AN AUTOMATED MACHINE LEARNING APPROACH FOR SMART WASTE MANAGEMENT SYSTEMS

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

ASHISH CHOWDARY VELAGA
(Reg.No.: 122003039, B.Tech. - CSE)
PILLALAMARRI SATHWIK
(Reg.No.: 122003181, B.Tech. - CSE)
TALASILA VENKATESH
(Reg.No.: 122003267, B.Tech. - CSE)

January 2022

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA - 613 401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 OF THE UGC ACT, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR - 613 401

Bonafide Certificate

This is to certify that the report titled **"An Automated Machine Learning Approach for Smart Waste Management Systems"** submitted as a requirement for the course, CSE300 : **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Ashish Chowdary Velaga (Reg. No.122003039, B.Tech. - CSE), Mr. Pillalamarri Sathwik (Reg. No.122003181, B.Tech. - CSE), Mr. Talasila Venkatesh (Reg. No.122003267, B.Tech. - CSE)** during the academic year 2021-22, in the School of Computing, under my supervision.

Signature of Project Supervisor : 

Name with Affiliation :Dr. M. Ifjaz Ahmed, Assistant Professor-II, IT/SoC

Date :08 – 01 – 2022

Mini Project *Viva voce* held on _____

Examiner – 1 **Examiner – 2**

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology and **Dr. B. Santhi**, Associate Dean, Department of Computer Application.

Our guide **Dr. M. Ifjaz Ahmed**, Assistant Professor - II, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing mean opportunity to showcase my skills through project.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

KNN	-	K Nearest Neighbors
SVM	-	Support Vector Machine
LR	-	Logistic Regression
DT	-	Decision Tree
RF	-	Random Forest
MLP	-	Multi Layer Perceptron
MCC	-	Matthews Correlation Coefficient
IoT	-	Internet of Things

NOTATIONS

FL	-	Filling Level
FL1	-	Filling Level Before
FL2	-	Filling Level After
ΔFL	-	Filling Level Change
V_{str}	-	Vibration Strength Score

ABSTRACT

One of the most significant issues created by the fast development of the urban population is waste management. The smart waste management system was developed to address the issue of trash transportation efficiency. The system will allow for the prediction of the recycling container's predicted emptying time. Predicting fill levels will allow redundant transportation to be avoided. Each recycling container in the system is fitted with a sensor that is positioned on the inside. For reliable detection of emptying a recycling container, measurements from the sensor positioned on top of the container are employed.

The existing manually constructed model and its modification, as well as traditional machine learning algorithms such as Artificial Neural Networks, K-nearest neighbors, Linear Regression, Support Vector Machine, and Random Forest, are among the methodologies explored. The existing manually constructed model's classification accuracy and recall were improved from 86.8% and 47.9% to 99.1% and 98.2% thanks to the introduction of machine learning algorithms. The Random Forest classifier, which was the highest performing option, enhanced the quality of forecasts for recycling container emptying times. The highest performing solution increased accuracy by 12.3% and recall by 50.3 percent, while also increasing the FI score by 0.347 and the MCC score by 0.366 percent.

KEYWORDS: Smart Waste Management, Emptying detection, Classification Algorithms, Data mining, Automated Machine Learning, Grid Search.

CHAPTER 1

SUMMARY OF THE BASE PAPER

Title	:	An Automated Machine Learning Approach for Smart Waste Management Systems.
DOI	:	10.1109/TII.2019.2915572
Journal Name	:	IEEE Transactions on Industrial Informatics
Publisher	:	IEEE
Year	:	2019
Indexed	:	SCI

One of the most significant issues created by the expanding urbanization is garbage management. Several steps make to an effective method for dealing with the waste management problem. To begin, we must consume things in a sensible manner in order to eliminate unnecessary waste. Following that, garbage disposal should be done in a systematic manner. Finally, garbage should be recycled to the greatest extent possible. Economic and environmental considerations should be taken into account when implementing these steps.

Waste disposal has a large impact on both factors, and optimizing it can increase the positive effects greatly. Simultaneously, there is a clear requirement that recycling bins be emptied on a regular basis to ensure them clean. Meeting this condition is tough in a situation with thousands of recycling stations (each one with numerous containers) distributed across a large geographic area.

A Smart Trash Management system that combines Internet of Things components is an enabling technology for tackling waste transportation optimization challenges. It will allow each recycling container to keep track of how full it is. The advanced ability of such a system will enable the forecast of a recycling container's estimated emptying time, i.e. whenever the bin container's filling level reaches a threshold value. By predicting fill levels, superfluous transportation can be avoided without violating the overfilling rule.

The primary purpose of this project is to investigate and test a method for determining when a container has been emptied. This issue can be broken down into the following sub-issues:

P1 Examine and assess the legacy solution's strengths and limitations.
P2 Compile statistics to estimate the legacy solution's confidence level, which will be utilized as a baseline for the alternative alternatives.
P3 Research a variety of strategies that might be used to solve the problem.
P4 Put one or more of the most promising solutions uncovered during the study phase into practise and test them.

P5 Examine the findings, compare them to the legacy solution, and draw implications for future enhancements.

The hardware includes an ultrasonic range sensor, accelerometer, and GSM module. The ultrasonic sensor detects the quantity of filling in a recycling bin at regular intervals throughout the day. Because the sensor is attached to the container's ceiling, the recorded range is a length from the ceiling to the current trash level. An accelerometer is also included in the gadget to provide additional data for detecting potential emptying. The accelerometer continuously measures the container's acceleration. If the acceleration reaches a certain threshold, the sensor sends an interrupt to the processor. The processor keeps track of how many interruptions it receives over a given amount of time.

The detection of emptying presents several difficulties. If the data recorded by the ultrasonic sensor or accelerometer surpassed the threshold, a threshold-based model would detect it. However, there are certain practical limitations to using such a model. Because of the ultrasonic sensor's physical characteristics, things other than garbage can be detected. It is possible to measure the level. Containers for recycling, for example usually include supporting structures or other components that are related to the ultrasonic pulses are disrupted by the emptying mechanism. As a result, a false echo is created. Because of the false echo, the Even if the recycling rate is 100%, the amount of filling will never be zero. Actually, the container is empty. This fact debunks the notion of Detecting the absence of garbage by measuring the absence of waste.

When the recycling bins are lifted with a crane and emptied, the accelerometer should record a single unique vibration sample. The emptying of a container, however, is not the only event that creates vibration samples. Additional vibrations are often felt when waste is thrown into a recycling container. As a result, accelerometer measurements may produce a large number of false positives.

The existing manual-engineered model consists of many combination rules and three thresholds. These thresholds were previously determined using expert knowledge in the existing model. A set of static rules is used in the existing manually developed model for emptying detection. Every second hour, these restrictions are enforced to keep an eye out for a fresh vibration strength score (denoted as V_{str}). The Only the vibration is used in the existing manually constructed model.

