Foundation of Machine learning Assignment 2

Gokulakrishnan B - DA24M007

About

In this assignment, I have chosen a dataset from kaggle that contains spam and ham messages. Then I used TFIDF vectoriser from sklearn to convert text messages into numerical embeddings. The I wrote classical machine learning algorithms like K Nearest Neighbour, Gaussian Naive Bayes from scratch (only importing numpy). I imported Support Vector Classifier from sklearn. Then I performed bagging operation on these 3 models by choosing the label with maximum vote. Now we can use predict the label (spam/ ham) of any text message by passing it as input to out bagged model.

Dataset Description

The data used for the training the models can be found here: https://www.kaggle.com/datasets/ashfakyeafi/spam-email-classification . It consist of 2 rows, category (spam/ ham) and the messages (text). It has 5572 rows. 4825 of them belong to class ham. 747 belong to class spam. The category column is modified to 0 for ham and 1 for spam. And text messages are encoded into numbers using TFIDF vectoriser. Now the data is split into train and validation to test the performance of the models.

Feature Extraction

Here, I used TFIDF vectoriser to extract the features from the text data. It transforms text data into numerical values by weighing terms based on their frequency within a document and across all documents.

The formula is

$$\text{TF-IDF}(t, d) = \text{TF}(t, d) \times \log\left(\frac{N}{\text{DF}(t)}\right)$$

Where,

TF(t,d) - term frequency of t in d,
N - total number of documents/ words
DF(t) - Document frequency

Implementation of ML algorithms

K Nearest Neighbour

Let K nearest neighbours of the test point vote their labels. We choose the label with maximum vote. No parameters are updated in the training phase. Just the training data gets stored. This algorithm is implemented as follows:

1. When fit method is called with the x and y, just store these. Don't perform any computation.

```
def fit(self,X,y):
            self.X_train = X
            self.y_train = y
```

2. During prediction, when a test point is given, first compute the distances of this test point with the training samples.

```
distances = np.linalg.norm(self.X_train - x , axis =1)
```

3. Select the k nearest points to the test point.

```
top_k_idx = np.argsort(distances)[:self.k]
```

4. Perform voting and choose the label with maximum vote.

```
cnt_0 = 0
        cnt_1 = 0

        for k in top_k_idx:
            if self.y_train[k] == 1:
                cnt_1 += 1
            else:
                cnt_0 += 1

        if cnt_0 > cnt_1:
            return 0
        return 1
```

The classification report on the validation data using KNN is as follows

	precision	recall	f1-score	support
0	0.94	0.99	0.97	1448
1	0.89	0.62	0.73	224
accuracy			0.94	1672
macro avg	0.92	0.80	0.85	1672
weighted avg	0.94	0.94	0.93	1672

Gaussian Naive Bayes

In fit method, we just learn the mean, variance and prior distribution of each label in the dataset. In the prediction phase, we compute log probability of each class and choose the one with highest log probability. It is implemented as follows:

1. In fit method, the below code snippet learns the mean, variance and prior distribution of each class.

```
for cls in self.classes:
        X_cls = X[y == cls]
        self.mean[cls] = np.mean(X_cls, axis=0)
        self.variance[cls] = np.var(X_cls, axis=0)
        self.prior[cls] = np.log(X_cls.shape[0] / X.shape[0])
```

2. Define the function that outputs the log probability of a feature x, given mean and variance using gaussian distribution formula.

$$P(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

On taking log on both sides, we get

$$\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{(x-\mu)^2}{2\sigma^2}$$

```
exponent = - ((x - mean) ** 2) / (2 * var)
return np.log(1 / np.sqrt(2 * np.pi * var)) + exponent
```

3. In prediction, compute the log gaussian probability of a given test point using above formula with both the means and variance of the 2 classes. Choose the class whose mean and variance result in highest sum of prior and log normal probability.

```
likelihood = self.gaussian_probability(x, self.mean[cls],
self.variance[cls])

class_probabilities[cls] = self.prior[cls] +
np.sum(likelihood)
```

The classification report on the validation data using Gaussian Naive Bayes is as follows

	precision	recall	f1-score	support
0	0.98	0.75	0.85	1448
1	0.36	0.92	0.52	224
accuracy			0.77	1672
macro avg	0.67	0.84	0.68	1672
weighted avg	0.90	0.77	0.80	1672

Support Vector Classifier

We imported the SVC class from the sklearn library. In fit method, we find the optimal hyperplane that maximised the margin between classes in the training data. In the predict method, for a given test point, we determine its position relative to the hyperplane to classify it to one of the two classes.

The classification report on the validation data using Support Vector Classifier is as follows

	precision	recall	f1-score	support
0	0.98	1.00	0.99	1448
1	1.00	0.90	0.94	224
accuracy			0.99	1672
macro avg	0.99	0.95	0.97	1672
weighted avg	0.99	0.99	0.99	1672

Bagged Model

We pass each test point to all the 3 mentioned model and choose the label with majority vote. It is implemented as below

```
def bagged_model(models,X):
    predictions=[]
    for i in X:
        pred = []
        for model in models:
            pred.append(model.predict([i]))
        if sum(pred) / len(models)>0.5:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

The classification report on the validation data using our bagged model is as follows

	precision	recall	f1-score	support
0	0.98	0.99	0.99	1448
1	0.96	0.88	0.92	224
accuracy			0.98	1672
macro avg	0.97	0.94	0.95	1672
weighted avg	0.98	0.98	0.98	1672

How to run the model

- Validation result - Run the *test_reports.py* file
  - Testing with own data - **from the *main.py* file, run *assignment2()* function** that returns the predictions as a list. Make sure you have .txt files inside the *test* folder in the same directory as *main.py* file.

Note: