MACHINE LEARNING

UNLOCKING INSIGHTS FOR HEART FAILURE PREDICTION

![](data:image;base64,)

WWW.LINKEDIN.COM/IN/TARUN2K3

# What is machine learning?

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 being explicitly programmed. The fundamental idea behind machine learning is to allow computers to learn from data and improve their performance over time.

There are three main types of machine learning:

![](data:image;base64,)

Supervised Learning

Unsupervised Learning

Reinforcement Learning

Algorithms are trained on labeled datasets, where input data is paired with corresponding output labels.

Classification (assigning labels) and regression (predicting continuous output).

In the real-world, supervised learning can be used for Risk Assessment, Image classification, Fraud Detection, spam filtering.

Algorithms operate on unlabeled data to inherent patterns or structures.

Clustering (grouping similar data points) and dimensionality reduction.

In the real-world, unsupervised learning can be used for Customer Segmentation, Principal Component Analysis (PCA), Anomaly Detection.

An agent learns decision-making by interacting with an environment, receiving feedback in the form of rewards or penalties.

Learn a strategy or policy maximizing cumulative rewards over time.

Game Playing – AlphaGo, Robotics - Autonomous Navigation, Recommendation Systems.

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

![](data:image;base64,)

Let Us Dive into an Experiment: Predicting Heart Failure with Machine Learning!

![](data:image;base64,)

To unravel patterns within the “heart\_failure\_clinical\_records\_dataset.csv” and revolutionize predictive insights in cardiovascular health.

**Experiment Overview:**

![](data:image;base64,)

**Dataset Exploration:**

Uncover the hidden gems within the clinical records dataset, featuring patient information, medical history, and health metrics.

![](data:image;base64,)

**Algorithms on Stage:**

**Support Vector Machines (SVM):** The precision of hyperplanes!

**Logistic Regression:** Unleashing logistic functions for probability modeling!

**Decision Tree Classifier:** Navigating data with tree structures!

**Random Forest Classifier:** Resembling the power of decision trees!

**K-Nearest Neighbors (KNN):** Connecting predictions based on proximity!

Let the Experiment Begin….

**Methodology:**

* **Data Preprocessing:**
* Addressed missing values.
* Standardized or normalized numerical features.
* One-hot encoded categorical variables.
* **Splitting Data:**
* 80% for training, 20% for testing.
* **Model Training:**
* Each algorithm trained on the training set.
* **Performance Evaluation:**
* Metrics used: Precision, Recall, Accuracy. Evaluated models on the test set

Stay Tuned for Results —>

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# Support Vector Machines (SVM)

Support Vector Machines (SVM) are supervised learning algorithms used for classification and regression tasks, particularly effective when dealing with non-linearly separable data.

## Key Concepts:

* **Hyperplane:**
* SVM identifies the optimal hyperplane that maximizes the margin between classes in the feature space.
* Especially useful for scenarios where linear separation is not feasible.
* **Kernel Trick:**
* SVM employs the kernel trick to handle non-linear relationships in data by mapping it into a higher-dimensional space.
* Enables SVM to capture complex patterns and make accurate predictions.
* **Support Vectors:**
* Support vectors are the data points closest to the hyperplane.
* They play a pivotal role in determining the position and orientation of the optimal hyperplane.
* **C Parameter:**
* A smaller C creates a larger margin but may allow for some misclassifications, while a larger C results in a smaller margin with fewer misclassifications.

![](data:image;base64,)

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# Logistic Regression

Logistic Regression is a widely used supervised learning algorithm for binary and multi-class classification tasks. Despite its name, it is used for classification, not regression. Logistic Regression models the probability that a given input belongs to a particular class.

## ➡ Multi-class Logistic Regression:

Extends the binary logistic regression to handle multiple classes using techniques like one-vs-rest.

## Key Concepts:

* **Sigmoid Function:**
* Logistic Regression uses the sigmoid (logistic) function to map any real-valued number to the range [0, 1].
* The sigmoid function ensures that the output can be interpreted as probability.
* **Decision Boundary:**
* The algorithm creates a decision boundary based on learned coefficients and features.
* For binary classification, the decision boundary separates data points into two classes.
* **Maximum Likelihood Estimation:**
* Logistic Regression maximizes the likelihood function to find the optimal parameters (weights) that best fit the observed data.

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# Decision Tree Classifier

A Decision Tree Classifier is a versatile supervised learning algorithm used for both classification and regression tasks. It makes decisions by recursively splitting the dataset based on feature conditions until a stopping criterion is met, forming a tree-like structure of decisions.

## Key Concepts:

* **Node Splitting:**
* The algorithm selects the most informative feature to split the data at each node.
* The goal is to maximize information gain (for classification) or variance reduction (for regression).
* **Decision Nodes:**
* Nodes in the tree represent decisions based on feature conditions.
* Each decision node splits the data into subsets, guiding the traversal of the tree.
* **Leaf Nodes:**
* Leaf nodes contain the final predicted output or class label.
* The algorithm assigns the majority class for classification tasks or the mean value for regression tasks.
* **Entropy and Information Gain:**
* For classification, Decision Trees use entropy to measure impurity.
* Information gain is the reduction in entropy achieved by a split and guides the tree construction.

![](data:image;base64,)

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# Random Forest Classifier

The Random Forest Classifier is an ensemble learning method based on Decision Trees. It constructs a multitude of Decision Trees during training and outputs the mode of the classes (classification) or the mean prediction (regression) of the individual trees.

## Key Concepts:

* **Ensemble of Trees:**
* Random Forest builds multiple Decision Trees independently during training.
* Each tree is trained on a random subset of the data, and features are randomly selected for each split.
* **Voting Mechanism:**
* For classification tasks, the mode (most frequent class) of the predictions from individual trees is taken as the final output.
* For regression, the mean prediction from all trees is used.
* **Bootstrap Aggregating (Bagging):**
* Random Forest employs bagging, a technique where each tree is trained on a bootstrap sample (randomly sampled with replacement) from the original dataset.
* **Feature Randomness:**
* At each split, a random subset of features is considered, preventing individual trees from dominating the ensemble.
* Reduces overfitting and increases robustness.

![](data:image;base64,)

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# K-Nearest Neighbors (KNN) Classifier

K-Nearest Neighbors (KNN) is a simple and intuitive supervised learning algorithm used for classification and regression tasks. It makes predictions based on the majority class or average value of the k-nearest neighbors in the feature space.

## Key Concepts:

* **Nearest Neighbors:**
* KNN classifies data points based on the majority class or average value of their k-nearest neighbors in the feature space.
* The distance metric (Euclidean, Manhattan, etc.) determines “closeness.”
* **Hyperparameter ‘k’:**
* ‘k’ represents the number of neighbors considered for classification.
* Small ‘k’ values lead to more flexible models but can be sensitive to noise. Larger ‘k’ values provide smoother decision boundaries.
* **Decision Rule:**
* For classification, the majority class among the neighbors determines the predicted class.
* For regression, the average of the neighbors’ values is taken.
* **Non-Parametric:**
* KNN is a non-parametric algorithm, meaning it does not make explicit assumptions about the underlying data distribution.

![](data:image;base64,)

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

# Choosing the Right Algorithms for Prediction: A Strategic Decision

![](data:image;base64,)

![](data:image;base64,)

In our mission to understand heart problems using the “heart\_failure\_clinical\_records\_dataset,” we chose some Machine learning algorithms to help us out.

![](data:image;base64,)

<—–Let see some coding part—–>

![](data:image;base64,)

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3

In [ ]:

# NumPy is often used for numerical operations  
# Pandas is commonly used for data cleaning, analysis, and exploration with tabular data  
import numpy as np  
import pandas as pd

## Loading Data

In [ ]:

data df = pd.read\_csv("heart\_failure\_clinical\_records\_dataset.csv") #load the dataset  
data df head() #show the first 5 rows from the dataset

Out[ ]:

age

anaemia

creatinine\_phosphokinase

diabetes

ejection\_fraction

high\_blood Pressure

platelets

serum\_creatinine

serum\_sodium

se

0

75.0

0

582

0

20

1

265000.00

1.9

130

1

55.0

0

7861

0

38

0

263358.03

1.1

136

2

65.0

0

146

0

20

0

162000.00

1.3

129

3

50.0

1

111

0

20

0

210000.00

1.9

137

4

65.0

1

160

1

20

0

327000.00

2.7

116

In [ ]:

#checking if there is any inconsistency in the dataset  
#as we see there are no null values in the dataset, so the data can be processed  
data df.info()

<class 'pandas.core.frame Frame'>  
RangeIndex: 299 entries, 0 to 298  
Data columns (total 13 columns):  
# Column Non-Null Count Dtype  
--- ------ -------------- -----  
0 age 299 non-null float64  
1 anaemia 299 non-null int64  
2 creatinine\_phosphokinase 299 non-null int64  
3 diabetes 299 non-null int64  
4 ejection\_fraction 299 non-null int64  
5 high\_blood Pressure 299 non-null int64  
6 platelets 299 non-null float64  
7 serum\_creatinine 299 non-null float64  
8 serum\_sodium 299 non-null int64  
9 sex 299 non-null int64  
10 smoking 299 non-null int64  
11 time 299 non-null int64  
12 DEATH\_EVENT 299 non-null int64  
dtypes: float64(3), int64(10)  
memory usage: 30.5 KB

## Visualizing data

In [ ]:

import seaborn as sns  
import matplotlib.pyplot as plt  
  
# Select features for the scatter plot  
selected\_features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection\_fraction',  
 'high\_blood Pressure', 'platelets', 'serum\_creatinine', 'serum\_sodium', 'sex', 'smoking', 'time']  
  
# Set up subplots  
fig, axes = plt.subplots(nrows=len(selected\_features), ncols=2, figsize=(15, 2 \* len(selected\_features)))  
  
# Plot scatter plots for each feature against 'DEATH\_EVENT'  
for i, feature in enumerate(selected\_features):  
 # Scatter plot for feature vs 'DEATH\_EVENT' (0)  
 sns.scatterplot(x=feature, y='age', hue='DEATH\_EVENT', data=data df, ax=axes[i, 0], palette='viridis',  
 alpha=0.7)  
 axes[i, 0].set\_title(f'Scatter Plot of {feature} vs age')  
 axes[i, 0].set\_xlabel(feature)  
 axes[i, 0].set\_ylabel('age')  
  
 # Scatter plot for feature vs 'DEATH\_EVENT' (1)  
 sns.scatterplot(x=feature, y='serum\_creatinine', hue='DEATH\_EVENT', data=data df, ax=axes[i, 1],  
 palette='viridis', alpha=0.7)  
 axes[i, 1].set\_title(f'Scatter Plot of {feature} vs serum\_creatinine')  
 axes[i, 1].set\_xlabel(feature)  
 axes[i, 1].set\_ylabel('serum\_creatinine')  
  
# Adjust layout  
plt.tight\_layout()  
plt.show()

![](data:image;base64,)

# Support vector machine (SVM)

In [ ]:

fromsklearn.model\_selectionimport train\_test\_split  
fromsklearn.svmimport SVC  
  
# Select features  
selected\_features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection fraction',  
 'high-blood-pressure', 'platelets', 'serum-creatinine', 'serum-sodium', 'sex', 'smoking', 'time']  
  
# Prepare the data for SVM  
X = data DF[selected\_features]  
y = data DF['DEATH\_EVENT']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create an SVM model  
model =SVC()  
  
# Train the model  
model.fit(X\_train, y\_train)  
  
# Evaluate the model  
accuracy =model.score(X\_test, y\_test)  
print(f"SVM Accuracy: {accuracy:.2f}")  
  
# Randomly sample rows from the Data Frame  
random\_sample =data DF[selected\_features].sample(n=1, random\_state=42)  
  
# Make a prediction  
y\_pred =model.predict(random\_sample)  
print("Predicted DEATH\_EVENT: ", y\_pred[0])  
  
SVM Accuracy: 0.58  
Predicted DEATH\_EVENT: 0

# Logistic Regression

In [ ]:

import pandas as pd  
fromsklearn.model\_selectionimport train\_test\_split  
fromsklearn.linear\_modelimport LogisticRegression  
fromsklearn.metricsimport accuracy\_score, confusion\_matrix, classification\_report  
  
# Load the dataset  
data DF =pd.read\_csv('heart\_failure\_clinical\_records\_dataset.csv')  
  
# Select features for the scatter plot  
selected\_features =['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection\_fractions',  
 'high-blood-pressure', 'platelets', 'serum-creatinine', 'serum-sodium', 'sex', 'smoking', 'time']  
  
# Prepare the data for Logistic Regression  
X =data DF[selected\_features]  
y =data DF['DEATH\_EVENT']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test =train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a Logistic Regression model  
logistic\_model =LogisticRegression()  
  
# Train the model  
logistic\_model.fit(X\_train, y\_train)  
  
# Evaluate the model  
y\_pred\_logistic =logistic\_model Predict(X\_test)  
  
# Calculate accuracy  
accuracy\_logistic =accuracy\_score(y\_test, y\_pred\_logistic)  
print(f"Logistic Regression Accuracy: {accuracy\_logistic:.2f}")  
  
# Confusion Matrix  
conf\_matrix =confusion\_matrix(y\_test, y\_pred\_logistic)  
print("Confusion Matrix:")  
print(conf\_matrix)  
  
# Classification Report  
classification\_rep =classification\_report(y\_test, y\_pred\_logistic)  
print("\nClassification Report:")  
print(classification\_rep)  
  
# Randomly sample rows from the Data Frame  
random\_sample =data DF[selected\_features].sample(n=1, random\_state=42)  
  
# Make a prediction using Logistic Regression  
pred\_logistic\_sample =logistic\_model Predict(random\_sample)

print("\nLogistic Regression Predicted DEATH\_EVENT:", pred\_logistic\_sample[0])  
  
Logistic Regression Accuracy: 0.80  
Confusion Matrix:  
[[33 2]  
[10 15]]  
  
Classification Report:

precision recall f1-score support 0 0.77 0.94 0.85 35 1 0.88 0.60 0.71 25

accuracy 0.80 60  
macro avg 0.82 0.77 0.78 60  
weighted avg 0.82 0.80 0.79 60  
  
Logistic Regression Predicted DEATH\_EVENT: 0  
  
## DecisionTreeClassifier  
  
In [ ]:  
  
```python  
import pandas as pd  
fromsklearn.model\_selection import train\_test\_split  
fromsklearn.tree import DecisionTreeClassifier  
  
# Load the dataset  
data df = pd.read\_csv('heart\_failure\_clinical\_records\_dataset.csv')  
data df .head()  
  
# Select features for the scatter plot  
selected\_features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection\_fraction',  
 'high\_blood Pressure', 'platelets', 'serum\_creatinine','serum\_sodium', 'sex', 'smoking', 'time']  
  
# Prepare the data for decision tree  
X = data df [selected\_features]  
y = data df ['DEATH\_EVENT']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a Decision Tree model  
model = DecisionTreeClassifier()  
  
# Train the model  
model.fit(X\_train, y\_train)  
  
# Evaluate the model  
accuracy = model.score(X\_test, y\_test)  
print(f"Decision tree Accuracy: {accuracy:.2f}")  
  
# Randomly sample rows from the Data Frame  
random\_sample = data df [selected\_features].sample(n=1, random\_state=42)  
  
# Make a prediction  
y\_pred = model predict(random\_sample)  
print("Predicted DEATH\_EVENT:", y\_pred [0])

Decision tree Accuracy: 0.65 Predicted DEATH\_EVENT: 1

## RandomForestClassifier

In [ ]:

import pandas as pd  
fromsklearn.model\_selection import train\_test\_split  
fromsklearn.ensemble import RandomForestClassifier # Import RandomForestClassifier  
  
# Select features for the scatter plot  
selected\_features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection\_fraction',  
 'high\_blood pressure', 'platelets', 'serum\_creatinine','serum\_sodium', 'sex', 'smoking', 'time']  
  
# Prepare the data for Random Forest  
X = data df [selected\_features]  
y = data df ['DEATH\_EVENT']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a Random Forest model  
model = RandomForestClassifier(random\_state=42) # Use RandomForestClassifier  
  
# Train the model

model.fit(X\_train, y\_train)  
  
# Evaluate the model  
accuracy = model.score(X\_test, y\_test)  
print(f"RandomForestClassifier Accuracy: {accuracy:.2f}")  
  
# Randomly sample rows from the Frame  
random\_sample = data DF[selected\_features].sample(n=1, random\_state=42)  
  
# Make a prediction  
y\_pred = model predict(random\_sample)  
print("Predicted DEATH\_EVENT: ", y\_pred[0])

RandomForestClassifier Accuracy: 0.75  
Predicted DEATH\_EVENT: 0

## KNeighborsClassifier

In [ ]:

import pandas as pd  
from sklearn.model\_selection import train\_test\_split  
fromsklearn\_neighbors import KNeighborsClassifier # Import KNeighborsClassifier  
fromsklearn.preprocessing import StandardScaler  
  
# Select features for the scatter plot  
selected\_features = ['age', 'anaemia', 'creatinine\_phosphokinase', 'diabetes', 'ejection\_fractions',  
 'high\_blood Pressure', 'platelets', 'serum\_creatinine', 'serum\_sodium', 'sex', 'smoking', 'time']  
  
# Prepare the data for K-Nearest Neighbors  
X = data df[selected\_features]  
y = data df['DEATH\_EVENT']  
  
# Split the data into training and testing sets  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
# Create a K-Nearest Neighbors model  
k = 5 # You can choose the value of k based on your requirements  
model = KNeighborsClassifier(n\_neighbors=k) # Use KNeighborsClassifier instead of RandomForestClassifier  
  
# Train the model  
model.fit(X\_train, y\_train)  
  
# Evaluate the model  
accuracy = model.score(X\_test, y\_test)  
print(f"K-Nearest Neighbors model Accuracy: {accuracy:.2f}")  
  
# Randomly sample rows from the Frame  
gain\_sample = data df[selected\_features].sample(n=1, random\_state=42)  
  
# Make a prediction  
y\_pred = model predict(random\_sample)  
print("Predicted DEATH\_EVENT: ", y\_pred[0])

K-Nearest Neighbors model Accuracy: 0.53  
Predicted DEATH\_EVENT: 1

Loading [MathJax]/jax/output/CommonHTML/fontes/TeX/fontdata.js

# General Conclusions

In the context of heart failure prediction from the same dataset, **Logistic Regression** emerges as the top performer with an accuracy of **80%**, demonstrating its effectiveness. **Random Forest Classifier** follows closely with **75%** accuracy, displaying robustness in handling complex relationships. **Decision Tree** performs moderately at **65%**. **Support Vector Machines (SVM)** achieved **58%**, showing potential sensitivity to the data’s linear separability. **K-Nearest Neighbors (KNN)** trails with **53%**, suggesting the need for parameter adjustments. While **Logistic Regression** and **Random Forest** excel, further exploration and parameter tuning could enhance the performance of SVM, Decision Tree, and KNN in this specific prediction task.

![](data:image;base64,)

Connect with me on LinkedIn!

www.linkedin.com/in/tarun2k3