Naive Bayes Classifier - ML Lab

Name: Rohan Suresh

SRN: PES1UG23AM240

AIML 5th SEM D SECTION

1. Introduction

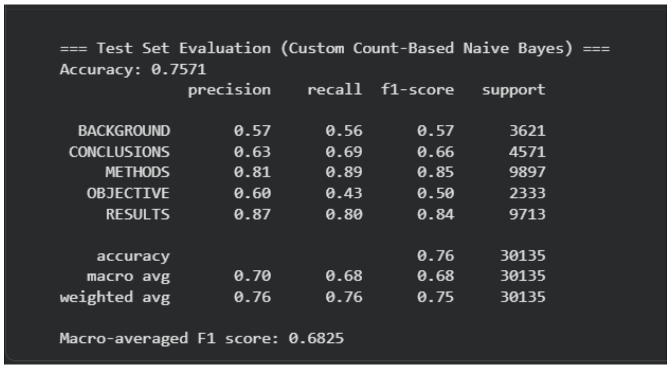
The purpose of this lab is to implement and evaluate probabilistic text classification using the Multinomial Naive Bayes algorithm. The dataset used is a subset of the PubMed 200k RCT dataset, which involves classifying abstract sentences into categories such as BACKGROUND, METHODS, RESULTS, OBJECTIVE, and CONCLUSION. The tasks include implementing Naive Bayes from scratch, tuning parameters using scikit-learn, and constructing an ensemble approximation of the Bayes Optimal Classifier.

2. Methodology

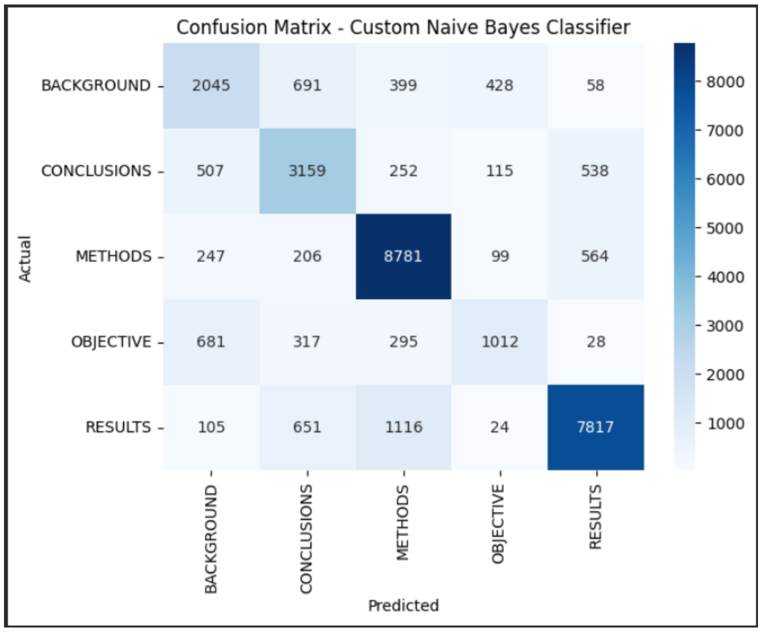
This lab was divided into three main parts as described below.

Part A: Multinomial Naive Bayes from Scratch

A custom implementation of the Multinomial Naive Bayes classifier was created to understand its inner workings. CountVectorizer was used to extract features from the text. Laplace smoothing (α = 1.0) was applied to avoid zero probabilities. The classifier calculates log priors and log likelihoods for each class, and predictions are made using argmax over log probabilities.

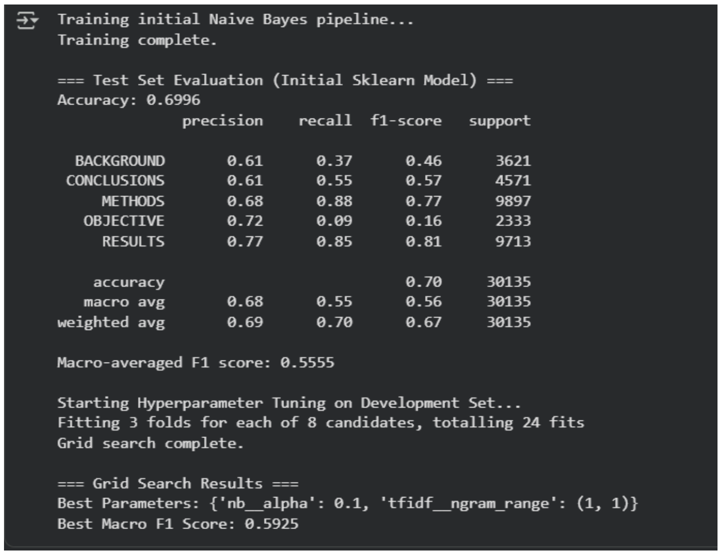


```
=== Test Set Evaluation (Custom Count-Based Naive Bayes) ===
Accuracy: 0.7571
              precision    recall  f1-score   support

  BACKGROUND       0.57      0.56      0.57      3621
 CONCLUSIONS       0.63      0.69      0.66      4571
     METHODS       0.81      0.89      0.85      9897
   OBJECTIVE       0.60      0.43      0.50      2333
     RESULTS       0.87      0.80      0.84      9713

    accuracy                           0.76     30135
   macro avg       0.70      0.68      0.68     30135
weighted avg       0.76      0.76      0.75     30135

Macro-averaged F1 score: 0.6825
```

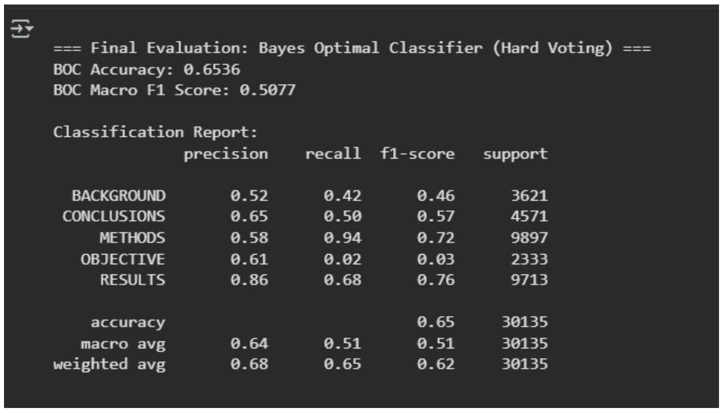


Part B: Sklearn MultinomialNB and Hyperparameter Tuning

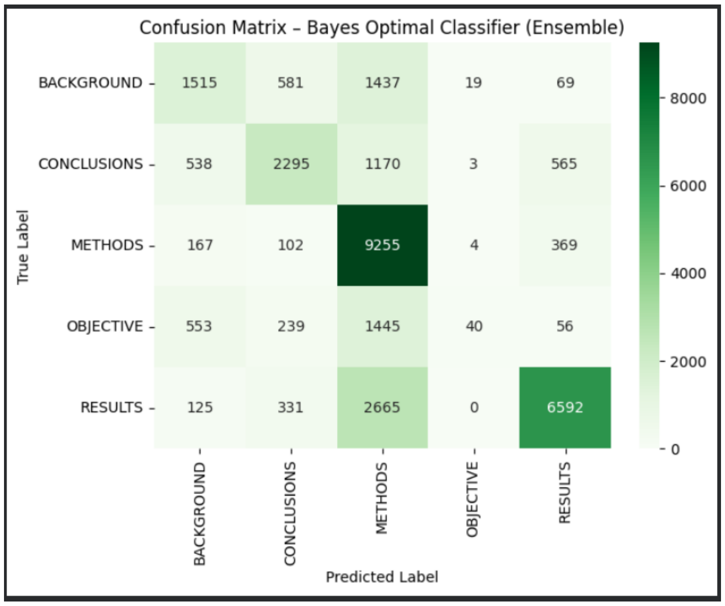
A scikit-learn pipeline combining TfidfVectorizer and MultinomialNB was implemented. GridSearchCV was used to optimize hyperparameters such as the n-gram range and smoothing parameter (alpha). The search used 3-fold cross-validation on the development set, and the best parameters were used to evaluate the model on the test set.



```
Training initial Naive Bayes pipeline...
Training complete.

=== Test Set Evaluation (Initial Sklearn Model) ===
Accuracy: 0.6996
              precision    recall  f1-score   support

  BACKGROUND       0.61      0.37      0.46      3621
 CONCLUSIONS       0.61      0.55      0.57      4571
     METHODS       0.68      0.88      0.77      9897
   OBJECTIVE       0.72      0.09      0.16      2333
     RESULTS       0.77      0.85      0.81      9713

    accuracy                           0.70     30135
   macro avg       0.68      0.55      0.56     30135
weighted avg       0.69      0.70      0.67     30135

Macro-averaged F1 score: 0.5555

Starting Hyperparameter Tuning on Development Set...
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Grid search complete.

=== Grid Search Results ===
Best Parameters: {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 1)}
Best Macro F1 Score: 0.5925
```

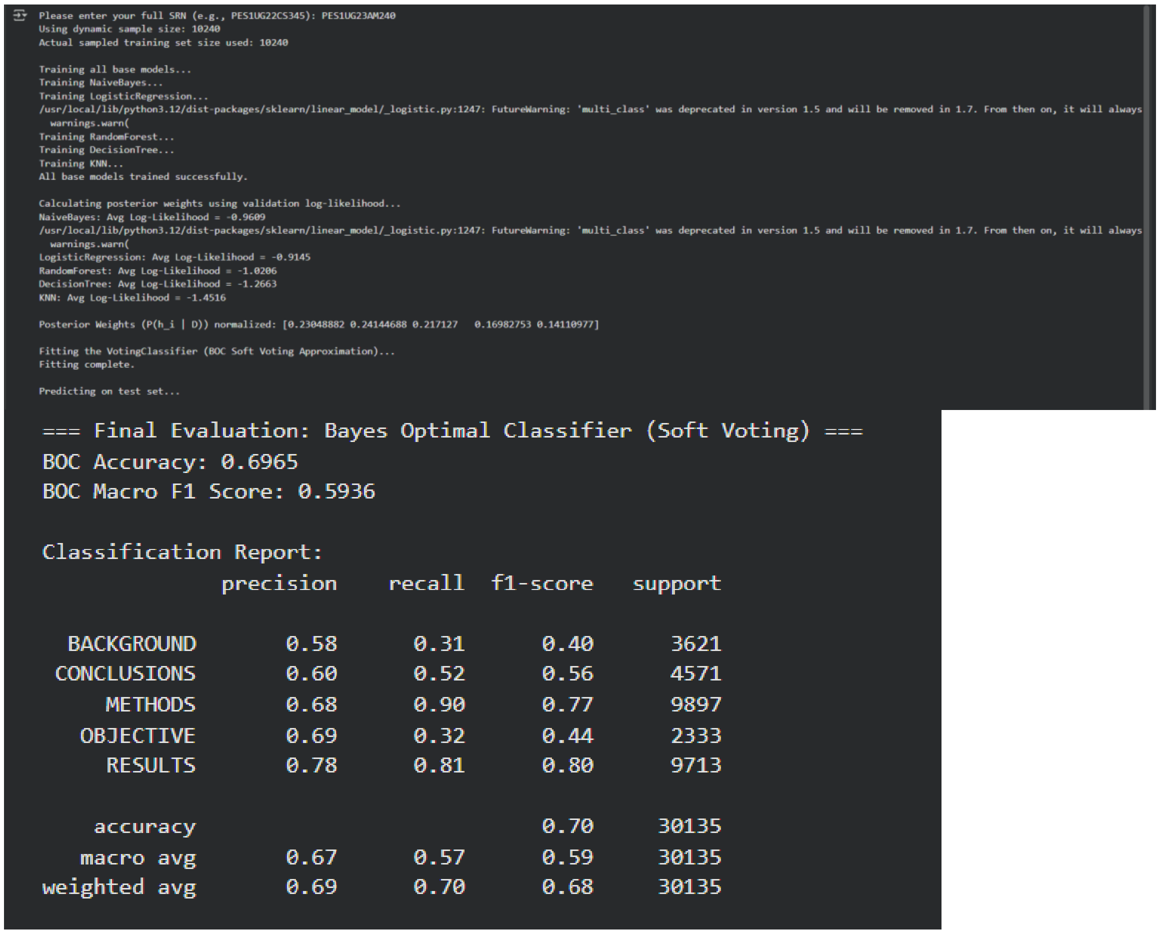


```
=== Final Evaluation: Bayes Optimal Classifier (Hard Voting) ===
BOC Accuracy: 0.6536
BOC Macro F1 Score: 0.5077

Classification Report:
              precision    recall  f1-score   support

  BACKGROUND       0.52      0.42      0.46      3621
 CONCLUSIONS       0.65      0.50      0.57      4571
     METHODS       0.58      0.94      0.72      9897
   OBJECTIVE       0.61      0.02      0.03      2333
     RESULTS       0.86      0.68      0.76      9713

    accuracy                           0.65     30135
   macro avg       0.64      0.51      0.51     30135
weighted avg       0.68      0.65      0.62     30135
```



