Project 1: Machine Learning
Naive Bayes and Logistic Regression for Text Classification

Instructor: Tahrima Rahman

Introduction

In this project you will implement and evaluate Naive Bayes and Logistic Regression for text classification.
Note from the TA: You must use Python 3.9 or later to implement your algorithms.

Evaluation Metrics

Before implementing the classifiers, it is essential to understand how to measure their performance. Below are the key evaluation metrics that you will report:

- **Accuracy:** This measures how often the model correctly classifies an email as spam or not spam. It is calculated as:

$$\text{Accuracy} = \frac{\text{Number of correctly classified emails}}{\text{Total number of emails}}$$

  A higher accuracy means the model is making fewer mistakes overall.

- **Precision:** This measures how many of the emails that the model classified as spam are actually spam. It helps answer the question: "When the model predicts spam, how often is it correct?" It is calculated as:

$$\text{Precision} = \frac{\text{Number of correctly classified spam emails}}{\text{Total number of emails predicted as spam}}$$

  A high precision means the model is making fewer mistakes when flagging emails as spam.

- **Recall:** This measures how many of the actual spam emails were correctly identified by the model. It helps answer the question: "Out of all the spam emails, how many did the model find?" It is calculated as:

$$\text{Recall} = \frac{\text{Number of correctly classified spam emails}}{\text{Total number of actual spam emails}}$$

  A high recall means the model is good at catching most spam emails, even if it occasionally mislabels some non-spam emails as spam.

- **F1 Score:** This metric provides a balance between precision and recall. It is useful when we need to ensure that both spam detection and avoiding false alarms are important. It is calculated as:

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

A high F1 score means the model is both precise in its spam predictions and good at catching actual spam.

These metrics together provide a complete picture of how well a model is performing. For example, a model with high precision but low recall correctly identifies only a few spam emails while missing many others. On the other hand, a model with high recall but low precision flags many non-spam emails as spam. The goal is to find a balance between the two.

0 Points Download the spam/ham (ham is not spam) datasets (see the zip file). The datasets were used in the Metsis et al. paper [1]. There are three datasets. You have to perform the experiments described below on all the three datasets. Each data set is divided into two (sub)sets: training set and test set. Each of them has two directories: spam and ham. All files in the spam and ham folders are spam and ham messages respectively.

20 points Your task is to transform a collection of emails into a structured numerical representation in the form of a **feature matrix**, where:

- **Columns represent features** (i.e., words from a predefined vocabulary).
- **Rows represent examples** (i.e., individual emails).
- Each entry in the matrix quantifies the presence of a word (column) in a given email (row).

This transformation is essential for applying machine learning algorithms, as they require numerical input rather than raw text.

You will use two approaches for this conversion: (1) the Bag of words approach and (2) the Bernoulli approach, described below.

1. Bag of Words (BoW) Approach

In this approach, we first construct a **vocabulary** consisting of all unique words appearing in the training set. Let the vocabulary contain w words. Thus, our feature matrix will have w columns since each word is a feature. Each email is then represented as a **vector of word frequencies**:

- The vector has a length of w, with each entry corresponding to a word in the vocabulary.
- The value of each entry is the **count** of that word in the email (i.e., how many times it appears).

For example, suppose our vocabulary contains the words:

$$\{\text{"data"}, \text{"machine"}, \text{"learning"}, \text{"email"}\}$$

If an email contains the text:

"Machine learning is fun. Machine learning is powerful."

The corresponding BoW vector (row in our feature matrix) would be:

$$[0, 2, 2, 0]$$

Here, *"machine"* appears **twice**, *"learning"* appears **twice**, and the words *"data"* and *"email"* do not appear at all.

This representation **captures word frequency**, which can be useful for distinguishing documents based on their content.

2. Bernoulli Approach (Binary Word Presence Approach)

The Bernoulli approach also uses the same **vocabulary of w words**, but instead of counting word occurrences, it records only whether a word is **present or absent** in the email:

- The vector has a length of w, where each entry is either **0** or **1**.
- **1** indicates that the corresponding word appears at least once in the email.
- **0** indicates that the word does not appear at all.

Using the same vocabulary as before:

$$\{\text{"data"}, \text{"machine"}, \text{"learning"}, \text{"email"}\}$$

For the same email text:

"Machine learning is fun. Machine learning is powerful."

The Bernoulli representation (row in our feature matrix) would be:

$$[0, 1, 1, 0]$$

Even though *"machine"* and *"learning"* appear twice, the Bernoulli approach **only marks their presence (1) rather than counting occurrences**. This representation is useful in cases where the frequency of a word is not as important as whether it appears at all (e.g., spam detection).

Both approaches transform unstructured text data into a structured format suitable for machine learning algorithms.

Converting the Datasets into Feature Matrices

Each dataset consists of a **training set** and a **test set**, both containing emails labeled as spam or not spam (ham). You will transform these datasets into structured numerical matrices using both the **Bag of Words** and **Bernoulli** representations.

1. **Building the Vocabulary:**

- Collect all unique words from the training set to create a fixed vocabulary (the features).
- Convert all text to lowercase and remove punctuation to ensure consistency.
- Ignore very common words (stopwords) such as "the," "is," and "and" to reduce noise.

2. **Generating Feature Matrices for Each Representation:**
   - **Bag of Words:**
     * For each email, count how many times each word from the vocabulary appears.
     * Store these counts in a row of the feature matrix.
   - **Bernoulli Representation:**
     * For each email, mark each word's presence as 1 if it appears at least once and 0 otherwise.
     * Store this binary information in a row of the feature matrix.
3. **Applying the Transformation to the Test Set:**
   - Use the same vocabulary built from the training set.
   - Convert each email in the test set into a feature vector using the same method as the training set.
   - Any word appearing in a test email but not in the vocabulary is ignored.

