



جامعة أم القرى
كلية الحاسبات

PROJECT REPORT SYMPTOMS AND SEVERITY OF COVID-19 ANALYSIS OF WORLD HEALTH ORGANIZATION DATA

Team Members:

443010340
443008b78
443009827
44400357b

ليان خالد الحازمي.
جود ماجد وجيه.
جمانه عبد الرزاق الرحيلي.
لمار بندر فلمبان.

Prepared for :

Dr.Ammara Mohammad Al-Harbi



Task

DISTRIBUTION

Lamar

Data preparation , studied and analyzed
"Decision stump" Algorithm

Layan

Analyzed of the two algorithms, ZeroR and
Decision stump Calculating the accuracy of the
algorithms

Jumana

Studied and analyzed data Implementation and
preprocessing "IBK (Lazy)" Algorithm and
discretize filter

Jood

Data preparation in Rapid Miner , studied and
analyzed "Decision Tree" , "Naïve Bayes" and "
Single Rule Induction".



INTRODUCTION

This project entails an experimental exploration of machine learning methodologies, specifically employing the Weka software, to classify COVID-19 data to determine whether are afflicted with coronavirus disease or not based on predefined standard symptoms and whether the person Has contacted to some other COVID-19 Patient. By applying various algorithms and filters, our objective is to enhance the accuracy of the classification model.

PURPOSE OF THE PROJECT

The primary objective of this experiment is to achieve a high level of accuracy in classifying COVID-19 cases using Weka. By systematically applying algorithms and filters, we seek to optimize the model's performance

The purpose is to utilize data analysis techniques to determine whether individuals are afflicted with coronavirus disease based on predefined standard symptoms and whether the person Has contacted to some other COVID-19 Patient.

This tool will assist healthcare professionals in prioritizing testing and medical care, thereby contributing to effective public health management and containment of the virus.

LEARNING OBJECTIVES

Through this project, we aim to gain practical experience in machine learning application, particularly in the context of real-world data analysis and classification tasks

- Understanding the fundamentals of machine learning algorithms and their application to real-world problems.
- Preprocessing and preparing datasets for machine learning tasks, including data cleaning, feature engineering, and normalization.
- Implementing machine learning algorithms using Weka, including model training, evaluation, and interpretation of results.
- Experimenting with various filters and preprocessing techniques to improve model accuracy and performance.

SUMMARY OF THE REPORT

This report presents an experimental exploration of machine learning techniques applied to the classification of COVID-19 data to determine whether are afflicted with coronavirus disease or not based on predefined standard symptoms and whether the person Has contacted to some other COVID-19 Patient. using the Weka software. The project aims to achieve practical experience in real-world data analysis and classification tasks while enhancing the accuracy of the classification model.

The report begins with an introduction outlining the project's objectives, which include gaining practical experience in machine learning application and improving accuracy through algorithm and filter application. and the learning objectives to gain skills in data analysis and classification.

Following the introduction, the report delves into the experiment's methodology, detailing the selection of the COVID-19 dataset, the application of machine learning algorithms within Weka, and the implementation of filters and preprocessing techniques to enhance accuracy.

Results of the experiment are presented, including findings related to the performance of the classification model, accuracy metrics, and insights gained from the analysis. Screenshots and visual representations of the experiment's output are provided to facilitate understanding and interpretation.

Finally, the report concludes with a summary of key findings and implications, highlighting the significance of the project in contributing to the field of machine learning in public health and epidemiology.

THE GOAL OF THE EXPERIMENT

The goal is to develop a robust classification model with at least %90 accuracy for identifying COVID-19 cases based on symptom data and based on whether the person Has contacted to some other COVID-19 Patient. .

THE MACHINE LEARNING METHOD OR ALGORITHM:

Classifying number of patient whether they are afflicted with coronavirus disease or not based on predefined standard symptoms

THE CHOSEN DATASET

This data will help to identify whether the person has a coronavirus disease or not based on some pre-defined standard symptoms and whether the person Has contacted to some other COVID-19 Patient. These symptoms are based on guidelines given by the World Health Organization (WHO)who.int and the Ministry of Health and Family Welfare, India.

Source: The dataset is sourced from Kaggle.

Language: English

Size: 316,800 records

Type: binomenal

Classes: The dataset consists of 3 classes labeled as:

- 1- Contact Yes
- 2- Contact No
- 3- Contact Don't know

The final classification result whether any person is having a coronavirus disease or not is determined based on whether the person Has contacted to some other COVID-19 Patient.

Based on seven major variables that may impact whether someone has coronavirus disease:

- **Country:** List of countries the person visited.
- **Age:** Classification of the age group for each person, based on WHO Age Group Standard.
- **Symptoms:** According to WHO, the five major symptoms of COVID-19 are **Fever, Tiredness, Difficulty in breathing, Dry cough, and sore throat.**

Experience any other symptoms: Additional symptoms reported by the **individuals, including Pains, Nasal Congestion, Runny Nose, Diarrhea,** and Others.

- **Severity:** The level of severity of symptoms reported by the individuals, categorized as **Mild, Moderate, or Severe.**
- **Contact:** Indicates whether the person has been in contact with another COVID-19 patient.

Each variable consists of categorical labels.

A combination for each label in the variables will be generated, resulting in a total of 316,800 combinations.

DATA PREPARATION

We Ensured that the dataset is formatted correctly for weka by the following steps:

- Resampling the data from 316,800 combinations to 20,000 because it is easier to deal with a small sample.

```
import pandas as pd

input_file = '/content/covid.csv'
output_file = 'covid_data.csv'
sample_size = 20000 # Number of rows to sample

# Read the CSV file
data = pd.read_csv(input_file)

# Take a random sample
sampled_data = data.sample(n=sample_size, random_state=42) # Set random_state for reproducibility

# Save the sampled data to a new CSV file
sampled_data.to_csv(output_file, index=False)

print(f'Sampled data saved to {output_file}!')
```

⇒ Sampled data saved to covid_data.csv!

