**WATER PUMP CONDITION PREDICTION USING ML**

* **BAALA VIDHYA SREE K C**

**Problem definition**

Water pump failure is one of the huge problems faced in the society today. May it be a small town or a big city, this particular issue is faced by every resident irrespective of their residential area. Here, Machine Leaning comes into play where one can able to identify if a pump is in working condition or not based on the provided details.

The aim of this problem is to find a suitable classification model for the pump sensor dataset for which the classification is to be made whether the pump’s working condition is normal, or is at a recovering stage or broken, if a particular set of attributes is given.

From the results obtained, the concerned worker or official can come to know the working condition of the pumps. If a failure is identified, then they can take measures to replace the pump or repair the pump according to the severity of failure.

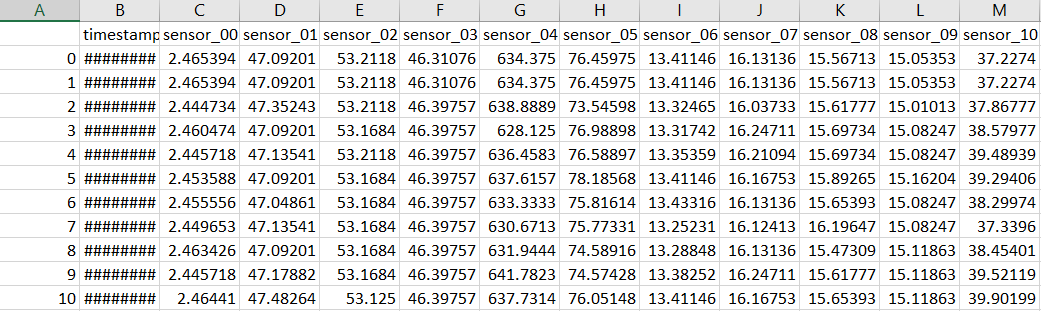
**Dataset**

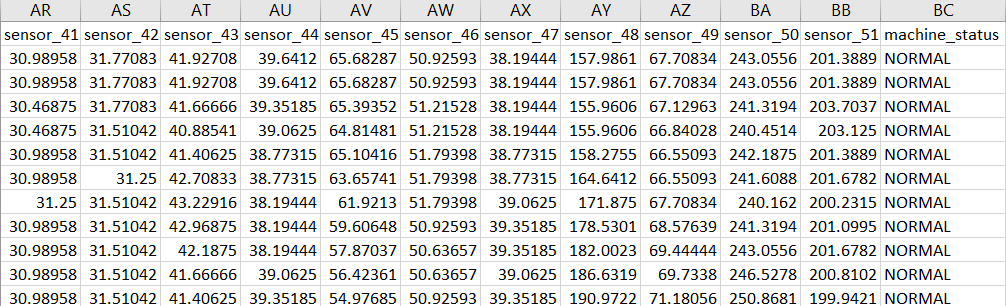
The ‘sensor’ dataset consists of the following attributes:

* Timestamp data.
* Sensor data (52 series): All values are raw values.
* Machine status: This is target label that is to be predicted when the failure will happen.

**Dataset Source:** <https://www.kaggle.com/nphantawee/pump-sensor-data>

**A sample data instances is shown below:**





**DATA PRE - PROCESSING**

First, the data is checked for null values. If any null value is present, the we fill the null values with either mean, median or mode values based on whether the variables are categorical or numeric variables. Then we try to remove the attributes of least importance. After this, we try to remove the outliers present in the data and we standardize it.

In this data, the attribute ‘sensor\_15’ is totally empty and so we can remove that column from the data. We can also remove the column ‘Unnamed: 0’ since it is merely an index column. Also, there is a considerable amount of null values present. So, to fill these values, we take median value of the corresponding column attributes to fill the null values present. Then we convert the column ‘timestamp’ into date and time and then drop the former column so that the latter columns can be used for analysis. After this, we try to find the outliers using z – score and abs values (absolute values). These values are used to see how far a data point is from the mean. So, here we consider data point values to be greater than 3 (an assumption, which also suits well for this multi - class dataset).Finally, we standardize the data before splitting into dependent and independent variables.

