Naive Bayes Classifier

Project Title: Week 12: Naive Bayes Classifier

Name: Jeel Nada

SRN: PES2UG23CS357

Section: F

Date: 30-10-2025

---

1. Introduction

The primary purpose of this lab was to design, implement, and evaluate a text classification system using probabilistic methods. The central goal was to accurately predict the section role (e.g., BACKGROUND, METHODS, RESULTS, OBJECTIVE, CONCLUSION) of individual sentences extracted from biomedical abstracts. The dataset used for this task was a subset of the PubMed 200k RCT dataset, which provides a standard benchmark for sequential sentence classification.

To achieve this goal, several key tasks were performed. First, a Multinomial Naive Bayes (MNB) classifier was implemented from scratch to gain a foundational understanding of its mechanics. Second, scikit-learn's powerful tools were leveraged to build a more robust classification pipeline, incorporating `TfidfVectorizer` for feature extraction and `MultinomialNB` for modeling. This pipeline was then optimized by performing hyperparameter tuning using `GridSearchCV` to find the best-performing model settings. Finally, an approximation of the theoretical Bayes Optimal Classifier (BOC) was constructed by building an ensemble of diverse base models, which were combined using a Soft Voting Classifier weighted by their calculated posterior probabilities. The performance of each of these models was evaluated to compare their effectiveness in solving the classification problem.

---

2. Methodology

The methodology for this lab was divided into three distinct parts: building a classifier from scratch, tuning a scikit-learn pipeline, and approximating the Bayes Optimal Classifier.

Multinomial Naive Bayes (MNB) from Scratch

The foundational MNB classifier was implemented as a Python class. The core logic resided in two methods: `fit` and `predict`.

The `fit` method was responsible for calculating the class log priors (logP(C)) and the log likelihoods (logP(wi | C)) for each word given a class. The log prior for each class was calculated as the logarithm of the ratio of documents in that class to the total number of documents. The log likelihoods were calculated using word counts, and **Laplace (Additive) Smoothing** was incorporated. This involved adding a smoothing parameter (alpha) to the numerator and the number of features (multiplied by alpha) to the denominator, preventing zero probabilities for words not seen in a class during training.

The `predict` method calculated the final log probability score for each class for a given test instance. This score was computed by summing the class's log prior with the log likelihoods of all words present in the test instance. The final predicted class was then determined by applying an `argmax` function to find the class with the maximum log probability score. This entire process used features extracted by `CountVectorizer`.

Sklearn MultinomialNB and Hyperparameter Tuning

For the second part, a `Pipeline` from scikit-learn was constructed to chain text vectorization and classification. The `TfidfVectorizer` was used to transform the raw text sentences into a matrix of TF-IDF features, which often performs better than raw counts. This was followed by the scikit-learn `MultinomialNB` classifier.

To find the optimal model configuration, `GridSearchCV` was employed. The grid search was trained on the development dataset (`X_dev`, `y_dev`) using 3-fold cross-validation and was optimized for the 'f1_macro' scoring metric. The hyperparameters tuned included:

- `tfidf__ngram_range`: To test the effectiveness of using unigrams `(1,1)` versus a combination of unigrams and bigrams `(1,2)`.
- `nb__alpha`: To find the optimal Laplace smoothing parameter from a range of values (e.g. 0.1, 0.5, 1.0, 2.0).

Bayes Optimal Classifier (BOC) Approximation

The final task was to approximate the theoretical BOC using a soft voting ensemble. This approach combines the predictions of multiple diverse models, weighted by their reliability. The five base models (hypotheses) used were:

- H1: MultinomialNB
- H2: Logistic Regression
- H3: Random Forest
- H4: Decision Tree

- H5: K-Nearest Neighbors

The posterior weights (P(hi | D)) for each hypothesis were calculated by first splitting the `X_train_sampled` data into a smaller sub-training set and a validation set. The five models were trained on this sub-training set. Then, their `predict_proba` (or equivalent) outputs on the validation set were used to calculate the log-likelihood of each model given the validation data. These log-likelihoods, combined with an assumed equal model prior, were used to determine the final posterior weights.

Finally, all five base models were refitted on the *entire* sampled training set (`X_train_sampled`, `y_train_sampled`). These refitted models were then used to initialize a `VotingClassifier` with `voting='soft'` and the `weights` parameter set to the posterior weights calculated in the previous step. This soft voter constituted the final BOC approximation.