Parameter	Legacy	Optimised
V_{str}	≥ 6	≥ 0
FL_2	$< 10\ \%$	$< 77\ \%$
ΔFL	$< -60\ \%$	$< -20\ \%$

Table 1.1 Optimized and legacy parameters for the legacy solution

The ultrasonic readings and the strength score (filling level) immediately before and after the vibration strength score in a timely manner As a result, the rules are vulnerable to minor measurement errors in the proximity of an emptying.

The filling level before the vibration strength score is FL1, and the filling level after is FL2. The change in filling level is ΔFL, and it is calculated with the formula ΔFL = FL2-FL1. Vstr must be greater than six, i.e., or more five interrupts must have been logged, for a vibration strength score to be regarded an emptying.

Traditional classification techniques were applied to the situation in problem to see if they might increase performance even more than the manually optimized results. Model that has been engineered Six algorithms for classification have been introduced. Six classification algorithms introduced before were investigated: ANN, kNN, LR, SVM, DT, and RF.

The algorithms are evaluated on the basis of metrics called Accuracy, Precision,Recall, F1 Score and MCC Score. The following are the formulae for each of the mentioned metrics :

Accuracy = (TP+TN) / (TP+FP+TN+FN)
Precision = TP / (TP+FP)
Recall = TP / (TP+FN)
F1 Score = 2*(Recall * Precision) / (Recall + Precision)
MCC Score = (TP . TN) - (FP . FN) / √ ((T P + F P) · (T P + F N) · (T N + F P) · (T N + F N)

Manually engineered model and machine learning approaches with the same characteristics as manually engineered models were the two types of solutions. When it came to appropriately categorising the real emptying, the current manually generated model performed poorly in the first category. This explains the low recall, F1, and MCC scores. The observed accuracy was poor since the data set was slanted towards non-emptying. Nonetheless, by tweaking the threshold values for the manually generated model, all tested performance metrics saw a significant improvement. It emphasizes the importance of data analysis as a method for developing expert systems.

It's worth mentioning that the second group included two different types of classification algorithms: ensemble classifiers (RF) and single-model classifiers (all but RF). In terms of a fair comparison of algorithms, ensemble classifiers should be evaluated independently, but in the context of the circumstances at hand, the acquired performance was the major goal. The use of single classifiers did not result in a qualitative performance boost when compared to the manually constructed model that was optimized. In actuality, all performance measurements were marginally better or equal to the manually generated model that was optimised while using ANN. Other single classifiers (SVM and DT) performed much better in some but not all metrics, or performed poorly in all indicators (kNN and LR).

CHAPTER 2

MERITS AND DEMERITS

One of the most significant issues created by the fast development of urban populations is waste management. Recycling stations should be emptied on a regular basis to keep them clean. The issues of garbage transportation are addressed by a smart waste management system that uses the Internet of Things and Machine Learning. Because inaccurate detection lowers the value of filling level predictions, detecting container emptying is a crucial stage in producing qualitative predictions.

A set of static rules is used in the existing manually developed model for emptying detection. Every second hour, these restrictions are enforced to keep an eye out for a fresh vibration strength score (denoted as V_{str}). The Only the vibration is used in the existing manually constructed model. The ultrasonic readings and the strength score (filling level) immediately before and after the vibration strength score in a timely manner As a result, the rules are vulnerable to modest measurement errors in the vicinity of an emptying.

As there is a flaw in manual engineering model, the classification algorithms are used on the data set to check whether there is an improvement in Accuracy, Recall, Precision, F1 Score & MCC Score.

Merits

The Auto ML approach is used in conjunction with data-driven methodology to solve a problem in the field of waste management, which is one of the most significant difficulties given by the increasing development of the urban population. An enabling technology for solving the issues of waste transportation optimization is a Smart Waste Management system that incorporates components of the Internet of Things.

Using readings from a sensor installed on top of the container, an automated machine learning approach is utilized to accurately detect the emptying of a recycling container. This method produces better results in a more efficient manner, with greater accuracy and precision, allowing us to estimate when the container will be full so that it may be emptied appropriately.

The best results were obtained using an iterative data-driven methodology in which the existing solution to the problem was first evaluated, then this solution was optimized using the gathered data set, then machine learning algorithms were implemented to the problem, and eventually, feature engineering was used to see if adding extra features would improve the results. The introduction of machine learning allowed the existing manually constructed model's accuracy of classification and recall to be improved.

Demerits

The work is only focused on the emptying detection component, as effective emptying detection is likely the most crucial a requirement for accurate forecasting of emptying times. In addition, the historical solution for predicting the emptying time. If a precise emptying detection is obtained, the situation will vastly improve.

Even if the results reveal excellent performance, additional areas of research may still be necessary. Is it possible to use the final solution or anything similar in the sensor? This would eliminate the need for superfluous communication while also allowing for parallelism. In other words, it might not increase the model's accuracy, but it might improve its computing performance.

The project should end with a description of an alternative solution to the problem of emptying detection, as well as any customization parameters. As a result, the end result should not be a production-ready implementation, but rather a road map for what to do. As a result, computational performance is not a priority right now.

Instead, the performance of the emptying detection is prioritized. This also means that live data isn't required; instead, historical data can be added during the solution's start-up phase. Finally, because the solution does not need to be production-ready, it can be implemented as a one-time-only solution.

CHAPTER 3

SOURCE CODE

```
# Importing the Required Libraries
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import warnings
warnings.filterwarnings('ignore')

# Importing the Dataset
from google.colab import files
uploaded=files.upload()

import io
df = pd.read_csv(io.BytesIO(uploaded['Smart_Bin.csv']))

# Information overview of the dataset
df.shape
df.info()

# Replacing the Missing Values
def removing_missing_values(column_name):
    df[column_name] = df[column_name].fillna(df[column_name].median())
i=1
for i in range(1,8):
    removing_missing_values(df.columns[i])
    i+=1
df
df.info()

# Label Encoding the Target variable
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()

df['Class']= label_encoder.fit_transform(df['Class'])
df['Class'].unique()

df['Container Type']= label_encoder.fit_transform(df['Container Type'])
df['Container Type'].unique()

df['Recyclable fraction']= label_encoder.fit_transform(df['Recyclable fraction'])
```

```
df['Recyclable fraction'].unique()

df.head()
```

 # Detecting the Outliers

```
import seaborn as sns

plt.figure(figsize=(20,15))
plt.subplot(2,4,1)
sns.boxplot(df['Class'])
plt.subplot(2,4,2)
sns.boxplot(df['FL_B'])
plt.subplot(2,4,3)
sns.boxplot(df['FL_A'])
plt.subplot(2,4,4)
sns.boxplot(df['VS'])
plt.subplot(2,4,5)
sns.boxplot(df['FL_B_3'])
plt.subplot(2,4,6)
sns.boxplot(df['FL_A_3'])
plt.subplot(2,4,7)
sns.boxplot(df['FL_B_12'])
plt.subplot(2,4,8)
sns.boxplot(df['FL_A_12'])
plt.show()
```