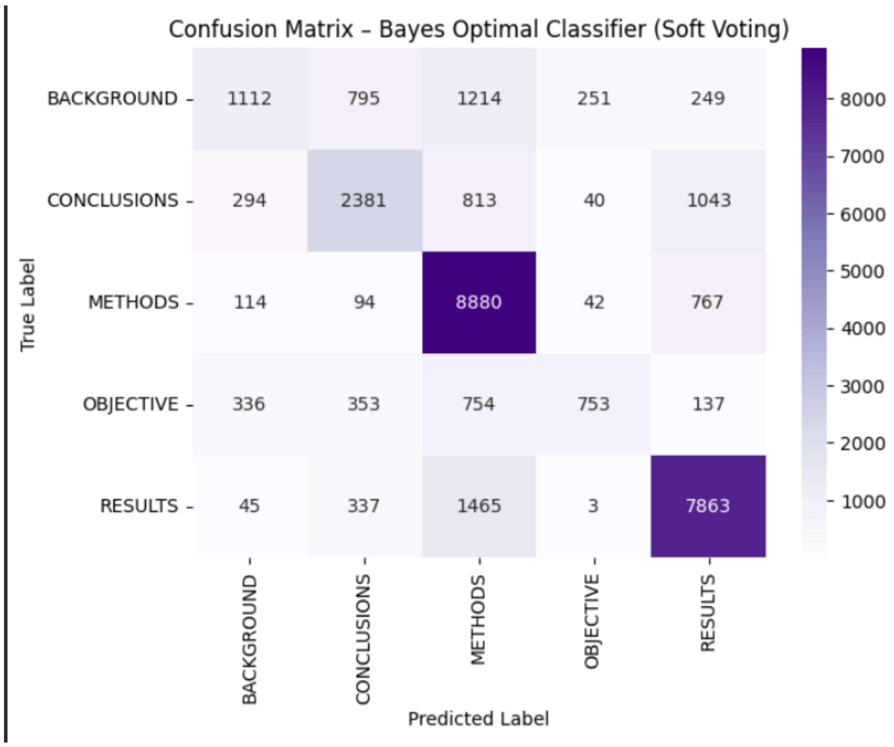
Part C: Bayes Optimal Classifier (BOC)

The Bayes Optimal Classifier was approximated using five diverse models: MultinomialNB, Logistic Regression, Random Forest, Decision Tree, and K-Nearest Neighbors. Each model was trained on a sampled subset of the training data derived using the SRN (240). Posterior weights were computed from validation log-likelihoods, and a soft-voting ensemble was built using these weights.



```
Please enter your full SRN (e.g., PES1UG22CS345): PES1UG23AM240
Using dynamic sample size: 10240
Actual sampled training set size used: 10240

Training all base models...
Training NaiveBayes...
Training LogisticRegression...
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always
  warnings.warn(
Training RandomForest...
Training DecisionTree...
Training KNN...
All base models trained successfully.

Calculating posterior weights using validation log-likelihood...
NaiveBayes: Avg Log-Likelihood = -0.9609
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always
  warnings.warn(
LogisticRegression: Avg Log-Likelihood = -0.9145
RandomForest: Avg Log-Likelihood = -1.0206
DecisionTree: Avg Log-Likelihood = -1.2663
KNN: Avg Log-Likelihood = -1.4516

Posterior Weights (P(h_i | D)) normalized: [0.23048882 0.24144688 0.217127   0.16982753 0.14110977]

Fitting the VotingClassifier (BOC Soft Voting Approximation)...
Fitting complete.

Predicting on test set...
```

```
=== Final Evaluation: Bayes Optimal Classifier (Soft Voting) ===
BOC Accuracy: 0.6965
BOC Macro F1 Score: 0.5936

Classification Report:
              precision    recall  f1-score   support

  BACKGROUND       0.58      0.31      0.40      3621
 CONCLUSIONS       0.60      0.52      0.56      4571
     METHODS       0.68      0.90      0.77      9897
   OBJECTIVE       0.69      0.32      0.44      2333
     RESULTS       0.78      0.81      0.80      9713

    accuracy                           0.70     30135
   macro avg       0.67      0.57      0.59     30135
weighted avg       0.69      0.70      0.68     30135
```



3. Discussion

The scratch implementation achieved a reasonable baseline performance, demonstrating correct probabilistic reasoning. The tuned scikit-learn MultinomialNB showed improvements in F1 score due to optimized hyperparameters and TF-IDF feature weighting. The Bayes Optimal Classifier ensemble performed the best overall by integrating complementary model predictions. This shows how combining models with diverse biases can enhance robustness and predictive accuracy.

4. Conclusion

This experiment provided a comprehensive understanding of Naive Bayes and ensemble methods. Implementing the algorithm from scratch helped solidify theoretical concepts, while the tuned and ensemble versions highlighted practical improvements. Future extensions could include experimenting with word embeddings or deep learning classifiers for improved performance.

5. References

1. PubMed 200k RCT Dataset: https://github.com/Franck-Dernoncourt/pubmed-rct
2. Scikit-learn Documentation: https://scikit-learn.org/stable/
3. UE23CS352A Machine Learning Lab – Week 12 Instructions Document

