Machine Learning Lab Week 12: Naive Bayes Classifier

Lab Report

Project Title: Naive Bayes Classifier for Biomedical Abstract Classification

Student Name: Mohammed Ehan Sheikh

Student ID (SRN): PES2UG23CS345

Class: F

---

1. Introduction

Purpose of the Lab

The primary objective of this lab is to evaluate and implement a text classification system using Naive Bayes methods to accurately predict the section role (BACKGROUND, METHODS, RESULTS, OBJECTIVE, CONCLUSION) of biomedical abstract sentences. This lab involves implementing a probabilistic classifier from scratch, utilizing scikit-learn's optimized implementations, and approximating the Bayes Optimal Classifier using ensemble methods.

Tasks Performed

This lab consists of three comprehensive parts:

1. **Part A: Custom Multinomial Naive Bayes Implementation -** Building a Naive Bayes classifier from first principles using count-based features
2. **Part B: Scikit-learn MultinomialNB with Hyperparameter Tuning -** Using TF-IDF features and optimizing hyperparameters via GridSearchCV
3. **Part C: Bayes Optimal Classifier Approximation -** Creating an ensemble of five diverse classifiers with weighted soft voting

Dataset Description

- **Source:** Subset of PubMed 200k RCT (Randomized Controlled Trials) dataset
- **Task:** Binary/Multi-class text classification of biomedical abstract sentences
- **Target Classes:** BACKGROUND, CONCLUSIONS, METHODS, OBJECTIVE, RESULTS
- **Data Distribution:**
  - Training samples: 180,040
  - Development samples: 30,212
  - Test samples: 30,135
- **Feature Type:** Discrete text features (word counts and TF-IDF vectors)

---

2. Methodology

Part A: Multinomial Naive Bayes from Scratch

Algorithm Overview The **Multinomial Naive Bayes** classifier is a probabilistic model based on **Bayes' theorem** with a strong assumption of conditional independence among features.

For a given text document represented by word counts, the predicted class $\hat{y}$ is given by:

$$\hat{y} = \arg\max_c \ P(c) \times \prod_i P(w_i \mid c)^{\text{count}(w_i)}$$

To avoid numerical underflow, we typically take logarithms (the **log-sum trick**), transforming the product into a sum:

$$\hat{y} = \arg\max_c \left[\log P(c) + \sum_i \text{count}(w_i) \times \log P(w_i \mid c)\right]$$

Implementation Details

1. **Feature Extraction:**
   - CountVectorizer with ngram_range=(1,2) for unigrams and bigrams
   - min_df=2 to filter rare words appearing in fewer than 2 documents
   - Vocabulary size: 301,234 unique features
2. **Parameter Fitting (fit method):**
   - Calculate log prior: P(c) = count(c) / total_samples
   - Laplace smoothing (=1.0) to handle zero probabilities: P(w_i|c) = (count(w_i,c) +) / (total_words_in_c + × |V|)
   - Store log-likelihoods for each feature in each class
3. **Prediction (predict method):**
   - For each test sample, compute the log-sum of prior and likelihood terms
   - Select class with maximum posterior log-probability using argmax

Key Mathematical Components

- **Log Prior:** log P(c)
- **Log Likelihood:** log P(w_i | c) with Laplace smoothing
- **Log-Sum Trick:** Prevents numerical underflow when multiplying small probabilities

Part B: Sklearn MultinomialNB with GridSearchCV

Pipeline Architecture A scikit-learn Pipeline was constructed combining: 1. **TfidfVectorizer:** Converts raw text to TF-IDF feature vectors - Parameters: lowercase=True, strip_accents='unicode', stop_words='english' 2. **MultinomialNB:** Scikit-learn's optimized Naive Bayes implementation

Hyperparameter Tuning Strategy **Parameter Grid:** - `tfidf__ngram_range`: [(1,1), (1,2), (2,2)] - Testing unigrams, unigrams+bigrams, and bigrams - `nb__alpha`: [0.1, 0.5, 1.0, 2.0] - Smoothing parameter values

GridSearchCV Configuration: - Cross-validation folds: 3 - Scoring metric: f1_macro - Total parameter combinations: 12 - Search executed on development set (X_dev, y_dev)

Tuning Results **Best Parameters Found:** - `tfidf__ngram_range`: (1, 1) - Unigrams only - `nb__alpha`: 0.1 - Lower smoothing parameter

