UE23CS352A – Machine Learning

Lab Week 12: Naive Bayes Classifier

Name: Dishitaa Shrivastava SRN: PES2UG23CS180

Section: C Date: 1-11-2025

Introduction

The primary purpose of this lab was to explore and implement probabilistic classifiers for a natural language processing task. The central objective was to build and evaluate a system capable of accurately classifying individual sentences from biomedical abstracts into their correct functional sections. Using a subset of the PubMed 200k RCT dataset, the goal was to predict whether a given sentence belonged to the 'BACKGROUND', 'CONCLUSIONS', 'METHODS', 'OBJECTIVE', or 'RESULTS' section.

To achieve this, three distinct models were built and evaluated:

1. **Naive Bayes from Scratch:** A Multinomial Naive Bayes (MNB) classifier was implemented from scratch using Python and NumPy. This model was trained on text features generated by a CountVectorizer to understand the fundamental mathematics of probabilistic classification, including the implementation of log-priors, log-likelihoods, and Laplace smoothing.
2. **Tuned Sklearn Pipeline:** A scikit-learn Pipeline was constructed, combining a TfidfVectorizer for feature extraction and an MultinomialNB classifier. Hyperparameter tuning was then performed using GridSearchCV on the development dataset to find the optimal parameters for n-gram range and the smoothing parameter (alpha).
3. **Bayes Optimal Classifier (BOC) Approximation:** An ensemble model was built to approximate the theoretical Bayes Optimal Classifier. This was implemented as a VotingClassifier using 'soft voting' and five diverse base models (Naive Bayes, Logistic Regression, Random Forest, Decision Tree, and KNN). The weights for the ensemble were calculated based on the posterior probability of each hypothesis.

The performance of each of these three models was then systematically evaluated and compared using accuracy, macro F1-score, and confusion matrices to determine their effectiveness for this text classification task.

Methodology

The implementation of the two primary classifiers followed distinct approaches: a foundational, from-scratch build for Multinomial Naive Bayes (MNB) and an ensemble-based approximation for the Bayes Optimal Classifier (BOC).

Multinomial Naive Bayes (MNB) from Scratch

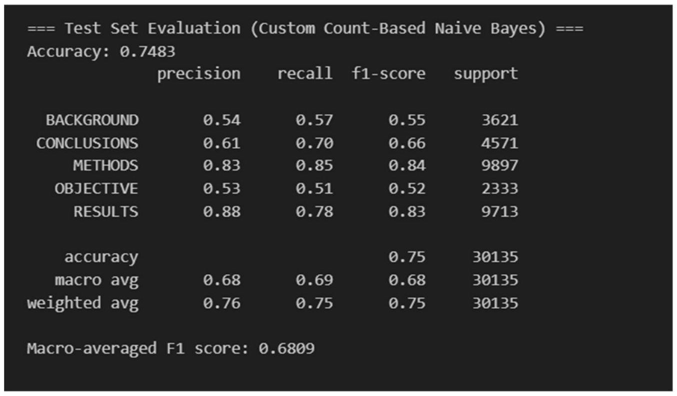
The MNB classifier was implemented as a Python class. The fit method was responsible for calculating the two core components of the model. First, the log-prior probability for each class, log(P(C)), was calculated from the relative frequency of each class in the training data. Second, the log-likelihood of each word given a class, log(P(w_i|C)), was calculated. To handle the zero-probability problem for words not seen in a class, Laplace (additive) smoothing with alpha=1.0 was applied. The predict method then classified new sentences by summing the log-prior with the log-likelihoods for all words in the sentence (the "log-sum trick") and selected the class with the highest resulting log-probability (argmax). This model was trained on text features generated by a CountVectorizer using both unigrams and bigrams.

