Project Report

**Muhammad Waleed Amir**

**F2021266233**

**V5**

GitHub link :- https://github.com/mwaleedamir/Final-project-Ai

# Introduction

## Machine Learning (ML):

Machine learning is a subset of artificial intelligence (AI) that focuses on the development of algorithms and statistical models that enable computers to perform tasks without explicit instructions. In traditional programming, developers provide explicit instructions to the computer on how to perform a task. In contrast, in machine learning, algorithms are trained on data to learn patterns and relationships, allowing the computer to make decisions or predictions without being explicitly programmed.

## Types of Machine Learning Techniques:

* 1. **Supervised Learning:**
     + Supervised learning is a type of machine learning where the algorithm learns from labelled data, which means it is provided with input-output pairs during the training process.
     + The goal of supervised learning is to learn a mapping function from input variables to output variables based on the labelled training data.

- Common supervised learning algorithms include linear regression, logistic regression, decision trees, random forests, support vector machines (SVM), and neural networks.

## Unsupervised Learning:

* + - Unsupervised learning is a type of machine learning where the algorithm learns from unlabeled data, which means it is not provided with explicit output labels during the training process.

- The goal of unsupervised learning is to find patterns, structures, or relationships in the data without guidance.

- Common unsupervised learning algorithms include clustering algorithms (e.g., K-means clustering, hierarchical clustering) and dimensionality reduction techniques (e.g., principal component analysis (PCA), t-distributed stochastic neighbour embedding (t-SNE)).

## Reinforcement Learning:

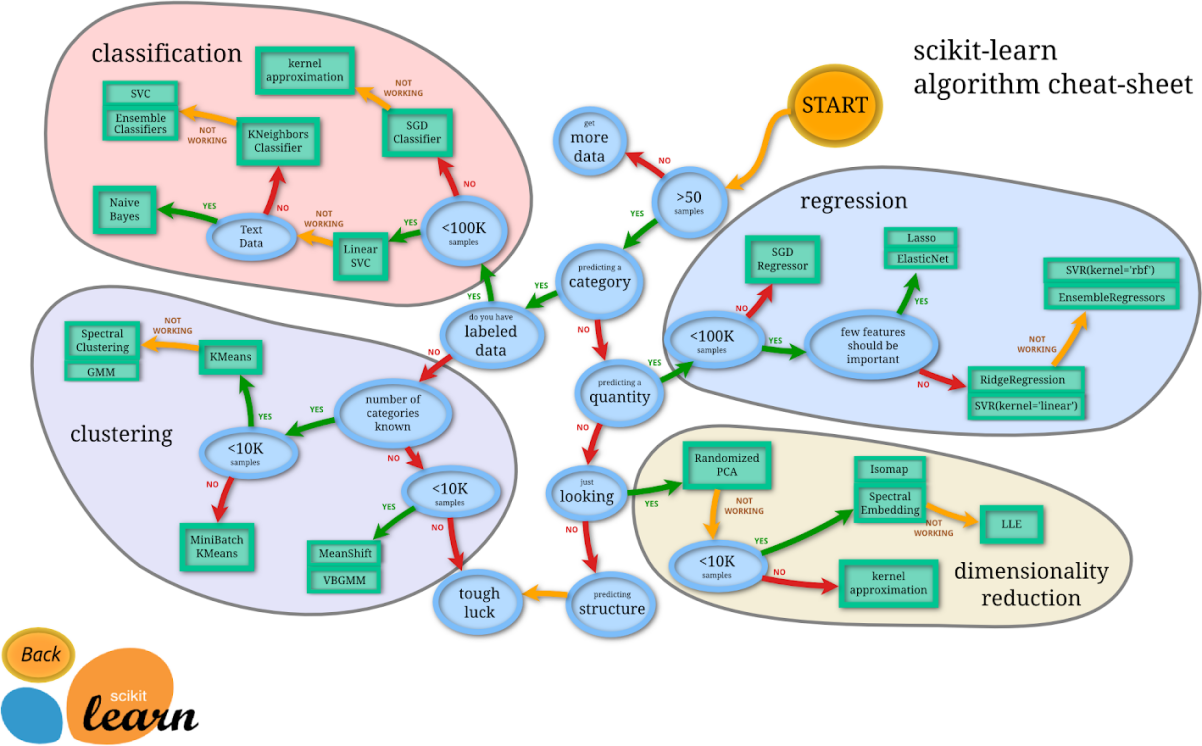
* + - Reinforcement learning is a type of machine learning where an agent learns to make decisions by interacting with an environment to achieve a goal.
    - The agent learns through trial and error, receiving feedback in the form of rewards or penalties based on its actions.
* The goal of reinforcement learning is to learn the optimal policy, which specifies the best action to take in a given state to maximize cumulative reward.
* Common reinforcement learning algorithms include Q-learning, deep Q-networks (DQN), policy gradient methods, and actor-critic methods.

# Aims & Objectives

The aims and objectives for this lab are as follows:

* Gain an understanding of the Sci-Kit library in Python.
* Study various Methods of Supervised Learning.
* Study various Terminologies related to learning.
* Compare various methods based on accuracy and time complexity.
* Train various models for the classification of datasets.

# Scikit in Python

Scikit-learn, commonly known as sklearn, is a widely used open-source Python library for machine learning that offers a simple and efficient toolkit for data mining and analysis. Built on top of NumPy, SciPy, and Matplotlib, it provides a consistent API for various algorithms, including classification, regression, clustering, and dimensionality reduction. Scikit-learn boasts an extensive collection of models and tools for model evaluation, selection, and preprocessing, making it suitable for both beginners and experienced practitioners. With its efficient implementation, seamless integration with other Python libraries, and active community support, scikit-learn has become an indispensable tool for data scientists, machine learning engineers, and researchers alike.

Features of Scikit are:

* 1. Comprehensive library for machine learning in Python
  2. Simple and efficient tools for data mining and analysis
  3. Consistent API design for easy algorithm switching
  4. Supports supervised and unsupervised learning algorithms
  5. Includes classification, regression, clustering, and dimensionality reduction algorithms
  6. Designed to be accessible and easy to use
  7. Integrates well with other Python libraries like NumPy, SciPy, and matplotlib
  8. Provides tools for model evaluation, hyperparameter tuning, and cross-validation
  9. Supports feature extraction and selection
  10. Large and active community with extensive documentation and tutorials

# Supervised Learning Algorithms

In discussing supervised learning algorithms like k-nearest neighbours (kNN), Support Vector Machines (SVM), Bayesian Naive Networks, and Perceptron Layers, it's important to highlight their unique characteristics and applications.

## k-Nearest Neighbors (kNN):

This algorithm is simple yet powerful, relying on the principle of similarity to make predictions. It classifies data points based on the majority class of their k-nearest neighbours in the feature space. kNN is non-parametric, meaning it does not make assumptions about the underlying data distribution, making it suitable for a wide range of applications such as classification and regression. However, its performance can degrade with high-dimensional data and large datasets due to computational costs.

## Support Vector Machines (SVM):

SVM is a versatile algorithm used for both classification and regression tasks. It works by finding the optimal hyperplane that best separates classes in the feature space while maximizing the margin between classes. SVM can handle linear and non-linear classification tasks using different kernel functions like linear, polynomial, and radial basis functions (RBF). SVMs are effective in high-dimensional spaces and are robust against overfitting, making them suitable for various applications such as text categorization, image classification, and bioinformatics.

## Bayesian Naive Networks:

Bayesian Naive Networks (BNN) are probabilistic graphical models that use Bayes' theorem to model the probability of different classes given the input features. Despite their "naive" assumption of feature independence, BNNs can achieve strong performance in classification tasks, especially when

dealing with high-dimensional data. They are computationally efficient, interpretable, and can handle missing data well. BNNs are widely used in text classification, spam filtering, and medical diagnosis.