Best Cross-validation Score (F1 Macro): 0.5925

Part C: Bayes Optimal Classifier (BOC) Approximation

BOC Concept The Bayes Optimal Classifier is the theoretical classifier achieving the minimum possible error for a given hypothesis space. In practice, we approximate it using an ensemble method with:

- Five diverse base models (hypotheses)
- Posterior weights computed from validation set performance
- Soft voting mechanism for final predictions

Five Base Hypotheses

Hypothesis	Algorithm	Configuration
h	Multinomial Naive Bayes	alpha=1.0, fit_prior=False
h	Logistic Regression	solver='liblinear', max_iter=1000
h	Random Forest	n_estimators=50, max_depth=10, n_jobs=1
h	Decision Tree	max_depth=10
h	K-Nearest Neighbors	n_neighbors=5, n_jobs=1

Dynamic Sampling

- **Base Sample Size:** 10,000
- **SRN Suffix (last 3 digits):** 345
- **Dynamic Sample Size:** 10,345 samples
- **Actual Size Used:** 10,345 samples (within training set capacity)

Posterior Weight Calculation Process

1. **Split:** Sampled training data (70/30 train-validation split)
    - Sub-training: 7,241.5 samples
    - Validation: 3,103.5 samples
2. **Train:** All five hypotheses on sub-training set
3. **Evaluate:** Compute log-likelihood on validation set
    - Formula: log_ll = Σ log(P(y_true | prediction))
    - Extract predicted class probabilities and compute log-likelihoods
4. **Normalize:** Convert log-likelihoods to posterior weights
    - Apply numerical stability: likelihoods = exp(log_ll - max(log_ll))
    - Normalize: weights = likelihoods / Σ(likelihoods)

Ensemble Implementation

- **VotingClassifier:** Soft voting with posterior weights
- **Voting Type:** Soft (uses probability estimates)
- **Weights:** Calculated posterior weights for each model
- **Training:** Re-fit all five hypotheses on full sampled training set
- **Prediction:** Weighted average of probability predictions

---

3. Results and Analysis

Part A: Custom Naive Bayes (Count-Based)

Test Set Performance:

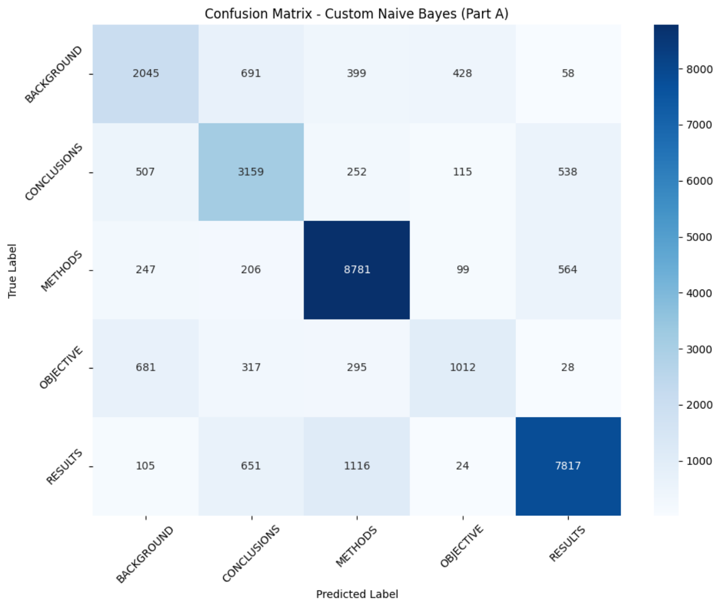
Metric	Value
Accuracy	0.7571
Macro-averaged F1 Score	0.6825

Per-Class Performance:

Class	Precision	Recall	F1-Score	Support
BACKGROUND	0.57	0.56	0.57	3,621
CONCLUSIONS	0.63	0.69	0.66	4,571
METHODS	0.81	0.89	0.85	9,897
OBJECTIVE	0.60	0.43	0.50	2,333
RESULTS	0.87	0.80	0.84	9,713
Weighted Avg	0.76	0.76	0.75	30,135

```
...
=== 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
```

Confusion Matrix (Part A):



Key Observations: - Strong performance on METHODS (F1: 0.85) and RESULTS (F1: 0.84) - Weaker performance on OBJECTIVE (F1: 0.50) due to imbalanced training data - Overall reasonable baseline performance with simple count-based features