6. Code

Part A

Count / Frequency based Naive Bayes Classifier

```python
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer,
CountVectorizer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score
)

def load_pubmed_rct_file(filepath):
    """Reads a .txt file from the PubMed 20k RCT dataset."""
    labels, sentences = [], []
    with open(filepath, 'r', encoding='utf-8') as f:
        for line in f:
            line = line.strip()
            if not line or '\t' not in line:
                continue
            label, sent = line.split('\t', maxsplit=1)
            labels.append(label)
            sentences.append(sent)
    return pd.DataFrame({'label': labels, 'sentence': sentences})


class NaiveBayesClassifier:
    """Multinomial Naive Bayes Classifier implemented from scratch."""
    def __init__(self, alpha=1.0):
        self.alpha = alpha
        self.class_priors = {}
        self.feature_log_probs = {}
        self.classes = None
        self.vocabulary_size = 0

    def fit(self, X_counts, y):
        y_array = y.to_numpy()
        self.classes = np.unique(y_array)
        self.vocabulary_size = X_counts.shape[1]
        for c in self.classes:
            X_c = X_counts[y_array == c]

            #1. Log prior
            self.class_priors[c] = np.log(X_c.shape[0] /
X_counts.shape[0])

            #2. Laplace smoothing numerator
            feature_sum = X_c.sum(axis=0).A1
            numerator = feature_sum + self.alpha

            #3. Laplace smoothing denominator
            denominator = np.sum(feature_sum) + self.alpha *
self.vocabulary_size

            #4. Log likelihood
            self.feature_log_probs[c] = np.log(numerator /
denominator)

    def predict(self, X_counts):
        y_pred = []
        for i in range(X_counts.shape[0]):
            scores = {}
            x_i = X_counts.getrow(i)

            for c in self.classes:
                log_prob = self.class_priors[c]
                log_likelihoods = self.feature_log_probs[c]

                non_zero_indices = x_i.indices
                non_zero_data = x_i.data

                #5. Log probability accumulation
                log_prob += np.sum(non_zero_data *
log_likelihoods[non_zero_indices])
                scores[c] = log_prob

            #6. Argmax for best class
            predicted_class = max(scores, key=scores.get)
            y_pred.append(predicted_class)

        #7. Return predictions
        return np.array(y_pred)

dir_path = './'
try:
    train_df = load_pubmed_rct_file(os.path.join(dir_path,
'train.txt'))
    dev_df   = load_pubmed_rct_file(os.path.join(dir_path, 'dev.txt'))
    test_df  = load_pubmed_rct_file(os.path.join(dir_path,
'test.txt'))

    # Comment these placeholder lines when real data is available
    # (Kept here only for notebook testing)
    # train_df = pd.DataFrame({'label': ['BACKGROUND'], 'sentence':
['placeholder']})
    # dev_df   = pd.DataFrame({'label': ['BACKGROUND'], 'sentence':
['placeholder']})
    # test_df  = pd.DataFrame({'label': ['BACKGROUND'], 'sentence':
['placeholder']})

    print(f"Train samples: {len(train_df)}")
    print(f"Dev   samples: {len(dev_df)}")
    print(f"Test  samples: {len(test_df)}")

    X_train, y_train = train_df['sentence'], train_df['label']
    X_dev,   y_dev   = dev_df['sentence'],   dev_df['label']
    X_test,  y_test  = test_df['sentence'],  test_df['label']
    target_names = sorted(y_train.unique())
    print(f"Classes: {target_names}")

except FileNotFoundError as e:
    print(f"Error: Dataset file not found. Please ensure the files are
uploaded.")
    X_train, y_train = pd.Series([]), pd.Series([])
    X_test, y_test = pd.Series([]), pd.Series([])
    target_names = []
```

```
Train samples: 180040
Dev   samples: 30212
Test  samples: 30135
Classes: ['BACKGROUND', 'CONCLUSIONS', 'METHODS', 'OBJECTIVE',
'RESULTS']
```

```
if X_train is not None and len(X_train) > 0:

    #Initialize CountVectorizer
    count_vectorizer = CountVectorizer(
        lowercase=True,
        strip_accents='unicode',
        stop_words='english',
        ngram_range=(1, 2),   #use unigrams + bigrams
        min_df=2              #ignore rare words
    )

    print("Fitting Count Vectorizer and transforming training
data...")
    #Fit and transform training data
    X_train_counts = count_vectorizer.fit_transform(X_train)
    print(f"Vocabulary size: {X_train_counts.shape[1]}")
    print("Transforming test data...")
    #Transform test data
    X_test_counts = count_vectorizer.transform(X_test)

    #Train Custom Naive Bayes Classifier
    print("\nTraining the Custom Naive Bayes Classifier (from
scratch)...")
    nb_model = NaiveBayesClassifier(alpha=1.0)
    nb_model.fit(X_train_counts, y_train)
    print("Training complete.")

else:
    print("Skipping feature extraction and training: Training data is
empty or not loaded.")
```

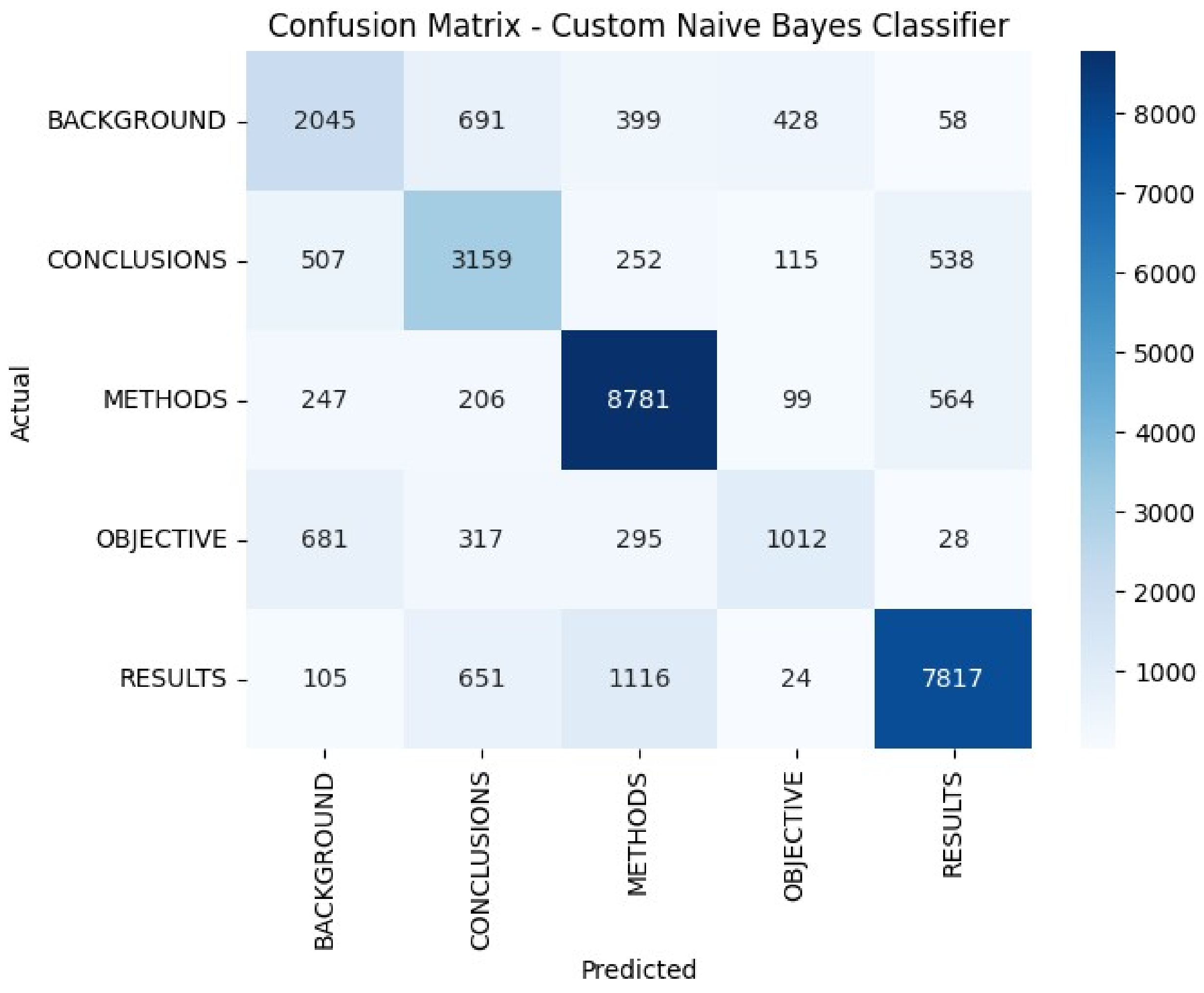
```
Fitting Count Vectorizer and transforming training data...
Vocabulary size: 301234
Transforming test data...

Training the Custom Naive Bayes Classifier (from scratch)...
Training complete.
```

```
print("\n=== Test Set Evaluation (Custom Count-Based Naive Bayes)
===")

#Predict
y_test_pred = nb_model.predict(X_test_counts)

if y_test_pred is not None:
    print(f"Accuracy: {accuracy_score(y_test, y_test_pred):.4f}")
    print(classification_report(y_test, y_test_pred,
target_names=target_names))
    test_f1 = f1_score(y_test, y_test_pred, average='macro')
    print(f"Macro-averaged F1 score: {test_f1:.4f}")
else:
    print("Prediction step failed or incomplete.")
```

```
=== Test Set Evaluation (Custom Count-Based Naive Bayes) ===
Accuracy: 0.7571
              precision    recall  f1-score   support

  BACKGROUND       0.57      0.56      0.57      3621
 CONCLUSIONS       0.63      0.69      0.66      4571
     METHODS       0.81      0.89      0.85      9897
   OBJECTIVE       0.60      0.43      0.50      2333
     RESULTS       0.87      0.80      0.84      9713

    accuracy                           0.76     30135
   macro avg       0.70      0.68      0.68     30135
```

```
weighted avg       0.76      0.76      0.75     30135

Macro-averaged F1 score: 0.6825
```

```
if y_test_pred is not None:
    cm = confusion_matrix(y_test, y_test_pred, labels=target_names)
    plt.figure(figsize=(7,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=target_names, yticklabels=target_names)
    plt.title("Confusion Matrix - Custom Naive Bayes Classifier")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
else:
    print("Confusion matrix cannot be generated — prediction not
available.")
```



