CS5691: Pattern Recognition and Machine Learning Assignment-3 - SPAM or HAM Report

Name: Dasari Himakar Sai
Roll No: CS21B022
Instructor: Arun Rajkumar

May 5, 2024

Note: To run the test emails, please run the file `svm.py`

Support Vector Machine (SVM) Algorithm

1.1 Introduction:

Support Vector Machine (SVM) is a supervised machine learning algorithm used for classification tasks. In the current context of spam detection, SVM is used to find an optimal hyperplane that separates the *spam* and *non-spam* classes with maximum margin (or with vector w, such that $\|w\|$ is minimum).

1.2 Data Set Selection:

For this task, I have selected the *"spam-detection-dataset"* from the *Hugging Face* website. The dataset is structured into two subsets: the *train* dataset and the *test* dataset. Each subset comprises a list of dictionaries, where each dictionary contains two key-value pairs: `text` and `label`. The `text` key corresponds to the text content of the email, stored as a `string` data type. Meanwhile, the `label` key denotes whether the email is categorized as `spam` or `non-spam`.

1.3 Feature Extraction:

I have utilized TF-IDF (Term Frequency-Inverse Document Frequency) vectorization, which I learnt in the Natural Language Processing course, for feature extraction. Using this the raw textual data is converted into a numerical representation that can used for running the SVM algorithm.

1.3.1 TF-IDF Vectorization:

TF-IDF is a measure that evaluates the importance of a term within a document relative to a corpus of documents. It consists of the following two main components:

- **Term Frequency (TF):** This component measures the frequency of a term (word) within a document. It indicates how often a particular word occurs in a document relative to the total number of words in that document. Words that occur frequently within a document are assumed to be more important for classification.
- **Inverse Document Frequency (IDF):** IDF measure the importance of term across a collection of documents, which is often called a *corpus*. It evaluates how common or rare a word is across all the documents in the dataset. Terms that occur frequently across all the documents are given less weights, and vice-versa.

By combining TF and IDF, TF-IDF assigns weights to each term in a document, reflecting its significance in distinguishing that document from others in the dataset. Terms that are frequent in a particular document but rare in others are given higher weights, indicating their importance in classification.

1.3.2 TF-IDF Vectorization Process:

The TF-IDF vectorization process involves the following steps:

- **Tokenization:** The raw text data (emails) are tokenized, splitting them into individual words. This step also removes unnecessary punctuation.
- **Stopword Removal:** Commonly occurring words that carry little to no significance, known as stopwords (e.g., "the," "is," "and"), are removed from the tokenized text. This helps in focusing on more meaningful terms that contribute to classification.
- **TF-IDF Calculation:** For each term in the tokenized text, the TF-IDF score is calculated based on its frequency within the document (TF) and its rarity across the entire dataset (IDF). This results in a numerical representation of each document, where each term is associated with a TF-IDF weight.
- **Vectorization:** The TF-IDF scores are organized into a matrix format, where each row represents a document (email) and each column represents a unique term in the dataset. This matrix serves as the feature matrix for training the SVM classifier.

The TF-IDF vectorizer was applied separately to both the training and testing datasets to ensure consistency in feature representation across different sets of emails. This ensures that the SVM classifier learns from a standardized feature space and generalizes well to unseen data during testing.

1.4 SVM Model:

Support Vector Machine (SVM) operates by finding the optimal hyperplane that separates different classes with the maximum margin, thereby minimizing classification errors. In the context of spam detection, SVM is applied to distinguish between spam and non-spam emails based on the features extracted from the email content.

The primal problem of the SVM is formulated as follows:

$$\min_{\mathbf{w},\epsilon} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\epsilon_i \quad \text{s.t., } (\mathbf{w}^T\mathbf{x}_i)y_i + \epsilon \geq 1 \text{ and } \epsilon_i \geq 0 \forall i$$

The solution to the dual of this problem can be obtained using Lagrangian duality:

$$\min_{0\leq\alpha\leq C} \boldsymbol{\alpha}^T\mathbf{1} - \frac{1}{2}\boldsymbol{\alpha}\mathbf{Y}^T\mathbf{X}^T\mathbf{X}\mathbf{Y}\boldsymbol{\alpha}$$

The $\mathbf{w}^*$ we obtain as solution to the above dual problem would be:

$$\mathbf{w}^* = \mathbf{X}\mathbf{Y}\boldsymbol{\alpha}$$

1.4.1 Linear Kernel SVM:

The linear kernel SVM assumes that the relationship between features and the target class can be effectively represented by a linear function. It seeks to find a straight line (or hyperplane) that best separates the spam and non-spam emails in the feature space.