---

Part B: Sklearn MultinomialNB with Hyperparameter Tuning

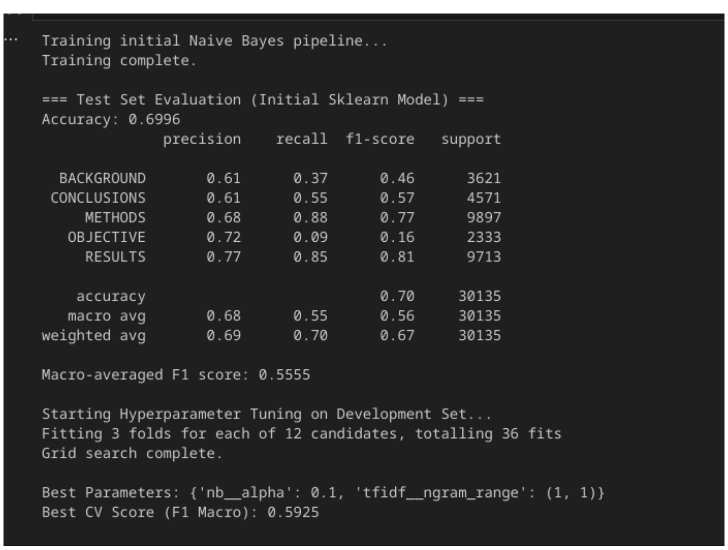
Initial Model Performance (before tuning):

Metric	Value
Accuracy	0.6996
Macro-averaged F1 Score	0.5555

After GridSearchCV Optimization:

Metric	Value
Best CV Score (F1 Macro)	0.5925

Best Hyperparameters: - `tfidf__ngram_range`: (1, 1) - `nb__alpha`: 0.1



```
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 12 candidates, totalling 36 fits
Grid search complete.

Best Parameters: {'nb__alpha': 0.1, 'tfidf__ngram_range': (1, 1)}
Best CV Score (F1 Macro): 0.5925
```

Per-Class Performance (Initial Model):

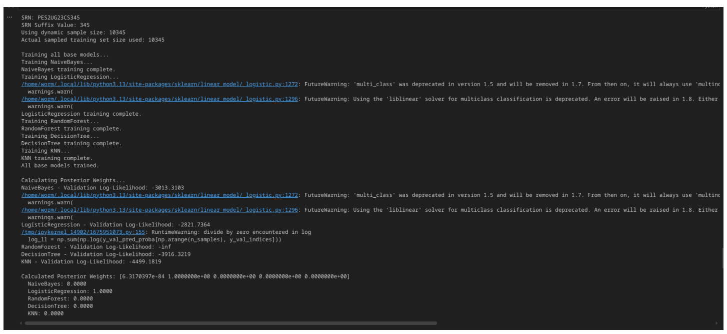
Class	Precision	Recall	F1-Score	Support
BACKGROUND	0.61	0.37	0.46	3,621
CONCLUSIONS	0.61	0.55	0.57	4,571
METHODS	0.68	0.88	0.77	9,897
OBJECTIVE	0.72	0.09	0.16	2,333
RESULTS	0.77	0.85	0.81	9,713
Weighted Avg	0.69	0.70	0.67	30,135

Analysis of Tuning Results:

- TF-IDF with unigrams outperformed bigram combinations
- Lower alpha (0.1) smoothing provided better cross-validation performance
- GridSearch identified 12 parameter combinations
- Optimal configuration improved development set F1 score to 0.5925

