UPPSALA
UNIVERSITET

Data Engineering II

Assignment 1

Jinglin Gao

May 10, 2023

1 Task 1.1

1.1 Q1

The hyperparameters found and the associated cross-validation score. How does it compare to the score for the default parameters?

In this task, I first try to use the whole dataset and it takes hours to train in one VM, so in order to shorter the process time I finally use 10000 data as the training set and 2000 data as a test set. Also, in order to solve the "out of memory" problem, I set up a head node with "ssh.small.highmem" flavor.

```
In [28]: X_train, X_test, y_train, y_test = train_test_split(data, label, train_size=10000, test_size=2000, random_state=42)

In [29]: rfc = RandomForestClassifier(random_state=0)
         rfc.fit(X_train, y_train)
         print(rfc.get_params())
         print(cross_val_score(rfc, X_train, y_train, cv=5))
         rfc.score(X_test,y_test)

         {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features':
         'sqrt', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_sample
         s_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_stat
         e': 0, 'verbose': 0, 'warm_start': False}
         [0.8115 0.793  0.8165 0.7995 0.801 ]

Out[29]: 0.8095
```

Figure 1: Train the model with default parameters

We can see that in the default parameters, the "max_depth" is set to None, "n_estimators" is set to 100, and the "ccp_alpha" is set to 0.0. Then we use the Ray tune and grid search method to find these three hyperparameters.

```
In [6]: import time
        from tune_sklearn import TuneGridSearchCV

In [7]: ray.init(address="auto")
        nodes = ray.nodes()
        worker_ips = [node['NodeManagerAddress'] for node in nodes if node['Alive'] == True]
        print(worker_ips)

        2023-05-08 13:07:46,614 INFO worker.py:1432 -- Connecting to existing Ray cluster at address: 192.168.2.103:637
        9...
        2023-05-08 13:07:46,628 INFO worker.py:1616 -- Connected to Ray cluster. View the dashboard at http://127.0.0.1:82
        65

        ['192.168.2.103', '192.168.2.211', '192.168.2.55']

In [11]: start_time = time.time()
         rf3 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [1, 2, 3, 4],
                       "n_estimators": [50, 100, 150, 200],
                       "ccp_alpha": [0.0, 0.001, 0.01, 0.1]}
         rf3 = TuneGridSearchCV(rf3, param_dist, n_jobs = -1)
         rf3.fit(X_train, y_train)
         end_time = time.time()
         print("Time consume: ", end_time - start_time)
         print(rf3.cv_results_)

         {'params': [{'max_depth': 1, 'n_estimators': 50, 'ccp_alpha': 0.0}, {'max_depth': 1, 'n_estimators': 50, 'ccp_alph
         a': 0.001}, {'max_depth': 1, 'n_estimators': 50, 'ccp_alpha': 0.01}, {'max_depth': 1, 'n_estimators': 50, 'ccp_alp
         ha': 0.1}, {'max_depth': 2, 'n_estimators': 50, 'ccp_alpha': 0.0}, {'max_depth': 2, 'n_estimators': 50, 'ccp_alph
         a': 0.001}, {'max_depth': 2, 'n_estimators': 50, 'ccp_alpha': 0.01}, {'max_depth': 2, 'n_estimators': 50, 'ccp_alp
         ha': 0.1}, {'max_depth': 3, 'n_estimators': 50, 'ccp_alpha': 0.0}, {'max_depth': 3, 'n_estimators': 50, 'ccp_alph
         a': 0.001}, {'max_depth': 3, 'n_estimators': 50, 'ccp_alpha': 0.01}, {'max_depth': 3, 'n_estimators': 50, 'ccp_alp
         ha': 0.1}, {'max_depth': 4, 'n_estimators': 50, 'ccp_alpha': 0.0}, {'max_depth': 4, 'n_estimators': 50, 'ccp_alph
         a': 0.001}, {'max_depth': 4, 'n_estimators': 50, 'ccp_alpha': 0.01}, {'max_depth': 4, 'n_estimators': 50, 'ccp_alp
         ha': 0.1}, {'max_depth': 1, 'n_estimators': 100, 'ccp_alpha': 0.0}, {'max_depth': 1, 'n_estimators': 100, 'ccp_alp
         ha': 0.001}, {'max_depth': 1, 'n_estimators': 100, 'ccp_alpha': 0.01}, {'max_depth': 1, 'n_estimators': 100, 'ccp_
         alpha': 0.1}, {'max_depth': 2, 'n_estimators': 100, 'ccp_alpha': 0.0}, {'max_depth': 2, 'n_estimators': 100, 'ccp_
         alpha': 0.001}, {'max_depth': 2, 'n_estimators': 100, 'ccp_alpha': 0.01}, {'max_depth': 2, 'n_estimators': 100, 'c
         cp_alpha': 0.1}, {'max_depth': 3, 'n_estimators': 100, 'ccp_alpha': 0.0}, {'max_depth': 3, 'n_estimators': 100, 'c
         cp_alpha': 0.001}, {'max_depth': 3, 'n_estimators': 100, 'ccp_alpha': 0.01}, {'max_depth': 3, 'n_estimators': 100,
         'ccp_alpha': 0.1}, {'max_depth': 4, 'n_estimators': 100, 'ccp_alpha': 0.0}, {'max_depth': 4, 'n_estimators': 100,
         'ccp_alpha': 0.001}, {'max_depth': 4, 'n_estimators': 100, 'ccp_alpha': 0.01}, {'max_depth': 4, 'n_estimators': 10
         0, 'ccp_alpha': 0.1}, {'max_depth': 1, 'n_estimators': 150, 'ccp_alpha': 0.0}, {'max_depth': 1, 'n_estimators': 15
```