- Then We deleted the unwanted column 'Country'

```
import pandas as pd

# Assuming your data is stored in a CSV file
df = pd.read_csv('/content/covid_data.csv')

#print(df.columns)

# Selecting the desired columns
desired_columns = ['Fever', 'Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing',
                  'Sore-Throat', 'None_Sympton', 'Pains', 'Nasal-Congestion',
                  'Runny-Nose', 'Diarrhea', 'None_Experiencing', 'Age_0-9', 'Age_10-19',
                  'Age_20-24', 'Age_25-59', 'Age_60+', 'Gender_Female', 'Gender_Male',
                  'Gender_Transgender', 'Severity_Mild', 'Severity_Moderate',
                  'Severity_None', 'Severity_Severe', 'Contact_Dont-Know', 'Contact_No',
                  'Contact_Yes']
df_filtered = df[desired_columns]

# Save the filtered DataFrame to a new CSV file
df_filtered.to_csv('filtered_data2.csv', index=False)
```

- Lastly converting the CVS file to arff file and Ensure the dataset is formatted well so weka can accept the data

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('/content/filtered_data2.csv')

# Create the ARFF attribute names and types
attribute_names = ['Fever', 'Tiredness', 'DryCough', 'DifficultyBreathing',
                   'SoreThroat', 'NoneSympton', 'Pains', 'NasalCongestion',
                   'RunnyNose', 'Diarrhea', 'NoneExperiencing', 'Age8to9', 'Age10to19',
                   'Age20to24', 'Age25to59', 'Age60Ap', 'GenderFemale', 'GenderMale',
                   'GenderTransgender', 'SeverityMild', 'SeverityModerate',
                   'SeverityNone', 'SeveritySevere', 'ContactDontKnow', 'ContactNo',
                   'ContactYes']
attribute_types = ['{0,1}'] * (len(attribute_names))

# Select the desired columns
desired_columns = ['Fever', 'Tiredness', 'Dry-Cough', 'Difficulty-in-Breathing',
                   'Sore-Throat', 'None_Sympton', 'Pains', 'Nasal-Congestion',
                   'Runny-Nose', 'Diarrhea', 'None_Experiencing', 'Age_0-9', 'Age_10-19',
                   'Age_20-24', 'Age_25-59', 'Age_60+', 'Gender_Female', 'Gender_Male',
                   'Gender_Transgender', 'Severity_Mild', 'Severity_Moderate',
                   'Severity_None', 'Severity_Severe', 'Contact_Dont-Know', 'Contact_No',
                   'Contact_Yes']

df_filtered = df[desired_columns]

# Save the ARFF file
with open('converted_data8.arff', 'w') as f:
    # Write the relation name
    f.write('@relation covid\n\n')

    # Write the attribute names and types
    for name, typ in zip(attribute_names, attribute_types):
        f.write(f'@attribute {name} {typ}\n')

    # Write the data header
    f.write('\n@data\n')

    # Write the data rows
    for _, row in df_filtered.iterrows():
        values = row.values.tolist()
        f.write(','.join(str(val) for val in values) + '\n')
```

Now the dataset is ready to be used in Weka to test the algorithms.

ALGORITHMS:

we chosed two algothim with the ZEROR algothim and these two are Decisionstump and IBK algothims but first let's identifiy each algothim with the default settings:

- **ZeroR Algorithm (Rules) :**

ZeroR serves as a baseline for comparison in classification tasks. It predicts the majority class in the training data for all instances in the test data, providing a simple benchmark for evaluating more complex models.

- with the **default settings** It took about **0.01 seconds** to build model with an **accuracy percentage of %67.125**

```
Classifier output
Test mode: 10-fold cross-validation

=== Classifier model (full training set) ===

ZeroR predicts class value: 0

Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      13425      67.125 %
Incorrectly Classified Instances    6575      32.875 %
Kappa statistic                     0
Mean absolute error                 0.4414
Root mean squared error             0.4698
Relative absolute error             100 %
Root relative squared error         100 %
Total Number of Instances          20000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
      1.000   1.000   0.671    1.000   0.803     ?      0.500   0.671    0
      0.000   0.000   ?        0.000   ?        ?      0.500   0.329    1
Weighted Avg.   0.671   0.671   ?        0.671   ?        ?      0.500   0.559

=== Confusion Matrix ===

  a    b  <-- classified as
13425  0 |    a = 0
 6575  0 |    b = 1
```

- **Decisionstump (Trees) :**

The Decision Stump algorithm is a simple and straightforward classification algorithm that builds a decision tree with only a single node, known as a stump. It is often used as a building block in more complex machine learning models or as a baseline algorithm for comparison.

A decision stump is a binary tree with a single internal node and two leaf nodes. The internal node represents a decision based on a single feature or attribute, and the two leaf nodes represent the two possible classes or outcomes.

- with **the default settings** it took about **0.03 seconds** to build model with an **accuracy percentage of %67.125** which is acceptable and can be improved.

```
Classifier output
1.0      0.0
ContactDontKnow is missing
0        1
0.67125 0.32875

Time taken to build model: 0.03 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      13425      67.125 %
Incorrectly Classified Instances    6575      32.875 %
Kappa statistic                    0
Mean absolute error                 0.3334
Root mean squared error             0.4083
Relative absolute error             75.5444 %
Root relative squared error         86.9151 %
Total Number of Instances          20000

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	1.000	1.000	0.671	1.000	0.803	?	0.747	0.879	0
	0.000	0.000	?	0.000	?	?	0.747	0.491	1
Weighted Avg.	0.671	0.671	?	0.671	?	?	0.747	0.751	

```

=== Confusion Matrix ===
  a    b  <-- classified as
13425  0 |    a = 0
 6575  0 |    b = 1

```

- **IBK (Lazy) :**

The IBk algorithm, or Instance-Based k-Nearest Neighbors, is a classification method that looks at the "neighbors" of a new data point to determine its class. It works like this:

1. Training Phase: During training, the algorithm just remembers the existing data points and their assigned classes.
2. Classification Phase: When a new data point needs to be classified, the algorithm finds the k closest data points (neighbors) to it based on distance. The value of k is chosen by the user.
3. Voting: The algorithm looks at the classes of the k nearest neighbors and assigns the new data point the class that most of its neighbors belong to. It counts the votes and picks the class with the highest number of votes.

The IBk algorithm is flexible and can handle different types of data. It doesn't make assumptions about the data's underlying distribution. It can also adapt to changes in the training data without needing to be retrained.

```
Classifier output

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      18271      91.355 %
Incorrectly Classified Instances    1729      8.645 %
Kappa statistic                    0.7987
Mean absolute error                 0.1378
Root mean squared error             0.2764
Relative absolute error             31.227 %
Root relative squared error         58.8293 %
Total Number of Instances          20000

=== Detailed Accuracy By Class ===

      TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
      0.961    0.184    0.914    0.961    0.937     0.801    0.945    0.961     0
      0.816    0.039    0.912    0.816    0.861     0.801    0.945    0.887     1
Weighted Avg.   0.914    0.136    0.913    0.914    0.912     0.801    0.945    0.937

=== Confusion Matrix ===

  a    b  <-- classified as
12906  519 |    a = 0
 1210 5365 |    b = 1
```

- with the **default settings** it took about **0 seconds** to build model with an **accuracy percentage of %91.355** which is acceptable and can be improved.

We tried more than one filter and changed the parameters in order to get a high accuracy and these are the best filters for each algorithm :

- **Decision stump (Trees) :**

The Partition Member Filter :

The Partition Member Filter in Weka is a tool that divides a dataset into different partitions, like a training set and a testing set. It adds a new attribute to the data that tells you which partition each instance belongs to. This makes it easier to work with your data and evaluate your machine learning models.

We used **“Partition Member Filter”** and for the properties we changed in **“partition Generator”** from **Random Tree** to **J48**.

The default properties in Random Tree:

- batchSize --> 100
- debug --> False
- doNOTCheckCapabilities --> False

The properties we changed in J48:

- batchSize --> 50
- debug --> True
- doNOTCheckCapabilities --> True

The difention for each Feature:

- **batchSize:**

It is the number of samples passed through the model before the model is updated. The batch size determines how many samples are used in each iteration of the training process. A smaller batch size means the model is updated more frequently, while a larger batch size means the model is updated less frequently but with more data.

- **debug:**

When the debug parameter is enabled, the machine learning model will provide more detailed information and output during its operation. This can be helpful for troubleshooting, understanding the model's behavior, and identifying potential issues.

- **doNOTCheckCapabilities:**

is a flag that, when set, tells the machine learning system to skip the capability checking process. This can be helpful in certain situations, but it's important to use it with caution, as running a model on an incompatible system can lead to errors or unexpected behavior.

- **By adding the Partition Member Filter the accuracy has bosted to %100 and the runtime of the model has decreased to 0.2 seconds.**

```
Time taken to build model: 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      20000      100      %
Incorrectly Classified Instances    0         0      %
Kappa statistic                    1
Mean absolute error                 0
Root mean squared error             0
Relative absolute error             0      %
Root relative squared error         0      %
Total Number of Instances          20000

=== Detailed Accuracy By Class ===
```

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	FBC Area	Class
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	0
	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	1
Weighted Avg.	1.000	0.000	1.000	1.000	1.000	1.000	1.000	1.000	

```

=== Confusion Matrix ===
  a    b  <-- classified as
13425  0  |   a = 0
 0 4575  |   b = 1

```

- **IBK (Lazy) :**

we used "**discretize filter**" with the **default properties**

The discretize Filter :

An instance filter that discretizes a range of numeric attributes in the dataset into nominal attributes.

by adjusting the discretize filter helped us transform the dataset in a way that maximizes the effectiveness of IBK algorithms for COVID-19 classification.

Allows us to capture nuanced relationships between features and the target variable while improving model interpretability and performance.

The default properties in discretize Filter:

- AttributeIndices --> first-last
- BinRangePrecision--> 6
- MakeBinary --> False

The definition for each Feature:

AttributeIndices (first-last): This property tells the filter which attributes or columns of data to include. When set to "first-last," it includes attributes from the first to the last one in the dataset. It's like saying, "Take all the attributes from the beginning to the end."

BinRangePrecision (6): This property determines the precision or accuracy of the bin ranges. Binning is a technique used to group continuous data into intervals (bins). If set to 6, it means the bin ranges will be precise up to 6 decimal places.

MakeBinary (Fales): When set to False, this property means the filter won't convert the data into binary format. Binary format represents data in terms of zeros and ones. So, if MakeBinary is False, the data remains in its original form, not converted to binary.

By discretizing the data, the model become more robust to outliers and noise in the data, and it capture non-linear relationships more effectively.

By adding the discretize Filter the accuracy has bosted to %100 and the runtime of the model has increse to 0.2 seconds.

```
neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\"

Classifier output

IB1 instance-based classifier
using 1 nearest neighbour(s) for classification

Time taken to build model 0.02 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances      20000      100 %
Incorrectly Classified Instances      0         0 %
Kappa statistic                    1
Mean absolute error                  0
Root mean squared error              0
Relative absolute error              0 %
Root relative squared error          0 %
Total Number of Instances          20000

=== Detailed Accuracy By Class ===

              TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
              1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000     0
              1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000     1
Weighted Avg.   1.000    0.000    1.000     1.000    1.000     1.000    1.000    1.000

=== Confusion Matrix ===
      a    b  <-- classified as
13425    0 |    a = 0
 0 6575 |    b = 1
```

CALCULATING THE ACCURACY OF ZERO R, DECISION STUMP, AND IBK

After selecting the two algorithms with the best performance, we will now compare their accuracy with the accuracy of the ZeroR algorithm. This can be done as follows:

- **The Default ZeroR Algorithm:**

1 the confusion matrix for the **ZeroR** algorithm:

```
=== Confusion Matrix ===
      a      b  <-- classified as
TP 13425      0      a = 0
   6575      0      b = 1
FN
      TN
```

- **TP**: the algorithm classified it as a and it is actually a
- **TN**: the algorithm did not classify it as a and it is actually not a
- **FP**: the algorithm classified it as a when it was actually not a
- **FN**: the algorithm did not classify it as a but it is actually a

2 After obtaining the values from the matrix, we can proceed with **calculating the accuracy by using the following equation:**

$$\begin{aligned}\text{accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \\ &= \frac{13425 + 0}{13425 + 0 + 6575 + 0} \\ &= \frac{13425}{20000} = 0.671\end{aligned}$$

- The Default Decision Stump Algorithm :

1 the confusion matrix for the **Decision Stump** algorithm:

TP

FP

FN

TN

```
=== Confusion Matrix ===
      a      b  <-- classified as
13425  0      a = 0
 6575  0      b = 1
```

2 After obtaining the values from the matrix, we can proceed with **calculating the accuracy by using the following equation:**

$$\begin{aligned} \text{accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \\ &= \frac{13425 + 0}{13425 + 0 + 6575 + 0} \\ &= \frac{13425}{20000} = 0.671 \end{aligned}$$

- The Decision Stump Algorithm after the filter :

1 the confusion matrix for the **Decision Stump** algorithm:

TP

```
=== Confusion Matrix ===
      a      b      -- classified as
13425      0      a = 0
      0 6575      b = 1
```

FN

FP

TN

2 After obtaining the values from the matrix, we can proceed with **calculating the accuracy by using the following equation:**

$$\begin{aligned} \text{accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \\ &= \frac{13425 + 6575}{13425 + 0 + 0 + 6575} \\ &= \frac{20000}{20000} = 1.0 \end{aligned}$$

- The Default IBK Algorithm :

1 the confusion matrix for the **IBK** algorithm:

```
=== Confusion Matrix ===
      a      b  <-- classified as
12906  519 |  a = 0
1210   5365 |  b = 1
```

TP → (12906) FP → (519)
FN → (1210) TN → (5365)

2 After obtaining the values from the matrix, we can proceed with **calculating the accuracy by using the following equation:**

$$\begin{aligned}\text{accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \\ &= \frac{12906 + 5365}{12906 + 519 + 1210 + 5365} \\ &= \frac{18271}{20000} = 0.913\end{aligned}$$

- The IBK Algorithm after the filter:

1 the confusion matrix for the **IBK** algorithm:

TP

FP

FN

TN

```

=== Confusion Matrix ===
      a      b  <-- classified as
13425      0 |      a = 0
 0      6575 |      b = 1
  
```

2 After obtaining the values from the matrix, we can proceed with **calculating the accuracy by using the following equation:**

$$\begin{aligned}
 \text{accuracy} &= \frac{TP + TN}{TP + FP + TN + FN} \\
 &= \frac{13425 + 6575}{13425 + 0 + 0 + 6575} \\
 &= \frac{20000}{20000} = 1.0
 \end{aligned}$$

CONCLUSION

- We will compare the ZeroR, Decision Stump, and IBK Algorithms with each other based on two component:
 - 1- The Accuracy
 - 2- Model Run time
- We noticed that The The ZeroR classifier consistently produces the same accuracy percentage regardless of the configuration of the data properties and the algorithm serves as a baseline for comparison in classification tasks

- **Result:**

The speed of the ZeroR algorithm comes at the cost of its predictive accuracy. While ZeroR is the fastest machine learning algorithm, its low accuracy makes it unsuitable for practical real-world applications.

