TRIBHUVAN UNIVERSITY

Institute of Science and Technology (IoST)

Samriddhi College

A Lab Report on Artificial Intelligence

LAB 7: Naive Bayes Classification for Email Spam Detection

Submitted To:

Bikram Acharya

Submitted By:

Kanchan Joshi (Roll no.10)

Date: October 04, 2025

1. Introduction

Naive Bayes classifiers represent a family of probabilistic algorithms grounded in Bayes' theorem, widely applied in tasks such as text categorization and email spam detection. These models assume conditional independence among feature variables-a simplification that, despite rarely holding perfectly in real-world data, enables efficient computation and often yields robust performance, particularly with high-dimensional or limited datasets.

This laboratory exercise implements a from-scratch Naive Bayes classifier for binary email spam detection using categorical features from email metadata. The approach demonstrates key probabilistic concepts, including prior estimation, likelihood computation, and posterior maximization, while providing interpretable insights into classification decisions. By focusing on a simulated spam dataset, the implementation highlights the algorithm's strengths in handling categorical data and its role as a baseline for more advanced machine learning techniques.

The complete source code, dataset, and experimental procedures are maintained in a dedicated repository for reproducibility and further exploration.

1.1 Theoretical Foundation

Bayes' Theorem

The core of Naive Bayes is Bayes' theorem, which updates the probability of a hypothesis based on new evidence:

$$P(c_k \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c_k)\, P(c_k)}{P(\mathbf{x})} \tag{1.1}$$

Here:

- $P(c_k \mid \mathbf{x})$: Posterior probability of class c_k given features $\mathbf{x}$.
- $P(\mathbf{x} \mid c_k)$: Likelihood of observing $\mathbf{x}$ under class c_k.
- $P(c_k)$: Prior probability of class c_k.

- $P(\mathbf{x})$: Marginal probability (evidence) of $\mathbf{x}$.

Independence Assumption

Naive Bayes assumes features are conditionally independent given the class:

$$P(x_1, x_2, \ldots, x_n \mid c_k) = \prod_{i=1}^{n} P(x_i \mid c_k) \tag{1.2}$$

This reduces complexity from exponential to linear in the number of features, making the model scalable, though it may introduce bias in correlated data.

Decision Rule

Classification selects the class maximizing the posterior (MAP estimate):

$$\hat{c} = \arg\max_{c_k} P(c_k \mid \mathbf{x}) = \arg\max_{c_k} P(c_k) \prod_{i=1}^{n} P(x_i \mid c_k) \tag{1.3}$$

The evidence $P(\mathbf{x})$ is omitted as it is constant across classes.

2. Problem Definition and Dataset

2.1 Dataset Description

The dataset simulates email metadata for spam detection with four categorical features and a binary target:

- **Sender**: Known (trusted sender) or Unknown.
- **Subject_Length**: Short, Medium, or Long.
- **Has_Attachments**: Yes or No.
- **Time_Sent**: Day or Night.
- **Target (Spam)**: Yes (spam) or No (legitimate).

These features capture common spam indicators: unknown senders, lengthy subjects, attachments, and off-hour sends. The dataset comprises 14 balanced instances (7 spam, 7 non-spam) for illustrative purposes.

2.2 Dataset Structure

The full dataset is presented below:

Index	Sender	Subject_Length	Has_Attachments	Time_Sent	Spam
0	Unknown	Short	Yes	Night	Yes
1	Unknown	Long	No	Day	Yes
2	Known	Medium	No	Day	No
3	Unknown	Short	Yes	Night	Yes
4	Known	Medium	No	Day	No
5	Unknown	Long	Yes	Night	Yes
6	Known	Short	No	Day	No
7	Unknown	Long	Yes	Night	Yes
8	Known	Medium	No	Day	No

Index	Sender	Subject_Length	Has_Attachments	Time_Sent	Spam
9	Unknown	Short	Yes	Night	Yes
10	Unknown	Long	No	Day	No
11	Known	Medium	No	Day	No
12	Known	Short	No	Day	No
13	Unknown	Long	Yes	Night	Yes

Table 2.1: Email Spam Detection Dataset

Data loading in Python (using Pandas for simulation):

```python
import pandas as pd
import numpy as np

# Simulated dataset
data = {
    'Sender': ['Unknown', 'Unknown', 'Known', 'Unknown', 'Known', 'Unknown', 'Known',
               'Unknown', 'Known', 'Unknown', 'Unknown', 'Known', 'Known', 'Unknown'],
    'Subject_Length': ['Short', 'Long', 'Medium', 'Short', 'Medium', 'Long', 'Short',
                       'Long', 'Medium', 'Short', 'Long', 'Medium', 'Short', 'Long'],
    'Has_Attachments': ['Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes',
                        'No', 'No', 'No', 'Yes'],
    'Time_Sent': ['Night', 'Day', 'Day', 'Night', 'Day', 'Night', 'Day', 'Night', 'Day',
                  'Night', 'Day', 'Day', 'Day', 'Night'],
    'Spam': ['Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No',
             'No', 'Yes']
}
df = pd.DataFrame(data)
X = df.drop('Spam', axis=1)
y = df['Spam']
print(df.shape) # Output: (14, 5)
```

Code 2.1: Dataset Loading

3. Probabilistic Classification Algorithm Development

The implementation uses NumPy and Pandas for data handling, focusing on categorical features without external libraries like scikit-learn for pedagogical value.

