**REAL TIME PREDICTIVE MAINTENANCE**

**A**

Project Report

Submitted in fulfilment of the Requirements for the award of the degree of **BACHELOR OF TECHNOLOGY**

### IN

**COMPUTER SCIENCE ENGINEERING**

**With**

**Specialization in BIG DATA**

**BY**

| **Name** | **Roll No.** | **SAPID** |
| --- | --- | --- |
| RISHI PANDEY | R172218097 | 500067313 |
| RAJESH YADAV | R172218155 | 500071212 |
| PARTH | R172218085 | 500067724 |
| SAHIL KUMAR SINGH | R172218102 | 500068621 |

### 

### Under the guidance of

**MR. GOURAV BATHLA**

### Department of Cybernetics, SCHOOL OF

COMPUTER SCIENCE



**UNIVERSITY OF PETROLEUM & ENERGY STUDIES**

### Bidholi Campus, Energy Acres, Dehradun – 248007.

### August 2021

**ACKNOWLEDGEMENT**

We wish to express our deep gratitude to our guide **Mr. Gourav Bathla**, for all advice, encouragement, and constant support he has given me throughout the project work. This work would not have been possible without his support and valuable suggestions.

We would like to thank all our friends for their help and constructive criticism during the project work.

| **Name** | **Roll No.** | **SAPID** |
| --- | --- | --- |
| RISHI PANDEY | R172218097 | 500067313 |
| RAJESH YADAV | R172218155 | 500071212 |
| PARTH | R172218085 | 500067724 |
| SAHIL KUMAR SINGH | R172218102 | 500068621 |

**TABLE 1:- TABLE OF CONTENTS**

| **Serial Number** | **Contents** | **Page Number** |
| --- | --- | --- |
| 1. | Introduction | 4 |
| 2. | Purpose | 5 |
| 3. | Scope | 5 |
| 4. | Objective | 5 |
| 5. | Real Time Predictive Maintenance | 5 |
| 6. | Regression | 6 |
| 7. | Logistic Regression | 6 |
| 8. | System requirements | 7 |
| 9. | Source code | 7 |
| 10. | Output screens | 13 |

# INTRODUCTION

We live in a world of machines. Everything that we use whether it's electricity, cars, processed foods etc , everything is coming from large scale industries. And these industries function with the help of machines. These machines work day and night to give us the comfort of our life. But as all good things have an end, so do these machines. Due to wear and tear, these machines start to break down which is a serious concern for a company as if any of the machines fails then the whole assembly has to be stopped which results in loss for the company.

So, to solve this problem , we are going to develop a Real Time Predictive Maintenance

System. The system will be fed the real time iot data generated by these machines in order to make predictions about the future risks of a machine failure. In our system, the key components will be Azure Event hub, Azure databricks and PowerBI. So, firstly the Iot data is streamed from Azure Event hub to Azure databricks. In Azure databricks, we will apply a pre-trained Ml model to the streaming data, the predictions for this data will then be streamed back to the Azure event hub . Then , the event hub will stream this data to powerBI where we will perform the actual visualisation of the predicted outcome.

Thus, how we will be developing a Real-Time Predictive Maintenance system.

# PURPOSE

The purpose of the project is to identify the important variables that affect the health of the machine parts of our dataset. Then using PySpark’s MLlib library we create a Ml model using a logistic regression algorithm. The accuracy of the system will be tested through the ROC curve.

We aim to provide more accurate results for machine failure on a real time basis .

# SCOPE

Predictive maintenance is a technique that uses data analysis tools and techniques to detect anomalies in our operation and possible defects in equipment and processes so you can fix them before or after they result in failure. The predictive maintenance can save billions of dollars for heavy industries like Power corporations, car manufacturer,

Railways etc.

# OBJECTIVE

* Stream the dataset from Azure Event Hub to Azure Databricks.
* Train the system using a logistic regression algorithm.
* Predicting which machine is going to fail.

**Real Time Predictive Maintenance**

A primary motivation of many investigators in the field has been to determine how to cut their losses on repair of parts and replacing heavy machineries. Replacing industry level equipment costs a fortune to the companies which can be used for research and developing new technology.