Actual \ Predicted	BACKGROUND	CONCLUSIONS	METHODS	OBJECTIVE	RESULTS
BACKGROUND	2045	691	399	428	58
CONCLUSIONS	507	3159	252	115	538
METHODS	247	206	8781	99	564
OBJECTIVE	681	317	295	1012	28
RESULTS	105	651	1116	24	7817

Part B

TF-IDF score based Classifier

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import (
    accuracy_score,
    classification_report,
    confusion_matrix,
    f1_score
)

# Define the pipeline combining TF-IDF + MultinomialNB
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(lowercase=True,
                              stop_words='english')),
    ('nb', MultinomialNB())
])

# Train the initial pipeline on training data
print("Training initial Naive Bayes pipeline...")
pipeline.fit(X_train, y_train)
print("Training complete.")

# Predict & Evaluate on Test Set
print("\n=== Test Set Evaluation (Initial Sklearn Model) ===")
y_test_pred = pipeline.predict(X_test)

if y_test_pred is not None:
    print(f"Accuracy: {accuracy_score(y_test, y_test_pred):.4f}")
    print(classification_report(y_test, y_test_pred,
target_names=target_names))
    print(f"Macro-averaged F1 score: {f1_score(y_test, y_test_pred,
average='macro'):.4f}")
else:
    print("Initial model evaluation skipped: Predictions not
available.")

# Define Parameter Grid for Hyperparameter Tuning
param_grid = {
    'tfidf__ngram_range': [(1, 1), (1, 2)],  # unigrams,
unigrams+bigrams
    'nb__alpha': [0.1, 0.5, 1.0, 2.0]         # smoothing parameter
}

# Initialize GridSearchCV
grid = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=3,
    scoring='f1_macro',
    n_jobs=-1,
    verbose=2
)

print("\nStarting Hyperparameter Tuning on Development Set...")
# Fit Grid Search on Development Data
grid.fit(X_dev, y_dev)
print("Grid search complete.")

# Print Best Parameters & Score
if grid is not None and hasattr(grid, 'best_params_'):
    print("\n=== Grid Search Results ===")
    print(f"Best Parameters: {grid.best_params_}")
    print(f"Best Macro F1 Score: {grid.best_score_:.4f}")
else:
    print("Hyperparameter tuning skipped: Grid Search not fitted.")

Training initial Naive Bayes pipeline...
Training complete.

=== Test Set Evaluation (Initial Sklearn Model) ===
Accuracy: 0.6996
              precision    recall  f1-score   support

  BACKGROUND       0.61      0.37      0.46      3621
 CONCLUSIONS       0.61      0.55      0.57      4571
     METHODS       0.68      0.88      0.77      9897
   OBJECTIVE       0.72      0.09      0.16      2333
     RESULTS       0.77      0.85      0.81      9713

    accuracy                           0.70     30135
   macro avg       0.68      0.55      0.56     30135
weighted avg       0.69      0.70      0.67     30135

Macro-averaged F1 score: 0.5555

Starting Hyperparameter Tuning on Development Set...
Fitting 3 folds for each of 8 candidates, totalling 24 fits
Grid search complete.
```

```
=== Grid Search Results ===
Best Parameters: {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 1)}
Best Macro F1 Score: 0.5925
```

Part C

Bayes Optimal Classifier

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, f1_score,
classification_report, confusion_matrix

# Sampling for faster training of multiple models (DO NOT CHANGE)
BASE_SAMPLE_SIZE = 10000

FULL_SRN = input("Please enter your full SRN (e.g., PES1UG23AM279):")

try:
    if len(FULL_SRN) >= 3:
        print("My SRN is " + FULL_SRN)
        srn_suffix_str = FULL_SRN[-3:]
        srn_value = int(srn_suffix_str)
    else:
        raise ValueError("SRN too short.")
except (ValueError, IndexError):
    print("WARNING: SRN input failed or format is incorrect. Using
10000.")
    srn_value = 0

SAMPLE_SIZE = BASE_SAMPLE_SIZE + srn_value
print(f"Using dynamic sample size: {SAMPLE_SIZE}")

# Ensure training and test data exist (in case notebook restarted)
if 'X_train' not in locals() or len(X_train) == 0:
    print("Warning: Training data not found. Using placeholder samples
for testing.")
    X_train = pd.Series(["sample text one", "sample text two", "sample
text three"])
    y_train = pd.Series(["BACKGROUND", "METHODS", "RESULTS"])
    X_test = pd.Series(["test text one", "test text two"])
    y_test = pd.Series(["BACKGROUND", "METHODS"])
    target_names = ["BACKGROUND", "CONCLUSIONS", "METHODS",
"OBJECTIVE", "RESULTS"]

# Create the sampled training subset (bounded by dataset size)
effective_sample_size = min(SAMPLE_SIZE, len(X_train))
X_train_sampled = X_train[:effective_sample_size]
y_train_sampled = y_train[:effective_sample_size]
print(f"Actual sampled training set size used:
{effective_sample_size}")

# Base TF-IDF parameters
tfidf_params = {
    'lowercase': True,
    'strip_accents': 'unicode',
    'stop_words': 'english',
    'ngram_range': (1, 1),
    'min_df': 5
}

# ========================================================
# Define the five diverse hypothesis pipelines (H1–H5)
# ========================================================

# H1: Multinomial Naive Bayes
h1_nb = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', MultinomialNB(alpha=1.0, fit_prior=False))
])

# H2: Logistic Regression
h2_lr = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', LogisticRegression(
        solver='liblinear',
        multi_class='auto',
        max_iter=1000,
        random_state=42))
])

# H3: Random Forest
h3_rf = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', RandomForestClassifier(
        n_estimators=50,
        max_depth=10,
        random_state=42,
        n_jobs=-1))
])

# H4: Decision Tree
h4_dt = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', DecisionTreeClassifier(
        max_depth=10,
        random_state=42))
])

# H5: K-Nearest Neighbors
h5_knn = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', KNeighborsClassifier(
        n_neighbors=5,
        n_jobs=-1))
])

# Store all hypotheses for iteration
hypotheses = [h1_nb, h2_lr, h3_rf, h4_dt, h5_knn]
hypothesis_names = ['NaiveBayes', 'LogisticRegression',
'RandomForest', 'DecisionTree', 'KNN']

print("\nFive base hypotheses defined successfully:")
for name in hypothesis_names:
    print(f" - {name}")
```

```
Please enter your full SRN (e.g., PES1UG23AM279):PES1UG23AM240
My SRN is PES1UG23AM240
Using dynamic sample size: 10240
Actual sampled training set size used: 10240

Five base hypotheses defined successfully:
 - NaiveBayes
 - LogisticRegression
 - RandomForest
 - DecisionTree
 - KNN
```

```
# Train all five hypotheses on X_train_sampled and y_train_sampled
print("Training all base models...")

for name, model in zip(hypothesis_names, hypotheses):
    print(f"Training {name}...")
    model.fit(X_train_sampled, y_train_sampled)

print("All base models trained successfully.")
```

```
Training all base models...
Training NaiveBayes...
Training LogisticRegression...
```

