Project Title:

Naïve Bayes Classifier

Name:

Maya Nithyanand Bhagath

SRN:

PES2UG23CS331

Course:

Machine Learning

Date:

30/10/2025

Introduction

Purpose of the Lab

The primary purpose of this lab was to implement and evaluate a probabilistic text classification system. The goal was to accurately classify individual sentences from biomedical abstracts into one of five predefined sections: BACKGROUND, CONCLUSIONS, METHODS, OBJECTIVE, or RESULTS.

Tasks Performed

Part A: We implemented a Multinomial Naive Bayes (MNB) classifier from scratch using Python. This model was trained on raw word counts (CountVectorizer) to understand its core mathematical and probabilistic components.

Part B: We used the TfidfVectorizer for feature extraction and MultinomialNB as the classifier from the scikit-learn library. We then performed hyperparameter tuning using GridSearchCV on the development dataset to find the optimal model parameters.

Part C: We approximated the theoretical Bayes Optimal Classifier (BOC) by training an ensemble of five models (Naive Bayes, Logistic Regression, Random Forest, Decision Tree, and KNN) and combining their predictions using a VotingClassifier weighted by their posterior probabilities.

Methodology

Multinomial Naïve Bayes:

- The MNB classifier was built by calculating two key components based on the training data.
- The Log Prior for each class represents the overall probability of a class appearing in the dataset.
- The Log Likelihood that we calculated represents how likely a specific word is to appear, given a specific class.
- The model combines these probabilities for each class, and the class with the highest probability is chosen as the prediction.

Tuned Sklearn MNB Pipeline:

- This used scikit-learn's Pipeline tool to streamline the process and find an optimal model.
- First, the TfidfVectorizer was used for feature extraction. This tool converts raw text into a matrix of features.
- Then we used GridSearchCV to automatically find the best-performing model. It used 3-fold cross-validation on dev.txt to find the exact combination of parameters that resulted in the highest F1-score.

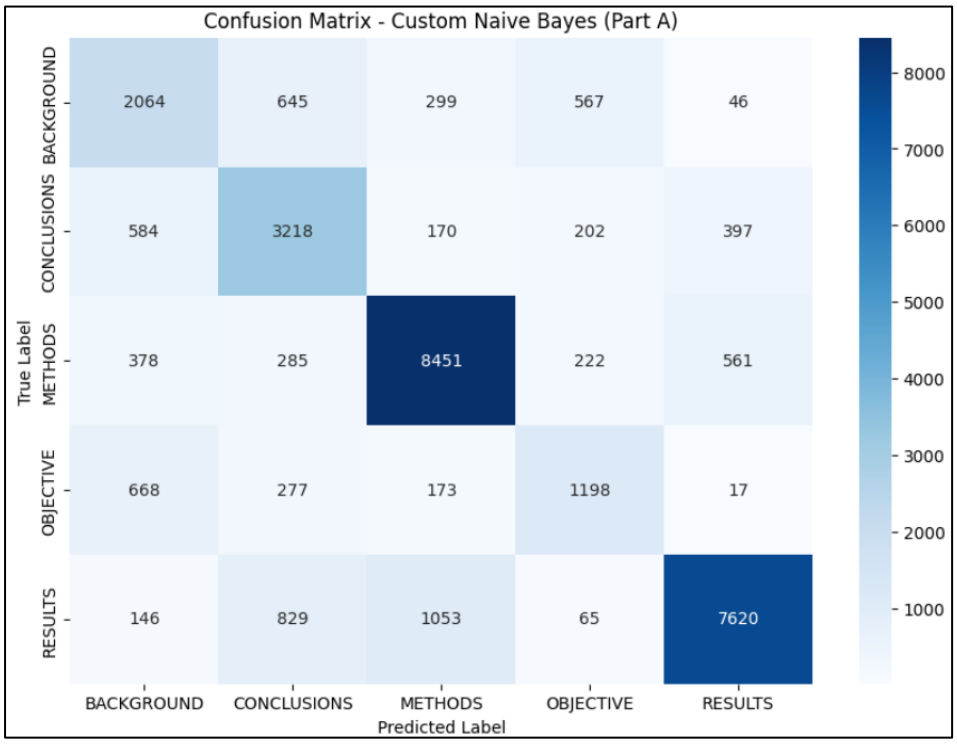
Bayes Optimal Classifier (BOC):

- BOC is a theoretical "perfect" classifier that achieves the lowest possible error. Since this model is not practical to build, we approximated it using a "soft-voting" ensemble.
- First, we defined a diverse set of five hypotheses- Multinomial Naive Bayes, Logistic Regression, Random Forest, Decision Tree, and K-Nearest Neighbors.
- Then we calculated Posterior Weights to determine how much to "trust" each model. To do this, the training data was split into sub-training set and validation set. We then calculated the loss scores, assigning a lower weight to models that had a lower log-loss score.
- Finally, we implemented the ensemble using a VotingClassifier set to voting='soft'. This voter was trained on the full sampled training dataset.
- Each of the models' predictions was then multiplied with the posterior weight to get a more robust prediction.

