

Introduction to Machine Learning (ML)

Definition:

Machine learning (ML) uses algorithms trained on data to create models that can perform tasks typically requiring human intelligence. These tasks include recognizing patterns, making decisions, and predictions

How it Works:

ML algorithms use a dataset to train a model. The training involves an iterative process where the model makes predictions, compares them to the actual outcomes, and adjusts its parameters to improve accuracy.

Types of Machine Learning:

- **Supervised Learning:**

The algorithm learns from a labeled dataset, meaning the correct answer is included in the training data.

- **Unsupervised Learning:**

The algorithm must find patterns and relationships in datasets without any labels.

- **Reinforcement Learning:**

The algorithm learns by making a sequence of decisions and receiving feedback in the form of rewards or penalties.

Applications:

Machine learning powers many applications we use daily, such as recommendation systems, search engines, spam filters, and speech recognition systems. Machine learning is a rapidly evolving field that continues to push the boundaries of what's possible with technology, making our interactions with digital systems more intuitive and efficient.

✓ Linear Regression

Definition:

Linear Regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The goal is to find the best-fit line or the linear equation that can be used to predict outcomes.

Equation

The linear equation for a simple linear regression, which involves a single independent variable, is:

where:

- y is the dependent variable
- x is the independent variable
- β_0 is the y-intercept
- β_1 is the slope of the line
- ϵ is the error term.

Purpose:

The primary use of Linear Regression is to predict values within a continuous range, rather than trying to classify them into categories. For example, predicting prices of houses, sales of a product, or temperature forecasts.

Assumptions:

Linear Regression assumes that there is a linear relationship between the independent and dependent variables. Other assumptions include homoscedasticity, independence of errors, and normal distribution of errors

Applications:

It's widely used in economics, business, engineering, and the natural and social sciences as it provides a clear understanding of the relationships between variables. Linear Regression is valued for its ease of interpretation and understanding, making it a staple for many predictive modeling tasks

Example:

✓ Import

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_regression
```

✓ Generate a random regression problem

```
lrX, lry = make_regression(n_samples=100, n_features=1, noise=10)
```

✓ Split

```
lrX_train, lrX_test, lry_train, lry_test = train_test_split(lrX, lry, test_size=0.2)
```

✓ Create linear regression object

```
lr = LinearRegression()
```

✓ Train the model using the training sets

```
lr.fit(lrX_train, lry_train)
```

▼ LinearRegression ⓘ ?

LinearRegression()

✓ Make predictions using the testing set

```
lry_pred = lr.predict(lrX_test)
```

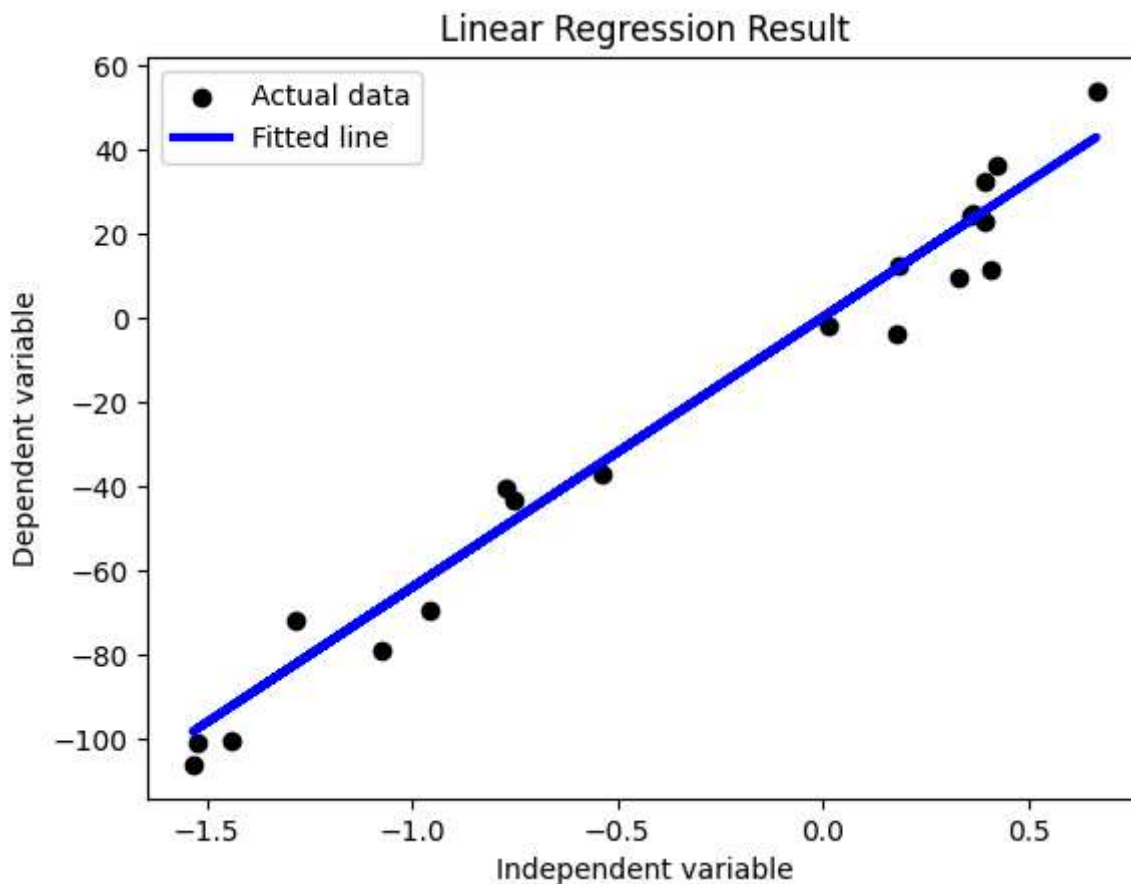
✓ Plot outputs