**Importing required dataset**

data = pd.read\_csv("/content/sensor.csv")

**Data description and information**

data.shape

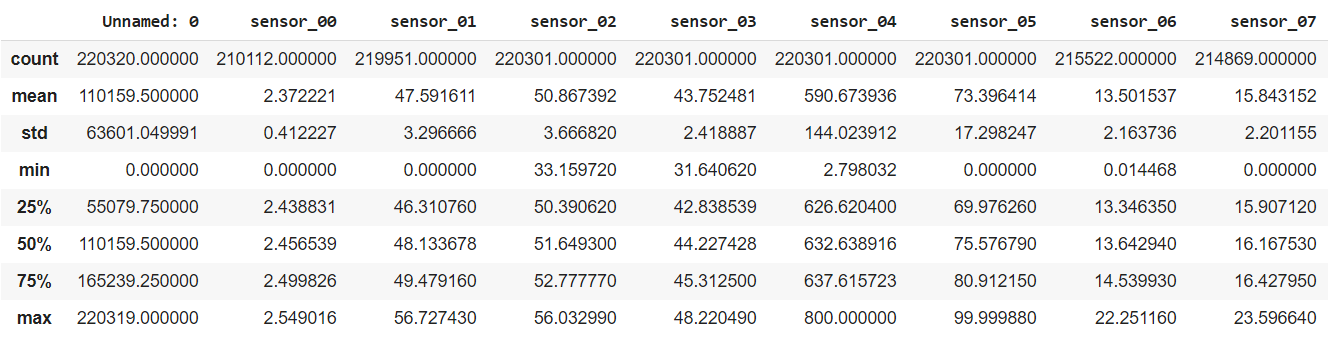
(220320, 55)

We have around 2 lacks of data instances for finding the condition of the water pump. Also we have 55 attributes, in which 52 attributes are sensor values.

data.describe()

data.info()

print('Number of duplicate rows: ',data.duplicated().sum())



All the data instances are of Float type, so we don’t need to change the sensors column. Attribute ‘timestamp’ should be converted as attribute ‘date’ and ‘time’ because it is being considered as object type.

**Removing the least important attributes**

data = data.drop('sensor\_15', 1)

data = data.drop('Unnamed: 0', 1)

data.shape

data['date'] = data['timestamp'].apply(lambda x: x.split(' ')[0])