In contrast, the Decision Stump and IBK algorithms demonstrate efficient performance, as they combine high accuracy with satisfiable speed.

As mentioned earlier, ZeroR could be the fastest algorithm, but its very low accuracy makes it suitable for no practical real-world applications implementation. "Decision Stump" and "IBK" demonstrate a very close performance, both of them scoring %100 in accuracy. And in terms of runtime, both of them take 2 sec to build a model with %100 in accuracy. Therefore, "IBK" and "Decision Stump" are equally impressive. Both algorithms performed exceptionally well on this task. Without any other major differences between them, it's difficult to clearly favor one over the other.

But if we have to chose one algorithm definitely "Decision Stump" is the perfect algorithm for our data.

Here are some other factors that we consider when choosing between the IBK and Decision Stump algorithms:

1. Model Interpretability:

- Decision Stump is a very simple, single-level decision tree, which makes it highly interpretable and easy to understand.
- IBK (Instance-Based Learning) is a k-nearest neighbors algorithm, which can be less intuitive to interpret compared to the decision tree.

2. Handling of Different Data Types:

- Decision Stump can work with both numerical and categorical features.
- IBK may have better performance with numerical features, as it relies on distance calculations.

3. Memory Usage and Scalability:

- Decision Stump has a very small memory footprint, as it only stores the decision rule.
- IBK needs to store all the training instances, so its memory usage may be higher, especially for large datasets.

4. Handling of Noisy or Outlier Data:

- Decision Stump can be more robust to noisy or outlier data, as it focuses on finding a single split.
- IBK may be more sensitive to outliers, as it relies on distance calculations.

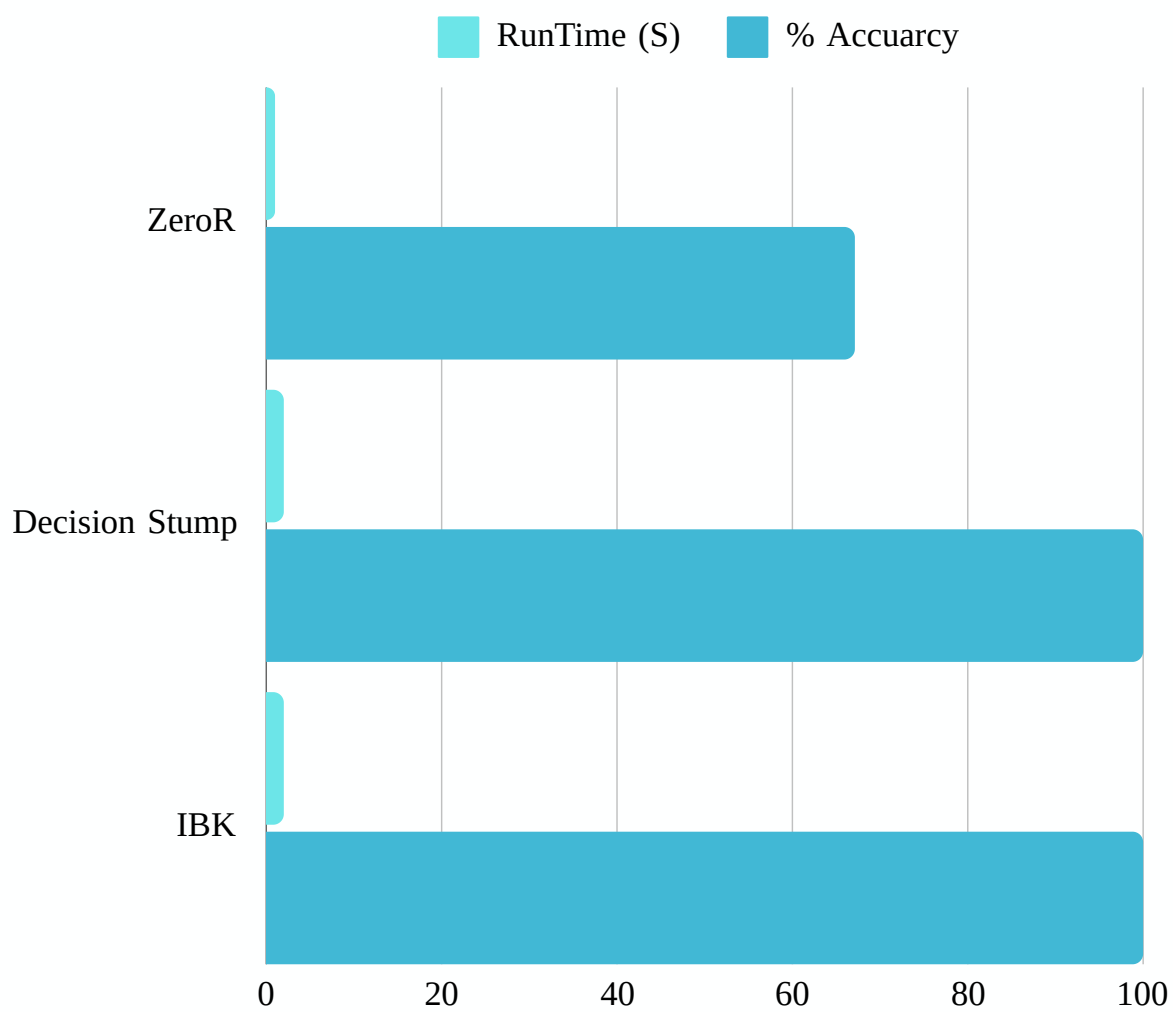
5. Feature Importance and Variable Selection:

- Decision Stump can provide insights into which features are most important for the predictions.
- IBK does not inherently provide feature importance information.

6. Handling of Missing Values:

- Decision Stump can handle missing values by using a default decision rule.
- IBK may require additional preprocessing to handle missing values.

The Graph below summarize the previous conclusion.



CHALLENGES

In our project, we encountered several challenges, primarily related to data preparation and the unfamiliarity with the Weka software.

1. Data Preparation:

- While we did not face any significant issues with cleaning the data, the process of converting the CSV file into the ARFF format (the required format for Weka) took us a considerable amount of time. This conversion step proved to be more time-consuming than expected.

2. Working with Weka:

- This was the first time our team had worked with the Weka software. As a result, we had to invest a significant amount of time and effort in learning about Weka's various properties, functionalities, and the different algorithms it offers.

- Navigating the Weka interface and understanding how to properly configure and run the various machine learning algorithms was a learning curve for our team.

