Report of Computer Lab 3

Course: Data Engineering II

Uppsala University

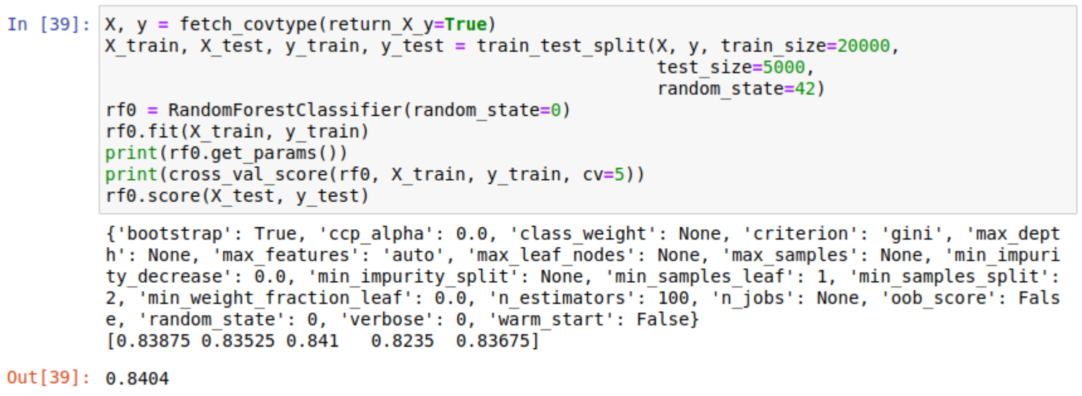
Junjie Chu

Part 1: Distributed machine learning

Task1.1

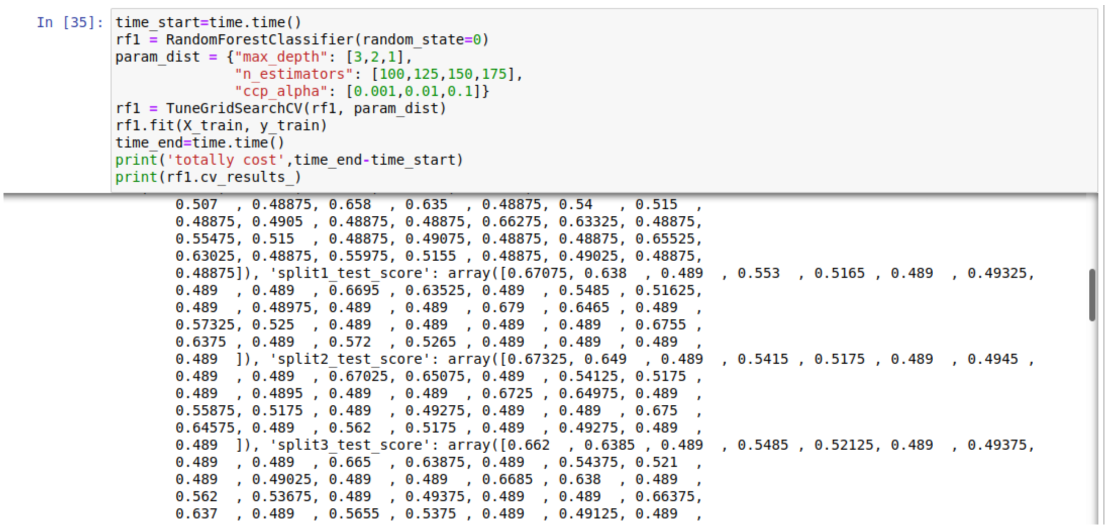
1. The hyperparameters found and the associated cross-validation score. How does it compare to the score for the default parameters?