Anticipating future actions is a key component of intelligence, specifically when it applies to real-time systems, such as industrial robots or autonomous machines. Predicting the health of machine parts in strategic settings is an important problem in many domains

**REGRESSION**

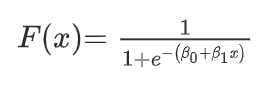
Regression analysis consists of a set of machine learning methods that allow us to predict a continuous outcome variable (y) based on the value of one or multiple predictor variables (x).

Briefly, the goal of a regression model is to build a mathematical equation that defines y as a function of the x variables. Next, this equation can be used to predict the outcome (y) on the basis of new values of the predictor variables (x).

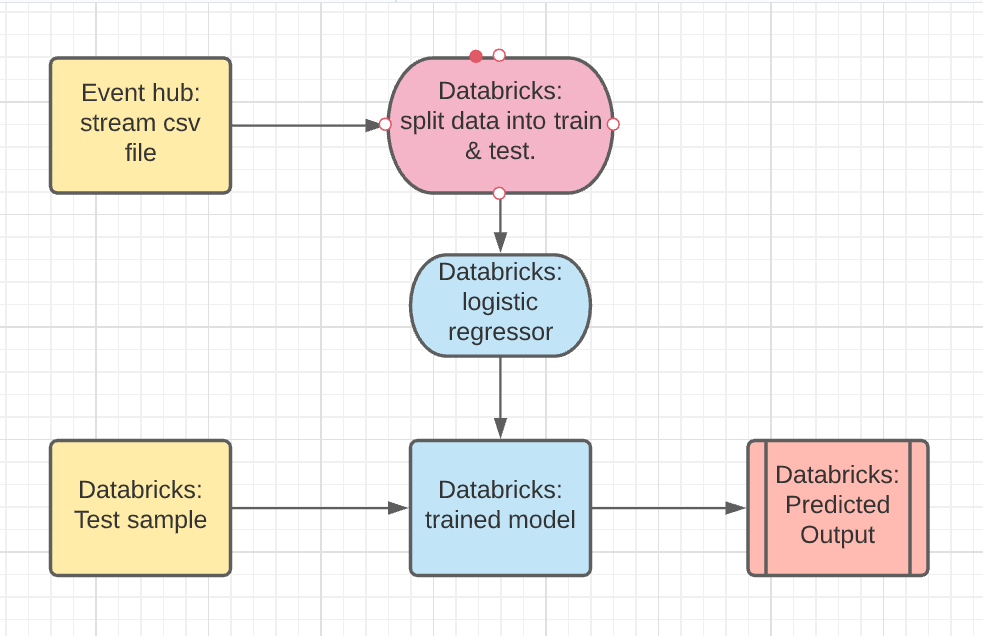
**Logistic Regression**

LR is a transformation of a linear regression using the sigmoid function. The vertical axis stands for the probability for a given classification and the horizontal axis is the value of x. It assumes that the distribution of y|x is Bernoulli distribution.

The formula of LR is as follows:



The logistic function applies a sigmoid function to restrict the y value from a large scale to within the range 0–1.



**Fig 1**

# 

# SYSTEM REQUIREMENTS

**TABLE 2:-** SOFTWARE REQUIREMENTS

| Operating System : | Windows or higher |
| --- | --- |
| Programming Language : | PySpark |
| Platform used : | Azure event hub, Azure databricks |

**TABLE 3:-** HARDWARE REQUIREMENTS

| Processor : | Pentium IV or higher |
| --- | --- |
| Disk Drive : | Hard Disk Drive |
| RAM : | 2GB or higher |

# Source Code

1. **Read data:-**

import org.apache.spark.sql.types.\_

import org.apache.spark.sql.functions.\_

import org.apache.spark.eventhubs.\_

import com.microsoft.azure.eventhubs.\_

val namespaceName = "azuretrial"

val eventHubName = "final"

val sasKeyName = "root"

val sasKey = "ryNdDPw2AaX8BvMYjyAWZMUQV8z+bh1W4SEkKaLuRQg="

val connStr = new com.microsoft.azure.eventhubs.ConnectionStringBuilder()

.setNamespaceName(namespaceName)

.setEventHubName(eventHubName)

.setSasKeyName(sasKeyName)

