Naive-Bayes

January 14, 2025

[ ]:
```
'''
    Naive Bayes -->

    Naive Bayes is a family of simple yet effective probabilistic classifiers␣
↪based on applying Bayes'
    Theorem with a strong (and often unrealistic) assumption of feature␣
↪independence.
    It's widely used in text classification, spam detection, sentiment␣
↪analysis, and medical diagnosis
    due to its simplicity and efficiency.

    It uses Bayes' Theorem, a formula for calculating conditional␣
↪probabilities, and assumes that all
    features contribute independently to the probability of a class.

    Imagine you're a detective trying to figure out which suspect committed a␣
↪crime based on evidence
    (features). If suspect A has more evidence pointing to them than suspect B,␣
↪you'll probably accuse
    suspect A. Naive Bayes works similarly by weighing the evidence␣
↪(probabilities of features) for each
    class and picking the one with the highest likelihood.
'''
```

[ ]:
```
'''
    Types of Naive Bayes -->

    Gaussian Naive Bayes :

    Assumes that the data is normally distributed.
    Commonly used for continuous data.

    Multinomial Naive Bayes :

    Suitable for discrete data like word counts or frequencies.
    Popular in text classification tasks.

```

```
    Bernoulli Naive Bayes :

    Designed for binary data (e.g., presence/absence of a feature).
    Often used in binary text classification.
'''
```

[ ]:
```
'''
    Steps in Naive Bayes Classification -->

    Prepare the Dataset :

    Extract features and label them.
    Divide the dataset into training and testing sets.

    Calculate Probabilities :

    Compute the prior probabilities of each class.
    Compute the likelihood P(feature|class)) for each feature.

    Apply Bayes' Theorem :

    Combine the priors and likelihoods for prediction.

    Predict the Class :

    Select the class with the highest posterior probability.
'''
```

[ ]:
```
'''
    Advantages -->

    Fast and Efficient: Works well with large datasets.
    Simple : Easy to implement and interpret.
    Performs Well on Sparse Data: Especially effective in text data.

    Limitations -->

    Feature Independence Assumption: Rarely true in real-world data.
    Zero Frequency Problem: If a feature value doesn't exist in the training␣
 ↪dataset, it gets
    zero probability. (Solution: Laplace Smoothing).
    Sensitive to Irrelevant Features: Can be misled if irrelevant features␣
 ↪dominate the dataset
'''
```

[9]:
```
#   Importing Libraries -->
```

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix, mean_squared_error
```

[3]:
```python
#   Importing Dataset -->

data = pd.read_csv('Data/Social_Network_Ads.csv')
data.head(10)
```

[3]:

	Age	EstimatedSalary	Purchased
0	19	19000	0
1	35	20000	0
2	26	43000	0
3	27	57000	0
4	19	76000	0
5	27	58000	0
6	27	84000	0
7	32	150000	1
8	25	33000	0
9	35	65000	0

[4]:
```python
x_data = data.iloc[:, :-1].values
y_data = data.iloc[:, -1].values
```

[5]:
```python
#   Splitting The Dataset -->

x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size=0.
 ↪2, random_state=42)
```

[6]:
```python
x_train
```

[6]:
```
array([[    27,  57000],
       [    46,  28000],
       [    39, 134000],
       [    44,  39000],
       [    57,  26000],
       [    32, 120000],
       [    41,  52000],
       [    48,  74000],
       [    26,  86000],
       [    22,  81000],
       [    49,  86000],
```

```
[    36,  54000],
[    40,  59000],
[    41,  80000],
[    26,  16000],
[    39,  79000],
[    59, 130000],
[    42,  64000],
[    53, 143000],
[    34, 112000],
[    57, 122000],
[    39,  71000],
[    47,  25000],
[    24,  19000],
[    36,  50000],
[    32, 150000],
[    48,  29000],
[    30, 107000],
[    60,  34000],
[    38,  61000],
[    33,  31000],
[    39,  71000],
[    55,  39000],
[    49,  39000],
[    43, 112000],
[    27,  20000],
[    26,  17000],
[    37,  93000],
[    42,  54000],
[    35,  61000],
[    29,  75000],
[    38,  80000],
[    45,  26000],
[    54, 108000],
[    46,  23000],
[    23,  28000],
[    37,  75000],
[    42,  65000],
[    35,  71000],
[    51, 146000],
[    39,  96000],
[    24,  89000],
[    58,  95000],
[    25,  22000],
[    41,  59000],
[    28,  89000],
[    42,  80000],
[    42, 108000],
```