In this task, when I use the full dataset, it takes about 6 hours to run in a VM. Thus, I use 20000 samples as training data set and 5000 samples as test data set.



```
In [39]: X, y = fetch_covtype(return_X_y=True)
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=20000,
                                                              test_size=5000,
                                                              random_state=42)
         rf0 = RandomForestClassifier(random_state=0)
         rf0.fit(X_train, y_train)
         print(rf0.get_params())
         print(cross_val_score(rf0, X_train, y_train, cv=5))
         rf0.score(X_test, y_test)

         {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_dept
         h': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impuri
         ty_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split':
         2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': Fals
         e, 'random_state': 0, 'verbose': 0, 'warm_start': False}
         [0.83875 0.83525 0.841   0.8235  0.83675]

Out[39]: 0.8404
```

The default parameters could be seen in the picture. The max_depth 'None' means there is no limitation of the depth of a tree. ccp_alpha '0.0' means there is no regularization term. These parameters will cause over-fitting. The score on test dataset is 0.8404 and the cross-validation score is shown in the picture above.

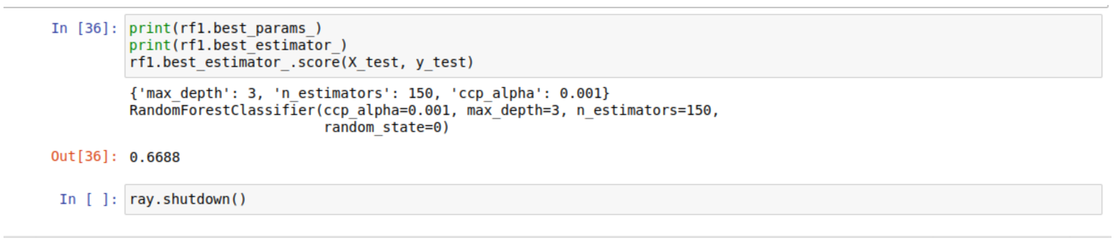


```
In [35]: time_start=time.time()
         rf1 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [3,2,1],
                       "n_estimators": [100,125,150,175],
                       "ccp_alpha": [0.001,0.01,0.1]}
         rf1 = TuneGridSearchCV(rf1, param_dist)
         rf1.fit(X_train, y_train)
         time_end=time.time()
         print('totally cost',time_end-time_start)
         print(rf1.cv_results_)

                0.507  , 0.48875, 0.658  , 0.635  , 0.48875, 0.54   , 0.515  ,
                0.48875, 0.4905 , 0.48875, 0.48875, 0.66275, 0.63325, 0.48875,
                0.55475, 0.515  , 0.48875, 0.49075, 0.48875, 0.48875, 0.65525,
                0.63025, 0.48875, 0.55975, 0.5155 , 0.48875, 0.49025, 0.48875,
                0.48875]), 'split1_test_score': array([0.67075, 0.638  , 0.489  , 0.553  , 0.5165 , 0.489  , 0.49325,
                0.489  , 0.489  , 0.6695 , 0.63525, 0.489  , 0.5485 , 0.51625,
                0.489  , 0.48975, 0.489  , 0.489  , 0.679  , 0.6465 , 0.489  ,
                0.57325, 0.525  , 0.489  , 0.489  , 0.489  , 0.489  , 0.6755 ,
                0.6375 , 0.489  , 0.572  , 0.5265 , 0.489  , 0.489  , 0.489  ,
                0.489  ]), 'split2_test_score': array([0.67325, 0.649  , 0.489  , 0.5415 , 0.5175 , 0.489  , 0.4945 ,
                0.489  , 0.489  , 0.67025, 0.65075, 0.489  , 0.54125, 0.5175 ,
                0.489  , 0.4895 , 0.489  , 0.489  , 0.6725 , 0.64975, 0.489  ,
                0.55875, 0.5175 , 0.489  , 0.49275, 0.489  , 0.489  , 0.675  ,
                0.64575, 0.489  , 0.562  , 0.5175 , 0.489  , 0.49275, 0.489  ,
                0.489  ]), 'split3_test_score': array([0.662  , 0.6385 , 0.489  , 0.5485 , 0.52125, 0.489  , 0.49375,
                0.489  , 0.489  , 0.665  , 0.63875, 0.489  , 0.54375, 0.521  ,
                0.489  , 0.49025, 0.489  , 0.489  , 0.6685 , 0.638  , 0.489  ,
                0.562  , 0.53675, 0.489  , 0.49375, 0.489  , 0.489  , 0.66375,
                0.637  , 0.489  , 0.5655 , 0.5375 , 0.489  , 0.49125, 0.489  ,
```

I remove the option max_depth 'None' and ccp_alpha '0.0'. The candidate parameters and part of the results of cross-validation is shown above.