 # Removing the Outliers

```
df['FL_B'].values[df['FL_B'].values > 100] = df['FL_B'].mean()
df['FL_B_3'].values[df['FL_B_3'].values > 100] = df['FL_B_3'].mean()
df['FL_B_12'].values[df['FL_B_12'].values > 100] = df['FL_B_12'].mean()
df['FL_A'].values[df['FL_A'].values > 100] = df['FL_A'].mean()
df['FL_A_3'].values[df['FL_A_3'].values > 100] = df['FL_A_3'].mean()
df['FL_A_12'].values[df['FL_A_12'].values > 100] = df['FL_A_12'].mean()

df.describe()
```

 # Calculation of Change in Fill Level

```
df['FL_C']=df['FL_A']-df['FL_B']
df['FL_C_3']=df['FL_A_3']-df['FL_B_3']
df['FL_C_12']=df['FL_A_12']-df['FL_B_12']

df.info()
```

 # Corrleation Matrix

```
import seaborn as sns
from matplotlib import rcParams
from matplotlib.cm import rainbow

corrmat=df.corr()
top_corr_features=corrmat.index
plt.figure(figsize=(15,15))
sns.heatmap(df[top_corr_features].corr(),annot=True,cmap='RdYlGn')
plt.show()
```

 # Standard Scaling

```
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
standardScaler=StandardScaler()

columns_to_scale=['FL_B','FL_A', 'FL_C', 'VS']
df[columns_to_scale]=standardScaler.fit_transform(df[columns_to_scale])
df.head()
```

 # Splitting the Dataset into Training and Testing Data

```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2)
train_X = train[['FL_A', 'FL_C', 'VS']]
train_y = train.Class

test_X = test[['FL_A', 'FL_C', 'VS']]
test_y = test.Class
```

 # K Nearest Neighbors

```
from sklearn.neighbors import KNeighborsClassifier

knn_scores=[]
for k in (range(1,21)):
  knn_classifier=KNeighborsClassifier(n_neighbors=k)
  knn_classifier.fit(train_X,train_y)
  knn_scores.append(knn_classifier.score(test_X,test_y))
```

 # Plotting the Graph of Model Scores for Different K Values

```
plt.figure(figsize=(30,30))
plt.plot([k for k in range(1,21)],knn_scores,color="blue")
for i in range(1,21):
  plt.text(i, knn_scores[i-1],(i,round(knn_scores[i-1],4)))
plt.xticks([i for i in range(1,21)])
plt.xlabel("Number of Neighbors (K)",color="Red",weight="bold",fontsize="18")
plt.ylabel("Scores",color="Red",weight="bold",fontsize="18")
plt.title("K Neighbors Classifier scores for different K values",color="Red",weight="bold",fo
ntsize="20")
#plt.figure(figsize=(30, 20))
plt.show()
plt.rcParams["font.weight"]="bold"
plt.rcParams["axes.labelweight"]="bold"
```

 # Plotting the Confusion Matrix

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import plot_confusion_matrix

KN_model = KNeighborsClassifier(8)
KN_model.fit(train_X,train_y)
pred=KN_model.predict(test_X)
KN_model.score(test_X,test_y)
cm=confusion_matrix(test_y,pred)
print(cm)
print("\n")
plot_confusion_matrix(KN_model, test_X, test_y, cmap=plt.cm.Blues);
```

 # Classification Report

```
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
MCC_KNN = round(a/b,3 )
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
```

```
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC Score : ', MCC_KNN)
```

 # Support Vector Machine

```
from sklearn.svm import LinearSVC

svm_model=LinearSVC()
svm_model.fit(train_X,train_y)
pred=svm_model.predict(test_X)
svm_model.score(test_X,test_y)
cm=confusion_matrix(test_y,pred)
print(cm)
print("\n")
plot_confusion_matrix(svm_model, test_X, test_y, cmap=plt.cm.Blues);
```

 # Classfication Report

```
# print(classification_report(test_y,pred))
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
MCC_SVM = round(a/b, 3)
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC score : ', MCC_SVM)
```

 # Logistic Regression

```
from sklearn.linear_model import LogisticRegression
classifier= LogisticRegression(random_state=0)
```

```
classifier.fit(train_X,train_y)
y_pred= classifier.predict(test_X)
cm= confusion_matrix(test_y,y_pred)
print(cm)
plot_confusion_matrix(classifier, test_X, test_y, cmap=plt.cm.Blues);
```

 # Classfication Report

```
#print(classification_report(test_y,pred))
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
MCC_SVM = round(a/b, 3)
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC score : ', MCC_SVM)
```

 # Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

clf_model=DecisionTreeClassifier()
clf_model.fit(train_X,train_y)
pred=clf_model.predict(test_X)
clf_model.score(test_X,test_y)
cm=confusion_matrix(test_y,pred)
print(cm)
print("\n")
plot_confusion_matrix(clf_model, test_X, test_y, cmap=plt.cm.Blues);
#print(classification_report(test_y,pred))
```

 # Classfication Report

```
#print(classification_report(test_y,pred))
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
MCC_DT = round(a/b, 3)
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC score :', MCC_DT)
```

 # MLP Neural Network Classifier

```
from sklearn.neural_network import MLPClassifier
classifier = MLPClassifier(hidden_layer_sizes=(150,100,50), max_iter=300,activation = 'relu'
,solver='adam',random_state=1)
classifier.fit(train_X, train_y)
y_pred = classifier.predict(test_X)
#clf_model.score(test_X,test_y)
cm=confusion_matrix(test_y,pred)
print(cm)
print("\n")
plot_confusion_matrix(classifier, test_X, test_y, cmap=plt.cm.Blues);
```

 # Classfication Report

```
#print(classification_report(test_y,pred))
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
```

```
MCC_NN = round(a/b, 3)
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC score :', MCC_NN)
```

 # Random Forest

```
from sklearn.ensemble import RandomForestClassifier

RF_scores = []
for k in range(1,21):
    ensemble_classifier=RandomForestClassifier(n_estimators=k)
    ensemble_classifier.fit(train_X,train_y)
    RF_scores.append(ensemble_classifier.score(test_X,test_y))


plt.figure(figsize=(30,30))
plt.plot([k for k in range(1,21)],RF_scores,color='blue')
for i in range(1,21):
    plt.text(i, RF_scores[i-1], (i, RF_scores[i-1]))
plt.xticks([i for i in range(1, 21)])
plt.xlabel('Number of Trees',color='Red',weight='bold',fontsize='12')
plt.ylabel('Scores',color='Red',weight='bold',fontsize='12')
plt.title('Random FOrest  scores ',color='Red',weight='bold',fontsize='12')
plt.show()
plt.rcParams["font.weight"]= "bold"

RF_model = RandomForestClassifier()
RF_model.fit(train_X,train_y)
pred = RF_model.predict(test_X)
RF_model.score(test_X,test_y)
#print(confusion_matrix(test_y,pred))
cm=confusion_matrix(test_y,pred)
print(cm)
print("\n")
plot_confusion_matrix(RF_model, test_X, test_y, cmap=plt.cm.Blues);
#print(classification_report(test_y,pred))
```