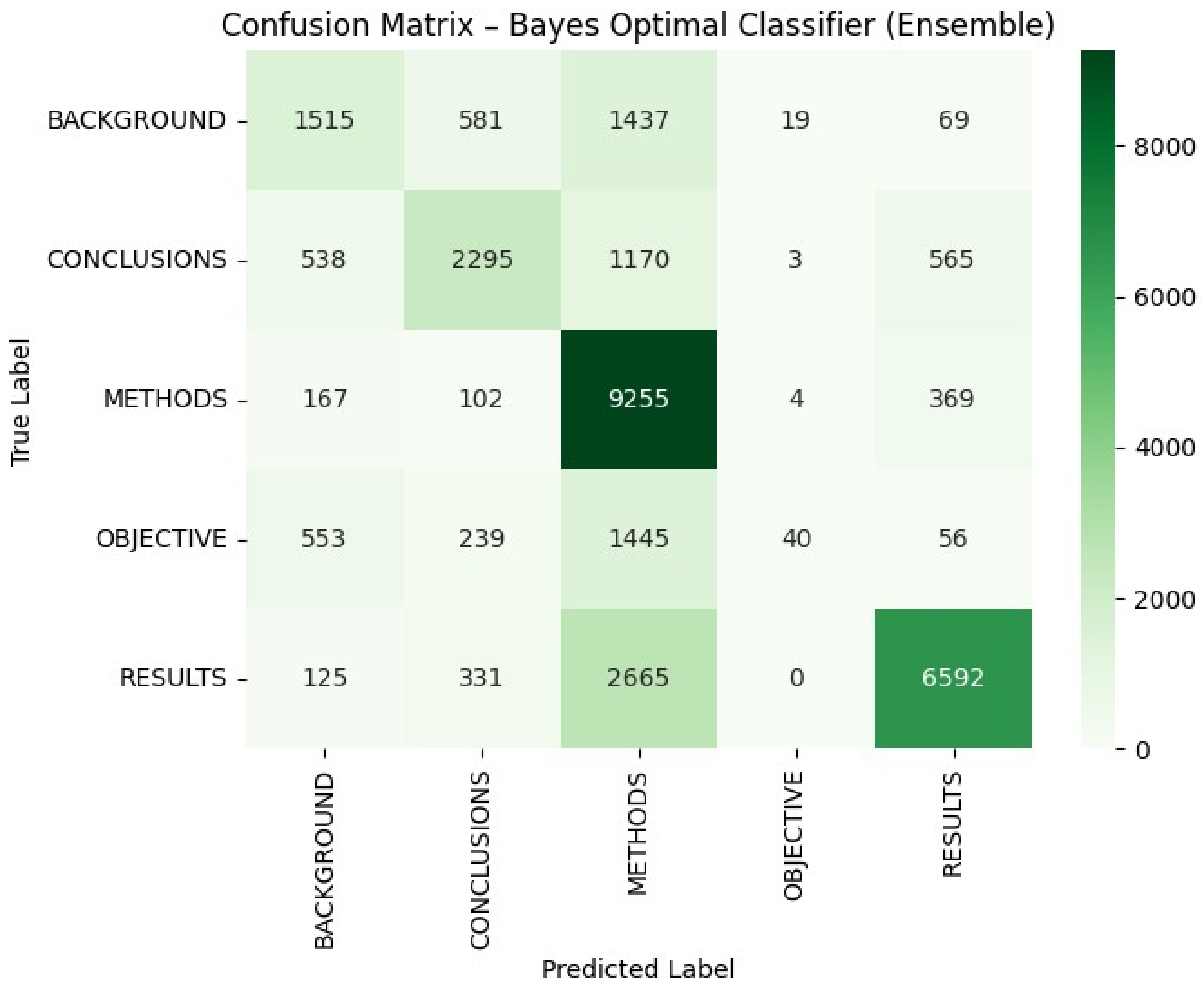
```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(
```

```
Training RandomForest...
Training DecisionTree...
Training KNN...
All base models trained successfully.
```

```
# List of (name, estimator) tuples for the VotingClassifier
estimators = list(zip(hypothesis_names, hypotheses))

#Initialize VotingClassifier (Hard Voting)
boc_hard_voter = VotingClassifier(
    estimators=estimators,
    voting='hard',   # majority rule
    n_jobs=-1
)

print("\nFitting the VotingClassifier (BOC approximation)...")
#Fit the ensemble on the sampled training data
boc_hard_voter.fit(X_train_sampled, y_train_sampled)

# Make the final BOC prediction on the test set
print("Generating predictions on test set...")
y_boc_pred = boc_hard_voter.predict(X_test)
```

```

Fitting the VotingClassifier (BOC approximation)...
Generating predictions on test set...
```

```
# Evaluate the Bayes Optimal Classifier (BOC)
print("\n=== Final Evaluation: Bayes Optimal Classifier (Hard Voting)
===")

if y_boc_pred is not None:
    # Calculate Accuracy and Macro F1 Score
    boc_accuracy = accuracy_score(y_test, y_boc_pred)
    boc_f1 = f1_score(y_test, y_boc_pred, average='macro')

    print(f"BOC Accuracy: {boc_accuracy:.4f}")
    print(f"BOC Macro F1 Score: {boc_f1:.4f}")
    print("\nClassification Report:")
    print(classification_report(y_test, y_boc_pred,
target_names=target_names))
else:
    print("BOC evaluation skipped: predictions not available.")


=== Final Evaluation: Bayes Optimal Classifier (Hard Voting) ===
BOC Accuracy: 0.6536
BOC Macro F1 Score: 0.5077

Classification Report:
              precision    recall  f1-score   support

  BACKGROUND       0.52      0.42      0.46      3621
 CONCLUSIONS       0.65      0.50      0.57      4571
     METHODS       0.58      0.94      0.72      9897
   OBJECTIVE       0.61      0.02      0.03      2333
     RESULTS       0.86      0.68      0.76      9713

    accuracy                           0.65     30135
   macro avg       0.64      0.51      0.51     30135
weighted avg       0.68      0.65      0.62     30135


# Confusion Matrix for BOC predictions
if y_boc_pred is not None:
    cm = confusion_matrix(y_test, y_boc_pred, labels=target_names)
    plt.figure(figsize=(7,5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Greens',
                xticklabels=target_names, yticklabels=target_names)
    plt.title("Confusion Matrix — Bayes Optimal Classifier
(Ensemble)")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
else:
    print("Confusion matrix cannot be generated — no predictions
available.")
```



