Objective

The goal of this lab is to implement N-gram modeling in Python using Google Colab to perform sentiment analysis on movie reviews. The task involved transforming raw text into numerical representations using unigram, bigram, and trigram models, training a Multinomial Naive Bayes classifier, and evaluating how increasing the N-gram range affects model performance, feature dimensionality, and runtime.

```
import nltk
nltk.download('movie_reviews')
```

```
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!
True
```

Dataset Description

For this lab, I used the movie_reviews dataset available in the NLTK library. The dataset contains 2,000 movie reviews, evenly divided into positive and negative sentiments (1,000 positive and 1,000 negative).

Each review is labeled as:

1 - Positive

0 - Negative

This dataset is commonly used for benchmarking sentiment analysis models.

```
from nltk.corpus import movie_reviews
import pandas as pd

documents = []
labels = []

for fileid in movie_reviews.fileids():
    documents.append(movie_reviews.raw(fileid))
    labels.append(movie_reviews.categories(fileid)[0])

df = pd.DataFrame({
    "review": documents,
    "sentiment": labels
})

# Convert labels to numeric
df["sentiment"] = df["sentiment"].map({"pos": 1, "neg": 0})

df.head()
```

	review	sentiment
0	plot : two teen couples go to a church party ,...	0
1	the happy bastard's quick movie review \ndamn ...	0
2	it is movies like these that make a jaded movi...	0
3	" quest for camelot " is warner bros . ' firs...	0
4	synopsis : a mentally unstable man undergoing ...	0

Double-click (or enter) to edit

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    df["review"],
    df["sentiment"],
    test_size=0.25,
    random_state=42,
    stratify=df["sentiment"]
)

print(len(X_train), len(X_test))
```

```
1500 500
```

Methodology

The following steps were performed:

1. Load the movie review dataset from NLTK.
2. Split the dataset into training and testing sets.
3. Convert the reviews into numerical form using CountVectorizer.
4. Build three models:

   Unigram (1-gram)

   Bigram (1–2 grams)

   Trigram (1–3 grams)
5. Train a Multinomial Naive Bayes classifier.
6. Evaluate model performance using -

   Accuracy

   Precision, Recall, F1-score

   Confusion Matrix

   Feature dimensionality

   Runtime comparison

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_uni = CountVectorizer(
    lowercase=True,
    stop_words="english",
    ngram_range=(1,1)
)

X_train_uni = vectorizer_uni.fit_transform(X_train)
X_test_uni = vectorizer_uni.transform(X_test)

print("Unigram feature count:", X_train_uni.shape[1])
```

```
Unigram feature count: 35161
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
clf_uni = MultinomialNB()
clf_uni.fit(X_train_uni, y_train)
```

```
▾ MultinomialNB
MultinomialNB()
```

N-gram modeling represents text as contiguous sequences of N words. While unigram models treat words independently, bigram and trigram models preserve partial word order and contextual meaning.

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

y_pred_uni = clf_uni.predict(X_test_uni)

print("Unigram Accuracy:", accuracy_score(y_test, y_pred_uni))
print("\nClassification Report:\n", classification_report(y_test, y_pred_uni))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_uni))
```

```
Unigram Accuracy: 0.8

Classification Report:
               precision    recall  f1-score   support

           0       0.80      0.79      0.80       250
           1       0.80      0.81      0.80       250

    accuracy                           0.80       500
   macro avg       0.80      0.80      0.80       500
weighted avg       0.80      0.80      0.80       500


Confusion Matrix:
 [[198  52]
 [ 48 202]]
```

Unigram Model Results

The unigram model considers individual words as features. After training the Naive Bayes classifier, the accuracy achieved was:

```
Unigram Accuracy: 0.80
```

This means the model correctly classified approximately 80% of the reviews in the test set.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_bi = CountVectorizer(
    lowercase=True,
    stop_words="english",
    ngram_range=(1,2)
)

X_train_bi = vectorizer_bi.fit_transform(X_train)
X_test_bi = vectorizer_bi.transform(X_test)

print("Bigram feature count:", X_train_bi.shape[1])
```

```
Bigram feature count: 415001
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix


clf_bi = MultinomialNB()
clf_bi.fit(X_train_bi, y_train)

y_pred_bi = clf_bi.predict(X_test_bi)

print("Bigram Accuracy:", accuracy_score(y_test, y_pred_bi))

print("Bigram Classification Report:\n", classification_report(y_test, y_pred_bi))
print("Bigram Confusion Matrix:\n", confusion_matrix(y_test, y_pred_bi))
```

```
Bigram Accuracy: 0.794
Bigram Classification Report:
               precision    recall  f1-score   support

           0       0.85      0.71      0.78       250
           1       0.75      0.88      0.81       250

    accuracy                           0.79       500
   macro avg       0.80      0.79      0.79       500
weighted avg       0.80      0.79      0.79       500

Bigram Confusion Matrix:
 [[178  72]
 [ 31 219]]
```

Bigram Model Results

The bigram model includes both single words and two-word combinations. This helps capture contextual phrases such as "not good."