3.1 Algorithm Architecture and Initialization

The `NaiveBayes` class initializes dictionaries for probabilities and stores training data.

```python
class NaiveBayes:
   def __init__(self):
      self.features = []
      self.likelihoods = {}
      self.class_priors = {}
      self.pred_priors = {}
      self.X_train = None
      self.y_train = None
      self.train_size = 0
      self.num_feats = 0
```

Code 3.1: Naive Bayes Class Framework

3.2 Model Training Procedure

The `fit` method computes priors and likelihoods.

```python
   def fit(self, X, y):
      self.features = list(X.columns)
      self.X_train = X
      self.y_train = y
      self.train_size = X.shape[0]
      self.num_feats = X.shape[1]

```

```
    # Initialize structures
    for feature in self.features:
        self.likelihoods[feature] = {}
        self.pred_priors[feature] = {}
        unique_vals = np.unique(self.X_train[feature])
        for val in unique_vals:
            self.pred_priors[feature][val] = 0
        for outcome in np.unique(self.y_train):
            for val in unique_vals:
                self.likelihoods[feature][val + '_' + outcome] = 0
            self.class_priors[outcome] = 0

    self._calc_class_prior()
    self._calc_likelihoods()
    self._calc_predictor_prior()
```

Code 3.2: Fit Implementation

3.3 Prior Probability Computation

Class priors $P(c_k)$ are empirical frequencies.

```
def _calc_class_prior(self):
    for outcome in np.unique(self.y_train):
        count = sum(self.y_train == outcome)
        self.class_priors[outcome] = count / self.train_size
```

Code 3.3: Class Prior Calculation

Output: `{'No': 0.5, 'Yes': 0.5}`

3.4 Likelihood Probability Computation

Likelihoods $P(x_i \mid c_k)$ are conditional frequencies.

```
def _calc_likelihoods(self):
    for feature in self.features:
        for outcome in np.unique(self.y_train):
            outcome_mask = self.y_train == outcome
            outcome_count = sum(outcome_mask)
            if outcome_count > 0:
                feat_dist = self.X_train.loc[outcome_mask, feature].value_counts()
```

```
            for val, count in feat_dist.items():
                self.likelihoods[feature][val + '_' + outcome] =
                    count / outcome_count
```

Code 3.4: Likelihood Calculation

3.5 Evidence Probability Computation

Marginals $P(x_i)$ are overall frequencies.

```
def _calc_predictor_prior(self):
    for feature in self.features:
        feat_dist = self.X_train[feature].value_counts()
        for val, count in feat_dist.items():
            self.pred_priors[feature][val] = count / self.train_size
```

Code 3.5: Marginal Prior Calculation

Example marginals (Sender feature): `{'Known': 0.4286, 'Unknown': 0.5714}`

3.6 Classification Prediction Method

Predictions maximize unnormalized posteriors to avoid underflow.

```
def predict(self, X):
    results = []
    X = pd.DataFrame(X, columns=self.features)
    for _, query in X.iterrows():
        posteriors = {}
        for outcome in np.unique(self.y_train):
            prior = self.class_priors[outcome]
            likelihood = 1.0
            for feat, val in zip(self.features, query):
                likelihood *= self.likelihoods[feat].get(val + '_' +
                    outcome, 1e-9) # Smoothing
            posterior = prior * likelihood
            posteriors[outcome] = posterior
        pred = max(posteriors, key=posteriors.get)
        results.append(pred)
    return np.array(results)
```

Code 3.6: Prediction Method

4. Experimental Evaluation and Performance Analysis

4.1 Model Training and Accuracy

Training on the full dataset yields:

```
from sklearn.metrics import accuracy_score # For comparison

nb_clf = NaiveBayes()
nb_clf.fit(X, y)
y_pred = nb_clf.predict(X)
accuracy = accuracy_score(y, y_pred)
print(f"Training Accuracy: {accuracy:.2%}")
```

Code 4.1: Training and Evaluation

Output: `Training Accuracy: 92.86%`

4.2 Query Example

For a new email ['Known', 'Short', 'No', 'Day']:

```
query = np.array([['Known', 'Short', 'No', 'Day']])
pred = nb_clf.predict(query)
print(f"Prediction: {pred[0]}")
```

Code 4.2: Single Prediction

Output: `Prediction:` No (Posterior: No=0.1224, Yes=0.0)

4.3 Comprehensive Probability Analysis

Posteriors for all instances:

Index	$P(\text{No} \mid \mathbf{x})$	$P(\text{Yes} \mid \mathbf{x})$	Prediction
0	0.0000	0.1574	Yes
1	0.0102	0.0058	No
2	0.2449	0.0000	No
3	0.0000	0.1574	Yes
4	0.2449	0.0000	No
5	0.0000	0.2099	Yes
6	0.1224	0.0000	No
7	0.0000	0.2099	Yes
8	0.2449	0.0000	No
9	0.0000	0.1574	Yes
10	0.0102	0.0058	No
11	0.2449	0.0000	No
12	0.1224	0.0000	No
13	0.0000	0.2099	Yes

Table 4.1: Posterior Probabilities and Predictions

4.4 Feature Impact Analysis

Unknown senders and nighttime sends strongly correlate with spam (likelihood > 0.8 for Yes). Attachments amplify risk when combined with unknowns.

Marginal distributions:

Feature	Value	Marginal $P(x_i)$
Sender	Known	0.43
	Unknown	0.57
Subject_Length	Short	0.36
	Medium	0.29
	Long	0.36
Has_Attachments	No	0.57
	Yes	0.43
Time_Sent	Day	0.57
	Night	0.43

Table 4.2: Feature Marginal Probabilities

5. Conclusion

This implementation demonstrates Naive Bayes' efficacy for spam detection, achieving 92.86% accuracy on a balanced dataset through transparent probabilistic modeling. Strengths include scalability and interpretability, while limitations-such as the independence assumption and zero-probability risks-suggest enhancements like Laplace smoothing or ensemble methods for production use.

Future work could extend to real corpora (e.g., Enron dataset) and incorporate continuous features via Gaussian Naive Bayes.