Manufacturing Industry 4.0: Classification of Mechanical Faults using Machine Learning Techniques

Abstract

Permanent Magnet Synchronous Machines (PMSMs) are crucial in industrial applications for their efficiency and precise control. However, stator faults pose a significant risk to their reliability. Effective fault detection strategies are vital for ensuring uninterrupted operations and preventing costly downtime. Traditional maintenance methods for detecting stator faults in PMSMs, such as periodic inspections and electrical parameter monitoring, have limitations. Periodic inspections are time-consuming and prone to missed faults, while electrical parameter monitoring may not provide early fault detection. Additionally, distinguishing between normal variations and actual faults can be challenging, leading to false alarms or overlooked faults. This study addresses the need for an advanced stator fault detection strategy that mitigates false alarms, which can disrupt operations and erode trust in the fault detection system. Accurate and timely fault detection is essential for minimizing downtime and maintenance costs while ensuring operational reliability and safety. The Downtime due to stator faults results in significant production losses and maintenance expenses. By developing a more effective fault detection strategy, downtime can be minimized, and maintenance activities can be planned proactively, leading to increased operational efficiency and reduced costs, enhancing operator safety, and preventing potential equipment damage or failures. The proposed system integrates advanced machine learning algorithms and sensor fusion techniques to enhance stator fault detection in PMSMs. By combining data from multiple sensors, including voltage, current, temperature, and vibration sensors, the system captures a comprehensive view of the machine's operating condition. Machine learning models trained on historical data analyze these multi-modal sensor data to identify patterns indicative of stator faults accurately. Additionally, algorithms for false alarm suppression ensure that only genuine fault conditions trigger maintenance alerts.

CHAPTER 1

INTRODUCTION

1.1 History

Permanent Magnet Synchronous Machines (PMSMs) have a rich history dating back to the late 19th century when electrical machinery began to revolutionize industrial processes. The concept of using permanent magnets to generate motion in synchronous machines emerged as a promising alternative to traditional electromagnets. In the early 20th century, significant advancements in magnet materials and manufacturing techniques facilitated the widespread adoption of PMSMs in various applications, including power generation, transportation, and industrial automation.

The development of PMSMs gained momentum during the mid-20th century with the advent of modern power electronics and control systems. The integration of solid-state devices such as transistors and thyristors enabled more precise control over motor operation, leading to improved efficiency and performance. As industries increasingly sought energy-efficient solutions, PMSMs emerged as a preferred choice due to their high efficiency and superior controllability.

In recent decades, advancements in materials science, motor design, and computational modeling have further propelled the evolution of PMSM technology. The integration of rare-earth magnets, such as neodymium and samarium-cobalt, has significantly enhanced motor performance while reducing size and weight. Moreover, advancements in sensor technology and data analytics have enabled the development of sophisticated monitoring and diagnostic systems for PMSMs, enhancing reliability and maintenance efficiency.

Despite their long history and widespread adoption, PMSMs continue to evolve, driven by ongoing research and technological innovation. Emerging trends such as Industry 4.0 and the Internet of Things (IoT) are shaping the future of PMSM technology, ushering in an era of smart, interconnected machines with enhanced monitoring, diagnostics, and predictive maintenance capabilities.

1.2 Research Motivation

The motivation behind the research on stator fault detection in PMSMs stems from the critical role these machines play in modern industrial processes and the need to ensure their reliable and uninterrupted operation. Stator faults pose a significant risk to the reliability and efficiency of PMSMs, leading to costly downtime, maintenance expenses, and potential safety hazards.

The motivation to develop advanced fault detection strategies arises from the limitations of traditional maintenance methods, which are often reactive, time-consuming, and prone to missed faults or false alarms. By leveraging cutting-edge technologies such as machine learning and sensor fusion, researchers aim to overcome these limitations and develop proactive fault detection systems capable of accurately identifying stator faults in real-time.

Furthermore, the increasing demand for energy efficiency, sustainability, and operational reliability in industrial applications underscores the urgency of developing more effective fault detection strategies for PMSMs. By minimizing downtime, optimizing maintenance schedules, and preventing catastrophic failures, advanced fault detection systems can contribute to cost savings, improved productivity, and enhanced safety in industrial environments.

1.3 Problem Statement

The problem addressed in this research is the need for an advanced stator fault detection strategy for PMSMs that mitigates false alarms and ensures accurate and timely fault detection. Traditional maintenance methods, such as periodic inspections and electrical parameter monitoring, are often inadequate for detecting stator faults reliably, leading to missed faults or false alarms that disrupt operations and erode trust in the fault detection system.

Accurate fault detection is essential for minimizing downtime, maintenance costs, and safety risks associated with stator faults. However, distinguishing between normal variations in operating conditions and actual fault conditions can be challenging, particularly in complex industrial environments with varying load conditions and environmental factors.

The goal of this research is to develop a robust and reliable fault detection system that leverages advanced machine learning algorithms and sensor fusion techniques to analyze multi-modal sensor data and accurately identify stator faults in real-time. By addressing the limitations of traditional maintenance methods and mitigating false alarms, the proposed system aims to enhance the operational reliability, efficiency, and safety of PMSMs in industrial applications.

1.4 Applications

Here are the applications of stator fault detection in Permanent Magnet Synchronous Machines (PMSMs) presented in a pointwise format:

- **Manufacturing Industry:**
  - Powering conveyor systems, robotic arms, and automated equipment.
  - Ensuring uninterrupted production processes and meeting production targets.
- **Renewable Energy Systems:**
  - Wind turbines and hydroelectric generators utilize PMSMs for power generation.
  - Detecting stator faults ensures reliable and efficient renewable energy production.
- **Transportation Sector:**
  - Electric vehicles and trains rely on PMSMs for propulsion systems.
  - Accurate fault detection is crucial for passenger safety and system reliability.
- **Robotics and Automation:**
  - Precision machinery and motion control systems in manufacturing, logistics, and healthcare are powered by PMSMs.
  - Fault detection prevents costly equipment damage and ensures uninterrupted operation.
- **Industrial Automation:**
  - PMSMs drive various automation systems in industrial settings, including material handling, assembly lines, and packaging.
  - Detecting stator faults minimizes downtime and maintenance costs, optimizing operational efficiency.
- **Power Distribution and Grid Stability:**

  - PMSMs are employed in grid stabilization systems and power distribution networks.
  - Fault detection enhances grid reliability and prevents disruptions in electrical supply.

- **Marine and Offshore Applications:**
  - Electric propulsion systems in marine vessels and offshore platforms utilize PMSMs.
  - Detecting stator faults ensures safe and efficient operation in maritime environments.

- **Aerospace and Defense:**
  - Aircraft actuation systems and military vehicles utilize PMSMs for motion control.
  - Fault detection enhances system reliability and mission readiness in aerospace and defense applications.

- **Mining and Resources:**
  - PMSMs power equipment used in mining operations, including excavators and conveyor belts.
  - Detecting stator faults minimizes downtime and maintenance expenses in resource extraction industries.

- **Commercial and Residential HVAC Systems:**
  - PMSMs are increasingly used in heating, ventilation, and air conditioning (HVAC) systems for energy efficiency.
  - Fault detection ensures reliable operation and energy savings in commercial and residential buildings.

CHAPTER 2

LITERATURE SURVEY

Electric vehicles (EVs) are attracting more and more attention in transportation due to enhanced performance, safety, and reduced environmental impacts. In particular, permanent magnet synchronous motors (PMSM) are applied widely as traction motors in EVs because of their high efficiency and power density. The healthy operation of the traction motor is crucial for the proper functioning of an EV. Since EV motors run in a harsh environment and complicated operating conditions, the stator winding insulation exhibits a higher failure rate [1]. This fault can lead to a catastrophic accident; therefore, timely identification and diagnosis of insulation faults for traction PMSMs are extremely important to ensure the safe operation of EVs.

It is reported that inter-turn short faults (ITSF) account for 21% of all motor faults [**2**], which can lead to reduced motor efficiency and power output and even catastrophic failure. The majority of ITSFs originate in winding faults, which are caused by insulation malfunctions [3], but rapidly evolve into more severe failures that substantially impact motors. On the one hand, short-circuit paths in the motor can lead to a decline in its performance. These paths allow currents to bypass the normal winding segments [**4**], leading to reduced output power and efficiency. For PMSMs, this type of fault can generate a magnetic field with a higher intensity than the coercivity of the magnets, leading to permanent demagnetization and machine damage. On the other hand, ITSFs cause excessive temperature rises in the motor. Excessive heat can accelerate the aging and embrittlement of insulation materials, potentially leading to burnouts and exacerbating the short-circuit phenomenon [5]. Furthermore, ITSFs increase motor noise and vibration. The presence of short-circuit paths introduces additional electromagnetic forces and vibrational forces in the motor, resulting in abnormal sounds and vibrations [6]. This not only adds to the noise pollution in the working environment but also risks loosening and damaging other components, further exacerbating the development of faults.

The impacts and losses caused by stator winding short circuits in electric motors are extremely severe [**7**]. Therefore, timely diagnosis and repair of these faults are crucial to ensure the safe operation and prolongation of the motor’s lifespan.

The health model of the Kalman filter is used to estimate the residual voltage drop of the rotor reference DQ axis under an ITSF [10]. This observer avoids the use of voltage sensors but does not reduce the diagnostic accuracy of the ITSF. Ali performed KF observations on the current and voltage signals respectively [11], using the residual signal as the fault detection index; this method was robust against different fault resistances. However, linear KF cannot be used for systems with significant nonlinearity. Since most systems are nonlinear, suboptimal state estimation techniques can be employed. The extended Kalman filter (EKF) is one of these suboptimal techniques [12], where the measurement and system model equations are linearized, enabling the application of the linear Kalman filter algorithm. Nonetheless, the linearization in EKF may introduce instability to the method, particularly when dealing with extremely nonlinear systems. To overcome the limitations of EKF, the unscented Kalman filter (UKF) was proposed in [13]. The UKF employs a set of sigma points to estimate the propagation of the mean and covariance matrix [14]. EKF and UKF were used to detect the percentage and location of faults [15]. Another difference in the method is that the ratio of short-circuit turns is used as the state estimator.

The ITSF diagnosis method based on the Luenberger state observer and current second-order harmonics was established in [16]. The advantage of this method is the ability to assess the severity of failures, as well as the efficiency with which failures can be detected at an early stage and under various operating conditions. A high-order sliding mode observer was used to estimate the rotor flux and three-phase stator current in the fault state [17]. By comparing the measured and estimated values of stator three-phase currents, a fault detection method was designed. This comparison produces a set of residuals that are sensitive to failure. The analysis of these residual signals can be used to detect the damage of the stator windings. An equivalent model of the single-phase interturn fault motor served as the observer [18], where the error between the measured current and the estimated current were corrected as the core of the fault severity estimator. A sufficiently accurate model is established to determine the variation of different variables in the motor under this fault condition, and then the residual generated by the sliding mode observer is used to detect the ITSF. In another study, a PMSM model of single-phase short-circuit fault is established, and a sliding mode observer is developed to extract voltage disturbance information from the derived equivalent control signal to detect interturn faults [19]. However, the Luenberger observer is sensitive to changes in motor parameters.

CHAPTER 3

EXISTING SYSTEM

3.1 Ridge Classifier

Ridge Classifier is a linear classification algorithm used in machine learning for distinguishing between two or more classes based on input features. It is an extension of the Ridge Regression algorithm and is particularly useful when dealing with datasets that have multicollinearity or when the number of features is larger than the number of samples. Below is a detailed explanation of the principle, working, and process of the Ridge Classifier, along with its disadvantages.

Principle:

The principle behind the Ridge Classifier lies in its regularization technique, similar to Ridge Regression. It adds a penalty term to the traditional loss function used in linear classifiers, such as logistic regression or linear SVM. This penalty term, based on the L2-norm of the coefficient vector, helps in controlling the model complexity and improving generalization performance by penalizing large coefficient values.

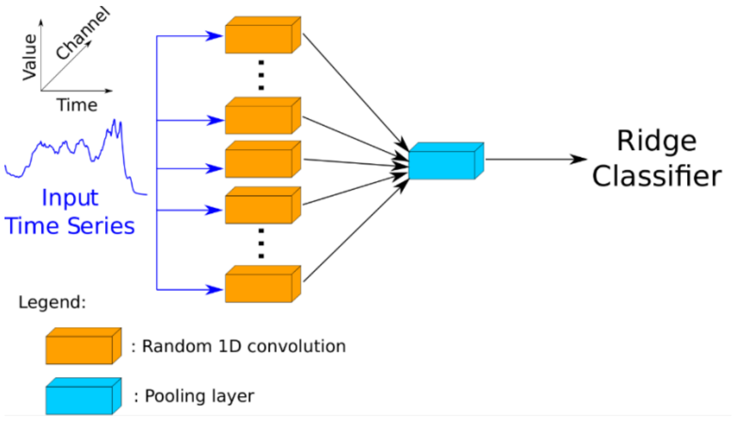


Fig. 1: Architectural diagram of Ridge Classifier model.

