**Libraries Used in the algorithms:**

1. **Pandas**- Python library for data manipulation and analysis. It provides data structures for efficiently storing and analyzing data, such as data frames.
2. **NumPy**- fundamental package for scientific computing in Python. It provides support for arrays, matrices, and mathematical functions.
3. **TensorFlow**- TensorFlow is an open-source machine learning framework developed by Google. In this code, it is used for building and training neural networks.
4. **Scikit-Learn**- provides various tools for data preprocessing.

**Label Encoder** is used to encode categorical labels into numerical values.

**train\_test\_split** is used to split the dataset into training and testing sets.

**Standard Scaler** is used for standardizing (scaling) the numerical features.

**metrics** module provides various evaluation metrics for classification problems, like precision, recall, and F1-score.

1. **Keras**- Keras is an open-source deep learning library that runs on top of other deep learning frameworks like TensorFlow.

**Sequential** is a linear stack of neural network layers.

**Dense** represents a fully connected layer in the neural network.

**Dropout** is a regularization technique to prevent overfitting.

**Adam** is an optimization algorithm for training neural networks.

1. **Label Encoder** encodes categorical columns into numerical values using LabelEncoder
2. **Standardization**: The numerical features in the dataset are standardized using StandardScaler. Standardization scales the features to have zero mean and unit variance.
3. **MinMaxScaler** is used to scale (normalize) the features to a specific range (usually [0, 1]).The training and testing sets are scaled separately to prevent data leakage.
4. **GridSearchCV** is used for hyperparameter tuning via grid search.
5. **MLPClassifier** is a Multi-Layer Perceptron (MLP) classifier from scikit-learn.
6. **time** is used for timing the code's execution.
7. **random\_seed** is used to set a seed for random number generation, ensuring reproducibility.
8. **random\_state** is set to ensure reproducibility.
9. **SimpleImputer** is a scikit-learn class used for imputing missing values in a dataset

Imputation is the process of replacing missing or NaN (Not a Number) values with a specific value, such as the mean, median, or a constant, to ensure that the dataset is complete and suitable for machine learning tasks.

1. **Perceptron** is a linear classification algorithm used for binary classification tasks
2. **RandomForestClassifier** is an ensemble machine learning model from scikit-learn that is used for classification tasks. It's an ensemble of decision trees.
3. **XGBClassifier** is the classifier from the XGBoost library for gradient boosting.
4. **LogisticRegression** is a linear classifier for binary classification. It is an ensemble learning method that builds a strong predictive model by combining the predictions of multiple weak learners, typically decision trees.
5. **StackingClassifier** is an ensemble technique used to create a meta-learner that combines predictions from multiple base classifiers.

**Terms of deep learning and NN used in code:**

* **Epochs:** An epoch is one complete pass through the entire training dataset. During one epoch, the model sees and learns from all the training examples.

Training a neural network involves adjusting the model's parameters (weights and biases) to minimize the loss function, which measures the error between the predicted values and the actual targets. Increasing the number of epochs allows the model to learn from the data more, potentially improving its performance.

* **Batch Size:** The batch size specifies how many examples are processed together in each forward and backward pass. It affects the model's learning dynamics.

smaller batch sizes can result in noisy updates.

Larger batch sizes can provide a smoother and more stable optimization process but might require more memory and can be slower to converge.

**Algorithms:**

**LSTM Model:**

LSTM is a type of recurrent neural network (RNN) layer used for sequence data.

Our model consists of **two LSTM layers**, one dropout layer, and a final dense layer with sigmoid activation for binary classification.

he first LSTM layer processes the input sequence and returns a sequence of outputs. The second LSTM layer takes the output sequence of the first layer and produces a single output.

A dropout layer is a regularization technique used to prevent overfitting in neural networks.

In this model, there is one dropout layer. It randomly drops a fraction of the input units (neurons) during training, which helps to prevent the model from relying too heavily on any single neuron and improves its generalization.

The final layer in the neural network is a dense layer, which is also known as a fully connected layer.

This dense layer has a single neuron, which is common in binary classification problems. The sigmoid activation function is applied to this neuron.

The sigmoid activation function squashes the output into the range [0, 1], making it suitable for binary classification where the goal is to predict a binary outcome (e.g., "attrition" or "no attrition").

The output value from this neuron will represent the probability that an employee will experience attrition, and a threshold (usually 0.5) is applied to classify whether an employee is likely to leave (1) or not (0).

n the provided code, the model is trained for 50 epochs, which means it will iterate over the entire training dataset 50 times. The choice of the number of epochs depends on the specific dataset and problem; it's often determined through experimentation and validation.