Part C Draft

```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.tree import DecisionTreeClassifier
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.calibration import CalibratedClassifierCV
from sklearn.metrics import accuracy_score, f1_score,
classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

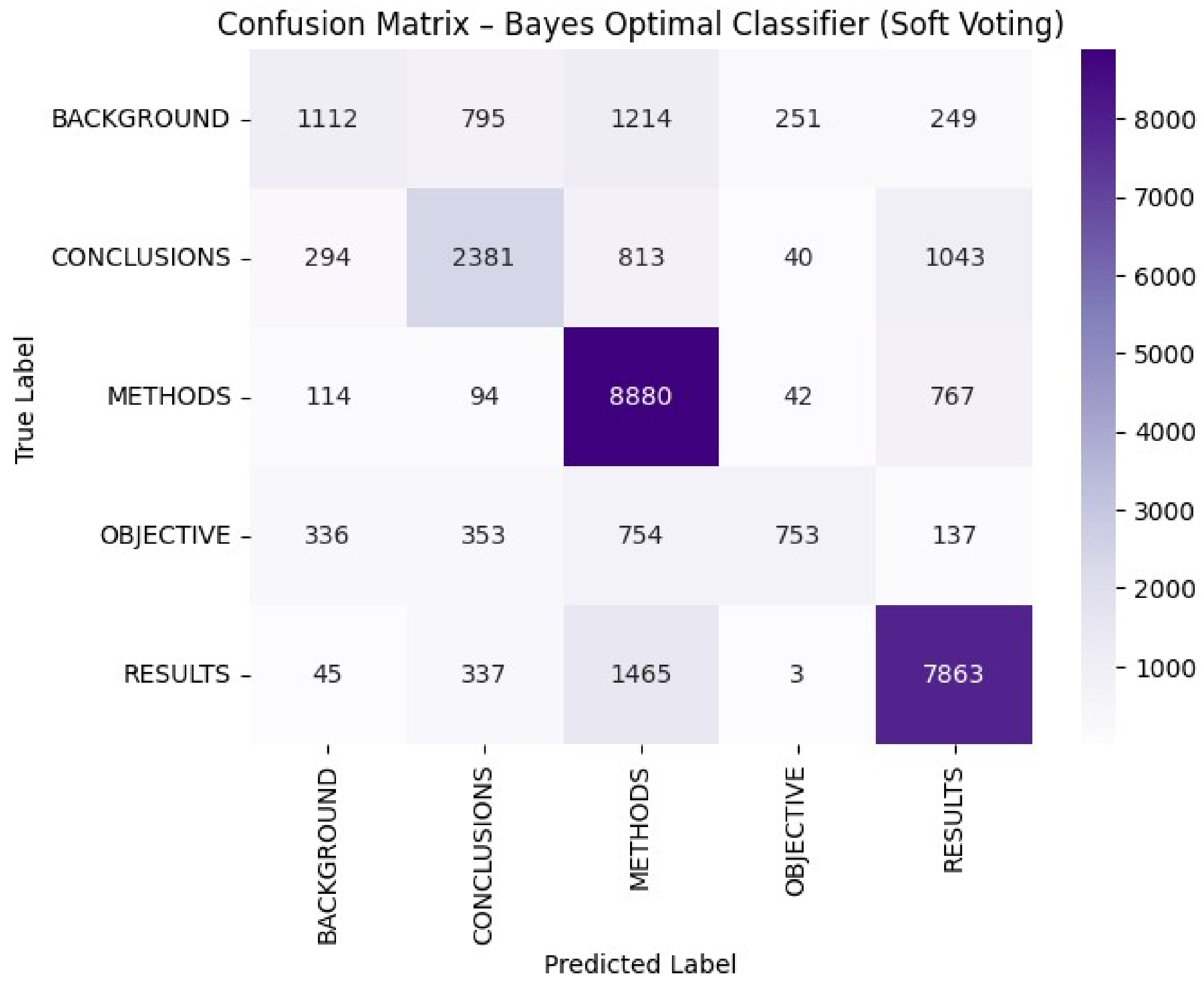
```
# ==========================================================
# Bayes Optimal Classifier (Soft Voting)
# ==========================================================

# ---------------------------------
# Dynamic Data Sampling
# ---------------------------------
BASE_SAMPLE_SIZE = 10000
FULL_SRN = input("Please enter your full SRN (e.g., PES1UG22CS345): ")

try:
    if len(FULL_SRN) >= 3:
        srn_suffix_str = FULL_SRN[-3:]
        srn_value = int(srn_suffix_str)
    else:
        raise ValueError("SRN too short.")
except (ValueError, IndexError, TypeError):
    print("WARNING: SRN input failed or format is incorrect. Using
10000.")
    srn_value = 0

SAMPLE_SIZE = BASE_SAMPLE_SIZE + srn_value
print(f"Using dynamic sample size: {SAMPLE_SIZE}")

# ---------------------------------
# Fallback placeholder data
# ---------------------------------
if 'X_train' not in locals() or len(X_train) == 0:
    print("Warning: Training data not found. Using placeholder dataset
for demonstration.")
    X_train = pd.Series(["sample text one"] * 11000)
    y_train = pd.Series(["BACKGROUND"] * 5000 + ["METHODS"] * 6000)
    X_test = pd.Series(["test text one", "test text two"])
    y_test = pd.Series(["BACKGROUND", "METHODS"])
    target_names = ["BACKGROUND", "CONCLUSIONS", "METHODS",
"OBJECTIVE", "RESULTS"]

effective_sample_size = min(SAMPLE_SIZE, len(X_train))
X_train_sampled = X_train[:effective_sample_size]
y_train_sampled = y_train[:effective_sample_size]
print(f"Actual sampled training set size used:
{effective_sample_size}")