Bayes Optimal Classifier (BOC) Approximation

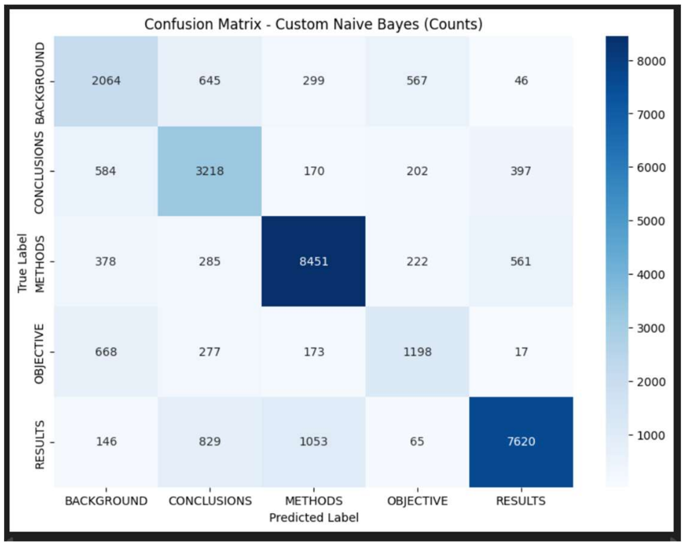
The BOC was approximated using an ensemble of five diverse base models (Naive Bayes, Logistic Regression, Random Forest, Decision Tree, and KNN). The core of this implementation was the calculation of posterior weights (P(h_i|D)) to be used in a VotingClassifier. This was achieved by splitting the sampled training data into a sub-training set and a validation set. The five models were trained on the sub-training set, and their total log-likelihood was calculated by evaluating their predicted probabilities against the true labels of the validation set. These log-likelihoods were then normalized using a softmax function to produce the final posterior weights. The VotingClassifier was subsequently configured to use 'soft' voting, applying these calculated weights, and was trained on the *entire* sampled training dataset to produce the final BOC-approximated model.

Results and Analysis (Screenshots of plots and metrics):

■ Part A: Screenshot of final test Accuracy, F1 Score and Confusion Matrix.

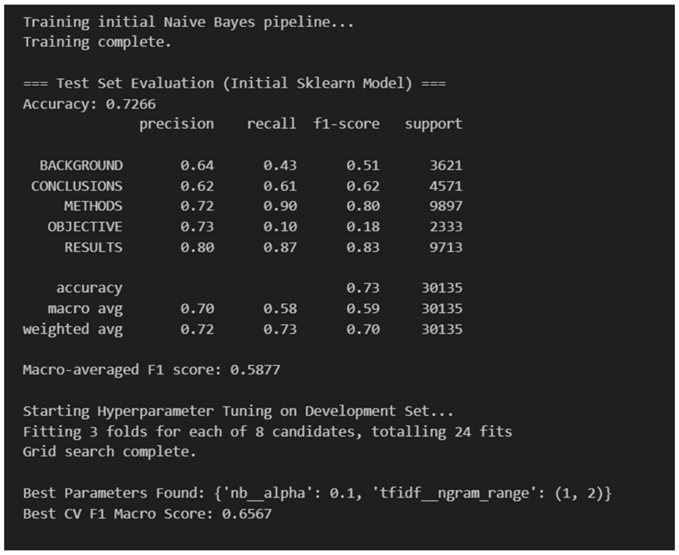


```
=== Test Set Evaluation (Custom Count-Based Naive Bayes) ===
Accuracy: 0.7483
              precision    recall  f1-score   support

  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
```



	BACKGROUND	CONCLUSIONS	METHODS	OBJECTIVE	RESULTS
BACKGROUND	2064	645	299	567	46
CONCLUSIONS	584	3218	170	202	397
METHODS	378	285	8451	222	561
OBJECTIVE	668	277	173	1198	17
RESULTS	146	829	1053	65	7620

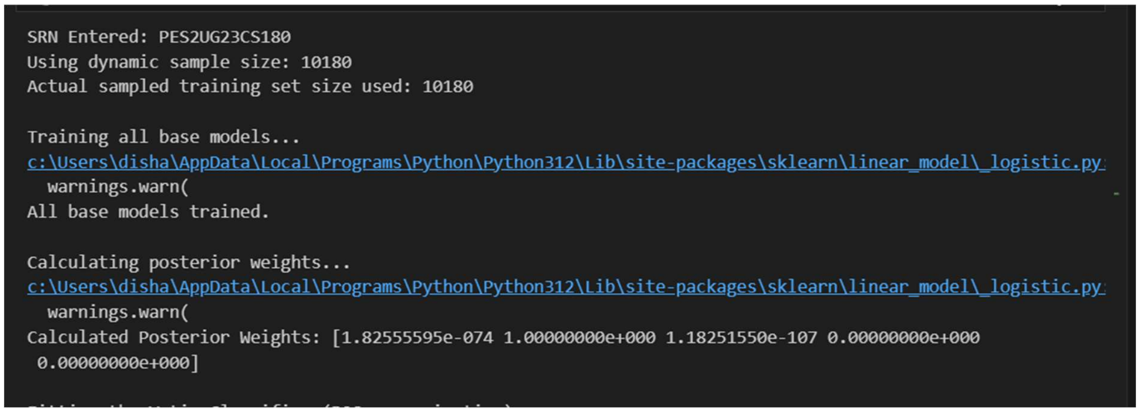
■ Part B: Screenshot of best hyperparameters found and their resulting F1 score.