The parameters of the best estimator is in the following pictures as well as its score on test dataset.

The accuracy is no as good as the default settings. But I would like to take it. Because it avoids over-fitting by adding ccp_alpha and limiting the depth of a tree. If I could run the whole dataset in the future, the result will be better.



```
In [36]: print(rf1.best_params_)
         print(rf1.best_estimator_)
         rf1.best_estimator_.score(X_test, y_test)

         {'max_depth': 3, 'n_estimators': 150, 'ccp_alpha': 0.001}
         RandomForestClassifier(ccp_alpha=0.001, max_depth=3, n_estimators=150,
                                random_state=0)

Out[36]: 0.6688

In [ ]: ray.shutdown()
```

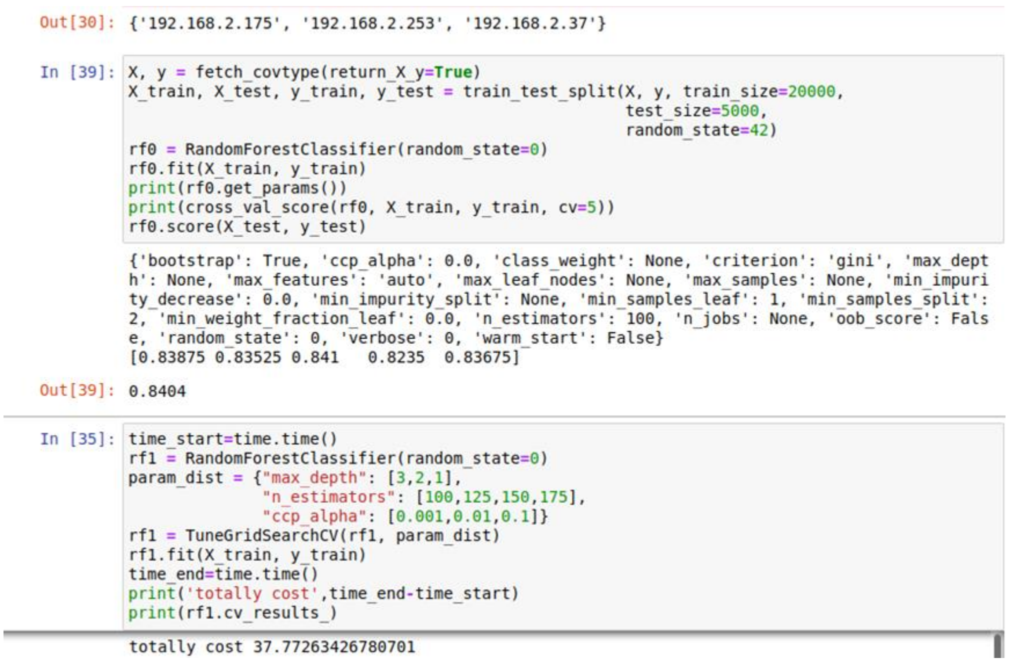
The all code the results of cross-validation could be seen via the following link.