 # Classfication Report

```
print(classification_report(test_y,pred))
TP = cm[1][1]
FP = cm[0][1]
TN = cm[0][0]
FN = cm[1][0]
b = ((TP+FP)*(TP+FN)*(TN+FP)*(TN+FN))
b = b**0.5
a = (TP*TN-FP*FN)
MCC_RF = round(a/b, 3)
Precision_Score = TP / (FP + TP)
Recall_Score = TP / (FN + TP)
Accuracy_Score = (TP + TN)/ (TP + FN + TN + FP)
F1_Score = 2* Precision_Score * Recall_Score/ (Precision_Score + Recall_Score)
print("Accuracy  : ",Accuracy_Score*100)
print("Precision : ",Precision_Score*100)
print("Recall    : ",Recall_Score*100)
print("F1 score  : ",F1_Score*100)
print('MCC score :', MCC_RF)
```

CHAPTER 4

SNAPSHOTS

4.1 K-Nearest Neighbors

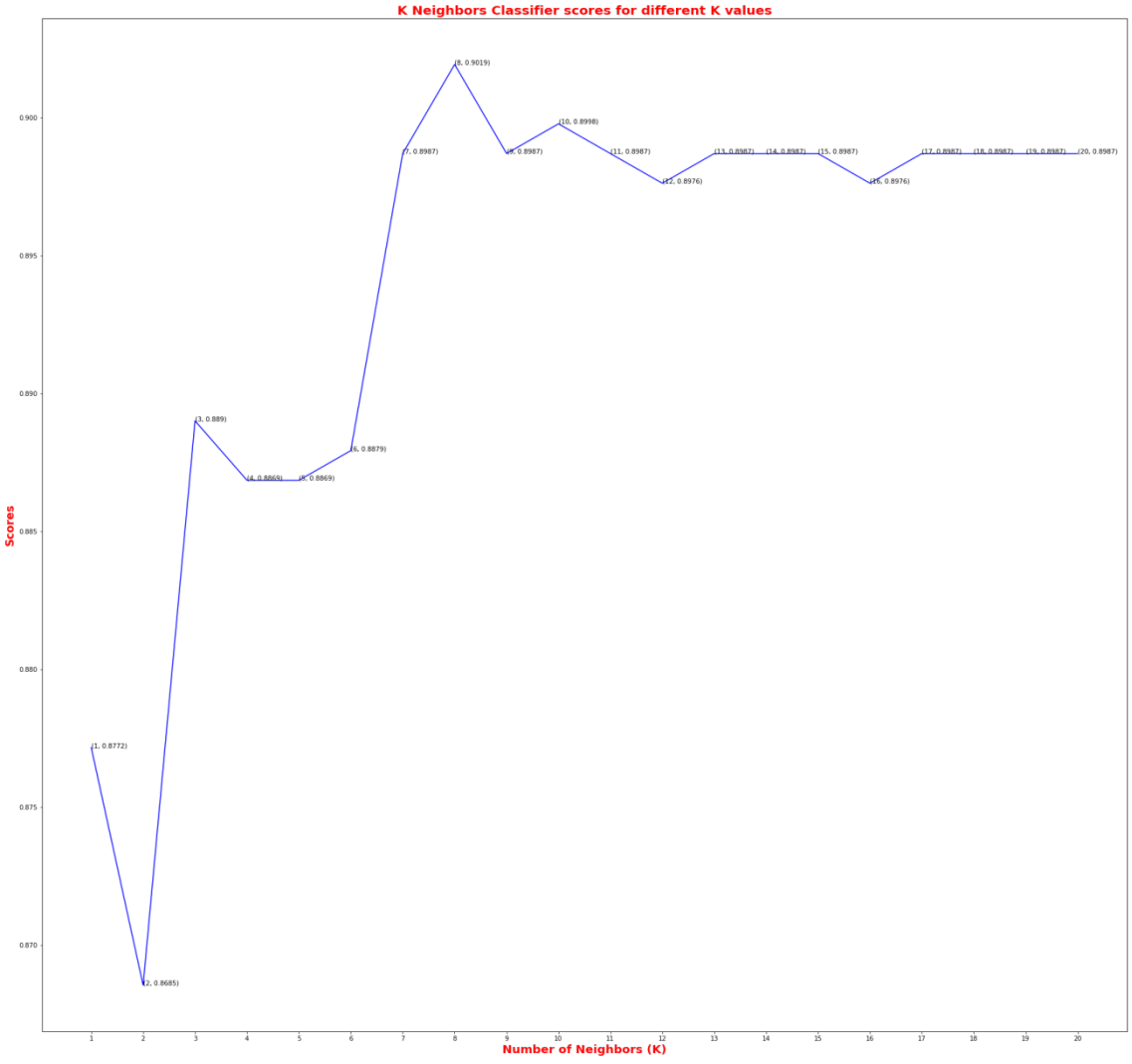

Fig 4.1.1 Classifier Scores

```
[[407  41]
 [ 50 430]]
```

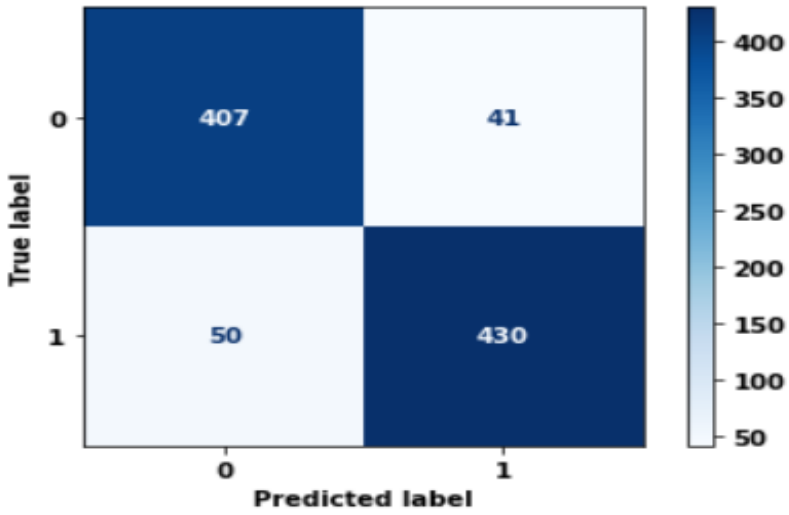

Fig 4.1.2 Confusion Matrix

```
Accuracy  :  90.19396551724138
Precision :  91.29511677282377
Recall    :  89.58333333333334
F1 score  :  90.4311251314406
MCC Score :  0.804
```