Figure 2: Train the model with Ray tune

```
In [10]: print(rf3.best_params_)
         print(rf3.best_estimator_)
         rf3.best_estimator_.score(X_test, y_test)

         {'max_depth': 4, 'n_estimators': 200, 'ccp_alpha': 0.0}
         RandomForestClassifier(max_depth=4, n_estimators=200, random_state=0)

Out[10]: 0.685
```

Figure 3: Best parameters found by ray tune process

The figures show the process I use the Ray tune to find the best parameters. Basically, we will give some candidate parameters and ray tune will try any possible combination and find out the best one. In the default setting there is no limitation of "max_depth", this might cause over-fitting, so I discarded this option.

The cross-validation score for the default parameters setting is 0.8095. However, for the Ray tune experiment, the score is 0.685. It shows that the accuracy is not as good as the default setting, here are some possible reasons:
1. I just use part of the dataset, so the training data is not large enough to get a better score.
2. I set the limitation of training depth at 4. If I give a higher choice of "max_depth" then the accuracy might be better. But there is also a risk of over-fitting.

1.2 Q2

The time taken to complete the tuning when using 1, 2, and 3 VMs of "small" flavor.

Time spent doing tests on three virtual machines.

```
In [7]: ray.init(address="auto")
        nodes = ray.nodes()
        worker_ips = [node['NodeManagerAddress'] for node in nodes if node['Alive'] == True]
        print(worker_ips)

        2023-05-08 13:07:46,614 INFO worker.py:1432 -- Connecting to existing Ray cluster at address: 192.168.2.103:637
        9...
        2023-05-08 13:07:46,628 INFO worker.py:1616 -- Connected to Ray cluster. View the dashboard at http://127.0.0.1:82
        65

        ['192.168.2.103', '192.168.2.211', '192.168.2.55']
```

```
In [11]: start_time = time.time()
         rf3 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [1, 2, 3, 4],
                       "n_estimators": [50, 100, 150, 200],
                       "ccp_alpha": [0.0, 0.001, 0.01, 0.1]}
         rf3 = TuneGridSearchCV(rf3, param_dist, n_jobs = -1)
         rf3.fit(X_train, y_train)
         end_time = time.time()
         print("Time consume: ", end_time - start_time)
         print(rf3.cv_results_)

         Time consume:  158.79856824874878
```

Figure 4: Time spent when using three VMs

Time spent doing tests on two virtual machines.

```
In [11]: nodes = ray.nodes()
         worker_ips = [node['NodeManagerAddress'] for node in nodes if node['Alive'] == True]
         print(worker_ips)

         ['192.168.2.103', '192.168.2.55']
```

```
In [12]: start_time = time.time()
         rf2 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [1, 2, 3, 4],
                       "n_estimators": [50, 100, 150, 200],
                       "ccp_alpha": [0.0, 0.001, 0.01, 0.1]}
         rf2 = TuneGridSearchCV(rf2, param_dist, n_jobs = -1)
         rf2.fit(X_train, y_train)
         end_time = time.time()
         print("Time consume: ", end_time - start_time)
         print(rf1.cv_results_)

         Time consume:  199.32639145851135
```

Figure 5: Time spent when using two VMs

Time spent doing tests only on the head node.

```
In [7]: ray.init(address="auto")
        nodes = ray.nodes()
        worker_ips = [node['NodeManagerAddress'] for node in nodes if node['Alive'] == True]
        print(worker_ips)

        2023-05-08 14:30:50,464 INFO worker.py:1432 -- Connecting to existing Ray cluster at address: 192.168.2.103:637
        9...
        2023-05-08 14:30:50,478 INFO worker.py:1616 -- Connected to Ray cluster. View the dashboard at http://127.0.0.1:82
        65

        ['192.168.2.103']

In [8]: start_time = time.time()
        rf1 = RandomForestClassifier(random_state=0)
        param_dist = {"max_depth": [1, 2, 3, 4],
                      "n_estimators": [50, 100, 150, 200],
                      "ccp_alpha": [0.0, 0.001, 0.01, 0.1]}
        rf1 = TuneGridSearchCV(rf1, param_dist, n_jobs = -1)
        rf1.fit(X_train, y_train)
        end_time = time.time()
        print("Time consume: ", end_time - start_time)
        print(rf1.cv_results_)

        Time consume:  235.8896448612213
```

