CS 5691 : Pattern Recognition and Machine Learning

Assignment 3

Mail Classifier models

Report

Santosh S R
ME20B157

Problem Statement :

Build a spam classifier from scratch. No training data will be provided. You are free to use whatever training data that is publicly available/does not have any copyright restrictions (You can build your own training data as well if you think that is useful). You are free to extract features as you think will be appropriate for this problem. The final code you submit should have a function/procedure which when invoked will be able to automatically read a set of emails from a folder titled test in the current directory. Each file in this folder will be a test email and will be named 'email.txt' ('email1.txt', 'email2.txt', etc). For each of these emails, the classifier should predict +1 (spam) or 0 (non Spam). You are free to use whichever algorithm learnt in the course to build a classifier (or even use more than one). The algorithms (except SVM) need to be coded from scratch. Your report should clearly detail information relating to the data-set chosen, the features extracted and the exact algorithm/procedure used for training including hyperparameter tuning/kernel selection if any. The performance of the algorithm will be based on the accuracy of the test set.

Data Set :

Data set downloaded from VENKATESH GARNEPUDI 's upload via KAGGLE at :
https://www.kaggle.com/datasets/venky73/spam-mails-dataset

The data set is composed of 5171 mails composed into index , spam or ham labels and a label_num that numerically tells whether it's spam or not with 1 or 0.

Data Preprocessing and split:

Data was cleaned to include only the required columns and some of the general high correlation - Low significance characters/ words [ie. symbols/ connector words/ numbers] were removed.

Cleaned Data to final Dataset for Classification:
A dictionary of all words was created and the Features function was used to index the occurrences of its words with respect to the dictionary. This group of binary marks is the processed dataset that would be used as the input after splitting.The train and test data were split in an 80:20 fashion.

Methods used:

1. Naive Bayes Classifier :

Initially all train data are read to estimate the probability of each feature value for each class and probability of each class based on individual feature probabilities are stored as a set of probabilities that form the trained framework.

Test is performed by checking each test data point's probability and classified accordingly with the highest overall probability.

Bayes Equation:

$$\mathrm{P}(\mathrm{C}_i/X) = \frac{P(X|C_i)\cdot P(C_i)}{P(X)}$$

Adapting this to individual probabilities of words w.r.t given train data

$$\mathrm{P(X\ /\ C}_i) = \prod_{j=1}^{d} P(x_j|C_i)$$

2. Logistic Regression Classifier :

The train data are initiated with a weight parameter that will classify the datapoint. This weight is updated using the update rule in each iteration according to the probability given by the link function [here Sigmoid function].
This weight is then used to classify the test data.

Sigmoid function:

$$\sigma(z) = \frac{1}{1+\exp(-z)}$$

Hypothesis function:

$$\mathrm{h(x)} = (\mathrm{w}^T x + b)$$

Update Rule for weight in each Iteration:

$$\mathrm{w}_{j+1} \leftarrow w_j - \eta \cdot \frac{\partial L(\mathbf{w},b)}{\partial w_j}$$

3. Support Vector Machine :

Start with a guess (w : hyperplane). Then the algorithm identifies critical points (support vectors) on wrong sides. Each iteration adjusts hyperplanes to maximize margin using these points. Repeats until the margin is good or stops after a set number of tries.

This iterative focus on support vectors helps SVM progressively create the best separation boundary.

Initial Condition:

$$\begin{aligned} \underset{\mathbf{w},\, b}{\text{minimize}} \quad & \|\mathbf{w}\|_2^2 \\ \text{subject to} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i - b) \geq 1 \quad \forall i \in \{1, \ldots, n\} \end{aligned}$$

Iteration equations:

$$w = \sum_{i=1}^{n} \alpha_i y_i X_i$$

$$\underset{\alpha}{maximize} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \left(X_i^T . X_j\right)$$

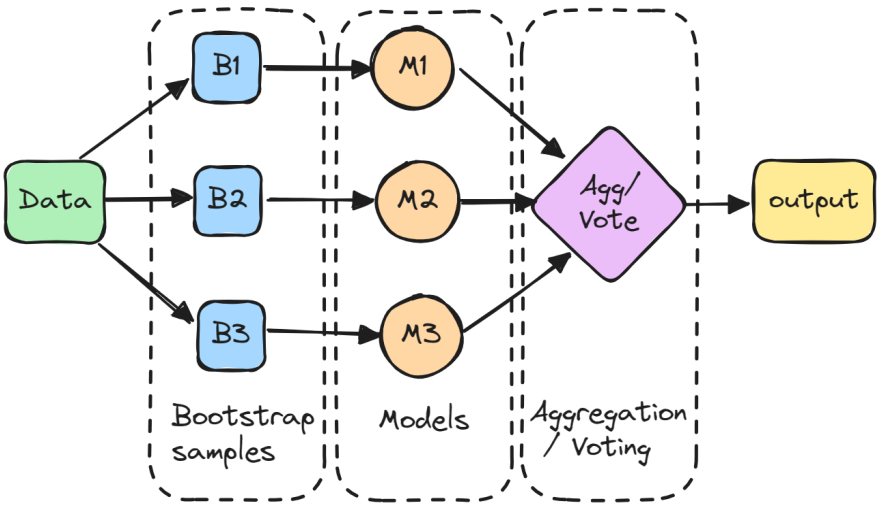
4. Ensemble models :

