Sai Nikhil (04/29/2021) thirandas.s@northeastern.edu

Problem 1. *This exercise will help you give a good understanding of the differences between bagging and boosting, with implementation.*

a) What are the major differences between boosting and bagging?

b) Prove the following upper bound result for training error of AdaBoost.

Assume $\epsilon_t < \frac{1}{2}$ *and let* $\gamma_t = \frac{1}{2} - \epsilon_t$. *Then the following bound holds*

$$\frac{1}{n} \left| \{ i : f(x_i) \neq y_i \} \right| \leq \prod_{t=1}^{T} \sqrt{1 - 4\gamma_t^2} \leq e^{-2 \sum_{t=1}^{T} \gamma_t^2}$$

where, ϵ_t *is the training error of expert* $f_t(x)$.

c) Implement a Random Forest and Boosting Classifier for a classification dataset from UCI Machine Learning Repository. Discuss the whole experimentation process.

Solution.

a)

Bagging	Boosting
The ensembled model is built from individual experts independently.	The ensembled model is built by adding new models that do well where the previous models fail.
This can happen parallelly.	This has to happen sequentially.
The final output is an equal weighted average/majority vote of the output of individual experts.	The final output is weighted average with a learnt probability distribution where more weight is given to those with better performance on training data.
Tries to reduce over-fitting problem.	Reduces bias.

b)

The algorithm we saw for AdaBoost in class, is as follows:

Algorithm 1 AdaBoost algorithm. Typically, $T = 100$.

Input:

$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$.
Weak learning algorithm a that maps $\mathcal{X}$ to $\mathcal{Y}$
Positive integer T

Initialization:

Initialize sampling distribution $p^{(1)}(i) = \frac{1}{n}$ for $\forall i \in \{1, 2, \ldots, n\}$

Loop:

for $t = 1$ to T
Sample data set $\mathcal{D}^{(t)}$ from $\mathcal{D}$ according to $p^{(t)}(i)$
Learn model $f_t(x)$ from $\mathcal{D}^{(t)}$
Calculate error ϵ_t on training data $\mathcal{D}$ as $\epsilon_t = \sum_{i:f_t(x_i)\neq y_i} p^{(t)}(i)$
Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
Set $w_t = \ln \frac{1}{\beta_t}$
Set $p^{(t+1)}(i) = \frac{p^{(t)}(i)}{Z} \cdot \begin{cases} \beta_t & \text{if } f_t(x_i) = y_i \\ 1 & \text{otherwise} \end{cases}$, where Z is a normalizer
end

Output:

$f(x) = \arg\max_{y \in \mathcal{Y}} \left(\sum_{t=1}^{T} w_t \cdot I(f_t(x) = y) \right)$

For a classification scenario,
Output of final classifier is, $f(x) = \arg\max_{y \in \mathcal{Y}} \left(\sum_{t=1}^{T} w_t \cdot I\left(f_t(x) = y\right) \right)$.
Let $y \in \{0, 1\}$. We have,

$$p^{(t+1)}(i) = \frac{p^{(t)}(i)}{Z_t} \cdot \begin{cases} \beta_t & \text{if } f_t\left(x_i\right) = y_i \\ 1 & \text{otherwise} \end{cases}, \text{ where } Z_t \text{ is a normalizer}$$

This can be written as,

$$p^{(t+1)}(i) = \frac{p^{(t)}(i) e^{-w_t y_i f_t(x_i)}}{Z_t}$$

Unwrapping the recursion, we get,

$$
\begin{aligned}
p^{(T)}(i) &= p^{(1)}(i) \cdot \frac{e^{-w_1 y_i f_1(x_i)}}{Z_1} \cdots \frac{e^{-w_{T-1} y_i f_{T-1}(x_i)}}{Z_{T-1}} \\
&= \frac{1}{N} \cdot \frac{e^{-y_i \sum_t w_t f_t(x_i)}}{\prod_t Z_t} \\
&= \frac{1}{N} \cdot \frac{e^{-y_i f(x_i)}}{\prod_t Z_t}
\end{aligned}
$$

Let us consider training error.

$$
\begin{aligned}
\text{training error } (f) &= \frac{1}{N} \sum_i \begin{cases} 1 & \text{if } y_i \neq f(x_i) \\ 0 & \text{else} \end{cases} \\
&= \frac{1}{N} \sum_i \begin{cases} 1 & \text{if } y_i f(x_i) \leq 0 \\ 0 & \text{else} \end{cases} \\
&\leq \frac{1}{N} \sum_i \exp\left(-y_i f(x_i)\right) \qquad \text{since } e^{-z} \geq 1 \text{ if } z \leq 0 \\
&= \textstyle\sum_i p^{(T)}(i) \prod_t Z_t \\
&= \textstyle\prod_t Z_t
\end{aligned}
$$

The normalization constant can be computed as follows:

$$
\begin{aligned}
Z_t &= \sum_i p^{(t)}(i) \times \begin{cases} e^{-w_t} & \text{if } f_t(x_i) = y_i \\ 1 & \text{if } f_t(x_i) \neq y_i \end{cases} \\
&= \sum_{i: f_t(x_i) = y_i} p^{(t)}(i) e^{-w_t} + \sum_{i: f_t(x_i) \neq y_i} p^{(t)}(i) \\
&= e^{-w_t}(1 - \epsilon_t) + \epsilon_t \\
&= 2\epsilon_t \\
&= 1 - \gamma_t \\
&\leq e^{-\gamma_t}
\end{aligned}
$$

where, the last inequality comes from $1 + x \leq e^x$ for all real x.
Substituting this in training error, we get,

$$
\frac{1}{n} \left| \{ i : f(x_i) \neq y_i \} \right| = \prod_{t=1}^{T} (1 - \gamma_t) \leq e^{-\sum_{t=1}^{T} \gamma_t}
$$

where, ϵ_t is the training error of expert $f_t(x)$.

