Report assignment 3

Battu Sushanth Sri Chandra cs21b018

May 2024

Introduction

Three algorithms have been implemented for spam classification:
1.SVM
2.Perceptron
3.Naive Baye's
The training data was sourced from Hugging face.co website.In the provided source codes, most variable,function names align closely with their respective algorithmic roles and meanings.The code is written in a way that makes it easy to understand, and it's further explained below along with the results obtained from its execution.

1 Support Vector Machine (SVM):

Extracting Data:

The "spam-detection-dataset" from the Hugging face.co website, dataset library was utilized, consisting of labeled email messages categorized as "spam" or "ham" (non-spam). The dataset is divided into training and test sets for model training and evaluation, respectively.

Data Preprocessing:

Text and corresponding labels were extracted from the dataset. Text data was vectorized using the TF-IDF (Term Frequency-Inverse Document Frequency) technique, converting textual data into numerical features suitable for machine learning models.

Model Training:

SVM classifiers were trained with both linear and RBF kernels. A range of regularization parameter values (C) was explored to find the optimal hyperparameters for each kernel.

Evaluation:

The trained models were evaluated on the test dataset to measure their performance. Accuracy scores were computed to assess the effectiveness of each model in classifying spam and non-spam emails.

Results:

```
C = 0.1 Accuracy (Linear Kernel): 99.26605504587155
C = 0.1 Accuracy (RBF Kernel): 98.64220183486239
C = 1.0 Accuracy (Linear Kernel): 99.74311926605505
C = 1.0 Accuracy (RBF Kernel): 99.63302752293578
C = 10.0 Accuracy (Linear Kernel): 99.74311926605505
C = 10.0 Accuracy (RBF Kernel): 99.6697247706422
C = 100.0 Accuracy (Linear Kernel): 99.4862385321101
C = 100.0 Accuracy (RBF Kernel): 99.6697247706422
```

Figure 1: Accuracies for various regulisation parameters for linear and rbf kernels

Linear Kernel:
Accuracy scores ranged from 99.26 percent to 99.74 percent, with the highest accuracy achieved at C = 1.0.
RBF Kernel:
Accuracy scores varied between 98.64 percent and 99.66 percent, peaking at C = 10.0

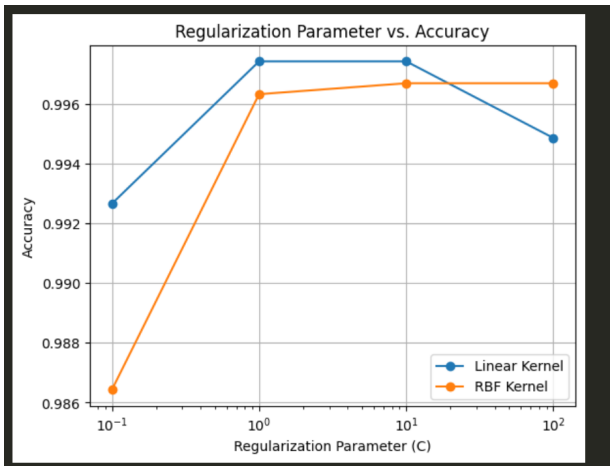

Figure 2: Regulisation parameter vs Accuracy for linear and rbf kernels

Performance Comparison:

1. The linear kernel exhibited consistent performance across different C values, while the RBF kernel's accuracy varied more significantly.

2. Both kernels achieved competitive accuracy in spam detection, with the linear kernel slightly outperforming the RBF kernel at optimal C values.

Generalization and Scalability:

1. The trained models demonstrated the ability to generalize well to unseen email data, indicating their potential for real-world application.

2. Further scalability tests could assess the models' performance with larger datasets to evaluate their robustness and efficiency.

Testing on folder of emails:

As the accuracy for linear kernel with reguliser as 1 is highest,so svm classifier with linear kernel,reguliser=1 is used.

predict_emails_from_folder() predicts the labels (spam or non-spam) for email files within a specified folder.It takes the folder path, pre-trained SVM classifier, and TF-IDF vectorizer as input.The function iterates through each file in the folder, checking for email files.Email content is read, converted to lowercase, and vectorized using TF-IDF.The pre-trained SVM classifier(SVM with reguliser=1 and kernel=linear) predicts the label for each email based on its vectorized representation. It returns a list of predicted labels for all email files in the folder.