Ensemble methods like bagging and boosting combine multiple weak learners to create a stronger learner, improving prediction accuracy and reducing variance/bias.

Bagging [*Bootstrap Aggregating*]:

Bagging leverages diversity through iterations: It draws bootstrap samples (X_i) of size m (m <= original data size) from the training data D with replacement. For each bootstrap sample, a base learner (h_i) is trained independently. During prediction for a new data point (x), the ensemble prediction (y_pred) is obtained by aggregating the predictions (h_i(x)) from all learners, often using a majority vote for classification tasks. This approach, by combining predictions from diverse models trained on slightly different data subsets, reduces variance and improves the overall robustness of the ensemble classifier.

PC: https://www.datacamp.com/



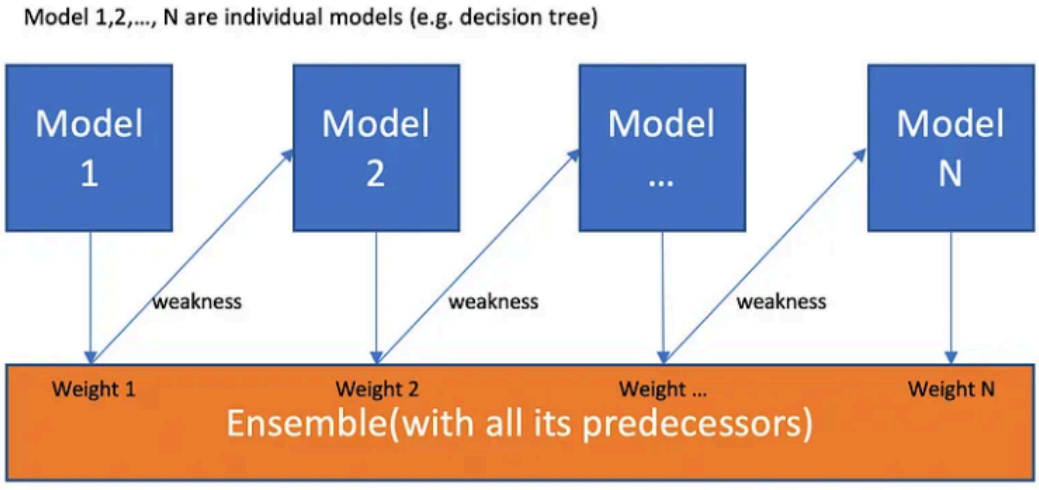
Boosting :

Boosting, unlike bagging, iteratively refines a sequence of weak learners. Here's how it works:

1. Start with equal weights for all data points.
2. Train a weak learner: Fit a learner to the data, but focus on points the previous learner struggled with (higher weights for misclassified points).
3. Update weights: Increase weights for misclassified points and decrease weights for correctly classified ones.
4. Repeat steps 2 and 3 for a set number of iterations: Each new learner focuses on the "hard" examples left by the previous learner, progressively improving the ensemble.
5. Combine predictions: Predictions from all learners are combined (e.g., weighted sum) to form the final prediction.

By iteratively focusing on challenging examples and adjusting weights, boosting creates a powerful ensemble that excels at handling complex data patterns.

PC: https://towardsdatascience.com/



Results:

The Test results of Each algorithm is given below:

Naive Bayes: ***93.3333%***

Logistic Regression: ***96.4251%***

SVM: ***97.9757%***

Boosting: ***95.8454%***

Bagging: ***97.5845%***

Observation:

Of all the algorithms used, SVM gave the best results. even though Naive Bayes assumes feature independence, we can see that it gives a test score of 93.3%. Logistic regression gave 96.42%. The Sci_Kit Learn Library initiated algorithm SVM and Ensemble models gave very good results themselves. We can observe that bagging has given good results, so the data set is on the simpler side with low variance.

The high accuracy across various algorithms suggests that the email data likely exhibits well-defined patterns that are effectively captured by different classification techniques. This could be due to the presence of clear features that distinguish spam from non-spam emails, such as: