CS 4375: Introduction to Machine Learning Project 1: Naive Bayes and Logistic Regression for Text Classification

In this project, you will build simple machine learning models to detect spam emails. You will implement and compare **Logistic Regression** and **two types of Naive Bayes** (*Multinomial Naive Bayes* and *Bernoulli Naive Bayes*) models.

Note from the TA: You must use Python 3.9 or later to implement your algorithms. You may use *numpy* for array operations and basic math (e.g., dot products, exponentiation, log), but you must implement the core learning algorithms (probability estimation, gradient updates) yourself — do not use `sklearn, tensorflow`, or other high-level ML libraries.

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: `enron1`, `enron2`, and `enron4` each representing emails from a different mailbox. You have to perform the experiments described below on all the three datasets. Each data set is divided into two sets: **training** and **test**. Each of them has two directories: `spam` and `ham`. All files in the `spam` and `ham` folders are spam and ham messages respectively.

Step 1: Data Preparation (20 points)

Your task is to transform a collection of emails into a structured numerical representation in the form of a **data 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. *Use the training set only to build the vocabulary; do not use the test set (to avoid leakage). Once you have chosen the features (vocabulary), apply exactly the same preprocessing to both train and test sets to construct numerical feature vectors.*

- **Bag of Words (BoW) Representation**:
  In this approach, we first construct a **vocabulary** consisting of all unique words appearing in the training set. Let the vocabulary contain n words. Thus, our data matrix will have n columns since each word is a feature. Each email is then represented as a **vector of word frequencies (counts)**:

- The vector has a length of n, 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.

- **Bernoulli Representation**:
  The Bernoulli approach also uses the same **vocabulary of n 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 n, 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.

  Going back to our previous example, 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).

Now perform the following steps to transform unstructured text data into a structured format using BoW and Bernoulli approaches suitable for machine learning algorithms.

Main Steps to Convert the Raw Text Datasets into Data Matrices

Each dataset consists of a **training set** and a **test set**, both containing spam and not spam (ham) emails (see their respective folders). 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). *Do not use the test set for vocabulary construction.*
   - Convert all text to lowercase and remove punctuation to ensure consistency (use the same rules for train and test).
   - Ignore very common words (stopwords) such as ”the,” ”is,” and ”and” to reduce noise. 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.
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 data 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 data 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.
4. **Storing the Datasets in CSV Format:** By applying these steps, you will create a total of **12 datasets** (3 datasets $\times$ 2 representations $\times$ train/test):
   - **6 training sets** (one BoW and one Bernoulli per dataset).
   - **6 test sets** (one BoW and one Bernoulli per dataset).

   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 (*positive class*).
  - **0** represents a non-spam (ham) email.
- Include a header row with column names (the last column name should be `label`), as in the example below.

