Jupyter Notebook: https://github.com/ben12385/DS4400-HW3

Readme: In the Github

**Problem 1 [Random Forest classifier]**

Used spambase dataset and converted it to 0 and 1 except for the last 3 value.

(a) Use an existing package to train a Random Forest classifier on the training set. Report accuracy, error, precision, and recall on both training and testing sets.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 10 | | 50 | | 100 | |
| Train | Test | Train | Test | Train | Test |
| Accuracy | 0.9974 | 0.9487 | 0.9994 | 0.9582 | 0.9994 | 0.9600 |
| Error | 0.0026 | 0.0513 | 0.0006 | 0.0418 | 0.0006 | 0.0400 |
| Precision | 0.9971 | 0.9417 | 1.0000 | 0.9451 | 1.0000 | 0.9492 |
| Recall | 0.9963 | 0.9272 | 10.9985 | 0.9492 | 0.9985 | 0.9492 |

(b) Implement your own Random Forest algorithm. The Random Forest training procedure takes as input the training dataset, the number of trees, and the number of features m ≤ d considered at every split (d is the total number of features in the dataset).

I faced quite a number of issues with the implementation. First there was an issue with how to deal with continuous value which are the last 3 so for my implementation I ignored the last 3. Even after ignoring the last 3 I was unable to obtain the same accuracy as the package with my implementation, after a few days of debugging I am still unable to figure out why. Therefore for the c, I will be using the package to vary the features.

(c) Vary the number of features m selected at random at each split. Consider m = d, m = d/2, and m = √ d. Report accuracy, error, precision, and recall on the training and testing set.

Used 50 trees

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | M=d | | M=d/2 | | M=sqrt(d) | |
| Train | Test | Train | Test | Train | Test |
| Accuracy | 0.9994 | 0.9434 | 0.9991 | 0.9504 | 0.9991 | 0.9539 |
| Error | 0.0006 | 0.0566 | 0.0009 | 0.0496 | 0.0009 | 0.0461 |
| Precision | 0.9993 | 0.9330 | 0.9985 | 0.9342 | 1.0000 | 0.9425 |
| Recall | 0.9993 | 0.9227 | 0.9993 | 0.9404 | 0.9978 | 0.9404 |

(d) Fix the number of features m = √ d. Compare your implementation with the package results for different number of trees (10, 50, and 100).

Use test set for comparison

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 10 | | 50 | | 100 | |
| Self | Package | Self | Package | Self | Package |
| Accuracy | 0.6084 | 0.9487 | 0.6040 | 0.9582 | 0.6066 | 0.9600 |
| Error | 0.3916 | 0.0513 | 0.3960 | 0.0418 | 0.3934 | 0.0400 |
| Precision | 0.5429 | 0.9417 | 0.4688 | 0.9451 | 0.5152 | 0.9492 |
| Recall | 0.0419 | 0.9272 | 0.0331 | 0.9492 | 0.0375 | 0.9492 |

My implementation has a lot of issues causing the difference to be so large. For some reason the recall is exceptionally low which means that my implementation is classifying a lot of positives to be negatives.

**Problem 2 [AdaBoost classifier]**

(a) Use an existing package to train an AdaBoost algorithm with 50 base classifiers. Use a decision tree as the base classification model. Report accuracy, error, precision, and recall on both training and testing sets.

|  |  |  |
| --- | --- | --- |
|  | Training Set | Testing Set |
| Accuracy | 0.95 | 0.94 |
| Error | 0.05 | 0.06 |
| Precision | 0.94 | 0.94 |
| Recall | 0.93 | 0.90 |

(b) Change the base classifier to logistic regression, but keep the number of base learners at 50. Report accuracy, error, precision, and recall on both training and testing set.

|  |  |  |
| --- | --- | --- |
|  | Training Set | Testing Set |
| Accuracy | 0.92 | 0.92 |
| Error | 0.08 | 0.08 |
| Precision | 0.92 | 0.92 |
| Recall | 0.88 | 0.86 |

(c) Compare the performance of the AdaBoost classifier with different number of base learners (10, 50, and 100).

|  |  |  |  |
| --- | --- | --- | --- |
|  | Estimator=10 | Estimator=50 | Estimator=100 |
| Accuracy | 0.9121 | 0.9382 | 0.9399 |
| Error | 0.0879 | 0.0618 | 0.0601 |
| Precision | 0.9093 | 0.9381 | 0.9364 |
| Recall | 0.8631 | 0.9029 | 0.9095 |

(d) Compare AdaBoost with Random Forest for the same number of base learners (consider 10, 50, and 100).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 10 | | 50 | | 100 | |
|  | Adaboost | Random Forest | Adaboost | Random Forest | Adaboost | Random Forest |
| Accuracy | 0.9121 | 0.9339 | 0.9382 | 0.9434 | 0.9399 | 0.9426 |
| Error | 0.0879 | 0.0661 | 0.0618 | 0.0566 | 0.0601 | 0.0574 |
| Precision | 0.9093 | 0.9415 | 0.9381 | 0.9369 | 0.9364 | 0.9388 |
| Recall | 0.8631 | 0.8874 | 0.9029 | 0.9183 | 0.9095 | 0.9139 |

**Problem 3 [Neural Networks]**

(a) Pick 3 configurations of Feed-Forward Neural Networks and describe for each: (1) number of layers; (2) number of hidden units on each layer; (3) activation functions.

First configuration has 2 layers with the first layer having 392 hidden units using Relu. The second layer having 28 hidden units using Relu.

Second configuration has 2 layers with the first layer having 392 hidden units using Sigmoid. The second layer having 28 hidden units using Sigmoid.

Third configuration has 3 layers with the first layer having 96 hidden units using Relu. The second layer having 48 hidden units using Relu. The third layer having 28 hidden units using Relu.

(b) Train models for all 3 architectures. Report performance metrics (loss function, accuracy, and error) on both training and testing data.

All configuration used binary cross entropy as the loss function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Data | | Testing Data | |
| Config | Accuracy | Error | Accuracy | Error |
| 1 | 0.9186 | 0.0814 | 0.9193 | 0.0807 |
| 2 | 0.9864 | 0.0136 | 0.9868 | 0.0132 |
| 3 | 0.9940 | 0.0060 | 0.9922 | 0.0078 |

(c) Pick 3 configurations of Convolutional Neural Networks and describe for each: (1) number of layers; (2) layer type (convolution, max pooling, fully connected) (3) filter size for convolution and max pooling layer; (4) number of hidden units on each layer; (5) activation functions.

First configuration has 2 convolution layer followed by a max pooling layer and then a fully connected layer. The first convolution layer uses 8 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The second convolution layer uses 16 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The max pooling uses 2 by 2 filters with a stride of 2. The fully connected layer has 50 hidden nodes and uses relu as its activation function.

Second configuration has 2 convolution layer followed by a max pooling layer and then 2 convolution layer followed by a max pooling layer again and then a fully connected layer. The first convolution layer uses 4 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The second convolution layer uses 8 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The max pooling uses 2 by 2 filters with a stride of 2. The third convolution layer uses 16 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The fourth convolution layer uses 32 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The max pooling uses 2 by 2 filters with a stride of 2. The fully connected layer has 50 hidden nodes and uses relu as its activation function.

Third configuration has 2 convolution layer followed by a max pooling layer and then 2 convolution layer followed by a max pooling layer again and then a fully connected layer. The first convolution layer uses 4 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The second convolution layer uses 8 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The max pooling uses 2 by 2 filters with a stride of 2. The third convolution layer uses 16 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The fourth convolution layer uses 32 filters of 5 by 5 with a stride of 1 and uses Relu as its activation function. The max pooling uses 2 by 2 filters with a stride of 2. The fully connected layer has 100 hidden nodes and uses relu as its activation function.

(d) Train models for all 3 architectures. Report performance metrics (loss function, accuracy, error) on both training and testing data. (e) Compare performance of Feed-Forward and Convolutional Neural Networks for this classification task.

All configuration used binary cross entropy as the loss function

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Training Data | | Testing Data | |
| Config | Accuracy | Error | Accuracy | Error |
| 1 | 0.9986 | 0.0014 | 0.9968 | 0.0032 |
| 2 | 0.9979 | 0.0021 | 0.9970 | 0.0030 |
| 3 | 0.9971 | 0.0029 | 0.9965 | 0.0035 |

The convolution neural network performs better than feed forward network and uses less nodes which decreasing the amount of computation power required.