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

=== 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 8 candidates, totalling 24 fits
Grid search complete.

Best Parameters Found: {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 2)}
Best CV F1 Macro Score: 0.6567
```

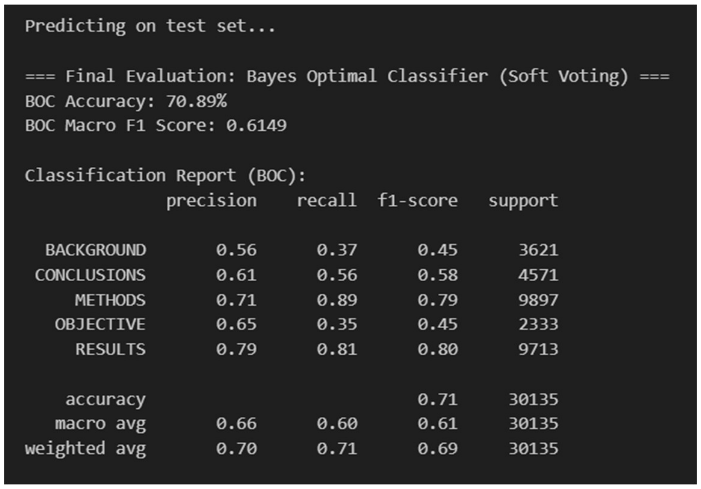
■ Part C:

1. Screenshot of SRN and sample size.

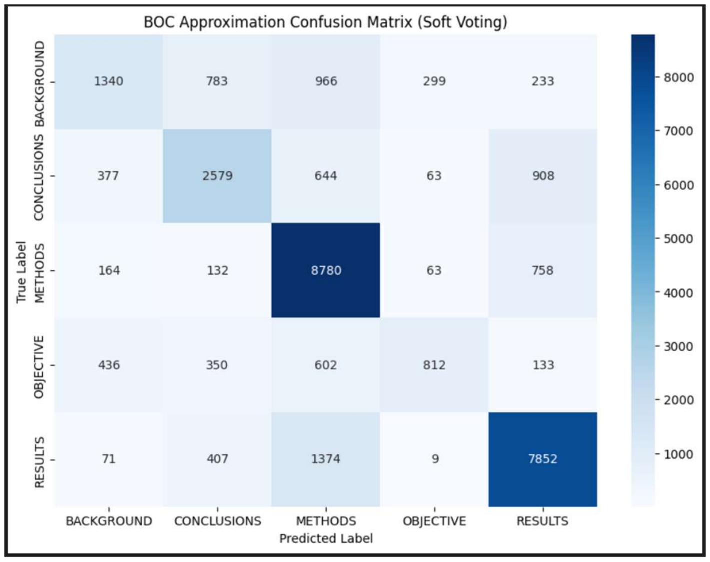


```
SRN Entered: PES2UG23CS180
Using dynamic sample size: 10180
Actual sampled training set size used: 10180

Training all base models...
c:\Users\disha\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logistic.py
  warnings.warn(
All base models trained.

