

Open in app ↗

Sign up

Sign in

Medium

Search

Write

Writing is for everyone. [Register for Medium Day](#)

Select the right Classification Model for Your Data

# Exploring Classification Algorithms: Guide to Select the Right Model for Your Data



Sakshi Babbar

Follow

9 min read · Jun 28, 2023



6



In the realm of classification problems, selecting the most appropriate algorithm plays a vital role in achieving accurate predictions. With numerous classification algorithms available, it is essential to understand

how to identify the best model for a given problem. This article aims to guide readers through this process using Python.

The article explores a comprehensive suite of classification algorithms, each with its own strengths, weaknesses, and interpretability characteristics. By evaluating and comparing these algorithms using appropriate model evaluation techniques and performance metrics, the best-performing model for a given problem can be determined.

To assist in choosing the most suitable algorithm, let's delve into the considerations and characteristics of each algorithm. The table below provides an overview of the algorithms, their use cases, strengths, weaknesses, interpretability ratings, and the factors that contribute to their interpretability:

Algorithm	Use Case	Strengths	Weaknesses	Interpretability	Interpretability Influencers
Logistic Regression	Binary or multi-class classification, linear decision boundary	Simple and interpretable, handles large feature spaces	Assumptions of linearity, sensitive to outliers	High	1. Linearity 2. Simplicity 3. Feature Space Handling
Naive Bayes	Text classification, spam filtering	Fast and scalable, handles high-dimensional data	Assumes independence of features	Moderate	1. Speed 2. High Dimensions 3. Independence
Decision Trees	Data with non-linear relationships, feature importance	Non-linear decision boundaries, interpretable	Prone to overfitting, can be unstable	Moderate	1. Non-linearity 2. Feature Importance 3. Stability
K Nearest Neighbors	Image recognition, recommendation systems	Non-parametric, captures local patterns	Sensitive to irrelevant features, computationally expensive	Low	1. Non-parametric 2. Local Patterns 3. Computational expensive
Support Vector machines	Text classification, image recognition	Effective in high-dimensional spaces, handles complex data	Less effective with large datasets, parameter tuning required	Low	1. High Dimensions 2. Complexity 3. Parameter Tuning

Algorithm Comparison: Assessing Interpretability and Performance in Classification

Considering these factors and characteristics, readers can make an informed decision when selecting the most appropriate algorithm for their specific problem. Each algorithm offers unique capabilities and trade-offs, and the interpretability influencers shed light on the factors that contribute to their interpretability.

Let's now dive into the world of classification algorithms and discover how to find the best model for your given problem!

## 1. Model Evaluation

- Cross-validation is employed to evaluate classification models.
- The data is divided into  $k$  subsets or folds for training and evaluation.
- Each model is trained on  $(k-1)$  subsets and evaluated on the remaining subset.
- This process is repeated  $k$  times to gain better confidence in the model's performance on unseen data.

## 2. Model Performance Metrics

- Several performance metrics are used to assess classification models.
- Metrics include accuracy, precision, recall, F1 score, confusion matrix, and ROC curve.
- Accuracy measures the overall correctness of the model's predictions.
- Precision quantifies the proportion of correctly predicted positive instances.
- Recall calculates the proportion of true positive instances identified by the model.
- F1 score combines precision and recall to provide a balanced measure.
- The confusion matrix displays the true positive, true negative, false positive, and false negative predictions.
- ROC curve plots the true positive rate against the false positive rate, allowing analysis of the model's trade-offs between sensitivity and specificity.

These model evaluation techniques and performance metrics aid in assessing and comparing the performance of different classification

algorithms and ultimately selecting the best model for a given problem.

### 3. Dataset Description

The dataset used in this article is sourced from the medical domain and is known as the “Heart Disease” dataset. It is obtained from the UCI Machine Learning Repository and can be accessed at [UCI Repository](#).

Comprising 303 instances, the “Heart Disease” dataset consists of 14 features representing various symptoms. The objective of this dataset is to predict the presence or absence of heart disease, making it a binary classification problem. The target variable takes the value of 1 to indicate the presence of heart disease and 0 to indicate its absence.

By employing this dataset, the article demonstrates the application of classification algorithms in the medical field. It allows us to showcase the effectiveness of different algorithms in accurately identifying individuals at risk of heart disease based on their symptom profiles.

It’s important to note that while the focus of this article revolves around the methodology of selecting the best classification algorithm, the concepts and techniques discussed can be applied to diverse datasets and problem domains.

### 4. Hands-on in Python

The steps below provide a comprehensive pipeline for implementing and evaluating classification algorithms on the heart disease dataset, aiding in the identification of the best-performing model.

#### Step 1: Import libraries

Begin by importing the necessary Python libraries such as scikit-learn, numpy, pandas, and seaborn to facilitate the implementation of classification algorithms and analysis of results.

```
# importing Pandas for data manipulation
import pandas as pd
import numpy as np
# importing classification models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
# importing matplotlib for visualization
from matplotlib.pyplot import boxplot
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
# importing method to perform cross validation
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
# importing methods for model evaluation
from sklearn import metrics
```

## Step 2: Load Dataset

Get Sakshi Babbar's stories in your inbox

Join Medium for free to get updates from this writer.

Enter your email

Subscribe

Load the heart disease dataset using pandas or similar libraries to access the data and prepare it for training and testing the classification models.

```
dataset = pd.read_csv("/content/heart.csv")
dataset.info()
```

### Step 3: Building training and test sets

Split the dataset into training and test sets, typically with an 80:20 ratio, where 80% of the data is used for training the models and 20% is reserved for evaluating their performance.

```
# My_data contains all data points from My_data set from first feature to 12th feature
My_data = dataset.iloc[:,0:13]
# My_target contains class information which is 13th feature in the data set of My_data
My_data_target=dataset.iloc[:,13]
X_train, X_test, Y_train, Y_test = train_test_split(My_data, My_data_target, test_size=0.2, random_state=42)
```

### Step 4: Building a suit of Classification Algorithms

Create a suite of classification algorithms by instantiating and configuring instances of each algorithm, such as logistic regression, Naive Bayes, decision tree, K-nearest neighbor, and support vector machine.

```
# creating a empty list
List_Classification_Models = []
# adding to list the instance of Logistic regression model
List_Classification_Models.append(('LR', LogisticRegression()))
# adding to list the instance of KNN model
List_Classification_Models.append(('KNN', KNeighborsClassifier()))
# adding to list the instance of SVM model
List_Classification_Models.append(('SVM', SVC()))
```

```
# adding to list the instance of Naive Bayes model
List_Classification_Models.append(('NaiveBayes', GaussianNB()))
# adding to list the instance of Decision tree
List_Classification_Models.append(('DT', DecisionTreeClassifier()))
```

## Step 5: Applying Cross-validation on Algorithms in Suit

Implement k-fold cross-validation to assess the performance of each algorithm. Train the models using (k-1) folds and evaluate them on the remaining fold. Repeat this process k times, ensuring a different fold is reserved for evaluation each time.

```
# creating empty lists to store results of cross validation and the name of the
Model_Eval_Score = []
Name_of_model = []
# applying cross validation on each algorithm in suit
for name, model_detail in List_Classification_Models:
    # initiating cross validation with 10 iterations
    kfold = KFold(n_splits=10)
    # applying cross validation with 10 iterations on the training data
    CV_Results = cross_val_score(model_detail, X_train, Y_train, cv=kfold)
    # adding result of each iteration in list
    Model_Eval_Score.append(CV_Results)
    # name of the corresponding algorithm is stored as follows
    Name_of_model.append(name)

# creating DataFrame with cross validation results where each row indicates the
# of the model on different cross validation iterations
CV_IterationsBy_model = pd.DataFrame(Model_Eval_Score, index=['LR', 'KNN', 'SVM',
                                                             'DT'])
print("The 10 cross validation results of each classification algorithm are: \n")
# printing the transpose of the data frame so that each column is a unique model
Table_Results_CV= pd.DataFrame(CV_IterationsBy_model.T)
```

Output:

The 10 cross validation results of each classification algorithm are:

	LR	KNN	SVM	NaiveBayes	DT
0	0.833333	0.666667	0.500000	1.000000	0.500000
1	0.666667	0.666667	0.666667	0.666667	0.666667



```

2  0.666667  0.833333  0.500000  1.000000  0.666667
3  0.833333  0.666667  0.666667  0.833333  0.500000
4  0.666667  0.500000  0.333333  1.000000  0.333333
5  0.333333  1.000000  1.000000  0.500000  0.500000
6  0.833333  0.833333  0.500000  0.833333  0.833333
7  0.666667  0.666667  0.666667  0.500000  0.833333
8  1.000000  0.666667  0.666667  0.666667  0.666667
9  1.000000  0.833333  0.666667  0.833333  0.833333

```

The mean performance of each regressor algorithm are:

```

LR          0.750000
KNN         0.733333
SVM         0.616667
NaiveBayes  0.783333
DT          0.633333
dtype: float64

```

The CV\_IterationsBy\_model dataframe captures the accuracy values achieved by each model during 10 iterative runs of cross-validation. For instance, in iteration 1, the Logistic regression model achieved an accuracy of 83%. However, in iteration 2, the accuracy dropped to 66.6%, indicating some variability in its performance. After analyzing the overall results, it can be concluded that Naive Bayes emerges as the best model for the given dataset, exhibiting an average accuracy of 78.3%. This suggests that Naive Bayes consistently performs well across multiple iterations and demonstrates its suitability for this particular dataset.

## Step 6: Applying the Model on Test set

Select the algorithm with the best performance based on cross-validation results and apply it to the test set to make predictions on unseen data.

```

# Fitting Naive Bayes model on the Training data set
NB_model = GaussianNB()
NB_model.fit(X_train, Y_train)
# Getting prediction on train and test sets
NB_model_pred_test= NB_model.predict(X_test)
# Computing Model Accuracy
print("Accuracy:",round(metrics.accuracy_score(Y_test, NB_model_pred_test),2) *
print ("-----")
# Printing confusion matrix
print ("Confusion matrix")
print ("-----")
print(metrics.confusion_matrix(Y_test, NB_model_pred_test))
# Model detailed classification report
target_names = ['class 0', 'class 1']
print ("-----")
print("Classification report", metrics.classification_report(Y_test, NB_model_pr

```

Output:

Accuracy: 78.0 %

-----

Confusion matrix

-----

[[95 20]

[34 94]]

-----

Classification report

precision

recall

f1-score

support

class 0      0.74      0.83      0.78      115

class 1      0.82      0.73      0.78      128

accuracy                      0.78      243

macro avg      0.78      0.78      0.78      243

weighted avg      0.78      0.78      0.78      243

he machine learning model achieved an accuracy of 78% on the test data. Additionally, class-wise precision, recall, and f1-score were computed and recorded. The confusion matrix provides detailed information about true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) for each class. For instance, the Naive Bayes model achieved 95 true positives and 94 true negatives. Therefore, the total number of correct