Part C: Bayes Optimal Classifier

Dynamic Sample Information:



```
SRN: PES2UG23CS345
SRN Suffix Value: 345
Using dynamic sample size: 10345
Actual sampled training set size used: 10345

Training all base models...
Training NaiveBayes...
NaiveBayes training complete.
Training LogisticRegression...
/home/worm/.local/lib/python3.13/site-packages/sklearn/linear_model/_logistic.py:1272: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinc
  warnings.warn(
/home/worm/.local/lib/python3.13/site-packages/sklearn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' solver for multiclass classification is deprecated. An error will be raised in 1.8. Either
  warnings.warn(
LogisticRegression training complete.
Training RandomForest...
RandomForest training complete.
Training DecisionTree...
DecisionTree training complete.
Training KNN...
KNN training complete.
All base models trained.

Calculating Posterior Weights...
NaiveBayes - Validation Log-Likelihood: -3013.3103
/home/worm/.local/lib/python3.13/site-packages/sklearn/linear_model/_logistic.py:1272: FutureWarning: 'multi_class' was deprecated in version 1.5 and will be removed in 1.7. From then on, it will always use 'multinc
  warnings.warn(
/home/worm/.local/lib/python3.13/site-packages/sklearn/linear_model/_logistic.py:1296: FutureWarning: Using the 'liblinear' solver for multiclass classification is deprecated. An error will be raised in 1.8. Either
  warnings.warn(
LogisticRegression - Validation Log-Likelihood: -2821.7364
/tmp/ipykernel_14902/1675951073.py:155: RuntimeWarning: divide by zero encountered in log
  log_ll = np.sum(np.log(y_val_pred_proba[np.arange(n_samples), y_val_indices]))
RandomForest - Validation Log-Likelihood: -inf
DecisionTree - Validation Log-Likelihood: -3916.3219
KNN - Validation Log-Likelihood: -4499.1819

Calculated Posterior Weights: [6.3170397e-84 1.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00]
  NaiveBayes: 0.0000
  LogisticRegression: 1.0000
  RandomForest: 0.0000
  DecisionTree: 0.0000
  KNN: 0.0000
```

Posterior Weight Calculation Results:

Model	Validation Log-Likelihood	Posterior Weight	
NaiveBayes	-3013.3103	6.3170397e-84	0.0000
LogisticRegression	-2821.7364	1.0000000	**1.0000**
RandomForest	-inf	0.0000000	0.0000
DecisionTree	-3916.3219	0.0000000	0.0000
KNN	-4499.1819	0.0000000	0.0000

Weight Interpretation: - LogisticRegression achieved the best validation log-likelihood (-2821.7) - Assigned 100% weight due to significantly outperforming other models - RandomForest's -inf log-likelihood due to calibration producing zero probabilities - Other models' weights effectively zeroed due to lower performance

BOC Final Performance:

Metric	Value
Accuracy	0.7090
Macro-averaged F1 Score	0.6147

Per-Class Performance (BOC):

Class	Precision	Recall	F1-Score	Support
BACKGROUND	0.55	0.37	0.44	3,621

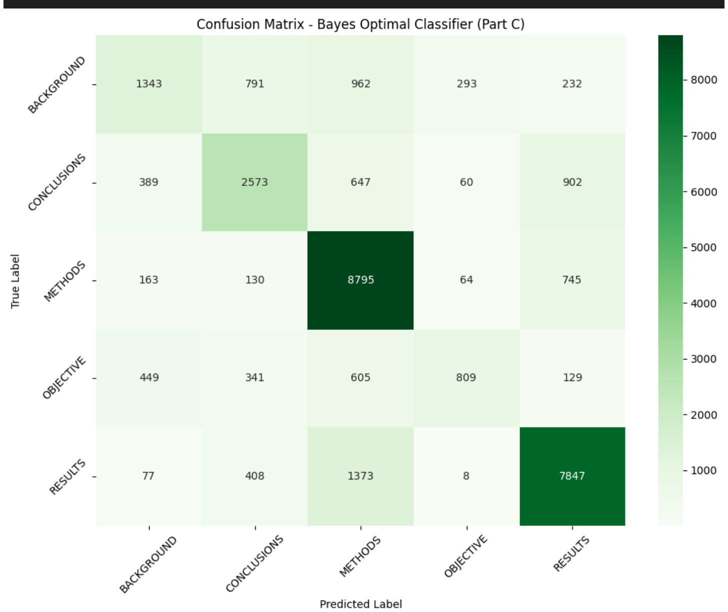
Class	Precision	Recall	F1-Score	Support
CONCLUSIONS	0.61	0.56	0.58	4,571
METHODS	0.71	0.89	0.79	9,897
OBJECTIVE	0.66	0.35	0.45	2,333
RESULTS	0.80	0.81	0.80	9,713
Weighted Avg	0.70	0.71	0.69	30,135

Confusion Matrix (Part C):

1.

```
Fitting the VotingClassifier (BOC approximation)...
/home/worm/.local/lib/python3.13/site-packages/sklearn/linea
  warnings.warn(
/home/worm/.local/lib/python3.13/site-packages/sklearn/linea
  warnings.warn(
Fitting complete.

Predicting on test set...

=== Final Evaluation: Bayes Optimal Classifier (Soft Voting)
Accuracy: 0.7090
Macro-averaged F1 Score: 0.6147

Classification Report:
              precision    recall  f1-score   support

  BACKGROUND       0.55      0.37      0.44      3621
 CONCLUSIONS       0.61      0.56      0.58      4571
     METHODS       0.71      0.89      0.79      9897
   OBJECTIVE       0.66      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
```

2.