Fig 4.1.3 Classification Report

4.2 Support Vector Machine

```
[[397  51]
 [ 57 423]]
```

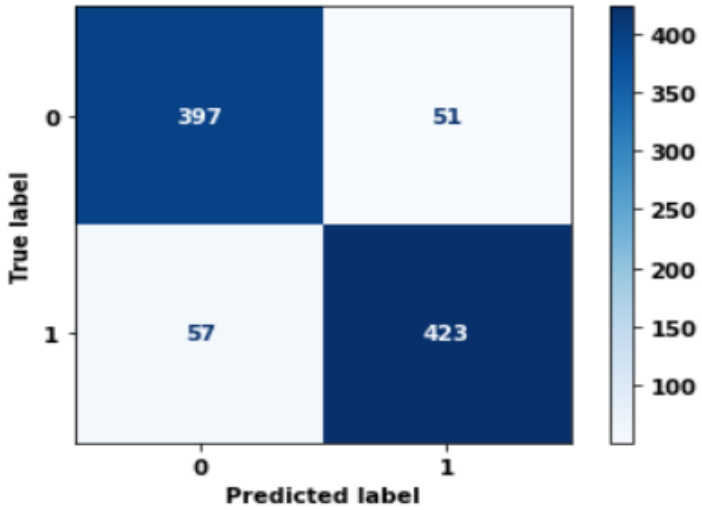

Fig 4.2.1 Confused Matrix

```
Accuracy  :  88.36206896551724
Precision :  89.24050632911393
Recall    :  88.125
F1 score  :  88.67924528301886
MCC score :  0.767
```

Fig 4.2.2 Classification Report

4.3 Logistic Regression

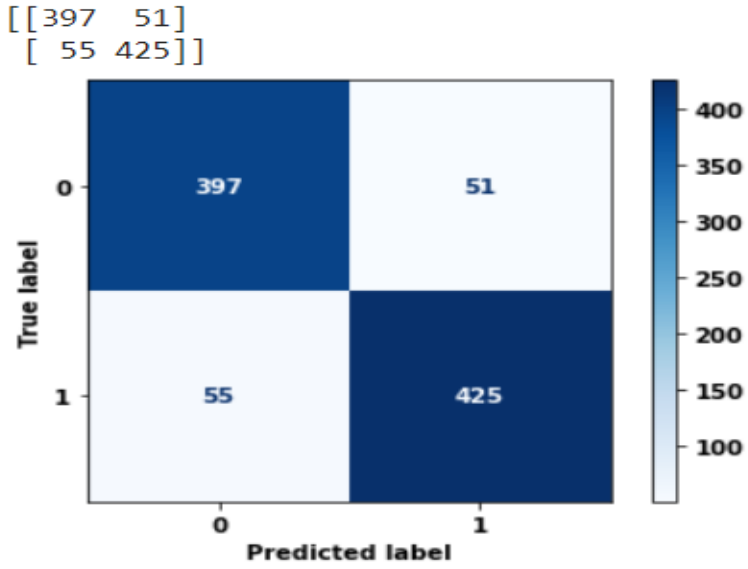

Fig 4.3.1 Confusion Matrix

```
Accuracy  :  88.57758620689656
Precision :  89.28571428571429
Recall    :  88.54166666666666
F1 score  :  88.91213389121339
MCC score :  0.771
```

Fig 4.3.2 Classification Report

4.4 Decision Tree

```
[[387  61]
 [ 63 417]]
```

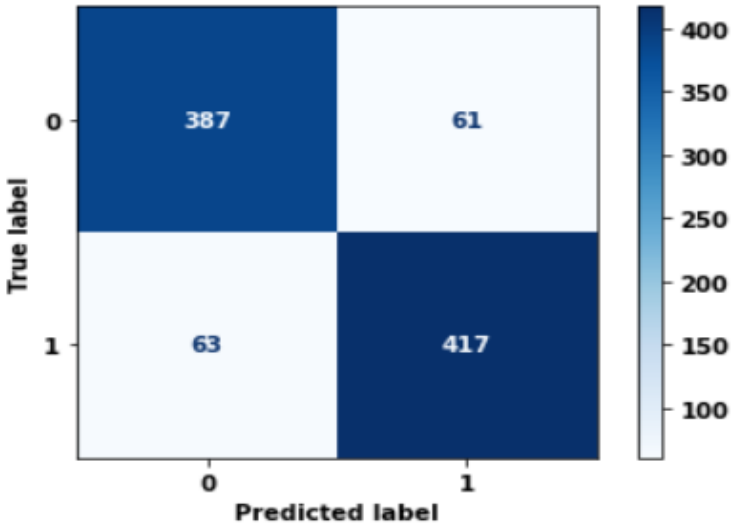

Fig 4.4.1 Confusion Matrix

```
Accuracy  :  86.63793103448276
Precision :  87.23849372384937
Recall    :  86.875
F1 score  :  87.05636743215031
MCC score : 0.732
```

Fig 4.4.2 Classification Report

4.5 MLP Neural Network

```
[[387  61]
 [ 63 417]]
```

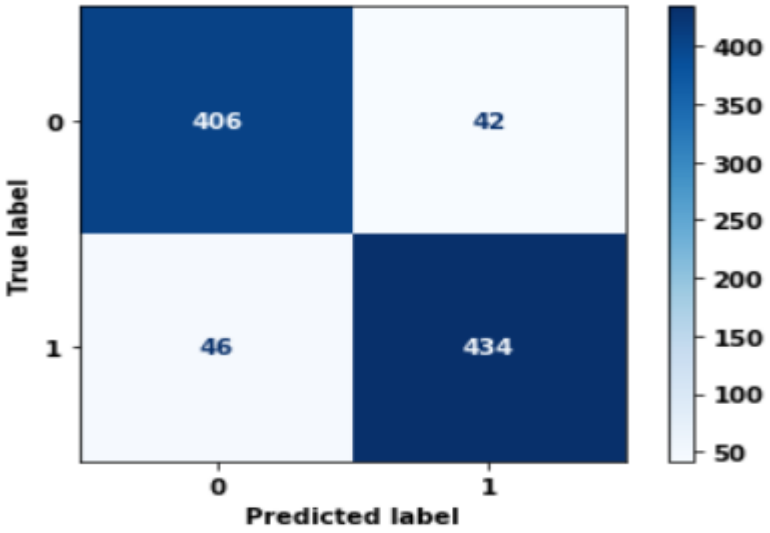

Fig 4.5.1 Confusion Matrix

```
Accuracy  :  86.63793103448276
Precision :  87.23849372384937
Recall    :  86.875
F1 score  :  87.05636743215031
MCC score : 0.732
```