The label for spam emails can be $+1$ and non-spam emails can be -1. Let w^* be the optimal sparse linear combination of the data points.

1.4.2 RBF Kernel SVM:

The radial basis function (RBF) kernel SVM is more flexible compared to the linear kernel SVM. It implicitly maps the input data into a higher-dimensional space, where it becomes easier to separate the classes. This allows the algorithm to capture non-linear relationships between features and the target class, making it suitable for tasks where the data is more complex or the decision boundary is non-linear. In the current context of spam detection, where the characteristics of spam emails can vary widely, the RBF kernel SVM may offer better performance by capturing the intricate patterns in the data.

1.4.3 Hyperparameter Tuning:

Hyperparameters play an important role in the SVM algorithm. The regularization parameter (C) was tuned to find the optimal balance between maximizing the margin and minimizing misclassification. Regularization parameters [0.1, 1, 2, 2.5, 3, 10, 100] were tested for both linear and RBF kernels.

1.4.4 Results:

The accuracy of the SVM classifier was evaluated on the test dataset for each combination of hyperparameters. The plot illustrates the relationship between the regularization parameter (C) and the accuracy of the classifier for both linear and RBF kernels.

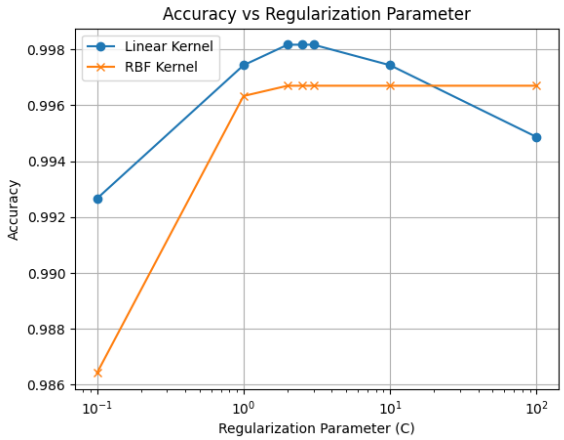


Figure 1: Accuracy vs Regularization Parameter (C) for Linear and RBF Kernels

From the results, it can be observed that the performance varies with different regularization parameters and kernel types. Fine-tuning of hyperparameters is essential to achieve optimal performance. It can be observed that for this particular dataset, the RBF Kernel performs better at higher regularization parameter values (Max accuracy attained: 0.996697247706422) and Linear Kernel performs better when the regularization parameter is of the order 1 (Max accuracy attained: 0.998165137614679).

Perceptron Algorithm

2.1 Introduction:

The Perceptron algorithm is a simple yet powerful linear classifier used for binary classification tasks. It iteratively updates its weights based on misclassifications until convergence or for a specified number of epochs.

2.2 Data Set Selection:

I have used the same *"spam-detection-dataset"* from the *huggingface* website for this task too.

2.3 Preprocessing:

Prior to training the Perceptron, the text data undergoes preprocessing. Each email text is converted to lowercase and split into individual words. The words are then mapped to a binary feature vector, where each element represents the presence or absence of a word in the email. To compute this binary feature, I have used a set to keep track of all the words that I have encountered in all the emails of train dataset. Then mapped each of them to unique indices. For each email, the presence of the word mapped to an index would be represented by a '1' in the feature vector and '0' otherwise.

2.4 Perceptron Training:

The Perceptron is trained using a specified number of epochs (or iterations). During each epoch, the weights of the Perceptron are updated as follows based on misclassifications.

Consider the following case– At the t th epoch, the datapoint $\mathbf{x}_i$ is misclassified. The Perceptron does the following:

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \mathbf{x}_i y_i$$

The training process continues until convergence or until the maximum number of epochs is reached.

