EMRAH SARIBOZ

HW-4 Report

Accuracy and run-time analysis

Model	Dataset	Run-time	Accuracy(Train)	Accuracy(Test)
Random Forest	Digit	0.13 seconds	1.00	0.97
Random Forest	Mammographic masses	0.06 seconds	0.94	0.77
Bagging	Digit	1.08 seconds	1.00	0.95
Bagging	Mammographic masses	0.77 seconds	0.94	0.77
AdaBoost	Digit	0.02 seconds	1	0.85
AdaBoost	Mammographic masses	0.21 seconds	0.94	0.74

- **Data Preprocessing**
  - I wrote dataPreprocessing.py to pre-process the dataset. I added the column names and imputed missing values with median of the corresponding column.

Analysis

- As we can understand from the above table, all the models are suffered from overfitting issue except bagging and random forest on Digit dataset.

Random Forest

- **Parameter Tuning**
  - n**_estimator**
    - I changed some parameter to see the accuracy change over the dataset. One of the parameters it takes is n_estimator which is the number of trees in the forest it will generate. The reason I want to do this is reduce the possibility of the overfitting issue. I used 50 to receive above results.
    - With increasing this number to 100, I didn't receive any improvement over the final accuracy. I tried 150 and still, there wasn't any improvement on both datasets.
  - **Max Depth**
    - If nothing is specified, it will not stop until all nodes are expanded. This technique used in pre-tuning to overcome the overfitting issue.
    - Changing this value to 6 reduced the accuracy on training to 0.98 and test went down to 0.95.
    - On Mammographic masses dataset, accuracy of the training reduced to 0.87 while accuracy on test increased to 0.83 which is an improvement.
  - **Criteria**

- Two criteria used in Random Forest are Gini index and Entropy. I got all the above result using the Gini index. Although this criterion didn't make any improvement over the accuracy, it doubled the run time on both datasets as expected. It requires more mathematical operations compare to Gini.

Bagging

I received the results of table using Decision tree as base classifier. Decision tree has great possibility suffer from overfitting compare to other algorithms because they can create complex trees and thus may not generalize on unseen data well. Thus, I started analyzing the decision tree to get better result on Bagging.

- **max_depth**
  - Setting this value to 5 increased the accuracy of test on MM dataset by 10% (to 0.86) while it reduced the accuracy of training to 0.83. This value determines when to stop expanding the leaves. Thus, it is used in pruning which is a technique to reduce the affect of overfitting.
- **Base estimator**
  - I tried to Support Vector Machine as base estimator; however, the results are terrible (kernel: RBF).
  - On digits dataset, I received 1.00 accuracy on training while 0.28 on test dataset. The total run-time of the program was 29 seconds! Changing the kernel to linear fixed the issue and I received 1.00 and 0.96 on training and testing dataset respectively.
  - Using linear SVM, I got 0.84 training accuracy and 0.85 testing accuracy on MM dataset. It is the best accuracy I got on this dataset so far.
- **N_estimator**
  - Changing this value from 100 to default(10), I got 0.90 for training and 0.80 for testing on MM datasets. Digit dataset gave 0.97 and 0.90 respectively.

AdaBoost

As in Bagging, I used decision tree as base estimator.

- **Max depth**
  - Setting this value to 4 reduced the training accuracy to 0.95 while increasing the accuracy of test dataset to 0.93 on digit dataset.
  - The accuracy of test dataset of MM dataset increased to 0.77 while keeping the training accuracy same which still indicates it suffers from overfitting. Although I am not sure why, MM dataset is not giving good results on decision tree algorithms.
- **Criterion**

    - Changing the criterion from gini index to enthropy did not make any change in both datasets accuracy rather than increasing the run time for both datasets (includes test and training samples)
- **Learning rate**
    - This determines the contribution of each classifier. Default value is 1 and there is a trade-off between n_estimator and learning rate. I reduced the learning rate to 0.1 while increasing the n_estimators to 130. For the digit dataset, I got 1.00 training accuracy and 0.97 testing accuracy. On the other hand, MM dataset gave 0.77 training accuracy and 0.75 testing accuracy.
- **Base Estimator**
    - I changed Decision tree to SVM. My accuracy for the both testing and training on digit dataset didn't change at all. However, SVM helped me to reduce the overfitting issue but reduced both training and testing accuracy to 0.69.

For this assignment, Bagging using linear SVM as a base estimator gave the best outcome for the MM dataset compares to others. Also, I want to note that, no matter what I tried, I couldn't improve the accuracy result for the decision tree (base estimator). On the other hand, the digit dataset gave almost the same for all algorithms.

- Besides all of these, I wanted to compare the classifier with the SVM separately on MM dataset (digit is toy dateset and it is not informing).

- Accuracy of the SVM on test dataset is 0.75 while random forest accuracy on SVM is 0.83.
- Accuracy of the SVM on test dataset is 0.75 while bagging has accuracy of 0.83 on SVM
- Accuracy of the SVM on test dataset is 0.76 while adaboost has accuracy of 0.77.

As we can see from above experiments, ensemble method is doing its job and provides better outcome for each method compare to single SVM.

AdaBoost Algorithm

Index	x	y	weights	$\hat{y}$	Updated weights
1	1.0	1	0.072	1	0.053
2	2.0	1	0.072	1	0.053
3	3.0	1	0.072	1	0.053
4	4.0	-1	0.072	-1	0.053
5	5.0	-1	0.072	-1	0.053
6	6.0	-1	0.072	-1	0.053

7	7.0	1	0.167	1	0.126
8	8.0	1	0.167	-1	0.248
9	9.0	1	0.167	-1	0.248
10	10.0	-1	0.072	-1	0.053

Steps

- 2-c)
  - Compute weighted error rate = w. ($\hat{y} \neq$ y)

Error rate = (0.072, 0.072, 0.072, 0.072, 0.072, 0.072, 0.167, 0.167, 0.167, 0.072) * $\begin{matrix}0\\0\\0\\0\\0\\0\\0\\1\\1\\0\end{matrix}$

Error rate = 0.334

- 2-d)
  - Compute Coefficients
  - $a_j = 0.5 \ln (1- e\_rate / e\_rate) \approx 0.35$
- 2-e)
  - Update weights
  - w = w * exp(-a_j * **$\hat{y}$ * y) for each element**

Index 1 : 0.072 * exp(-0.35 x 1 x 1) = 0.050

Index 2 : 0.072 * exp(-0.35 x 1 x 1) = 0.050

Index 3 : 0.072 * exp(-0.35 x 1 x 1) = 0.050

Index 4 : 0.072 * exp(-0.35 x -1 x -1) =0.050

Index 5 : 0.072 * exp(-0.35 x -1 x -1) = 0.050

Index 6 : 0.072 * exp(-0.35 x -1 x -1) =0.050

Index 7 : 0.072 * exp(-0.35 x 1 x 1) =0.117

Index 8 : 0.072 * exp(-0.35 x 1 x -1) =0.23

Index 9 : 0.072 * exp(-0.35 x 1 x -1) =0.23

Index 10 : 0.072 * exp(-0.35 x -1 x -1) = 0.05

- 2-f) Normalize weights to sum to 1:
- $\mathbf{w = w / \sum_i w_i}$
  ➔ $\mathbf{\sum_i w_i = 7 * 0.050 + 1 * 0.117 + 2 * 0.23 = 0.927}$
- 0.050 / 0.927 ≈ **0.053**
- 0.117 / 0.927 ≈ **0.126**
- 0.23 / 0.927 ≈ **0.248**