**KNN Algorithm Explained Simply**

The **K-Nearest Neighbors (KNN)** algorithm is a type of **supervised learning** algorithm used for classification and regression. Here's a simple breakdown:

1. **Training Phase**: KNN doesn't need a traditional training phase like other algorithms. Instead, it stores the entire dataset. It "learns" from the data during the prediction phase.
2. **Prediction**:
   * Given a new data point, the algorithm looks at the **K** closest data points to it in the training dataset.
   * It calculates the **distance** between the new point and all the points in the dataset. The most common distance metric is **Euclidean distance**.
   * The algorithm then classifies the new point based on the majority label (in classification) or averages the values (in regression) of the K nearest neighbors.
3. **Choosing K**:
   * K is a number that determines how many nearest neighbors should be considered. A small value (like 1) makes the model sensitive to noise, while a large value might over-smooth the prediction.

**Steps in KNN**

1. Choose the number of neighbors (K).
2. Calculate the distance between the new point and all the points in the training set.
3. Sort the distances and identify the K nearest neighbors.
4. For **classification**, assign the most frequent class label among the K neighbors. For **regression**, take the average of the values from the K nearest neighbors.

**KNN Algorithm Code (Python Example)**

Here's a simple Python code using the KNN algorithm for classification:

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Step 1: Load dataset (Iris dataset for this example)

iris = load\_iris()

X = iris.data # Features

y = iris.target # Labels (target)

# Step 2: Split dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Step 3: Initialize KNN classifier with K=3 (you can change K as needed)

knn = KNeighborsClassifier(n\_neighbors=3)

# Step 4: Fit the model on training data

knn.fit(X\_train, y\_train)

# Step 5: Make predictions on the test data

y\_pred = knn.predict(X\_test)

# Step 6: Evaluate the model's accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy of KNN classifier: {accuracy \* 100:.2f}%")

.

**KNN Algorithm Case Study: Predicting Whether a Customer Will Buy a Product**

**Problem Statement:**

A retail company wants to predict whether a customer will buy a product based on some features such as age, income, and past shopping behavior. The goal is to build a predictive model using the K-Nearest Neighbors (KNN) algorithm to make these predictions. The dataset contains historical data of customers, including their age, income, and whether they made a purchase.

**Dataset:**

The dataset contains the following features:

1. **Age**: The age of the customer.
2. **Income**: The income level of the customer.
3. **Previous Purchases**: Whether the customer has made a purchase in the past (binary: 0 = No, 1 = Yes).
4. **Target**: Whether the customer will buy the product in the future (binary: 0 = No, 1 = Yes).

Here’s a small snippet of the dataset:

| **Age** | **Income** | **Previous Purchases** | **Will Buy (Target)** |
| --- | --- | --- | --- |
| 25 | 45000 | 1 | 1 |
| 30 | 60000 | 0 | 0 |
| 35 | 80000 | 1 | 1 |
| 40 | 90000 | 1 | 1 |
| 22 | 35000 | 0 | 0 |
| 28 | 70000 | 0 | 1 |
| 45 | 95000 | 1 | 1 |
| 33 | 55000 | 0 | 0 |

**Solution Approach Using KNN:**

1. **Step 1: Load and Preprocess Data**
   * Load the dataset into a Pandas DataFrame.
   * Split the data into features (X) and the target variable (y).
   * Preprocess the data (e.g., scale the features for better performance).
2. **Step 2: Train-Test Split**
   * Split the dataset into a training set (80%) and a testing set (20%) using train\_test\_split from scikit-learn.
3. **Step 3: Apply KNN Algorithm**
   * Initialize a KNN classifier with a suitable value for k (number of nearest neighbors). For this case, we will choose k=3.
   * Train the model on the training data.
4. **Step 4: Evaluate the Model**
   * Test the model on the test set and evaluate the predictions.
   * Use metrics like accuracy to measure the performance of the model.
5. **Step 5: Predictions**
   * Based on the trained model, predict whether new customers will buy the product.

