Day55_Naive_Bayes_Classifier

July 30, 2025

Naive Bayes Classifier

Naive Bayes is a family of **probabilistic classifiers** based on Bayes' Theorem, with the "naive" assumption of conditional independence between features.

It works surprisingly well for many real-world problems like spam detection, sentiment analysis, and recommendation systems.

Bayes Theorem

Bayes' Theorem allows us to reverse conditional probabilities:

P(A|B) = (P(B|A) * P(A)) / P(B)

- **P(A|B)**: Posterior Probability (What we want to find)
- **P(B|A)**: Likelihood
- **P(A)**: Prior Probability
- **P(B)**: Marginal Probability

Types of Naive Bayes:

1. **GaussianNB** – works with continuous data, assumes Gaussian (normal) distribution.
2. **BernoulliNB** – works with binary data (0/1 features).
3. **MultinomialNB** – works with counts (e.g., word frequencies in NLP).

Real-Life Examples

- Booking Sites: "80% seats sold out" → Predict urgency using probabilities
- Emails: Classify as spam or not based on keywords
- Sentiment Analysis: Based on word distribution, classify a review as positive or negative

Real-Life Example: Manual Naive Bayes Calculation – Spam Classification

Problem:

We want to classify whether a new email is **Spam** or **Not Spam** based on whether it contains the words `"offer"` and `"win"`.

Step 1: Training Data

Email ID	offer	win	Class
E1	1	1	Spam
E2	1	0	Spam
E3	0	1	Not Spam
E4	0	0	Not Spam

Email ID	offer	win	Class
E5	1	1	Spam

Step 2: New Email to Predict

- The email contains both **"offer"** and **"win"**
- Features: offer = 1, win = 1

We calculate:

- P(Spam | offer=1, win=1)
- P(Not Spam | offer=1, win=1)

Step 3: Naive Bayes Components

Prior Probabilities

- P(Spam) = 3/5 = 0.6
- P(Not Spam) = 2/5 = 0.4

Likelihoods

Among Spam emails (3 total): - P(offer=1 | Spam) = 3/3 = 1.0
- P(win=1 | Spam) = 2/3 0.67

Among Not Spam emails (2 total):

- P(offer=1 | Not Spam) = 0/2 = 0
- P(win=1 | Not Spam) = 1/2 = 0.5

Step 4: Naive Bayes Formula (Ignoring denominator):

P(Spam | offer=1, win=1) P(offer=1 | Spam) $\times$ P(win=1 | Spam) $\times$ P(Spam)
= 1.0 $\times$ 0.67 $\times$ 0.6 = **0.402**

No P(Not Spam | offer=1, win=1)
= 0 $\times$ 0.5 $\times$ 0.4 = **0**

Final Prediction:

Since **0.402 > 0**, the email is predicted to be **SPAM**.

Note: Laplace Smoothing (Bonus)

To avoid zero probabilities: - P(offer=1 | Not Spam) = (0 + 1) / (2 + 2) = 0.25

Smoothing avoids multiplying by 0 and improves model stability.

1 Importing Libraries

[2]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.metrics import accuracy_score, confusion_matrix,␣
  ↪classification_report
import warnings
warnings.filterwarnings('ignore')
```

2 Load and Explore Dataset

[3]:
```
# Load dataset
dataset = pd.read_csv(r"C:\Users\Lenovo\Downloads\logit classification.csv")
dataset.head()
```

[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

3 Feature Selection and Splitting

[4]:
```
X = dataset[["Age", "EstimatedSalary"]].values
y = dataset["Purchased"].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=42)
```

4 Feature Scaling

4.1 Standardization (for Gaussian & Bernoulli)

[5]:
```
# Standardization (for Gaussian & Bernoulli)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

4.2 MinMax Scaling (for MultinomialNB)

[6]:
```
minmax = MinMaxScaler()
X_train_minmax = minmax.fit_transform(X_train)
X_test_minmax = minmax.transform(X_test)
```

