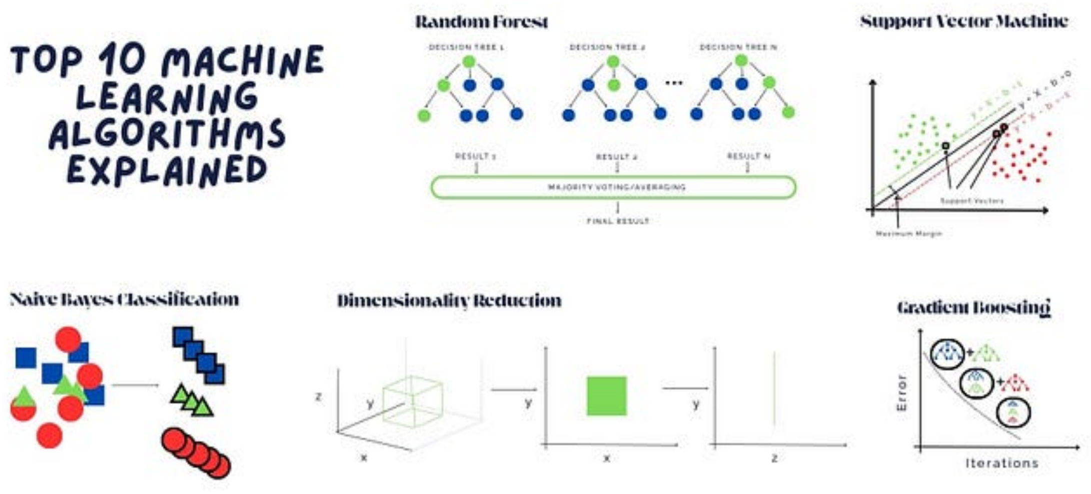
Freedium

< Go to the original



Top 10 Common ML Algorithms Every Data Scientist Should Know (Part 2)

Are you frustrated with Machine Learning? I've put together a simple guide covering the most common ML algorithms to help clear things up.

Rita Angelou

Follow

androidstudio · June 18, 2025 (Updated: June 19, 2025) · Free: No

When I first started learning machine learning, I was completely overwhelmed by all the algorithms out there. Which one should I use? What do they actually do? And how do I know if I'm using them correctly? Sound familiar? If so, you're not alone.

In my previous article I explained 5 ML algorithms using real-life to have a more visual look of how they work.

If you want to review the first 5 or haven't read my previous article yet this is it:

10 ML Algorithms Every Data Scientist Should Know — Part 1

I understand well that machine learning might sound intimidating. But once you break down the common algorithms, you'll...

medium.com

Now, let's quickly start with the other 5 ML algorithm explanations that I owe you.

1. Random Forest: Combines multiple decision trees to make predictions.

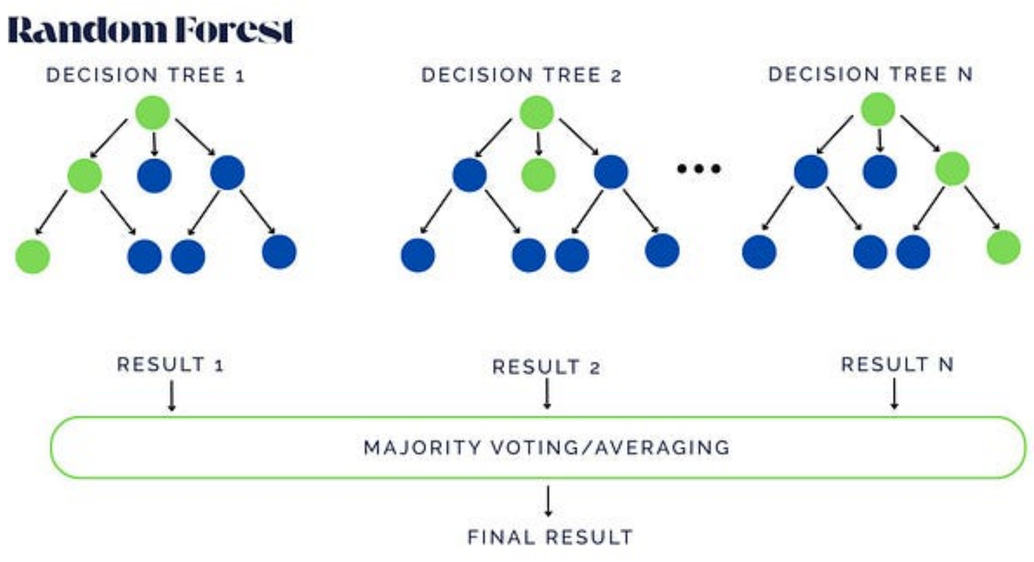


Image created by Author

that's what the Random Forest algorithm is about. It uses the results of multiple Decision Trees to make a combined prediction. It is used both for classification and regression.

Examples:

- Predicting loan defaults
- Classifying diseases
- Ranking product recommendations

Python code example: We are going to use a classic, clean dataset from *scikit-learn.* It is used for **binary classification,** to predict whether a tumor is **malignant (cancerous)** or **benign (non-cancerous)** based on various features.

Features (Independent Variables)

There are **30 numeric features** extracted from digitized images of a breast mass. These features describe characteristics of the cell nuclei present in the image (like mean radius, mean texture, mean perimeter etc.).

Target (Dependent Variable)

- 0 = **Malignant**
- 1 = **Benign**

Copy

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
```

```
# Load dataset
data = load_breast_cancer()
X = data.data           # Features
y = data.target         # Labels (0 = malignant, 1 = benign)

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Initialize the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf.fit(X_train, y_train)

# Make predictions
y_pred = rf.predict(X_test)

# Show the first 5 predictions vs. actual values
for i in range(5):
    print(f"Predicted: {y_pred[i]} — Actual: {y_test[i]}")

# Evaluate the model
print("Model Accuracy:", accuracy_score(y_test, y_pred).round(2))

# Print top 5 important features
feature_importances = pd.Series(rf.feature_importances_, index=data
print(f"\n 5 Most Important Features:  \n{feature_importances.sort
Outcome:

Predicted: 1 — Actual: 1
Predicted: 0 — Actual: 0
Predicted: 0 — Actual: 0
Predicted: 1 — Actual: 1
Predicted: 1 — Actual: 1
Model Accuracy: 0.96

 5 Most Important Features:
worst area              0.153892
worst concave points    0.144663
mean concave points     0.106210
```

What does model accuracy mean?

Our Model Accuracy is 0.96: This means the model correctly classified **96%** of the tumors in the test set.

Can we use Random Forest for regression problems?

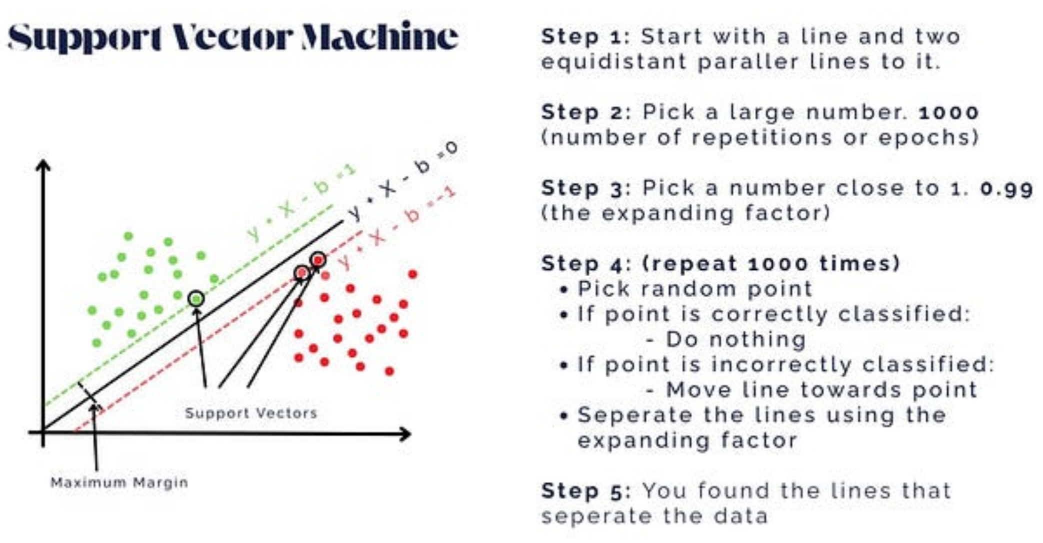
Yes we can. Let's make a python code example for regression using the **California Housing dataset**, which predicts house prices based on various features.

Copy

```
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

# Load the California housing dataset
data = fetch_california_housing()
X = data.data           # Features (e.g., income, population)
y = data.target         # Target (median house value)

# Split into training and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Initialize the Random Forest Regressor
rf_regressor = RandomForestRegressor(n_estimators=100, random_stat

# Train the model
rf_regressor.fit(X_train, y_train)

# Make predictions
y_pred = rf_regressor.predict(X_test)
```

```
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Optional: Print a few predictions
for i in range(5):
    print(f"Predicted: {y_pred[i]:.2f} — Actual: {y_test[i]:.2f}")

print(f"Mean Squared Error: {mse:.2f}")
print(f"R-squared (R²) Score: {r2:.2f}")
Outcome:

Predicted: 0.51 — Actual: 0.48
Predicted: 0.74 — Actual: 0.46
Predicted: 4.92 — Actual: 5.00
Predicted: 2.53 — Actual: 2.19
Predicted: 2.27 — Actual: 2.78
Mean Squared Error: 0.26
R-squared (R²) Score: 0.81
```

2. Support Vector Machine (SVM): Finds the optimal hyperplane that best separates classes



Step 1: Start with a line and two equidistant paraller lines to it.

Step 2: Pick a large number. **1000** (number of repetitions or epochs)

Step 3: Pick a number close to 1. **0.99** (the expanding factor)

Step 4: (repeat 1000 times)
- Pick random point
- If point is correctly classified:
    - Do nothing
- If point is incorrectly classified:
    - Move line towards point
- Seperate the lines using the expanding factor

Step 5: You found the lines that seperate the data

Image created by Author

finding the optimal hyperplane that best separates data points into different classes, maximizing the margin between them.

Examples:

- Face detection in images
- Classifying text documents
- Recognizing digits in images

Python code example: We will use the same dataset as above to create our example in SVM. We are using SVC (Support Vector Classification) from *sklearn* library that is used for classification problems. In simple terms, it finds the **optimal hyperplane** that best separates your data into classes.

Copy

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data          # Features (30 numeric variables)
y = data.target        # Target (0 = malignant, 1 = benign)

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Initialize and train the SVM model
model = SVC(kernel='linear')  # Linear kernel is good for this dat
model.fit(X_train, y_train)

# Make predictions
predictions = model.predict(X_test)
```

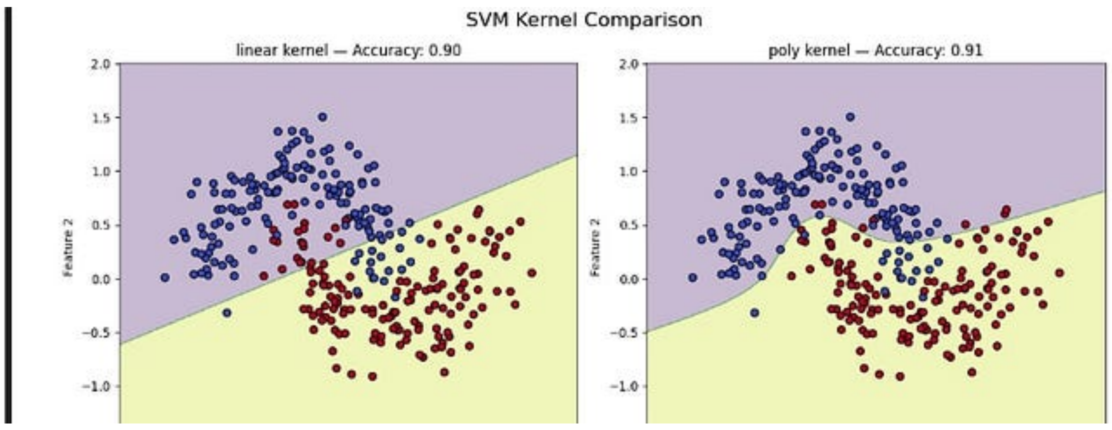
```
for i in range(5):
    print(f"Predicted: {predictions[i]} — Actual: {y_test[i]}")

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, predictions).round(2))
Outcome:

Predicted: 1 — Actual: 1
Predicted: 0 — Actual: 0
Predicted: 0 — Actual: 0
Predicted: 1 — Actual: 1
Predicted: 1 — Actual: 1
Accuracy: 0.96
```

> ## But what happens when my data isn't linearly separable?

There are other kernel types. The most common are:

- 'linear' : When data is linearly seperable
- 'poly' : When data has curved boundaries
- 'rbf' : Most versatile
- 'sigmoid' : Rare in practice

Take a look in the image below so you can have an idea of how each kernel type works.



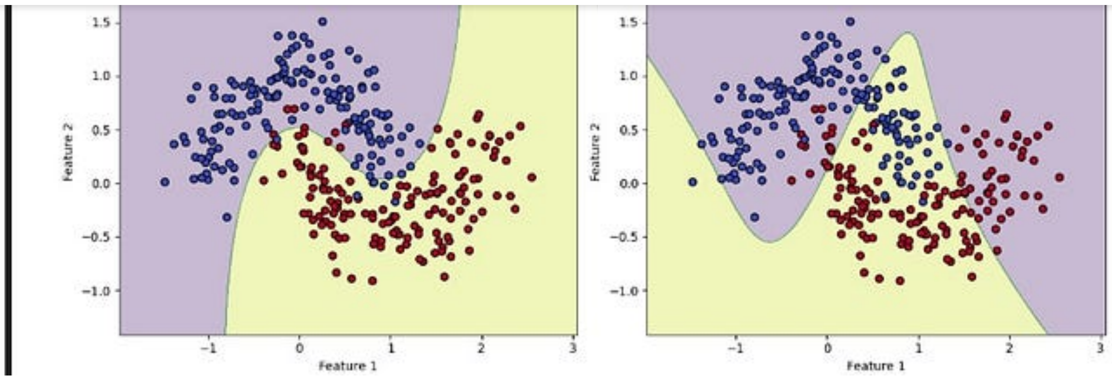


Image created by Author

Can we use SVM for regression problems?

As I said before yes SVM can be used for regression. Instead of SVC (Support Vector Classification) that we used in our code, you can use **SVR (Support Vector Regression)** instead.

3. Naive Bayes: Uses Bayes' Theorem for classification

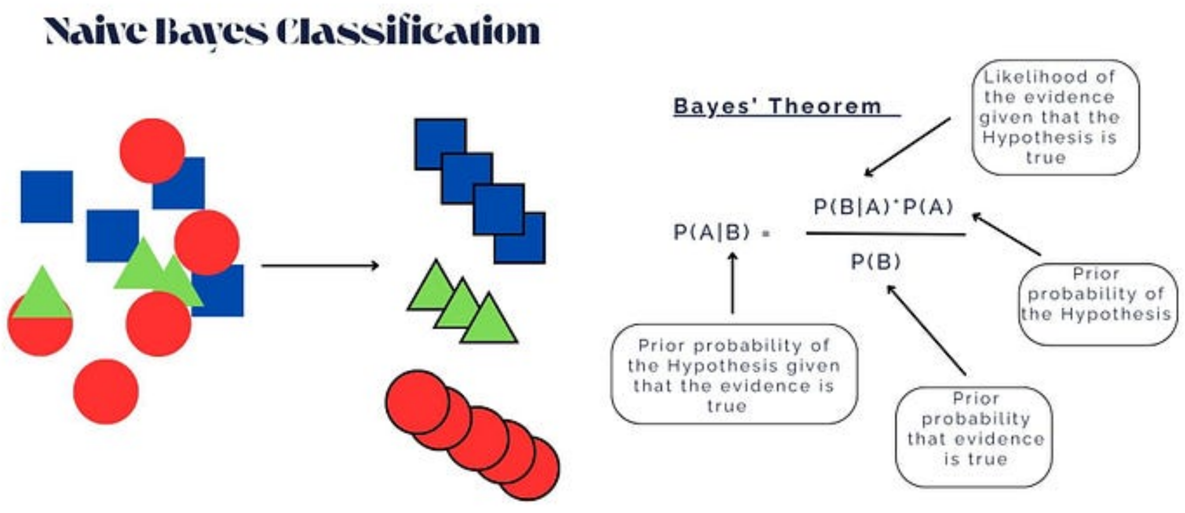

Naive Bayes Classification
Bayes' Theorem
P(A|B) = P(B|A)*P(A) / P(B)
Likelihood of the evidence given that the Hypothesis is true
Prior probability of the Hypothesis
Prior probability of the Hypothesis given that the evidence is true
Prior probability that evidence is true

Image created by Author

The Naive Bayes algorithm is a supervised machine learning method of each other, hence the "naive" assumption.

What is Bayes' Theorem?

Bayes' Theorem calculates the probability of an event based on prior knowledge of conditions related to the event.

Examples:

- Spam filtering
- Sentiment analysis of reviews
- Categorizing news articles

Python code example: Let's use again the breast cancer dataset for our predictions.

Copy

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# Load the Breast Cancer dataset
data = load_breast_cancer()
X = data.data          # Features
y = data.target        # Target labels: 0 = malignant, 1 = benign

# Split into training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Initialize the Gaussian Naive Bayes model
model = GaussianNB()

# Train the model
model.fit(X_train, y_train)
```

```
predictions = model.predict(X_test)

# Show the first 5 predictions vs. actual values
for i in range(5):
    print(f"Predicted: {predictions[i]} — Actual: {y_test[i]}")

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, predictions).round(2))
Outcome:

Predicted: 1 — Actual: 1
Predicted: 0 — Actual: 0
Predicted: 0 — Actual: 0
Predicted: 1 — Actual: 1
Predicted: 1 — Actual: 1
Accuracy: 0.97
```

4. Gradient Boosting Algorithms (GBA): Combines weak models for strong prediction

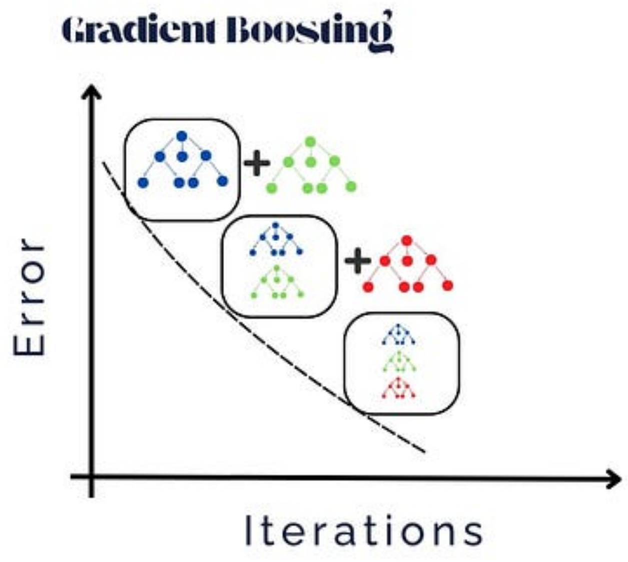


Image created by Author

Gradient Boosting is an ensemble learning technique that combines several weak models, usually decision trees, into a strong predictive

In simple terms, it builds one model at a time, and each new model corrects the errors made by the previous ones.

Examples:

- Ranking search engine results
- Credit scoring
- Fraud detection

Most common Gradient boosting algorithms are:

- XGBoost (Extreme Gradient Boosting)
- LightGBM (Light Gradient Boosting Machine)
- GradientBoostingClassifier / Regressor (from *scikit-learn*)
- CatBoost

Python code example: In this example we will use the **Digits dataset.**

The **Digits dataset** is a classic dataset for practicing **image classification** using machine learning. It contains images of **handwritten digits** (0 through 9), and the task is to classify which digit is shown in each image.

Copy

```
from sklearn.datasets import load_digits
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report

# Load the digits dataset
digits = load_digits()
X = digits.data       # Features: 64 pixel values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_siz

# Create and train the Gradient Boosting model
model = GradientBoostingClassifier(n_estimators=100, learning_rate
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Show the first 5 predictions vs. actual values
for i in range(5):
    print(f"Predicted: {y_pred[i]} — Actual: {y_test[i]}")

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred).round(2))
Outcome:

Predicted: 6 — Actual: 6
Predicted: 9 — Actual: 9
Predicted: 3 — Actual: 3
Predicted: 7 — Actual: 7
Predicted: 2 — Actual: 2
Accuracy: 0.97
```

5. Dimensionality Reduction Algorithms: Reduces the number of features

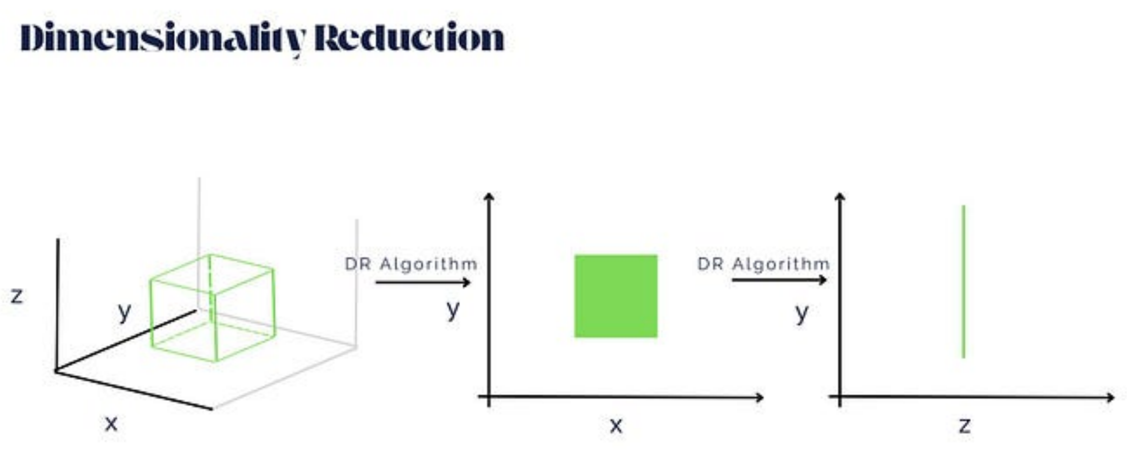


Image created by Author

Dimensionality reduction algorithms aim to reduce the number of features (dimensions) in a dataset while preserving its essential characteristics.

When we use them?

These algorithms are not used for making classification or regression predictions directly.

I included them in this article because they help reduce high-dimensional datasets to lower-dimensional spaces.

So, what is used for?

It is used **before** prediction models to:

- Remove noise or irrelevant features.
- Reduce computation time (fewer features).
- Help avoid overfitting.

By reducing high-dimensional data to 2D or 3D, those algorithms lets you **visualize class separation** or clusters.

Examples:

- Visualizing high-dimensional data
- Speeding up training time
- Noise reduction in data

- PCA: A linear technique that transforms data into a new coordinate system, where the principal components (directions of maximum variance) are ordered by the amount of variance they explain.
- t-SNE: A non-linear technique particularly useful for visualizing high-dimensional data in 2D or 3D. t-SNE focuses on preserving local neighborhood relationships.
- UMAP: A non-linear dimensionality reduction technique that aims to preserve both local and global structure of the data.
- LDA: A supervised technique that seeks to find the best linear combination of features to separate different classes.

Python code example: We use again the breast cancer dataset.

What we are doing with this code:

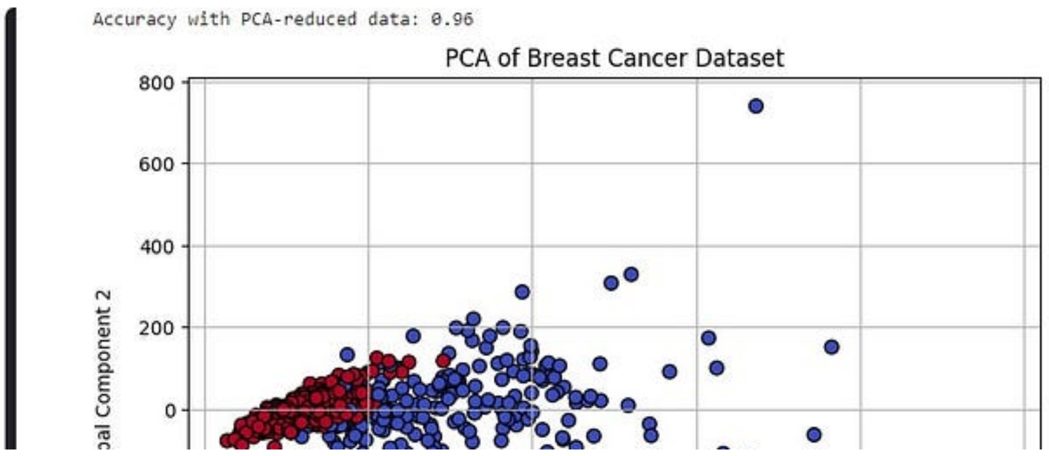
- **PCA** compresses the original 30 features into just **2 dimensions**, preserving most of the variance.
- We use **Logistic Regression** on the reduced data to make prediction.
- We create a **plot** helps visualize how well PCA separates the two classes.
- Despite reducing the number of features, models often still achieve high accuracy.

Copy

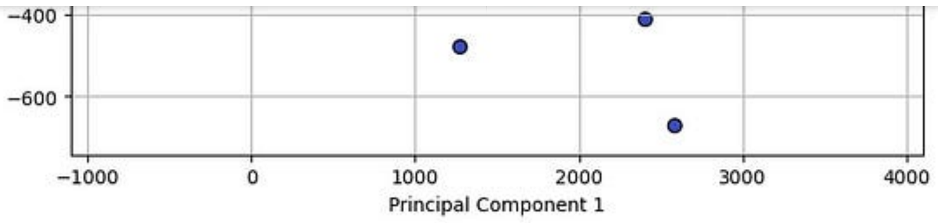
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_breast_cancer
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

Freedium