**Code Implementation for KNN:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import StandardScaler

# Step 1: Load the dataset

data = {

'Age': [25, 30, 35, 40, 22, 28, 45, 33],

'Income': [45000, 60000, 80000, 90000, 35000, 70000, 95000, 55000],

'Previous Purchases': [1, 0, 1, 1, 0, 0, 1, 0],

'Will Buy': [1, 0, 1, 1, 0, 1, 1, 0]

}

df = pd.DataFrame(data)

# Step 2: Split data into features and target variable

X = df[['Age', 'Income', 'Previous Purchases']] # Features

y = df['Will Buy'] # Target variable

# Step 3: Split 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)

# Step 4: Feature Scaling (optional, but recommended for KNN)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Step 5: Initialize KNN classifier with k=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Step 6: Train the model

knn.fit(X\_train\_scaled, y\_train)

# Step 7: Make predictions on the test set

y\_pred = knn.predict(X\_test\_scaled)

# Step 8: Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy of KNN classifier: {accuracy \* 100:.2f}%")

**Naive Bayes in Layman Terms**

Imagine you're trying to predict whether someone will buy a product based on a few features, like their age, income, and location. In this case, you're asking a simple question: "Given these features, how likely is it that this person will buy the product?"

**Naive Bayes** is a method that helps answer this question using **probabilities**. It’s like a detective who solves mysteries by looking at the probabilities of different events happening, based on past data. It works by calculating the probability of each outcome (e.g., buying or not buying) for different feature combinations (e.g., age, income, etc.), and then picking the outcome with the highest probability.

The "naive" part of Naive Bayes comes from the assumption that all features are independent of each other — like saying that knowing someone's income doesn’t affect knowing their age or location. This assumption is often not strictly true, but it simplifies the calculations and works surprisingly well in many cases.

**Steps of Naive Bayes**

1. **Calculate Probabilities**: Based on historical data, you calculate how likely each feature is to occur for each possible outcome. For example, if most people who bought the product were in the 30-40 age range, that’s a strong signal that age is related to the decision.
2. **Combine Probabilities**: For a new person (whose data you’re trying to predict), you combine the probabilities for each feature to get the overall likelihood of each outcome.
3. **Pick the Best Outcome**: Finally, you choose the outcome (e.g., "will buy" or "won’t buy") with the highest probability.

**Example of Naive Bayes in Action**

Let’s say we have a simple dataset of people with their **age**, **income**, and whether they **bought the product**. We want to predict if a new person will buy the product based on their age and income.

Here’s how Naive Bayes would work:

1. Calculate the probability of each feature (age, income) given the outcome ("will buy" and "won’t buy").
2. Multiply those probabilities to get a combined probability for each outcome.
3. Choose the outcome with the highest probability.

# Importing necessary libraries

from sklearn.naive\_bayes import GaussianNB

import numpy as np

import pandas as pd

# Sample data: Age, Income, and whether they bought the product (1 = Yes, 0 = No)

data = {

'Age': [25, 30, 35, 40, 22, 30, 50, 60],

'Income': [45000, 60000, 80000, 90000, 35000, 50000, 120000, 110000],

'Bought': [0, 0, 1, 1, 0, 0, 1, 1] # 1 = Bought, 0 = Didn't Buy

}

# Convert the data into a DataFrame

df = pd.DataFrame(data)

# Defining features (X) and target variable (y)

X = df[['Age', 'Income']] # Features: Age and Income

y = df['Bought'] # Target: Whether they bought the product

# Initialize the Naive Bayes model

model = GaussianNB()

# Train the model using the data

model.fit(X, y)

# Now, let's make a prediction for a new person

# New person's age = 32, income = 65000

new\_person = np.array([[32, 65000]])

# Predicting whether they will buy the product

prediction = model.predict(new\_person)

if prediction == 1:

print("This person is likely to buy the product.")

else:

print("This person is unlikely to buy the product.")

**Naive Bayes Case Study: Email Spam Classification**

**Scenario:** Let’s imagine you're working at an email service provider, and your job is to create a system that can automatically classify emails as "spam" or "not spam." The goal is to build a model that can look at various characteristics of an email — such as the frequency of certain words — and predict whether it is spam or not.