2.5 Perceptron Testing:

After training, the Perceptron is tested on the test dataset to evaluate its performance. For each data point in the test dataset, the prediction that is obtained from the Perceptron that is trained above is compared with the true label. The accuracy of the Perceptron is calculated as the ratio of correctly classified emails to the total number of emails in the test dataset.

2.6 Results:

The Perceptron is trained with a varying number of epochs ranging from 1 to 10. For each number of epochs, the accuracy of the Perceptron on the test dataset is recorded. It can be observed that for the dataset I have chosen, convergence is reached at 4th epoch itself. The plot shows the relationship between the number of epochs and the accuracy of the Perceptron. It provides insights into how the accuracy changes as the Perceptron is trained for more epochs. The maximum accuracy reached is 0.9963302752293578.

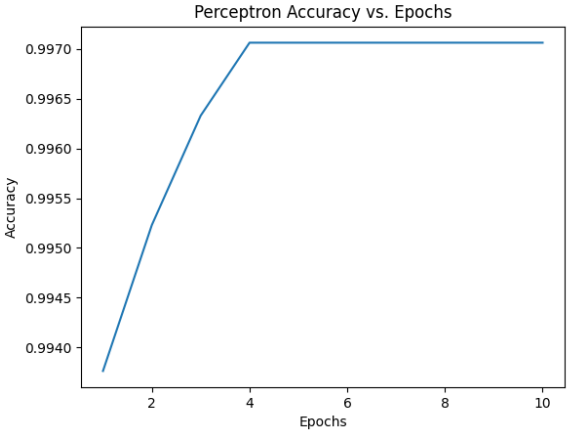


Figure 2: Perceptron Accuracy vs Epochs

Logistic Regression Algorithm

3.1 Introduction:

Logistic Regression is a widely used classification algorithm that models the probability of a binary outcome based on one or more predictor variables. I have implemented this Logistic Regression algorithm for the task of spam detection.

3.2 Data Set Selection & Preprocessing:

I have used the same *"spam-detection-dataset"* from the *huggingface* website for this task too. The preprocessing logic is also exactly the same one used for the implementation of the Perceptron algorithm.

3.3 Logistic Regression:

The Logistic Regression algorithm is used for classification tasks leveraging the concept of probabilities.

For some datapoint $\mathbf{x}$, the probability that it is of label 1 is given by:

$$\Pr(y = 1 \mid \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}$$

The log-likelihood function in terms of $\mathbf{w}$, when given a dataset $\texttt{Data} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_n, y_n)\}$ is given by:

$$\text{log-L}(\mathbf{w}, \texttt{Data}) = \sum_{i=1}^{n} (1 - y_i)(-\mathbf{w}^T\mathbf{x}) - \log(1 + e^{-\mathbf{w}^T\mathbf{x}})$$

We wish to maximize this log-Likelihood function, but there is no closed form solution for this. But we can perform gradient ascent (as we are maximizing) to arrive at the following solution.

$$\begin{aligned}
\nabla\text{log-L}(\mathbf{w}, \texttt{Data}) &= \sum_{i=1}^{n} y_i\mathbf{x}_i - \mathbf{x}_i \left(\frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}\right) \\
&= \sum_{i=1}^{n} \mathbf{x}_i \left(y_i - \left(\frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}}\right)\right)
\end{aligned}$$

We finally obtain the following update rule:

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \nabla \text{log-L}(\mathbf{w}, \texttt{Data})$$
$$\mathbf{w}_{\text{new}} = \sum_{i=1}^{n} \mathbf{x}_i \left(y_i - \left(\frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}} \right) \right)$$

Here η is the learning rate which can be tinkered with and for a datapoint $(\mathbf{x}_i, y_i)$, y_i is its label and the term $\frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}_i}}$ can be thought of as the probability of that this point being given the label 1.

3.4 Logistic Regression Training:

The Logistic Regression model is trained using a specified learning rate and a fixed number of epochs. During each epoch, the weights of the model are updated based on the error between the predicted probability and the actual label. The training process continues until convergence or until the maximum number of epochs is reached.

The logistic function (sigmoid) is used to transform the linear combination of features and weights into a probabilit score between 0 and 1. This sigmoid function ensure that the output of the Logistic Regression model lies within the range of probabilities, making it suitable for binary classification tasks.