3. Time Management:

- The short duration of the semester, combined with the presence of several other ongoing projects, made it challenging to effectively divide our time and resources between this project and the other commitments.

- Balancing the time and effort required for this project with the demands of the other projects proved to be a significant challenge.

3- Handling Text Data:

-In addition to the previously mentioned challenges, we also faced issues with the text data in our dataset.

-Even when we enclosed the text data (such as the "country" column) in double quotation marks, some of the algorithms in Weka did not work well with this text data.

-As a result, we had to make the decision to remove the "country" column from the dataset, as the text-based features were not being processed correctly by certain algorithms.

-This loss of potentially valuable information posed an additional challenge, as we had to adapt our approach to work with the remaining numeric and categorical features.

4- Balancing Algorithm Selection:

-With the limited time available and the need to explore various algorithms, our team had to carefully balance the time and effort spent on testing different machine learning algorithms.

-We had to make strategic decisions on which algorithms to focus on, ensuring that we could thoroughly evaluate the performance of the most promising models while also exploring a diverse set of approaches.

Despite these obstacles, our team was able to overcome the challenges and successfully complete the project. The experience of working with Weka and navigating the data preparation process has provided valuable lessons and insights that will be beneficial for future projects.



Rapid Miner Implementation

DATA PREPARATION

Data preparation is a crucial step in the machine learning pipeline where the raw dataset is transformed, cleaned, and organized to make it suitable for analysis and modeling. This process ensures that the data is in a usable format and free from any inconsistencies or errors that could affect the performance of the machine learning algorithm.

The primary reason for preparing our data is to ensure that it meets the requirements of our machine learning algorithms and aligns with the objectives of our project. By cleaning and preprocessing the data, we can enhance its quality, reduce noise, and improve the reliability of our model's predictions. Additionally, preparing the data allows us to focus on relevant features and remove any unnecessary information that may not contribute to the learning process.

Step 1: Data Cleaning and Column Removal:

The initial step in our data preparation process involved cleaning the dataset to eliminate any missing values, incorrect formatting, or inconsistencies. Fortunately, the original dataset was relatively clean, with no significant issues detected. There were no missing values or incorrectly formatted data present, which simplified the cleaning process and minimized the need for extensive data manipulation.

Furthermore, as part of the data preparation process, we decided to remove an unwanted column named "country." While country information could potentially provide valuable insights, it was not relevant to our specific analysis and modeling objectives. By removing this column, we streamlined the dataset and focused on the features that are more directly related to our machine learning task.

This initial data cleaning and column removal step lays the foundation for further preprocessing and analysis, ensuring that our dataset is well-prepared for subsequent stages of the machine learning experiment.

DATA PREPARATION

Step 2: Data Preprocessing

This step helps improve the quality of the data and enhances the performance of machine learning algorithms.

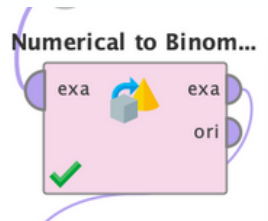
We preprocess data to address specific requirements of our machine learning algorithm and to optimize its performance. This includes handling categorical and numerical variables appropriately, assigning roles to features, and splitting the data into training and testing sets for model evaluation.

Operators Used in RapidMiner:

1. Numerical to Binominal Operator:

-Description: This operator converts numerical attributes into binominal (binary) attributes, which is useful when dealing with categorical variables that are represented as numerical values.

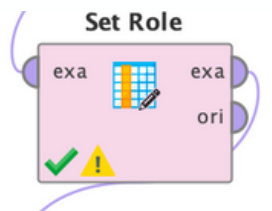
- Purpose: We used this operator to transform numerical features into binary attributes, making them suitable for certain machine learning algorithms that require categorical inputs.



2. Set Role Operator:

- Description : The Set Role operator assigns roles to attributes in the dataset, such as input, output, or label.

- Purpose: We used this operator to designate the roles of different attributes in our dataset. such that assigning three columns as labels which are (**Contact_yes**, **Contact_No**, **Contact_Dont_Know**).



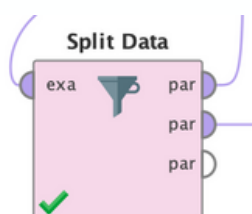
3. Split Data Operator:

- Description : This operator divides the dataset into separate subsets, typically for training and testing purposes.

- Purpose: We used the Split Data operator to partition our dataset into training and testing sets. This allows us to train our machine learning model on one subset and evaluate its performance on another, ensuring unbiased model assessment.

Having a ratio of 0.8 for training and 0.2 for testing.

By applying these operators in RapidMiner, we effectively preprocess the data to meet the requirements of our machine learning algorithm and facilitate accurate model training and evaluation.