We’ll use **Naive Bayes** to solve this problem because it works well for text classification tasks, where we treat words as features that may be conditionally independent of each other. This means, for example, the presence of the word “free” in an email doesn't depend on the presence of the word “offer,” though both may indicate spam.

**Dataset:**

The dataset contains emails, where each email is labeled as either "spam" (1) or "not spam" (0). Each email also contains a set of features that are simply the frequencies of certain words, such as "free," "offer," "buy," etc.

**Data:**

Let’s simulate a small dataset:

| **Email** | **Free** | **Offer** | **Buy** | **Money** | **Spam** |
| --- | --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 0 |
| 5 | 1 | 1 | 0 | 1 | 1 |

In the dataset:

* "Free," "Offer," "Buy," and "Money" are features.
* "Spam" is the target variable (1 = spam, 0 = not spam).

# Importing necessary libraries

from sklearn.naive\_bayes import GaussianNB

import numpy as np

import pandas as pd

# Creating the dataset

data = {

'Free': [1, 1, 0, 0, 1],

'Offer': [1, 0, 1, 1, 1],

'Buy': [0, 1, 0, 1, 0],

'Money': [0, 1, 0, 1, 1],

'Spam': [1, 1, 0, 0, 1] # Target variable

}

# Convert the data into a DataFrame

df = pd.DataFrame(data)

# Defining features (X) and target variable (y)

X = df[['Free', 'Offer', 'Buy', 'Money']] # Features: Free, Offer, Buy, Money

y = df['Spam'] # Target: Whether the email is Spam (1) or Not Spam (0)

# Initialize the Naive Bayes model

model = GaussianNB()

# Train the model using the data

model.fit(X, y)

# Now, let's make predictions for new emails

# New email 1: Free=1, Offer=0, Buy=1, Money=0

new\_email\_1 = np.array([[1, 0, 1, 0]])

# New email 2: Free=0, Offer=1, Buy=0, Money=0

new\_email\_2 = np.array([[0, 1, 0, 0]])

# Predicting whether these emails are spam or not

prediction\_1 = model.predict(new\_email\_1)

prediction\_2 = model.predict(new\_email\_2)

print("Email 1 Prediction: Spam" if prediction\_1 == 1 else "Email 1 Prediction: Not Spam")

print("Email 2 Prediction: Spam" if prediction\_2 == 1 else "Email 2 Prediction: Not Spam")

**Explanation of the Code:**

1. **Data Preparation**:
   * We have a small dataset with four features: "Free," "Offer," "Buy," and "Money."
   * The target variable "Spam" indicates whether the email is spam (1) or not spam (0).
2. **Model Training**:
   * We use the GaussianNB() model from sklearn.naive\_bayes, which is suitable for classification tasks. It calculates probabilities based on the given features and the target variable.
3. **Making Predictions**:
   * We predict the labels ("Spam" or "Not Spam") for two new emails by passing their features to the model. Each new email is represented by an array of features like [Free, Offer, Buy, Money].

**Output:**

Email 1 Prediction: Spam

Email 2 Prediction: Not Spam

**Interpretation of the Output:**

* **Email 1**: Based on the words in the email, the model predicts that it is likely **spam** because it contains the words "buy" (which could be a signal of a promotional offer) and "free."
* **Email 2**: The model predicts **not spam** for this email because it contains fewer spam-related words (it has "offer," but the model learned that "offer" alone doesn't necessarily indicate spam without other signals like "free" or "buy").
* **===========**

KNN algorithm

====================

It doesn't require separate training phase

The training happens during prediction phase

That means it learns from the data during the prediction phase

----------------------

Given a new data point,the algorithm should chosose k closest data points

to it in the training set

It uses eucledean distance formula for calculation

calculates distance bw new point and all other points.