https://github.com/Junjie-Chu/DataEngineering2/blob/main/a3/task1.1.ipynb

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

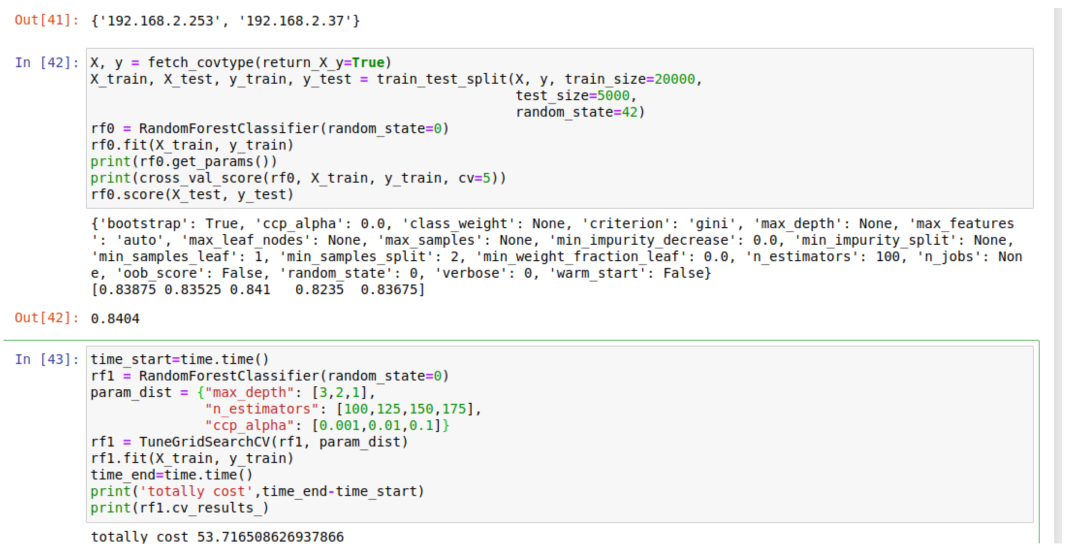
Size of training dataset = 20000, test dataset = 5000.

When using 3 VMs:



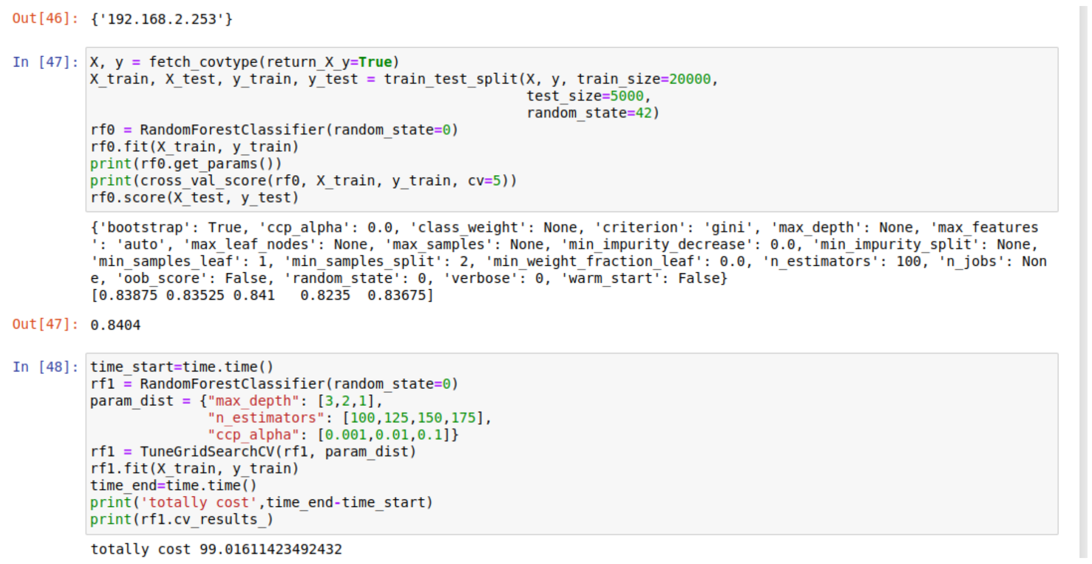
```
Out[30]: {'192.168.2.175', '192.168.2.253', '192.168.2.37'}

In [39]: X, y = fetch_covtype(return_X_y=True)
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=20000,
                                                             test_size=5000,
                                                             random_state=42)
         rf0 = RandomForestClassifier(random_state=0)
         rf0.fit(X_train, y_train)
         print(rf0.get_params())
         print(cross_val_score(rf0, X_train, y_train, cv=5))
         rf0.score(X_test, y_test)

         {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_dept
         h': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impuri
         ty_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split':
         2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': Fals
         e, 'random_state': 0, 'verbose': 0, 'warm_start': False}
         [0.83875 0.83525 0.841   0.8235  0.83675]

Out[39]: 0.8404

In [35]: time_start=time.time()
         rf1 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [3,2,1],
                       "n_estimators": [100,125,150,175],
                       "ccp_alpha": [0.001,0.01,0.1]}
         rf1 = TuneGridSearchCV(rf1, param_dist)
         rf1.fit(X_train, y_train)
         time_end=time.time()
         print('totally cost',time_end-time_start)
         print(rf1.cv_results_)

         totally cost 37.77263426780701
```