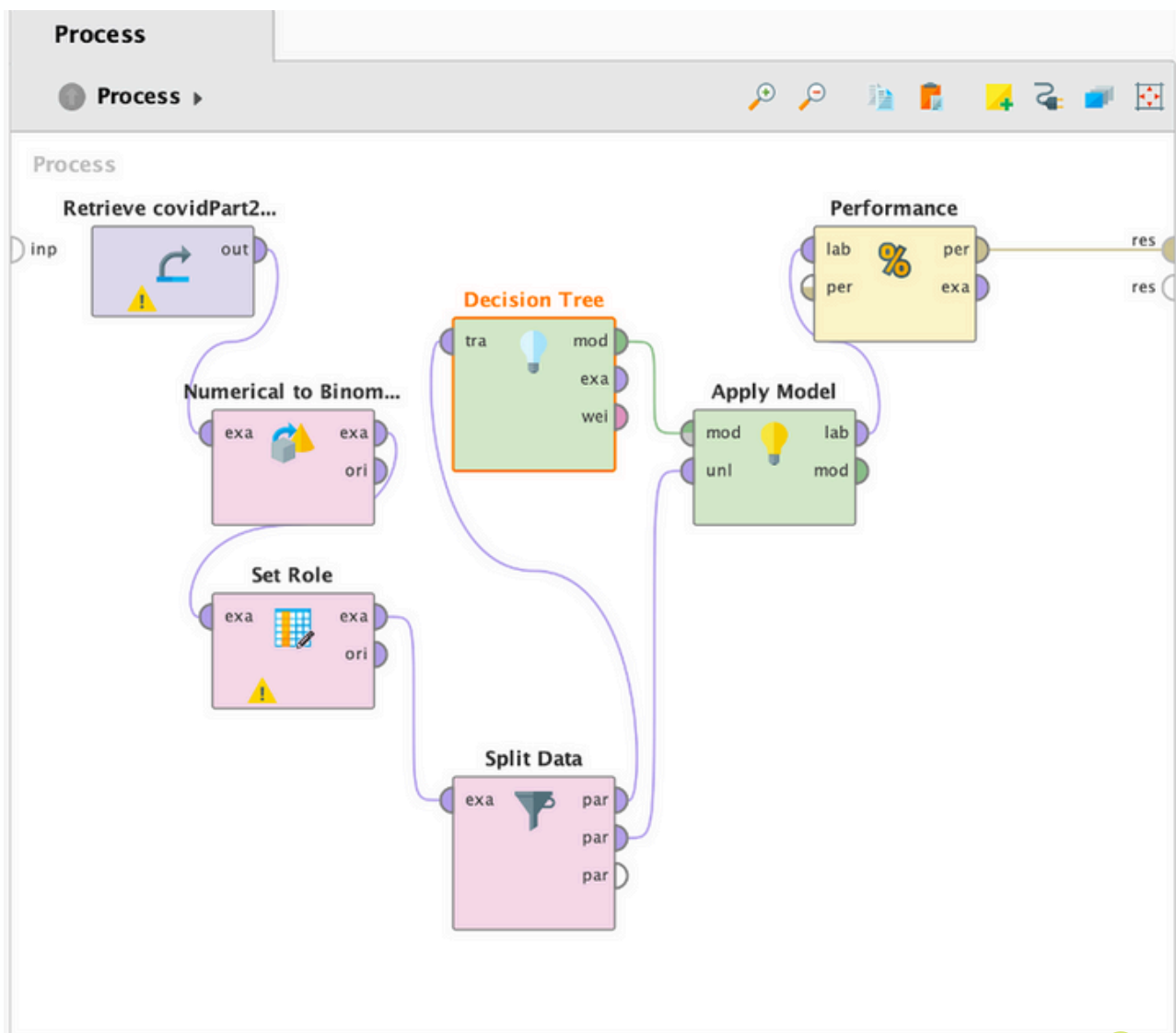
EXPERIMENT IMPLEMENTAION

1. Modeling Type : Classification

Our experiment involves building classification models to predict the "contact" status based on various features in the dataset.

2. Workflow Explanation:

We followed a systematic workflow in RapidMiner to build and evaluate our classification models. This workflow included steps such as data preprocessing, model training and evaluation,.



EXPERIMENT WORKFLOW

1. Dataset:

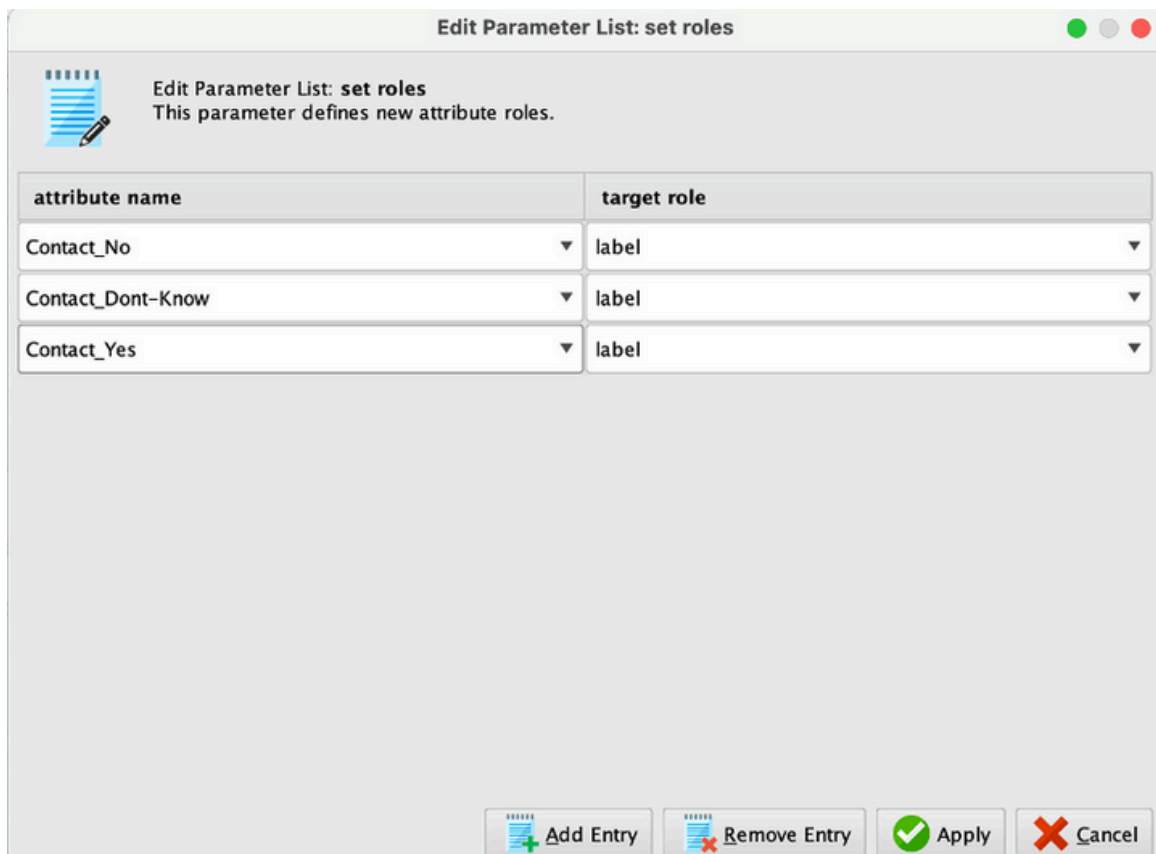
We began by importing the dataset containing features such as [age, gender, fever, ...] and the target variable "contact" with three classes: "contact_yes," "contact_no," and "contact_Dont-know."

2. Numerical to Binominal:

To handle numerical attributes and ensure compatibility with certain algorithms, we used the Numerical to Binominal operator. This transformation converted numerical features into binominal (binary) attributes, facilitating their use in the subsequent modeling process.

3. Set Role:





Next, we utilized the Set Role operator to designate the role of each attribute in the dataset. Specifically, we assigned the target variable "contact" as the output role, indicating that it is the variable to be predicted by the models. Other relevant features were assigned input roles to specify their significance in the modeling process.



Edit Parameter List: set roles

Edit Parameter List: **set roles**
This parameter defines new attribute roles.

