Department of Computer Science Engineering

UE23CS352A: Machine Learning Lab
Week 12: Naive Bayes Classifier

1. Lab Overview

Welcome to the lab on probabilistic classification using the Naive Bayes algorithm. The primary goal of this lab is to evaluate a text classification system using Naive Bayes methods, to accurately predict the section role (BACKGROUND, METHODS, RESULTS, OBJECTIVE, CONCLUSION) of biomedical abstract sentences,

The dataset used is a subset of the **PubMed 200k RCT** dataset, focusing on classifying abstract sentences into one of five categories.

2. Objectives

In this lab, you will:

- Implement the Multinomial Naive Bayes classifier from scratch.
- Utilize scikit-learn's tools for text vectorization (CountVectorizer, TfidfVectorizer) and modeling (MultinomialNB).
- Perform hyperparameter tuning using GridSearchCV to find the optimal model settings.
- Approximate the Bayes Optimal Classifier (BOC) using an ensemble method built using diverse base models (hypothesis) and a Soft Voting Classifier using calculated posterior weights.

3. Dataset Description

The data consists of sentences extracted from medical abstracts (PubMed Randomised Controlled Trials).

- **Source:** Subset of PubMed 200k RCT (text classification).
- **Target:** Predict the appropriate section title (label) for a given sentence (sentence).
- **Classes (Target Names):** BACKGROUND, CONCLUSIONS, METHODS, OBJECTIVE, RESULTS.
- **Splits:** The data is provided in three files (train.txt, dev.txt, test.txt) and is loaded by the provided load_pubmed_rct_file function.

4. Key Concepts

- **Multinomial Naive Bayes (MNB):** A probabilistic classifier suitable for classification with discrete features (like word counts or TF-IDF values in text data). It relies on the strong, "naive" assumption of conditional independence between features given the class.

- **Laplace Smoothing (Additive Smoothing):** A technique used in MNB to handle words that appear zero times in a specific class. By adding a smoothing parameter (α, typically 1), it prevents zero probabilities, ensuring the model remains stable and generalizable.

- **Log-Sum Trick:** Logarithms are used to prevent numerical underflow when multiplying many small probabilities (likelihoods), as $\log(A \times B) = \log(A) + \log(B)$.

- **Bayes Optimal Classifier (BOC):** The theoretical classifier that yields the lowest possible classification error for a given problem space. In practice, we approximate it using ensemble methods, such as a Hard Voting Classifier composed of models with diverse strengths.

5. Instructions and Tasks

You must complete all the // TODO: sections in the provided Notebook.

Part A: Multinomial Naive Bayes from Scratch

In this section, you will implement the core mathematical components of the Multinomial Naive Bayes classifier to understand its mechanism.

1. **Data Loading:** Ensure the data loading cell is correctly executed to populate X_train, y_train, X_dev, y_dev, X_test, and y_test.
2. **Custom Classifier Logic:** Complete the NaiveBayesClassifier class:
   - In the fit method, implement the calculation of the log prior ($\log P(C)$) and the log likelihood ($\log P(w_i|C)$) for each class, ensuring you include Laplace Smoothing.
   - In the predict method, implement the calculation of the final log probability for a new instance, which is the sum of the log prior and the log likelihood contributions. Use argmax to select the class with the maximum score.
3. **Feature Extraction:** Initialize CountVectorizer and set appropriate parameters for ngram_range (e.g., single words or bigrams) and min_df (e.g., to ignore words that appear very infrequently).
4. **Training and Evaluation:** Train your custom nb_model using the Count-based features from the training set and evaluate its performance on the test set (X_test_counts).
5. **Visualization:** Generate and display a visual Confusion Matrix (heatmap) for the custom classifier's predictions on the test set.

Part B: Sklearn MultinomialNB and Hyperparameter Tuning

In this section, you will utilize scikit-learn's MultinomialNB with TF-IDF features and optimize its hyperparameters.

1. **Initial Pipeline:** Define a Pipeline named pipeline that chains a TfidfVectorizer and a MultinomialNB classifier using default parameters. Train it on X_{train} and evaluate its metrics on X_{test}.

2. **Hyperparameter Grid:** Define the param_grid for GridSearchCV. You must tune at least two parameters across two components:
   - tfidf__ngram_range: Experiment with unigrams, bigrams, or both.
   - nb__alpha: Experiment with different values of the smoothing parameter (e.g., [0.1, 0.5, 1.0, 2.0]).
3. **Grid Search:** Initialize and fit GridSearchCV using the pipeline and param_grid.
   - Crucial: Fit the grid search on the development data (X_dev, y_dev).
   - Use cv=3 and scoring='f1_macro'.
4. **Reporting:** Print the best_params_ and the corresponding best_score_ found by the grid search.

Part C: Bayes Optimal Classifier

The final task is to approximate the theoretical Bayes Optimal Classifier (BOC). You will use five diverse base models, all trained on a sampled subset of the main training data.

- H_1: Multinomial NB
- H_2: Logistic Regression
- H_3: Random Forest
- H_4: Decision Tree
- H_5: K-Nearest Neighbors

You must complete the following steps in the provided notebook:

1. **Posterior Weight Calculation:**
   - Split the X_train_sampled and y_train_sampled into a smaller sub-training set and a validation set.
   - Train all five base hypotheses (H_1 to H_5) on the sub-training set.
   - Calculate the log-likelihood for each model by evaluating its predict_proba results against the true labels of the validation set.
   - Use the calculated log-likelihoods and equal model priors to determine the final posterior weights ($\mathrm{P}(h_i|D)$) for each hypothesis.
2. **Model Refitting:** Refit all five hypotheses (H_1 to H_5) on the full sampled training set (X_train_sampled, y_train_sampled).
3. **Ensemble Implementation:**
   - Initialize the VotingClassifier (boc_soft_voter) using the five refitted hypotheses.
   - Crucially, set the voting='soft' parameter and use the calculated posterior weights ($\mathrm{P}(h_i|D)$) to assign the weights for the voter.
   - Train the boc_soft_voter on the full sampled training set (X_train_sampled, y_train_sampled).
4. **Prediction and Evaluation:**
   - Make the final predictions on the full test set (X_test)
   - Calculate and print the final evaluation metrics: Accuracy and Macro F1 Score.
   - Generate and display the Classification Report and the Confusion Matrix visualization.

6. Expected Deliverables

1. **Completed Jupyter Notebook** (.ipynb)
    - All // TODO: sections filled.
    - Fully executed with all outputs (metrics, plots, reports) visible.
    - Clean, well-documented, error-free code.
2. **Lab Report** (.pdf)
    - **Title Page** (Project Title, Your Name, SRN, Course, Date).
    - **Introduction** (Purpose of the lab, tasks performed).
    - **Methodology** (Briefly describe the implementation approach for MNB and BOC)
    - **Results and Analysis** (Screenshots of plots and metrics):
        - Part A: Screenshot of final test Accuracy, F1 Score and Confusion Matrix.
        - Part B: Screenshot of best hyperparameters found and their resulting F1 score.
        - Part C:
            1. Screenshot of SRN and sample size.
            2. Screenshot of BOC final Accuracy, F1 Score and Confusion Matrix.
    - **Discussion:** Compare the performance of your scratch model (Part A) vs. the tuned Sklearn model (Part B) vs. the BOC approximation (Part C).