038462     0.035918    0.035918     0.035918     0.035918
3     0.497864     0.486621    0.486621     0.486621     0.486621
4     0.267096     0.251408    0.251408     0.251408     0.251408

   MZ19987.596  MZ19990.235  MZ19992.874  MZ19995.513  Class
0     0.449296     0.449296     0.449296     0.449296      1
1     0.619718     0.619718     0.619718     0.619718      1
2     0.035918     0.035918     0.035918     0.035918      1
3     0.486621     0.486621     0.486621     0.486621      1
4     0.251408     0.251408     0.251408     0.251408      1

[5 rows x 15156 columns]
```

To identify categorical columns and encode them, import the LabelEncoder. This step aims to verify if the dataset contains categorical variables that have not been encoded as numbers. Once the LabelEncoder is imported, print the first few rows of the DataFrame after encoding the categorical column.

```
[11] import pandas as pd
     from sklearn.preprocessing import LabelEncoder

     # Encode categorical columns (if any)
     le = LabelEncoder()
     categorical_columns = df.select_dtypes(include=['object']).columns
     df[categorical_columns] = df[categorical_columns].apply(lambda col: le.fit_transform(col.astype(str)))

     # Display the DataFrame after encoding categorical columns
     print("\nDataFrame after encoding categorical columns:")
     print(df.head())
```

```
DataFrame after encoding categorical columns:
   id  MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  MZ0.000810195  \
0   1     0.494626    0.263735    0.321841       0.220934       0.297622
1   2     0.258063    0.406593    0.321841       0.069771       0.333335
2   3     0.537636    0.032966    0.321841       0.209307       0.404762
3   4     0.000000    0.395605    0.310347       0.197673       0.404762
4   5     0.526884    0.395605    0.367817       0.383719       0.488099

   MZ0.001428564  MZ0.002221123  MZ0.003187869  MZ0.004328805  ...  \
0       0.316458       0.154763       0.223685       0.304346  ...
1       0.354432       0.321431       0.144740       0.260869  ...
2       0.113927       0.369049       0.223685       0.536231  ...
3       0.455701       0.416666       0.210527       0.420292  ...
4       0.392405       0.238094       0.500000       0.362316  ...

   MZ19974.404  MZ19977.042  MZ19979.68  MZ19982.319  MZ19984.957  \
0     0.483622     0.449296    0.449296     0.449296     0.449296
1     0.631765     0.619718    0.619718     0.619718     0.619718
2     0.038462     0.035918    0.035918     0.035918     0.035918
3     0.497864     0.486621    0.486621     0.486621     0.486621
4     0.267096     0.251408    0.251408     0.251408     0.251408

   MZ19987.596  MZ19990.235  MZ19992.874  MZ19995.513  Class
0     0.449296     0.449296     0.449296     0.449296      1
1     0.619718     0.619718     0.619718     0.619718      1
2     0.035918     0.035918     0.035918     0.035918      1
3     0.486621     0.486621     0.486621     0.486621      1
4     0.251408     0.251408     0.251408     0.251408      1