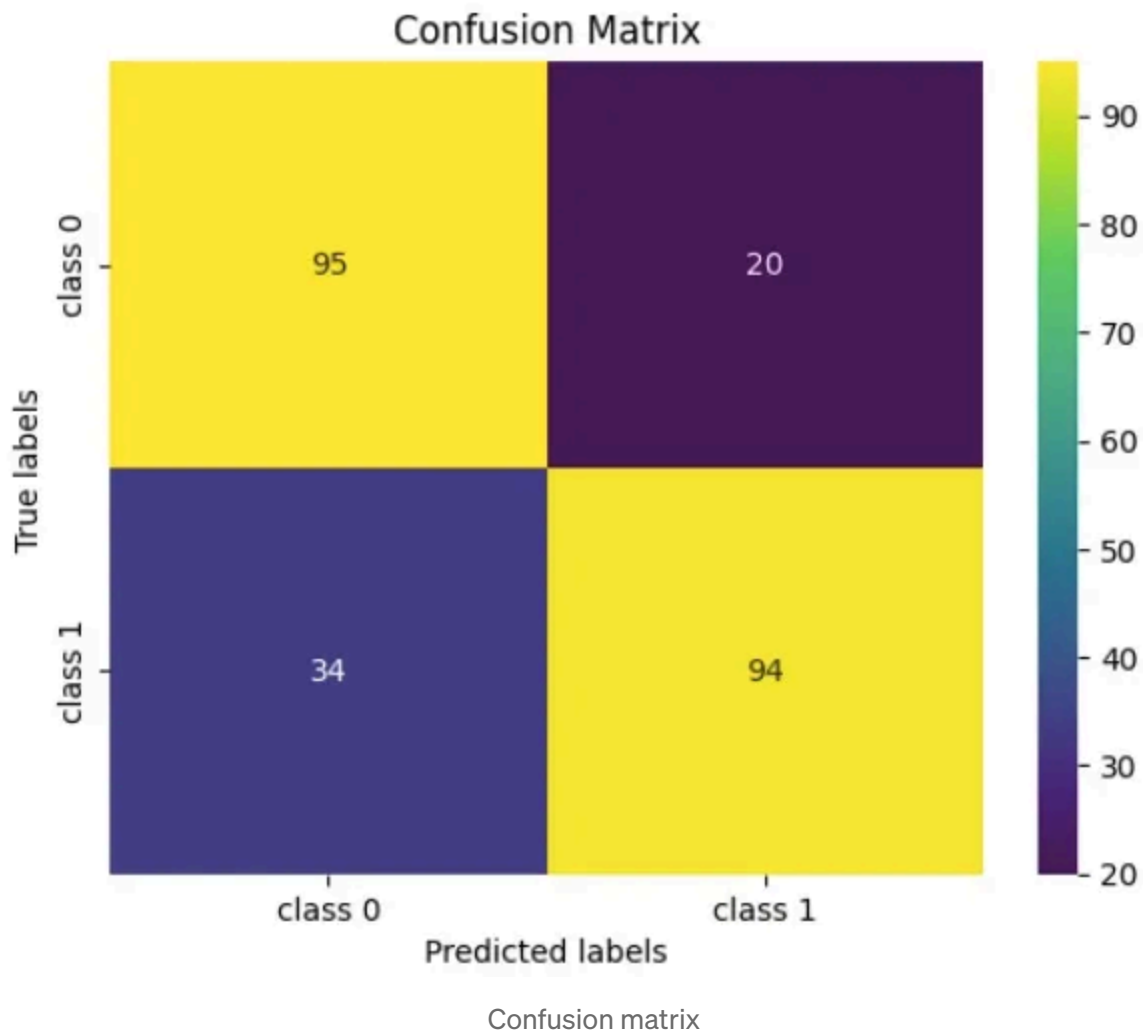
predictions made by the model is 189. Conversely, there were 20 false positives and 34 false negatives, resulting in a total of 54 wrong predictions. To visually interpret the confusion matrix, a heatmap can be created using the seaborn library. The code snippet below demonstrates the process of representing the confusion matrix using seaborn, enhancing visualization and interpretation.

## Step 7: Plotting Confusion Matrix using Seaborn

Visualize the performance of the chosen algorithm by plotting the confusion matrix, which showcases the true positive, true negative, false positive, and false negative predictions using the Seaborn library.

```
ax= plt.subplot()
Confusion_matrix=metrics.confusion_matrix(Y_test, NB_model_pred_test)
sns.heatmap(Confusion_matrix, annot=True, ax = ax, cmap="viridis"); #annot=True
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['class 0', 'class 1']); ax.yaxis.set_ticklabels(['class
```

Output:



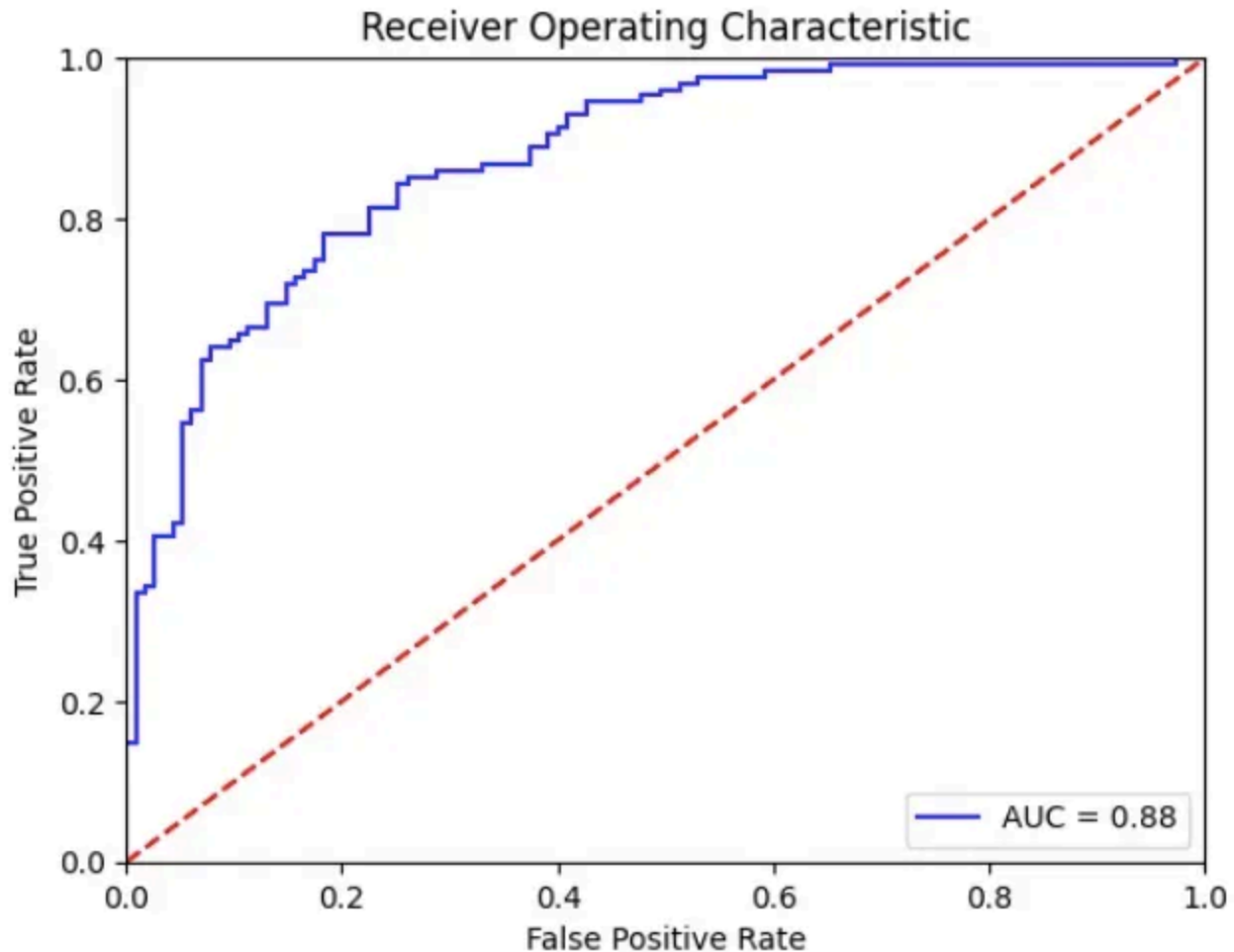
## Step 8: Plotting ROC Curve

Generate the Receiver Operating Characteristic (ROC) curve to analyze the trade-off between the true positive rate and false positive rate of the classification model, providing insights into its performance.

```
NB_model_pred_proba= NB_model.predict_proba(X_test)
preds = NB_model_pred_proba[:,1]
fpr, tpr, threshold = metrics.roc_curve(Y_test, preds)
roc_auc = metrics.auc(fpr, tpr)
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1], 'r--')
plt.xlim([0, 1])
```

```
plt.ylim([0, 1])  
plt.ylabel('True Positive Rate')  
plt.xlabel('False Positive Rate')  
plt.show()
```

Below is the ROC generated



## Conclusion

The process of selecting the most appropriate classification algorithm for a given problem involves careful evaluation and comparison of various algorithms. In this article, a comprehensive exploration of classification algorithms, including logistic regression, Naive Bayes, decision tree, K-nearest neighbor, and support vector regression, has been presented.

By employing cross-validation techniques and performance metrics such as accuracy, precision, recall, F1 score, confusion matrix, and ROC curve, the performance of each algorithm has been assessed. This evaluation process provides valuable insights into the strengths, weaknesses, and interpretability characteristics of the algorithms.

It is evident that decision tree algorithms offer a moderate level of interpretability, making them suitable for situations where understanding the model's decision-making process is crucial. Logistic regression algorithms strike a balance between interpretability and performance. Each algorithm has its own unique attributes, enabling informed decision-making based on the specific requirements of the problem at hand.

The Python implementation presented in this article has provided a practical guide to applying these algorithms. The step-by-step instructions, including importing libraries, loading the heart disease dataset, creating training and test sets, building a suite of algorithms, applying cross-validation, and visualizing performance through confusion matrices and ROC curves, empower readers to confidently select the most appropriate algorithm for their own classification tasks.

By following this approach and leveraging the strengths of various algorithms, data scientists and machine learning practitioners can make informed decisions and achieve accurate predictions in classification problems.

## **Further Resources and Implementation Details**

For more detailed information and access to the complete Python code used in this study, please visit my GitHub page. There, you will find comprehensive documentation and resources that provide a deeper

understanding of the implementation process, including the classification algorithms, model evaluation techniques, and performance metrics. Feel free to explore the code, experiment with different parameters, and adapt it to your specific needs. Visit the [link](#) to access the code and further enhance your understanding of classification algorithm selection and evaluation.

