Assignment – 2
Part – A

QA1) What is the key idea behind bagging? Can bagging deal both with high variance (overfitting) and high bias (underfitting)?

Bagging (Bootstrap Aggregating) is a machine learning ensemble approach that seeks to decrease the variance of a model by merging numerous models trained on various subsets of the original data. The primary idea of bagging is to incorporate randomization into the training process, reducing the chance of overfitting.

Bagging processes and steps include:

1. **Create multiple bootstrap samples:** Make several random samples from the original data collection. The size of each sample should be the same as the original data set.

2. **Train multiple models:** On each of the bootstrap samples, train a different model. Depending on the difficulty, the models may be of the same or various sorts.

3. **Aggregate predictions:** For each new instance that must be predicted, utilize all the trained models to make a forecast, and then aggregate their predictions by taking the average (for regression problems) or the majority vote. (For classification problems).

4. **Evaluate performance:** Evaluate the bagged model's performance on a validation set or using cross-validation. Compare the bagged model's performance against a single model trained on the original data set.

Bagging helps to deal with large variation by minimizing the effect of outliers and reducing the model's sensitivity to individual data points. Bagging can smooth out predictions by merging several models trained on various samples of data.

However, bagging may not be effective in dealing with high bias. If the model is too simple and fails to capture the underlying patterns in the data, bagging may only help to reduce the variance of the model but not improve its overall performance. In such cases, other techniques such as boosting or increasing the complexity of the model may be more effective.

QA2) Why bagging models are computationally more efficient when compared to boosting models with the same number of weak learners?

Bagging (Bootstrap Aggregating) and Boosting are two common ensemble learning approaches used in machine learning to improve model performance. While both approaches entail employing several weak learners to create a strong learner, they differ in how the weak learners are combined.

Bagging entails training many instances of the same model on various random subsets of the training data and then aggregating these models' predictions by average or taking the majority vote. Boosting, on the other hand, entails systematically instructing weak learners, with each new student being educated to rectify the faults produced by the prior learners.

When it comes to why Bagging models are computationally more efficient than Boosting models with the same number of weak learners, the following arguments might be considered:

1. **Parallelization:** Because each instance of the model is trained independently on a different subset of the training data, bagging models are simply parallelized. As a result, the calculation may be distributed across numerous processors or workstations, resulting in quicker training times. Boosting, on the other hand, necessitates sequential instruction of learners, which cannot be parallelized as much.

2. **Reduce overfitting:** Bagging minimizes model variance by training on diverse subsets of data, which aids in decreasing overfitting. Boosting, on the other hand, can occasionally result in overfitting since the model is taught to correct the mistakes made by earlier learners, which might result in over-emphasizing specific patterns in the data.

3. **Less Complex weak learners:** Bagging reduces overfitting, allowing for the use of fewer complicated weak learners while still achieving decent results. This means that with bagging, each instance of the model may be trained more quickly, resulting in shorter total training periods. Boosting, on the other hand, necessitates more complicated weak learners since each subsequent learner must repair the errors of prior learners.

4. **No need for weight:** At each iteration of boosting, the data is reweighted to provide additional weight to the misclassified samples. This necessitates more computations, which might slow down the training process. Bagging, on the other hand, trains each instance of the model on an equal portion of the data, removing the requirement for reweighting.

Bagging models are computationally more efficient than Boosting models with the same number of weak learners because they can parallelize, avoid overfitting, employ fewer complicated weak learners, and remove the requirement for reweighting.

QA3) James is thinking of creating an ensemble mode to predict whether a given stock will go up or down in the next week. He has trained several decision tree models, but each model is not performing any better than a random model. The models are also very similar to each other. Do you think creating an ensemble model by combining these tree models can boost the performance? Discuss your answer.

Creating an ensemble model may frequently assist to enhance the overall performance of individual models, especially when the various models have diverse strengths and shortcomings. However, in James' situation, there are a few of variables that may restrict the usefulness of the ensemble approach:

1. **Poor individual model performance:** If decision tree models perform no better than random, merging them is unlikely to result in a meaningful improvement. Ensemble approaches are often most effective when the individual models have at least some predictive capabilities.

2. **High similarity between models:** Ensemble models rely on variety among distinct models to capture diverse elements of the issue, making the overall model more accurate and resilient. If the decision tree models are highly like one other, the ensemble model may not profit as much from their combination.

Given these concerns, it's probable that combining these decision tree models may not result in a considerable improvement in performance. Instead, James should examine the following techniques to improving his stock prediction models:

1. **Diversify the base models:** Instead of only decision trees, James may use additional machine learning methods (such as support vector machines, neural networks, or logistic regression) to build a more diversified range of models for the ensemble.