---

k-->how many nearest neighbours should be considered

For classification,assign the most frequent class label among

the k-neighbours

For regression take the avg of values from the k-nearest neighbours

----

d(p,q)=sqrt[(summation i=1 to n (pi-qi)^2]

p=p1,p2..pn ====>first data point

q=q1,q2...qn===>second data point

n-->no of dimensions for features

p=x1,y1 q=x2,y2

d(p,q)=sqrt[(x2-x1)^2+(y2-y1)^2]

=====================================

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

iris=load\_iris()

X=iris.data

y=iris.target

#split data 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 knn classifier for k=3

knn=KNeighborsClassifier(n\_neighbors=3)

#fit the model on training data

knn.fit(X\_train,y\_train)

#make predictions on test data

y\_pred=knn.predict(X\_test)

#calculate accuracy

accuracy=accuracy\_score(y\_test,y\_pred)

print("Accuracy: ",accuracy)

#adding few more details

print("Predicted classes: ",y\_pred)

print("Actual classes: ",y\_test)

print("Predicted classes: ",iris.target\_names[y\_pred])

print("Actual classes: ",iris.target\_names[y\_test])

#ploting the data

import matplotlib.pyplot as plt

plt.scatter(X[:,0],X[:,1],c=y)

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.title("Actual classes")

plt.show()

plt.scatter(X[:,0],X[:,1],c=knn.predict(X))

plt.xlabel("Sepal Length")

plt.ylabel("Sepal Width")

plt.title("Predicted classes")

plt.show()

==================================

A retail company wants to predict whether a customer

will buy a product based on some features like age,

income,past history

The goal is to build a predictive model using KNN

to make the prediction

Age Income previous\_purchase Target(Will buy)

--------------------------------------------

25 45000 1 1

30 60000 0 0

35 80000 1 1

40 90000 1 1

45 35000 0 0

33 70000 0 1

22 55000 0 0

============================================

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

data={

'Age':[25,30,35,40,45,33,22],

'Income':[45000,54000,55000,60000,75000,64000,52000],

'Previous\_purchase':[0,1,1,1,1,0,1],

'Will buy':[0,1,0,1,1,0,1] #TARGET

}

import pandas as pd

df=pd.DataFrame(data)

X=df[['Age','Income','Previous\_purchase']]

y=df['Will buy']

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2)

model=KNeighborsClassifier(n\_neighbors=3)

model.fit(X\_train,y\_train)

y\_pred=model.predict(X\_test)

print(accuracy\_score(y\_test,y\_pred))

#accuracy score 0.5 tells that the model is not good

#the model is good when the accuracy score is 1

#the model is bad when the accuracy score is 0

#plot the graph

import matplotlib.pyplot as plt

plt.scatter(df['Age'],df['Income'])

plt.xlabel('Age')

plt.ylabel('Income')

plt.show()

===================================

Naive bayes

===================================

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.naive\_bayes import MultinomialNB

from sklearn.metrics import accuracy\_score

from sklearn.naive\_bayes import GaussianNB

data={

'Age':[23,25,27,29,31,33,35,37,39,41],

'Income':[48000,55000,60000,75000,84000,90000,96000,102000,110000,118000],

'Bought':[0,0,0,0,0,1,1,1,1,1]

}

df=pd.DataFrame(data)

print(df)

X=df[['Age','Income']]

y=df['Bought']

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.2,

random\_state=0)

model=GaussianNB()

model.fit(X\_train,y\_train)

new\_data=np.array([[45,100000]])

prediction=model.predict(new\_data)

print(prediction)

if prediction==1:

print('He will buy')

else:

print('He will not buy')

==========================================================

spam email or not

=========================================================

import numpy as np

# Dataset

data = {

'Free': [1, 1, 0, 0, 0, 0],

'Offer': [0, 0, 1, 1, 1, 0],

'Buy': [0, 0, 0, 0, 0, 1],

'Money': [1, 0, 1, 0, 0, 0],

'Spam': [1, 1, 1, 0, 0, 0] # Target variable

}

# Separate features and target

features = ['Free', 'Offer', 'Buy', 'Money']

X = np.array([data[f] for f in features]).T

#X is the array of emails data[f] has the values of the feature f example: data['Free']=[1,1,0,0,0,0]

#.T is used to transpose the array

#transpose is used to interchange the rows and columns required for the array to be in the correct format

y = np.array(data['Spam'])

#y is the array of classes data['Spam']=[1,1,1,0,0,0]

def calculate\_probabilities(X, y):

classes = np.unique(y) #unique is used to get the unique values of the array

#classes will be the unique values of the array y where y is the array of classes