.setSasKey(sasKey)

val customEventhubParameters =

EventHubsConf(connStr.toString())

.setMaxEventsPerTrigger(50)

val incomingStream = spark.readStream.format("eventhubs").options(customEventhubParameters.toMap).load()

val messages =

incomingStream

.withColumn("Offset", $"offset".cast(LongType))

.withColumn("Time (readable)", $"enqueuedTime".cast(TimestampType))

.withColumn("Timestamp", $"enqueuedTime".cast(LongType))

.withColumn("Data", $"body".cast(StringType))

.select("Data")

//messages.writeStream.outputMode("append").format("console").option("truncate", false).start().awaitTermination()

messages.writeStream.format("delta").outputMode("append").option("checkpointLocation","/data/events/\_checkpoints/data\_file\_1").table("i2")

1. **Write data:-**

import scala.collection.JavaConverters.\_

import com.microsoft.azure.eventhubs.\_

import java.util.concurrent.\_

import scala.collection.immutable.\_

import scala.concurrent.Future

import scala.concurrent.ExecutionContext.Implicits.global

val namespaceName = "azuretrial"

val eventHubName = "final"

val sasKeyName = "root"

val sasKey = "ryNdDPw2AaX8BvMYjyAWZMUQV8z+bh1W4SEkKaLuRQg="

val connStr = new ConnectionStringBuilder()

.setNamespaceName(namespaceName)

.setEventHubName(eventHubName)

.setSasKeyName(sasKeyName)

.setSasKey(sasKey)

val pool = Executors.newScheduledThreadPool(1)

val eventHubClient = EventHubClient.createFromConnectionString(connStr.toString(), pool)

def sleep(time: Long): Unit = Thread.sleep(time)

def sendEvent(message: String, delay: Long) = {

sleep(delay)

val messageData = EventData.create(message.getBytes("UTF-8"))

eventHubClient.get().send(messageData)

System.out.println("Sent event: " + message + "\n")

}

val df = spark.read.format("csv").option("header","true").option("sep", ",").option("inferSchema", "true").load("dbfs:/FileStore/tables/ai4i2020\_\_\_ai4i2020\_\_test\_.csv")

df.collect().foreach { row =>

sendEvent(row.mkString(" "), 5000)

}

1. **Codes for Ml model:-**

* dataset = spark.read.load("dbfs:/FileStore/tables/ai4i2020\_\_\_ai4i2020\_\_6\_.csv", format = "csv", header= "true", inferSchema = "true")
* from pyspark.sql.functions import col,count,when

dataset.select([count(when(col(c).isNull(), c)).alias(c) for c in dataset.columns]).show()

* trainDF, testDF = dataset.randomSplit([0.8, 0.2], seed=42)

print(trainDF.cache().count()) # Cache because accessing training data multiple times

print(testDF.count())

* from pyspark.ml.feature import StringIndexer, OneHotEncoder

categoricalCols = ["Product\_ID", "Type"]

# The following two lines are estimators. They return functions that we will later apply to transform the dataset.

stringIndexer = StringIndexer(inputCols=categoricalCols, outputCols=[x + "Index" for x in categoricalCols]).setHandleInvalid("keep")

encoder = OneHotEncoder(inputCols=stringIndexer.getOutputCols(), outputCols=[x + "OHE" for x in categoricalCols])

# The label column ("Machine failure") is also a string value - it has two possible values, "PASS" and "FAIL".

# Convert it to a numeric value using StringIndexer.

labelToIndex = StringIndexer(inputCol="Machine\_failure", outputCol="label")

* stringIndexerModel = stringIndexer.fit(trainDF)

display(stringIndexerModel.transform(trainDF))

* from pyspark.ml.feature import VectorAssembler

# This includes both the numeric columns and the one-hot encoded binary vector columns in our dataset.

numericCols = ["UDI", "Air\_temperature", "Process\_temperature", "Rotational\_speed", "Torque","TWF", "HDF","PWF","OSF","RNF"]

assemblerInputs = [c + "OHE" for c in categoricalCols] + numericCols

vecAssembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")

* from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol="features", labelCol="label", regParam=1.0)

* from pyspark.ml import Pipeline

# Define the pipeline based on the stages created in previous steps.

pipeline = Pipeline(stages=[stringIndexer, encoder, labelToIndex , vecAssembler, lr])