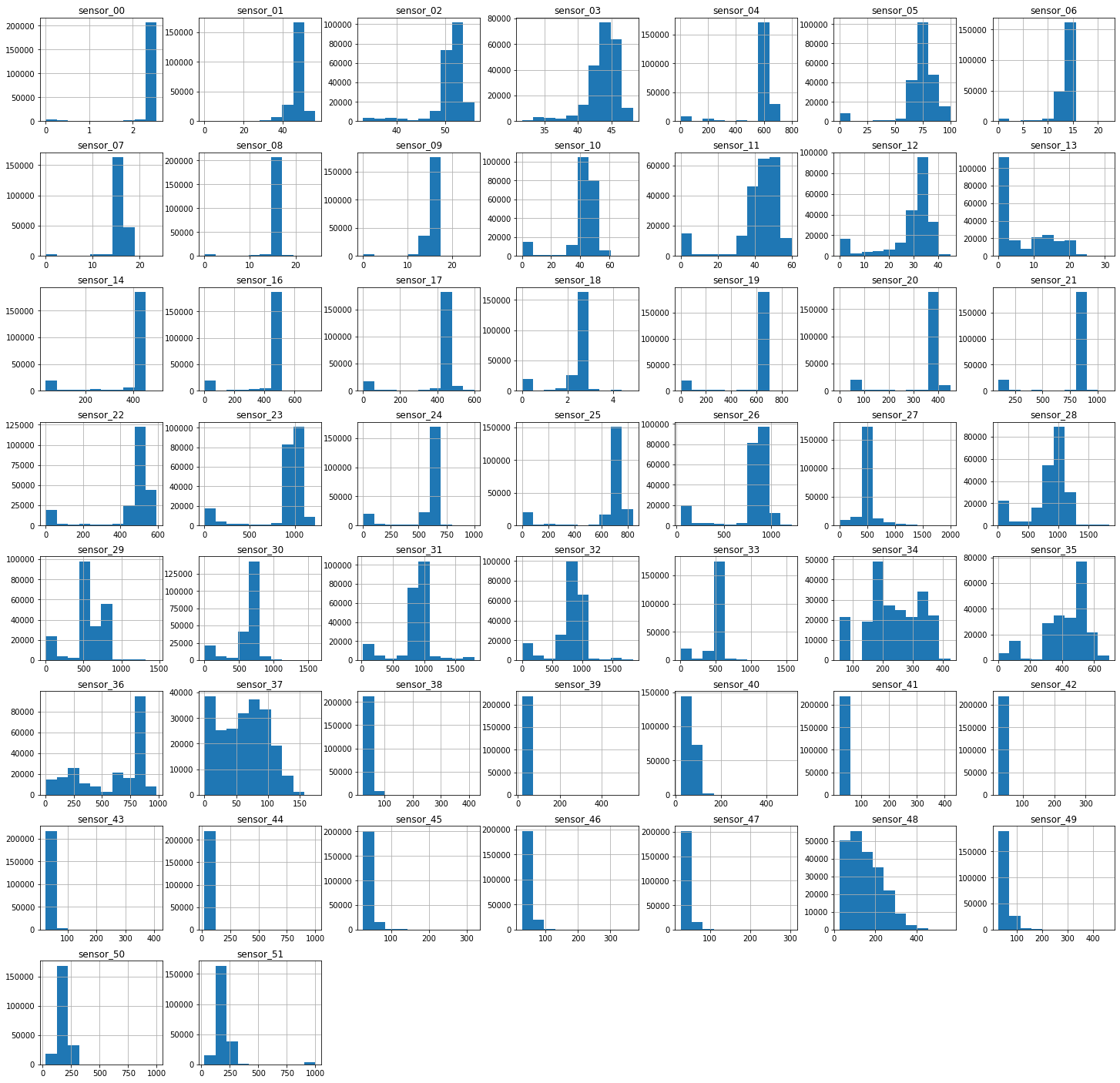
data['time'] = data['timestamp'].apply(lambda x: x.split(' ')[1])

data = data.drop(['timestamp'], 1)

data\_imputed = data.fillna(data.median())

data\_imputed.hist(figsize=(25,25))

plt.show()



*Figure 1 Visualization of data using Histogram*

From the above graph, we can infer that there are outliers in the values. So we remove the outliers from the data by normalize the all the values (i.e., by finding z scores) and remove those instances which have outliers.

**Removing outliers using z-scores**

z\_scores = zscore(data\_imputed.iloc[:,:51])

abs\_z\_scores = np.abs(z\_scores)

filtered\_entries = (abs\_z\_scores < 3).all(axis=1)

data\_outlir = data\_imputed[filtered\_entries]

**Feature scaling**

scaler = MMS()

data\_std = scaler.fit\_transform(data\_outlir.iloc[:,:51])

data\_std = pd.DataFrame(data\_std)

data\_std.shape

**Multi - class description**

NORMAL 165016

RECOVERING 183

BROKEN 2

Name: machine\_status, dtype: int64

The description shows our data is highly biased for the ‘NORMAL’ class having 165016 data instances. We can’t make the dataset unbiased by resampling, as ‘RECOVERING’ and ‘BROKEN’ classes have data instance very less comparing to ‘NORMAL’ class. So we proceed for finding ‘NORMAL’ condition (i.e., whether the water pump is normal).

**Inference**

From the above results, the following are inferred:

* From the dataset description, we can come to know about the distribution of the dataset, the size and statistical measures like mean, standard deviation, quartiles and range.
* From the dataset information, we come to know about the data type of each column, their non – null count and the memory consumed by this dataset.
* From the results of function info (), we come to know that there are some null values present in some attributes. As said earlier, the attributes ‘Unnamed: 0’ and ‘sensor\_15’ are removed. Then after splitting ‘timestamp’ into date and time, we remove that attribute also. Then the null values are filled with median of the respective columns.
* The outliers are removed using z – scores and abs values after which the dataset gets reduced in size.
* Then we check for the Class distribution, for which we obtain a ‘multi – class’ distribution containing classes like ‘NORMAL’, ‘RECOVERING’ and ‘BROKEN’.

