**TF-IDF vector** is a numerical representation of a text document. It quantifies how important a word is in a document relative to a collection of documents.

**TF**: Term Frequency

**IDF**: Inverse Document Frequency

### **Term Frequency (TF)**

Term Frequency (TF) is a numerical statistic used in Natural Language Processing (NLP) and Information Retrieval (IR) to measure how frequently a term (word) appears in a document. It is a component of the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm, which is widely used in search engines and text mining.

The frequency of a term in a document

Formula:

TF(t,d) = ftd/nd

Where ftd = number of times term t appears in document d.

Where nd​ = total number of terms in document d

**TF-IDF vector sample code:**

from sklearn.feature\_extraction.text import TfidfVectorizer

import pandas as pd

# Sample corpus (documents)

documents = [

"the cat sat on the mat",

"the dog sat on the log",

"the dog chased the cat"

]

# Initialize the vectorizer

vectorizer = TfidfVectorizer()

# Fit and transform the documents

tfidf\_matrix = vectorizer.fit\_transform(documents)

# Convert TF-IDF matrix to dense form

tfidf\_dense = tfidf\_matrix.toarray()

# Get feature names (terms)

terms = vectorizer.get\_feature\_names\_out()

# Create a DataFrame for readability

df = pd.DataFrame(tfidf\_dense, columns=terms)

# Display the TF-IDF vectors

print(df)

**Inverse Document Frequency (IDF)**

Inverse Document Frequency (IDF) is a key concept in text mining and Natural Language Processing (NLP). It measures how important or rare a word is across a collection of documents (called a corpus). IDF and Term Frequency (TF) are used to form the widely used TF-IDF metric.

Measures how rare a term is across all documents

Formula:

IDF(t)=log(n/1+dft​)

N = total number of documents

dft​ = number of documents containing term t

IDF Sample Code:

import math

# Sample corpus (3 documents)

documents = [

"the cat sat on the mat",

"the dog sat on the log",

"the dog chased the cat"

]

# Step 1: Tokenize and lowercase

tokenized\_docs = [doc.lower().split() for doc in documents]

# Step 2: Count document frequencies (df)

N = len(tokenized\_docs) # Total number of documents

df = {}

for doc in tokenized\_docs:

unique\_terms = set(doc) # Avoid counting same term twice in a document

for term in unique\_terms:

df[term] = df.get(term, 0) + 1

# Step 3: Compute IDF for each term

idf = {}

for term, doc\_count in df.items():

idf[term] = math.log(N / (1 + doc\_count)) # Smoothing with +1

# Step 4: Display results

print("IDF Scores:")

for term, score in idf.items():

print(f"{term}: {score:.4f}")

TF-IDF Score

TF-IDF stands for Term Frequency–Inverse Document Frequency. It is a numerical statistic that reflects the importance of a word in a document relative to a collection of documents (called a corpus). It is widely used in search engines, text mining, and machine learning for feature extraction from text.

Formula:

TFIDF(t,d) = TF(t,d) x IDF(t)

Sample code:

import math

from collections import Counter

# Sample corpus (documents)

documents = [

"the cat sat on the mat",

"the dog sat on the log",

"the dog chased the cat"

]

# Step 1: Tokenize documents

tokenized\_docs = [doc.lower().split() for doc in documents]

N = len(tokenized\_docs)

# Step 2: Compute Term Frequencies (TF)

tf\_list = []

for doc in tokenized\_docs:

term\_counts = Counter(doc)

total\_terms = len(doc)

tf = {term: count / total\_terms for term, count in term\_counts.items()}

tf\_list.append(tf)

# Step 3: Compute Document Frequencies (df)

df = {}

for doc in tokenized\_docs:

for term in set(doc):

df[term] = df.get(term, 0) + 1

# Step 4: Compute Inverse Document Frequency (IDF)

idf = {term: math.log(N / (1 + df[term])) for term in df} # +1 for smoothing

# Step 5: Compute TF-IDF

tfidf\_list = []

for tf in tf\_list:

tfidf = {term: tf[term] \* idf[term] for term in tf}

tfidf\_list.append(tfidf)

# Step 6: Display TF-IDF scores

for i, tfidf in enumerate(tfidf\_list):

print(f"\nTF-IDF for Document {i+1}:")

for term, score in sorted(tfidf.items(), key=lambda x: -x[1]):

print(f" {term}: {score:.4f}")

**Linear Regression** is a supervised machine learning algorithm that predicts a continuous output based on one or more input features. It assumes a linear relationship between the independent variable(s) and the dependent variable.

Types of Linear Regression:

Simple Linear Regression is a statistical method to model the relationship between two variables:

Independent variable (X) – the input feature

Dependent variable (Y) – the output or prediction

Formula:

y = mx + b

y: predicted value

x: input value

m: slope (how much y changes per unit of x)

b: intercept (value of y when x = 0)

The goal is to find the best values for m and b that minimize the error (difference between predicted and actual y).

Sample code:

# Sample data

X = [1, 2, 3, 4, 5]

Y = [2, 4, 5, 4, 5]

# Step 1: Calculate means

mean\_x = sum(X) / len(X)

mean\_y = sum(Y) / len(Y)

# Step 2: Calculate slope (m) and intercept (b)

numerator = sum((X[i] - mean\_x) \* (Y[i] - mean\_y) for i in range(len(X)))

denominator = sum((X[i] - mean\_x) \*\* 2 for i in range(len(X)))

m = numerator / denominator

b = mean\_y - m \* mean\_x

# Step 3: Predict values

def predict(x):

return m \* x + b

# Step 4: Print results

print(f"Slope (m): {m}")

print(f"Intercept (b): {b}")

print("Predicted Y values:")

for x in X:

print(f"x = {x} -> y = {predict(x):.2f}")

Sample code with scikit-learn:

from sklearn.linear\_model import LinearRegression

import numpy as np

# Prepare data

X = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Feature must be 2D

Y = np.array([2, 4, 5, 4, 5]) # Target variable

# Create model and fit it

model = LinearRegression()

model.fit(X, Y)

# Get parameters

print(f"Slope (m): {model.coef\_[0]}")

print(f"Intercept (b): {model.intercept\_}")

# Predict

predictions = model.predict(X)

print("Predictions:", predictions)

**Scikit-learn** (also known as sklearn) is one of the most popular and powerful open-source machine learning libraries in Python. It provides simple and efficient tools for data mining, data analysis, and machine learning, built on top of NumPy, SciPy, and matplotlib.

Sample code for Visualising the Regression Line

import matplotlib.pyplot as plt

plt.scatter(X, Y, color='blue', label='Actual data')

plt.plot(X, predictions, color='red', label='Regression line')

plt.xlabel('X')

plt.ylabel('Y')

plt.title('Simple Linear Regression')

plt.legend()

plt.show()

**Multiple Linear Regression** is an extension of Simple Linear Regression used when you want to model the relationship between two or more independent variables (features) and one dependent variable (target).

Formula:

y = b0 + b1 x1 + b2 x2 + ⋯ + bn xn + ε

Where

y: predicted value (dependent variable)

x1,x2,…,xn​: independent variables (features)

b0​: intercept (bias)

b1,b2,…,bn​: coefficients (slopes for each feature)

ε: error term

Sample code using sci-kit learn:

import numpy as np

from sklearn.linear\_model import LinearRegression

# Example data: Predict Y based on X1 and X2

# Features: [x1, x2]

X = np.array([

[1, 2],

[2, 1],

[3, 4],

[4, 3],

[5, 5]

])

# Target variable

y = np.array([2, 3, 6, 7, 10])

# Create and train model

model = LinearRegression()

model.fit(X, y)

# Coefficients and intercept

print(f"Intercept (b0): {model.intercept\_}")

print(f"Coefficients (b1, b2): {model.coef\_}")

# Predict new values

X\_new = np.array([[6, 6]])

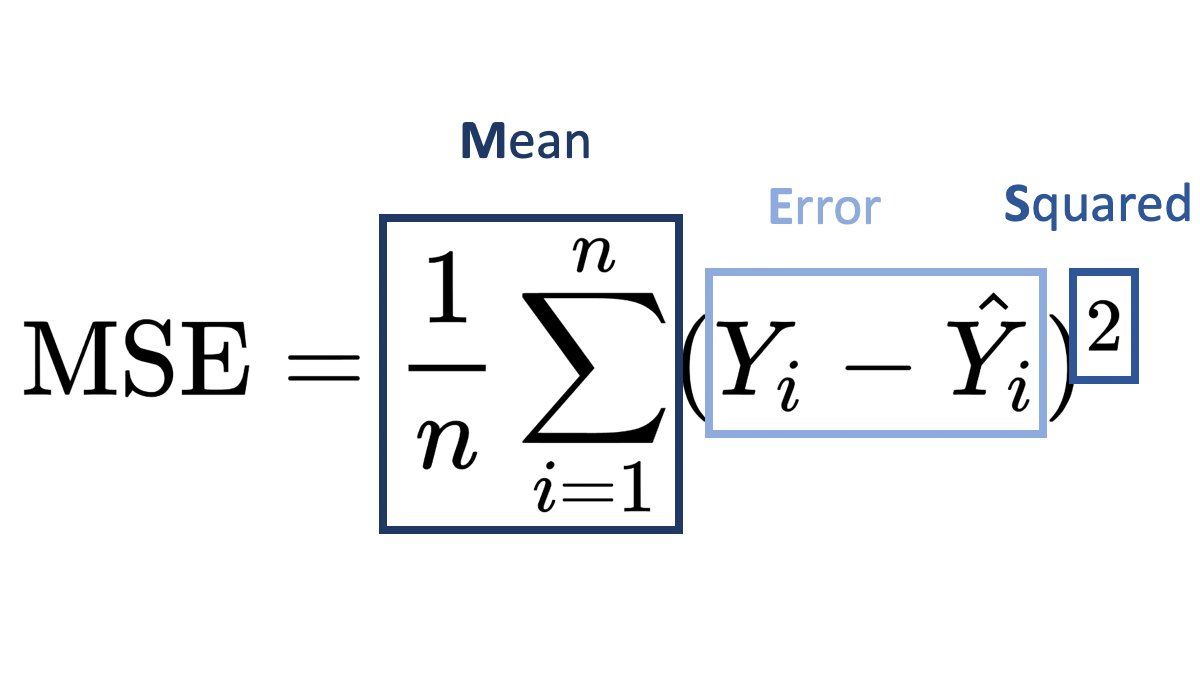
prediction = model.predict(X\_new)

print(f"Prediction for [6,6]: {prediction[0]}")

**Mean Squared Error (MSE**) is a common regression metric used to measure how well a model's predictions match the actual values.

It calculates the average of the squared differences between predicted and actual values.

Formula:



Where

yi​: actual (true) value

y^​i​: predicted value

n: total number of data points

Sample Code:

# True values and predicted values

y\_true = [3, -0.5, 2, 7]

y\_pred = [2.5, 0.0, 2, 8]

# Manual MSE calculation

n = len(y\_true)

mse = sum((y\_true[i] - y\_pred[i]) \*\* 2 for i in range(n)) / n

print(f"Mean Squared Error (Manual): {mse}")

Sample code with sci-kit learn:

from sklearn.metrics import mean\_squared\_error

y\_true = [3, -0.5, 2, 7]

y\_pred = [2.5, 0.0, 2, 8]

# scikit-learn MSE

mse = mean\_squared\_error(y\_true, y\_pred)

print(f"Mean Squared Error (sklearn): {mse}")