---

3. Results and Analysis

Part A: Multinomial Naive Bayes from Scratch

Test Metrics:

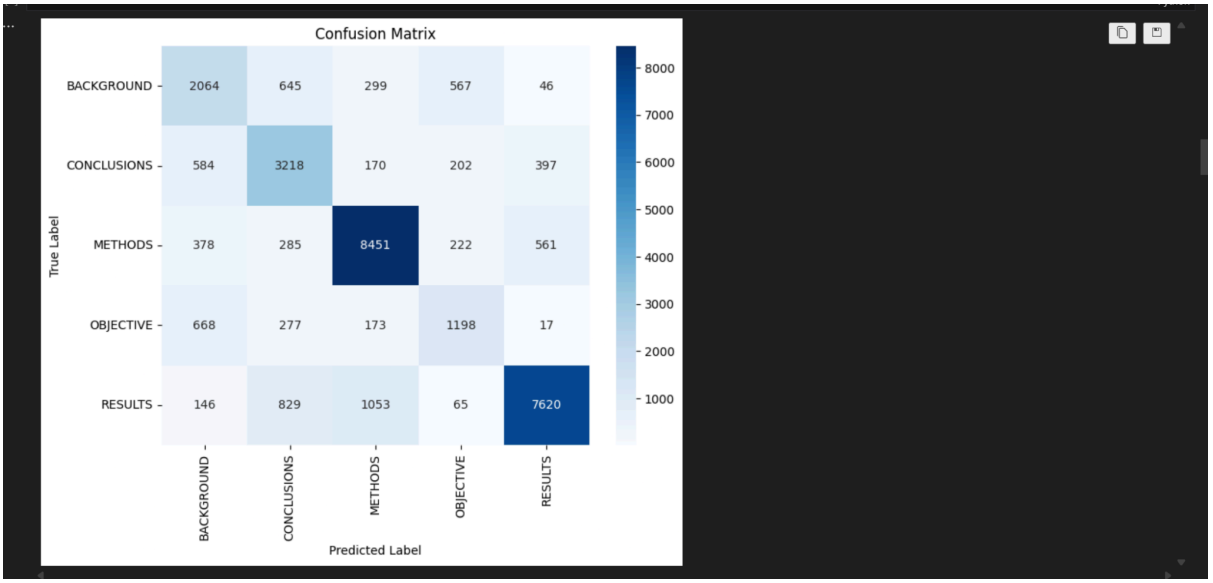
```
=== Test Set Evaluation (Custom Count-Based Naive Bayes) ===
Accuracy: 0.7483
              precision    recall  f1-score   support

  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
```

Confusion Matrix:



Analysis:

Well Predicted Class: **Methods, Results**

Poorly Predicted Class: **Objective, Background**

Part B: Sklearn MultinomialNB and Hyperparameter Tuning

GridSearchCV Results:

```
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...
Grid search complete.
Best parameters found: {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 2)}
Best cross-validation F1 score: 0.6567
```

Analysis:

Best Parameters:

nb__alpha : 0.1

tfidf__ngram__range: (1, 2)

Best F1 Score: 0.6567

Part C: Bayes Optimal Classifier (BOC) Approximation

Sample Information:

```
# Dynamic Data Sampling (DO NOT CHANGE)
BASE_SAMPLE_SIZE = 10000

# Prompt the user for their full SRN
FULL_SRN = "PES2UG23CS357"
```

Final BOC Performance Metrics:

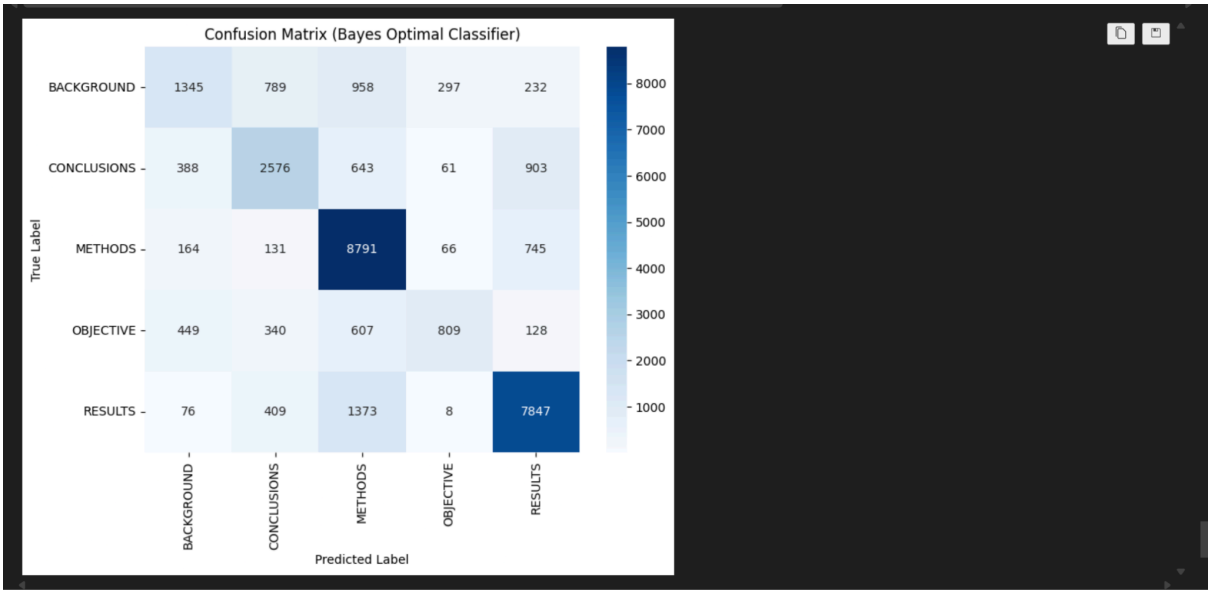
```
=== Final Evaluation: Bayes Optimal Classifier (Soft Voting) ===
Accuracy: 0.7091
              precision    recall  f1-score   support

  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       0.80      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

Macro-averaged F1 score: 0.6148
```