**EXPLORATORY DATA ANALYSIS (EDA)**

EDA involves exploring the datasets by means of visualizations. With the help of EDA, we can able to identify the methods that can be used for further analysis and classification.

In this dataset, First, we check for the multi - collinearity of the dataset. This can be visualized with the help of a line graph, for which the variances of the dataset are used as values to plot the graph. Then, we filter the dataset with the correlation by setting up a threshold value. Then we encode the categorical variable ‘machine\_status’. Then we visualize the correlation using heatmap and the categorical variable using scatter plot. Then, we split the final data using ‘train\_test\_split’ function, which can be used for further classifications.

**Checking for Mutli – collinearity**

data\_va = data\_std.var(axis= 0)

data\_vas = data\_va.sort\_values(ascending=False)

y = data\_vas.values

x = range(len(y))

plt.figure(figsize = (5,5))

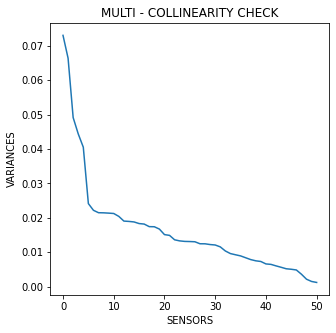
plt.plot(x, y)

plt.title("MULTI - COLLINEARITY CHECK")

plt.xlabel("SENSORS")

plt.ylabel("VARIANCES")

plt.show()



*Figure 2 Plot for Multi - collinear check*

From the above plot, we can infer that all the sensor attributes have different variances and are non-collinear.

**Removing attributes which are highly correlated**

def correlation(dataset, threshold):

    col\_corr = set()

    corr\_matrix = dataset.corr()

    for i in range(len(corr\_matrix.columns)):

        for j in range(i):

            if ((abs(corr\_matrix.iloc[i, j]) >= threshold) or (abs(corr\_matrix.iloc[i, j]) <= (-1\*threshold))) and (corr\_matrix.columns[j] not in col\_corr):

                colname = corr\_matrix.columns[i]

                col\_corr.add(colname)

                if colname in dataset.columns:

                    del dataset[colname]

    return dataset

**Encoding categorical variables**

le = LE()

Y = le.fit\_transform(data\_outlir['machine\_status'])

**Heatmap and correlation**

X = correlation(data\_std, 0.7)

cor = X.corr()

figure, axes = plt.subplots(figsize=(25, 25))

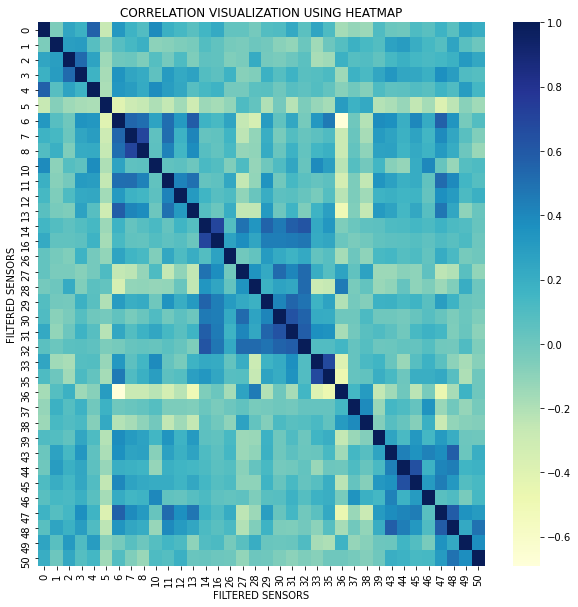
sns.heatmap(cor, ax = axes, cmap = "YlGnBu", linecolor = 'black')

plt.title("CORRELATION VISUALIZATION USING HEATMAP")

plt.xlabel("FILTERED SENSORS")

plt.ylabel("FILTERED SENSORS")

plt.show()



*Figure 3 Correlation heatmap*

Attributes with high correlation have been removed and represented in Heatmap.

