CS5691 - Pattern Recognition in Machine Learning

Programming Assignemnt 3

lab Report

Name - Jay Jitendra Prajapati

Roll number - ME21B143

Contents

1 Aim . . . 3
2 About the Dataset . . . 3
3 Feature Extraction . . . 3
4 Architecture of the classifier . . . 5
5 Algorithm . . . 6
6 Hyper-Paramter Tuning and Cross validation . . . 8
7 Cross validation Results . . . 8
8 Results . . . 9

1 Aim

- The objective of this assignment is to develop a spam classifier using labeled training data.

2 About the Dataset

- The Enron Dataset has been utilized for this study, consisting of a substantial volume of emails. However, a subset comprising 2000 spam emails and 2000 ham (non-spam) emails was selected for the purpose of training. It is noteworthy that the Enron dataset lacks emails pertinent to the Indian context, prompting the addition of supplementary emails. For validation, an additional set of 200 spam emails and 200 ham/non-spam emails were sourced from the same dataset.
- For the access to the finalized dataset, please refer to the following link: Final Dataset Dwonload

3 Feature Extraction

- The frequency of each word in the training set is used as our feature. In the dataset, numerous words irrelevant to classification, commonly referred to as stop words, were identified and subsequently eliminated in the following procedure
- A dictionary encompassing all possible words occurring across the dataset was created, along with the respective frequency counts for each word.
- A graphical representation depicting the word index versus frequency was generated. Subsequently, words exceeding a predefined threshold (3000 in our instance) were filtered out.

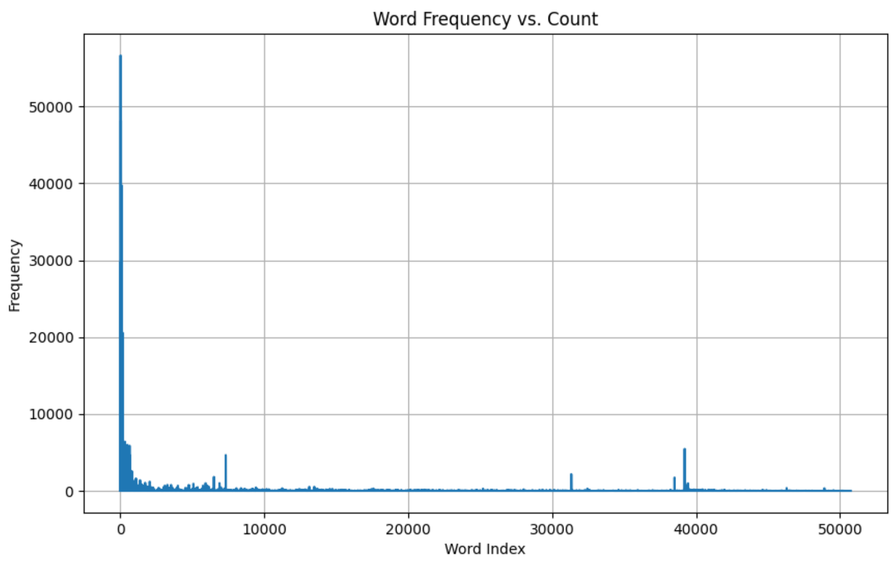

Figure 1: Schematic representation of word frequency versus count

- Upon analysis of the frequency distribution of words within the dataset, it was observed that words exceeding a frequency threshold of 3000 were deemed irrelevant for classification purposes.
- Consequently, a rigorous filtration process was undertaken to remove such words from the emails within the dataset. This strategic curation ensures the elimination of potentially confounding variables, thereby enhancing the efficacy of subsequent analyses and classification algorithms.

4 Architecture of the classifier

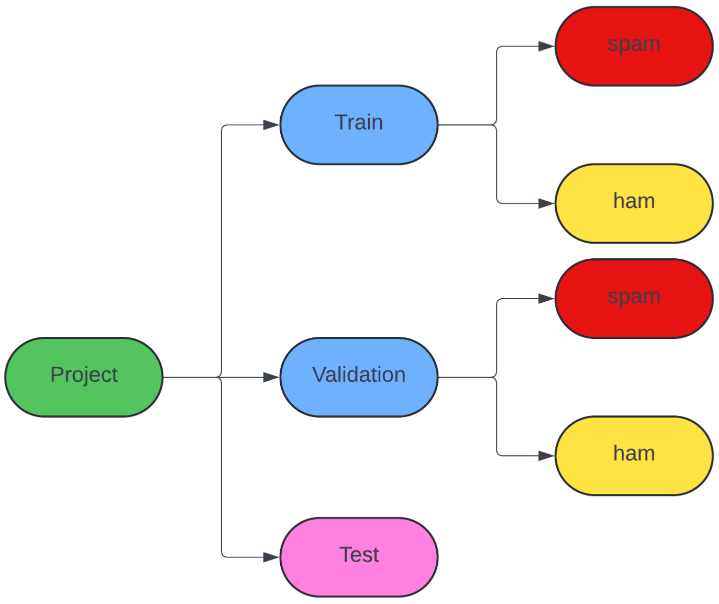

Figure 2: structure of the folder

- Figure 2 illustrates the hierarchical arrangement of folders within the dataset. The program selectively retrieves training data from folders denoted in blue, where the folder designation determines the label assigned to the data. Conversely, the **'test'** folder, highlighted in pink, comprises emails designated for classification using the developed classifier algorithm.