# Initialize dictionaries to store probabilities

feature\_prob = {cls: {} for cls in classes}

#empty {} specifies that the dictionary is empty

# Calculate prior probability

#where cls is the class to which the email belongs

#np.sum(y == cls) is the number of emails

# belonging to that class

class\_prob = {cls: np.sum(y == cls) / len(y) for cls in classes}

#in the above line,

# we are calculating the prior probability of each class

#example: if there are 3 classes and 100 emails

#that is cls.sum(y==cls) will be 30, 50, 20

#and len(y) is 100

#so the prior probability of class 1 will be 30/100

#class 2 will be 50/100 and class 3 will be 20/100

#probability of class 1, 2 and 3 will be 0.3, 0.5 and 0.2 respectively

for cls in classes:

X\_cls = X[y == cls]

# Filter data by class where X is the array of emails and y is the array of classes

for idx, feature in enumerate(features):

feature\_prob[cls][feature] = (np.sum(X\_cls[:, idx]) + 1) / (len(X\_cls) + 2)

#if cls is 1 and feauture=free 1 represents the email is spam and

# 0 represents the email is not spam

#X\_cls is the array of emails that belong to class 1

#X\_cls[:, idx] will be the array of values of the feature at index idx

#: is the row index and idx is the column index

#: is used to select all the rows

#np.sum(X\_cls[:, idx]) will be the number of emails that have the feature

#feature\_prob[cls][feature] will be the probability of the feature given the class

#in the above line, we are calculating the probability of the feature given the class

#example: if there are 3 classes and 100 emails

#that is cls.sum(y==cls) will be 30, 50, 20

#tracing with number of emails that have the feature

#if the number of emails that have the feature is 10

#then the probability of the feature given the class will be 10+1/30+2

#which is 11/32

#that is feature\_prob[1][free]=np.sum(X\_cls[:, 0]) + 1) / (len(X\_cls) + 2

#len(X\_cls) is the number of emails that belong to the class=30

#len(X\_cls)+2 is the number of emails that belong to the class+number of features

#which is 30+2=32

#therefore, feature\_prob[1][free]=11/32

#if the number of emails that have the feature is 20

#then the probability of the feature given the class will be 20+1/50+2

#which is 21/52

#if the number of emails that have the feature is 5

#then the probability of the feature given the class will be 5+1/20+2

#which is 6/22

#probability of the feature given the class will be 11/32, 21/52,

return class\_prob, feature\_prob

# Calculate probabilities

class\_prob, feature\_prob = calculate\_probabilities(X, y)

# Function to classify a new email

def classify(email):

scores = {}

for cls in class\_prob:

scores[cls] = np.log(class\_prob[cls]) # Start with prior

#the above line is used to calculate the prior probability of the class

#np.log is used to calculate the natural logarithm of the probability

#example, if the prior probability of class 1 is 0.3

#then the natural logarithm of the probability will be np.log(0.3)

#which is -1.2039728043259359

for idx, val in enumerate(email):

# Iterate over features

#idx is the index of the feature

#val is the value of the feature

#enumerate is used to iterate over the features

if val == 1:

scores[cls] += np.log(feature\_prob[cls][features[idx]])

#if the value of the feature is 1

#then the probability of the feature given the class will be added to the score

#example: if the value of the feature is 1 and the probability of the feature given the class is 11/32

#then the score will be 11/32

#if the value of the feature is 1 and the probability of the feature given the class is 21/52

#then the score will be 21/52

else:

scores[cls] += np.log(1 - feature\_prob[cls][features[idx]])

#if the value of the feature is 0

#then the probability of the feature given the class will be added to the score

#example: if the value of the feature is 0 and the probability of the feature given the class is 11/32

#then the score will be 11/32

#if the value of the feature is 0 and the probability of the feature given the class is 21/52

#then the score will be 21/52

return max(scores, key=scores.get) #returns the class with the maximum score

new\_email = [1, 1, 0, 0]

result = classify(new\_email)

print("The email is", "Spam" if result == 1 else "Not Spam")

new\_email2=[0,1,0,0]

result=classify(new\_email)

print("The email is", "Spam" if result == 1 else "Not Spam")

=======================