5 Model Training and Evaluation

5.1 GaussianNB

[7]:
```
print("=== GaussianNB ===")
gnb = GaussianNB()
gnb.fit(X_train, y_train)
y_pred_gnb = gnb.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred_gnb))
print(confusion_matrix(y_test, y_pred_gnb))
print(classification_report(y_test, y_pred_gnb))
```

```
=== GaussianNB ===
Accuracy: 0.925
[[50  2]
 [ 4 24]]
              precision    recall  f1-score   support

           0       0.93      0.96      0.94        52
           1       0.92      0.86      0.89        28

    accuracy                           0.93        80
   macro avg       0.92      0.91      0.92        80
weighted avg       0.92      0.93      0.92        80
```

5.2 BernoulliNB

[8]:
```
print("=== BernoulliNB ===")
bnb = BernoulliNB()
bnb.fit(X_train_scaled, y_train)
y_pred_bnb = bnb.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, y_pred_bnb))
print(confusion_matrix(y_test, y_pred_bnb))
print(classification_report(y_test, y_pred_bnb))
```

```
=== BernoulliNB ===
Accuracy: 0.7875
[[49  3]
 [14 14]]
              precision    recall  f1-score   support
```

```
           0       0.78      0.94      0.85        52
           1       0.82      0.50      0.62        28

    accuracy                           0.79        80
   macro avg       0.80      0.72      0.74        80
weighted avg       0.79      0.79      0.77        80
```

5.3 MultinomialNB

[9]:
```
print("=== MultinomialNB ===")
mnb = MultinomialNB()
mnb.fit(X_train_minmax, y_train)
y_pred_mnb = mnb.predict(X_test_minmax)
print("Accuracy:", accuracy_score(y_test, y_pred_mnb))
print(confusion_matrix(y_test, y_pred_mnb))
print(classification_report(y_test, y_pred_mnb))
```

```
=== MultinomialNB ===
Accuracy: 0.65
[[52  0]
 [28  0]]
              precision    recall  f1-score   support

           0       0.65      1.00      0.79        52
           1       0.00      0.00      0.00        28

    accuracy                           0.65        80
   macro avg       0.33      0.50      0.39        80
weighted avg       0.42      0.65      0.51        80
```

6 Observations

- **GaussianNB** works best for continuous data like Age and Salary.
- **BernoulliNB** assumes binary features — accuracy may be lower here.
- **MultinomialNB** requires positive features — MinMaxScaler helps here.

Always choose the right NB variant depending on the feature types in your dataset

7 Summary: Naive Bayes Classifiers

In this notebook, we explored and implemented three types of Naive Bayes algorithms:

7.1 GaussianNB

- Best suited for continuous features like **Age** and **Salary**
- Achieved **92.5% accuracy**

- Provided a balanced performance with high precision and recall
- **Recommended for this dataset**

7.2 BernoulliNB

- Designed for **binary features** (0 or 1)
- Achieved **78.75% accuracy**
- Performed well for class 0, but poorly for class 1
- **Not ideal** for this type of dataset

7.3 MultinomialNB

- Designed for **count-based features** (e.g., word frequencies)
- Achieved **65% accuracy**
- Completely failed to predict class 1
- **Not suitable** for continuous features

7.4 Key Takeaways

- Naive Bayes is a **fast and simple** algorithm with solid performance for classification tasks.
- Choosing the **right variant (Gaussian, Bernoulli, Multinomial)** is critical based on your feature types.
- Always consider **scaling**, especially for BernoulliNB and GaussianNB, to improve performance.
- **GaussianNB** gave the best results in our case due to the nature of our numerical features.

7.5 Real-Life Tip:

Naive Bayes is commonly used in **spam detection, recommendation systems, and text classification**. Even though it makes strong independence assumptions, it performs surprisingly well in many real-world situations.