Working:

1. **Objective Function:** The objective function of the Ridge Classifier combines the traditional loss function with the regularization term. Mathematically, it can be expressed as: $J(\theta)=\text{Loss}(\theta)+\alpha\sum i=1n\theta i2J(\theta)=\text{Loss}(\theta)+\alpha\sum i=1n\theta i2$ where $J(\theta)J(\theta)$ is the

total cost function, Loss(θ)Loss(θ) represents the loss function (e.g., logistic loss for binary classification), $\alpha\alpha$ is the regularization parameter, and $\sum i{=}1n\theta i2\sum i{=}1n\theta i2$ is the L2-norm of the coefficient vector.

2. **Regularization Term:** Similar to Ridge Regression, the regularization term in Ridge Classifier penalizes large coefficient values, thereby constraining the model complexity and reducing the risk of overfitting. By adding this term to the objective function, Ridge Classifier promotes smoother decision boundaries and improves the model's ability to generalize to unseen data.

3. **Optimization:** The optimization process in Ridge Classifier involves finding the optimal values of the coefficient vector $\theta\theta$ that minimize the total cost function $J(\theta)J(\theta)$. This optimization problem can be solved using various optimization algorithms, such as gradient descent or convex optimization techniques.

4. **Regularization Parameter Tuning:** Similar to Ridge Regression, the regularization parameter $\alpha\alpha$ in Ridge Classifier controls the degree of regularization. Cross-validation techniques can be used to select the optimal value of $\alpha\alpha$ that maximizes the model's performance on validation data while preventing overfitting.

5. **Model Evaluation:** Once the Ridge Classifier model is trained and tuned, it is evaluated using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, or area under the ROC curve (AUC). These metrics assess the model's performance in correctly classifying instances into their respective classes.

Disadvantages:

— **Limited Feature Selection:** Ridge Classifier does not perform feature selection inherently, similar to Ridge Regression. It may retain all features in the model, even if some features are irrelevant or redundant.

— **Assumption of Linearity:** Ridge Classifier assumes a linear relationship between the input features and the class labels. If the true decision boundary is non-linear, Ridge Classifier may not capture it accurately.

— **Sensitivity to Outliers:** Ridge Classifier is sensitive to outliers, particularly when the regularization parameter is not properly tuned. Outliers can influence the decision boundary and affect the classification performance.

— **Difficulty in Interpretability:** The regularization term in Ridge Classifier can make the interpretation of the model coefficients less straightforward, similar to Ridge Regression. It becomes challenging to assess the individual impact of each feature on the class labels.

— **Requirement of Regularization Parameter Tuning:** Ridge Classifier requires tuning of the regularization parameter ($\alpha\alpha$) to achieve optimal model performance. Selecting the appropriate value of $\alpha\alpha$ involves a trade-off between bias and variance, requiring careful experimentation.

— **Inability to Handle Non-linear Relationships:** Ridge Classifier is inherently a linear model and cannot capture non-linear relationships between the input features and the class labels. If the decision boundary is non-linear, Ridge Classifier may underperform compared to non-linear classification algorithms.

— **Possible Overfitting with Large Sample Sizes:** In cases where the sample size is significantly larger than the number of features, Ridge Classifier may still overfit the data, similar to Ridge Regression. Despite the regularization term, Ridge Classifier may struggle to generalize well when the number of observations is very large.

CHAPTER 4

PROPOSED SYSTEM

4.1 Overview

This project focuses on the development and evaluation of stator fault detection strategies in Permanent Magnet Synchronous Machines (PMSMs) using machine learning techniques. Let's break down the key components and functionalities of the code:

- **Importing Libraries and Modules:** By importing necessary libraries and modules such as NumPy, Pandas, Matplotlib, Seaborn, scikit-learn, and CatBoost. These libraries provide functionalities for data manipulation, visualization, model building, and evaluation.

- **Importing Dataset:** The dataset containing various electrical parameters of PMSMs is imported using Pandas' **read_csv** function. This dataset serves as the foundation for training and testing machine learning models for stator fault detection.

- **Data Analysis and Visualization:** Exploratory data analysis (EDA) techniques are employed to gain insights into the dataset's characteristics. Descriptive statistics, correlation analysis, and visualization using Seaborn are utilized to understand the distribution of data and identify patterns relevant to stator fault detection.

- **Data Preprocessing:** Data preprocessing steps such as handling missing values, encoding categorical variables, and splitting the dataset into independent variables (features) and the target variable (stator fault) are performed. Additionally, the dataset is divided into training and testing sets using scikit-learn's **train_test_split** function.

- **Model Building:** Two classification algorithms, namely Ridge Classifier and CatBoost Classifier, are chosen for stator fault detection. Ridge Classifier is a linear classification algorithm, while CatBoost Classifier is a gradient boosting algorithm specifically designed to handle categorical features efficiently. Both models are trained using the training data.

— **Performance Evaluation:** The performance of each classifier is evaluated using various evaluation metrics such as accuracy, precision, recall, F1-score, and confusion matrix. The se metrics provide insights into the models' ability to accurately classify instances into their respective classes, including the detection of stator faults..

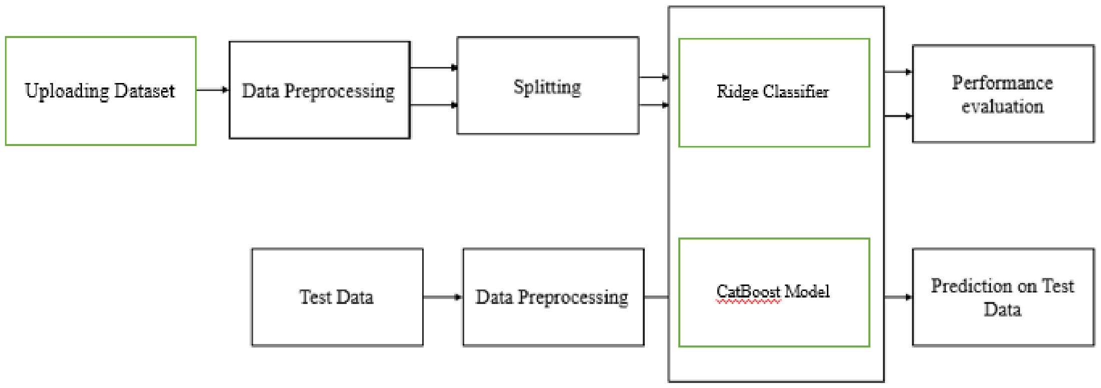


Figure 4.1: Block Diagram of Proposed System.

4.2 Data Preprocessing

Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model. It is the first and crucial step while creating a machine learning model. When creating a machine learning project, it is not always a case that we come across the clean and formatted data. And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data pre-processing task. A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data pre-processing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

- Getting the dataset
- Importing libraries
- Importing datasets
- Finding Missing Data
- Encoding Categorical Data

- Splitting dataset into training and test set

Importing Libraries: To perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: Numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

import numpy as nm

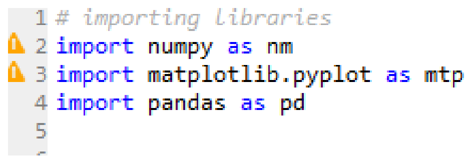
Here we have used nm, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is matplotlib, which is a Python 2D plotting library, and with this library, we need to import a sub-library pyplot. This library is used to plot any type of charts in Python for the code. It will be imported as below:

import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. Here, we have used pd as a short name for this library. Consider the below image:


```python
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

Handling Missing data: The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset. There are mainly two ways to handle missing data, which are:

- By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

- By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc.

Encoding Categorical data: Categorical data is data which has some categories such as, in our dataset; there are two categorical variables, Country, and Purchased. Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So, it is necessary to encode these categorical variables into numbers.

4.3 Splitting the Dataset

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model. Suppose if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models. If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:

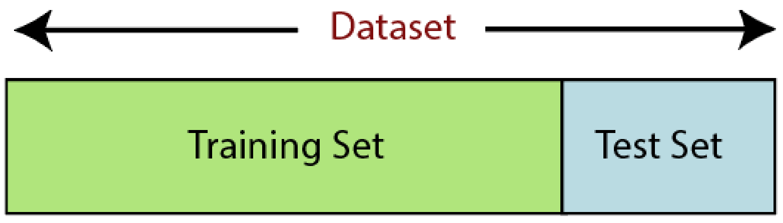


Figure 4.2: Splitting the dataset.

Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state=0)

Explanation

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
- x_train: features for the training data
- x_test: features for testing data
- y_train: Dependent variables for training data
- y_test: Independent variable for testing data
- In train_test_split() function, we have passed four parameters in which first two are for arrays of data, and test_size is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.
- The last parameter random_state is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

4.4 CatBoost Classifier:

CatBoost Classifier is a gradient boosting algorithm designed for classification tasks, particularly when dealing with categorical features. It belongs to the family of ensemble learning methods and is known for its robustness, efficiency, and ability to handle categorical variables without the need for extensive preprocessing. Below is a detailed explanation of the principle, working, and process of the CatBoost Classifier, along with its disadvantages.

Principle:

The principle behind the CatBoost Classifier lies in its gradient boosting framework, which combines multiple weak learners (decision trees) to create a strong predictive model. CatBoost stands for "Categorical Boosting," indicating its capability to handle categorical features effectively. It employs a variant of gradient boosting that incorporates techniques to handle categorical variables and mitigate overfitting.

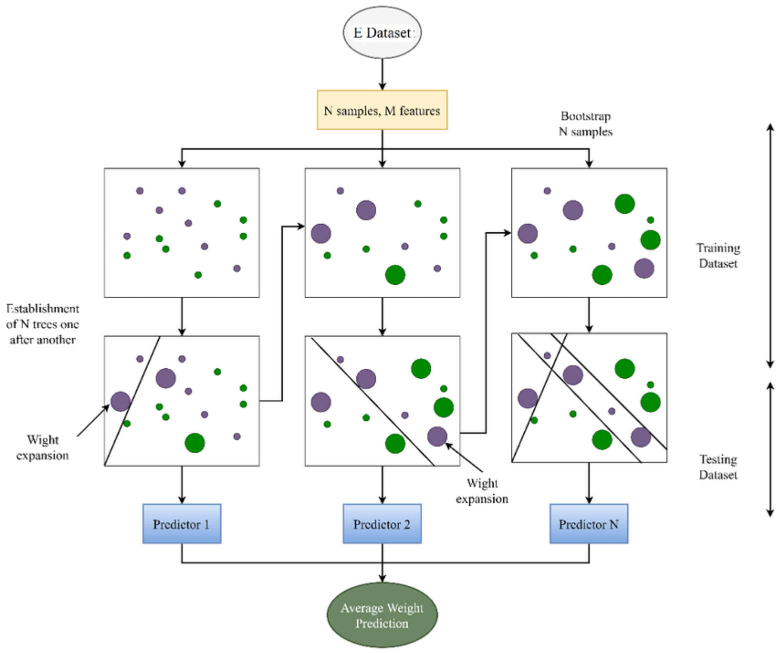


Figure 4.3: Cat Boost Classifier Model Diagram.

Working:

1. Gradient Boosting Objective Function: The objective function of gradient boosting aims to minimize the loss between the predicted values and the true labels. In CatBoost, the objective function typically involves a differentiable loss function such as log-loss for binary classification or multinomial logistic loss for multiclass classification. Mathematically, the objective function can be expressed as:

$$J(\theta)=\sum i=1NL(yi,F(xi))J(\theta)=\sum i=1NL(yi,F(xi))$$

Where:

- $J(\theta)J(\theta)$ represents the total loss function.
- NN is the total number of samples.
- $yiyi$ is the true label of sample ii.
- $F(xi)F(xi)$ is the predicted value for sample ii.

- $L(yi,F(xi))L(yi,F(xi))$ is the loss function that measures the discrepancy between the true label and the predicted value for sample ii.

2. Regularization Techniques: CatBoost incorporates various regularization techniques to prevent overfitting and improve model generalization. One such technique is depth regularization, which penalizes deeper trees to control model complexity. Mathematically, the depth regularization term can be expressed as:

$$Rdepth=\lambda\sum t=1T\gamma t2Rdepth=\lambda\sum t=1T\gamma t2$$

Where:

- $RdepthRdepth$ represents the depth regularization term.
- $\lambda\lambda$ is the regularization parameter controlling the strength of regularization.
- TT is the total number of trees in the ensemble.
- $\gamma t\gamma t$ is the depth of tree tt.

3. Learning Rate Adaptation: CatBoost introduces an adaptive learning rate strategy that adjusts the learning rate dynamically based on the current tree structure and feature importance. The adaptive learning rate $\eta\eta$ for each tree can be computed as:

$$\eta = \frac{1}{1+\text{num\_trees}\cdot\text{leaf\_estimation\_iterations}}$$

Where:

- $\eta\eta$ is the adaptive learning rate.
- num_treesnum_trees is the current number of trees in the ensemble.
- leaf_estimation_iterationsleaf_estimation_iterations is the number of iterations used for leaf value estimation.

4. Categorical Feature Handling: CatBoost employs an efficient algorithm to handle categorical features during tree construction. It assigns numerical values to categorical variables based on their frequency and the target variable's response, preserving their semantic meaning. The categorical feature handling process is integral to CatBoost's performance and efficiency.

5. Parallel and GPU Training: CatBoost implements efficient parallelization techniques and supports GPU acceleration, allowing for fast training on large datasets. By leveraging parallel processing and GPU resources, CatBoost achieves significant speedups compared to traditional gradient boosting implementations.

Disadvantages:

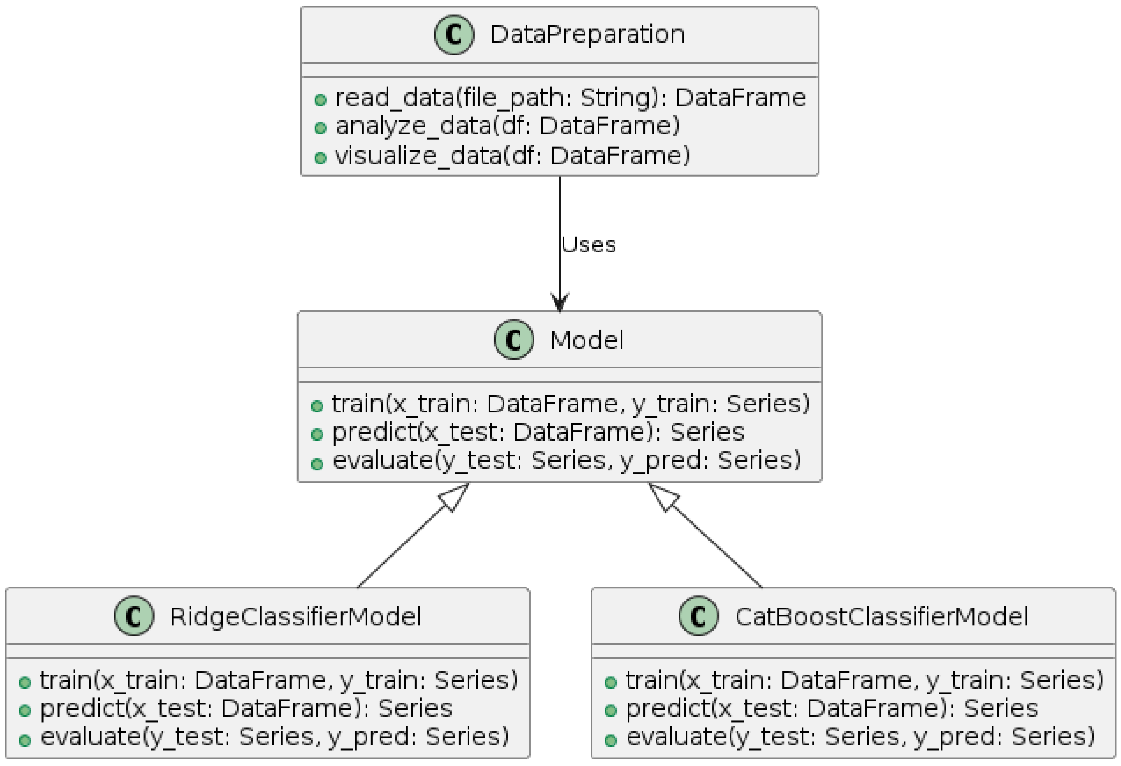
- **Slow Training Time:** Despite its efficiency improvements, CatBoost training can still be slower compared to simpler algorithms like logistic regression or decision trees. The algorithm's complexity and the need for extensive tree building iterations contribute to longer training times, especially on large datasets.
- **High Memory Consumption:** CatBoost requires significant memory resources, particularly when dealing with high-dimensional datasets or datasets with a large number of categorical features. The algorithm's internal data structures and the need to store intermediate results during training contribute to high memory consumption.
- **Sensitivity to Hyperparameters:** CatBoost performance is sensitive to hyperparameter tuning, including parameters related to tree depth, learning rate, regularization, and feature combinations. Finding the optimal set of hyperparameters can be challenging and may require extensive experimentation.
- **Potential Overfitting:** Despite its regularization techniques, CatBoost is susceptible to overfitting, especially when training on noisy or small datasets. Careful hyperparameter tuning and regularization strategies are necessary to prevent overfitting and ensure good generalization performance.
- **Limited Interpretability:** Like other ensemble learning methods, CatBoost models are inherently complex, making them less interpretable compared to simpler models like logistic regression or decision trees. Understanding the individual contributions of features or the decision-making process of the model can be challenging.
- **Difficulty in Handling Imbalanced Data:** CatBoost may struggle to effectively handle imbalanced datasets, where one class is significantly more prevalent than the others. While it provides options for class weighting and sampling techniques, finding the right balance between different strategies can be challenging.

— **Dependency on Data Quality:** The performance of CatBoost heavily depends on the quality and representativeness of the training data. Noisy or biased data can lead to suboptimal model performance and may require preprocessing steps such as data cleaning or feature engineering.

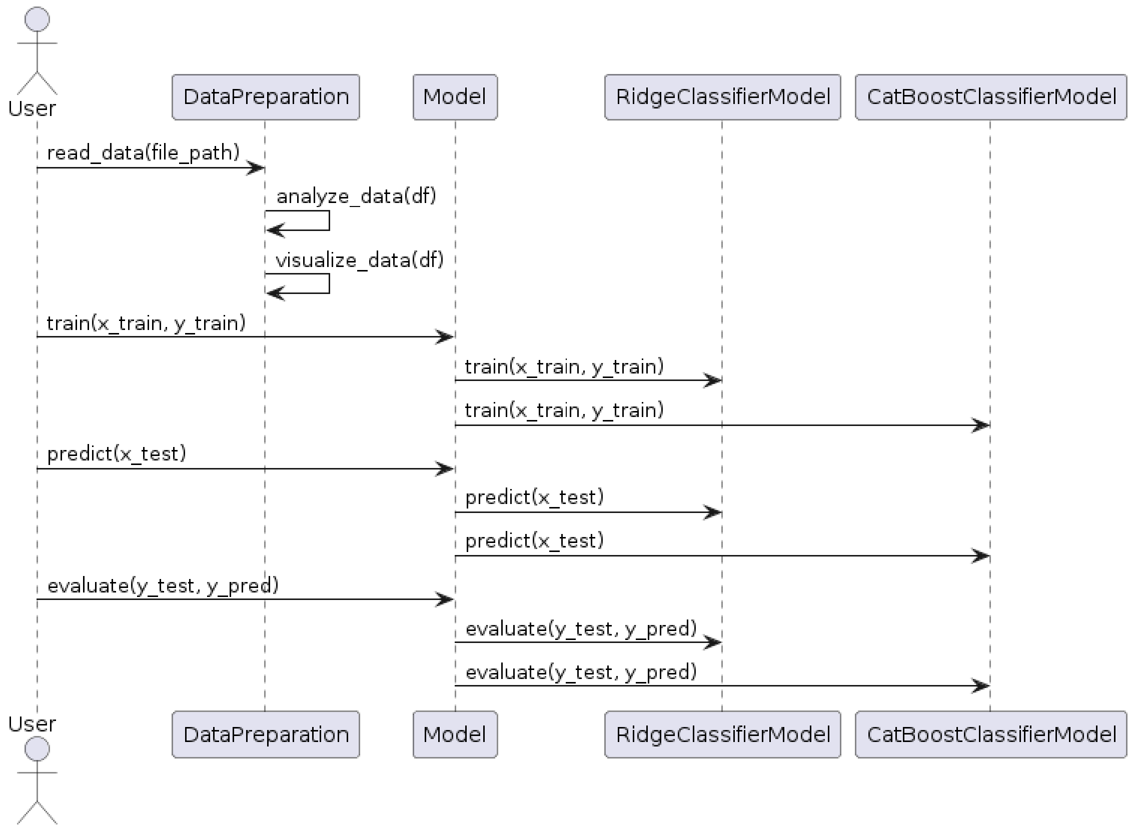
CHAPTER 5

UML DIAGRAMS

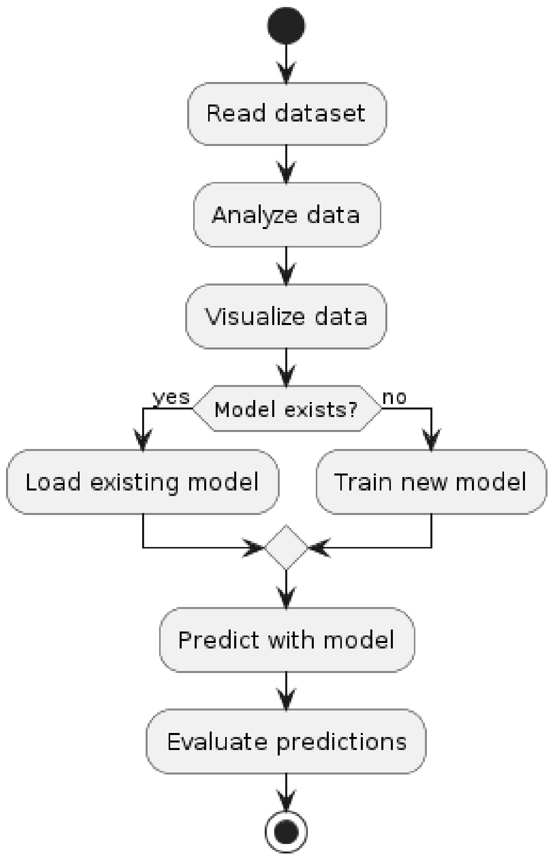
Class Diagram: Class diagram is *a static diagram. It represents the static view of an application*.


DataPreparation
read_data(file_path: String): DataFrame
analyze_data(df: DataFrame)
visualize_data(df: DataFrame)
Uses
Model
train(x_train: DataFrame, y_train: Series)
predict(x_test: DataFrame): Series
evaluate(y_test: Series, y_pred: Series)
RidgeClassifierModel
train(x_train: DataFrame, y_train: Series)
predict(x_test: DataFrame): Series
evaluate(y_test: Series, y_pred: Series)
CatBoostClassifierModel
train(x_train: DataFrame, y_train: Series)
predict(x_test: DataFrame): Series
evaluate(y_test: Series, y_pred: Series)

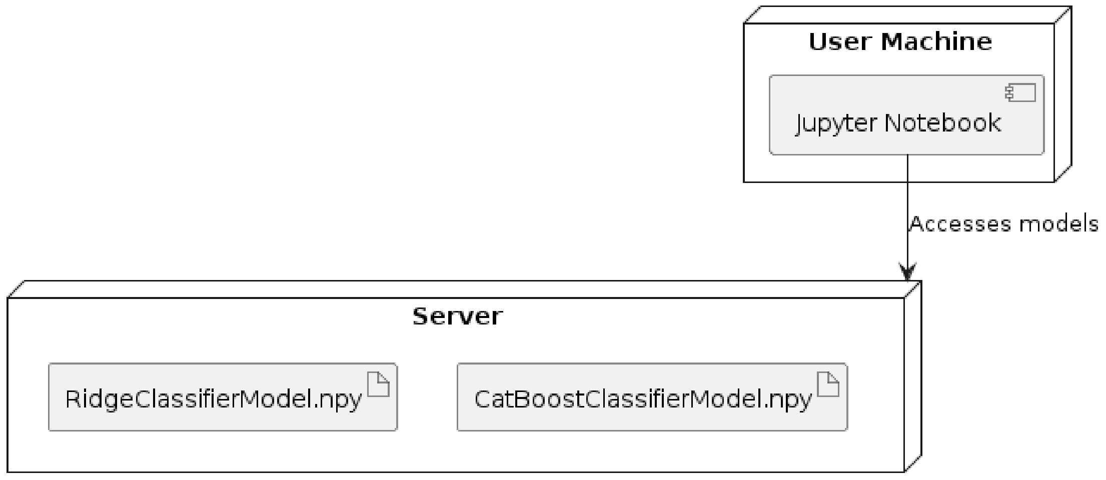
Sequence Diagram: Sequence diagram is an interaction diagram that details how operations are carried out.



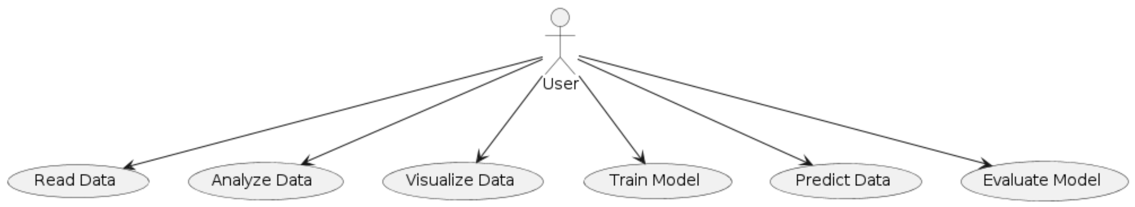
<u>Activity diagram</u>: Activity diagram is another important diagram in UML to *describe the dynamic aspects of the system*.



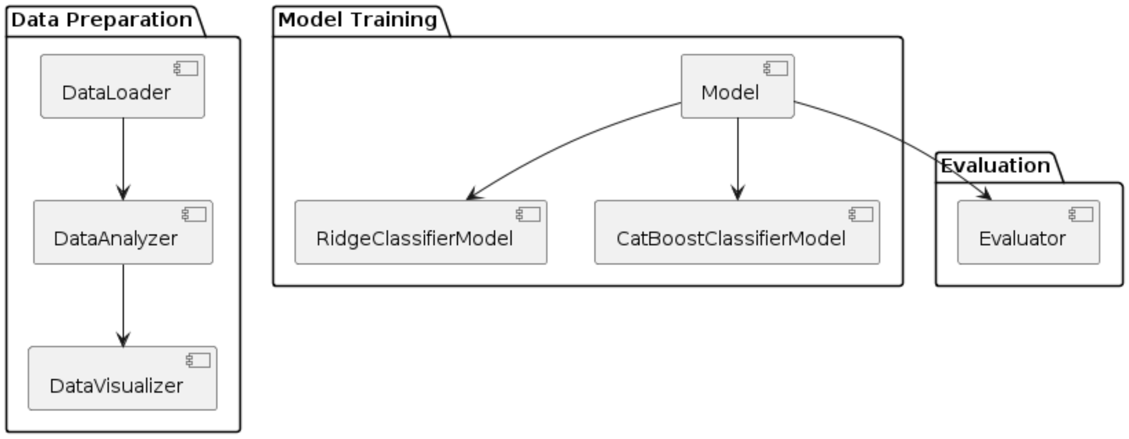
<u>Deployment diagram</u>: The deployment diagram *visualizes the physical hardware on which the software will be deployed*.



<u>Usecase diagram:</u> The purpose of use case diagram is *to capture the dynamic aspect of a system*.



Component diagram: Component diagram describes the organization and wiring of the physical components in a system.



CHAPTER 6

SOFTWARE ENVIRONMENT

What is Python?

Below are some facts about Python.

- Python is currently the most widely used multi-purpose, high-level programming language.
- Python allows programming in Object-Oriented and Procedural paradigms. Python programs generally are smaller than other programming languages like Java.
- Programmers have to type relatively less and indentation requirement of the language, makes them readable all the time.
- Python language is being used by almost all tech-giant companies like – Google, Amazon, Facebook, Instagram, Dropbox, Uber… etc.

The biggest strength of Python is huge collection of standard library which can be used for the following –

- Machine Learning
- GUI Applications (like Kivy, Tkinter, PyQt etc.)
- Web frameworks like Django (used by YouTube, Instagram, Dropbox)
- Image processing (like Opencv, Pillow)
- Web scraping (like Scrapy, BeautifulSoup, Selenium)
- Test frameworks
- Multimedia

Advantages of Python

Let’s see how Python dominates over other languages.

1. Extensive Libraries

Python downloads with an extensive library and it contain code for various purposes like regular expressions, documentation-generation, unit-testing, web browsers, threading, databases, CGI, email, image manipulation, and more. So, we don’t have to write the complete code for that manually.