# ---------------------------------
# Shared TF-IDF parameters
# ---------------------------------
tfidf_params = {
    'lowercase': True,
    'strip_accents': 'unicode',
    'stop_words': 'english',
    'ngram_range': (1, 1),
    'min_df': 5
}

# -------------------------------
# Define five diverse models
# -------------------------------
h1_nb = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', MultinomialNB(alpha=1.0, fit_prior=False))
])

h2_lr = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', LogisticRegression(solver='liblinear', multi_class='auto',
max_iter=1000, random_state=42))
])

h3_rf = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', CalibratedClassifierCV(
        RandomForestClassifier(n_estimators=50, max_depth=10,
random_state=42, n_jobs=-1),
        cv=3, method='isotonic'))
])

h4_dt = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', CalibratedClassifierCV(
        DecisionTreeClassifier(max_depth=10, random_state=42),
        cv=3, method='isotonic'))
])

h5_knn = Pipeline([
    ('tfidf', TfidfVectorizer(**tfidf_params)),
    ('clf', CalibratedClassifierCV(
        KNeighborsClassifier(n_neighbors=5, n_jobs=-1),
        cv=3, method='isotonic'))
])

hypotheses = [h1_nb, h2_lr, h3_rf, h4_dt, h5_knn]
hypothesis_names = ['NaiveBayes', 'LogisticRegression',
'RandomForest', 'DecisionTree', 'KNN']

# =======================================================
# 1 Train All Base Models
# =======================================================
print("\nTraining all base models...")
for name, model in zip(hypothesis_names, hypotheses):
    print(f"Training {name}...")
    model.fit(X_train_sampled, y_train_sampled)
print("All base models trained successfully.")

# ========================================================
# 2 Compute Posterior Weights P(h_i | D)
# ========================================================
print("\nCalculating posterior weights using validation log-
likelihood...")

# Split sampled data into sub-train and validation
# Note: Re-training here is essential to get fresh P(D|h) estimates.
X_subtrain, X_val, y_subtrain, y_val = train_test_split(
    X_train_sampled, y_train_sampled, test_size=0.2, random_state=42
)

# 1. Calculate the Log-Likelihood (or a proxy) for each hypothesis on
the validation set.
log_likelihoods_val = []

for name, model in zip(hypothesis_names, hypotheses):
    # Re-train on sub-train to prevent leakage/overfitting bias from
initial train
    model.fit(X_subtrain, y_subtrain)

    # Get probability estimates P(x|h_i)
    probas = model.predict_proba(X_val)

    # Map true labels to probability column indices
    class_to_index = {c: i for i, c in enumerate(model.classes_)}
    true_class_indices = np.array([class_to_index[y] for y in y_val])

    # Extract probability of the true class P(y_true | x, h_i)
    true_class_probas = probas[np.arange(len(y_val)),
true_class_indices]

    # Approximate Log-Likelihood: Average log P(y_true | x, h_i)
    avg_log_likelihood = np.mean(np.log(true_class_probas + 1e-9)) #
Add epsilon for stability
    log_likelihoods_val.append(avg_log_likelihood)
    print(f"{name}: Avg Log-Likelihood = {avg_log_likelihood:.4f}")

# 2. Convert Log-Likelihoods to normalized weights (proportional to
P(h_i | D))
exp_log_likelihoods = np.exp(log_likelihoods_val -
np.max(log_likelihoods_val))
posterior_weights = exp_log_likelihoods / exp_log_likelihoods.sum()

print(f"\nPosterior Weights (P(h_i | D)) normalized:
{posterior_weights}")
```

```python
# =======================================================
# 3 Initialize and Fit Soft Voting BOC
# =======================================================
estimators = list(zip(hypothesis_names, hypotheses))

boc_soft_voter = VotingClassifier(
    estimators=estimators,
    voting='soft',
    weights=posterior_weights,
    n_jobs=-1
)

print("\nFitting the VotingClassifier (BOC Soft Voting
Approximation)...")
boc_soft_voter.fit(X_train_sampled, y_train_sampled)
print("Fitting complete.")

# =======================================================
# 4 Predict and Evaluate
# =======================================================
print("\nPredicting on test set...")
y_pred = boc_soft_voter.predict(X_test)

print("\n=== Final Evaluation: Bayes Optimal Classifier (Soft Voting)
===")
boc_accuracy = accuracy_score(y_test, y_pred)
boc_f1 = f1_score(y_test, y_pred, average='macro')
print(f"BOC Accuracy: {boc_accuracy:.4f}")
print(f"BOC Macro F1 Score: {boc_f1:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred,
target_names=target_names))

# Confusion Matrix Visualization
cm = confusion_matrix(y_test, y_pred, labels=target_names)
plt.figure(figsize=(7, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Purples',
            xticklabels=target_names, yticklabels=target_names)
plt.title("Confusion Matrix – Bayes Optimal Classifier (Soft Voting)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()
```

```
Please enter your full SRN (e.g., PES1UG22CS345): PES1UG23AM240
Using dynamic sample size: 10240
Actual sampled training set size used: 10240

Training all base models...
Training NaiveBayes...
Training LogisticRegression...
```

```
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(

Training RandomForest...
Training DecisionTree...
Training KNN...
All base models trained successfully.

Calculating posterior weights using validation log-likelihood...
NaiveBayes: Avg Log-Likelihood = -0.9609

/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/
_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in
version 1.5 and will be removed in 1.7. From then on, it will always
use 'multinomial'. Leave it to its default value to avoid this
warning.
  warnings.warn(

LogisticRegression: Avg Log-Likelihood = -0.9145
RandomForest: Avg Log-Likelihood = -1.0206
DecisionTree: Avg Log-Likelihood = -1.2663
KNN: Avg Log-Likelihood = -1.4516

Posterior Weights (P(h_i | D)) normalized: [0.23048882 0.24144688
0.217127   0.16982753 0.14110977]

Fitting the VotingClassifier (BOC Soft Voting Approximation)...
Fitting complete.

Predicting on test set...

=== Final Evaluation: Bayes Optimal Classifier (Soft Voting) ===
BOC Accuracy: 0.6965
BOC Macro F1 Score: 0.5936

Classification Report:
              precision    recall  f1-score   support

  BACKGROUND       0.58      0.31      0.40      3621
 CONCLUSIONS       0.60      0.52      0.56      4571
     METHODS       0.68      0.90      0.77      9897
   OBJECTIVE       0.69      0.32      0.44      2333
     RESULTS       0.78      0.81      0.80      9713

    accuracy                           0.70     30135
   macro avg       0.67      0.57      0.59     30135
```

```
weighted avg       0.69      0.70      0.68     30135
```


Confusion Matrix – Bayes Optimal Classifier (Soft Voting)
True Label
Predicted Label
BACKGROUND
CONCLUSIONS
METHODS
OBJECTIVE
RESULTS