Figure 6: Time spent when using one VM

When I was doing the distributed learning process, there are always errors showing that workers were killed due to memory pressure. I guess this is because I use "ssh.small" flavor for the two worker nodes while I use "ssh.small.highmem" flavor for the head node. So for the worker node there might be not enough memories to run the experiments, which will affect the performance.

The time spent decreases when I use more VMs. When I use all three VMs, the speed-up is 1.49. The strong scalability is not good enough. And reason is might because some workers got killed during the process.

2 Task 1.2

2.1 Q1

Equipped with this background knowledge, explain (in max 0.5 page 11pt Arial) the main difference between data parallel training strategies and model parallel training strategies. In which situations are the respective strategies most appropriate to use? Assuming you also need to do hyperparameter tuning, in which situations would you consider distributing the training of individual neural networks rather than distributing test cases for the tuning pipeline?

Data parallel training strategy replicates the model on every computational resource to generate gradients independently and then communicates those gradients at each iteration to keep model replicas consistent.[1] Data parallelism focuses on distributing the data across different nodes[3], each device computes the forward and backward passes on a subset of the data. The gradients are then aggregated across all devices and used to update the model parameters. Basically, the data used in each node is different while the same model is used for every node. Also, it is fast for small networks but slow for large networks since large amounts of data need to be transferred between processors all at once.

Model parallelism, on the other hand, is focused on the model architecture distribution. Each device computes a portion of the model's forward and backward passes. The output of one device is then used as the input to another device, and the gradients are communicated across the devices to update the model parameters.

In summary, the main difference between data parallelism and model parallelism is in how the workload is divided and the computations are distributed across multiple devices. Data parallelism partitions the data and trains with the same model while model parallelism partitions the model so each device trains a different part of the model on the same data.

Data parallelism is most appropriate when the dataset is large, and the network can fit in the memory of a single node. This strategy is also effective when the model is highly parameterized, and the training process requires many epochs to converge. On the other hand, model parallelism is most appropriate when the model is too large to fit in the memory of a single node, and when the computation of each layer is computationally intensive.[2]

When it comes to hyperparameter tuning, distributing test cases is like data parallelism, and distributing the training of individual neural networks is like model parallelism. So I will consider distributing the training of individual neural networks when the network is too large to fit in the memory of a single node, and also we do have a lot of hyperparameters (the model is very large). In this case, the model is partitioned across the computing nodes, and each node trains a separate neural network with a different set of hyperparameters. If the number of hyperparameters is small (which means the model is small and fits in a single node) and the evaluation of each configuration is computationally expensive, then considering distributing the test cases will be more reasonable.

3 Task 2

3.1 Set up and run fully distributed federated training for Keras

Here I use three virtual machines to deploy the FEDn. One for the base services (MongoDB and Minio) + reducer, one for one combiner, and one to run 2 clients. We use docker and docker-compose to connect everything. The setup network is shown below:

Figure 7: Network of FEDn

Figure 8: The combiner

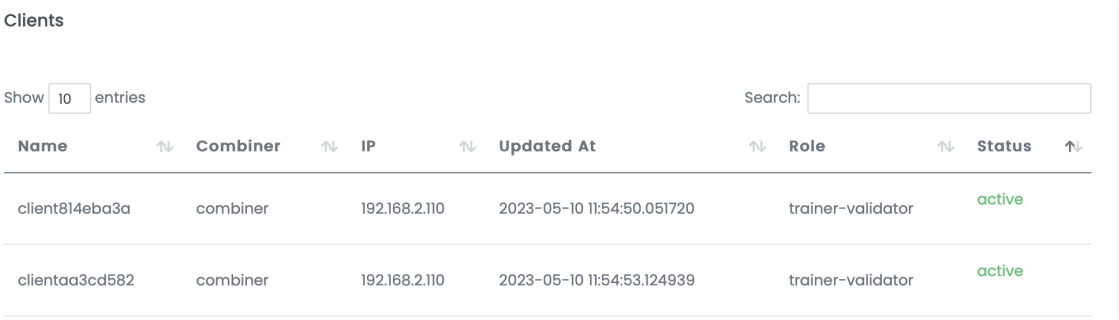

Figure 9: The clients

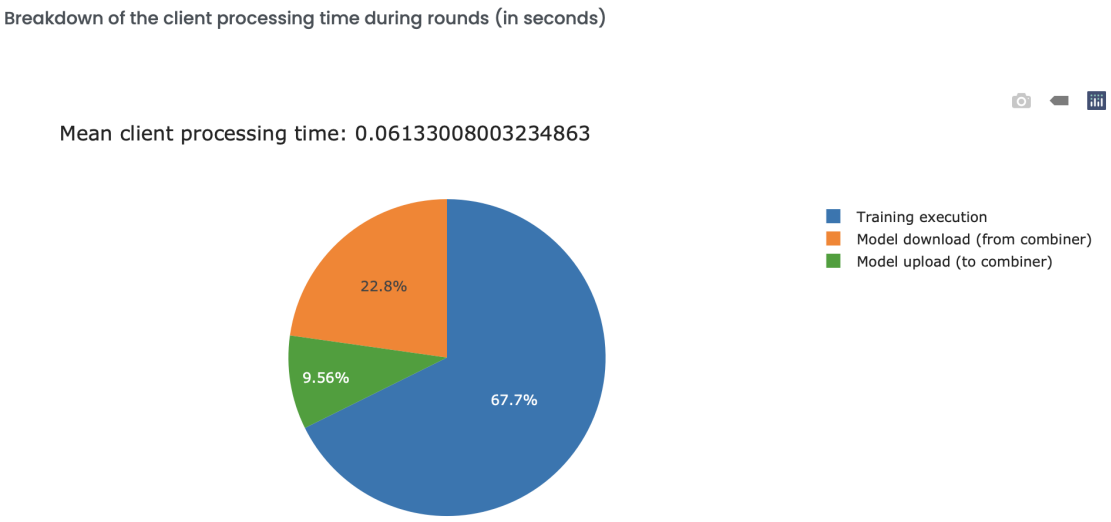

Figure 10: The dashboard

We set up the base services on 192.168.2.36, the combiner is run on 192.168.2.130, and the two clients run on 192.168.2.110.

3.2 Q1

Write a short reflection on how federated learning differs from distributed machine learning. Focus on statistical and system heterogeneity.

In distributed machine learning there is usually a data manager who distributes all data to every client. This data manager can be either a central server or a third party. The division of data on different clients is usually uniform and random. They have the statistical characteristics of the independent and identical distribution. In federated machine learning, each client train model based on their own data. There is no ”data distribution” process, the central server just sends

training missions to clients who are ready to train the model. The clients are usually smart devices, like smartphones, they will collect data locally and train models locally, and then send back the parameters to a central server. The central server does not have access to the data clients have.

Since federated machine learning has clients collect their own data for training, the data usually do not have similar statistical properties due to user behavior. This is different from distributed machine learning.

Another difference is in the system heterogeneity. In distributed machine learning, the computing nodes are typically homogeneous and are coordinated by a centralized master node. In contrast, in federated learning, the computing nodes are often highly heterogeneous in terms of hardware, operating system, and network connectivity. Moreover, in federated learning, the nodes operate autonomously and only communicate with a centralized server during specific phases of the training process.

3.3 Q2

Write a short discussion on how federated learning relates to the Parameter Server strategy for distributed machine learning.

A parameter server training cluster consists of workers and parameter servers. Variables are created on parameter servers and they are read and updated by workers in each step. This strategy relies on a centralized server to store and update model parameters, it can be efficient for some settings, but it can also be a bottleneck if there are many clients sending data to the server at the same time.

On the other hand, in federated learning, each client trains the model locally using its own data and sends only model updates to a central coordinator, the central coordinator aggregates these updates to update the global model and sends the updated model back to the clients for further training.

The relation between parameter server strategy and federated learning is that the structure is similar. For example, in FEDn we have combiners that work as parameter servers and clients that work as workers. This is similar to the parameter server's structure. Also, their pipeline of work is similar. The clients (workers) train the model locally and upload the updates to the parameter servers (combiners), and then the parameter servers (combiners) update the global model.

References

[1] Shen Li **and** Yanli Zhao. "PyTorch Distributed: Experiences on Accelerating Data Parallel Training". **in***arXiv preprint arXiv:2006.15704*: (2020).

[2] Alex Sergeev **andothers**. *FSDP: Enabling Large-scale Data-parallel Training of Image and Language Models with Full-dimensional Sparse Parameters*. `https://engineering.fb.com/2021/07/15/open-source/fsdp/`. Accessed: May 11, 2023. 2021.

[3] Wikipedia. *Data parallelism — Wikipedia, The Free Encyclopedia*. [Online; accessed 10-May-2023]. 2021. URL: `https://en.wikipedia.org/wiki/Data_parallelism`.