2. Extensible

As we have seen earlier, Python can be extended to other languages. You can write some of your code in languages like C++ or C. This comes in handy, especially in projects.

3. Embeddable

Complimentary to extensibility, Python is embeddable as well. You can put your Python code in your source code of a different language, like C++. This lets us add scripting capabilities to our code in the other language.

4. Improved Productivity

The language’s simplicity and extensive libraries render programmers more productive than languages like Java and C++ do. Also, the fact that you need to write less and get more things done.

5. IOT Opportunities

Since Python forms the basis of new platforms like Raspberry Pi, it finds the future bright for the Internet Of Things. This is a way to connect the language with the real world.

6. Simple and Easy

When working with Java, you may have to create a class to print ‘Hello World’. But in Python, just a print statement will do. It is also quite easy to learn, understand, and code. This is why when people pick up Python, they have a hard time adjusting to other more verbose languages like Java.

7. Readable

Because it is not such a verbose language, reading Python is much like reading English. This is the reason why it is so easy to learn, understand, and code. It also does not need curly braces to define blocks, and indentation is mandatory. This further aids the readability of the code.

8. Object-Oriented

This language supports both the procedural and object-oriented programming paradigms. While functions help us with code reusability, classes and objects let us model the real world. A class allows the encapsulation of data and functions into one.

9. Free and Open-Source

Like we said earlier, Python is freely available. But not only can you download Python for free, but you can also download its source code, make changes to it, and even distribute it. It downloads with an extensive collection of libraries to help you with your tasks.

10. Portable

When you code your project in a language like C++, you may need to make some changes to it if you want to run it on another platform. But it isn't the same with Python. Here, you need to code only once, and you can run it anywhere. This is called Write Once Run Anywhere (WORA). However, you need to be careful enough not to include any system-dependent features.

11. Interpreted

Lastly, we will say that it is an interpreted language. Since statements are executed one by one, debugging is easier than in compiled languages.

Any doubts till now in the advantages of Python? Mention in the comment section.

Advantages of Python Over Other Languages

1. Less Coding

Almost all of the tasks done in Python requires less coding when the same task is done in other languages. Python also has an awesome standard library support, so you don't have to search for any third-party libraries to get your job done. This is the reason that many people suggest learning Python to beginners.

2. Affordable

Python is free therefore individuals, small companies or big organizations can leverage the free available resources to build applications. Python is popular and widely used so it gives you better community support.

The 2019 Github annual survey showed us that Python has overtaken Java in the most popular programming language category.

3. Python is for Everyone

Python code can run on any machine whether it is Linux, Mac or Windows. Programmers need to learn different languages for different jobs but with Python, you can professionally build web apps, perform data analysis and machine learning, automate things, do web scraping and also build games and powerful visualizations. It is an all-rounder programming language.

Disadvantages of Python

So far, we've seen why Python is a great choice for your project. But if you choose it, you should be aware of its consequences as well. Let's now see the downsides of choosing Python over another language.

1. Speed Limitations

We have seen that Python code is executed line by line. But since Python is interpreted, it often results in slow execution. This, however, isn't a problem unless speed is a focal point for the project. In other words, unless high speed is a requirement, the benefits offered by Python are enough to distract us from its speed limitations.

2. Weak in Mobile Computing and Browsers

While it serves as an excellent server-side language, Python is much rarely seen on the client-side. Besides that, it is rarely ever used to implement smartphone-based applications. One such application is called Carbonnelle.

The reason it is not so famous despite the existence of Brython is that it isn't that secure.

3. Design Restrictions

As you know, Python is dynamically typed. This means that you don't need to declare the type of variable while writing the code. It uses duck-typing. But wait, what's that? Well, it just means that if it looks like a duck, it must be a duck. While this is easy on the programmers during coding, it can raise run-time errors.

4. Underdeveloped Database Access Layers

Compared to more widely used technologies like JDBC (Java DataBase Connectivity) and ODBC (Open DataBase Connectivity), Python's database access layers are a bit underdeveloped. Consequently, it is less often applied in huge enterprises.

5. Simple

No, we're not kidding. Python's simplicity can indeed be a problem. Take my example. I don't do Java, I'm more of a Python person. To me, its syntax is so simple that the verbosity of Java code seems unnecessary.

This was all about the Advantages and Disadvantages of Python Programming Language.

Modules Used in Project

NumPy

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code
- Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary datatypes can be defined using NumPy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Pandas

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. Python was majorly used for data munging and preparation. It had very little contribution towards data analysis. Pandas solved this problem. Using Pandas, we can accomplish five typical steps in the processing and analysis of data, regardless of the origin of data load, prepare, manipulate, model, and analyze. Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Matplotlib

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and Ipython shells, the Jupyter Notebook, web application servers, and four graphical user interface toolkits. Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, error charts, scatter plots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

For simple plotting the pyplot module provides a MATLAB-like interface, particularly when combined with Ipython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Scikit – learn

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python. It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use. Python

Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular high-level programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac

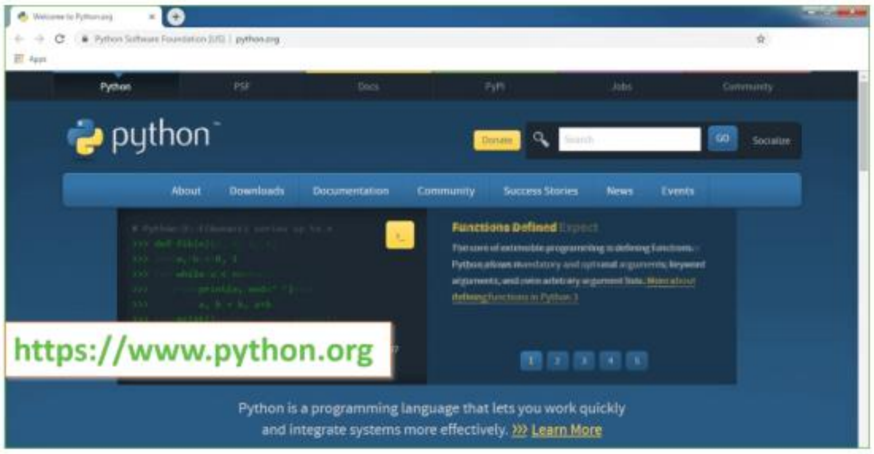
There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e. operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here.The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

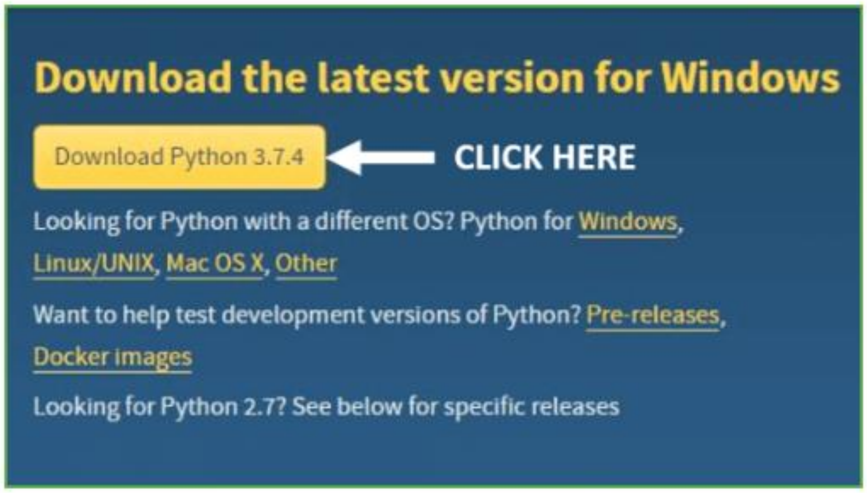
Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: https://www.python.org



Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.

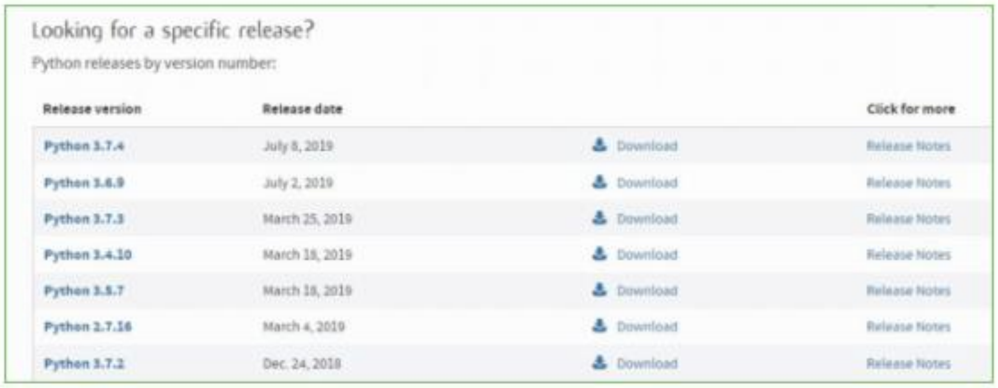

Download the latest version for Windows

Looking for Python with a different OS? Python for Windows, Linux/UNIX, Mac OS X, Other

Want to help test development versions of Python? Pre-releases, Docker images

Looking for Python 2.7? See below for specific releases

Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

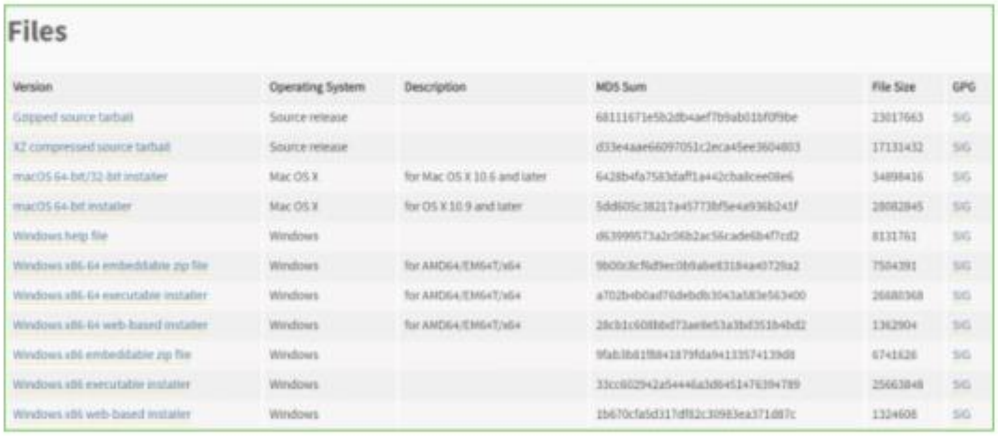

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.



Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4aef7b9ab01bf0f9be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aae66097051c2eca45ee3604803	17131432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daff1a442cba8cee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		d63999573a2c56b2ac56cade6b4f7cd2	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9b00c8cf6d9ec0b9abe83184a40729a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0ad76debdb3043a583e563400	26680368	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28cb1c608bbd73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3b81f8841879fda94133574139d8	6741626	SIG
Windows x86 executable installer	Windows		33cc602942a54446a3d6451476394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317df82c30983ea371d87c	1324608	SIG

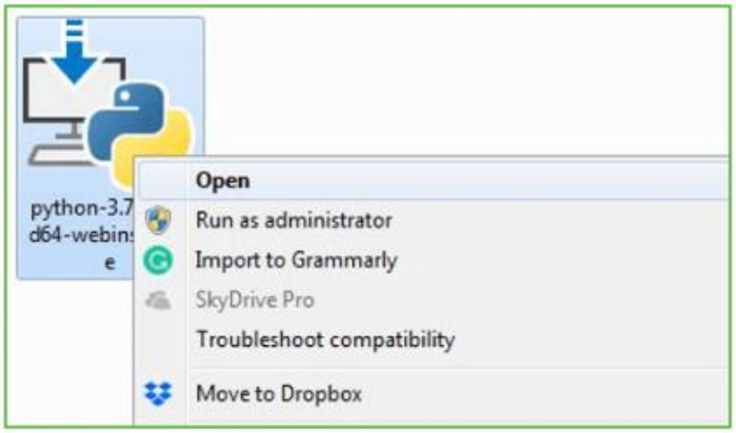
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e. Installation

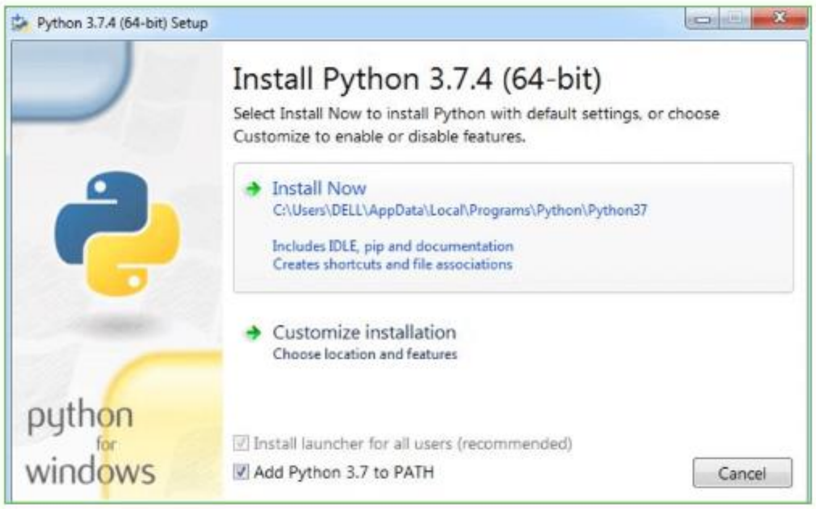
Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, Make sure to put a tick on Add Python 3.7 to PATH.



Python 3.7.4 (64-bit) Setup

Install Python 3.7.4 (64-bit)

Select Install Now to install Python with default settings, or choose Customize to enable or disable features.

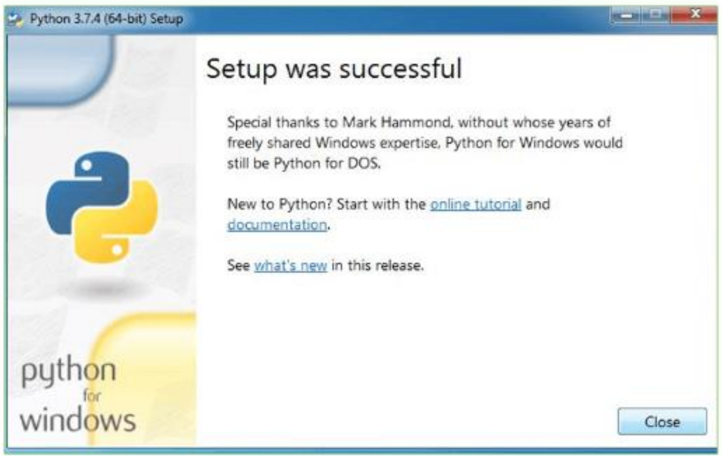
Install Now
C:\Users\DELL\AppData\Local\Programs\Python\Python37

Includes IDLE, pip and documentation
Creates shortcuts and file associations

Customize installation
Choose location and features

☑ Install launcher for all users (recommended)
☑ Add Python 3.7 to PATH

Step 3: Click on Install NOW After the installation is successful. Click on Close.



Python 3.7.4 (64-bit) Setup

Setup was successful

Special thanks to Mark Hammond, without whose years of freely shared Windows expertise, Python for Windows would still be Python for DOS.

New to Python? Start with the online tutorial and documentation.

See what's new in this release.

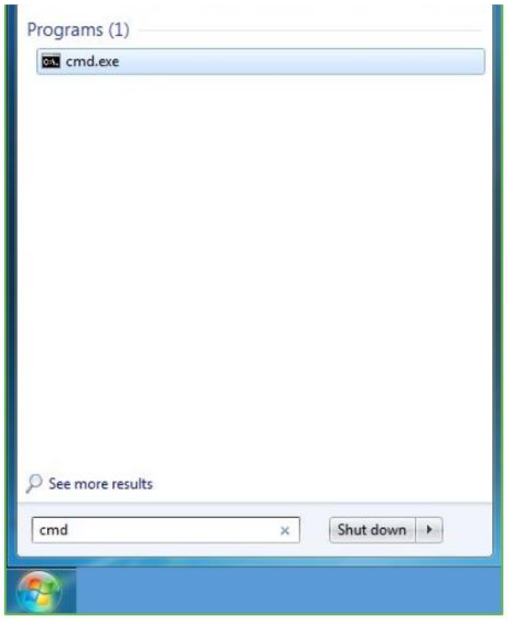
With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

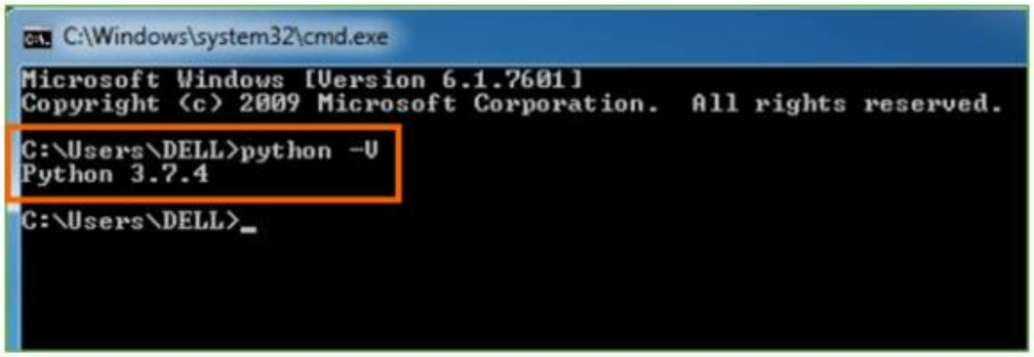
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type python –V and press Enter.



C:\Windows\system32\cmd.exe

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\DELL>python -V
Python 3.7.4

C:\Users\DELL>_
```

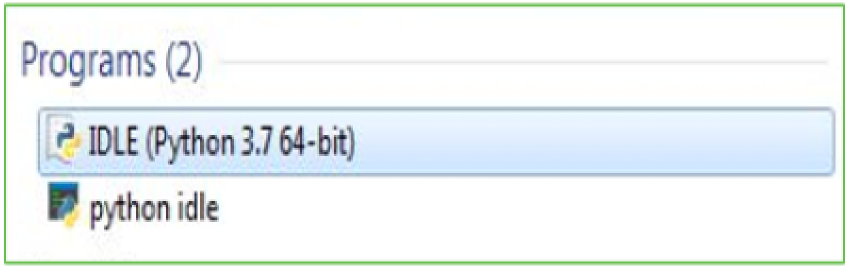
Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

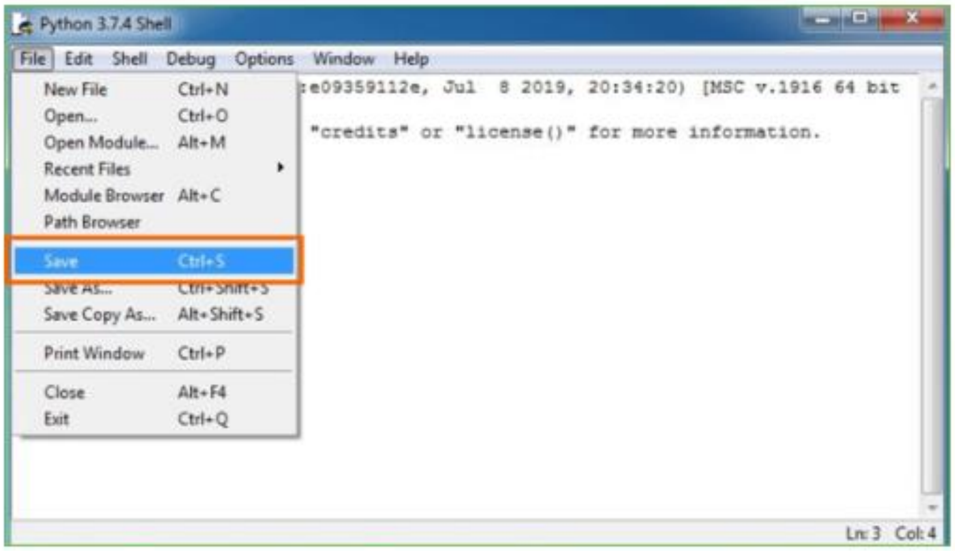
Step 1: Click on Start

Step 2: In the Windows Run command, type "python idle".



Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

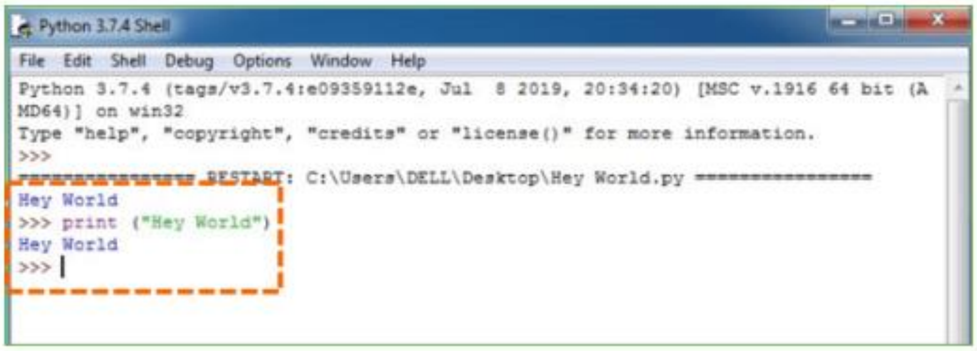
Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



```
:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit
 "credits" or "license()" for more information.
```

Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g. enter print ("Hey World") and Press Enter.



```
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul  8 2019, 20:34:20) [MSC v.1916 64 bit (A
MD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================ RESTART: C:\Users\DELL\Desktop\Hey World.py ================
Hey World
>>> print ("Hey World")
Hey World
>>>
```

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won’t work.