```
[    46,  96000],
[    47, 113000],
[    33,  28000],
[    19,  25000],
[    49,  89000],
[    31,  15000],
[    30,  79000],
[    48, 141000],
[    32, 117000],
[    37,  71000],
[    18,  86000],
[    42,  79000],
[    27,  84000],
[    40,  65000],
[    57,  74000],
[    26,  15000],
[    26,  80000],
[    29,  43000],
[    33, 149000],
[    39,  42000],
[    54, 104000],
[    36,  33000],
[    46,  32000],
[    40, 142000],
[    37,  62000],
[    29, 148000],
[    37,  57000],
[    35,  50000],
[    42,  53000],
[    35,  38000],
[    41,  30000],
[    40,  72000],
[    26,  15000],
[    31,  68000],
[    35,  53000],
[    35,  25000],
[    30,  89000],
[    41,  72000],
[    28, 123000],
[    46,  82000],
[    22,  63000],
[    45,  22000],
[    30,  49000],
[    34,  25000],
[    40,  75000],
[    32, 117000],
[    23,  82000],
```

```
[    26,  80000],
[    48, 131000],
[    59, 143000],
[    35,  55000],
[    34,  43000],
[    39,  61000],
[    27,  96000],
[    60,  83000],
[    24,  55000],
[    58, 144000],
[    53, 104000],
[    35,  79000],
[    36,  99000],
[    57,  60000],
[    37, 137000],
[    33,  43000],
[    41,  71000],
[    52,  21000],
[    52, 150000],
[    37,  70000],
[    26,  84000],
[    26,  72000],
[    26,  52000],
[    41,  60000],
[    31,  66000],
[    37, 144000],
[    38,  61000],
[    31,  34000],
[    42,  75000],
[    46, 117000],
[    36,  52000],
[    38,  71000],
[    49,  88000],
[    57,  33000],
[    48, 138000],
[    47,  50000],
[    33,  69000],
[    37, 146000],
[    20,  82000],
[    40,  47000],
[    35,  22000],
[    20,  36000],
[    45,  45000],
[    26,  43000],
[    58, 101000],
[    40,  57000],
[    38, 112000],
```

```
[    37,  80000],
[    49,  28000],
[    36,  75000],
[    41,  72000],
[    35,  60000],
[    43, 129000],
[    41,  87000],
[    38, 113000],
[    58,  23000],
[    26,  32000],
[    32,  18000],
[    41,  52000],
[    31,  18000],
[    35,  88000],
[    48,  35000],
[    27,  89000],
[    35,  97000],
[    42,  73000],
[    21,  68000],
[    41,  72000],
[    33,  60000],
[    39, 134000],
[    28,  84000],
[    46,  88000],
[    24,  58000],
[    31, 118000],
[    50,  88000],
[    20,  82000],
[    32, 135000],
[    20,  86000],
[    35,  27000],
[    29,  43000],
[    21,  88000],
[    35,  59000],
[    45,  32000],
[    60,  42000],
[    35,  91000],
[    35,  44000],
[    18,  44000],
[    42, 149000],
[    45,  79000],
[    40,  60000],
[    24,  23000],
[    33,  51000],
[    42,  70000],
[    55, 130000],
[    50,  44000],
```

```
[    48, 119000],
[    19,  76000],
[    41,  72000],
[    40,  71000],
[    27,  88000],
[    36, 126000],
[    35,  75000],
[    35,  58000],
[    34, 115000],
[    35,  73000],
[    60, 108000],
[    25,  87000],
[    27,  54000],
[    21,  16000],
[    37,  74000],
[    35,  39000],
[    54,  70000],
[    47,  30000],
[    38,  50000],
[    35, 147000],
[    35,  77000],
[    41,  79000],
[    37,  33000],
[    60,  46000],
[    28,  59000],
[    23,  66000],
[    23,  63000],
[    30,  17000],
[    25,  33000],
[    59,  83000],
[    58,  38000],
[    18,  82000],
[    46,  59000],
[    27,  17000],
[    58,  47000],
[    48,  30000],
[    49,  65000],
[    50,  36000],
[    53,  72000],
[    40,  57000],
[    52, 114000],
[    59,  42000],
[    36,  63000],
[    42, 104000],
[    37,  52000],
[    48,  33000],
[    59,  29000],
```

```
[    37,  79000],
[    40,  61000],
[    49,  74000],
[    25,  90000],
[    30,  15000],
[    40,  78000],
[    24,  84000],
[    38,  50000],
[    45, 131000],
[    21,  72000],
[    35,  23000],
[    35,  20000],
[    31,  89000],
[    30,  80000],
[    47,  47000],
[    27,  90000],
[    35,  72000],
[    30, 116000],
[    39, 122000],
[    29,  83000],
[    41,  63000],
[    48,  90000],
[    38,  59000],
[    32,  18000],
[    39,  75000],
[    26,  81000],
[    39, 106000],
[    22,  55000],
[    36, 118000],
[    60,  42000],
[    28,  55000],
[    51, 134000],
[    49,  28000],
[    36,  60000],
[    56, 104000],
[    27,  58000],
[    24,  32000],
[    34,  72000],
[    28,  32000],
[    50,  20000],
[    33,  41000],
[    29,  47000],
[    22,  18000],
[    30, 135000],
[    47, 105000],
[    46,  79000],
[    48, 134000],
```

```
       [    47,  49000],
       [    49, 141000],
       [    32, 100000],
       [    38,  71000],
       [    19,  26000],
       [    37,  77000],
       [    47,  51000],
       [    40,  57000],
       [    36, 125000],
       [    20,  74000],
       [    31,  58000],
       [    41,  45000],
       [    42,  54000],
       [    28,  37000],
       [    39,  73000],
       [    28,  85000],
       [    38,  51000],
       [    47,  43000],
       [    37,  72000],
       [    49,  36000],
       [    45,  22000],
       [    35,  72000],
       [    24,  27000],
       [    26,  35000],
       [    43, 133000],
       [    39,  77000],
       [    32,  86000]], dtype=int64)
```

[7]:
```
y_train
```

[7]:
```
array([0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1,
       1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0], dtype=int64)
```

[ ]:
```
#   Scaling The Values -->

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

[11]:
```
x_test
```

[11]:
```
array([[ 0.79753468, -1.40447546],
       [ 2.07309956,  0.51542886],
       [-0.96863208, -0.76450736],
       [ 0.99377543,  0.74814454],
       [-0.87051171, -1.22993871],
       [-0.77239133, -0.24089709],
       [ 0.89565505,  1.06812859],
       [-0.87051171,  0.36998156],
       [ 0.20881242,  0.13726589],
       [ 0.40505317, -0.15362871],
       [-0.28178945, -0.15362871],
       [ 1.4843773 , -1.05540195],
       [-1.45923396, -0.64814952],
       [-1.75359508, -1.37538601],
       [-0.77239133,  0.4863394 ],
       [-0.28178945,  1.09721805],
       [ 1.38625693, -0.93904411],
       [ 0.79753468,  0.10817643],
       [ 0.11069205, -0.82268628],
       [ 1.77873843, -0.29907601],
       [-1.55735433, -1.25902817],
       [-0.87051171,  0.28271318],
       [ 0.89565505, -1.37538601],
       [ 2.07309956,  0.16635535],
       [-1.85171546, -1.49174384],
       [ 1.28813655, -1.37538601],
       [ 0.40505317,  0.28271318],
       [-0.0855487 , -0.50270222],
       [ 1.68061805,  1.59173886],
       [-1.85171546, -1.43356492],
       [ 0.79753468, -0.85177573],
       [-1.85171546, -0.00818141],
       [-0.18366908,  2.14443859],
       [-0.96863208,  0.25362372],
       [ 0.20881242,  1.06812859],
       [-0.28178945,  0.13726589],
       [-0.0855487 , -0.4445233 ],
       [ 0.01257167, -0.15362871],
       [-1.16487283, -1.17175979],
```

```
        [-1.94983583, -0.06636033],
        [ 0.99377543, -1.08449141],
        [-1.36111358, -0.4445233 ],
        [-1.94983583, -0.53179168],
        [ 0.89565505, -1.46265438],
        [-1.75359508, -0.61906006],
        [ 0.60129393,  1.99899129],
        [-0.87051171, -0.26998655],
        [-0.67427095,  0.02090805],
        [ 0.99377543, -0.85177573],
        [-0.37990983, -0.79359682],
        [-1.26299321,  0.25362372],
        [ 1.4843773 ,  0.3408921 ],
        [ 0.01257167, -0.4445233 ],
        [-1.26299321,  0.28271318],
        [-0.0855487 ,  0.28271318],
        [-1.06675246, -1.14267033],
        [ 2.17121993,  0.92268129],
        [-1.16487283,  1.38811264],
        [-0.67427095,  0.10817643],
        [-0.67427095,  0.16635535],
        [ 0.3069328 , -0.56088114],
        [-0.28178945, -0.38634438],
        [ 1.38625693,  0.57360778],
        [-0.96863208,  0.4863394 ],
        [-0.96863208, -0.32816546],
        [-1.06675246,  1.94081237],
        [ 0.40505317,  0.57360778],
        [ 0.89565505,  2.14443859],
        [ 0.11069205, -0.32816546],
        [-0.4780302 ,  1.24266535],
        [ 1.38625693,  1.96990183],
        [-1.85171546,  0.42816048],
        [-1.06675246, -0.35725492],
        [-1.45923396, -1.46265438],
        [ 0.89565505, -1.05540195],
        [-0.28178945, -0.5899706 ],
        [ 1.77873843,  1.82445454],
        [ 1.58249768, -1.28811763],
        [-0.28178945, -0.67723898],
        [-0.0855487 ,  0.22453427]])
```

