Machine Learning-Based Fault Prediction in Transmission Lines: A Comparative Study of Diverse Models for Enhanced Reliability in Power Networks

(AID-505: Course Project)

Pranay Majumder
pranay_m@mfs.iitr.ac.in
Enrolment No: 23566007

Vicky Kashinath jha
vicky_kj@mfs.iitr.ac.in
Enrolment No: 23566018

Abstract

In this project, we explore and compare the performance of various machine learning models for a given task. The objective is to assess the accuracy, precision, recall, and F1 score of different classifiers to identify the most suitable model for our specific problem. The models under consideration include Logistic Regression, k-Nearest Neighbours (KNN), Support Vector Machines (SVM), Naive Bayes, Decision Trees, Random Forest, Artificial Neural Networks (ANN), and AdaBoost. The evaluation is conducted through extensive experimentation on a labeled dataset, employing metrics that provide insights into the models' predictive capabilities and generalization to unseen data. The findings from this comparative analysis aim to guide the selection of the most effective machine learning model for the given task, contributing valuable insights for practical applications. The increasing demand for a reliable and resilient power grid has led to a growing interest in leveraging machine learning for fault prediction in transmission lines. This research presents a comprehensive comparative study of diverse machine learning models to enhance the reliability of power networks. The outcomes of this research contribute valuable insights into the selection of optimal fault prediction systems, ultimately advancing the robustness and efficiency of power distribution networks.

1. Main Objective

- Predicting what kind of Fault is Encountered in the Transmission Line by Using the different ML Models
- Compare the Result Obtained From different ML Models in order to get optimal Fault Prediction System

2. Introduction

In modern power distribution networks, ensuring the continuous and reliable supply of electricity is paramount. Transmission lines, forming the backbone of these networks, play a crucial role in maintaining the integrity and efficiency of the entire system. However, the inherent vulnerability of these lines to faults poses a significant challenge, as disruptions can lead to widespread outages, economic losses, and potential safety hazards.

To address this challenge, the integration of advanced technologies, particularly machine learning, has emerged as a promising avenue for enhancing fault prediction capabilities in transmission lines. Machine learning models, equipped with the ability to discern intricate patterns from vast datasets, offer a data-driven approach to pre-emptively identify and mitigate potential faults. This research endeavors to delve into the realm of machine learning-based fault prediction specifically tailored for transmission lines, with a focus on providing an in-depth comparative analysis of diverse models.

The primary objective of this study is to meticulously evaluate and compare the performance of various machine learning models in predicting faults within transmission lines. Models under consideration include Logistic Regression, k-Nearest Neighbours (KNN), Support Vector Machines (SVM), Naive Bayes, Decision Trees, Random Forest, Artificial Neural Networks (ANN), and AdaBoost. Each model brings distinct characteristics, strengths, and potential limitations, making them suitable for different scenarios.

Through an extensive comparative study, we aim to assess key performance metrics such as accuracy, precision, recall, and F1 score. These metrics serve as benchmarks to gauge the predictive capabilities of each model, offering valuable insights into their effectiveness and generalization to unforeseen circumstances. The ultimate goal is to identify the most suitable machine learning model for fault prediction in transmission lines, contributing to the enhanced reliability and resilience of power networks.

As the demand for a reliable and resilient power grid continues to escalate, the findings from this research hold significant implications for practical applications. By selecting and implementing an optimal fault prediction system, power utilities can proactively address potential issues, minimize downtime, and fortify the overall efficiency of power distribution networks. This study, therefore, seeks to provide a comprehensive understanding of the comparative performance of diverse machine learning models, laying the foundation for advancements in fault prediction technology and its integration into critical infrastructure systems

 # Features/Column

1. G (Ground)
2. A (Phase A)
3. B (Phase B)
4. C (Phase C)
5. Ia (Current in Phase A)
6. Ib (Current in Phase B)
7. Ic (Current in Phase C)
8. Va (Voltage in Phase A)
9. Vb (Voltage in Phase B)
10. Vc (Voltage in Phase C)

3. Dataset and Pre-processing

The dataset, available on Kaggle, has 7861 samples of data. We have total 10 Features. By observing the values of G, A, B, C we can say what kind of Fault it is. So, on the basis of the values of G, A, B, C we create a New Column, that will give information regarding different faults, Here We have Total Five types of Faults, “Line A to Ground Fault”, “Line B to Line C Fault”, “Line A Line B to Ground Fault”, “Line A Line B Line C Fault”,” Line A Line B Line C to Ground Fault” and along with that we have some samples that represent “No Fault”. So, here we have total six Non numerical value’s that must be convert to categorical value in order to train the ML Models. For Converting it into a categorical value here we use Label Encoding that will assign distinct value to each Fault.

a. Scaling the data

Scaling is a fundamental pre-processing step in machine learning, designed to standardize the range of features within a dataset for various essential reasons. By ensuring that all features share a similar scale, scaling prevents certain features from dominating the learning process, particularly in algorithms sensitive to the magnitude of input variables, such as k-Nearest Neighbours and clustering algorithms. It facilitates quicker and more efficient convergence in optimization algorithms, promotes interpretability in linear models by making coefficients comparable, and enhances the overall performance of distance-based algorithms like Support Vector Machines. Additionally, scaling mitigates sensitivity to initial conditions, fosters robustness, and ensures uniform application of regularization techniques across features, contributing to the stability and effectiveness of machine learning models.

For our Model also we use Standard Scaler in Case of Logistic Regression, K Nearest Neighbours and SVM in order to improve the Accuracy. For KNN it is important to use scaling in order to eliminate the dominance of large distances.

b. Visualising the data

Here we have total Five types of fault, so we plot voltage and current waveform for each Fault. Here we show both Faulted system (LG Fault) and No fault Situation (Healthy System) in order to get better Visualization.

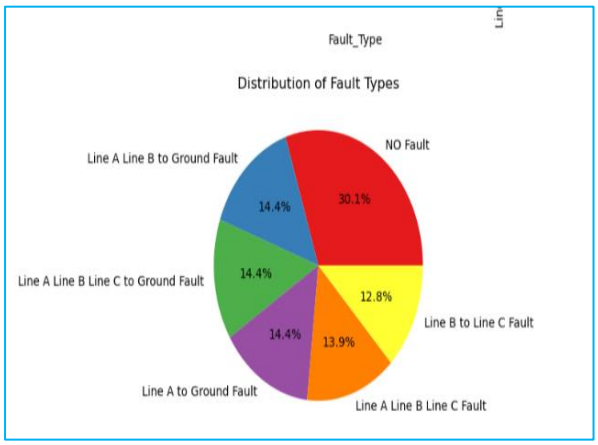
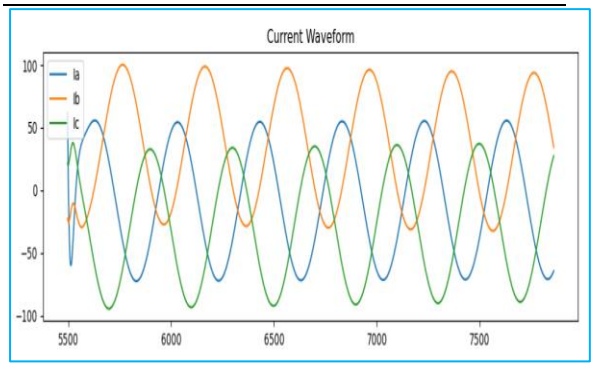


Figure 1: Distribution of Fault

System Without Fault



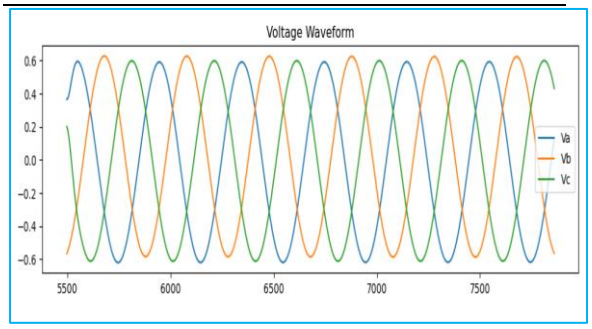
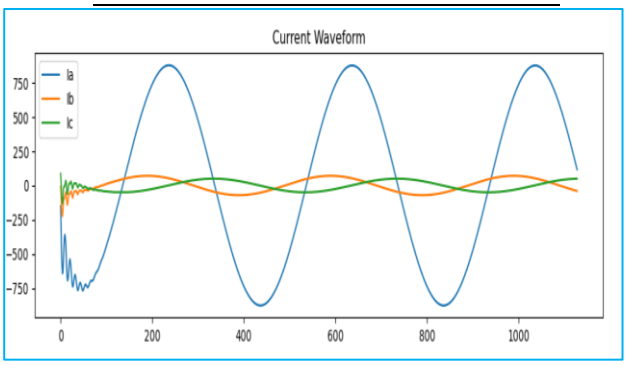


Figure2: Voltage and Current Waveform of Healthy System

From the below Current Graph we can observe that during the Fault the current B and C phases becomes Zero, because During LG Fault there is Short Circuit happen between Line A and Ground. And at that other Two phases(B,C) are remain open, That's why current in other two phases are Zero.

System With Fault (LG Fault)



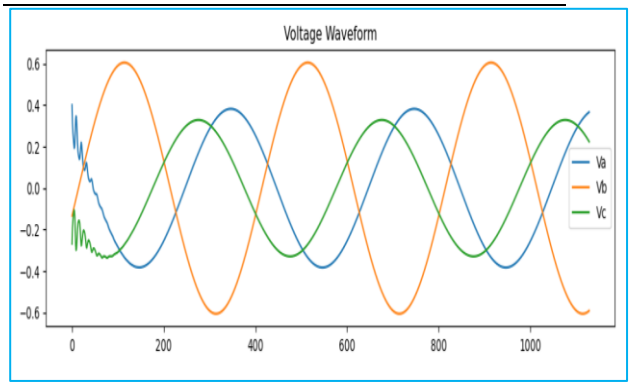


Figure 3: Voltage and Current waveform of Faulty System

c. Split into train, validation, and test set

Here we split our data set into 80% and 20%. Training set contain total 6288 samples and Testing set Contains 1573 samples, here we also use **random state=30** in order to control the randomness of the data splitting process. When you set **random state** to a specific value, you ensure that the random splitting is reproducible. This means that if you run the code with the same **random state** value again, you will get the exact same split of data into training and testing sets.

4. Logistic Regression

Logistic Regression is a fundamental and widely used classification algorithm in the field of machine learning. Despite its name, it is primarily employed for binary classification problems, predicting the probability that an instance belongs to a particular class. Logistic Regression is especially valuable when the dependent variable is categorical, and the relationship between the features and the probability of a certain outcome is sought. At the core of Logistic Regression is the sigmoid function (or logistic function). The sigmoid function maps any real-valued number to a value between 0 and 1. The equation is

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

where z is a linear combination of input features and their corresponding weights.

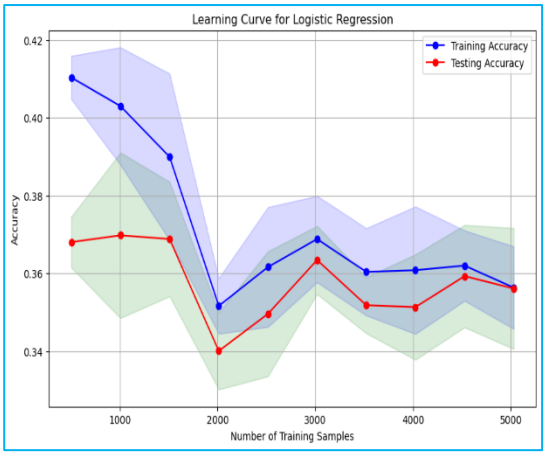


Figure 4: Learning Curve for Logistic Regression

Here as the Number of Samples are increasing the Training Accuracy decreases and Test Accuracy is Increases. Here our aim is Always to make Test Accuracy near to Training Accuracy. So here we can observe both Training and test accuracy meets at a Accuracy of around 34%, Which indicate the Actual Accuracy of the Model.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	223
1	0.27	0.17	0.21	213
2	1.00	0.15	0.26	219
3	0.00	0.00	0.00	227
4	0.00	0.00	0.00	223
5	0.33	1.00	0.50	468
accuracy			0.34	1573
macro avg	0.27	0.22	0.16	1573
weighted avg	0.28	0.34	0.21	1573

Figure 5: Classification Report of Logistic Regression

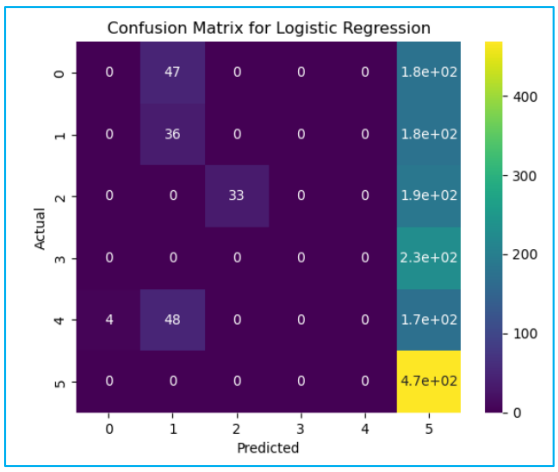


Figure 6: Confusion Matrix for Logistic Regression Model

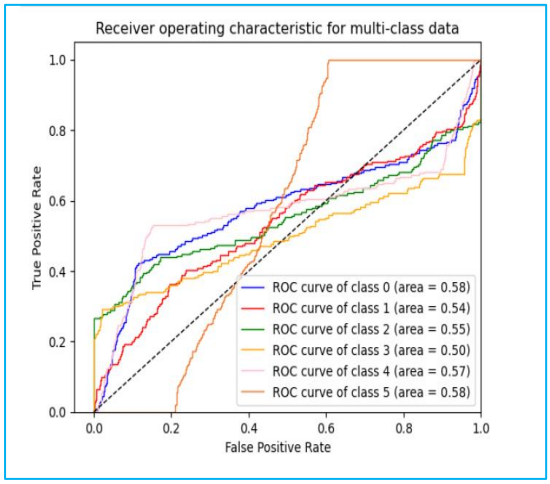


Figure 7: ROC for Logistic Regression

Here the Accuracy we obtain is around 34% which is bit less so, in that case one can think to use standard scaler but we don't get significant change in the Accuracy even after performing the scaling, So Logistic Regression Model not work good for this dataset, so we go for other ML Model's in subsequent slides because here our aim to suggest optimal ML algorithm for Fault Classification.

Key Points About Logistic Regression

- Logistic Regression is primarily designed for binary classification problems. Logistic Regression cannot be used in multi-class problems, that's why here we got very less Accuracy that is around 34%.
- The logistic function (sigmoid function) is fundamental to Logistic Regression It transforms the linear combination of input features and weights into values between 0 and 1, representing probabilities.
- Logistic Regression creates a linear decision boundary in the feature space. Instances on one side of the boundary are predicted as one class, while those on the other side are predicted as the other class.
- Logistic Regression can be less effective when dealing with imbalanced datasets where one class is significantly underrepresented. It may require additional techniques such as resampling or adjusting class weights.
- If there are too many features, Logistic Regression can overfit.

5. K-Nearest Neighbours (KNN)

K-Nearest Neighbours (KNN) is a versatile and straightforward machine learning algorithm used for both classification and regression tasks. It is part of the category of instance-based or lazy learning algorithms, meaning it doesn't explicitly learn a model during training but instead memorizes the training instances. The prediction for a new instance is based on the majority class (for classification) or the average value (for regression) of its k-nearest neighbours in the feature space. KNN relies on a distance metric (usually Euclidean distance) to measure the similarity between instances in the feature space. The smaller the distance, the more similar the instances are considered.

The parameter 'K' represents the number of nearest neighbours to consider when making predictions. It is a critical hyperparameter that significantly influences the algorithm's performance. A smaller 'K' value leads to more flexible models but may be sensitive to noise,

while a larger 'K' value results in smoother decision boundaries but may overlook local patterns.

KNN can be sensitive to the curse of dimensionality, where the effectiveness of the algorithm diminishes as the number of features increases. High-dimensional spaces make it challenging to identify meaningful nearest neighbours.

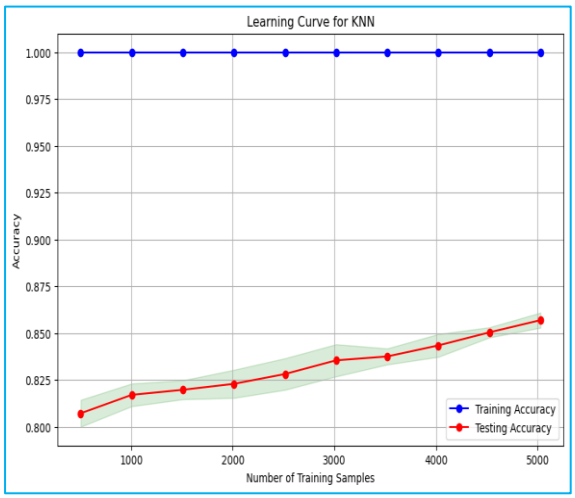


Figure 8: Learning Curve for KNN

Here Training Accuracy is Always maintained at 100%. And as number of samples are increases the Test Accuracy are Also increases. Here the Test accuracy that we obtained is around 89%.

	precision	recall	f1-score	support
0	0.63	0.67	0.65	223
1	0.63	0.59	0.61	213
2	1.00	0.97	0.98	219
3	0.97	1.00	0.98	227
4	1.00	1.00	1.00	223
5	1.00	1.00	1.00	468
accuracy			0.89	1573
macro avg	0.87	0.87	0.87	1573
weighted avg	0.89	0.89	0.89	1573

Figure 9: Classification Report for KNN

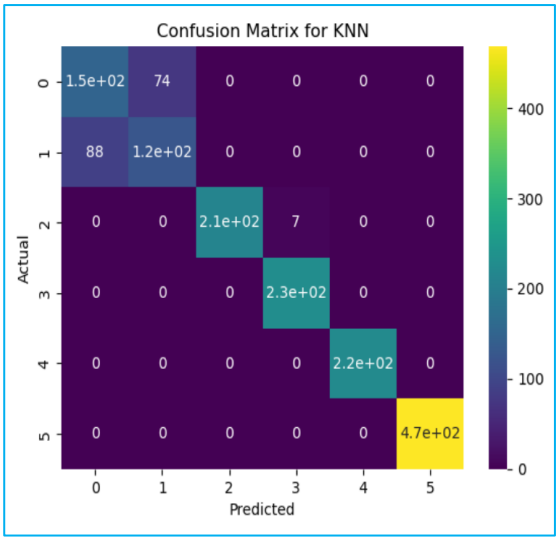


Figure 10: Confusion Matrix for KNN

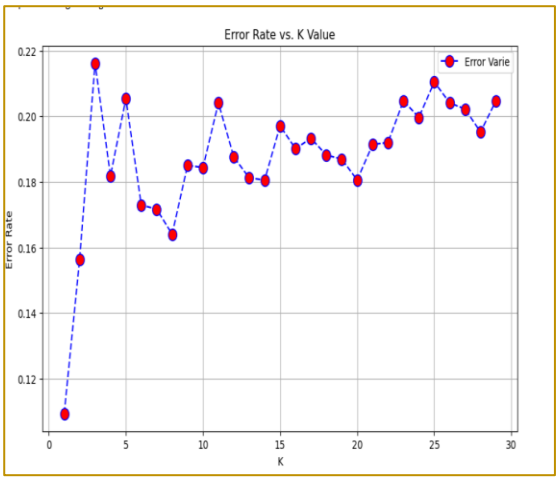


Figure 11: Error Rate vs K

Here we can observe as “K” value is increases error is increases so we minimum error when K=1. And Minimum error indicates Maximum Accuracy, so if We set K=1 here We will get max Accuracy. Here The Accuracy That we obtain is around 89% (Test Accuracy) which is much better than Logistic Regression model.

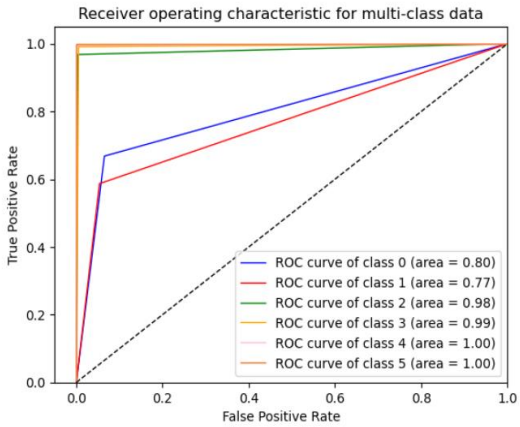


Figure 12: ROC for KNN

Key Points About KNN

- Simple and intuitive algorithm.
- Effective for both classification and regression tasks.
- Easily adaptable to new data.
- Computationally expensive, especially with large datasets.
- Prone to the curse of dimensionality in high-dimensional spaces.
- Lack of a systematic approach for handling imbalanced datasets.

6. Support Vector Machine

Support Vector Machines (SVM) are powerful supervised machine learning algorithms used for both classification and regression tasks. SVMs aim to find the optimal hyperplane that separates data points of different classes in a high-dimensional space. The central idea is to identify the decision boundary (hyperplane) that maximally separates the classes while considering the margin, which is the distance between the hyperplane and the nearest data points (support vectors). In SVM, a hyperplane is a decision boundary that separates data points of different classes. For a binary classification task, the hyperplane is a (d-1)-dimensional subspace in a d-dimensional feature space. Support vectors are the data points that lie closest to the decision boundary. These points are crucial in determining the optimal hyperplane and defining the margin. SVM focuses on maximizing the margin between support vectors.

SVMs can handle non-linear decision boundaries by mapping the input features into a higher-dimensional space using a kernel function. Common kernel functions include linear, polynomial, radial basis function (RBF or Gaussian), sigmoid and polynomial. In cases where the data is not perfectly separable, a soft margin SVM allows for some misclassifications, introducing a penalty term for errors. This enhances the model's ability to generalize to new data.

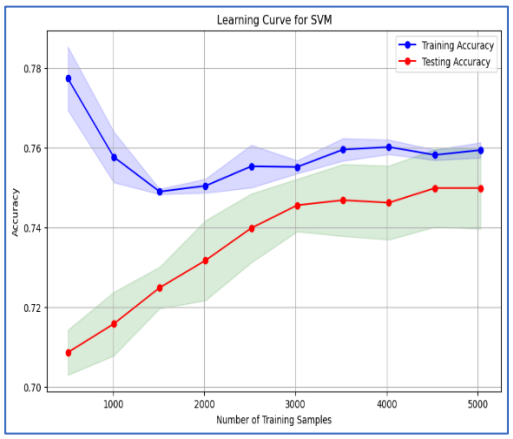


Figure 13: Learning Curve for SVM

Here Also we can see the Test Accuracy is increases as Number of samples are Increases and it is try to catch the Train Accuracy. Because Here our Aim is to obtain Test Accuracy Similar to the Train Accuracy which indicates Low Variance and Low bias.

	precision	recall	f1-score	support
0	0.50	0.26	0.35	223
1	0.45	0.42	0.44	213
2	0.72	0.80	0.76	219
3	0.80	0.85	0.83	227
4	0.84	0.89	0.86	223
5	0.88	1.00	0.94	468
accuracy			0.75	1573
macro avg	0.70	0.70	0.69	1573
weighted avg	0.73	0.75	0.73	1573

Figure 14: Classification Report for SVM

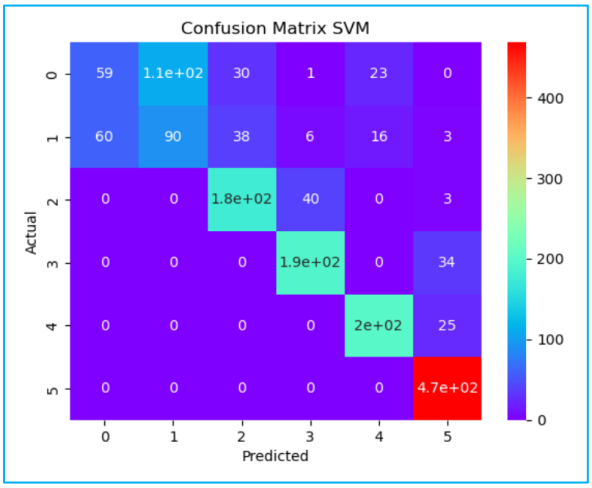


Figure 15: Confusion Matrix for SVM

Hyper Parameter Tunning

We basically perform Hyper Parameter tunning in order to improve the Model Accuracy. Here We consider kernel as "rbf". And we perform Grid Search for different values of C (Regularization Parameter) and gamma. The parameter **gamma** is a crucial hyperparameter that defines the influence of a single training example. The choice of the gamma parameter can significantly impact the performance of the SVM model, especially when using non-linear kernels like the Radial Basis Function (RBF) or Gaussian kernel.