CHAPTER 7

SYSTEM REQUIREMENTS

Software Requirements

The functional requirements or the overall description documents include the product perspective and features, operating system and operating environment, graphics requirements, design constraints and user documentation.

The appropriation of requirements and implementation constraints gives the general overview of the project in regard to what the areas of strength and deficit are and how to tackle them.

- Python IDLE 3.7 version (or)
- Anaconda 3.7 (or)
- Jupiter (or)
- Google colab

Hardware Requirements

Minimum hardware requirements are very dependent on the particular software being developed by a given Enthought Python / Canopy / VS Code user. Applications that need to store large arrays/objects in memory will require more RAM, whereas applications that need to perform numerous calculations or tasks more quickly will require a faster processor.

• Operating system	:	Windows, Linux
• Processor	:	minimum intel i3
• Ram	:	minimum 4 GB
• Hard disk	:	minimum 250GB

CHAPTER 8

FUNCTIONAL REQUIREMENTS

Output Design

Outputs from computer systems are required primarily to communicate the results of processing to users. They are also used to provides a permanent copy of the results for later consultation. The various types of outputs in general are:

- External Outputs, whose destination is outside the organization
- Internal Outputs whose destination is within organization and they are the
- User's main interface with the computer.
- Operational outputs whose use is purely within the computer department.
- Interface outputs, which involve the user in communicating directly.

Output Definition

The outputs should be defined in terms of the following points:

- Type of the output
- Content of the output
- Format of the output
- Location of the output
- Frequency of the output
- Volume of the output
- Sequence of the output

It is not always desirable to print or display data as it is held on a computer. It should be decided as which form of the output is the most suitable.

Input Design

Input design is a part of overall system design. The main objective during the input design is as given below:

- To produce a cost-effective method of input.
- To achieve the highest possible level of accuracy.
- To ensure that the input is acceptable and understood by the user.

Input Stages

The main input stages can be listed as below:

- Data recording
- Data transcription
- Data conversion
- Data verification
- Data control
- Data transmission
- Data validation
- Data correction

Input Types

It is necessary to determine the various types of inputs. Inputs can be categorized as follows:

- External inputs, which are prime inputs for the system.
- Internal inputs, which are user communications with the system.
- Operational, which are computer department’s communications to the system?
- Interactive, which are inputs entered during a dialogue.

Input Media

At this stage choice has to be made about the input media. To conclude about the input media consideration has to be given to;

- Type of input
- Flexibility of format
- Speed
- Accuracy
- Verification methods
- Rejection rates
- Ease of correction
- Storage and handling requirements
- Security
- Easy to use
- Portability

Keeping in view the above description of the input types and input media, it can be said that most of the inputs are of the form of internal and interactive. As

Input data is to be the directly keyed in by the user, the keyboard can be considered to be the most suitable input device.

Error Avoidance

At this stage care is to be taken to ensure that input data remains accurate form the stage at which it is recorded up to the stage in which the data is accepted by the system. This can be achieved only by means of careful control each time the data is handled.

Error Detection

Even though every effort is make to avoid the occurrence of errors, still a small proportion of errors is always likely to occur, these types of errors can be discovered by using validations to check the input data.

Data Validation

Procedures are designed to detect errors in data at a lower level of detail. Data validations have been included in the system in almost every area where there is a possibility for the user to commit errors. The system will not accept invalid data. Whenever an invalid data is keyed in, the system immediately prompts the user and the user has to again key in the data and the system will accept the data only if the data is correct. Validations have been included where necessary.

The system is designed to be a user friendly one. In other words the system has been designed to communicate effectively with the user. The system has been designed with popup menus.

User Interface Design

It is essential to consult the system users and discuss their needs while designing the user interface:

User Interface Systems Can Be Broadly Clasified As:

- User initiated interface the user is in charge, controlling the progress of the user/computer dialogue. In the computer-initiated interface, the computer selects the next stage in the interaction.
- Computer initiated interfaces

In the computer-initiated interfaces the computer guides the progress of the user/computer dialogue. Information is displayed and the user response of the computer takes action or displays further information.

User Initiated Interfaces

User initiated interfaces fall into two approximate classes:

- Command driven interfaces: In this type of interface the user inputs commands or queries which are interpreted by the computer.
- Forms oriented interface: The user calls up an image of the form to his/her screen and fills in the form. The forms-oriented interface is chosen because it is the best choice.

Computer-Initiated Interfaces

The following computer – initiated interfaces were used:

- The menu system for the user is presented with a list of alternatives and the user chooses one; of alternatives.
- Questions – answer type dialog system where the computer asks question and takes action based on the basis of the users reply.

Right from the start the system is going to be menu driven, the opening menu displays the available options. Choosing one option gives another popup menu with more options. In this way every option leads the users to data entry form where the user can key in the data.

Error Message Design

The design of error messages is an important part of the user interface design. As user is bound to commit some errors or other while designing a system the system should be designed to be helpful by providing the user with information regarding the error he/she has committed.

This application must be able to produce output at different modules for different inputs.

Performance Requirements

Performance is measured in terms of the output provided by the application. Requirement specification plays an important part in the analysis of a system. Only when the requirement specifications are properly given, it is possible to design a system, which will fit into required environment. It rests largely in the part of the users of the existing system to give the requirement specifications because they are the people who finally use the system. This is because the requirements have to be known during the initial stages so that the system can be designed according to those requirements. It is very difficult to change the system once it has been designed and on the other hand designing a system, which does not cater to the requirements of the user, is of no use.

The requirement specification for any system can be broadly stated as given below:

- The system should be able to interface with the existing system
- The system should be accurate
- The system should be better than the existing system
- The existing system is completely dependent on the user to perform all the duties.

CHAPTER 9

SOURCE CODE

```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, 
confusion_matrix, classification_report

import os

from sklearn.linear_model import RidgeClassifier

from catboost import CatBoostClassifier


# Importing dataset

df = pd.read_csv(r'Dataset\dataset.csv')


# Data analysis

df.info()

df.describe()

df.corr()

df.isnull().sum()

df['target'].unique()
```

```
# Data Visualization

sns.set(style="darkgrid")

plt.figure(figsize=(12, 6))

ax = sns.countplot(x=df['target'], data=df, palette="Set3")

plt.title("Count Plot")

plt.xlabel("Categories")

plt.ylabel("Count")

labels = ['No Fault','Fault']

ax.set_xticklabels(labels)


for p in ax.patches:

    ax.annotate(f'{p.get_height()}', (p.get_x() + p.get_width() / 2., p.get_height()),

                ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),

                textcoords='offset points')


plt.show()


# Declaring independent and dependent variables

x = df.drop(['target','profile_id'],axis = 1)

y = df['target']


# Train-test split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30, random_state=42)
```

```
# Performance evaluation
precision = []
recall = []
fscore = []
accuracy = []

def performance_metrics(algorithm, predict, testY):
    testY = testY.astype('int')
    predict = predict.astype('int')
    p = precision_score(testY, predict, average='macro') * 100
    r = recall_score(testY, predict, average='macro') * 100
    f = f1_score(testY, predict, average='macro') * 100
    a = accuracy_score(testY, predict) * 100
    accuracy.append(a)
    precision.append(p)
    recall.append(r)
    fscore.append(f)
    print(algorithm + ' Accuracy: ' + str(a))
    print(algorithm + ' Precision: ' + str(p))
    print(algorithm + ' Recall: ' + str(r))
    print(algorithm + ' FSCORE: ' + str(f))
    report = classification_report(predict, testY, target_names=labels)
    print('\n' + algorithm + " classification report\n", report)
    conf_matrix = confusion_matrix(testY, predict)
    plt.figure(figsize=(5, 5))
    ax = sns.heatmap(conf_matrix, xticklabels=labels, yticklabels=labels, annot=True,
cmap="Blues", fmt="g")
    ax.set_ylim([0, len(labels)])
    plt.title(algorithm + " Confusion matrix")
    plt.ylabel('True class')
    plt.xlabel('Predicted class')
    plt.show()


# Ridge Classifier model building
ridge_model_path = 'model/RidgeClassifier.npy'
if os.path.exists(ridge_model_path):
    ridge_classifier = np.load(ridge_model_path, allow_pickle=True).item()
else:
    ridge_classifier = RidgeClassifier()
    ridge_classifier.fit(x_train, y_train)
    np.save(ridge_model_path, ridge_classifier)


y_pred_ridge = ridge_classifier.predict(x_test)
performance_metrics('RidgeClassifier', y_pred_ridge, y_test)


# CatBoost Classifier model building
catboost_model_path = 'model/CatBoostClassifier.npy'
if os.path.exists(catboost_model_path):
    catboost_classifier = CatBoostClassifier().load_model(catboost_model_path)
else:
    catboost_classifier = CatBoostClassifier()
    catboost_classifier.fit(x_train, y_train, verbose=False)
    catboost_classifier.save_model(catboost_model_path)

y_pred_catboost = catboost_classifier.predict(x_test)

catboost_accuracy = accuracy_score(y_test, y_pred_catboost)
print("CatBoost Classifier Accuracy:", catboost_accuracy)
performance_metrics('CatBoost Classifier', y_pred_catboost, y_test)

# Tabular form of Performance Metrics
columns = ["Algorithm Name", "Precison", "Recall", "FScore", "Accuracy"]
values = []
algorithm_names = ["Ridge Classifier", "CatBoost Classifier"]
for i in range(len(algorithm_names)):
    values.append([algorithm_names[i], precision[i], recall[i], fscore[i], accuracy[i]])
temp = pd.DataFrame(values, columns=columns)
temp

# Uploading testing dataset
test = pd.read_csv("Dataset/test.csv")
```

```
# Model prediction on test data

predict = catboost_classifier.predict(test)

for i, p in enumerate(predict):

    if p == 0:

        print(test.iloc[i])

        print("Model Predicted of Row {} Test Data is --->".format(i), labels[0])

    elif p == 1:

        print(test.iloc[i])

        print("Model Predicted of Row {} Test Data is --->".format(i), labels[1])
```

CHAPTER 10

RESULTS AND DESCRIPTION

10.1 Implementation Description

The project involves developing a fault detection system for Permanent Magnet Synchronous Machines (PMSMs) using machine learning models, specifically Ridge Classifier and CatBoost Classifier. The goal is to improve the accuracy and reliability of detecting faults in PMSMs to ensure uninterrupted operations and proactive maintenance.

First, the necessary libraries are imported, including **numpy**, **pandas**, **matplotlib**, **seaborn**, and several from **sklearn** for data processing and model building. The dataset is loaded from a CSV file into a pandas DataFrame. Initial data analysis is performed using methods like **info()**, **describe()**, and **corr()** to understand the dataset's structure and characteristics. Missing values are checked with **isnull().sum()**, and the unique values of the target variable are identified.

For data visualization, seaborn is used to create a count plot of the target variable, showing the distribution of fault and no-fault cases. This helps in understanding the class imbalance in the dataset, if any. The dataset is then split into independent variables (features) and the dependent variable (target). The features are all columns except 'target' and 'profile_id', while the target variable is the 'target' column.

The dataset is divided into training and testing sets using **train_test_split** with a 70-30 split. This ensures that 70% of the data is used for training the model, and 30% is used for testing its performance.

A function **performance_metrics** is defined to evaluate the models' performance. It calculates precision, recall, F1 score, and accuracy, and prints these metrics along with a classification report and a confusion matrix. This function is crucial for comparing the effectiveness of different models.

The Ridge Classifier model is then built. If a pre-trained model exists, it is loaded; otherwise, a new Ridge Classifier is trained on the training data and saved for future use. Predictions are made on the test data, and the performance of the Ridge Classifier is evaluated using the **performance_metrics** function.

Similarly, the CatBoost Classifier model is built. If a pre-trained model exists, it is loaded; otherwise, a new CatBoost Classifier is trained on the training data and saved. Predictions are made on the test data, and the model's performance is evaluated.

The performance metrics for both models are stored and displayed in a tabular format using pandas DataFrame. This comparison helps in identifying the better performing model in terms of precision, recall, F1 score, and accuracy.

Finally, the model is tested on a separate test dataset. Predictions are made using the CatBoost Classifier, and the results are printed, indicating whether each row in the test dataset is predicted as a fault or no-fault case.