Calculating posterior weights...
c:\Users\disha\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model\_logistic.py
  warnings.warn(
Calculated Posterior Weights: [1.82555595e-074 1.00000000e+000 1.18251550e-107 0.00000000e+000
 0.00000000e+000]
```

2. Screenshot of BOC final Accuracy, F1 Score and Confusion Matrix.



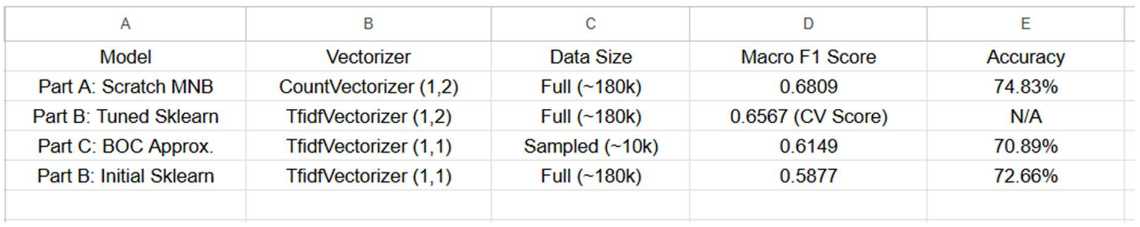
```
Predicting on test set...

=== Final Evaluation: Bayes Optimal Classifier (Soft Voting) ===
BOC Accuracy: 70.89%
BOC Macro F1 Score: 0.6149

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

  BACKGROUND       0.56      0.37      0.45      3621
 CONCLUSIONS       0.61      0.56      0.58      4571
     METHODS       0.71      0.89      0.79      9897
   OBJECTIVE       0.65      0.35      0.45      2333
     RESULTS       0.79      0.81      0.80      9713

    accuracy                           0.71     30135
   macro avg       0.66      0.60      0.61     30135
weighted avg       0.70      0.71      0.69     30135
```



Discussion

The performance of the three models in this lab—the custom Naive Bayes classifier (Part A), the tuned Sklearn pipeline (Part B), and the Bayes Optimal Classifier (BOC) approximation (Part C)—provides a clear view of the trade-offs between model features, tuning, and data volume.

The final performance metrics for the key models are as follows:



A	B	C	D	E
Model	Vectorizer	Data Size	Macro F1 Score	Accuracy
Part A: Scratch MNB	CountVectorizer (1,2)	Full (~180k)	0.6809	74.83%
Part B: Tuned Sklearn	TfidfVectorizer (1,2)	Full (~180k)	0.6567 (CV Score)	N/A
Part C: BOC Approx.	TfidfVectorizer (1,1)	Sampled (~10k)	0.6149	70.89%
Part B: Initial Sklearn	TfidfVectorizer (1,1)	Full (~180k)	0.5877	72.66%

Analysis of Results:

1. **Part A vs. Part B (Initial):** A key finding is that the custom-built classifier from Part A (Macro F1: **0.6809**) significantly outperformed the *initial* Sklearn model from Part B (Macro F1: **0.5877**). The primary difference was the feature engineering. The Part A model used CountVectorizer with **bigrams** (ngram_range=(1,2)), while the initial Part B model used TfidfVectorizer with only **unigrams** (the default). This strongly suggests that for this dataset, the contextual information provided by bigrams was more beneficial than the term-weighting of TF-IDF.

2. **Impact of Hyperparameter Tuning (Part B):** The GridSearchCV in Part B successfully identified the weakness of the initial model. The tuning process, which ran on the development set, found that the optimal parameters were {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 2)}. This confirms the previous finding: adding **bigrams** was the key to improving performance. The resulting best F1 score on the validation set (0.6567) was a substantial improvement over the baseline.

3. **Performance of the BOC Approximation (Part C):** The BOC model yielded the lowest Macro F1 score (**0.6149**) and accuracy (**70.89%**). This result is **entirely expected** and is the central lesson of Part C. The models in Parts A and B were trained on the full dataset of ~180,000 samples, whereas the BOC model was intentionally trained on a small, dynamic sample of only ~10,000. This 94% reduction in training data is the primary reason for the lower performance. The model, even as an ensemble, simply did not have enough data to learn the complex patterns.

Conclusion:

This lab successfully demonstrated the implementation of Naive Bayes from scratch and the power of scikit-learn's Pipeline and GridSearchCV tools. The comparison showed that feature engineering (specifically the inclusion of bigrams) was a highly effective method for improving model performance. Finally, the BOC approximation starkly illustrated the principle that a massive-scale dataset will often outperform a more complex model (or ensemble) that is trained on a limited set of data.