**Contribution of the filtered attributes**

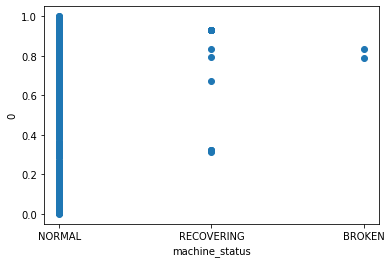
for i in data\_std.columns:

    plt.scatter( data\_outlir['machine\_status'] , data\_std[i] )

    plt.xlabel('machine\_status')

    plt.ylabel(i)

    plt.show()



*Figure 4 Sample plots for visualizing the contributions of the attributes*

**Inference**

From the above visualizations, we can see that after filtering of attributes, the correlation between those filtered attributes are fairly strong and so we can proceed the classification with these filtered attributes.

Also, we can able to see the distribution of the ‘multi – class’ variable resulted from the capturing of each sensor. From these scatter plots, we can see that the class ‘NORMAL’ is widely present than the other two classes.

**MACHINE LEARNING MODELS**

Here comes the Machine Learning concept into action. We can use classification models for prediction since the dataset contains ‘multi – class’ classification data. So, we make several classification models like the following:

1. Logistic Regression
2. Naive Bayes Classification
3. Decision Tree Classification
4. Random Forest Classification
5. Support Vector Machine (SVM) Classification
6. K Nearest Neighbours (KNN) Classification

**Logistic Regression**

lr = LR(C = 0.2)

clf1 = lr.fit(X\_train, y\_train)

y\_pred1 = clf1.predict(X\_test)

print("\n1. Logistic Regression Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred1,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred1))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred1))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred1, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred1))

1. Logistic Regression Classifier

CLASSIFICATION REPORT

precision recall f1-score support

1 1.00 1.00 1.00 54470

2 1.00 0.94 0.97 47

accuracy 1.00 54517

macro avg 1.00 0.97 0.98 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[54470 0]

[ 3 44]]

ACCURACY SCORE : 0.999944971293358

PRECISION SCORE : 0.9999449268444918

ROC-AUC Score : 0.9680851063829787

**Inference**

We have performed logistic regression classification for our data,

* The logistic regression implemented has given the Confusion matrix with 54470 true negatives, 0 false positives, 3 false negatives and 44 true positives.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 1.00 for class 2 (i.e., ‘RECOVERING’ class) shows that 100% of Recovering class is classified correctly.
* 99.9% of Precision, quantifies that the number of positive class predictions that actually belong to the positive class.
* It has the accuracy of 99.99% for the Logistic Regression classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 97% correct.

**Naive Bayes Classification**

nb = NB()

clf2 = nb.fit(X\_train, y\_train)

y\_pred2 = clf2.predict(X\_test)

print("\n2. Naive Bayes Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred2,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred2))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred2))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred2, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred2))

2. Naive Bayes Classifier

CLASSIFICATION REPORT

precision recall f1-score support

0 0.00 1.00 0.00 0

1 1.00 1.00 1.00 54470

2 0.46 0.96 0.63 47

accuracy 1.00 54517

macro avg 0.49 0.99 0.54 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[ 0 0 0]

[ 0 54418 52]

[ 1 1 45]]

ACCURACY SCORE : 0.9990094832804446

PRECISION SCORE : 0.4879663832794401

ROC-AUC Score : 0.9676179353069619

**Inference**

We have performed Naive Bayes Classification for our data,

* Most of the predictions were predicted correctly.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 0.46 for class 2 (i.e., ‘RECOVERING’ class) shows that 46% of Recovering class is classified correctly. Precision of 0.00 for class 0 (i.e., ’BROKEN’ class) shows that 0% of Broken class is classified correctly.
* 48% of Precision quantifies the number of positive class predictions that actually belong to the positive class.
* Recall quantifies the number of positive class predictions made out of all positive examples in the dataset (100% belongs to the normal and recovering machines, 96% belongs to the broken machines).
* It has the accuracy of 99.9% for the Naive Bayes classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 97% correct.