## Perceptron Layer:

The Perceptron is one of the simplest neural network architectures, consisting of a single layer of computational units (perceptrons) connected to input features with weighted connections. Each perceptron computes a weighted sum of inputs and applies a threshold function to produce the output. Perceptron layers can learn linear decision boundaries and are often used as building blocks for more complex neural network architectures like multi-layer perceptrons (MLPs). While limited to linearly separable problems, they are computationally efficient and easy to interpret.

Every Algorithm has some hyperparameters, which can be tuned for better accuracy and time complexity.

# Basic Terminologies related to Learning

**Algorithm:** A set of rules or instructions followed to solve a particular problem, such as a machine learning task.

**Model:** A representation of a system or process that is learned from data and used to make predictions or decisions.

**Feature:** An input variable used in a machine learning model. Features are also referred to as attributes or variables.

**Label:** The output variable in a supervised learning problem that the model aims to predict.

Training Data: The dataset used to train a machine learning model. It consists of input-output pairs used to learn the model.

**Test Data:** The dataset used to evaluate the performance of a trained machine learning model. It is separate from the training data and is used to assess how well the model generalizes to new, unseen data.

**Validation Set:** A subset of the training data used to tune hyperparameters and evaluate different models during the model selection process.

**Overfitting:** A phenomenon where a machine learning model learns the training data too well, capturing noise and irrelevant patterns, leading to poor performance on new data.

**Underfitting:** A phenomenon where a machine learning model is too simple to capture the underlying patterns in the data, leading to poor performance on both the training and test data.

**Hyperparameters:** Parameters of a machine learning model that are set before the learning process begins, such as the learning rate or the number of hidden units in a neural network.

Feature Engineering: The process of selecting, transforming, or creating new features from the raw data to improve the performance of machine learning models.

**Cross-Validation:** A technique used to assess the performance of a machine learning model by splitting the data into multiple subsets, training the model on some subsets, and evaluating it on others.

**True Positive (TP):** The number of correctly predicted positive instances (e.g., correctly identifying a disease as present).

**True Negative (TN):** The number of correctly predicted negative instances (e.g., correctly identifying a disease as absent).

**False Positive (FP):** The number of incorrectly predicted positive instances (e.g., incorrectly identifying a disease as present when it is actually absent, also known as a Type I error).

**False Negative (FN):** The number of incorrectly predicted negative instances (e.g., incorrectly identifying a disease as absent when it is actually present, also known as a Type II error).

**Confusion Matrix**: A table that is used to describe the performance of a classification model on a set of test data for which the true values are known. It is nxn matrix that contains counts of true class against predicted class.

The format for confusion matrix of results of model trained on 2 class (Positive/Negative) data is as:

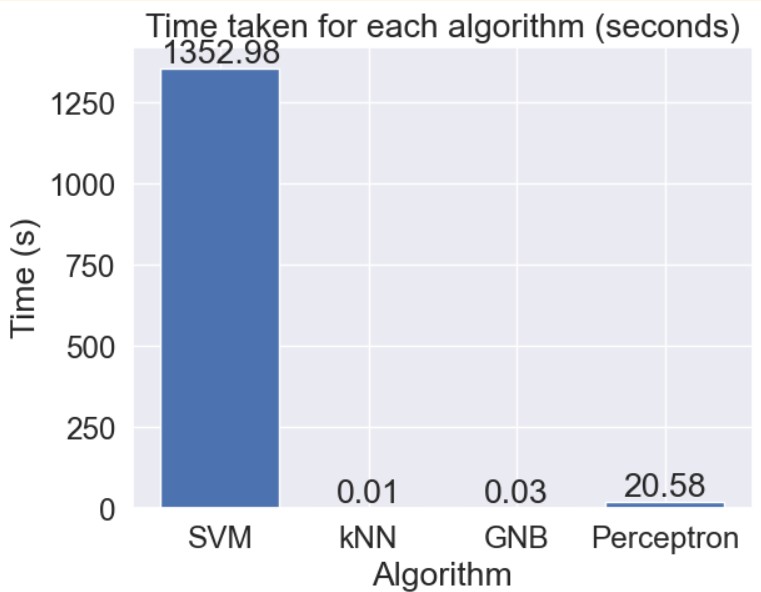
|  |  |  |
| --- | --- | --- |
|  | **Predicted Positive** | **Predicted Negative** |
| **Actual Positive** | True Positive (TP) | False Negative (FN) |
| **Actual Negative** | False Positive (FP) | True Negative (TN) |

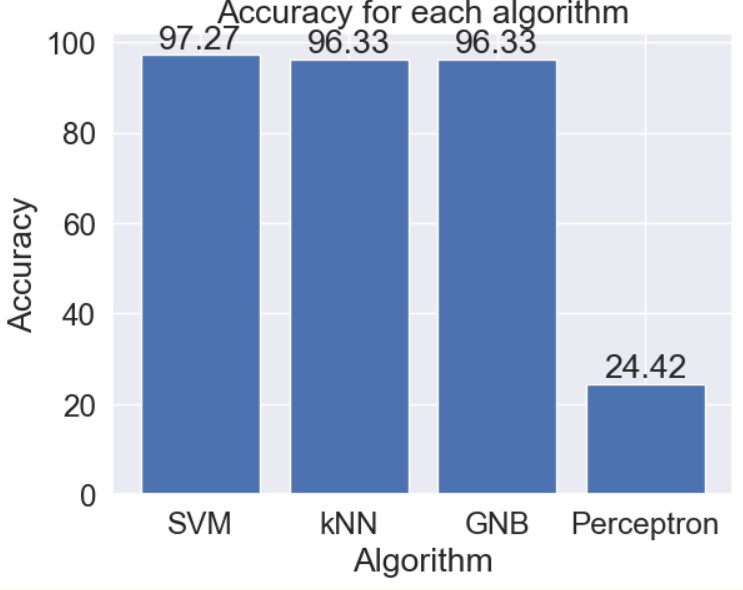
# Results for HR Churn Prediction Dataset

|  |  |
| --- | --- |
| **Size** | 14999, 10 |
| **Features** | satisfaction\_level, last\_evaluation, number\_project, average\_montly\_hours, time\_spend\_company, Work\_accident, promotion\_last\_5years, dept, salary  (9 Features) |
| **Labels** | left (True/False) |
| **Test Size** | 30% |

## Confusion Matrices

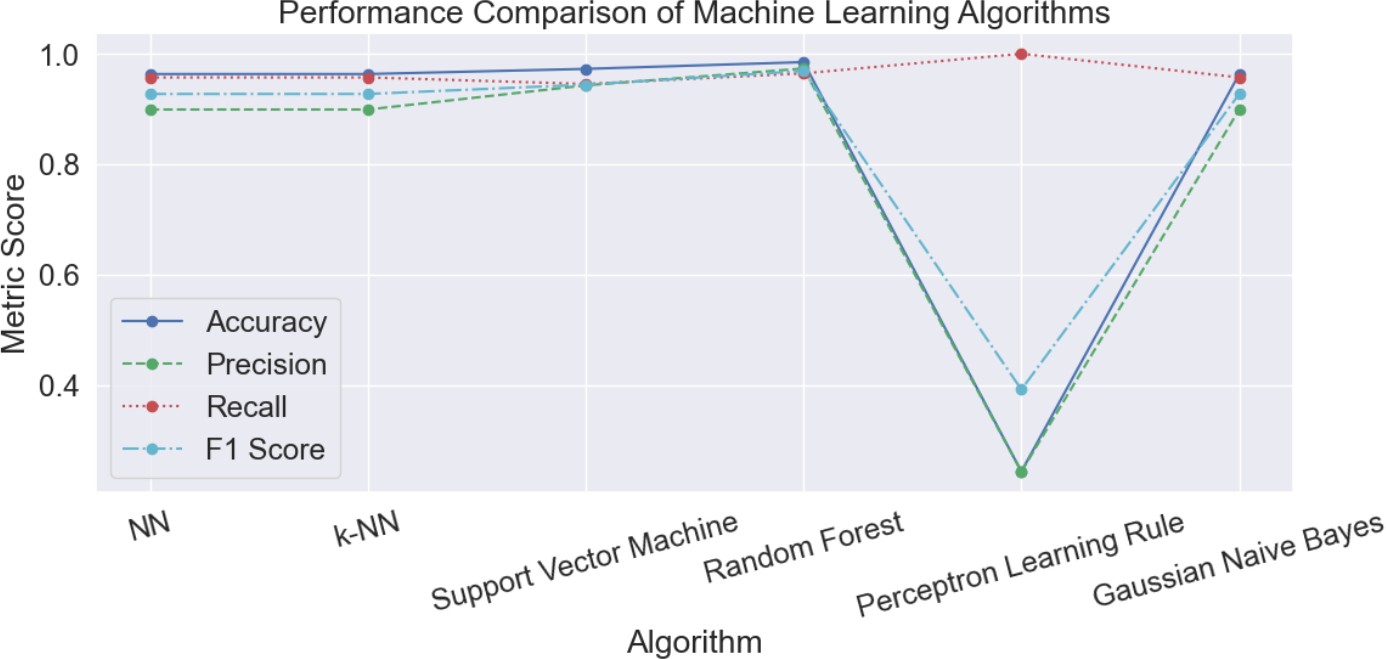
|  |  |
| --- | --- |
|  |  |
|  |  |





**Comments:** For this imbalance class dataset of HR Churn, perceptron has performed the worst while all the other algorithms SVM, GNB and kNN have performed the best based on the accuracy score. The simple explanation would be the perceptron classifier's linear nature, which made it inefficient to capture the non-linearity of the dataset misclassifying one complete class altogether. While the SVM performed the best due to the inherent non-linearity in its algorithm and the hyperparameter tuning being done in the coding it was computationally very expensive. It is not wrong to say that GNB and kNN overall performed the best regarding time complexity and performance metrics.

(The below graph also has the result of Random Forest whose discussion or coding is not mentioned here as it was not the requirement.)



# Sample Code Used

## Importing the libraries:

**import pandas as pd**

**import matplotlib.pyplot as plt import seaborn as sns**

**import numpy as np sns.set()**

***#%matplotlib inline***

**from sklearn.model\_selection import train\_test\_split**

**from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, f1\_score, recall\_score**

**from sklearn.metrics import classification\_report from sklearn.model\_selection import GridSearchCV from sklearn.svm import SVC**

**from sklearn.preprocessing import StandardScaler**

**from sklearn.neighbors import KNeighborsClassifier from sklearn.ensemble import RandomForestClassifier from sklearn.utils import resample**