```
# Load dataset
data = load_breast_cancer()
X = data.data          # 30 features
y = data.target        # 0 = malignant, 1 = benign

# Apply PCA to reduce to 2 components
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test

# Train Logistic Regression
model = LogisticRegression()
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy with PCA-reduced data: {accuracy:.2f}")

# Plot the 2D PCA result with labels
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, cmap='coolwarm', edgeco
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.title("PCA of Breast Cancer Dataset")
plt.grid(True)
plt.show()
```



```
Accuracy with PCA-reduced data: 0.96
```



How PCA really works?

When you apply PCA, it transforms the original features (in this case, 30 of them) into a new set of **principal components**, which are **linear combinations** of the original features. Each component is made by combining the original features with different weights (called **loadings**).

To have a better understanding of what those 2 components are consist of you can use the code below that shows you which original features contribute most to the two principal components.

Copy

```
import pandas as pd

# Identify top contributing features to each component
loadings = pd.DataFrame(
    pca.components_.T,  # Transpose to get features as rows
    columns=['PC1', 'PC2'],
    index=data.feature_names
)

# Display top 5 contributing features to PC1
print("\nTop 5 contributing features to PC1:")
print(loadings['PC1'].abs().sort_values(ascending=False).head())

# Display top 5 contributing features to PC2
print("\nTop 5 contributing features to PC2:")
print(loadings['PC2'].abs().sort_values(ascending=False).head())
Outcome:
```

```
area error          0.055727
worst perimeter     0.049458
mean perimeter      0.035076
Name: PC1, dtype: float64

Top 5 contributing features to PC2:
mean area          0.851824
worst area         0.519742
mean perimeter     0.062748
worst texture      0.013215
mean radius        0.009287
Name: PC2, dtype: float64
```

Understanding and mastering these common ML algorithms is essential for any aspiring data scientist. Each serves a unique purpose and shines under different data conditions. Practice, experiment, and explore further, it's the best way to keep growing and getting better!

I hope I helped you to clear your mind around Machine Learning. There are many more to explore and learn but is a great start!

> If you found this article helpful show some appreciation there and give it a clap!

Must read articles:

Into The World of Hypothesis Testing — Part 1: Introduction

Have you ever wondered how to determine whether something you believe is actually true?

medium.com

with Python Example

Machine learning (ML) is only as good as the data it learns from. Poor-quality data leads to inaccurate predictions...

medium.com

Train-Test Split in Python: A Step-by-Step Guide with Example for Accurate Model Evaluation

If you're new to machine learning, you've likely encountered the train-test split method. This fundamental technique...

medium.com

Random Forest Regression in Python — How to use it in a Predictive Analysis

What is Predictive Analysis?

medium.com

#machine-learning #python #data-science #python-ml