**Decision Tree Classification**

dt = tree.DecisionTreeClassifier(criterion='entropy')

clf3 = dt.fit(X\_train, y\_train)

y\_pred3 = clf3.predict(X\_test)

print("\n3. Decision Tree Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred3,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred3))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred3))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred3, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred3))

3. Decision Tree Classifier

CLASSIFICATION REPORT

precision recall f1-score support

0 0.00 1.00 0.00 0

1 1.00 1.00 1.00 54470

2 0.98 0.96 0.97 47

accuracy 1.00 54517

macro avg 0.66 0.99 0.66 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[ 0 0 0]

[ 2 54467 1]

[ 1 1 45]]

ACCURACY SCORE : 0.9999082854889301

PRECISION SCORE : 0.9999629147632989

ROC-AUC Score : 0.9680769035463598

**Inference**

We have performed Decision Tree Classification for our data,

* Most of the predictions were predicted correctly.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 0.98 for class 2 (i.e., ‘RECOVERING’ class) shows that 98% of Recovering class is classified correctly. Precision of 0.00 for class 0 (i.e., ’BROKEN’ class) shows that 0% of Broken class is classified correctly.
* 99.9% of Precision quantifies the number of positive class predictions that actually belong to the positive class.
* Recall quantifies the number of positive class predictions made out of all positive examples in the dataset (100% belongs to the normal and recovering machines, 96% belongs to the broken machines).
* It has the accuracy of 99.9% for the Decision tree classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 97% correct.

**Random Forest Classification**

rf = RFC(max\_depth=5, random\_state=0)

clf4 = rf.fit(X\_train, y\_train)

y\_pred4 = clf4.predict(X\_test)

print("\n4. Random Forest Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred4,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred4))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred4))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred4, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred4))

4. Random Forest Classifier

CLASSIFICATION REPORT

precision recall f1-score support

1 1.00 1.00 1.00 54470

2 1.00 0.96 0.98 47

accuracy 1.00 54517

macro avg 1.00 0.98 0.99 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[54470 0]

[ 2 45]]

ACCURACY SCORE : 0.999963314195572

PRECISION SCORE : 0.9999816419444852

ROC-AUC Score : 0.9787234042553192

**Inference**

We have performed Random forest Classification for our data,

* The logistic regression implemented has given the Confusion matrix with 54470 true negatives, 0 false positives, 2 false negatives and 45 true positives.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 1.00 for class 2 (i.e., ‘RECOVERING’ class) shows that 100% of Recovering class is classified correctly.
* 99.99% of Precision quantifies the number of positive class predictions that actually belong to the positive class.
* Recall quantifies the number of positive class predictions made out of all positive examples in the dataset (100% belongs to the normal and recovering machines, 96% belongs to the broken machines).
* It has the accuracy of 99.9% for the Random Forest classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 98% correct.

**Support vector machine (SVM) classification**

svm = SVC(kernel = 'rbf')

clf5 = svm.fit(X\_train, y\_train)

y\_pred5 = clf5.predict(X\_test)

print("\n5. SVM Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred5,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred5))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred5))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred5, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred5))

5. SVM Classifier

CLASSIFICATION REPORT

precision recall f1-score support

1 1.00 1.00 1.00 54470

2 1.00 0.96 0.98 47

accuracy 1.00 54517

macro avg 1.00 0.98 0.99 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[54470 0]

[ 2 45]]

ACCURACY SCORE : 0.999963314195572

PRECISION SCORE : 0.9999816419444852

ROC-AUC Score : 0.9787234042553192

**Inference**

We have performed SVM Classification for our data,

* The logistic regression implemented has given the Confusion matrix with 54470 true negatives, 0 false positives, 2 false negatives and 45 true positives.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 1.00 for class 2 (i.e., ‘RECOVERING’ class) shows that 100% of Recovering class is classified correctly.
* 99.99% of Precision quantifies the number of positive class predictions that actually belong to the positive class.
* Recall quantifies the number of positive class predictions made out of all positive examples in the dataset (100% belongs to the normal and recovering machines, 96% belongs to the broken machines).
* It has the accuracy of 99.9% for the Naive Bayes classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 97% correct.