$$K(x_i, x_j) = \exp\left(-\gamma \cdot \|x_i - x_j\|^2\right)$$

After Hyper Parmeter Tunning we set C=100 (Regularization Parameter), Kernel="rbf" and gamma=0.001 which increase the Accuracy of the Model from 75% to 94%, shows in **Figure 17.**

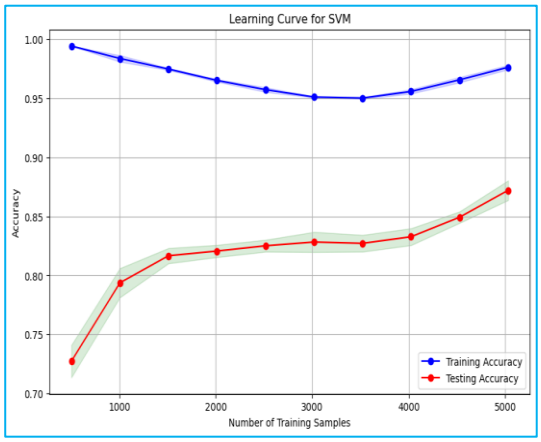


Figure 16: Learning Curve for SVM after Hyper Parameter Tunning

Here we can observe that after Performing Hyper Parameter Tunning both Training and Test Accuracy are Drastically Increases

	precision	recall	f1-score	support
0	0.79	0.83	0.81	223
1	0.82	0.78	0.80	213
2	1.00	0.99	0.99	219
3	1.00	0.99	1.00	227
4	0.99	1.00	1.00	223
5	1.00	1.00	1.00	468
accuracy			0.94	1573
macro avg	0.93	0.93	0.93	1573
weighted avg	0.94	0.94	0.94	1573

Figure 17: Classification Report for SVM after Hyper Parameter Tunning

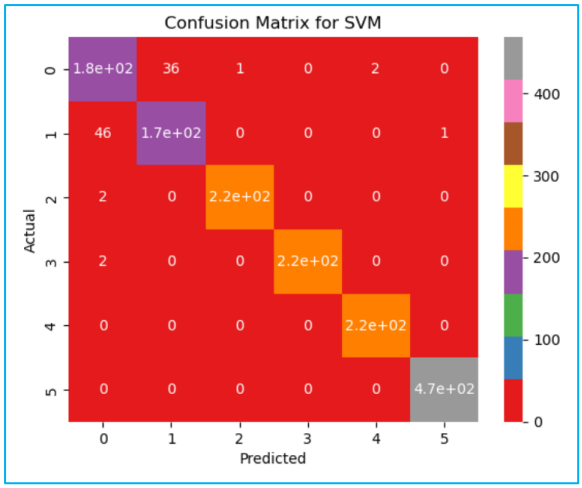


Figure 18: Confusion Matrix for SVM After Performing Hyper Parameter Tunning

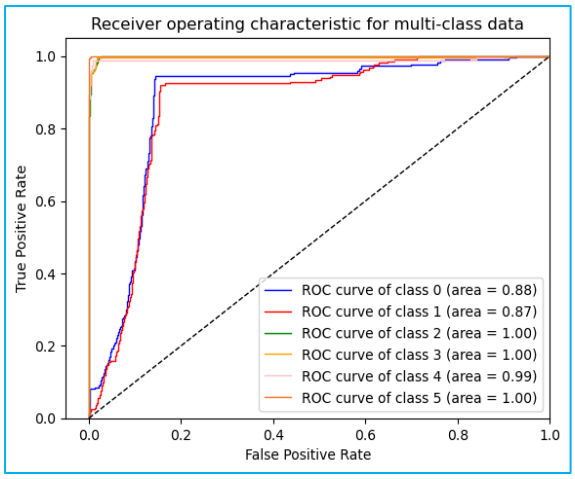


Figure 19: ROC for SVM After Performing Hyper Parameter Tunning

Key Points for SVM

- SVMs perform well in high-dimensional spaces, making them suitable for tasks with many features.
- SVMs are less prone to overfitting, especially in high-dimensional spaces. The margin concept helps prevent the model from fitting noise in the data.
- SVMs can handle both linear and non-linear classification tasks through the use of different kernel functions.
- SVMs can be computationally intensive, particularly with large datasets. Training time increases significantly as the dataset size grows.
- The choice of an appropriate kernel function is crucial. The performance of the SVM is influenced by the selected kernel, and the optimal kernel may vary for different datasets.
- The C parameter in SVM controls the trade-off between achieving a smooth decision boundary and classifying training points correctly. A small C value allows for a larger margin but may lead to misclassifications, while a large C value enforces correct classification but may result in a smaller margin.

7. Decision Tree

A Decision Tree is a versatile and interpretable supervised machine learning algorithm used for both classification and regression tasks. It represents a flowchart-like structure where each internal node denotes a decision based on a specific feature, each branch represents an outcome of that decision, and each leaf node represents the final prediction or decision. Decision Trees are particularly popular due to their simplicity, transparency, and ability to capture complex relationships in the data.

A Decision Tree consists of nodes. Each node in the tree represents a decision or a test on a specific feature. The topmost node in the tree is called the root node. It represents the initial decision or test that is applied to the input data. The process of dividing a node into two or more child nodes based on a specific feature and a threshold is called splitting. The goal is to increase the purity or homogeneity of the data at each node. The decision criteria at each internal node depend on

the task (classification or regression). For classification, common criteria include Gini impurity and entropy, while for regression, mean squared error is often used.

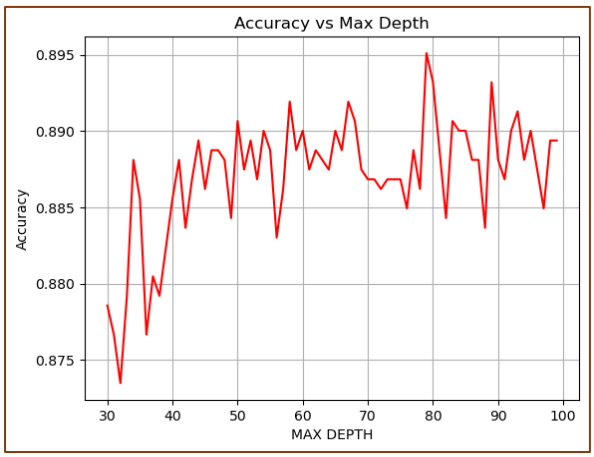


Figure 20: Plot between Accuracy Vs Max Depth

From the above Figure it can be observed that as MAX DEPTH of the Decision Tree is increases its Accuracy is Also increases. Max Depth are basically a Hyper Parameter that can be use-full during Hyper Parameter Tunning. By selecting proper value of Max depth we can maximize the accuracy. Apart from **Max Depth** there are other Hyper parameters also like **min_samples_leaf, max_features, splitter** etc. We can obtain proper values of this Hyper parameter by using Grid Search that may further improve the Accuracy. One thing we have to remember Hyper parameter tunning Not always ensure improvement of Accuracy.

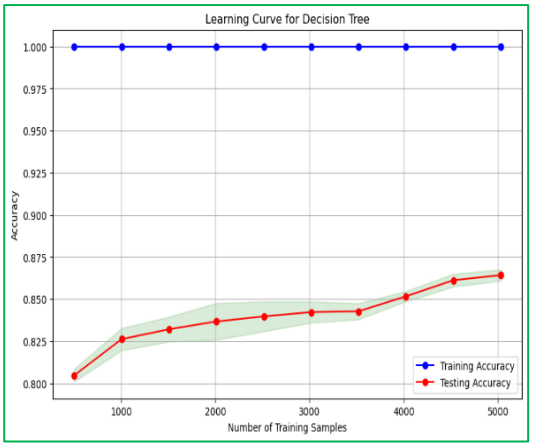


Figure 21: Learning Curve for Decision Tree

From above figure we can observe that the Decision Tree perfectly match the Training set that's why training Accuracy is maintained at 100%. And as Number of Samples Are increases Test accuracy are also increase. Here The Accuracy that we obtain is around 89%. So, for this dataset Decision Tree performs well.

	precision	recall	f1-score	support
0	0.63	0.66	0.65	223
1	0.62	0.60	0.61	213
2	0.99	0.99	0.99	219
3	1.00	0.99	0.99	227
4	1.00	1.00	1.00	223
5	1.00	1.00	1.00	468
accuracy			0.89	1573
macro avg	0.87	0.87	0.87	1573
weighted avg	0.89	0.89	0.89	1573

Figure 22: Classification Report for Decision Tree

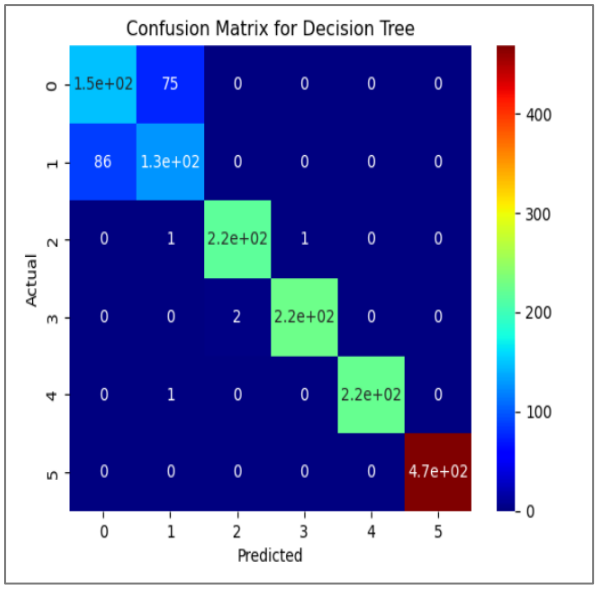


Figure 23: Confusion Matrix for Decision Tree

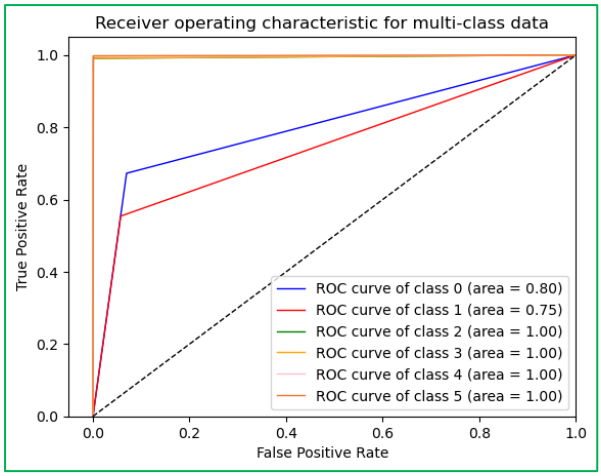


Figure 24: ROC for Decision Tree

Key Points for Decision Tree

- Decision Trees are prone to overfitting, especially when the tree depth is not controlled. Pruning or limiting the tree depth is crucial to prevent overfitting.
- Decision Trees can be sensitive to noisy data, outliers, or small variations in the dataset. Small changes can result in different tree structures.
- Decision Trees can handle both numerical and categorical features, making them versatile for a wide range of datasets.
- Decision Trees are easy to interpret and visualize. The flowchart-like structure provides a clear understanding of the decision-making process.
- Decision Trees provide a measure of feature importance, indicating which features are more influential in making predictions.
- Decision Trees are widely used for classification tasks, including spam detection, customer churn prediction, and medical diagnosis.

8. Random Forest

Random Forest is an ensemble learning method that builds a multitude of decision trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees. It is known for its robustness, versatility, and ability to handle complex relationships in data. Random Forest is an extension of the decision tree algorithm and overcomes some of its limitations, such as overfitting.

Random Forest belongs to the family of ensemble learning methods, where multiple models are combined to improve overall performance. In the case of Random Forest, the individual models are decision trees. Random Forest uses a technique called bagging, which involves training each decision tree on a random subset of the training data. This introduces diversity among the trees, reducing the risk of overfitting. Random Forest uses a technique called bagging, which

involves training each decision tree on a random subset of the training data. This introduces diversity among the trees, reducing the risk of overfitting. Random Forest uses a technique called bagging, which involves training each decision tree on a random subset of the training data. This introduces diversity among the trees, reducing the risk of overfitting.

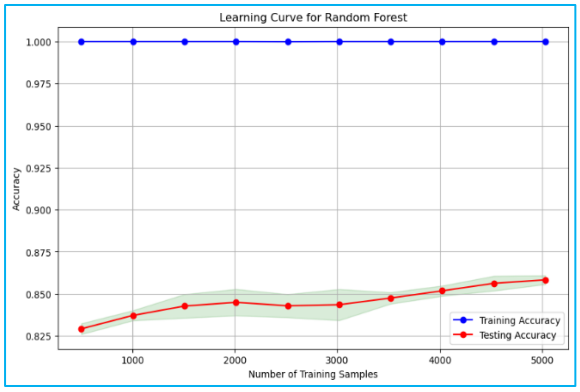


Figure 25: Learning Curve for Random Forest

From above figure we can observe that the Random Forest perfectly match the Training set that's why training Accuracy is maintained at 100%. And as Number of Samples Are increases Test accuracy are also increase. Here The Accuracy that we obtain is around 88%. Here we consider n_estimator=100 that represent Number of Decision Tree are used to formed Random- forest. The number of decision trees in the forest, known as n_estimators, is a hyperparameter that influences the performance of the Random Forest. Increasing the number of trees generally improves performance but also increases computational cost.

	precision	recall	f1-score	support
0	0.57	0.62	0.60	223
1	0.57	0.52	0.54	213
2	1.00	1.00	1.00	219
3	1.00	1.00	1.00	227
4	1.00	1.00	1.00	223
5	1.00	1.00	1.00	468
accuracy			0.88	1573
macro avg	0.86	0.86	0.86	1573
weighted avg	0.88	0.88	0.88	1573

Figure 26: Classification Report for Random Forest

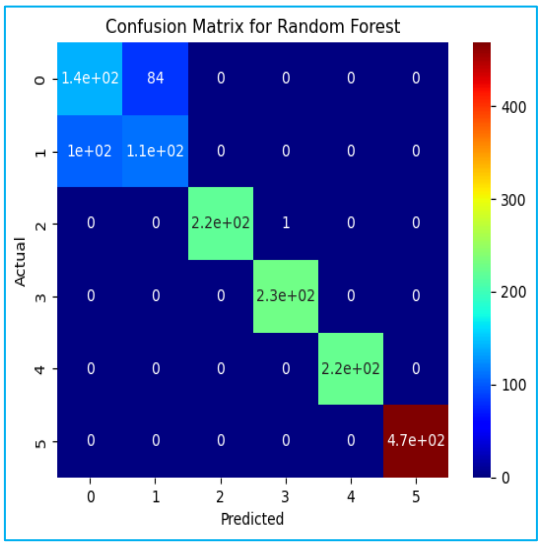


Figure 27: Confusion Matrix for Random Forest

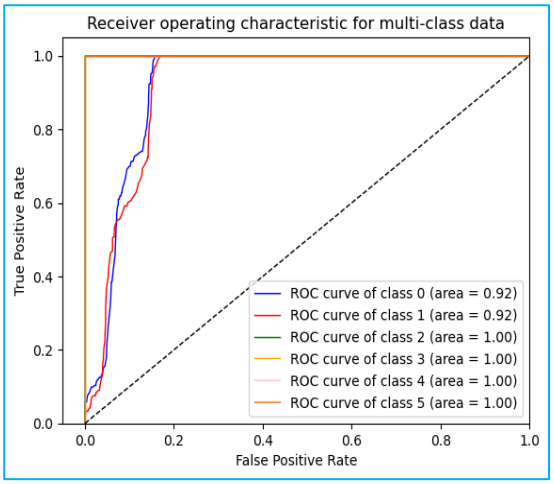


Figure 28: ROC Curve for Random Forest

Key Points About Random Forest

- Random Forest mitigates overfitting by combining multiple decision trees trained on different subsets of data and features.
- Random Forest can handle both classification and regression tasks, making it versatile for various types of machine learning problems.
- Random Forest is robust to noisy data and outliers due to the averaging effect of multiple trees.
- Random Forest can handle missing values in the dataset without the need for imputation.
- The interpretability of a Random Forest model decreases with the increasing number of trees. Training a large number of decision trees can be computationally expensive, especially for large datasets.

9. Adaboost Classifier (Boosting Technique)

AdaBoost, short for Adaptive Boosting, is an ensemble learning algorithm that combines the predictions of multiple weak learners (typically decision trees) to create a strong learner. AdaBoost assigns different weights to the training instances, emphasizing the misclassified instances in subsequent iterations. It is known for its adaptability and effectiveness in improving model performance by focusing on challenging examples.

AdaBoost employs weak learners, which are models that perform slightly better than random chance. In practice, decision trees with limited depth (stumps) are often used as weak learners. AdaBoost trains a sequence of weak learners iteratively. In each iteration, the algorithm assigns higher weights to instances that were misclassified by the previous weak learners. The weak learners' predictions are combined through a weighted sum. The weights are assigned based on the weak learners' accuracy in the previous iteration.

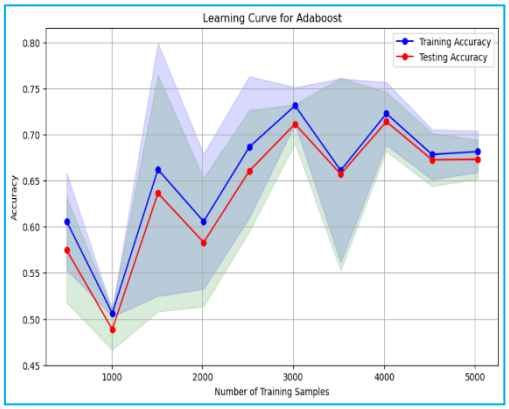


Figure 29: Learning Curve for Ada boost Classifier

Here we can See both Training and Testing Accuracy are Fallows each other as Number of Samples are increasing. The Accuracy that we obtain using Ada-boost Classifier is around 71%, Which is much larger than Logistic Regression but smaller than SVM (94%), Decision Tree (89%) and Random Forest (88%).

	precision	recall	f1-score	support
0	0.55	0.13	0.22	223
1	0.50	0.72	0.59	213
2	0.48	0.93	0.63	219
3	0.78	0.20	0.32	227
4	0.91	0.95	0.93	223
5	0.93	1.00	0.97	468
accuracy			0.71	1573
macro avg	0.69	0.65	0.61	1573
weighted avg	0.73	0.71	0.66	1573

Figure 30: Classification Report for Ada-boost Classifier

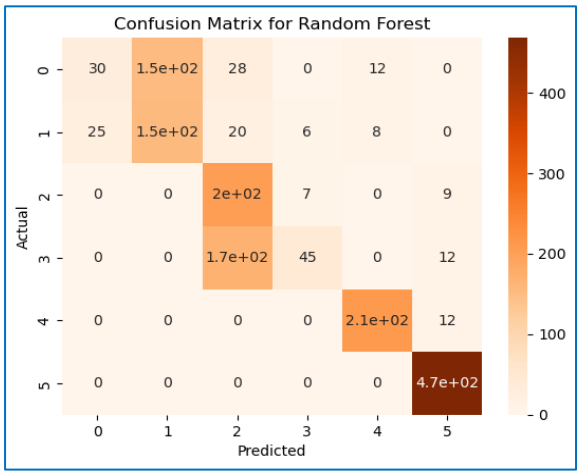


Figure 31: Confusion Matrix for Ada-boost Classifier

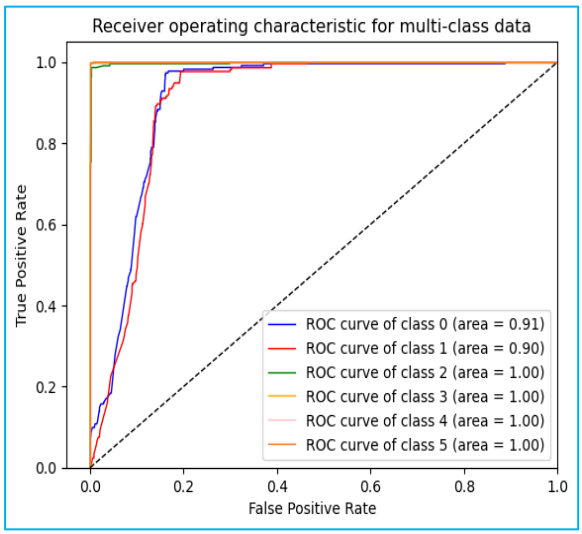


Figure 32: ROC for Adaboost Classifier

Key Points for Adaboost Classifier

- AdaBoost adapts to complex datasets and focuses on improving the classification of challenging instances.
- AdaBoost is versatile and can be used with different base learners, making it applicable to various types of data.
- AdaBoost mitigates overfitting by giving less weight to instances that are classified correctly in earlier iterations.
- AdaBoost works well even with weak learners, producing a strong classifier by combining their outputs.
- AdaBoost can be sensitive to noisy data and outliers, impacting the performance of weak learners.
- AdaBoost can be computationally expensive, especially when using a large number of iterations or complex weak learners.

10. Naive Bayes

Naive Bayes is a probabilistic machine learning algorithm based on Bayes' theorem, which describes the probability of an event based on prior knowledge of conditions related to the event. Despite its simplicity and the assumption of feature independence (hence the term "naive"), Naive Bayes is widely used for classification tasks, especially in text and document categorization, spam filtering, and sentiment analysis.

There are different types of Naive Bayes classifiers, including:

Gaussian Naive Bayes: Assumes that the features follow a Gaussian (normal) distribution.
As our dataset is continuous valued so we use Gaussian Naïve Bayes.

Multinomial Naive Bayes: Suitable for discrete data, often used in text classification.

Bernoulli Naive Bayes: Appropriate for binary features, where each feature is considered as a binary event.

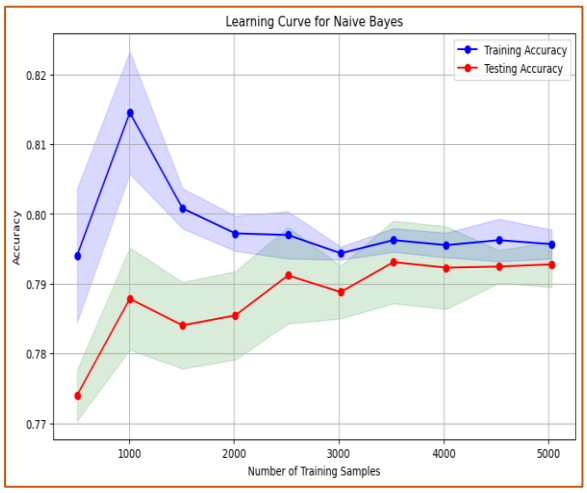


Figure 33: Learning Curve for Naïve Bayes

From Above figure it can be observed that as number of training example increases Test Accuracy also increases and it catches the Training Accuracy. The Accuracy that we obtain is Around 80%. And we also Obtain Precision, Recall and F1-Score for different Classes.

	precision	recall	f1-score	support
0	0.49	0.75	0.59	223
1	0.51	0.12	0.19	213
2	0.85	0.87	0.86	219
3	0.85	0.91	0.88	227
4	0.92	0.91	0.91	223
5	0.94	1.00	0.97	468
accuracy			0.80	1573
macro avg	0.76	0.76	0.73	1573
weighted avg	0.79	0.80	0.77	1573

Figure 34: Classification Report for Naive Bayes

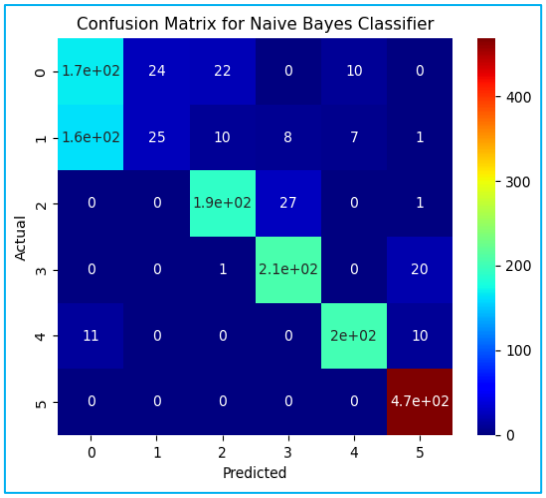


Figure 35: Confusion Matrix for Naive Bayes

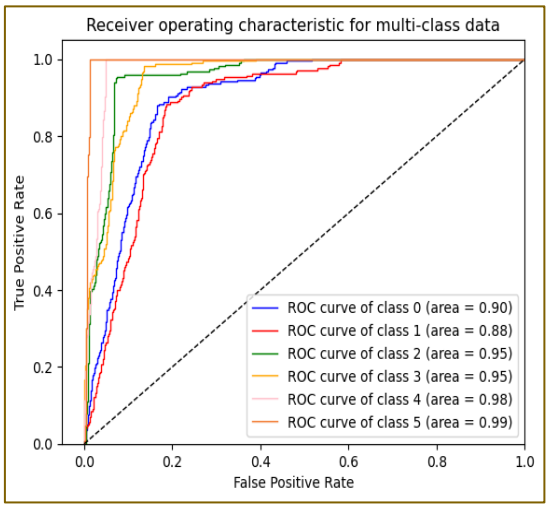


Figure 36: ROC for Naive Bayes

Key Points For Naive Bayes

- Naive Bayes is simple to implement and computationally efficient.
- Naive Bayes generally has low training time, making it suitable for large datasets.
- It can perform well even with limited training data.
- Naive Bayes is robust to irrelevant features, as it assumes independence among features.

- Naive Bayes can be sensitive to outliers, especially in the Gaussian Naive Bayes variant.
- The assumption of feature independence may not hold in all real-world scenarios, potentially impacting accuracy.
- If a feature in the test set has not been seen in the training set, the probability becomes zero, leading to difficulties in calculating posterior probabilities.

Naive Bayes is a simple yet effective algorithm, particularly suitable for text classification and other tasks where the independence assumption aligns well with the data. Its ease of implementation and good performance in certain scenarios make it a valuable tool in the machine learning toolkit.

11. ANN (Artificial Neural Network)

Artificial Neural Networks (ANNs) are a class of machine learning models inspired by the structure and functioning of the human brain. They consist of interconnected nodes, called neurons, organized in layers. ANNs are used for a variety of tasks, including classification, regression, pattern recognition, and more complex tasks such as image and speech recognition.

Neurons are the basic units in an ANN, analogous to neurons in the human brain. Each neuron receives input, processes it through an activation function, and produces an output. ANNs are organized into layers: input layer, hidden layers, and output layer. The input layer receives the initial data, hidden layers process information, and the output layer produces the final result. Connections between neurons are assigned weights, which determine the strength of the connection. Additionally, each neuron has a bias term that influences its activation. Learning in ANNs involves adjusting weights and biases based on training data.

The activation function of a neuron determines its output based on the weighted sum of inputs. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU). Feedforward is the process of passing data through the network from input to output. Backpropagation is the learning algorithm used to adjust weights and biases based on the difference between predicted and actual outputs during training. The loss function measures the difference between the predicted and actual outputs. During training, the goal is to minimize this loss, and backpropagation is used to update weights and biases accordingly.

Figure 37: Learning Curve for ANN

- Learning curves provide a visual assessment of how well a model is learning from the training data. They allow you to quickly observe whether the model is improving, plateauing, or degrading in performance.
- Learning curves help identify signs of overfitting or underfitting. Overfitting occurs when a model performs well on the training data but poorly on unseen data, while underfitting indicates that

the model is too simple to capture the underlying patterns in the data.

- Learning curves can guide the determination of early stopping criteria during training. If the validation performance plateaus or degrades while the training performance continues to improve, it may be an indication to stop training to prevent overfitting.
- Learning curves help in determining the optimal size of the training set. They show whether the model's performance continues to improve with more data or if a plateau(stable level) is reached, indicating that additional data may not be beneficial.

	precision	recall	f1-score	support
0	0.53	0.58	0.56	223
1	0.51	0.38	0.44	213
2	0.92	0.99	0.95	219
3	0.96	0.96	0.96	227
4	0.92	0.99	0.96	223
5	0.99	1.00	1.00	468
accuracy			0.85	1573
macro avg	0.81	0.82	0.81	1573
weighted avg	0.84	0.85	0.84	1573

Figure 38: Classification Report for ANN

For ANN we Use total Four Layer Where Two is Hidden Layer and one for Input Layer and Other one is for output Layer. The activation Function we used is “relu” and solver we use is “adam” for getting better performance from the model. The Accuracy That we obtain is Around 85% which is Comparable with SVM, Decision Tree and Random Forest.

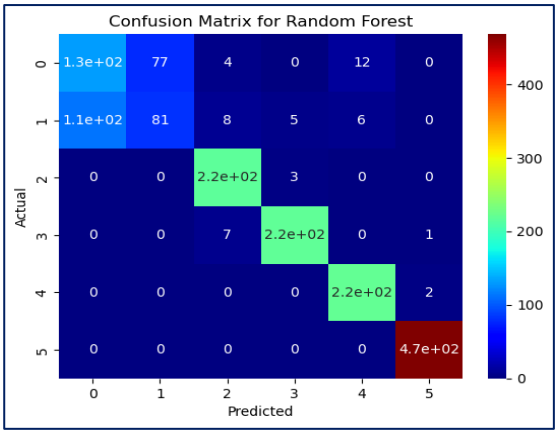


Figure 39: Confusion Matrix for ANN

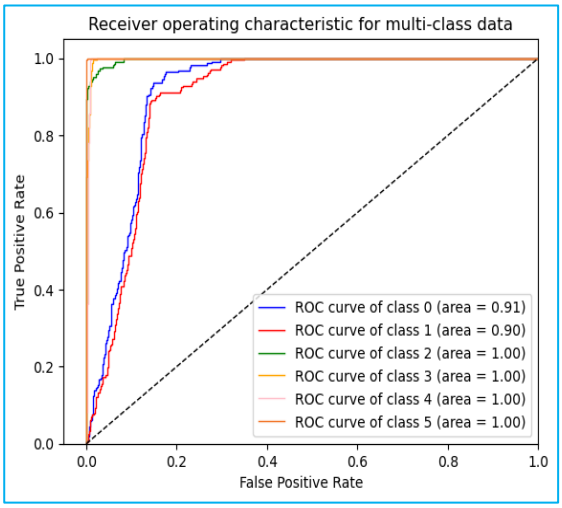


Figure 40: ROC Curve for ANN

Key Features about ANN

- ANNs can be applied to a wide range of tasks, from simple classification to complex pattern recognition.
- ANNs automatically learn relevant features from the data, eliminating the need for manual feature engineering.
- The non-linear activation functions in ANNs allow them to model complex relationships in data.
- ANNs can process multiple inputs simultaneously, enabling parallel computation.

- ANNs often require large amounts of data for effective training.
- Training deep networks can be computationally intensive and may require specialized hardware.

12. Results and Conclusion

So, Here We basically Compare total Eight ML Models, Among them SVM perform well giving accuracy of Around 94% after performing Hyper parameter Tunning. Decision Tree and Random Forest gives almost Similar accuracy of 89%. Here We don't get any significant change in Accuracy in case of Random Forest it is almost equal with Decision Tree. With KNN (K=1) we got an Accuracy of Around 89% and with ANN we got an Accuracy of 85%. But for Adaboost we got an Accuracy of Around 70% which is lesser Compared to other Algorithm, but we can improve its Accuracy by Appling Proper Hyper Parameter Tunning. And for Logistic Regression we got an Accuracy of around 34% from that we can conclude that Logistic Regression Not suitable for Multi Class, it is only Suitable for binary Classification.

Here We perform the analysis for Multi class so here we have total six precision, six Recall and six F1-score in each Classification Report and in each Confusion Matrix. We also Provide ROC plot for all individual class by Considering One vs rest.

Accuracy are not perfectly define the model performance in case imbalance data set. So, we go for other Parameter like Recall, which is the ratio of correctly predicted positive observations to the all observations in the actual class

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

So we found that SVM has Higher Recall is around 0.9427. Means 94.3% of the Fault are Correctly Classified, in summary, a recall of 0.9427 in the context of fault analysis implies a high sensitivity of the machine learning algorithms to detect faults but the training time of SVM is high due to Hyper parameter Tunning. So apart from SVM, Radom Forest and Decision Tree are also giving good Recall and they have less training time, so one can use this model on daily basis for prediction of Fault.

Precision that we obtain for SVM is 0.943 So, a precision of 0.943 suggests that out of all instances predicted as faults, 94.3% are true faults, and only a small fraction (5.7%) are false alarms. So Here Decision Tree and Random Forest also have Good Precision. So we can Choose any of them because the training time of the SVM are Large. Apart from SVM, KNN also give good result but it is not suitable for Large Data set and also it is computationally expensive because it produce result after finding distance between the nearest neighbours.

Compute Precision, Recall, and F1 Score for Each Class:

Macro-Averaging:

- Calculate the macro-average for precision, recall, and F1 score by averaging the scores across all classes. This method treats all classes equally and does not consider class imbalances:

Macro-Average Precision (Macro-P):

$$\text{Macro-P} = \frac{\sum_{i=1}^{N} P_i}{N}$$

Macro-Average Recall (Macro-R):

$$\text{Macro-R} = \frac{\sum_{i=1}^{N} R_i}{N}$$

Macro-Average F1 Score (Macro-F1):

$$\text{Macro-F1} = \frac{\sum_{i=1}^{N} F1_i}{N}$$

- N is the number of classes.

Micro-Averaging:

- Calculate the micro-average for precision, recall, and F1 score by aggregating the counts of true positives, false positives, and false negatives across all classes and then calculating the metrics:

Micro-Average Precision (Micro-P):

$$\text{Micro-P} = \frac{\sum_{i=1}^{N} \text{True Positives}_i}{\sum_{i=1}^{N} (\text{True Positives}_i + \text{False Positives}_i)}$$

Micro-Average Recall (Micro-R):

$$\text{Micro-R} = \frac{\sum_{i=1}^{N} \text{True Positives}_i}{\sum_{i=1}^{N} (\text{True Positives}_i + \text{False Negatives}_i)}$$

Micro-Average F1 Score (Micro-F1):

$$\text{Micro-F1} = \frac{2 \times \text{Micro-P} \times \text{Micro-R}}{\text{Micro-P} + \text{Micro-R}}$$

- Micro-averaging is suitable when there are imbalances in class frequencies.

Weighted Averaging:

- If class imbalances are present and you want to consider the influence of each class based on its size, you can calculate weighted averages. Weights are usually assigned based on the number of true instances for each class.

Weighted Average Precision (Weighted-P):

$$\text{Weighted-P} = \frac{\sum_{i=1}^{N}(\text{True Positives}_i/(\text{True Positives}_i+\text{False Positives}_i))\times\text{Weight}_i}{\sum_{i=1}^{N}\text{Weight}_i}$$

Weighted Average Recall (Weighted-R):

$$\text{Weighted-R} = \frac{\sum_{i=1}^{N}(\text{True Positives}_i/(\text{True Positives}_i+\text{False Negatives}_i))\times\text{Weight}_i}{\sum_{i=1}^{N}\text{Weight}_i}$$

Weighted Average F1 Score (Weighted-F1):

$$\text{Weighted-F1} = \frac{\sum_{i=1}^{N}F1_i\times\text{Weight}_i}{\sum_{i=1}^{N}\text{Weight}_i}$$

- The weights (Weight_i) are typically proportional to the number of true instances for each class.

ML Model Evaluation Table

Name of ML Model	Accuracy	Precision	Recall	F1-Score
Logistic Regression	34	0.275	0.341	0.213
KNN	89.25	0.8926	0.8925	0.8923
SVM	94.27	0.943	0.9427	0.9427
Decision Tree	89.31	0.8938	0.8931	0.893
Random Forest	87.98	0.8795	0.8798	0.8795
Adaboost	70.629	0.7317	0.7062	0.6638
Naive Bayes	80.038	0.79	0.8	0.774
ANN	84.8696	0.839	0.8486	0.8421

Figure 41: Result of all ML MODELS in Tabulated Form

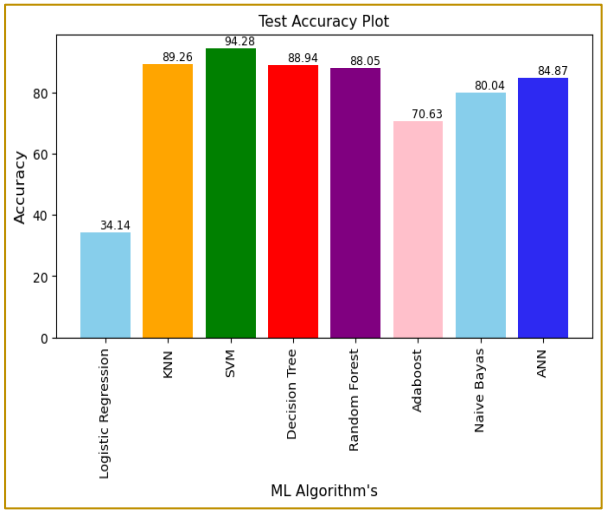


Figure 42: Comparison of Accuracy for Different ML Models

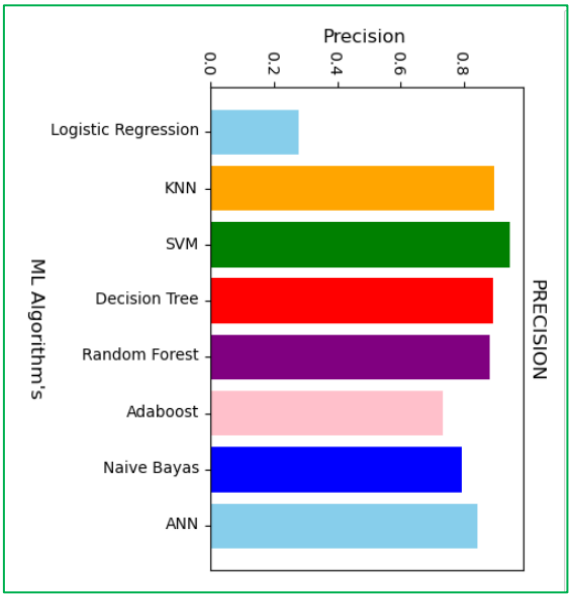


Figure 43: Comparison of Precision for Different ML Models

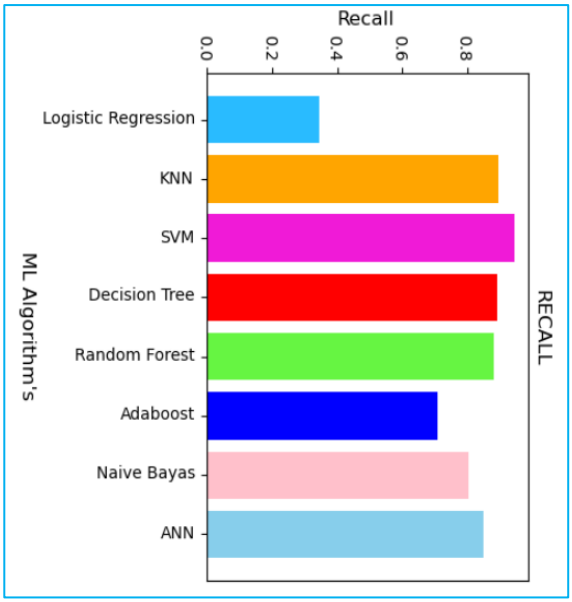


Figure 43: Comparison of Recall for Different ML Models

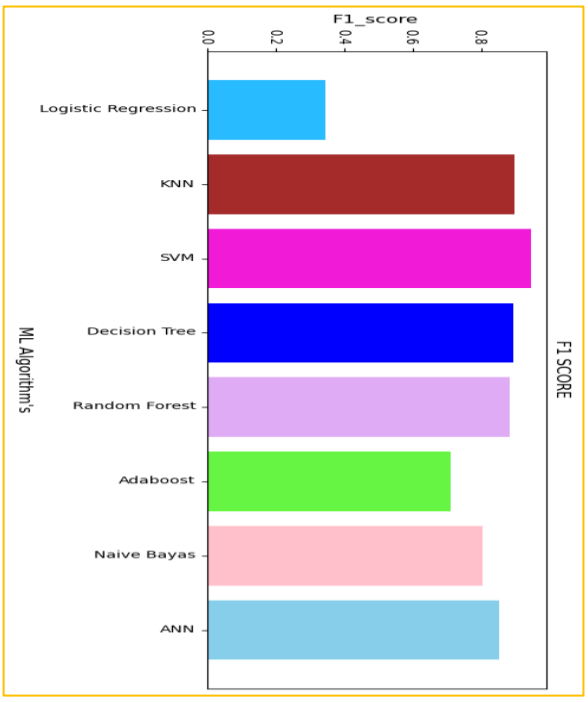


Figure 43: Comparison of F1 score for Different ML Models

13.References:

1. Electrical Fault Dataset is available on Kaggle:"/kaggle/input/electric-faultline-detection/train_dataset.csv"
2. Y. Freund, R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," Journal of Computer and System Sciences, 1997.DOI: 10.1006/jcss.1997.1504
3. C. Cortes, V. Vapnik, "Support-Vector Networks," Machine Learning, 1995.
4. L. Breiman, "Random Forests," Machine Learning, 2001.
5. D. E. Rumelhart, G. E. Hinton, R. J. Williams, "Learning Representations by Backpropagating Errors," Nature, 1986.