```
plt.scatter(lrX_test, lry_test, color='black', label='Actual data')
plt.plot(lrX_test, lry_pred, color='blue', linewidth=3, label='Fitted line')

plt.title('Linear Regression Result')
plt.xlabel('Independent variable')
plt.ylabel('Dependent variable')

plt.legend()

plt.show()
```



✓ Validation

To validate your Linear Regression model, you can use several statistical metrics to assess its performance. Here are some common methods:

- R-squared (Coefficient of Determination):

This metric indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, Generally, a higher R-squared value means that the model fits the data better.

```
from sklearn import metrics
```

```
lrr_squared = metrics.r2_score(lry_test, lry_pred)
print('R-squared: ', lrr_squared)
```

```
R-squared:  0.9721095523164569
```

- **✓ Mean Absolute Error (MAE):**

This is the average of the absolute differences between the predicted values and the actual values. It gives an idea of how big the errors are on average. The Mean Absolute Error (MAE) is considered better when it is closer to 0.

```
lrmae = metrics.mean_absolute_error(lry_test, lry_pred)
print('Mean Absolute Error: ', lrmae)
```

```
Mean Absolute Error:  7.148104288854194
```

- **✓ Mean Squared Error (MSE):**

This is the average of the squares of the errors. It penalizes larger errors more than MAE does. The Mean Squared Error (MSE) is considered better when it is closer to 0.

```
lrmse = metrics.mean_squared_error(lry_test, lry_pred)
print('Mean Squared Error: ', lrmse)
```

```
Mean Squared Error:  71.12663664494545
```

- **✓ Root Mean Squared Error (RMSE):**

This is the square root of MSE. It's in the same units as the dependent variable and is often used to compare different regression models. The Root Mean Squared Error (RMSE) is better when it is as close to 0 as possible.

```
lrrmse = np.sqrt(lrmse)
print('Root Mean Squared Error: ', lrrmse)
```

```
Root Mean Squared Error:  8.433660927790816
```

- **✓ At the End make it better**

```
import pandas as pd
```

```
lr_results = pd.DataFrame(['Simple Linear Regression', lrrmse, lrmse, lrmae, lrr_squared]).t
lr_results.columns = ['Method', 'RMSE', 'MSE', 'MAE', 'R2']
lr_results
```

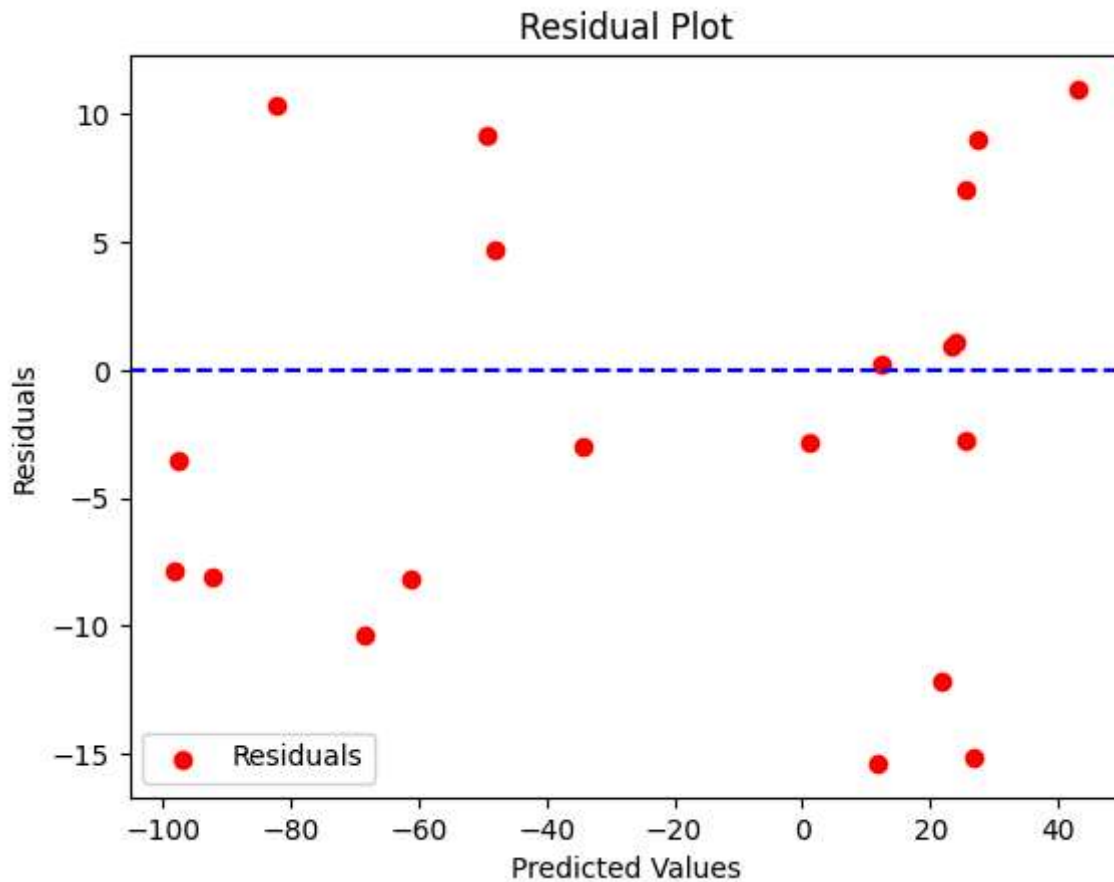
	Method	RMSE	MSE	MAE	R2
0	Simple Linear Regression	8.433661	71.126637	7.148104	0.97211

✓ Residuals plots

Additionally, you can use residual plots to visually check for the validity of your model. Residuals are the differences between the actual values and the predicted values. A well-fitted model will have residuals that are randomly scattered around zero.

```
residuals = lry_test - lry_pred

plt.scatter(lry_pred, residuals, color='red', label='Residuals')
plt.axhline(y=0, color='blue', linestyle='--')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.title('Residual Plot')
plt.legend()
plt.show()
```



✓ Support Vector Machine (SVM)

Support Vector Machine (SVM) is a powerful and versatile supervised machine learning algorithm used for both classification and regression tasks, though it is primarily known for classification. Here's an overview of SVM: Objective: The main goal of SVM is to find the optimal hyperplane that separates different classes in the feature space with the maximum margin possible¹.

Core Idea:

SVM classifies data by finding the optimal hyperplane which maximizes the margin between different classes in an N-dimensional space.

Hyperplane:

This is the decision boundary that separates different classes. The dimension of the hyperplane depends on the number of input features.

Support Vectors:

These are the data points nearest to the hyperplane, which influence its position. SVM aims to maximize the distance (margin) from these points to the hyperplane

Margin:

It's the distance between the hyperplane and the nearest data point from either class. A larger margin is associated with a lower generalization error of the classifier

Kernels:

SVM can handle linear and non-linear data. For non-linear data, it uses kernel functions to transform the input space into a higher-dimensional space where a linear separator can be used.

Robustness to Outliers:

SVM is known for its robustness to outliers. It focuses on the support vectors and the margin, which helps in ignoring the outliers while finding the hyperplane¹.

Applications:

SVM is used in various domains like text classification, image recognition, spam detection, and more due to its effectiveness in high-dimensional spaces and its ability to handle nonlinear relationships.

SVM's ability to find the maximum separating hyperplane by considering the closest points of different classes makes it a strong model, especially when dealing with complex datasets.

Example:

✓ Import

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

✓ Load The Iris Dataset from sklearn

This data sets consists of 3 different types of irises' (Setosa, Versicolour, and Virginica) petal and sepal length, stored in a 150x4 numpy.ndarray

The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

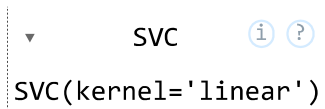
```
iris = datasets.load_iris()
svmX = iris.data[:, :2]
svmy = iris.target
```

✓ Split

```
svmX_train, svmX_test, svmy_train, svmy_test = train_test_split(svmX, svmy, test_size=0.2, r
```

✓ Create an SVM classifier with a linear kernel