2 Perceptron

Extracting data:

1. Dataset Loading: It seems to be using a function called load_dataset() to load a dataset related to spam detection. The dataset is loaded from a source named Hugging face.co website.

2. Initialization: It initializes variables n_emails to count the number of emails processed and an empty list emails to store the email texts along with their labels.

3. Iteration through Dataset: It iterates through the training split of the loaded dataset. For each item in the dataset, it assumes there are features named 'text' and 'label'.

4. Extracting Text and Label: It extracts the text and label from each item in the dataset. If the label is 'spam', it assigns the label -1 to the email (indicating it's spam). Otherwise, it assigns the label 1 (indicating it's not spam).

5. Building Email List: It constructs a list named emails where each item is a list containing the text of an email as the first element and its corresponding label as the second element. This list will eventually contain all the emails and their labels.

6. Counting Emails: It increments the count of processed emails (n_emails) with each iteration.

Creating Vocublary:

This function, create_vocabulary, is designed to generate a vocabulary from a collection of emails.
The function takes a list of emails as its parameter.It initializes an empty defaultdict named vocabulary. defaultdict is a subclass of the built-in dict class in Python that provides a default value for each key. In this case, the default value is set to zero. Processing: It iterates through each email in the input list. For each email, it splits the text into individual words using the split() method. It iterates through each word in the email and increments the count of that word in the vocabulary dictionary. If the word is encountered for the first time, its count is

initialized to 1. Finally, it returns the vocabulary dictionary, which contains unique words from all the emails along with their respective counts.

Converting emails to binary:

The convert_to_binary function transforms a list of emails and their labels into a binary representation based on a provided vocabulary. Each email is converted into a binary vector indicating the presence or absence of words from the vocabulary. The resulting binary vectors, along with the original labels, are returned as a list of tuples.
The algorithm first initializes an empty list to store the binary representations of the emails. Then, it iterates through each email in the input list along with its label. For each email, it splits the text into individual words and computes a set of unique words present in that email. Next, it iterates through the words in the provided vocabulary. For each word, it checks whether it is present in the set of unique words from the email. If the word is present, it assigns a value of 1 to indicate its presence in the binary representation; otherwise, it assigns 0 to indicate its absence. This process generates a binary vector representing the presence or absence of words from the vocabulary in the email. Finally, the algorithm stores this binary representation along with the original label in the output list.

Perceptron algorithm:

1. Initialization: The function initializes the weight vector omega with zeros. It also initializes old_omega with ones, itr to count the number of iterations, and accepts an optional parameter epochs to define the maximum number of training iterations.

2. Training Loop: It iterates through the training data for a specified number of epochs or until convergence is achieved. Convergence is determined when there is no change in the weight vector between consecutive iterations.

3. Activation Calculation: For each training example, it calculates the activation by taking the dot product of the weight vector and the input features.

4. Prediction: It makes a prediction based on the sign of the activation. If the activation is greater than or equal to zero, it predicts class 1; otherwise, it predicts class -1.

5. Weight Update: If the prediction does not match the true label y, it updates the weight vector omega by adding the product of the true label y and the input features x. This update is designed to move the decision boundary closer to correctly classify the misclassified example.

6. Convergence Check: It checks for convergence by comparing the current weight vector with the previous one. If there is no change, it indicates convergence and terminates the training loop.

7. Return: The function returns the learned weight vector omega, which represents the parameters of the trained perceptron model.

perceptron_predict:

This function is just made to test the algorithms with data from HuggingFace.com.

1. Prediction Calculation: It calculates the predictions for the test data X_test using the learned weight vector omega. It computes the dot product of each test example with the weight vector to obtain the raw predictions.

2. Sign Function: It applies the sign function to the raw predictions to determine the class labels. If the raw prediction is positive, it assigns a label of 1; otherwise, it assigns a label of -1.

3. Handling Zero Predictions: It iterates through the predictions and checks for any instances where the prediction is 0. If a prediction is 0, it sets the prediction to 1. This step ensures that all predictions are either 1 or -1, which is essential for consistent interpretation in classification tasks.