The bigram model achieved:

```
Bigram Accuracy: 0.794
```

The accuracy is slightly lower than the unigram model in this case.

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer_tri = CountVectorizer(
    lowercase=True,
    stop_words="english",
    ngram_range=(1,3)
)

X_train_tri = vectorizer_tri.fit_transform(X_train)
X_test_tri = vectorizer_tri.transform(X_test)

print("Trigram feature count:", X_train_tri.shape[1])
```

```
Trigram feature count: 876984
```

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

clf_tri = MultinomialNB()
clf_tri.fit(X_train_tri, y_train)

y_pred_tri = clf_tri.predict(X_test_tri)
```

```
print("Trigram Accuracy:", accuracy_score(y_test, y_pred_tri))
from sklearn.metrics import

print("Trigram Classification Report:\n", classification_report(y_test, y_pred_tri))
print("Trigram Confusion Matrix:\n", confusion_matrix(y_test, y_pred_tri))
```

```
Trigram Accuracy: 0.786
```

Trigram Model Results

The trigram model includes single words, two-word combinations, and three-word combinations.

The trigram model achieved:

```
Trigram Accuracy: 0.786
```

Although trigram modeling captures more contextual information, it significantly increases the number of features, which can introduce sparsity and reduce performance.

```
results = {
    "Unigram": accuracy_score(y_test, y_pred_uni),
    "Bigram": accuracy_score(y_test, y_pred_bi),
    "Trigram": accuracy_score(y_test, y_pred_tri)
}

results
```

```
{'Unigram': 0.8, 'Bigram': 0.794, 'Trigram': 0.786}
```

Comparative Analysis

As the N-gram range increased:

Feature dimensionality grew dramatically (35k -> 415k -> 876k features)

Runtime increased significantly (1.6s -> 3.5s -> 6.1s)

Accuracy slightly decreased

This demonstrates the curse of dimensionality. Higher-order N-grams generate many rare combinations, leading to sparse feature matrices. With a relatively small dataset (2,000 reviews), the classifier cannot effectively learn from these sparse higher-order patterns.

Although N-grams help capture word order (for example, distinguishing “not good” from “good”), increasing N does not necessarily improve predictive performance when training data is limited.

```
import time
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import MultinomialNB


def run_model(ngram_range):
    start = time.time()

    vectorizer = CountVectorizer(stop_words="english", ngram_range=ngram_range)
    Xtr = vectorizer.fit_transform(X_train)
    Xte = vectorizer.transform(X_test)
    clf = MultinomialNB()
    clf.fit(Xtr, y_train)
    acc = clf.score(Xte, y_test)

    runtime = time.time() - start
    return acc, runtime, Xtr.shape[1]

results = {
    "(1,1)": run_model((1,1)),
    "(1,2)": run_model((1,2)),
    "(1,3)": run_model((1,3))
}

results
```

```
{'(1,1)': (0.8, 1.3129618167877197, 35161),
 '(1,2)': (0.794, 4.493814468383789, 415001),
 '(1,3)': (0.786, 6.0590410232543945, 876984)}
```

```
import pandas as pd

comparison = pd.DataFrame([
    {"Model": "Unigram (1,1)", "Accuracy": results["(1,1)"][0], "Runtime_sec": results["(1,1)"][1], "
    {"Model": "Bigram (1,2)",  "Accuracy": results["(1,2)"][0], "Runtime_sec": results["(1,2)"][1], "
    {"Model": "Trigram (1,3)", "Accuracy": results["(1,3)"][0], "Runtime_sec": results["(1,3)"][1], "
])

comparison
```

	Model	Accuracy	Runtime_sec	Features
0	Unigram (1,1)	0.800	1.312962	35161
1	Bigram (1,2)	0.794	4.493814	415001
2	Trigram (1,3)	0.786	6.059041	876984

Final Analysis and Conclusion

In this lab, I implemented unigram, bigram, and trigram models to perform sentiment analysis on movie reviews using the Multinomial Naive Bayes classifier.

The results show:

```
Unigram Accuracy: 0.80

Bigram Accuracy: 0.794

Trigram Accuracy: 0.786
```

Although higher-order N-grams capture more contextual information, they also dramatically increase the dimensionality of the feature space. With a relatively small dataset (2,000 reviews), the additional complexity did not significantly improve performance.

In this experiment, the unigram model provided the best balance between accuracy and computational efficiency. This demonstrates that while N-gram modeling enhances contextual understanding, increasing N does not always guarantee better results due to sparsity and the curse of dimensionality.

Overall, this lab successfully demonstrates how N-gram modeling can be applied to sentiment analysis using Python and machine learning techniques.