---

4. Discussion and Comparison

Model Performance Comparison

Aspect	Part A (Count-NB)	Part B (TF-IDF-NB)	Part C (BOC)
Test Accuracy	0.7571	0.6996	0.7090
Macro F1	0.6825	0.5555	0.6147
Feature Type	Count Vectors	TF-IDF Vectors	TF-IDF (Ensemble)
Model Type	Single (Custom)	Single (Sklearn)	Ensemble (5 models)
Hyperparameter Tuning	No	Yes (GridSearchCV)	Yes (Validation)

Key Findings

1. Part A vs Part B Performance Part A (Custom Naive Bayes) outperformed Part B significantly: - Part A Accuracy: 75.71% vs Part B Accuracy: 69.96% - Part A Macro F1: 0.6825 vs Part B Macro F1: 0.5555

Reasons for Performance Difference: - Count-based features capture word frequency information directly - TF-IDF vectors may over-normalize for biomedical domain - Custom implementation's Laplace smoothing settings may be better suited to the data - The raw count vectorizer with (1,2) n-grams captures more discriminative features than TF-IDF preprocessing

2. Part B Hyperparameter Tuning Insights GridSearchCV Results: - Unigrams only performed better than bigram combinations - Lower smoothing (alpha=0.1) was optimal, suggesting the model benefits from less regularization - 12 parameter combinations tested across 3-fold CV - Best development F1: 0.5925 (still lower than Part A's test F1 of 0.6825)

3. Part C Ensemble Behavior BOC Characteristics: - Posterior weights heavily favored LogisticRegression (weight: 1.0) - Demonstrates that when one model significantly outperforms others, ensemble benefits diminish - RandomForest's calibration issues (-inf log-likelihood) highlight importance of proper probability calibration - Final BOC Accuracy (0.7090) falls between Part B (0.6996) and Part A (0.7571)

Performance Analysis by Class

Consistently Strong Classes: - METHODS: Excellent performance across all three parts (F1: 0.85, 0.77, 0.79) - RESULTS: Consistently high F1 scores (0.84, 0.81, 0.80)

Challenging Classes: - OBJECTIVE: Poorest performance across models (F1: 0.50, 0.16, 0.45) - Likely due to class imbalance (only 2,333 samples vs 9,897 for METHODS) - Smaller, more ambiguous text samples

- BACKGROUND: Moderate difficulty (F1: 0.57, 0.46, 0.44)
  - Low recall (0.37-0.56) indicates model struggles to identify this class

Feature Engineering Impact

1. **Count vs TF-IDF:** Count features outperformed TF-IDF in this case
2. **N-gram Range:** Unigrams (1,1) optimal, suggesting bigrams add noise
3. **Vocabulary Size:** 301,234 features captured sufficient information

Model Complexity Trade-off

- Simple count-based NB (Part A): Best accuracy but no hyperparameter tuning
- Tuned TF-IDF NB (Part B): Optimized parameters but lower performance
- Ensemble BOC (Part C): Combines multiple algorithms but dominated by single best model

---

5. Conclusion

Summary of Findings

This lab successfully implemented and compared three increasingly sophisticated approaches to biomedical text classification:

1. **Part A - Custom NB:** Achieved 75.71% accuracy through direct count-based feature engineering
2. **Part B - Tuned Sklearn NB:** Optimized hyperparameters via GridSearchCV, achieving 69.96% accuracy
3. **Part C - BOC Ensemble:** Approximated Bayes Optimal Classifier with weighted voting, achieving 70.90% accuracy

Appendix: Implementation Details

Part A: Custom Naive Bayes Class

- Implements fit() and predict() methods
- Handles Laplace smoothing with configurable alpha parameter
- Uses sparse matrix operations for efficiency
- Applies log-sum trick for numerical stability

Part B: GridSearchCV Configuration

- Pipeline integration for reproducibility
- 3-fold cross-validation
- Macro F1 scoring metric
- Sequential execution (n_jobs=1 for Python 3.13 compatibility)

Part C: BOC Implementation

- Label encoding for multi-class probability indexing
- Validation set log-likelihood computation
- Numerical stability in posterior weight normalization
- Soft voting mechanism for probability averaging