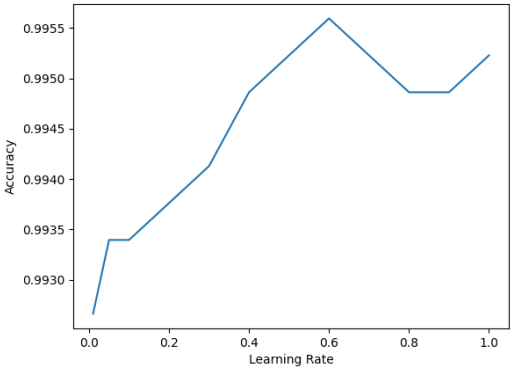
3.5 Logistic Regression Testing:

After training, the Logistic Regression model is tested on the test dataset to evaluate its performance. For each data point in the test dataset, the prediction that is obtained from the Logistic Regression Model that is trained above is compared with the true label. To get the prediction, get the probabilities of both the labels, using the final $\mathbf{w}$, for the test data point. Return the one whose probability is greater than 0.5. The accuracy of the model is calculated as the ratio of correctly classified emails to the total number of emails in the test dataset.

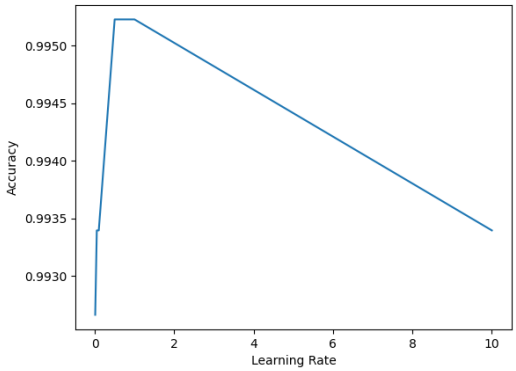
3.6 Results:

The Logistic Regression model is trained using various learning rates ranging from 0.01 to 1. For each learning rate, the model's accuracy on the test dataset is recorded. The plot illustrates the relationship between the learning rate and the accuracy of the Logistic Regression model. It provides insights into how the choice of learning rate affects the model's performance.

I ran the code with 10 epochs over the learning rates: [0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9 1]. Maximum accuracy attained is 0.9952293577981651. It can also be observed that the accuracy decreases at higher learning rates such as $\eta = 10$.



(a) Logistic Regression Accuracy vs Learning Rate



(b) Logistic Regression Accuracy vs Learning Rate (Accuracy decreases at high learning rates)

Figure 3: Comparison of logistic regression accuracy vs. learning rate

Conclusion

In conclusion, Perceptron algorithm, Logistic Regression, and Support Vector Machine (SVM) with TF-IDF vectorization were explored and evaluated for the email spam detection task.

The Perceptron algorithm, a binary linear classification method, demonstrated impressive accuracy of 99.63% in distinguishing between spam and non-spam emails. Its iterative weight updates based on misclassifications contribute to its effectiveness in learning from the data. However, further experimentation with different datasets and hyperparameter tuning could potentially enhance its performance.

Logistic Regression, which predicts the probability of an input belonging to a specific class using the sigmoid function, achieved a peak accuracy of 98.35% at $\eta = 0.6$. This method effectively models the probability of an email being spam or non-spam based on its features. Yet, like the Perceptron, Logistic Regression could benefit from additional exploration of hyperparameters and learning rates to further refine its performance.

SVM with TF-IDF vectorization emerged as the most promising choice for spam classification tasks. Leveraging appropriate feature extraction techniques and hyperparameter tuning, SVM effectively distinguishes between spam and non-spam emails. The SVM model achieved peak accuracies of 99.66% and 98.81% for RBF and linear kernels, respectively, at a regularization parameters value of 2 for Linear kernel and 100 for RBF kernel, among the values that I have considered. Its robust performance, especially in handling both linear and non-linear classification challenges, makes SVM particularly suitable for complex datasets where precision and reliability are paramount.

In conclusion, while each algorithm showed promise in spam detection, SVM stood out as the most effective choice for our data analysis needs. Its superior performance metrics, especially its high accuracy rates, demonstrate its robustness and efficacy in combating email spam. Further experimentation and tuning strategies could unlock even greater potential, solidifying SVM's position as a valuable tool in email spam detection and classification.