[5 rows x 15156 columns]
```

2.2.3 Data Normalization

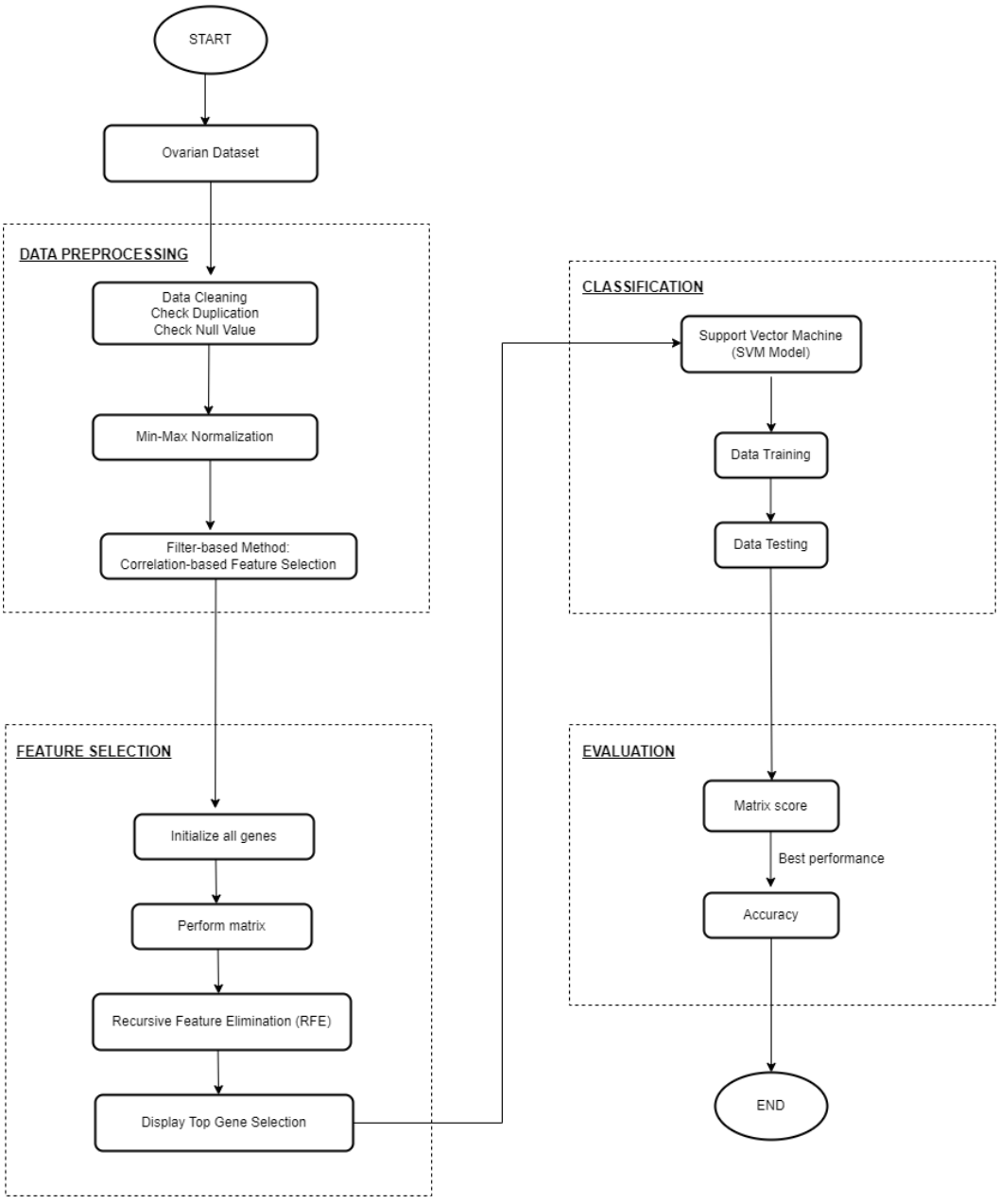
After reading the CSV file into a Pandas DataFrame named ‘data’, the next step involves selecting the numerical columns for normalization, which excludes the ‘id’ and ‘Class’. Subsequently, the selected numerical columns are converted to numeric format. Moreover, the Min-Max scaling method is applied to normalize the values within the selected numerical columns. Finally, all rows of the DataFrame are printed after the normalization process.

```
[ ] import pandas as pd
    from sklearn.preprocessing import MinMaxScaler

    # Load CSV file into a Pandas DataFrame
    data = pd.read_csv('ovariancancer.csv')

    # Select numerical columns to normalize excluding the first and last columns
    columns_to_exclude = ['id', 'Class']
    numerical_columns_to_normalize = data.drop(columns_to_exclude, axis=1).columns

    # Check and handle non-numeric values
    data[numerical_columns_to_normalize] = data[numerical_columns_to_normalize].apply(pd.to_numeric,errors='coerce')

    # Normalize numerical using Min-Max scaling
    scaler = MinMaxScaler()
    data[numerical_columns_to_normalize] = scaler.fit_transform(data[numerical_columns_to_normalize])

    # Display all rows after normalization
    print(data[numerical_columns_to_normalize])
```

```
     MZ-7.86E-05  MZ2.18E-07  MZ9.60E-05  MZ0.000366014  MZ0.000810195  \
0       0.494626    0.263735    0.321841       0.220934       0.297622
1       0.258063    0.406593    0.321841       0.069771       0.333335
2       0.537636    0.032966    0.321841       0.209307       0.404762
3       0.000000    0.395605    0.310347       0.197673       0.404762
4       0.526884    0.395605    0.367817       0.383719       0.488099
5       0.397850    0.395605    0.298853       0.372092       0.333335
6       0.645160    0.307689    0.241378       0.372092       0.392857
7       0.720432    0.351644    0.344829       0.465117       0.452385
8       0.537636    0.307689    0.425287       0.348837       0.333335
9       0.526884    0.494503    0.333335       0.279072       0.523812
10      0.720432    0.351644    0.459768       0.244189       0.273813
11      0.559141    0.505492    0.528738       0.209307       0.273813
12      0.440859    0.340655    0.356323       0.732556       0.226190
13      0.569893    0.450548    0.471268       0.453490       0.595238
14      0.451612    0.593407    0.528738       0.232562       0.500003
15      0.623655    0.527469    0.436781       0.418607       0.559525
16      0.516131    0.582418    0.344829       0.546510       0.619047
17      0.569893    0.395605    0.505750       0.430235       0.690479
18      0.483869    0.505492    0.459768       0.476745       0.583334
19      0.591398    0.505492    0.367817       0.546510       0.500003
20      0.677423    0.307689    0.436781       0.430235       0.583334
```

3. Model Development

Flow Chart for Model Development



Importing the required libraries. `pandas` are used for data manipulation, `numpy` for numerical operations, and `sklearn` for machine learning tasks such as splitting data, training Support Vector Machine (SVM), and performing Recursive Feature Elimination (RFE).

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.feature_selection import RFE
```

Read the dataset into a Pandas DataFrame. Ensure that the correct actual filename of the dataset.

```
# Load the dataset
data = pd.read_csv('csv_result-Ovarian.csv')
```