**Sequential Feed forward Model:**

The term "feedforward" indicates that information flows in one direction through the network, from the input layer to the output layer, without any loops or cycles.

A feedforward neural network consists of multiple layers:

Input Layer: The input layer receives the raw input data. Each neuron in this layer corresponds to a feature in the input data.

Hidden Layers: These are one or more layers located between the input and output layers. Neurons in hidden layers do not directly interact with the external environment but rather process and transform the information.

Output Layer: The output layer produces the network's final predictions. The number of neurons in this layer depends on the nature of the problem, e.g., one neuron for binary classification or multiple neurons for multiclass classification.

Each layer consists of interconnected neurons (also known as nodes or units). Neurons in adjacent layers are connected by weighted connections. These weights determine the strength of the connections. Each neuron applies an activation function to its input. Common activation functions include the rectified linear unit (ReLU), sigmoid, and hyperbolic tangent (tanh). Activation functions introduce non-linearity into the model.

The process of computing the output of the network from the input data is known as forward propagation. Input features are multiplied by weights, passed through activation functions, and the results are passed to the next layer.

**Single Layer Perceptron Model:**

A Single Layer Perceptron (SLP) is one of the simplest forms of artificial neural networks and a linear binary classifier. It consists of a single layer of artificial neurons (perceptrons) and is used for binary classification tasks. A **perceptron** is the fundamental building block of a Single Layer Perceptron model. It takes multiple inputs (features) and combines them with weighted connections.

The **activation function** of a perceptron is typically a step function, such as the Heaviside step function. Each input feature is associated with a weight, which is learned during the training process. Weights determine the importance of each feature.

The **bias** is a learned constant term that shifts the decision boundary

The perceptron makes binary decisions by separating the feature space into two regions. These regions are determined by the decision boundary, which is a hyperplane in the feature space.

In our code, Perceptron model is created and trained with a maximum of 2000 iterations.

**Bagging Model:**

Bagging is an ensemble technique that aims to improve the stability and accuracy of a machine learning model by combining multiple instances of the same base model.

Bagging is an ensemble learning technique that combines multiple instances of a base classifier (in this case, a Random Forest) to improve predictive performance.

Random subsets of the training data (with replacement) are created, and a base model is trained on each of these subsets.

Predictions from each base model are aggregated, typically by majority voting (for classification) or averaging (for regression).

Bagging is effective in reducing overfitting and increasing the model's robustness.

It's widely used with decision tree-based models, such as Random Forest, to create a more powerful ensemble learner. In this code, a Random Forest classifier is used as the base model within the BaggingClassifier.

Bagging is useful when the base model has high variance, and it helps reduce this variance by taking an ensemble of multiple models trained on different subsets of data.

The BaggingClassifier is created with the base classifier, the number of base classifiers to use (controlled by n\_estimators), and a random state for reproducibility.

**Boosting Model:**

Boosting is an ensemble machine learning technique that combines multiple weak learners (e.g., decision trees) to create a strong learner. It is based on the idea of learning from mistakes.

In boosting, each base model (e.g., decision tree) is trained sequentially, and the focus is on the samples that were misclassified by the previous models. These misclassified samples are given more weight, and the next model tries to classify them correctly.

In our code, Categorical columns are one-hot encoded using pd.get\_dummies. This converts categorical variables into binary (0 or 1) columns for each category, and drop\_first=True omits the first category to avoid multicollinearity.

An **XGBoost** classifier is initialized with various hyperparameters:

**booster**: Gradient boosting method.

**max\_depth**: Maximum depth of the trees.

**learning\_rate**: Step size shrinkage to prevent overfitting.

**n\_estimators**: Number of boosting rounds.

**subsample and colsample\_bytree**: Fraction of samples and features used per tree.

**objective**: The learning task, which is binary classification in this case.

**Stacking Model:**

Stacking, or stacked generalization, is an ensemble machine learning technique that combines predictions from multiple base models (or learners) to create a more powerful meta-learner.

The idea is to leverage the strengths of different base models and have a meta-learner that learns how to optimally combine their predictions.

In this code, two base classifiers (Random Forest and Gradient Boosting) are used to make predictions. The predictions of these base classifiers are then combined using a Logistic Regression meta-classifier.

A StackingClassifier is created. It combines the predictions of the base classifiers (defined in base\_classifiers) using the meta-classifier (defined in final\_estimator).

**Implementation:**

Every python file is supposed to be run individually while the backend of website running should be done by activating **app.py** file and follow the link that is shown in console.