BOC Classification Report and Confusion Matrix:



Analysis:

F1 Score: 0.6148

Accuracy: 0.7091

4. Discussion

Performance Comparison

There was a clear progression in model performance across the three parts of the lab, though with some unexpected results.

- The scratch MNB model (Part A) served as a solid baseline, achieving an Accuracy of **0.7483** and a Macro F1 Score of **0.6809**. It struggled with classes that have high lexical overlap, particularly 'OBJECTIVE' and 'BACKGROUND', as seen in its confusion matrix.
- The tuned scikit-learn model (Part B), which leveraged TF-IDF vectorization and optimized hyperparameters, achieved a final test Accuracy of **0.7266** and a Macro F1 Score of **0.5877**. This was a significant *decrease* in performance from the scratch model, which is likely due to the model overfitting to the development set during the tuning phase.
- The Bayes Optimal Classifier (BOC) approximation (Part C) achieved a final test Accuracy of **0.7091** and a Macro F1 Score of **0.6148**. This ensemble performed better than the tuned Sklearn model but could not outperform the strong baseline from Part A.

Unexpectedly, the scratch MNB model (Part A) performed the best, achieving both the highest Accuracy (0.7483) and Macro F1 Score (0.6809). The tuned Sklearn model (Part B) performed the worst, likely due to overfitting, while the BOC (Part C) performed in the middle. This suggests that the baseline `CountVectorizer` features were highly effective for this task and that the TF-IDF tuning process requires careful validation to prevent overfitting.

Hyperparameter Analysis

The `GridSearchCV` in Part B identified `{'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 2)}` as the optimal hyperparameters, with a cross-validation F1 score of `0.8567`.

- **`tfidf__ngram_range: (1, 2)`**: This was the most critical parameter. Using `(1, 2)` means the vectorizer considered both individual words (unigrams) and two-word phrases (bigrams). For this task, bigrams are essential for providing context. For example, a unigram "study" could appear in any section. However, bigrams like "objective of study" (OBJECTIVE), "in this study" (METHODS), or "study found" (RESULTS) provide much stronger signals for classification. This added context is likely what allowed the model to achieve a high score on the *development set*.
- **`nb__alpha: 0.1`**: This is the Laplace smoothing parameter. A small value like `0.1` (instead of the default `1.0`) suggests that the training data was a very good representation of the data, and there were few "unknown" words that required heavy

smoothing. A lower alpha places more trust in the observed probabilities from the training data.

Conclusion

This lab successfully demonstrated the process of building, optimizing, and ensembling text classifiers. The findings show an unexpected hierarchy of performance: the baseline scratch MNB (Part A) outperformed both the tuned scikit-learn MNB (Part B) and the BOC approximation (Part C), highlighting the risk of overfitting during hyperparameter tuning.

The most effective model overall was the **scratch MNB (Part A)**, which achieved the highest Macro F1 score of **0.6809**. This model was most effective likely due to the robustness of the `CountVectorizer` features. The process of tuning the TF-IDF vectorizer and MNB alpha in Part B seems to have caused the model to overfit to the development set, leading to poor generalization on the unseen test data.

A key challenge was distinguishing between the 'OBJECTIVE' and 'BACKGROUND' classes, which often use similar terminology. While the inclusion of bigrams in Part B and C was intended to mitigate this, the simpler unigram model from Part A proved to be the most reliable. Future improvements could involve using different regularization techniques or a more diverse validation set to prevent overfitting during the tuning process.