10.2 Dataset Description

Here's a description of the features present in the dataset:

- **u_q**: This represents the quadrature-axis voltage component in the motor's control system. It is an important parameter in field-oriented control (FOC) of electric motors, impacting torque production.
- **coolant**: This feature measures the temperature or flow rate of the coolant used to maintain the motor's temperature. Effective cooling is crucial to prevent overheating and ensure optimal motor performance.
- **stator_winding**: This represents the temperature of the stator winding. Stator winding temperature is critical for diagnosing insulation issues and preventing damage due to excessive heat.
- **u_d**: Similar to u_q, this represents the direct-axis voltage component. Together, u_d and u_q are used to control the motor's magnetic field and torque production.
- **stator_tooth**: This measures the temperature of the stator tooth. The stator tooth temperature can indicate overheating or hot spots that may lead to motor failure.

— **motor_speed**: This feature records the rotational speed of the motor, typically measured in revolutions per minute (RPM). Motor speed is a fundamental operational parameter affecting performance and efficiency.

— **i_d**: This represents the direct-axis current component. In FOC, i_d is used to control the magnetizing current of the motor, impacting the efficiency and stability of the motor operation.

— **i_q**: This represents the quadrature-axis current component, which directly affects the torque produced by the motor.

— **pm**: stands for permanent magnet temperature. This is crucial in motors with permanent magnets, as excessive heat can demagnetize the magnets, reducing efficiency and torque.

— **stator_yoke**: This measures the temperature of the stator yoke. The stator yoke temperature helps in monitoring the overall thermal state of the motor.

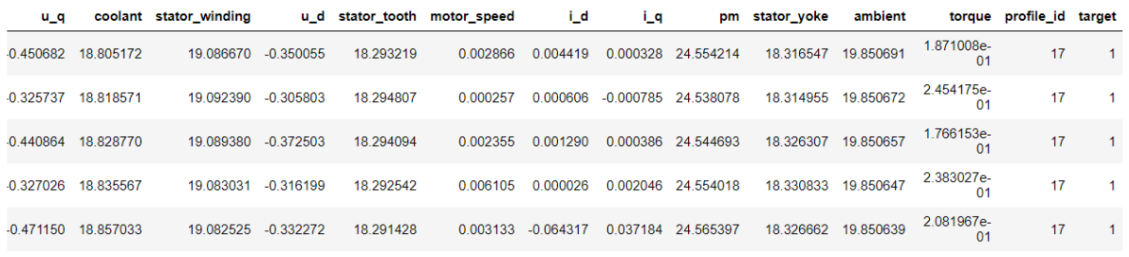
— **ambient**: This feature records the ambient temperature around the motor. Ambient temperature affects the motor's cooling efficiency and overall thermal management.

— **torque**: This measures the torque produced by the motor, an essential performance metric indicating the motor's ability to perform work.

— **profile_id**: This is an identifier for different operational profiles or test conditions. Each profile ID might correspond to a specific test scenario or configuration.

— **target**: This is the target variable for the machine learning model, indicating whether a fault is present (binary classification). Typically, a value of 0 might indicate 'No Fault' and a value of 1 might indicate 'Fault'.

10.3 Results Description



u_q	coolant	stator_winding	u_d	stator_tooth	motor_speed	i_d	i_q	pm	stator_yoke	ambient	torque	profile_id	target
-0.450682	18.805172	19.086670	-0.350055	18.293219	0.002866	0.004419	0.000328	24.554214	18.316547	19.850691	1.871008e-01	17	1
-0.325737	18.818571	19.092390	-0.305803	18.294807	0.000257	0.000606	-0.000785	24.538078	18.314955	19.850672	2.454175e-01	17	1
-0.440864	18.828770	19.089380	-0.372503	18.294094	0.002355	0.001290	0.000386	24.544693	18.326307	19.850657	1.766153e-01	17	1
-0.327026	18.835567	19.083031	-0.316199	18.292542	0.006105	0.000026	0.002046	24.554018	18.330833	19.850647	2.383027e-01	17	1
-0.471150	18.857033	19.082525	-0.332272	18.291428	0.003133	-0.064317	0.037184	24.565397	18.326662	19.850639	2.081967e-01	17	1

Figure 1: Presents the Sample dataset of the Stator false dataset.

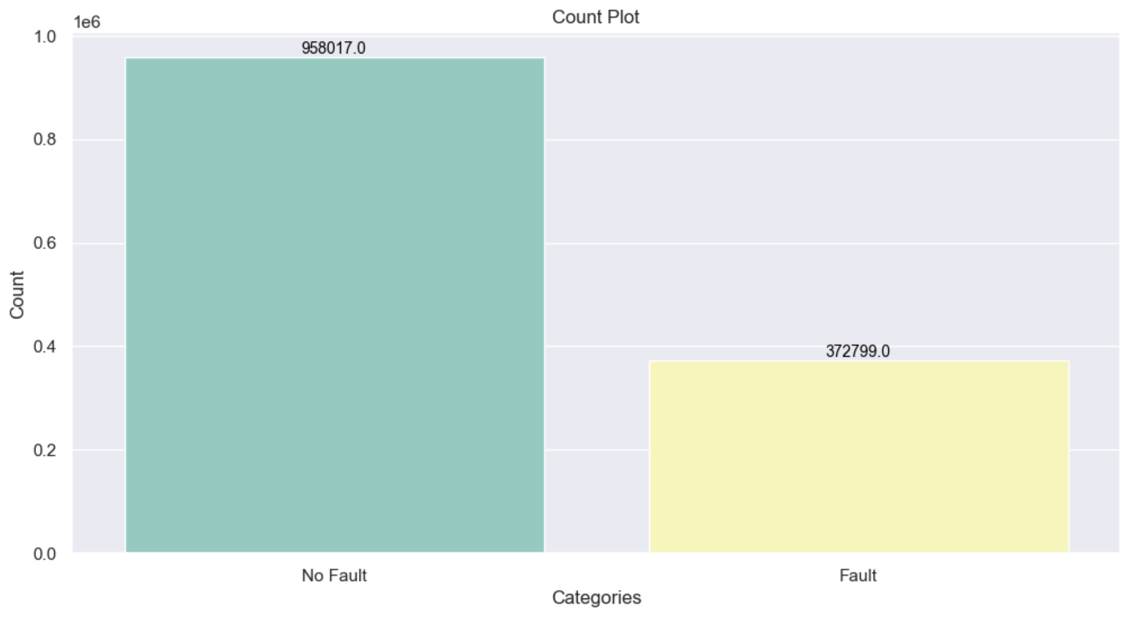
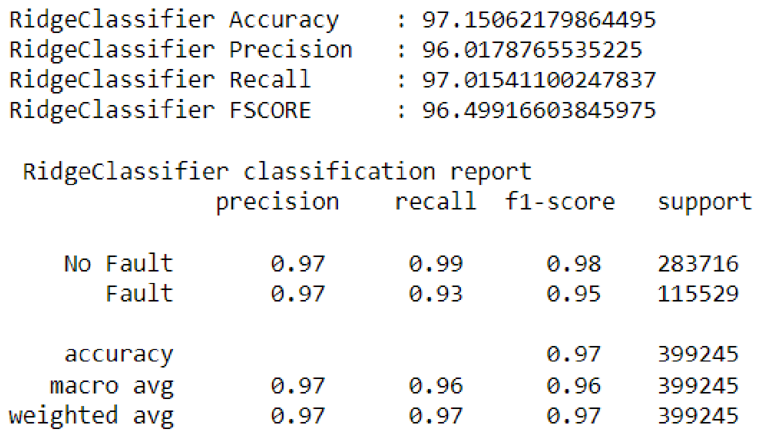


Fig. 2: Presents the count plot of Stator false dataset.



```
RidgeClassifier Accuracy    : 97.15062179864495
RidgeClassifier Precision   : 96.0178765535225
RidgeClassifier Recall      : 97.01541100247837
RidgeClassifier FSCORE      : 96.49916603845975

 RidgeClassifier classification report
              precision    recall  f1-score   support

    No Fault       0.97      0.99      0.98    283716
       Fault       0.97      0.93      0.95    115529

    accuracy                           0.97    399245
   macro avg       0.97      0.96      0.96    399245
weighted avg       0.97      0.97      0.97    399245
```

Fig 3: Shows a classification report of a Ridge Classifier model.

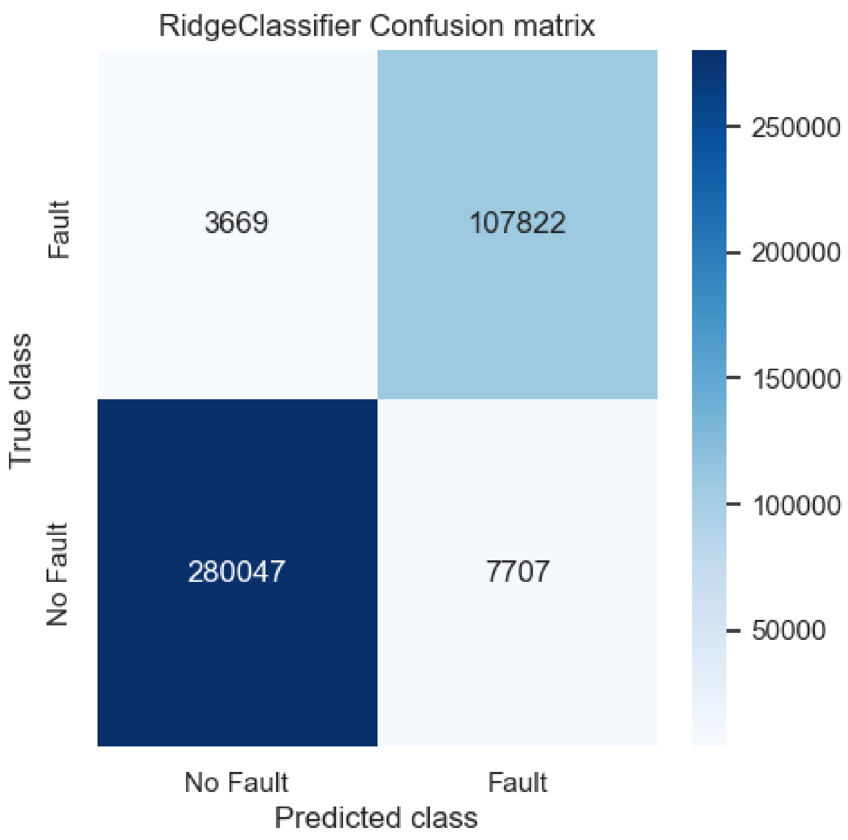
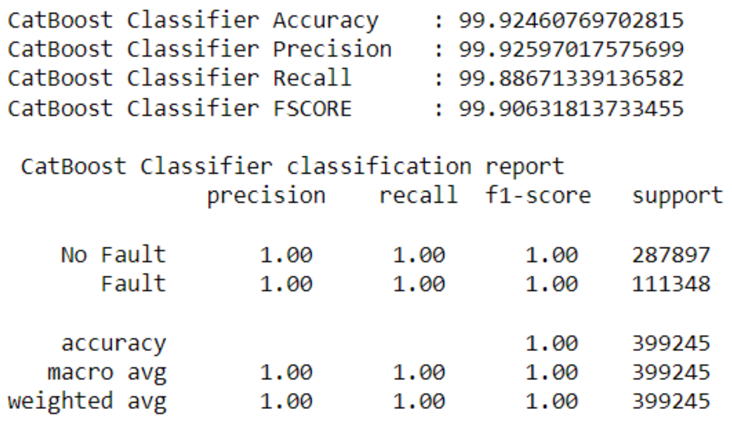


Fig 4: Confusion matrix of Ridge Classifier model.



```
CatBoost Classifier Accuracy     : 99.92460769702815
CatBoost Classifier Precision    : 99.92597017575699
CatBoost Classifier Recall       : 99.88671339136582
CatBoost Classifier FSCORE       : 99.90631813733455

 CatBoost Classifier classification report
              precision    recall  f1-score   support

    No Fault       1.00      1.00      1.00    287897
       Fault       1.00      1.00      1.00    111348

    accuracy                           1.00    399245
   macro avg       1.00      1.00      1.00    399245
weighted avg       1.00      1.00      1.00    399245
```

Fig 5: Shows a classification report of a CatBoost Classifier model.