[13]:
```
x_test
```

[13]:
```
array([[ 0.79753468, -1.40447546],
       [ 2.07309956,  0.51542886],
       [-0.96863208, -0.76450736],
```

```
[ 0.99377543,  0.74814454],
[-0.87051171, -1.22993871],
[-0.77239133, -0.24089709],
[ 0.89565505,  1.06812859],
[-0.87051171,  0.36998156],
[ 0.20881242,  0.13726589],
[ 0.40505317, -0.15362871],
[-0.28178945, -0.15362871],
[ 1.4843773 , -1.05540195],
[-1.45923396, -0.64814952],
[-1.75359508, -1.37538601],
[-0.77239133,  0.4863394 ],
[-0.28178945,  1.09721805],
[ 1.38625693, -0.93904411],
[ 0.79753468,  0.10817643],
[ 0.11069205, -0.82268628],
[ 1.77873843, -0.29907601],
[-1.55735433, -1.25902817],
[-0.87051171,  0.28271318],
[ 0.89565505, -1.37538601],
[ 2.07309956,  0.16635535],
[-1.85171546, -1.49174384],
[ 1.28813655, -1.37538601],
[ 0.40505317,  0.28271318],
[-0.0855487 , -0.50270222],
[ 1.68061805,  1.59173886],
[-1.85171546, -1.43356492],
[ 0.79753468, -0.85177573],
[-1.85171546, -0.00818141],
[-0.18366908,  2.14443859],
[-0.96863208,  0.25362372],
[ 0.20881242,  1.06812859],
[-0.28178945,  0.13726589],
[-0.0855487 , -0.4445233 ],
[ 0.01257167, -0.15362871],
[-1.16487283, -1.17175979],
[-1.94983583, -0.06636033],
[ 0.99377543, -1.08449141],
[-1.36111358, -0.4445233 ],
[-1.94983583, -0.53179168],
[ 0.89565505, -1.46265438],
[-1.75359508, -0.61906006],
[ 0.60129393,  1.99899129],
[-0.87051171, -0.26998655],
[-0.67427095,  0.02090805],
[ 0.99377543, -0.85177573],
[-0.37990983, -0.79359682],
```

```
        [-1.26299321,  0.25362372],
        [ 1.4843773 ,  0.3408921 ],
        [ 0.01257167, -0.4445233 ],
        [-1.26299321,  0.28271318],
        [-0.0855487 ,  0.28271318],
        [-1.06675246, -1.14267033],
        [ 2.17121993,  0.92268129],
        [-1.16487283,  1.38811264],
        [-0.67427095,  0.10817643],
        [-0.67427095,  0.16635535],
        [ 0.3069328 , -0.56088114],
        [-0.28178945, -0.38634438],
        [ 1.38625693,  0.57360778],
        [-0.96863208,  0.4863394 ],
        [-0.96863208, -0.32816546],
        [-1.06675246,  1.94081237],
        [ 0.40505317,  0.57360778],
        [ 0.89565505,  2.14443859],
        [ 0.11069205, -0.32816546],
        [-0.4780302 ,  1.24266535],
        [ 1.38625693,  1.96990183],
        [-1.85171546,  0.42816048],
        [-1.06675246, -0.35725492],
        [-1.45923396, -1.46265438],
        [ 0.89565505, -1.05540195],
        [-0.28178945, -0.5899706 ],
        [ 1.77873843,  1.82445454],
        [ 1.58249768, -1.28811763],
        [-0.28178945, -0.67723898],
        [-0.0855487 ,  0.22453427]])
```

[14]:
```
#   Building The Model -->

model = GaussianNB()
model.fit(x_train, y_train)
```

[14]:
```
GaussianNB()
```

[16]:
```
#   Predicting Result -->

y_pred = model.predict(x_test)
y_pred
```

[16]:
```
array([1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0], dtype=int64)
```

[18]:
```python
#   Checking Accuracy -->

acc_score = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
```

[20]:
```python
print("Accuracy Score --> ", acc_score)
```

```
Accuracy Score -->  0.9375
```

[21]:
```python
print("Mean Square Error --> ", mse)
```

```
Mean Square Error -->  0.0625
```

[22]:
```python
print("Confusion Matrix -->\n\n", conf_matrix)
```

```
Confusion Matrix -->

 [[50  2]
 [ 3 25]]
```

[23]:
```python
print("Classification Report -->\n\n", class_report)
```

```
Classification Report -->

              precision    recall  f1-score   support

           0       0.94      0.96      0.95        52
           1       0.93      0.89      0.91        28

    accuracy                           0.94        80
   macro avg       0.93      0.93      0.93        80
weighted avg       0.94      0.94      0.94        80
```