Classification Models

Evaluation Metrics

Choosing Right Model

**Written by Sakshi Babbar**

Follow

7 followers · 2 following

Passionate Machine Learning & Data Science Expert | AWS Specialist | Speaker | Mentor | Committed to driving innovation & empowering through online education


## No responses yet



Write a response

What are your thoughts?

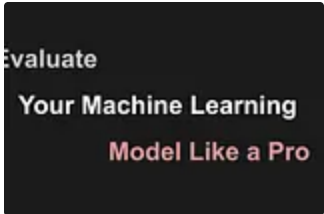
More from Sakshi Babbar

 Sakshi Babbar

Unveiling the Power of Validation:  
Assessing Classification Models

In the field of machine learning, classification models operate through two major steps: learning and testing. The learning...

Jun 26, 2023  3




See all from Sakshi Babbar

Recommended from Medium



 MatMaq

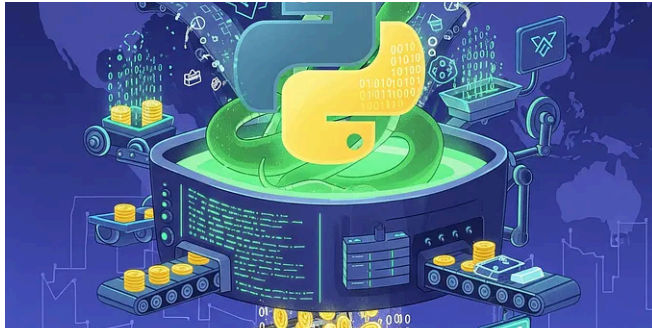
 In Long. Sweet. Valuable. by Ossai Chinedum



## The 90% of ML Teams Are Still Building Models Like It's 2019 (An...

The distributed machine learning approach that's quietly transforming enterprise AI

★ Aug 18 🖱 317 💬 3

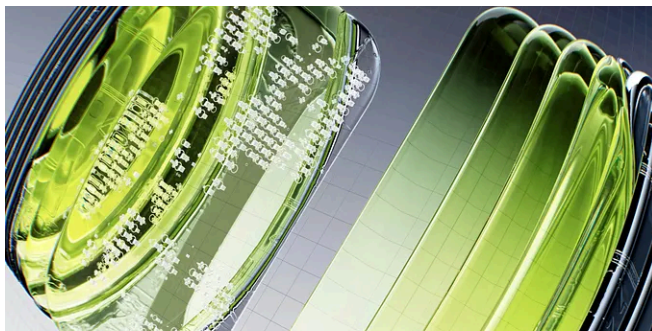


**PY** In Python in Plain English by Suleman Safdar

## The Python Tool I Built in a Weekend That Now Pays My Rent

How I turned a tiny automation script into a paid product using libraries, clean OOP, and ...

★ Aug 11 🖱 2.9K 💬 54



**AI** In Artificial Intelligence in Plain English by Olubusolami S...

## I Finally Understood "Attention is All You Need" After So Long. Here...

It's been almost 2 years since I first encountered the "Attention is all you need"...

## I'll Instantly Know You Used Chat Gpt If I See This

Trust me you're not as slick as you think

★ May 16 🖱 22K 💬 1329



**IE** In Intern Elite by Surbi Karki

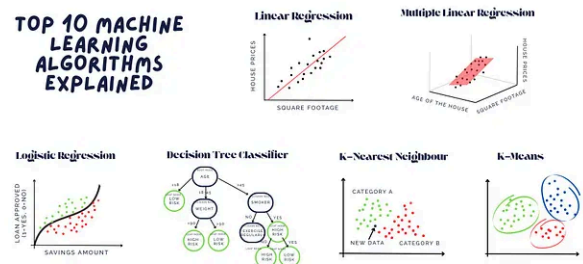
## Day 3 with Scikit-Learn and Matplotlib: From Visualizing Data...

Today's immersion in the world of data science was rich in visual discovery and...

May 20 🖱 2



### TOP 10 MACHINE LEARNING ALGORITHMS EXPLAINED



**LD** In Learning Data by Rita Angelou

## 10 ML Algorithms Every Data Scientist Should Know—Part 1

I understand well that machine learning might sound intimidating. But once you break down...

Jul 12  622  10



Jun 10  33



See more recommendations