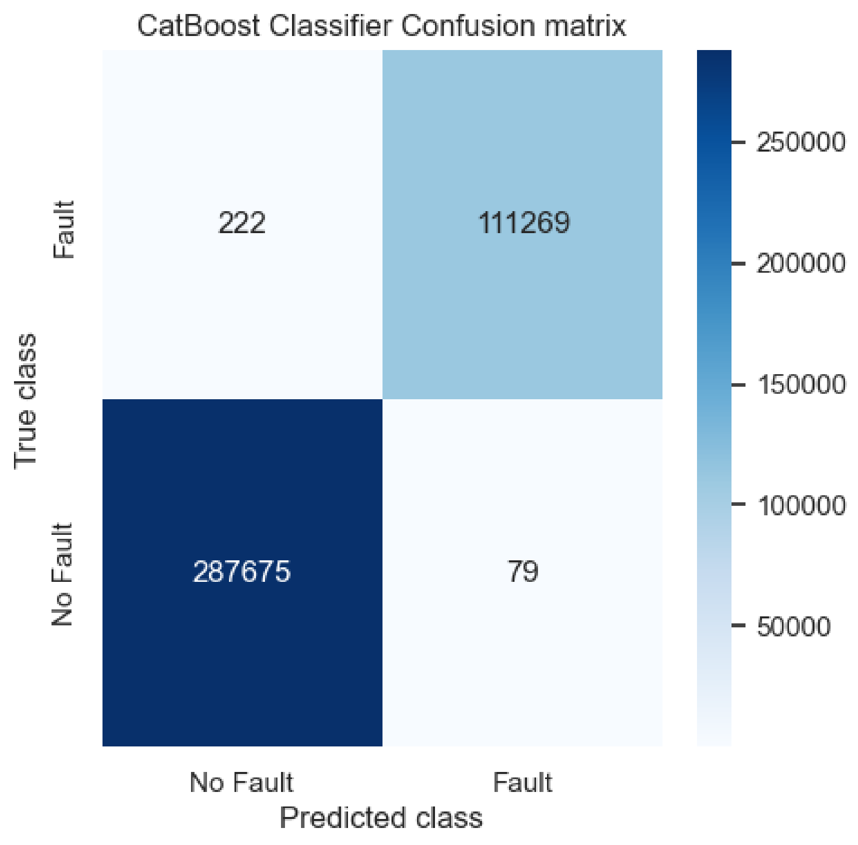
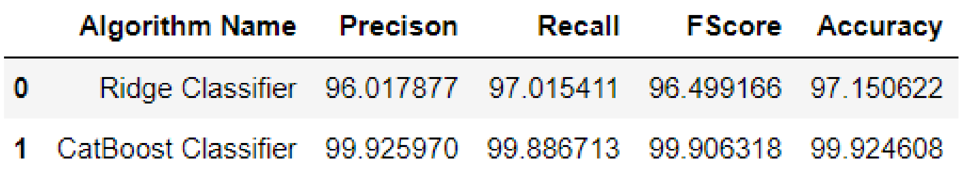


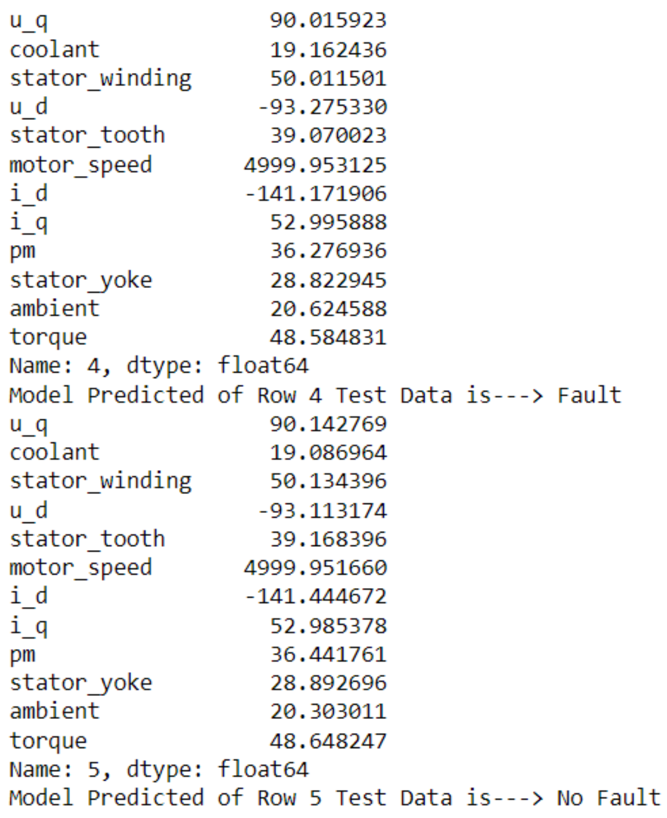
Fig 6: Confusion matrix of CatBoost Classifier model.

The figure 4 confusion matrix of the Ridge Classifier model visually represents the performance of the model in classifying different categories of mouth diseases. It provides a clear overview of the true positive, true negative, false positive, and false negative predictions made by the model for each class. The figure 5 classification report of the CatBoost Classifier model presents a detailed summary of the model's performance in terms of precision, recall, F1-score, and support for each class. It offers insights into the model's ability to correctly classify instances of each disease category. The figure 6 confusion matrix of the CatBoost Classifier model illustrates the model's performance but specifically for this classifier. It provides a visual representation of how well the model predicts the actual classes of Fitness activities, aiding in understanding its activities.



	Algorithm Name	Precison	Recall	FScore	Accuracy
0	Ridge Classifier	96.017877	97.015411	96.499166	97.150622
1	CatBoost Classifier	99.925970	99.886713	99.906318	99.924608

Fig 7: Presents the Comparison table of performance metrics



```
u_q                 90.015923
coolant             19.162436
stator_winding      50.011501
u_d                -93.275330
stator_tooth        39.070023
motor_speed       4999.953125
i_d               -141.171906
i_q                 52.995888
pm                  36.276936
stator_yoke         28.822945
ambient             20.624588
torque              48.584831
Name: 4, dtype: float64
Model Predicted of Row 4 Test Data is---> Fault
u_q                 90.142769
coolant             19.086964
stator_winding      50.134396
u_d                -93.113174
stator_tooth        39.168396
motor_speed       4999.951660
i_d               -141.444672
i_q                 52.985378
pm                  36.441761
stator_yoke         28.892696
ambient             20.303011
torque              48.648247
Name: 5, dtype: float64
Model Predicted of Row 5 Test Data is---> No Fault
```

Fig 8: Proposed CatBoost Classifier model Prediction on test data.

The figure 5 comparison table of performance metrics presents a comprehensive overview of the performance of different classifiers, such as Ridge Classifier and CatBoost Classifier. It allows for a direct comparison of metrics such as accuracy, precision, recall, and F1-score, enabling stakeholders to make informed decisions about model selection.

The figure 6 proposed CatBoost Classifier model's prediction of fault on a test data demonstrates the practical application of the model.

CHAPTER 11

CONCLUSION AND FUTURE SCOPE

The development of advanced stator fault detection strategies for PMSMs is essential for ensuring reliable and uninterrupted operation in industrial applications. By leveraging cutting-edge technologies such as machine learning and sensor fusion, researchers aim to overcome the limitations of traditional maintenance methods and develop proactive fault detection systems capable of accurately identifying stator faults in real-time.

Looking ahead, future research in this field may focus on further improving the accuracy, robustness, and scalability of fault detection algorithms, as well as exploring novel sensor technologies and data analytics techniques. Additionally, integrating fault detection systems with predictive maintenance and condition monitoring platforms can enable more proactive and data-driven maintenance strategies, further enhancing the reliability and efficiency of PMSMs in industrial environments.

REFERENCES

[1] Upadhyay, A.; Alaküla, M.; Márquez-Fernández, F.J. Characterization of Onboard Condition Monitoring Techniques for Stator Insulation Systems in Electric Vehicles—A Review. In Proceedings of the IECON 2019—45th Annual Conference of the IEEE Industrial Electronics Society, Lisbon, Portugal, 14–17 October 2019; Volume 1, pp. 3179–3186.

[2] Xian, R.; Wang, L.; Zhang, B.; Li, J.; Xian, R.; Li, J. Identification Method of Interturn Short Circuit Fault for Distribution Transformer Based on Power Loss Variation. IEEE Trans. Ind. Inform. 2003, 20, 2444–2454.

[3] Faiz, J.; Nejadi-Koti, H.; Valipour, Z. Comprehensive Review on Inter-turn Fault Indexes in Permanent Magnet Motors. IET Electr. Power Appl. 2017, 11, 142–156.

[4] Wang, Z.; Yang, J.; Ye, H.; Zhou, W. A Review of Permanent Magnet Synchronous Motor Fault Diagnosis. In Proceedings of the 2014 IEEE Conference and Expo Transportation Electrification Asia-Pacific (ITEC Asia-Pacific), Beijing, China, 31 August–3 September 2014; pp. 1–5.

[5] Wu, C.; Guo, C.; Xie, Z.; Ni, F.; Liu, H. A Signal-Based Fault Detection and Tolerance Control Method of Current Sensor for PMSM Drive. IEEE Trans. Ind. Electron. 2018, 65, 9646–9657.

[6] Wang, X.; Wang, Z.; Xu, Z.; Cheng, M.; Wang, W.; Hu, Y. Comprehensive Diagnosis and Tolerance Strategies for Electrical Faults and Sensor Faults in Dual Three-Phase PMSM Drives. IEEE Trans. Power Electron. 2018, 34, 6669–6684.

[7] Orlowska-Kowalska, T.; Wolkiewicz, M.; Pietrzak, P.; Skowron, M.; Ewert, P.; Tarchala, G.; Krzysztofiak, M.; Kowalski, C.T. Fault Diagnosis and Fault-Tolerant Control of PMSM Drives–State of the Art and Future Challenges. IEEE Access 2022, 10, 59979–60024.

[8] Akrad, A.; Hilairet, M.; Diallo, D. Design of a Fault-Tolerant Controller Based on Observers for a PMSM Drive. IEEE Trans. Ind. Electron. 2011, 58, 1416–1427.

[9] Dai, X.; Gao, Z. From Model, Signal to Knowledge: A Data-Driven Perspective of Fault Detection and Diagnosis. IEEE Trans. Ind. Inform. 2013, 9, 2226–2238.

[10] Mansouri, B.; Idrissi, H.J.; Venon, A. Inter-Turn Short-Circuit Failure of PMSM Indicator Based on Kalman Filtering in Operational Behavior. In Proceedings of the Annual Conference of the PHM Society, Scottsdale, AZ, USA, 21–26 September 2019; Volume 11.

[11] Namdar, A.; Samet, H.; Allahbakhshi, M.; Tajdinian, M.; Ghanbari, T. A Robust Stator Inter-Turn Fault Detection in Induction Motor Utilizing Kalman Filter-Based Algorithm. Measurement 2022, 187, 110181.

[12] Lee, D.; Park, H.J.; Lee, D.; Lee, S.; Choi, J.-H. A Novel Kalman Filter-Based Prognostics Framework for Performance Degradation of Quadcopter Motors. IEEE Trans. Instrum. Meas. 2023, 73, 1–12.

[13] Chang, C.-C.; Cheng, T.-H. Motor-Efficiency Estimation and Control of Multirotors Comprising a Cooperative Transportation System. IEEE Access. 2023, 11, 36566–36578.

[14] Hasan, A.; Tahavori, M.; Midtiby, H.S. Model-Based Fault Diagnosis Algorithms for Robotic Systems. IEEE Access 2023, 11, 2250–2258.

[15] El Sayed, W.; Abd El Geliel, M.; Lotfy, A. Fault Diagnosis of PMSG Stator Inter-Turn Fault Using Extended Kalman Filter and Unscented Kalman Filter. Energies 2020, 13, 2972.

[16] Mahmoudi, A.; Jlassi, I.; Cardoso, A.J.M.; Yahia, K.; Sahraoui, M. Inter-Turn Short-Circuit Faults Diagnosis in Synchronous Reluctance Machines, Using the Luenberger State Observer and Current's Second-Order Harmonic. IEEE Trans. Ind. Electron. 2021, 69, 8420–8429.

[17] Guezmil, A.; Berriri, H.; Pusca, R.; Sakly, A.; Romary, R.; Mimouni, M.F. Detecting Inter-Turn Short-Circuit Fault in Induction Machine Using High-Order Sliding Mode Observer: Simulation and Experimental Verification. J. Control Autom. Electr. Syst. 2017, 28, 532–540.

[18] Bouakoura, M.; Naït-Saïd, M.-S.; Nait-Said, N. Incipient Inter-Turn Short Circuit Fault Estimation Based on a Faulty Model Observer and Ann-Method for Induction Motor Drives. Recent Adv. Electr. Electron. Eng. Former. Recent Pat. Electr. Electron. Eng. 2019, 12, 374–383.

[19] Vasilios, I.C. Detection of PMSM Inter-Turn Short-Circuit Based on a Fault-Related Disturbance Observer. Int. J. Simul. Syst. Sci. Technol. 2020, 21, 31.1–31.7.