**K-Nearest Neighbour (KNN) Classification**

knn = KNN(n\_neighbors = 5, metric = 'minkowski', p = 2)

clf = knn.fit(X\_train, y\_train)

y\_pred = clf.predict(X\_test)

print("\n6. KNN Classifier\n")

print('\n CLASSIFICATION REPORT\n', c\_r(y\_test, y\_pred,zero\_division=1))

print('\n CONFUSION MATRIX\n', c\_m(y\_test, y\_pred))

print('\n ACCURACY SCORE  : ', a\_s(y\_test, y\_pred))

print('\n PRECISION SCORE : ', p\_s(y\_test, y\_pred, average='macro', zero\_division = 1))

print('\n ROC-AUC Score   :', RAS(y\_test, y\_pred))

6. KNN Classifier

CLASSIFICATION REPORT

precision recall f1-score support

1 1.00 1.00 1.00 54470

2 1.00 0.98 0.99 47

accuracy 1.00 54517

macro avg 1.00 0.99 0.99 54517

weighted avg 1.00 1.00 1.00 54517

CONFUSION MATRIX

[[54470 0]

[ 1 46]]

ACCURACY SCORE : 0.999981657097786

PRECISION SCORE : 0.9999908208037305

ROC-AUC Score : 0.9893617021276595

**Inference**

We have performed KNN Classification for our data,

* The logistic regression implemented has given the Confusion matrix with 54470 true negatives, 0 false positives, 1 false negatives and 46 true positives.
* Precision of 1.00 for class 1 (i.e., ’NORMAL’ class) shows that 100% of Normal class is classified correctly. Precision of 1.00 for class 2 (i.e., ‘RECOVERING’ class) shows that 100% of Recovering class is classified correctly.
* 99.99% of Precision quantifies the number of positive class predictions that actually belong to the positive class.
* Recall quantifies the number of positive class predictions made out of all positive examples in the dataset (100% belongs to the normal and recovering machines, 98% belongs to the broken machines).
* It has the accuracy of 99.9% for the KNN classification.
* AUC (Area under ROC curve) score provides an aggregate measure of performance across all possible classification thresholds, thus it shows that this model’s prediction are 99% correct.

**ROC-AUC Curve for all models**

pred\_prob1 = clf1.predict\_proba(X\_test)

pred\_prob2 = clf2.predict\_proba(X\_test)

pred\_prob3 = clf3.predict\_proba(X\_test)

pred\_prob4 = clf4.predict\_proba(X\_test)

pred\_prob5 = clf5.predict\_proba(X\_test)

pred\_prob6 = clf.predict\_proba(X\_test)

from sklearn.metrics import roc\_curve

fpr1, tpr1, thresh1 = roc\_curve(y\_test, pred\_prob1[:,1], pos\_label=1)

fpr2, tpr2, thresh2 = roc\_curve(y\_test, pred\_prob2[:,1], pos\_label=1)

fpr3, tpr3, thresh3 = roc\_curve(y\_test, pred\_prob3[:,1], pos\_label=1)

fpr4, tpr4, thresh4 = roc\_curve(y\_test, pred\_prob4[:,1], pos\_label=1)

fpr5, tpr5, thresh5 = roc\_curve(y\_test, pred\_prob5[:,1], pos\_label=1)

fpr6, tpr6, thresh6 = roc\_curve(y\_test, pred\_prob6[:,1], pos\_label=1)

random\_probs = [0 for i in range(len(y\_test))]

p\_fpr, p\_tpr, \_ = roc\_curve(y\_test, random\_probs, pos\_label=1)

import matplotlib.pyplot as plt

plt.style.use('seaborn')

plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='LR')

plt.plot(fpr2, tpr2, linestyle='--',color='green', label='NB')

plt.plot(fpr3, tpr3, linestyle='--',color='red', label='DT')

plt.plot(fpr4, tpr4, linestyle='--',color='purple', label='RF')

plt.plot(fpr5, tpr5, linestyle='--',color='yellow', label='SVM')

plt.plot(fpr6, tpr6, linestyle='--',color='grey', label='KNN')

plt.plot(p\_fpr, p\_tpr, linestyle='--', color='blue')

plt.title('ROC curve')

