Emre YILMAZ
1901042606

Q-1.)

Algorithm for Random Forest :

Input :

- Training set S
- Number of trees to grow, T
- Number of features to consider at each split, m

Output :

- A set of decision trees, forest.

Algorithm :

1. Initiliaze an empty forest.
2. For t=1 to D, do
   - 2.1 Create a bootstrapped (1) sample St from S
   - 2.2 Create a decision tree Dt using St
     - At each node of the tree, randomly select m features from the available features.
     - Choose the best split among these m features based on a attribute selection criteria (2). (for example, Gini, information gain etc.)
     - Split the node into child nodes using the selected best split
     - Repeat the process for each child node until a stopping criterion is met. (3)
   - 2.3 Add Dt to tree list. (Forest)
3. Return the forest

Notes :

* To make predictions, output the mode of the predictions in the forest (T trees).

(1) => It is a technique for creating multiple subsets of a dataset by sampling with replacement. → Bootstrapping

(2) => Attribute selection criteria is a heuristic for selecting the splitting criterion that "best" seperates given data partition.

Go to next page.

(2) Continue => It provides a ranking for each attribute describing the given training tuples.

(3)

Q-2.)

* Decision tree is a single tree to make predictions.
  Random forest is an ensemble method that builds multiple decision trees and combines their predictions.

* Decision tree is trained on the entire dataset
  Random forest is trained on random subset that is got using bootstrapping.

* Random forest is more resistant to overfitting due to combination of multiple trees and random feature selection.

* Random forest is less prone to outliers.

* Random forest takes a long time to build.

* Decision trees are easy to visualise and interpret. They do not seem complex.

* Decision tree considers all features at each split.
  Random forest considers a random subset of features.

In summary, random forests are likely to give better results than decision trees since they avoid overfitting and less prone to outliers.

Q-3.)

* The number of trees
* Attribute selection criteria (Gini, Information Gain Etc.)
* Max. depth of each decision tree.
* Max. features to consider when finding optimal split.
* Min. number of samples required to split a node.
* Min. number of samples required to be at leaf node.

These are the main parameters of random forest. The reference is Scikit-learn. Moreover we can talk about 2 more parameters

* random_state, controls both the randomness of the bootstrapping and the sampling of the features to consider
* bootstrap, whether bootstrap samples are used when building trees.

Q-4.) (

Compared models =)

1. K-nearest-neighbor
   * Training time complexity = O(1) since model stores whole data simply.
   * Classifying time complexity => O(n*d) where n is size of training set and d is the number of features.

Although classifying time complexity can be reduce using some improvement techniques such as "by presorting and arranging the stored tuples into search trees" or "parallel implementation", I will consider classical implementation of KNN.

Go to next page

2-) Logistic Regression =)

* Training complexity of logistic regression is $O(n * m)$ where n is sample count and m is feature count.
* Prediction complexity is $O(m)$ where m is feature count.

3-) Naive Bayes Classifier =)

* Training complexity of this model is $O(n * d)$ where n is number of instances of training set and d is number of feature.
* Prediction complexity is $O(d)$ where d is the number of features.

- Random Forest Classifier =)

* Training time complexity is $O(T * n * d \log d)$ where T is tree number in forest, n is training sample number, d is the number of feature
* Prediction time complexity is $O(T. \log d)$ where T is number of trees in the forest, d is feature number.

- Logistic regression has faster training complexity especially for datasets with fewer features.
Logistic regression only computes sum of features and applying a sigmoid function that is much more efficient than Random Forest

- Naive Bayes is the one of the fastest models for training and it is faster than Random Forest.
Naive Bayes calculates the probability of each class given the input features. It is very fast.

- go to next page -

As we see, Random Forest is computationally intensive and slow for both prediction and training especially with many trees and features.

Q-5.)

We have various strategies to improve accuracy of Random Forest.

* Feature Engineering such as feature selection. It provides a lot of advantages. For example, it helps us dealing with curse of dimensionality, it helps to prevent overfitting, it improves generalization etc.

* Data Preprocessing is first thing to do. If we handle missing values and detect outliers, accuracy can be improved.