Fig 4.5.2 Classification Report

4.6 Random Forest

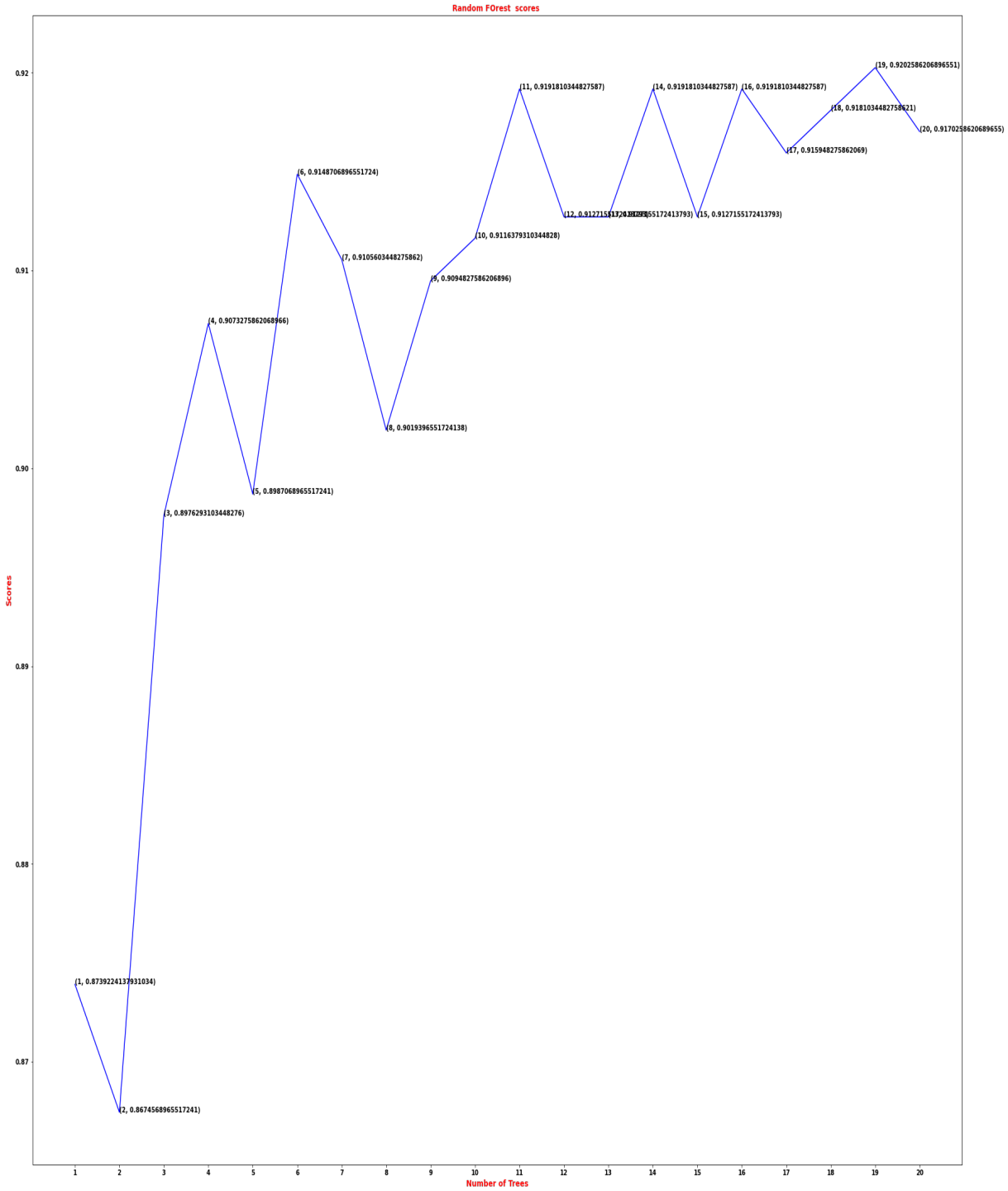

Fig 4.6.1 Classifier Scores

```
[[420  28]
 [ 43 437]]
```

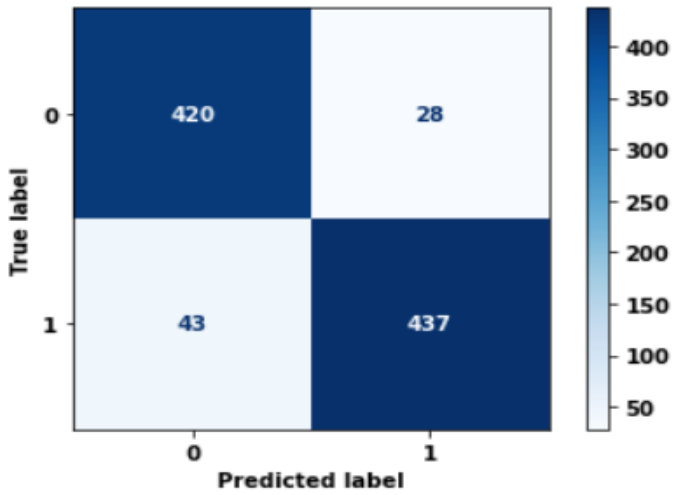

Fig 4.6.2 Confusion Matrix

```
Accuracy  :  92.34913793103449
Precision :  93.97849462365592
Recall    :  91.04166666666667
F1 score  :  92.4867724867725
MCC score : 0.847
```

Fig 4.6.3 Classification Report

CHAPTER 5

CONCLUSION AND FUTURE PLANS

5.1 Conclusion

The difficulty of determining when a recycling container has been emptied using measurements supplied by a smart sensor positioned on top of the recycle container Starting with a legacy solution, the results were steadily improved across numerous cycles of model modifications until the final classifier, an optimized random forest classifier, was reached. The filling level before a suspected emptying is used as a feature, as well as three distinct filling level changes (instant change, change between three hours, and change between twelve hours).

Two criteria were established early on in the project to assess the feasibility of a solution for moving on with the exploration to a production level implementation. To begin, the overall categorization accuracy should be at least 90%. Second, the recall, or the percentage of correctly categorized emptying among all real emptying, should be at least 95%. The final solution had an accuracy of 99.1 percent and a recall of 98.2 percent on the testing data, therefore both requirements were met.

As a result, it is concluded that the final solution is appropriate for a real-world scenario provided measurements preceding and after the point of interest by twelve hours may be allowed.

One of the objectives of the solution is to be able to use it in a real-world setting. However, twelve hours of data from subsequent vibration samples is not always accessible at the time of analysis in the real system. This is a challenge that must be solved before the solution can be implemented.

Algorithm	Accuracy	Recall	F1 Score	MCC Score	Precision
kNN	90.193	89.583	90.431	0.804	91.295
SVM	88.362	88.125	88.679	0.767	89.240
LR	88.577	88.541	88.912	0.771	89.285
DT	86.637	86.875	87.056	0.732	87.238
MLP NN	86.637	86.875	87.056	0.732	87.238
RF	92.349	91.041	92.486	0.847	93.978

Table 5.1 Performance Comparison of Considered Solutions

5.2 Future Works

Future work is divided into two sections: what has to be done in order to utilize the results, and what areas may benefit from additional investigation.

5.2.1 Using Results in Production

An appropriate implementation of the solution must be established in order to apply the results in the Smart Recycling system. The thesis' implementation loads both the training and testing data from a static file, performs the training and testing, and then leaves the programme.