2. **Optimize the decision tree models:** To increase the performance of the individual decision trees, James may attempt adjusting hyperparameters, utilizing feature selection approaches, or using other pre-processing methods.

3. **Use ensemble-specific techniques:** James may combine the models using ensemble-specific approaches like bagging, boosting, or stacking, which are meant to utilize the strengths of each individual model and increase overall performance.

4. **Incorporate additional data:** To increase the effectiveness of his models, James may require more or higher-quality data. He may think about adding new features, using different data sources, or growing his training dataset.

while an ensemble model may not significantly boost the performance of James's decision tree models given their poor individual performance and high similarity, he can explore other strategies to improve the models and their combination.

QA4) Consider the following Table that classifies some objects into two classes of edible (+) and non- edible (-), based on some characteristics such as the object color, size, and shape. What would be the Information gain for splitting the dataset based on the "Size" attribute?

Color---color_	Size---size_	Shape—shape_	Edible?—Edible?_
Yellow	Small	Round	+ +
Yellow	Small	Round	- -
Green	Small	Irregular	+ +
Green	Large	Irregular	- -
Yellow	Large	Round	+ +
Yellow	Small	Round	+ +
Yellow	Small	Round	+ +
Yellow	Small	Round	+ +
Green	Small	Round	- -
Yellow	Large	Round	- -
Yellow	Large	Round	+ +
Yellow	Large	Round	- -
Yellow	Large	Round	- -
Yellow	Large	Round	- -
Yellow	Small	Irregular	+ +
Yellow	Large	Irregular	+ +

To calculate the information gain for splitting based on the "Size" attribute, we must first compute the entropy of the original dataset and the weighted average entropy of the two subsequent datasets after dividing based on "Size."

The entropy of the original dataset is given by:

Entropy(S) = -p(+) log2 p(+) - p(-) log2 p(-)

Where

- p(+) is the proportion of edible objects.
- p(-) is the proportion of non-edible objects.

In this case, there are 9 edible objects and 6 non-edible objects, so:

p(+) = 9/(9+6) = 0.6 p(-) = 6/(9+6) = 0.4

Entropy(S) = -0.6 log2 0.6 - 0.4 log2 0.4 = 0.971

Next, we need to calculate the entropy of the two resulting datasets after splitting based on "Size". There are 9 objects that are small and 6 that are large.

For the dataset where Size = small:

p(+) = 6/9 = 0.6667 p(-) = 3/9 = 0.3333

Entropy(Size = small) = -0.6667 log2 0.6667 - 0.3333 log2 0.3333 = 0.9183

For the dataset where Size = large:

p(+) = 3/6 = 0.5 p(-) = 3/6 = 0.5

Entropy(Size = large) = -0.5 log2 0.5 - 0.5 log2 0.5 = 1

Now, we can calculate the weighted average entropy after splitting based on "Size":

Weighted Average Entropy(Size) = (9/15) * Entropy(Size = small) + (6/15) * Entropy(Size = large) = (0.6 * 0.9183) + (0.4 * 1) = 0.95098

Finally, we can calculate the information gain by subtracting the weighted average entropy after splitting from the entropy of the original dataset:

Information Gain(Size) = Entropy(S) - Weighted Average Entropy(Size) = 0.971 - 0.95098 = 0.02002

Therefore, the information gain for splitting based on the "Size" attribute is 0.02002.

QA5) Why is it important that the m parameter (number of attributes available at each split) to be optimally set in random forest models? Discuss the implications of setting this parameter too small or too large.

In random forest models, the m parameter, also known as the maximum number of features, examined at each split, is a critical hyperparameter. It limits the number of characteristics that may be utilized to divide each tree, hence controlling the model's unpredictability.

If the m parameter is set too low, the model will not have enough features to pick from, which might result in underfitting. Underfitting happens when the model is too simplistic and fails to capture the complexity of the data, resulting in poor performance on both the training and test sets. This is due to the model's inability to identify key patterns and correlations in the data due to the restricted number of characteristics available for splitting. In this scenario, raising the m parameter can assist to enhance the model's performance by allowing it to examine additional features.

If the m parameter is set too big, the model will be overly flexible and may overfit the data. Overfitting happens when the model is too complicated and fits the noise in the training data rather than the underlying patterns and correlations. This results in strong performance on the training set but poor performance on the test set. In this scenario, lowering the m parameter can assist to improve model performance by reducing the number of features available for splitting and the model's complexity.

Setting the m parameter optimally is crucial in random forest models since it has a substantial influence on the model's performance. It is crucial to strike a balance between having enough features to capture interesting patterns and correlations in the data and not having too many features, which can lead to overfitting. The appropriate value of m depends on the dataset and should be discovered through experimentation and cross-validation.