**from sklearn.naive\_bayes import GaussianNB**

**Loading the dataset:**

***# Load the data***

**data=**

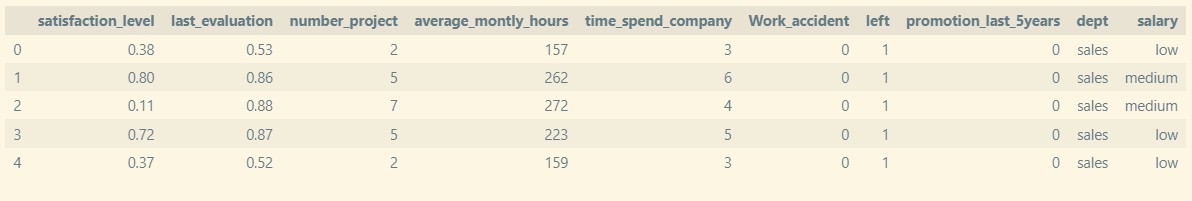
**pd.read\_csv('D:/University/Study**

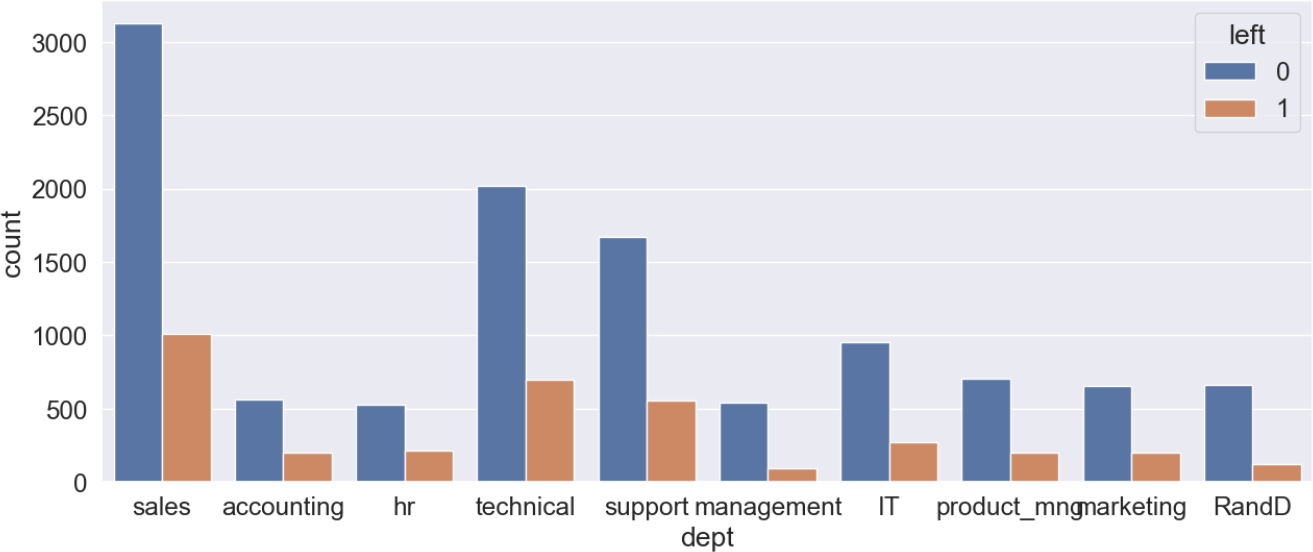
**Material/8th Semester/Intelligence**

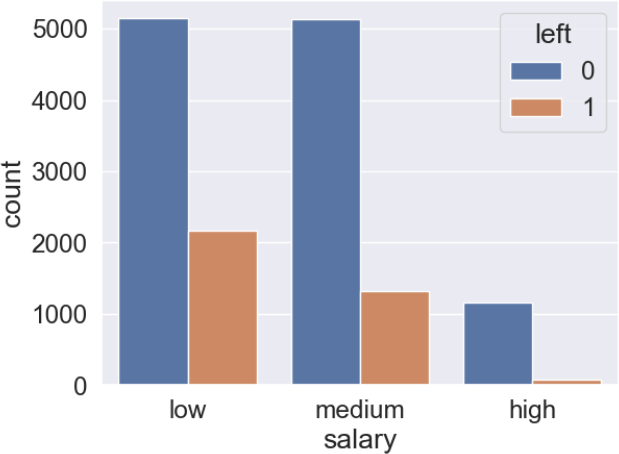
**Systems/Theory/Assignments/Assignment\_5/HR\_comma\_sep.csv')**

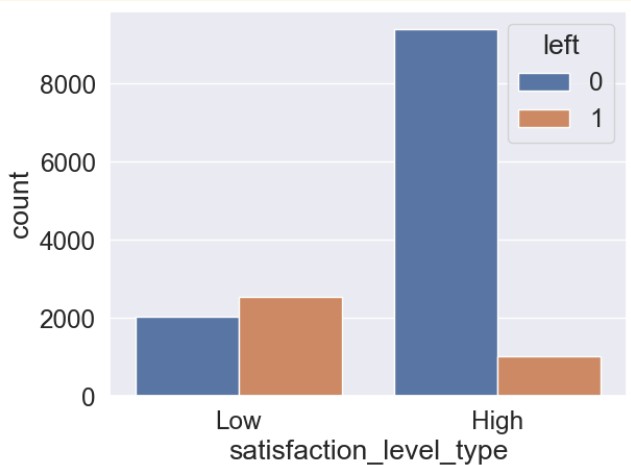
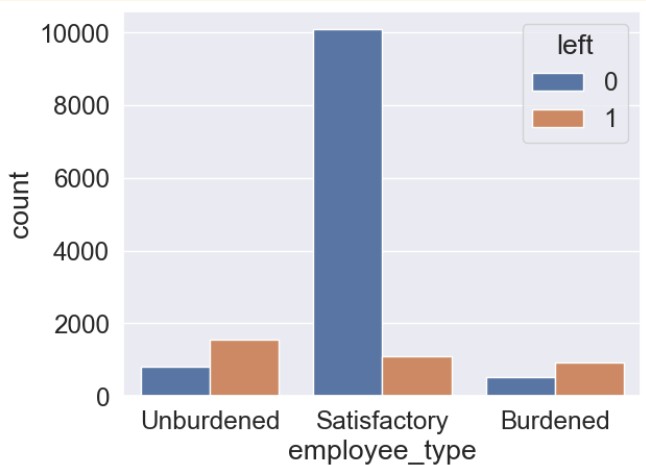
***# Display the top 5 rows.***