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
```

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.

Step 2: Logistic Regression (40 points)

Implement the Logistic Regression algorithm with L2 regularization for both BoW and Bernoulli representations. That is for each dataset {`enron1, enron2, enron4`}, and for each representation {`BoW,Bernoulli`} you will learn a logistic regression classifier. The total number of classifiers learned will be 6. See the supplementary notes provided with this project on implementing logistic regression efficiently.

Implementation Details:

- Use gradient descent to learn each logistic regression classifier.
- Perform **hyperparameter tuning** by selecting an optimal λ value for L2-regularization.
- Use all three variants of gradient descent to learn the parameters of each classifier. As an example, in minibatch gradient descent, you may try batch sizes of $\{50, 100\}$. Set a suitable number of epochs or run your algorithm until convergence.
- Split the training data into **70% training** and **30% validation** for tuning λ (*use the training split only; do not use the test set for tuning*). As an example, you may try the following values for λ: $\{0.01, 0.1, 1.0, 10.0\}$.
- After selecting the best λ, train the model on the combined training and validation set with the best λ that you found in the previous step.
- Report **accuracy, precision, recall, and F1-score** on the test set (*classify as spam if* $P(y{=}1 \mid x) \geq 0.5$). See section on "Evaluation Metrics" at the end of this document on how to compute each of these scores.

Important: Ensure a suitable learning rate (e.g. 0.001 or 0.01) to avoid slow convergence or divergence. Set a hard limit on the number of iterations (e.g. 500 iterations) to control runtime. An example results table is show in Table 1

Table 1: Logistic Regression Results

Dataset	Repr.	GD Variant	Best λ	Accuracy	Precision	Recall	F1
Enron1	BoW	Batch GD	–	–	–	–	–
Enron1	BoW	Mini-batch GD	–	–	–	–	–
Enron1	BoW	SGD	–	–	–	–	–
Enron1	Bern	Batch GD	–	–	–	–	–
Enron1	Bern	Mini-batch GD	–	–	–	–	–
Enron1	Bern	SGD	–	–	–	–	–
Enron2	BoW	Batch GD	–	–	–	–	–
Enron2	BoW	Mini-batch GD	–	–	–	–	–
Enron2	BoW	SGD	–	–	–	–	–
Enron2	Bern	Batch GD	–	–	–	–	–
Enron2	Bern	Mini-batch GD	–	–	–	–	–
Enron2	Bern	SGD	–	–	–	–	–
Enron4	BoW	Batch GD	–	–	–	–	–
Enron4	BoW	Mini-batch GD	–	–	–	–	–
Enron4	BoW	SGD	–	–	–	–	–
Enron4	Bern	Batch GD	–	–	–	–	–
Enron4	Bern	Mini-batch GD	–	–	–	–	–
Enron4	Bern	SGD	–	–	–	–	–

Step 3: Multinomial Naive Bayes (20 points)

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

- Apply **add-one Laplace smoothing** ($\alpha = 1$) to handle zero probabilities.
- Perform all probability calculations in **log-space** to prevent numerical underflow. Note that, to avoid underflow during test time prediction, do not exponentiate the log-probabilities.
- 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.

Table 2: Naive Bayes results for different variants.

Dataset	NB Variant	Accuracy	Precision	Recall	F1
Enron1	Multinomial (BoW)	–	–	–	–
Enron1	Bernoulli	–	–	–	–
Enron2	Multinomial (BoW)	–	–	–	–
Enron2	Bernoulli	–	–	–	–
Enron4	Multinomial (BoW)	–	–	–	–
Enron4	Bernoulli	–	–	–	–

Step 4: Bernoulli Naive Bayes (20 points)

Implement the **Bernoulli Naive Bayes** algorithm (also known as *Binary/Discrete Naive Bayes*), which models each word's 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.

Step 5: What to Submit

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

- The AI transcript file containing all your interactions with the AI assistant related to the project. This should include every prompt and response in chronological order, without removing intermediate attempts or mistakes. Save it as plain text or PDF and name it using the convention `cs4375_project1_transcript_(your student_id).pdf`. Instructions on using AI for completing your projects are included in a separate document; please refer to it for details.
- 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 test datasets (*treat spam = 1 as the positive class for precision/recall/F1*):
  - Multinomial Naive Bayes (Bag of Words model) (see example table 2)
  - Bernoulli Naive Bayes (Binary Discrete model) (see example 2)
  - Logistic Regression (both Bag of Words and Bernoulli models) (see example table of results 1

  Report results separately for each test dataset (enron1, enron2, enron4). Do not average across datasets.
- In the report, describe how hyperparameters were tuned (e.g., values of λ, iteration limits).
- Answer the following questions in your report:
  1. Did Naive Bayes or Logistic Regression perform better? Why?
  2. Which combination of algorithm and data representation yielded the best performance? Why?
  3. Did Multinomial Naive Bayes perform better than Logistic Regression on the Bag of Words representation? Explain.
  4. Did Bernoulli Naive Bayes perform better than Logistic Regression on the Bernoulli representation? Explain.
  5. Which variant of gradient descent was better in terms of speed and/or accuracy when learning logistic regression classifiers.

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.

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.