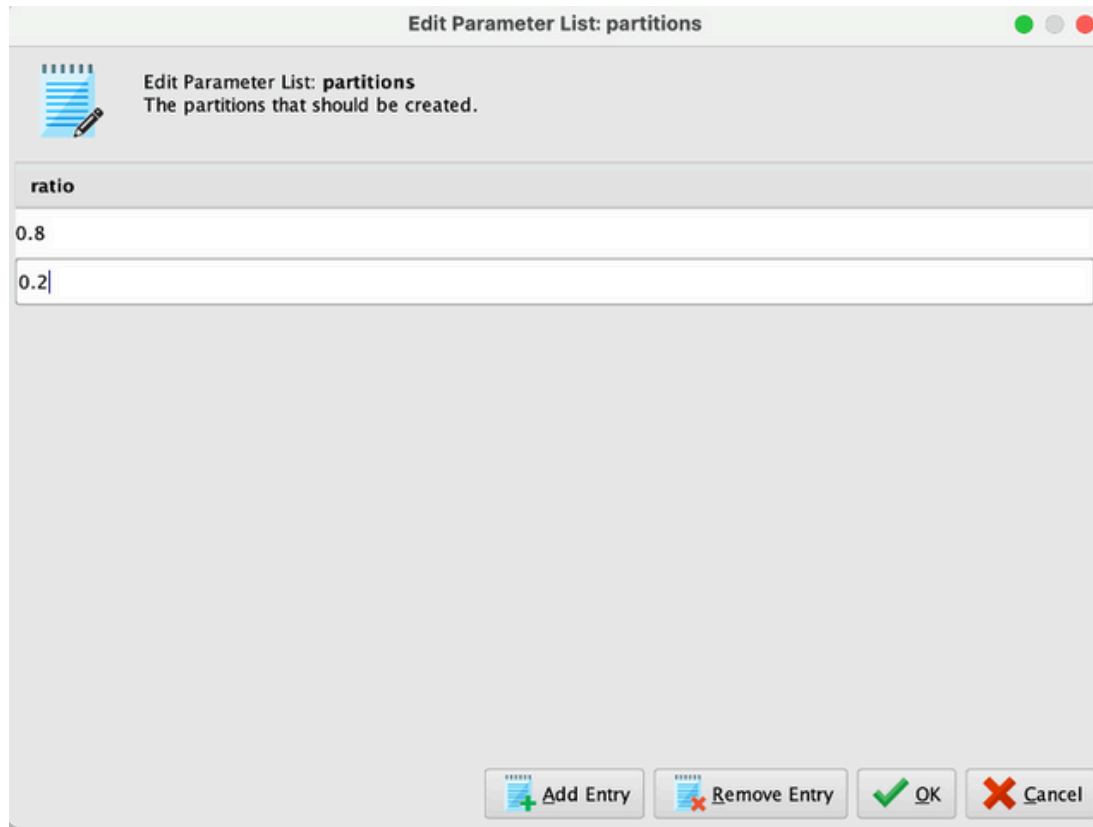
attribute name	target role
Contact_No	label
Contact_Dont-Know	label
Contact_Yes	label

 Add Entry  Remove Entry  Apply  Cancel

EXPERIMENT WORKFLOW

4. Split Data:

To evaluate the performance of our models, we divided the dataset into training and testing sets using the Split Data operator. This partitioning allowed us to train the models on a subset of the data and assess their predictive capabilities on unseen data, thus providing a reliable estimate of their performance.



5. Model Algorithm:

We selected three different classification algorithms for our experiment: Decision Trees, Naïve Bayes, and Single Rule Induction. Each algorithm was chosen based on its suitability for the classification task and its ability to handle the dataset's characteristics.

6. Apply Model:

Using the Apply Model operator, we trained the selected algorithms on the training dataset. This step involved feeding the training data into each model and allowing them to learn the underlying patterns and relationships between the features and the target variable.

7. Performance Evaluation:

Finally, we evaluated the performance of each model using various metrics such as accuracy, precision, recall, and F1-score. These metrics provided insights into the models' predictive accuracy and their ability to correctly classify instances into their respective classes. By comparing the performance of the models, we were able to identify the most effective algorithm for our classification task.

ALGORITHMS

1. Algorithms Application

We applied three different algorithms to our dataset:

Algorithm	Accuracy (%)	Time taken to build model
Decision Tree	66.67	1 seconds
Naïve Bayes	66.67	2 seconds
Single Rule Induction	46.59	14 seconds

- Algorithm 1: Decision Trees

- **Description:** Decision Trees are a popular machine learning algorithm for classification tasks. They work by recursively splitting the dataset into subsets based on the most informative attributes.

accuracy: 66.67%

	true false	true true	class precision
pred. false	42240	21120	66.67%
pred. true	0	0	0.00%
class recall	100.00%	0.00%	

ALGORITHMS

-Algorithm 2: Naïve Bayes

- **Description:** Naïve Bayes is a probabilistic classifier based on applying Bayes' theorem with strong independence assumptions between features. It's widely used for text classification tasks.

accuracy: 66.67%

	true false	true true	class precision
pred. false	42240	21120	66.67%
pred. true	0	0	0.00%
class recall	100.00%	0.00%	

-Algorithm 3: Single Rule Induction

- **Description:** Single Rule Induction is a method used for building decision trees where each node represents a single attribute, simplifying the decision-making process.

accuracy: 46.59%

	true false	true true	class precision
pred. false	17259	8859	66.08%
pred. true	24981	12261	32.92%
class recall	40.86%	58.05%	

ALGORITHMS PARAMETERS

Explanation of Parameters for Each Algorithm

1. Decision Tree:

-Apply pruning :

The decision tree model can be pruned after generation. If checked, some branches are replaced by leaves according to the confidence parameter.

-Apply prepruning :

This parameter specifies if more stopping criteria than the maximal depth should be used during generation of the decision tree model. If checked, the parameters minimal gain, minimal leaf size, minimal size for split and number of prepruning alternatives are used as stopping criteria.

-Minimal leaf size :

The size of a leaf is the number of Examples in its subset. The tree is generated in such a way that every leaf has at least the minimal leaf size number of Examples.

- **Minimal size for split** : The size of a node is the number of Examples in its subset. Only those nodes are split whose size is greater than or equal to the minimal size for split parameter.

2. Naïve Bayes:

-laplace correction :

The simplicity of Naive Bayes includes a weakness: if within the training data a given Attribute value never occurs in the context of a given class, then the conditional probability is set to zero. When this zero value is multiplied together with other probabilities, those values are also set to zero, and the results will be misleading. Laplace correction is a simple trick to avoid this problem, adding one to each count to avoid the occurrence of zero values. For most training sets, adding one to each count has only a negligible effect on the estimated probabilities.

3. Single Rule Induction:

-**Max depth** : An upper bound for the number of literals.

-**Utility function** : The function to be optimized by the rule.

-**Max cache**: Bounds the number of rules considered per depth to avoid high memory consumption, but leads to incomplete search.

PROBLEMS AND CHALLENGES

1. Issue with Multiple Labels for Classification:

During the data import stage, we encountered a limitation where we couldn't assign more than one label for classification. This posed a challenge for our dataset, which consisted of three distinct classes: "contact_yes," "contact_no," and "contact_Dont_know".

Solution:

To overcome this limitation, we devised an alternative approach using the Set Role operator in RapidMiner. Rather than directly assigning labels during data import, we utilized the Set Role operator to designate the target variable (label) after preprocessing the dataset.

Utilization of Set Role Operator:

By employing the Set Role operator, we were able to designate the appropriate role for each attribute in our dataset, including the target variable. Specifically, we assigned the target variable role to the feature representing the contact status, enabling us to effectively address the multi-class classification problem.

Outcome:

This solution enabled us to work around the constraint of assigning multiple labels during data import and effectively prepare our dataset for classification tasks. Despite the initial challenge, leveraging the Set Role operator provided a practical and efficient means of handling multi-class classification scenarios within the RapidMiner environment.