# Define the pipeline model.

pipelineModel = pipeline.fit(trainDF)

# Apply the pipeline model to the test dataset.

predDF = pipelineModel.transform(testDF)

* display(predDF.select("features", "label", "prediction", "probability"))
* display(pipelineModel.stages[-1], predDF.drop("prediction", "rawPrediction", "probability"), "ROC")
* from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator

bcEvaluator = BinaryClassificationEvaluator(metricName="areaUnderROC")

print(f"Area under ROC curve: {bcEvaluator.evaluate(predDF)}")

mcEvaluator = MulticlassClassificationEvaluator(metricName="accuracy")

print(f"Accuracy: {mcEvaluator.evaluate(predDF)}")

* from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

paramGrid = (ParamGridBuilder()

.addGrid(lr.regParam, [0.01, 0.5, 2.0])

.addGrid(lr.elasticNetParam, [0.0, 0.5, 1.0])

.build())

* # Create a 3-fold CrossValidator
* cv = CrossValidator(estimator=pipeline, estimatorParamMaps=paramGrid, evaluator=bcEvaluator, numFolds=3, parallelism = 4)

# Run cross validations. This step takes a few minutes and returns the best model found from the cross validation.

cvModel = cv.fit(trainDF)

* # Use the model identified by the cross-validation to make predictions on the test dataset

cvPredDF = cvModel.transform(testDF)

# Evaluate the model's performance based on area under the ROC curve and accuracy

print(f"Area under ROC curve: {bcEvaluator.evaluate(cvPredDF)}")

print(f"Accuracy: {mcEvaluator.evaluate(cvPredDF)}")