The training and saving of the model should be done via a competent implementation. Then, during the initialization phase, just load the model and get new samples for classification. Once the containers have uploaded their data, the samples will be delivered.

The characteristics must also be extracted in order to do this. The devices simply send raw data, but the final model takes an average of the filling level, which means the data must first be transformed to filling level and then averaged over three to twelve hours.

Before executing the final classifier, check to see if a sample contains twelve hours of data, and if it does, run the final classifier. If that is not the case, then utilize the legacy

characteristics or as much data as is available at the time of classification; nevertheless, the result must be labeled as a tentative prediction with a specific label. Then, once the unit has uploaded even more data, rerun the preliminary classifications for all prior classifications.

Another approach is to include extra capability in the sensor itself. When sensing a vibration, take repeated measurements or set a timer to submit data precisely twelve hours after the vibration was detected. Of course, because the machines are battery-powered, this would reduce their energy efficiency, but it is a possibility to investigate.

5.2.2 Areas for Continued Research

Even if the results show exceptional performance, additional areas of research may still be necessary. Is it possible to use the final solution or anything similar in the sensor? This would eliminate the need for superfluous communication while also allowing for parallelism. In other words, it might not increase the model's accuracy, but it might improve its computing performance. An extra hard- ware might be advantageous for the future iteration of the sensor, as mentioned during the brainstorm, and should be studied further. A gyroscope, magnetometer, or other accelerometer that records independent values for each of the axes is an example of such hardware.

Another area for future research is how to handle model changes. For example, unlike random forests, neural networks include ways for adding only a few additional samples to the training set without having to restart the training from the beginning. Furthermore, if the model deviates from its initial objective, repeatedly recreating the model might represent a risk to the system.

Finally, continuous effort is required to achieve the goal of this project, namely the prediction of emptying intervals. What can be done to increase the predictability of emptying time predictions? It's not out of the realm of possibility that a whole different technique is required. Other data sources may be of assistance. It's also possible that having access to entirely new sorts of data, such as weather forecasts, local event information, or holiday seasons, might help enhance historical predictions.

CHAPTER 6

REFERENCES

1. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and Robust Automated Machine Learning," in Advances in Neural Information Processing Systems 28, 2015, pp. 2962–2970.

2. V. Catania and D. Ventura, "An approch for monitoring and smart planning of urban solid waste management using smart-m3 platform," in Proceedings of the 15th Conference of Open Innovations Association FRUCT, FRUCT'15, (Helsinki, Finland, Finland), pp. 4:24–4:31, FRUCT Oy, 2014.

3. C. Davenport and A. M. Kessler, "Proposed rule for big trucks aims at cutting fuel emissions," The New York Times, Jun 2015. [Online; accessed 28-February-2018].

4. J. W. Lu, N. B. Chang, L. Liao, and M. Y. Liao, "Smart and green urban solid waste collection systems: Advances, challenges, and perspectives," IEEE Systems Journal, vol. 11, pp. 2804–2817, Dec 2017.

5. A. Kanawaday and A. Sane, "Machine learning for predictive maintenance of industrial machines using IoT sensor data," in 2017 8th IEEE International Conference on Software Engineering and Service Science (ICSESS), pp. 87–90, Nov 2017.

6. H. Almer, "Machine learning and statistical analysis in fuel consumption prediction for heavy vehicles," Master's thesis, KTH, School of Computer Science and Communication (CSC), 2015.

7. D.Fumo,"Types of machine learning algorithms you should know." https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861, Jun 2017. [Online; accessed 23-April-2018].

CHAPTER 7

APPENDIX

Considered Solutions to the Problem of Detecting Emptying

During the project's early stages, a brainstorming session was held to provide the groundwork for future research. Another motive was to investigate a larger range of options rather than sticking with the original choice. Figure A.1 shows the mind-map that was created as a result of this brainstorm. Sensor data processing, sensor, verification, and system level data processing are the four sub-areas of the mind-map. The sensor component comprises of the sensor's current and projected measurements. Some of the proposed metrics are tweaks to existing ones, such as dividing the accelerometer value into one measurement per axis.

In the sensor data processing section, methods are provided for processing that should take place in the sensor rather than on the server. Filtering out the accelerometer so that just the z-axis is used, and employing a pattern-based detection of emptying are two examples. In order to identify an emptying, the system level data processing incorporates methods that employ past references to uncover trends over time and across sensors. These solutions must be installed on the server side of the system since storage is required. Finally, solutions to check the performance and validity of any solution are offered in the verification section.

Container Types in Data Set

The data set used in this project contains samples from all container types currently present in the system for the fractions studied. All container types are displayed with their respective relative size in the data set and the number of positive and negative samples. It can be used to describe why some container types may perform worse than others and if the final solution is applicable to a real world scenario.

Decision Bounds for Different Classifiers

One part of the evaluation was to visualize the decision bounds for all algorithms as a comparison. To completely cover the comparison three decision plots are needed.