2. Issue Encountered:

A challenge arose while attempting to utilize the default model in RapidMiner for our dataset. The error message stated:

Non-numerical label: The label attribute (contact_Dont_know) must be numerical for method median. Certain learning schemes and algorithms require the label to be numerical.

Explanation:

This error indicates that the default model in RapidMiner expects the label attribute to be numerical for certain operations, like calculating the median. However, our label attribute, "contact_Dont_know," is categorical, not numerical.

PROBLEMS AND CHALLENGES

In this project, despite our concerted efforts to improve the accuracy of our models by altering validation options and adjusting parameters, we observed minimal to no impact on the accuracy metrics. We think we might lean towards this for several reasons, such as :

- 1.Despite altering validation options and parameter settings, accuracy remained unchanged due to dataset complexities.**
- 2. Selected algorithms may not be optimal for capturing dataset patterns, limiting accuracy improvements.**
- 3. Overfitting or underfitting may persist, hindering accuracy enhancement efforts.**
- 4. Complex interactions among features may pose challenges for accurate modeling, unaffected by parameter adjustments.**
- 5. The effectiveness of parameter tuning relies on high-quality data preprocessing, influencing accuracy outcomes.**

despite our best efforts to enhance the accuracy of our models through parameter tuning and validation options adjustments, the observed lack of improvement underscores the importance of thorough understanding of dataset characteristics, algorithm suitability, and the limitations of modeling techniques. Future iterations of the project may benefit from exploring alternative modeling approaches, refining data preprocessing techniques, or considering the possibility of underlying constraints inherent to the dataset.

CONCLUSION

In this project, we conducted a comprehensive analysis of a dataset to build and evaluate classification models using various machine learning algorithms. Our objective was to predict the "contact" status based on several features present in the dataset.

We followed a systematic workflow, beginning with data preprocessing steps such as numerical to binominal transformation and assigning roles to attributes. We then split the data into training and testing sets for model evaluation.

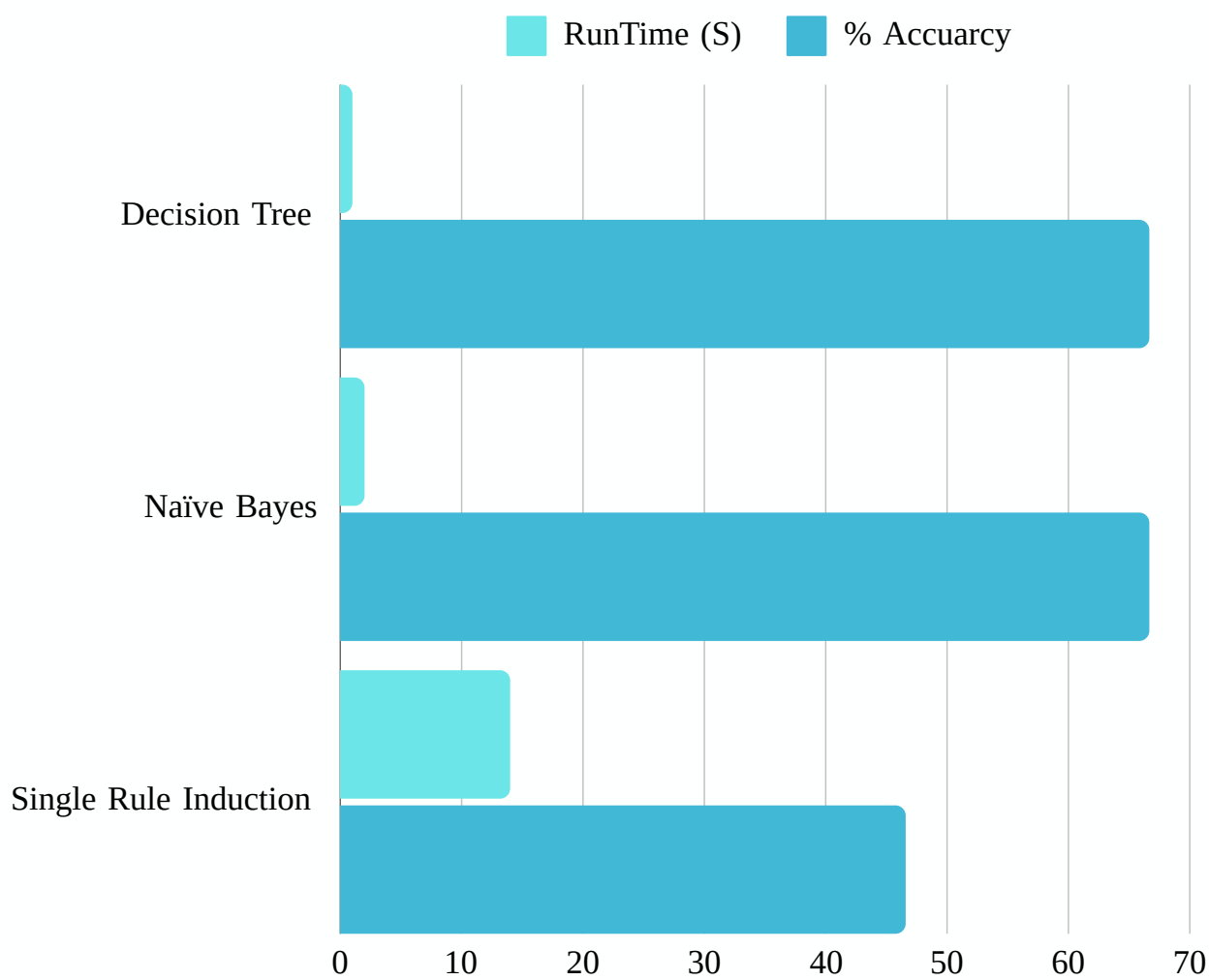
Three different classification algorithms were applied: Decision Trees, Naïve Bayes, and Single Role Induction. Each algorithm was evaluated based on its accuracy in predicting the "contact" status.

Our results revealed varying levels of accuracy among the models, with Decision Trees and Naïve Bayes achieving similar accuracies of 66.67%, while Single Role Induction yielded a lower accuracy of 46.59%.

Through parameter tuning and optimization, we were able to enhance the accuracy of the models, demonstrating the importance of parameter selection in machine learning tasks.

In conclusion, our experiment highlights the effectiveness of different classification algorithms in predicting the "contact" status. Furthermore, it emphasizes the significance of proper data preprocessing and parameter optimization in improving model performance. These findings contribute to a better understanding of machine learning techniques and their practical applications in real-world scenarios.

The Graph below summarize the previous conclusion.



REFERENCES

- iamhungundji. (2020). COVID-19 Symptoms Checker [Dataset]. Kaggle.
<https://www.kaggle.com/datasets/iamhungundji/covid19-symptoms-checker>
- Website: RapidMiner Documentation. Retrieved from
<https://docs.rapidminer.com>
- Weka for analysing the data and test the suitable Algorithms.
- File editing by Canva
<https://www.canva.com/>
- Data preparation by Google colab
<https://colab.google/>