* cvPredDF.createOrReplaceTempView("finalPredictions")
* %sql

SELECT Type, prediction, count(\*) AS count

FROM finalPredictions

GROUP BY Type, prediction

ORDER BY Type

* %sql

SELECT Product\_ID, prediction, count(\*) AS count

FROM finalPredictions

WHERE prediction = 1

GROUP BY Product\_ID, prediction

ORDER BY Product\_ID

* %sql

SELECT UDI, prediction, count(\*) AS count

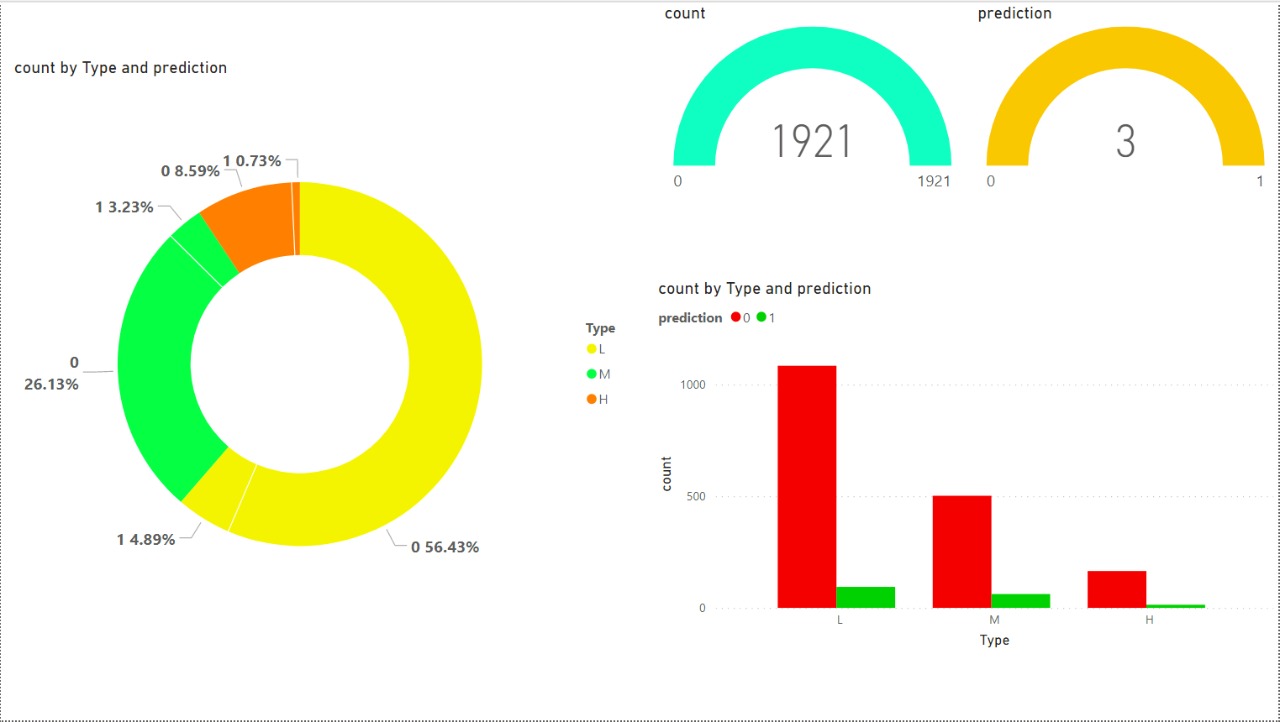
FROM finalPredictions

WHERE prediction = 1

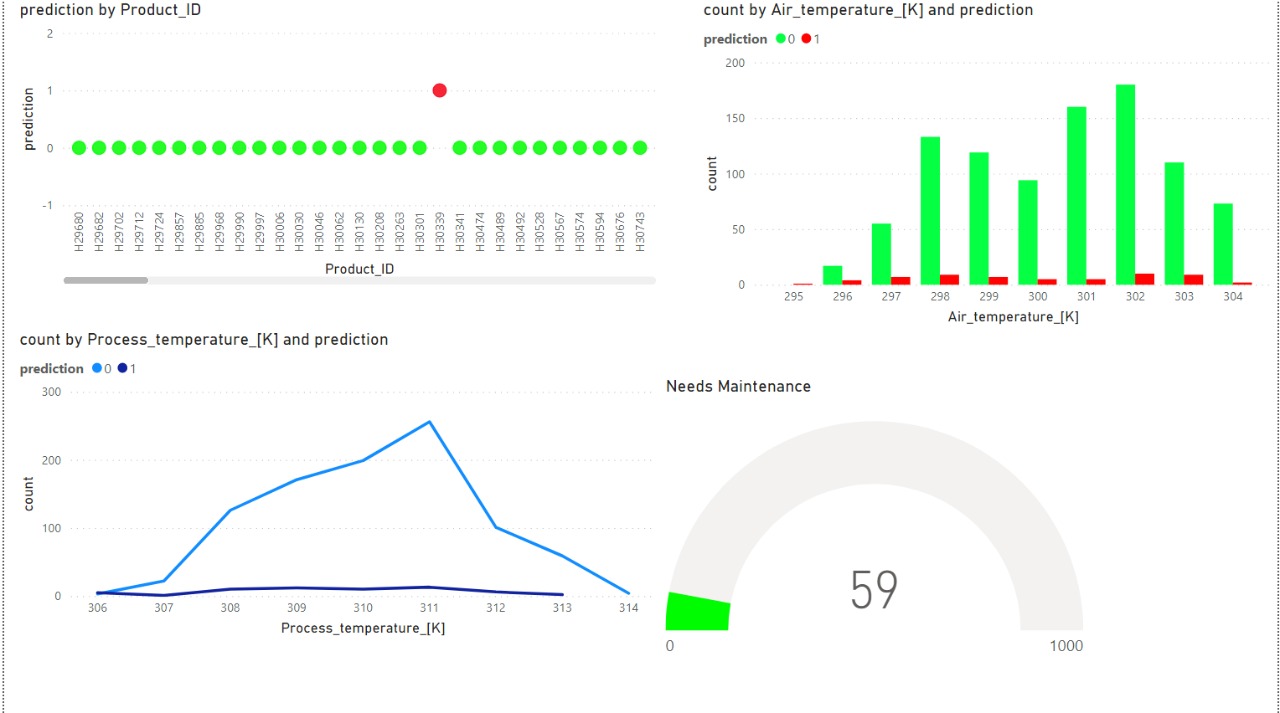
GROUP BY UDI, prediction

ORDER BY UDI

**OUTPUT SCREENS:**

****

**Fig.2**

****

**Fig.3**