By applying these steps, you will create a total of **12 datasets**:

- **6 training sets** (one for each approach per dataset).
- **6 test sets** (one for each approach per dataset).

You may use any text processing library (e.g., NLTK) to help with tokenization and text preprocessing. However, be sure to cite any external tools used in your report.

Storing the Datasets in CSV Format

Each dataset should be stored as a **CSV (Comma-Separated Values)** file to ensure compatibility with machine learning libraries. The format of the CSV file should be as follows:

- Each row corresponds to a single email.
- The first w columns represent the feature values for each word in the vocabulary.
- The last column contains the label, where:
  * **1** represents a spam email.
  * **0** represents a non-spam (ham) email.

CSV Format Example

If the vocabulary consists of three words: *"offer," "free," "win"*, and we have three emails, the CSV file would look like:

```
offer,free,win,label
1,2,0,1
0,1,1,0
1,0,1,1
```

For the **Bag of Words representation**:

- The numbers in each feature column represent the **count** of the corresponding word in the email.

For the **Bernoulli representation**:

- The numbers in each feature column are **either 0 or 1**, indicating the presence or absence of the corresponding word.

Dataset Naming Convention

Each dataset should follow the naming format:

`dataset_representation_set.csv`

where:

- `dataset`: The dataset name (`enron1`, `enron2`, or `enron4`).
- `representation`: The text representation method:
  - `bow` for the **Bag of Words** model.
  - `bernoulli` for the **Bernoulli model**.
- `set`: The dataset split:
  - `train` for the training set.
  - `test` for the test set.

Filenames that you will submit

- `enron1_bow_train.csv`, `enron1_bow_test.csv`
- `enron1_bernoulli_train.csv`, `enron1_bernoulli_test.csv`
- `enron2_bow_train.csv`, `enron2_bow_test.csv`
- `enron2_bernoulli_train.csv`, `enron2_bernoulli_test.csv`
- `enron4_bow_train.csv`, `enron4_bow_test.csv`
- `enron4_bernoulli_train.csv`, `enron4_bernoulli_test.csv`

Guidelines

- Use **lowercase** filenames for consistency.
- Separate words with **underscores (_)** instead of spaces.
- Ensure all datasets follow this structure for easy identification and automated processing.

25 points Multinomial Naive Bayes for Text Classification

Implement the **Multinomial Naive Bayes** algorithm for text classification as described in `http://nlp.stanford.edu/IR-book/pdf/13bayes.pdf` (refer to Figure 13.2). This algorithm is specifically designed for text data and operates on the **Bag of Words** representation.

Implementation Details:

- Apply **add-one Laplace smoothing** to handle zero probabilities.
- Perform all probability calculations in **log-space** to prevent numerical underflow.
- Train your model using the **Bag of Words** training datasets and evaluate its performance on the corresponding test datasets.

After training the model, report the **accuracy, precision, recall, and F1-score** on the test set.

Important: Use only the datasets generated using the **Bag of Words** approach for this part.

25 points Discrete Naive Bayes for Binary Text Classification

Implement the **Discrete Naive Bayes** algorithm (also known as Bernoulli Naive Bayes), which models each words presence or absence in a document.

Implementation Details:

- Use **add-one Laplace smoothing** to avoid zero probabilities.
- Perform all computations in **log-space** to prevent underflow.
- Train your model using the **Bernoulli** training datasets and evaluate its performance on the corresponding test datasets.

After training, report the **accuracy, precision, recall, and F1-score** on the test set.

Important: Use only the datasets generated using the **Bernoulli** approach for this part.

30 points Logistic Regression with ℓ_2 Regularization

Implement the **Maximum Conditional A Posteriori (MCAP) Logistic Regression** algorithm with ℓ_2 regularization.

Implementation Details:

- Use gradient ascent to learn the model parameters.
- Perform **hyperparameter tuning** by selecting an optimal λ value for ℓ_2 regularization.
- Split the training data into **70% training** and **30% validation** for tuning λ.

- After selecting the best λ, train the model on the full training set.
- Report **accuracy, precision, recall, and F1-score** on the test set.

Since Logistic Regression can handle both feature representations, use both the **Bag of Words** and **Bernoulli** datasets.

Important: Ensure a suitable learning rate to avoid slow convergence or divergence. Set a hard limit on the number of iterations to control runtime.

What to Turn in?

Make sure to submit a single zip file containing the following:

- Your complete code implementation, along with a `README` file that provides instructions for compiling and running the code.
- All **12 CSV files** corresponding to the generated datasets, following the specified naming convention.
- A detailed write-up reporting accuracy, precision, recall, and F1-score for the following models on all three datasets:
  - Multinomial Naive Bayes (Bag of Words model)
  - Discrete Naive Bayes (Bernoulli model)
  - Logistic Regression (both Bag of Words and Bernoulli models)
- In the report, describe how hyperparameters were tuned (e.g., values of λ, iteration limits).
- Answer the following questions in your report:
  1. Which combination of algorithm and data representation yielded the best performance? Why?
  2. Did Multinomial Naive Bayes perform better than Logistic Regression on the Bag of Words representation? Explain.
  3. Did Discrete Naive Bayes perform better than Logistic Regression on the Bernoulli representation? Explain.

References

[1] V. Metsis, I. Androutsopoulos, and G. Paliouras, "Spam Filtering with Naive Bayes - Which Naive Bayes?" Proceedings of the 3rd Conference on Email and Anti-Spam (CEAS 2006), Mountain View, CA, USA, 2006.