```
clf = SVC(kernel='linear')
clf.fit(svmX_train, svmy_train)
```



```
SVC
SVC(kernel='linear')
```

✓ Predict using the test set

```
svmy_pred = clf.predict(svmX_test)
```

✓ Create a mesh to plot the decision boundaries

```
h = .02 # step size in the mesh
svmx_min, svmx_max = svmX[:, 0].min() - 1, svmX[:, 0].max() + 1
svmy_min, svmy_max = svmX[:, 1].min() - 1, svmX[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(svmx_min, svmx_max, h), np.arange(svmy_min, svmy_max, h))
```

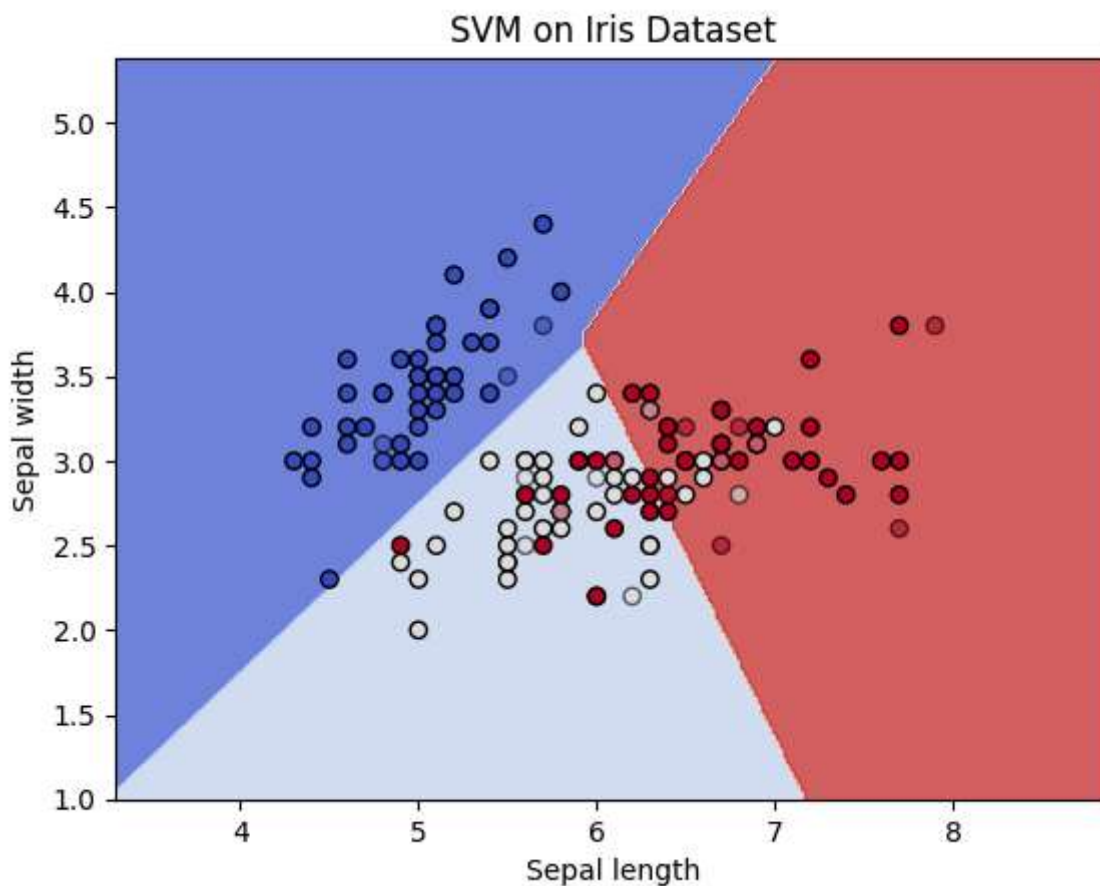
✓ Plot outputs

```
# Plot the decision boundary by assigning a color to each point in the mesh
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

# Plot also the training points
plt.scatter(svmX_train[:, 0], svmX_train[:, 1], c=svmy_train, cmap=plt.cm.coolwarm, edgecolor=
# and testing points
plt.scatter(svmX_test[:, 0], svmX_test[:, 1], c=svmy_test, cmap=plt.cm.coolwarm, edgecolors=

plt.xlabel('Sepal length')
plt.ylabel('Sepal width')
plt.title('SVM on Iris Dataset')

# Show the plot
plt.show()
```



✓ Validation

Confusion Matrix

A Confusion Matrix is a table used in machine learning to evaluate the performance of a classification model on a set of test data. It's particularly useful for visualizing the accuracy of a

model and understanding its errors.

```
cm = confusion_matrix(svmy_test, svmy_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=clf.classes_)
disp.plot()

plt.show()
```

