Lab 11.5: Text Classification using Naive Bayes – Product Review Classification

Learning Objectives

- Understand text classification in NLP
- Perform text preprocessing
- Convert text into numerical features
- Train Naive Bayes classifier
- Evaluate model using metrics

```
# STEP 2 — Import Libraries

import numpy as np          # for numerical operations
import pandas as pd         # for data handling
import re                   # for text cleaning
import string               # for punctuation removal

from sklearn.model_selection import train_test_split   # to split dataset
from sklearn.feature_extraction.text import TfidfVectorizer  # to convert
from sklearn.naive_bayes import MultinomialNB          # Naive Bayes mode
from sklearn.metrics import accuracy_score, precision_score, recall_score
```

```
# STEP 3 — Load Dataset

# Sample dataset (students can replace with Kaggle/Amazon/Flipkart dataset)
data = {
    "review": [
        "Great product, really loved it",
        "Worst purchase ever",
        "Average quality, not bad",
        "Excellent! Highly recommended",
        "Not worth the money",
        "Very happy with this product",
        "Terrible experience",
        "Okayish performance",
        "Superb build quality",
        "Bad packaging and poor quality"
    ],
    "sentiment": ["Positive", "Negative", "Neutral", "Positive", "Negative",
}
```

```
df = pd.DataFrame(data)

# Display first 5 rows
df.head()
```

	review	sentiment
0	Great product, really loved it	Positive
1	Worst purchase ever	Negative
2	Average quality, not bad	Neutral
3	Excellent! Highly recommended	Positive
4	Not worth the money	Negative

Next steps: Generate code with df New interactive sheet

```
# Dataset size and class distribution
print("Dataset size:", df.shape)
print("\nClass distribution:")
print(df['sentiment'].value_counts())
```

```
Dataset size: (10, 2)

Class distribution:
sentiment
Positive    4
Negative    4
Neutral     2
Name: count, dtype: int64
```

```
# STEP 4 — Text Preprocessing

def preprocess_text(text):
    text = text.lower()  # lowercase
    text = re.sub(f"[{string.punctuation}]", "", text)  # remove punctuation
    return text

df['clean_review'] = df['review'].apply(preprocess_text)

df[['review', 'clean_review']].head()
```

	review	clean_review
0	Great product, really loved it	great product really loved it
1	Worst purchase ever	worst purchase ever
2	Average quality, not bad	average quality not bad
3	Excellent! Highly recommended	excellent highly recommended
4	Not worth the money	not worth the money

```
# STEP 5 — Feature Extraction using TF-IDF

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(df['clean_review'])

y = df['sentiment']

print("Feature matrix shape:", X.shape)
print("Sample feature names:", vectorizer.get_feature_names_out()[:10])
```

```
Feature matrix shape: (10, 23)
Sample feature names: ['average' 'bad' 'build' 'excellent' 'experience' 'grea
 'loved' 'money']
```

```
# STEP 6 — Train-Test Split

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

print("Training size:", X_train.shape)
print("Testing size:", X_test.shape)
```

```
Training size: (8, 23)
Testing size: (2, 23)
```

```
# STEP 7 — Train Naive Bayes Model

model = MultinomialNB()
model.fit(X_train, y_train)

print("Model trained successfully!")
```

```
Model trained successfully!
```

```
# STEP 8 — Model Evaluation

y_pred = model.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1 Score:", f1_score(y_test, y_pred, average='weighted'))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.5
Precision: 0.25
Recall: 0.5
F1 Score: 0.3333333333333333

Confusion Matrix:
[[1 0]
 [1 0]]

Classification Report:
              precision    recall  f1-score   support

    Negative       0.50      1.00      0.67         1
    Positive       0.00      0.00      0.00         1

    accuracy                           0.50         2
   macro avg       0.25      0.50      0.33         2
weighted avg       0.25      0.50      0.33         2

/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:15
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:15
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:15
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.12/dist-packages/sklearn/metrics/_classification.py:15
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

STEP 9 — Result Analysis

- The Naive Bayes classifier shows decent performance on small dataset.
- Positive and Negative classes are easier to distinguish.
- Neutral class may overlap with other sentiments.
- Misclassification occurs due to limited training data.
- Naive Bayes assumes conditional independence which is not always true.
- TF-IDF improves feature representation.

- Larger datasets will improve performance.
- Informal text and mixed sentiments make classification harder.