- The following figure 3 depicts the overarching procedure employed in training our classifier. The **'Train.py'** script is utilized to process both training and validation data, employing cross-validation techniques to yield the optimal classifier. Subsequently, the trained model and count vectorizer objects are saved as '.object' files within the same directory for utilization in classification tasks.

- The **'Classify.py'** script accepts input in the form of model and count vectorizer objects, alongside test data, and generates the desired accuracy metrics based on the evaluation of the test dataset.

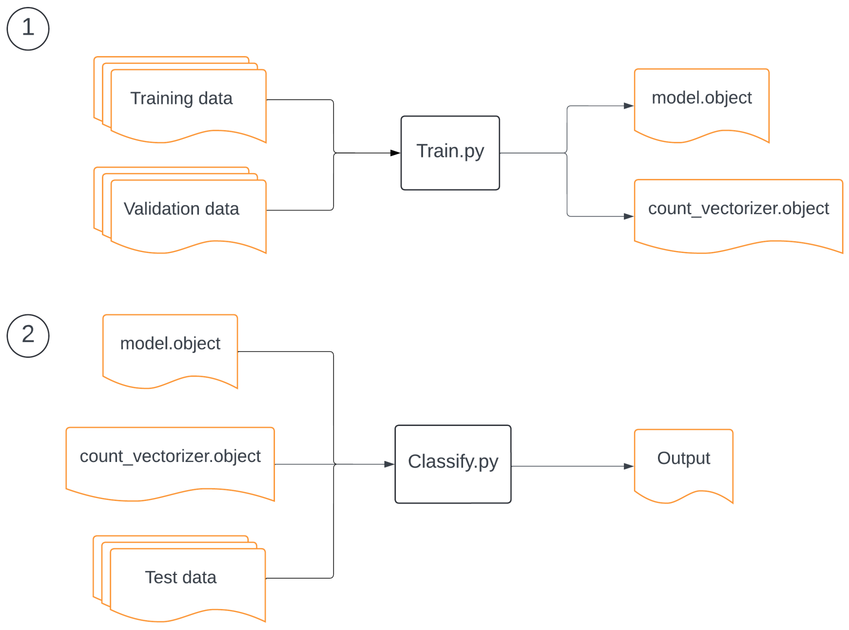

Figure 3: structure of the folder

5 Algorithm

- **Naive Bayes Algorithm:**
  - **Description:** Naive Bayes algorithm is straightforward to implement. However, it relies on the class conditional independence assumption, which may not hold

true in real-world scenarios, leading to loss of data and reduced effectiveness and hence not used.

- **Support Vector Machine (SVM):**
  - **Description:** SVM is the chosen model for its speed and accuracy compared to Naive Bayes. It utilizes word frequencies for training. The implementation involves the following steps:
    1. **Step 1:** Load the dataset from the training and validation folders using the `load_data` function. Convert emails to lowercase and remove stop words.
    2. **Step 2:** Extract features using the `CountVectorizer` function from the `sklearn` library.
    3. **Step 3:** Train the model using different kernels. Initially, employ a linear kernel and evaluate accuracy on the validation dataset. Then, utilize an RBF kernel with varying values of C. If accuracy improves, store this model along with the extracted features.
    4. **Step 4:** After trying all kernels, select the best model and extracted features, saving them as `model.object` and `count_vectorizer.object`, respectively, representing the final classifier.

- **Procedure for Test Email Classification:**
  1. **Step 1:** Load the test dataset from the designated folder using the `Load_data` function. This function converts emails to lowercase and returns them as a list.
  2. **Step 2:** Retrieve the SVM model stored in `model.object` along with the associated feature extraction parameters stored in `count_vectorizer.object`. Employ

the `CountVectorizer` to extract features from the test emails.

3. **Step 3:** Utilize the SVM model to classify the test emails based on the extracted features.
4. **Step 4:** Record the predicted classifications into the file named `'output.txt'` for further analysis.

6 Hyper-Paramter Tuning and Cross validation

- For the purpose of cross-validation, the validation dataset comprises 200 spam and 200 ham emails. These emails, randomly selected from the larger dataset, offer valuable insights into the performance of the classification algorithm.
- The following kernels have been selected for evaluation:
  1. **Linear kernel**
  2. **Radial Basis Function kernel**
  3. **Polynomial kernel**
- In the RBF kernel, the 'C' hyperparameter is fine-tuned by cross-validating the Support Vector Machine (SVM) model using the validation dataset. Subsequently, the model demonstrating the highest accuracy on the validation set is chosen for further analysis.

7 Cross validation Results

- The table below presents the results obtained after cross-validating various SVM models using the validation dataset.

Kernels used	**Accuracy Percentage**
Linear kernel	95.26184538653366
RBF kernel with C=1	95.51122194513717
RBF kernel with C=2	**96.75810473815461**
RBF kernel with C=3	96.25935162094763
Polynomial kernel with degree=2	76.05985037406484
Polynomial kernel with degree=3	61.59600997506235
Polynomial kernel with degree=4	57.10723192019949

Table 1: Cross Validation results on various kernels

Conclusion: Upon evaluation, the **RBF kernel** with a regularization parameter of **C=2** emerges as the most effective classifier, yielding the highest accuracy among all evaluated models. Therefore, it is selected as the optimal classifier for the task at hand

8 Results

- To commence the classification process, execute the '**Classify.py**' file. Ensure that the '**model.object**' and '**count _vectorizer.object**' files are copied from the zip archive to the same directory as the code file prior to execution. The output will be saved to a file named '**output.txt**' and displayed on the console.