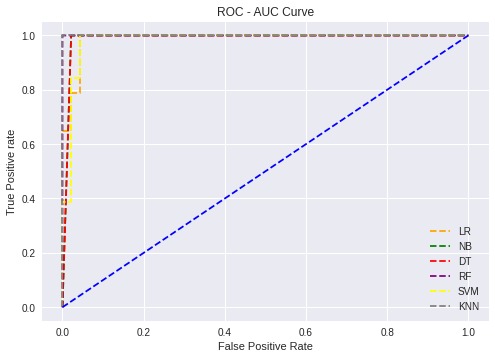
plt.xlabel('False Positive Rate')

plt.ylabel('True Positive rate')

plt.legend(loc='best')

plt.savefig('ROC',dpi=300)

plt.show();

****

*Figure 5 ROC-AUC curves for all ML models*

**Inference**

From the above ROC-AUC curve, we can infer that all the models perform well. The AUC score of 0.97, 0.97, 0.97, 0.98, 0.98, 0.99 for Logistic regression, Naive Bayes classification, Decision Tree classification, Random Forest classification, Support vector machine, k-Nearest Neighbour classification indicates that in can distinguish the classes well with slight errors.

**MODEL OPTIMIZATION**

Model Optimization is done to obtain the ‘best’ design/model relative to a set of constraints. Here, model optimization is done with the help of k - Fold Cross Validation method. In this method, the untouched part of the dataset undergoes testing through the models in terms of folds. Here we take k value as 5. At each iteration, one of the folds gets tested through the models. This optimization can be visualized using boxplot. From the boxplot, we can tell which model has most performance and that model can be suggested to the officials for future references.

**Model optimization**

from sklearn import model\_selection

models = []

models.append(('LR', lr))

models.append(('NB', nb))

models.append(('DT', dt))

models.append(('RF', rf))

models.append(('SVM', svm))

models.append(('KNN', knn))

**Evaluate each model in turn**

results = []

names = []

scoring = 'accuracy'

for name, model in models:

  kfold = model\_selection.KFold(n\_splits=5)

  cv\_results = model\_selection.cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring=scoring)

  results.append(cv\_results)

  names.append(name)

  msg = "%s   : %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

  print(msg)

**Visualization**

fig = plt.figure()

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

plt.xlabel("CLASSIFICATION MODEL")

plt.ylabel("CROSS VALIDATION SCORES")

ax.set\_xticklabels(names)

plt.show()

LR : 0.999864 (0.000049)

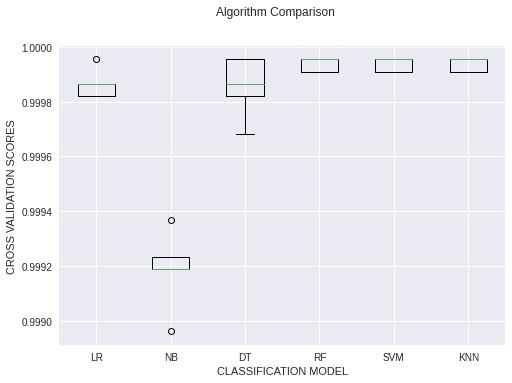
NB : 0.999187 (0.000131)

DT : 0.999837 (0.000097)

RF **:** 0.999937 (0.000022)

SVM **:** 0.999928 (0.000036)

KNN **:** 0.999937 (0.000022)



*Figure 6 Cross validation score*

**Inference**

* From the Cross Validation Scores and the Box plot, we can infer that the suitable model for this problem is actually 2 models namely ‘RANDOM FOREST’ and ‘KNN’.
* These 2 models are selected on the fact that they have high cross validation scores and minimum standard deviations which help us to predict the correct result with less error.

**Conclusion**

From the above results we can infer how various models are suitable for predicting the condition of the water pump. Among the implemented model KNN classifier and Random Forest classifier show good result with less error. All the implemented models have given 99% of accuracy; the slight errors are visualized only through ROC-AUC curve graph. From the results obtained, the concerned worker or official can check the working condition of the pumps. If a failure is identified, then they can take measures to replace the pump or repair the pump according to the severity of failure.