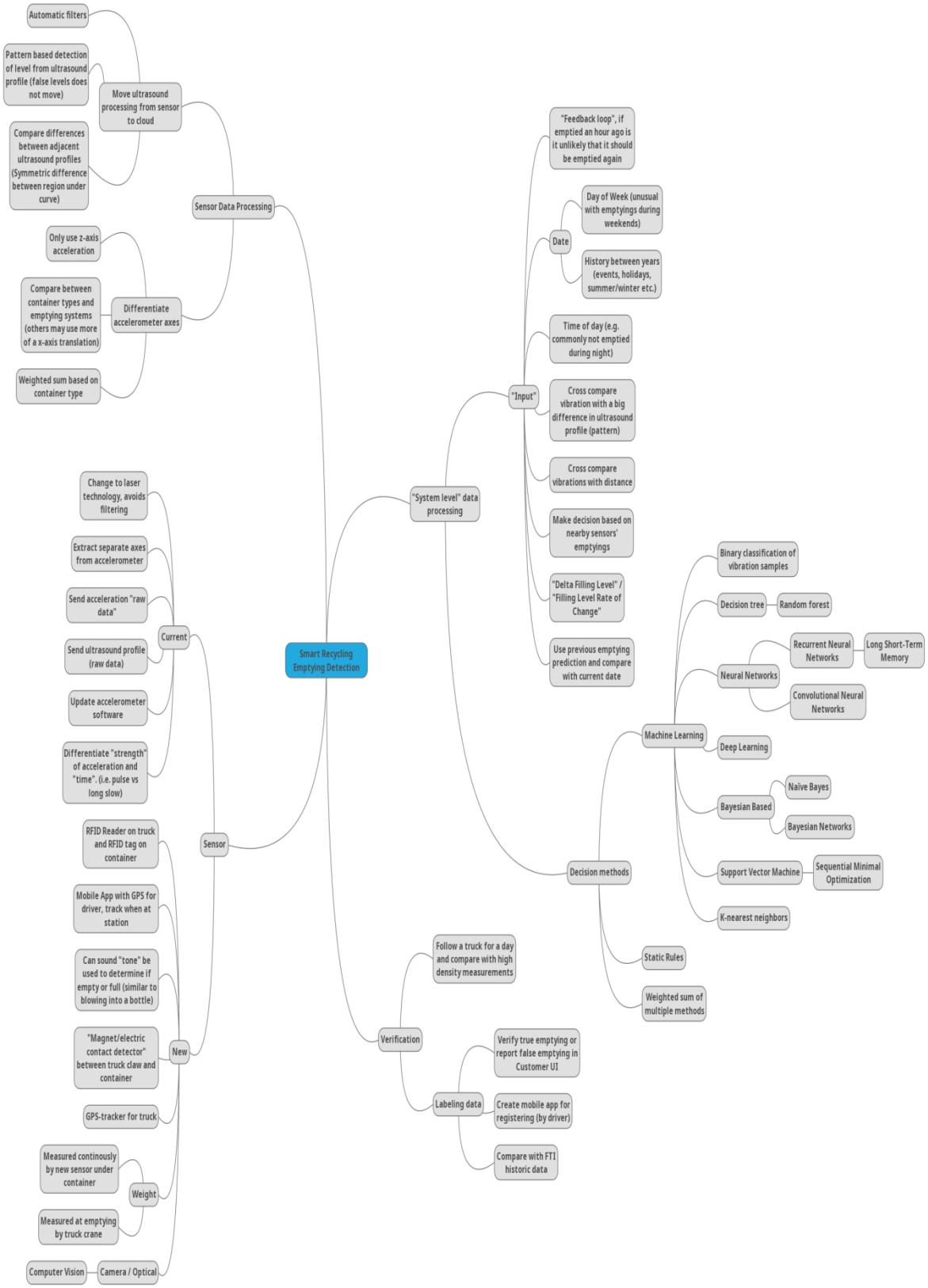

Fig. 7.1 The mind-map which is the result of an early brainstorm session about possible solutions to the problem, both utilizing existing and new techniques

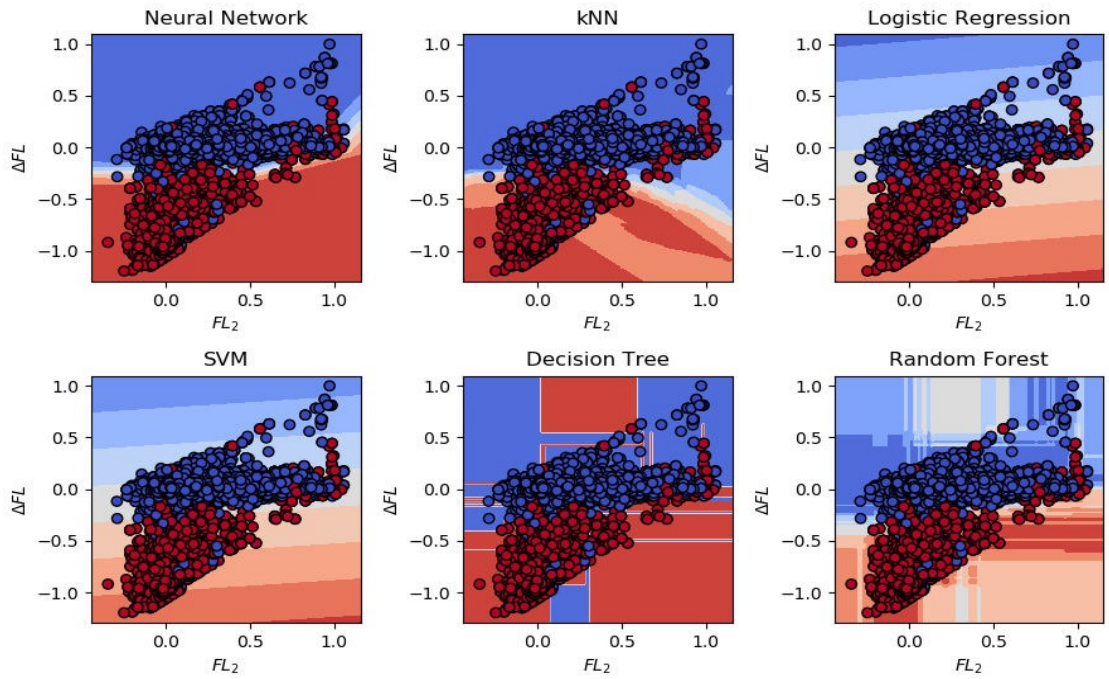

Fig. 7.2 Decision boundary for the studied classifiers when looking only at filling level change (ΔFL) and vibration strength (V_{str})

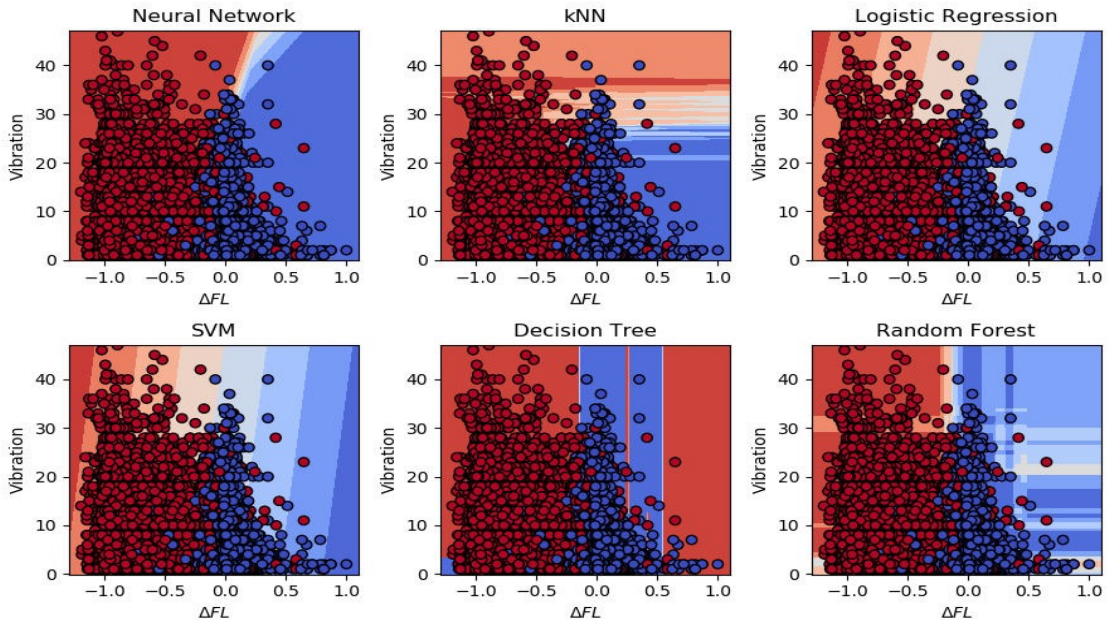

Fig. 7.3 Decision boundary for the studied classifiers when looking only at filling level after (FL2) and vibration strength (V_{str})