**data.head()**

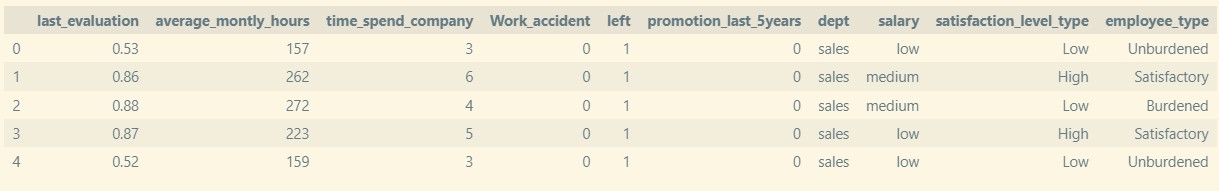








## After Feature Engineering:



**print(x\_train.shape, y\_train.shape) print(x\_test.shape, y\_test.shape)**

**(10499, 23) (10499, 1)**

**(4500, 23) (4500, 1)**

## Support Vector Machine

***# Feature scaling***

**scaler = StandardScaler()**

**x\_train\_scaled = scaler.fit\_transform(x\_train) x\_test\_scaled = scaler.transform(x\_test)**

***# Grid search with SVM and hyperparameter tuning***

**param\_grid = {'C': [0.01, 0.1, 1, 10], 'gamma': [0.01, 0.1, 1, 10], 'kernel': ['linear', 'rbf']}**

**svm\_grid\_search = GridSearchCV(SVC(), param\_grid, cv=5) svm\_grid\_search.fit(x\_train\_scaled, y\_train)**

***# Get the best model and predictions***

**best\_svm\_model = svm\_grid\_search.best\_estimator\_**

**y\_pred\_svm = best\_svm\_model.predict(x\_test\_scaled)**

**Accuracy: 0.9726666666666667**

**F1-Score : 0.9441163107678329**

**Recall: 0.9454049135577798**

**Precision: 0.9428312159709619**

**K-Nearest Neighbour**

***# Making instance and training the model***

**model\_nn = KNeighborsClassifier(n\_neighbors=1) model\_nn.fit(x\_train\_scaled, y\_train)**

***# Get predictions***

**y\_pred\_nn = model\_nn.predict(x\_test\_scaled)**

**Accuracy: 0.9633333333333334**

**F1-Score : 0.9272807404142795**

**Recall: 0.9572338489535942**

**Precision: 0.8991452991452992**

## Gaussian Naive Baye's Algorithm

**model\_gnb = GaussianNB()**

**model\_gnb.fit(x\_train\_scaled, y\_train)**

**y\_pred\_gnb = model\_nn.predict(x\_test\_scaled)**

**Accuracy: 0.9633333333333334**

**F1-Score : 0.9272807404142795**

**Recall: 0.9572338489535942**

**Precision: 0.8991452991452992**

## Perceptron Network:

**class Perceptron:**

**def init (self, learning\_rate=0.01, max\_iter=100): self.learning\_rate = learning\_rate**

**self.max\_iter = max\_iter**

**self.weights = None *# Initialize weights as None***

**def fit(self, X, y): """**

**Train the Perceptron model on the given data X and target labels y.**

**Args:**

**X: A 2D numpy array of training features (shape: n\_samples, n\_features).**

**y: A 1D numpy array of binary target labels (shape: n\_samples,).**

**"""**

***# Add a bias term (column of ones) to the feature matrix***

**X = np.hstack((np.ones((X.shape[0], 1)), X))**

**self.weights = np.zeros(X.shape[1]) *# Initialize weights with zeros***

***# Training loop***

**for \_ in range(self.max\_iter): errors = 0**

**for i, x\_i in enumerate(X):**

**predicted = np.dot(self.weights, x\_i) actual = y[i]**

***# Update weights if prediction is wrong***

**if predicted \* actual <= 0: errors += 1**

**self.weights += self.learning\_rate \* actual \* x\_i**

***# Early stopping if no errors occur in an iteration***

**if errors == 0: break**

**def predict(self, X): """**

**Predict class labels for new data points X.**

**Args:**

**X: A 2D numpy array of new data points (shape: n\_samples, n\_features).**

**Returns:**

**A 1D numpy array of predicted class labels (shape: n\_samples,).**

**"""**

***# Add a bias term to the feature matrix***

**X = np.hstack((np.ones((X.shape[0], 1)), X))**

**predictions = np.sign(np.dot(X, self.weights)) return predictions**

**learning\_rates =[1,0.1,0.01,0.001,0.0001] y\_train\_np = y\_train.to\_numpy()**

**j=1**

**for i in learning\_rates: print()**

**model\_perceptron = Perceptron(learning\_rate=i, max\_iter=1000)**

**model\_perceptron.fit(x\_train\_scaled, y\_train\_np)**

**y\_pred\_plr = model\_perceptron.predict(x\_test\_scaled)**

**Accuracy\_1: 0.24422222222222223**

**F1-Score\_1: 0.3925701018038936**

**Recall\_1: 1.0**

**Precision\_1: 0.24422222222222223**

**Accuracy\_2: 0.24422222222222223**

**F1-Score\_2: 0.3925701018038936**

**Recall\_2: 1.0**

**Precision\_2: 0.24422222222222223**

**Accuracy\_3: 0.24422222222222223**

**F1-Score\_3: 0.3925701018038936**

**Recall\_3: 1.0**

**Precision\_3: 0.24422222222222223**

**Accuracy\_4: 0.24422222222222223**

**F1-Score\_4: 0.3925701018038936**

**Recall\_4: 1.0**

**Precision\_4: 0.24422222222222223**

**Accuracy\_5: 0.24422222222222223**

**F1-Score\_5: 0.3925701018038936**

**Recall\_5: 1.0**

**Precision\_5: 0.24422222222222223**

## Conclusion:

A basic insight into various ML Algorithms for supervised learning was developed and working of “sklearn” in python was demonstrated.

## References:

1. Harvard CS 50 AI-Learning: [Lecture 4 - CS50's Introduction to Artificial Intelligence with Python](https://cs50.harvard.edu/ai/2024/notes/4/)
2. Scikit Documentation : [scikit-learn: machine learning in Python — scikit-learn 1.4.1 documentation](https://scikit-learn.org/stable/)

## Upcoming:

1. Studying Model Optimization Techniques
2. Studying Reinforcement Learning and Neural Networks
3. Studying Unsupervised Learning

## Considerations:

1. Considering about ethical concerns related to ML and AI
2. Implementing AI for real-world sustainable Applications