* Adding more data will improve the accuracy.

* Cross-Validation assesses the model's performance and helps us to identify potential over fitting and under fitting issues.

* Hyperparameter Tuning

  - Experiment with different tree numbers and find the optimal number.
  - Adjust the maximum depth
  - Experiment with different different max. feature value and find the optimal number

Q-6.)

There may be several options to improve performance.

* Parallel Processing

Training => Random Forest inherently supports parallel processing. Each tree can be trained independently. Utilizing multi-core CPUs can improve performance.

Prediction => Also, prediction tasks can be parallelized.

* Reducing Tree Number

It can improve the performance but it affects accuracy and robustness negatively.

* Limiting Tree Depth

Limiting the max. depth can reduce both training and prediction performance.

Q-7.)

There are several strategies and adaptations that can be considered to make random forests more suitable for online learning:

* Sliding Window

Maintain a sliding window of the most recent data. As new data comes in, oldest data is removed the window and the model is retrained with the new window of data.

* Incremental Learning

Train each new decision tree with new data while preserving the knowledge gained from previous trees. The oldest tree can be removed after new tree is trained.

Q-8 .)

Let's create a new semi-supervised learning technique. It will combine feature-similarity based labelling and provides a cluster refinement. We will label the unlabelled data based on feature similarities, then refine labels using a clustering process.

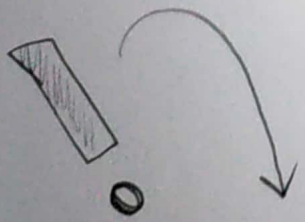
Algorithm =>

Inputs:

- X_labeled // Features of labelled data
- y_labelled // Labels of labelled data
- X_unlabelled // Features of unlabelled data
- max_iterations
- threshold // similarity threshold

1. y_unlabelled = array[len(X_unlabelled)]
   // Initilioze labels for unlabelled data

2. for iteration in range (max_iterations)
   for i in range (len(X_unlabelled))
   for x_l, y_l in (X_labeled, y_labeled)
   if (isSimilar (X_unlabelled[i], x_l, threshold)
   y_unlabelled[i] = y_l
   break
   combined_data = X_labelled + newly labeled
   clusters = clustering (combined_data)
   y_unlabelled = refine_labels (y_unlabelled, clusters)
   // Some update on similarity func or clustering
   If check_finish (y_unlabelled) // no change
   break
   return y_unlabelled



* You can firstly read the explanation in the next page

Q-8.) Continue

1. We firstly propogate labels from labeled dataset to the unlobeled dataset bosed on similority. The functio is Similar calculates this similority and if this Similority is greater than threshold, we assign label of labelled instance to unlobelled instance.

2. After that, we apply clustering and refine the labels of unlobelled instances.

3. If there is no change in new labels of unlobelled dataset or we reached the maximum iteration, finish the function. Else, go to step-1 and continue to iteration.

* I am aware of this function seems complicated and unsuccessful. However, it is really difficult task to generate a new technique that is unseen in literature

* I both use labelled data and a unsupervised method clustering. So, this function is semi-supervised.

* Also, there is a missing modification in the function. If feature similority and clustering results are the same (if their labels the same) there is no problem. However, if they are different, we must change similority functio or clustering parameters. Pseudo-code does not include this modification.

* I did my best...

Q-9.)

One of the most common transfer learning technique is feature extraction. A pre-trained model is used as a feature extractor, where the learned representations from the pre-trained model serve as input features for a new task-specific model.

We simply add a new classifier which will be trained from scratch, on top of the pre-trained model so that we can repurpose the feature maps learned previously for the dataset.

We do not need to retrain the entire model. The base convolutional network already contains features that are generically useful for classifying.

Algorithm:

1. Load the pre-trained model without the top layer
2. Freeze the layers of the pre-trained model to prevent them from being updated during training.
3. Add new fully connected layer that matches the number of classes in the dataset.
4. Compile the model with appropriate loss function and optimizer.
5. Train the model with new dataset.
6. Evaluate the model with the test data.