The dataset contains column name ‘id’, which is excluded from the analysis to avoid unnecessary features. The `errors = ignore` is an argument to handle cases where ‘id’ is not present.

```
# Exclude 'id'
if 'id' in data.columns:
    columns_to_exclude = ['id']
    data = data.drop(columns=columns_to_exclude, errors='ignore')
```

Features X and target variable y from the dataset. X contains all columns except the last one and y contains the last column.

```
# Assuming the target column is the last column, separate features and target
X = data.iloc[:, :-1]  # Features (all columns kecuali last)
y = data.iloc[:, -1]  # Target variable (last column)
```

This part performs the correlation-based feature selection. It calculates the correlation matrix, identifies highly correlated features, and removes them. The features with a correlation coefficient greater than 0.95 are excluded.

```
# Correlation-based feature selection
correlation_matrix = X.corr().abs()
upper_triangle = correlation_matrix.where(np.triu(np.ones(correlation_matrix.shape), k=1).astype(bool))
high_corr_features = [col for col in upper_triangle.columns if any(upper_triangle[col] > 0.95)]
X_selected = X.drop(columns=high_corr_features)
```

The data is split into training and testing sets using the `train_test_split` function. 80% of the data is used for training and 20% is used for testing.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

SVM-RFE is applied using a linear Support Vector Machine as the estimator. It selects the specific number of the top gene `num_top_genes` based on feature importance.

```
# SVM-RFE
svm_classifier = SVC(kernel='linear')
num_top_genes = 5  # Specify the number of top genes to display
rfe = RFE(estimator=svm_classifier, n_features_to_select=num_top_genes)
rfe.fit(X_train, y_train)
```

Determine the selected features based on the optimal threshold. The `rfe.support_` attribute provides a boolean mask of the selected features.

```
# Get the selected features based
optimal_threshold = 0.8  # Adjust optimal value
selected_features = X_train.columns[rfe.support_]
```

Lastly, the top genes contributing to classification are displayed. It prints the maximum values of the selected features in descending order.

```
# Display the top genes contributing to classification
top_genes = X[selected_features].max().sort_values(ascending=False)
print(f"Top {num_top_genes} Genes for Classification:")
print(top_genes)
```

The output provides the top 5 genes selected by the SVM-RFE (Support Vector Machine with Recursive Features Elimination) algorithm for classification. The list gene has a score of 1.0, indicating that it was assigned the highest rank by the SVM-RFE algorithm. The score of 1.0 suggests that the gene is considered highly important for classification.

```
Top 5 Genes for Classification:
MZ2.7921478    1.0
MZ2.9182952    1.0
MZ244.66041    1.0
MZ261.88643    1.0
MZ433.90794    1.0
dtype: float64
```

4. Model Evaluation

To perform the model evaluation of the performance of the machine learning SVM-RFE, we use the accuracy metric. The code line from `sklearn.model_selection` import `cross_val_score` in Python imports the `cross_val_score` function from Scikit-Learn's model_selection module. This function is a powerful tool for performing cross-validation, a technique used in machine learning to evaluate model performance. It works by splitting the dataset into multiple subsets (folds), training the model on a portion of the data, and testing it on the remaining data, iteratively across different folds. The function computes scores (such as accuracy, F1 score, etc.) for each fold and returns an array of these scores, allowing practitioners to assess the model's performance across different data subsets.

```
from sklearn.model_selection import cross_val_score
```

Reads the dataset from the 'csv_result-Ovarian.csv' file.

```
# Load the dataset
data = pd.read_csv('csv_result-Ovarian.csv')
```

The 5-fold cross-validation ('cv=5') is performed using the SVM classifier after feature selection. During cross-validation, we specify accuracy and F1 score as the evaluation metrics. `cv_accuracy_svm.mean()` is used to compute the average accuracy across the 5 folds whereas `cv_f1_svm.mean()` is used to compute the average F1 score across the 5 folds.

```
# Perform cross-validation with SVM classifier
cv_accuracy_svm = cross_val_score(svm_classifier, X_svm_rfe, y_encoded, cv=5, scoring='accuracy')
cv_f1_svm = cross_val_score(svm_classifier, X_svm_rfe, y_encoded, cv=5, scoring='f1')
```

The Average Accuracy represents the average accuracy achieved by the SVM classifier after feature selection across the 5 folds while Average F1 Score represents the average f1 score achieved by the SVM classifier after feature selection across the 5 folds.

```
# Print average accuracy and F1 score after SVM-RFE and classification
print(f"Average Accuracy after SVM-RFE: {cv_accuracy_svm.mean() * 100:.2f}%")
print(f"Average F1 Score after SVM-RFE: {cv_f1_svm.mean():.4f}")
```

The output indicates that, on average, the Support Vector Machine (SVM) classifier, after feature selection through Recursive Feature Elimination (SVM-RFE), achieved perfect performance in classifying instances within the dataset across different folds of cross-validation. Both the reported 100.00% average accuracy and 1.0000 average F1 score suggest flawless prediction results on this particular dataset.

```
Average Accuracy after SVM-RFE: 100.00%
Average F1 Score after SVM-RFE: 1.0000
```