When using 2 VMs:



```
Out[41]: {'192.168.2.253', '192.168.2.37'}

In [42]: X, y = fetch_covtype(return_X_y=True)
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=20000,
                                                             test_size=5000,
                                                             random_state=42)
         rf0 = RandomForestClassifier(random_state=0)
         rf0.fit(X_train, y_train)
         print(rf0.get_params())
         print(cross_val_score(rf0, X_train, y_train, cv=5))
         rf0.score(X_test, y_test)

         {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features
         ': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
         'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': Non
         e, 'oob_score': False, 'random_state': 0, 'verbose': 0, 'warm_start': False}
         [0.83875 0.83525 0.841   0.8235  0.83675]

Out[42]: 0.8404

In [43]: time_start=time.time()
         rf1 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [3,2,1],
                       "n_estimators": [100,125,150,175],
                       "ccp_alpha": [0.001,0.01,0.1]}
         rf1 = TuneGridSearchCV(rf1, param_dist)
         rf1.fit(X_train, y_train)
         time_end=time.time()
         print('totally cost',time_end-time_start)
         print(rf1.cv_results_)

         totally cost 53.716508626937866
```

When using 1 VM:



```
Out[46]: {'192.168.2.253'}

In [47]: X, y = fetch_covtype(return_X_y=True)
         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=20000,
                                                             test_size=5000,
                                                             random_state=42)
         rf0 = RandomForestClassifier(random_state=0)
         rf0.fit(X_train, y_train)
         print(rf0.get_params())
         print(cross_val_score(rf0, X_train, y_train, cv=5))
         rf0.score(X_test, y_test)

         {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features
         ': 'auto', 'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None,
         'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': Non
         e, 'oob_score': False, 'random_state': 0, 'verbose': 0, 'warm_start': False}
         [0.83875 0.83525 0.841   0.8235  0.83675]

Out[47]: 0.8404

In [48]: time_start=time.time()
         rf1 = RandomForestClassifier(random_state=0)
         param_dist = {"max_depth": [3,2,1],
                       "n_estimators": [100,125,150,175],
                       "ccp_alpha": [0.001,0.01,0.1]}
         rf1 = TuneGridSearchCV(rf1, param_dist)
         rf1.fit(X_train, y_train)
         time_end=time.time()
         print('totally cost',time_end-time_start)
         print(rf1.cv_results_)

         totally cost 99.01611423492432
```

The time spent decreases when the number of VMs increases. When using 3 VMs, the speed-up is nearly 2.67. The Ray-tuning has a good strong scalability.

Task1.2

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?

Difference:

Data parallel training strategies in a distributed training setting is when each replica trains on the same model but over different data. After every few iterations, all replicas synchronize, either with one-another (all-reduce) or via a central server (parameter server). This usually scales up nicely and one can also see some algorithmic speedup due to averaging. Model parallel training strategies is when each replica trains over same data but uses different part of the model. In this case, we need to consider a bit more since (a) the model needs to be large enough (memory) to justify going over the network (b) the split needs to be careful enough such that there the computation/communication is reasonable.

In conclusion, data parallel strategies mean the same model on different nodes but different batches of data on them while model parallel strategies mean that the same data on the nodes but different parts of a model used.

