CSE454 Data Mining Homework 1

1) Precondition: A training set $S := (x_1, y_1), \dots, (x_n, y_n)$, features F, and number of trees in Forest B

```
1  function RandomForest(S, F)
2      H ← ∅
3      for i ∈ 1, ..., B do
4          $S^{(i)}$ ← A bootstrap sample from S.
5          $h_i$ ← RandomizedTreeLearn($S^{(i)}$, F)
6          H ← H ∪ {$h_i$}
7      end for
8      return H
9  end function
10 function RandomizedTreeLearn(S, F)
11     At each node
12         f ← very smal subset of F
13         Split on best feature in f
14     return The learned tree
15 end function
```

2) Random Forest and Decision Tree are classification methods. However, they differ from each other in three aspects. Firstly, Decision Tree operates as a standalone model. It makes decisions based on some set of rules derived from the training data. On the other hand, Random Forest is an ensemble model consisting of multiple decision trees. Secondly, the training of these models is different. A Decision Tree undergoes training via a recursive binary splitting process, selecting the optimal feature for data division at each node based on a specified criterion. Conversely, Random Forest employs a technique known as bagging, or bootstrap aggregating. It builds multiple Decision Trees, each trained on a random subset of the training data. The final prediction is then determined by aggregating the outputs of these individual trees throug voting. Finally, Random Forest selects features to use in decision trees randomly. Contrary to Random Forest, in Decision Tree, all features are considered at each split, and the algorithm chooses the one that yields the most favorable split according to the selected criterion.

Random Forest is more likely to give better results than Decision Tree because of 3 reasons. Firstly, Random Forest is an ensemble method, meaning it builds multiple Decision Trees and merges them together to get a more accurate and stable prediction. Each tree in the forest is trained on a random subset of the data and makes its own predictions. The final prediction is then determined by a majority vote. Secondly, Decision Tree tends to overfit the training data. Random Forest mitigates this issue by training multiple trees on a different subsets of the data. Thirdly, Random Forest also introduces randomness by considering only a random subset of features at each split in each tree. This helps to disassociate the trees and ensures that each tree in the forest is making decisions based on different aspects of the data.

3) n_estimators: The number of trees in the forest.
max_depth: The maximum depth of the individual trees.
min - samples - split: The minimum number of samples required to split an internal node.
min - samples _ leaf: The minimum number of samples required to split a leaf node.
max_features: The number of features to consider when looking for the best split.
min - features: The number of samples to draw from X to train each base estimator
Criterion: The function used to measure the quality of a split.
bootstrap: Whether to bootstrap samples when building trees.

4) Here are some well-known classifiers and their complexities:
a. Logistic Regression: O(m.n.k) [m = number of training examples, n = number of features, k = number of iterations until convergence]
b. Naive Bayes: O(m.n) [m = number of training examples, n = number of features]
c. K-Nearest Neighbors: O(m.n.k) [m = number of training examples, n = number of features, k = number of neighbors]
d. Decision Tree: O(m.n.log(m)) [m = number of training examples, n = number of features]
e. Random Forest: O(m.n.k.log(m)) [m = number of training examples, n = number of features k = number of trees]

As it can be seen from the time complexities of the above classification algorithms, Random Forest is one of the highest time complexities. Because Random Forest consist of multiple Decision Trees, it may seem inefficient. However, the trade-off can be worth because Random Forest generally provides more accurate results.

5) There are a lot of strategies to increase the accuracy of Random Forest. Here are some general strategies:
a. Feature Engineering: Creating and selecting relevant features are important factors to increase accuracy.
b. Data Balancing: Random Forest is sensitive to class imbalance, so avoiding class imbalance may increase accuracy.
c. Outlier Handling: Outliers can have a disproportionate impact on decision tree models. Robust preprocessing techniques can lead to improved model accuracy.

6) There are a lot of strategies to increase the performance of Random Forest. Here are some general strategies which can increase the performance when applied properly:

a. Tune Tree-Specific Hyperparameters: Adjusting hyperparameters specific to individual trees, such as maximum depth, minimum samples split, and minimum samples per leaf can help model's performance.

b. Bootstrap Sampling Variations: While Random Forest inherently introduces randomness through bootstrap sampling, experimenting with different sampling techniques or adjusting the sample size can impact model's performance.

c. Uncorrelated High Performing Trees: As Thomas G. Dietterich proposed, performance of ensemble learner is highly dependent on two factors: one is the accuracy of each component learner; the other is the diversity among these components. So, to increase the Random Forest (ensemble learner) performance, trees with high accuracy and low correlation between other trees must be found and used.

7) Here are some improvement strategies in Random Forest for online data:

a. Incremental Learning: To not have to retrain entire model when new data arrives, incremental learning strategy can be used. This can be achieved by training a new tree and adding it to existing ensemble.

b. Hyperparameter Tuning: Finding the optimal set of hyperparameters for the algorithm increases the performance of the model.

c. Feature Selection: Using more informative parameters may increase the performance of the Random Forest. This approach can be done with some techniques such as PCA or Lasso.

8) Semi-supervised learning is a machine learning paradigm that lies between supervised learning and unsupervised learning. In supervised learning, models are trained on labelled data, where each example in the training set is paired with the corresponding target label. On the other hand, in unsupervised learning, models are trained on unlabelled data, and the algorithm must find patterns or structure in the data without explicit guidance.

Semi-supervised learning combines elements of both approaches by using small amount of labelled data along with a larger amount of unlabelled data for training. The idea is to leverage the benefits of having labelled examples while also taking advantage of the potentially vast amounts of unlabelled data that may be easier and cheaper to obtain.

The semi-supervised learning technique I plan to create is "Consensus-based Semi-supervised Learning with Clustering". This method involves treating the entire dataset as if it has no labels initially. It employs clustering models, typical of unsupervised learning, to group the data. Subsequently, the unlabelled data points are assigned labels based on a consensus derived from the labelled data. This consensus-based labelling approach enhances the overall understanding of the data and facilitates the integration of labelled and unlabelled datasets.

Precondition: UD is unlabelled data, LD is labelled data, CD is combined data.

```
1 function Consensus-based-Clustering (CD)
2     Apply multiple clustering techniques to CD.
3     Label clusters based on consensus derived from LD.
4     Train the model with both newly LD and old LD.
5     Use trained model to make predictions on the UD which consensus cannot be achieved.
6     return CD
7 end function.
```

CamScanner ile tarandı

9) Transfer learning is a machine learning technique where a model trained on one task is adapted to work on a second related task. The idea behind transfer learning is to leverage the knowledge gained during the training of one model on a particular task and apply it to a different but related task. One of the popular transfer learning techiques is fine-tuning. Fine-tuning involves taking a pre-trained model and training it further on a new dataset. The process of fine-tuning involves replacing the last layer of the pre-trained model with a new layer that has same number of output classes as the new dataset. The weights of the new layer initialized randomly, and the entire model is then trained on the new dataset. For example, converting a model that performs object detection for cars to detect trucks is a fine-tuning technique.

Precondition: New dataset S, pre-trained model P

```
function Fine_tuning(P, S)
    Modify last layer of P
    Freeze remaining layers.
    Set up training parameters
    Train P with S.
    return P
end function.
```

Here is the simple outline of how fine-tuning technique be applied.