If we choose, $y \in \{-1, +1\}$ and slightly change the algorithm like below, we get the desired result as stated in the class.

1. We need to take $w_t = \frac{1}{2}\ln(\frac{1}{\beta_t})$ instead.

2. We need to set,

$$p^{(t+1)}(i) = \frac{p^{(t)}(i)}{Z_t} \times \begin{cases} e^{-w_t} & \text{if } y_i = f_t(x_i) \\ e^{w_t} & \text{if } y_i \neq f_t(x_i) \end{cases}$$

   instead.

c)

For Random Forest implementation from scratch, please refer to **RandomForest.ipynb**. For comparison of Random Forest with AdaBoost implementation (I used the scikit-learn implementation), please refer to **CompareBaggingVsBoosting.ipynb**.

For a Random Forest Classifier implemented from scratch, I followed the steps for Boosting as discussed in the class. I used gini index as the node impurity measure for the decision tree. This is equivalent to a random classifier. A more robust measure could be entropy. However, I just wanted to explore and see performance of gini index in action. For evaluation of the algorithm, I used classification accuracy and also used 5-fold cross validation. Random Forest is a bagging ensemble and this is a classification problem, hence I use majority voting algorithm to predict the class for a particular input. My dataset is Sonar Dataset[1], picked from UCI Machine Learning repository[2]. I compared the out for different sample sizes $\in \{10\%, 50\%, 100\%\}$ and different number of trees $\in \{1, 5, 10\}$. My results are as follows:

```
Sample size: 0.100000
Trees: 1
Scores: [60.97560975609756, 53.65853658536586, 68.29268292682927,
   68.29268292682927, 60.97560975609756]
Mean Accuracy: 62.439%
Trees: 5
Scores: [65.85365853658537, 53.65853658536586, 41.46341463414634,
   68.29268292682927, 56.09756097560976]
Mean Accuracy: 57.073%
Trees: 10
Scores: [58.536585365853654, 70.73170731707317, 70.73170731707317,
   68.29268292682927, 68.29268292682927]
Mean Accuracy: 67.317%

Sample size: 0.500000
Trees: 1
Scores: [60.97560975609756, 68.29268292682927, 46.34146341463415,
   70.73170731707317, 78.04878048780488]
Mean Accuracy: 64.878%
Trees: 5
Scores: [68.29268292682927, 70.73170731707317, 70.73170731707317,
   78.04878048780488, 73.17073170731707]
Mean Accuracy: 72.195%
Trees: 10
Scores: [80.48780487804879, 70.73170731707317, 78.04878048780488,
   75.60975609756098, 80.48780487804879]
Mean Accuracy: 77.073%

Sample size: 1.000000
Trees: 1
```

```
Scores: [70.73170731707317, 73.17073170731707, 75.60975609756098,
   70.73170731707317, 80.48780487804879]
Mean Accuracy: 74.146%
Trees: 5
Scores: [80.48780487804879, 65.85365853658537, 65.85365853658537,
   80.48780487804879, 63.41463414634146]
Mean Accuracy: 71.220%
Trees: 10
Scores: [82.92682926829268, 82.92682926829268, 85.36585365853658,
   75.60975609756098, 82.92682926829268]
Mean Accuracy: 81.951%
```

We can clearly see that as the number of trees increases the accuracy of the classifier increases. However, the training time also increases. The accuracy of the classifier becomes almost constant after number of trees 10. Hence, if we are okay with this accuracy we need not add more trees and spend more time for training. Another observation is that sample size can also be considered a hyperparameter. If we pick a low sample size, I am getting a low final accuracy. So, it is better choose a larger sample (of the order of size of the dataset). The accuracy of the classifer decreases from number of trees = 1 to number of trees = 5. This is clearly because of the dataset split in cross validation which can be seen in the scores array right below it. Also, are taking the average accuracy over all the datasets of cross-validation. If we take the maximum instead, the accruacy of the classifer increases as the number of trees increases.

Later, I wanted to see how library functions are performing. Hence, I decided to use scikit-learn implemenation and compare various ensemblers. Here, I am comparing, single Decision Tree, Random Forest (Bagging) ensembler and AdaBoost (Boosting) ensembler. This time I am using make_moons[3] dataset from scikit-learn. This is a simple toy dataset to visualize clustering and classification algorithms. It makes two interleaving half circles. The accuracy measures in various cases are as follows:

```
Decision Tree Accuracy = 0.7545
Random Forest Accuracy = 0.7965
AdaBoost Accuracy = 0.833
```

As we can see, compared to Decision Tree model, accuracy of Random Forest Classifier go up by 5%. The number of trees I used is 100. When compared to Decision Tree model, accuracy of AdaBoost go up by 10%. The number of individual trees I used here is also 100. This clearly shows the power of ensembling techniques. The increase in test accuracy in mainly attributed to the stochastic nature of algorithm. Overall, this exercise gave me a good understanding of how ensembling techniques work and see the practical applications of boosting and bagging and their role in handling bias-variance tradeoff.

□

References

[1] Gorman, R. P., and Sejnowski, T. J. (1988). Connectionist Bench (Sonar, Mines vs. Rocks) Data Set [https://archive.ics.uci.edu/ml/datasets/Connectionist+Bench+(Sonar,+Mines+vs.+Rocks)].

[2] Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

[3] Moons dataset of scikit-learn. Obtained from sklearn.datasets.make_moons. [https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_moons.html].