Results and Analysis

Part A:

```
=== Test Set Evaluation (Custom Count-Based Naive Bayes) ===
Accuracy: 0.7483
              precision    recall  f1-score   support

  BACKGROUND       0.54      0.57      0.55      3621
 CONCLUSIONS       0.61      0.70      0.66      4571
     METHODS       0.83      0.85      0.84      9897
   OBJECTIVE       0.53      0.51      0.52      2333
     RESULTS       0.88      0.78      0.83      9713

    accuracy                           0.75     30135
   macro avg       0.68      0.69      0.68     30135
weighted avg       0.76      0.75      0.75     30135

Macro-averaged F1 score: 0.6809
```



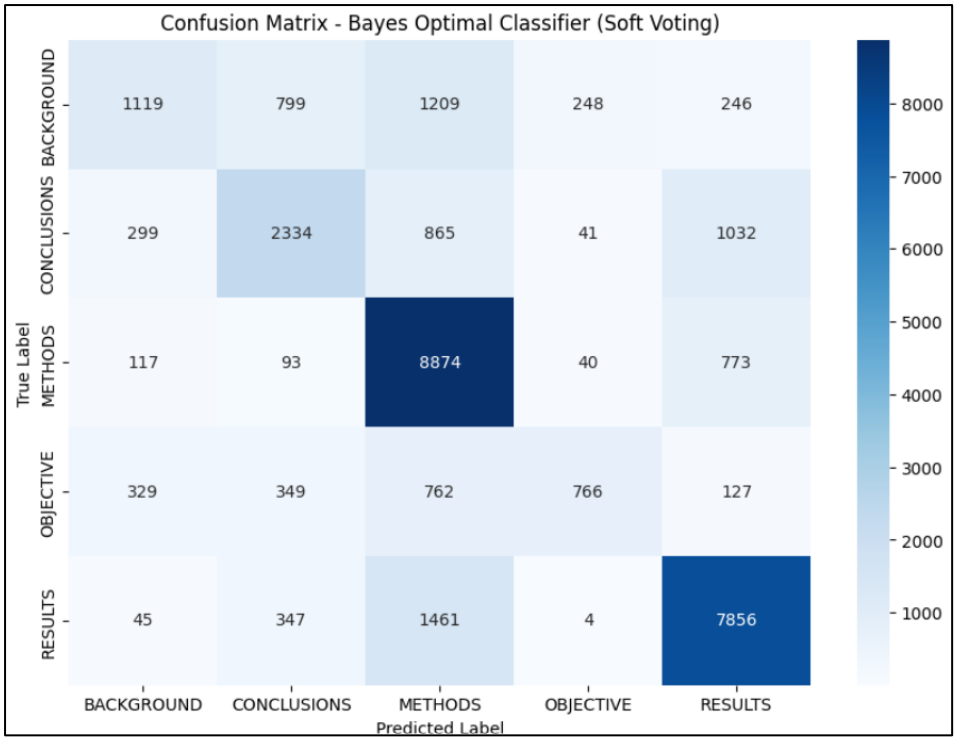
Part B:

```
=== Test Set Evaluation (Initial Sklearn Model) ===
Accuracy: 0.7266
              precision    recall  f1-score   support

  BACKGROUND       0.64      0.43      0.51      3621
 CONCLUSIONS       0.62      0.61      0.62      4571
     METHODS       0.72      0.90      0.80      9897
   OBJECTIVE       0.73      0.10      0.18      2333
     RESULTS       0.80      0.87      0.83      9713

    accuracy                           0.73     30135
   macro avg       0.70      0.58      0.59     30135
weighted avg       0.72      0.73      0.70     30135

Macro-averaged F1 score: 0.5877

Starting Hyperparameter Tuning on Development Set...
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Grid search complete.

Best Parameters found: {'nb__alpha': 0.1, 'tfidf__min_df': 3, 'tfidf__ngram_range': (1, 2)}
Best cross-validation (f1_macro) score on Dev set: 0.6998
```

Part C:

```
Please enter your full SRN (e.g., PES1UG22CS345): PES2UG23CS331
Using dynamic sample size: 10331
Actual sampled training set size used: 10331

Training all base models on the full sampled data...
Training NaiveBayes on 10331 samples...
Training LogisticRegression on 10331 samples...
/usr/local/lib/python3.12/dist-packages/sklearn/linear_model/_logistic.py:1247: FutureWarning: 'multi_class' was deprecated in version 1.5 a
  warnings.warn(
Training RandomForest on 10331 samples...
Training DecisionTree on 10331 samples...
Training KNN on 10331 samples...
All base models trained.
```

```
=== Final Evaluation: Bayes Optimal Classifier (Soft Voting) ===
Accuracy: 0.6952
Macro F1 Score: 0.5932

Classification Report (BOC):
              precision    recall  f1-score   support

  BACKGROUND       0.59      0.31      0.40      3621
 CONCLUSIONS       0.60      0.51      0.55      4571
     METHODS       0.67      0.90      0.77      9897
   OBJECTIVE       0.70      0.33      0.45      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
```



Comparison of Performance

The F1 scores of the 3 approaches were as follows:

- Part A (Custom MNB): 0.6809
- Part B (Tuned Sklearn): 0.5877
- Part C (BOC Ensemble): 0.5932

The Tuned Sklearn Pipleine performed the best, and the BOC ensemble performed the worst.

- Part A used raw word counts, where common words (like "and", "in", "of") are treated as highly important. Part B down-weights these common words and gives more importance to words that are rare but highly informative for a specific class. This leads to a much cleaner and more predictive set of features.
- Part A used non-optimized parameters, while Part B used GridSearchCV to find the optimal values of parameters to maximise model performance.
- Since Part C was only trained on a small subset of the dataset, it could not learn the complex patterns that Part B was able to learn from the full dataset. Therefore, it performed worse than Part C.