4. Return: Finally, it returns the array of predicted class labels for the test data. These predictions can be compared with the true labels to evaluate the performance of the perceptron model.

Testing on Hugging face test data:

Test emails and their labels are extracted from HuggingFace.com, following a process similar to the one described above. These test emails are then fed as input to the perceptron_predict function, and the predictions are recorded.
The accuracy is computed as follows: A binary array is generated, where each element is set to 1 if the prediction matches the corresponding test label, and 0 otherwise. The mean of this array is then calculated, representing the proportion of correct predictions. This proportion is multiplied by 100 to obtain the accuracy score, which indicates the percentage of correctly classified instances in the test data.
The accuracy recorded for hugging face test data is 99.63302752293578

Testing on folder of emails:

The read_emails_from_folder function facilitates the prediction of spam or non-spam labels for emails stored as text files within a specified folder. It iterates through each file in the folder. Upon discovering such files, it reads their content, converting it to lowercase for consistency. Next, it encodes the email text into a binary representation based on the presence of words from a given vocabulary. Using a pre-trained perceptron model represented by the weight vector omega, the function then makes predictions for each encoded email. Predictions are appended to a list, with binary labels (0 for non-spam and 1 for spam), and this list is returned. Overall, this function streamlines the process of evaluating emails for spam detection based on a pre-existing perceptron model and vocabulary.

3 Naive Baye's

Extracting data:

Email data was extracted in a similar manner to the perceptron algorithm. Two variables, n_spam and n_non_spam, were initialized and incremented whenever a spam or non-spam email was encountered, respectively, during the data extraction process.

probabilities of spam and non_spam emails are calculated as (n_spam/total_no_of_emails) and (n_non_spam/total_no_of_emails)

Creating vocublary:

Vocabulary creation was conducted similarly to the perceptron algorithm. The only distinction lies in the creation of the vocabulary. In this process, dictionaries named spam_vocab and non_spam_vocab were initialized. They store the number of spam and non-spam emails containing a particular word, respectively. The function returns the vocabulary, spam_vocab, and non_spam_vocab.

predicting probabilities:

calculate_word_probabilities() computes the conditional probabilities of each word given spam and non-spam classes. It takes three dictionaries as input: vocabulary (containing all unique words), spam_vocab (word frequencies in spam emails), and non_spam_vocab (word frequencies in non-spam emails).The function applies Laplace smoothing to handle unseen words by adding 1 to the numerator and adjusting the denominator accordingly.It returns a dictionary containing word probabilities for each word in the vocabulary, with entries for both spam and non-spam classes.

predicting email label:

predict_email_label() predicts the label (spam or non-spam) for a given email text based on word probabilities and class priors. Input Parameters: It takes the email text, word probabilities dictionary, and probabilities of spam (p_spam) and non-spam (p_non_spam) classes. The function splits the email text into individual words and calculates the logarithm of probabilities for each class.For each word in the email text, it retrieves the corresponding word probabilities from the word_probabilities dictionary and updates the logarithmic probabilities for spam and non-spam classes accordingly.The function compares the logarithmic probabilities for spam and non-spam classes and returns the label with the higher probability, represented as an integer (1 for spam, 0 for non-spam).

Testing data from hugging face:

Test emails were extracted from hugging face.com .Now the test emails were iterated and each email was given as input to predict probabilities function.Using the prediction array similarly from above ,accuracy is calculated.
Accuracy for hugging face tes data is 99.04587155963303 percent

Testing on folder of emails:

This function operates similarly to the perceptron algorithm described earlier.

4 Conclusion:

In conclusion, when comparing the accuracy of Support Vector Machines (SVM), perceptron, and Naive Bayes algorithms for the task at hand, SVM demonstrates the highest accuracy, followed by perceptron and Naive Bayes. SVMs typically offer superior performance in classification tasks, leveraging the capability to find optimal hyperplane boundaries in high-dimensional spaces. Perceptron, although less complex, also performs reasonably well but may not handle complex data distributions as effectively as SVMs. Naive Bayes, while simple and computationally efficient, often assumes independence between features and may struggle with correlated or non-linear relationships in the data, resulting in lower accuracy compared to SVMs and perceptron. Therefore, in this scenario, SVM emerges as the most accurate classifier, followed by perceptron and Naive Bayes.