Situations :

If the worker nodes do not have public memory, and the local memory capacity is limited, and the amount of training data is large and cannot be stored in the local memory, we need to divide the data set and assign it to each worker node. The worker nodes are based on their own local data for model training. That is to use "data parallel mode".

Model parallelism is suitable for scenarios where the model itself or part of it is large, and the training acceleration effect of pure data parallelism is not good or pure data parallelism cannot be performed.

For example, the following similar scenario:

- Scenarios with extremely large parameters such as GPT2 and T5 in the NLP field. Because the memory consumption of the model is too large, pure data parallelism cannot be trained.
- Scenarios with large parameters such as BertLarge in the NLP field. There is a Feature Map (Activation) between layers in the model, and the amount of communication is much smaller than that of Weights.
- Taking the VGG-16 model as an example, the video memory of each layer is unbalanced, and pure data parallelism will cause a large waste of video memory.

Situations when doing hyperparameter tuning:

The task1.1 I think is a task parallelism. The whole dataset and the whole model are used in all worker nodes (but the parameters of models are different). This method works when the dataset and model is not very large.

When the size of dataset is very large and model is not that big, we could use a data parallel method. We could combine data parallelism and task parallelism together. For example, there are many candidate models, and we group them into N groups. There are N tasks in total. And when approaching each task, we use data parallelism and several worker nodes.

I think when I face a very large model, I will face such challenges:

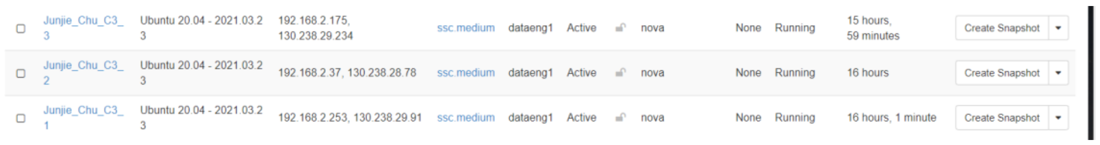
- ➢ The distributed training gradient synchronization communication volume increases.
- ➢ Limited by the GPU memory, the training batch size becomes smaller, which leads to an increase in the proportion of training communication and a decrease in the efficiency of distributed expansion.
- ➢ A too small Batch Size will cause the model training to fluctuate greatly and affect the convergence effect.

In this case, I will consider distributing the training of individual neural networks when I try to do its hyperparameter tuning.

Part 2:

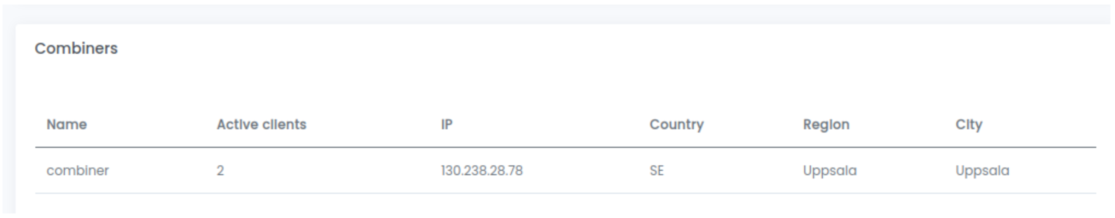
Results of fully distributed federated training:

The VMs I used:



☐	Junjie_Chu_C3_3	Ubuntu 20.04 - 2021.03.23	192.168.2.175, 130.238.29.234	ssc.medium	dataeng1	Active		nova	None	Running	15 hours, 59 minutes	Create Snapshot
☐	Junjie_Chu_C3_2	Ubuntu 20.04 - 2021.03.23	192.168.2.37, 130.238.28.78	ssc.medium	dataeng1	Active		nova	None	Running	16 hours	Create Snapshot
☐	Junjie_Chu_C3_1	Ubuntu 20.04 - 2021.03.23	192.168.2.253, 130.238.29.91	ssc.medium	dataeng1	Active		nova	None	Running	16 hours, 1 minute	Create Snapshot

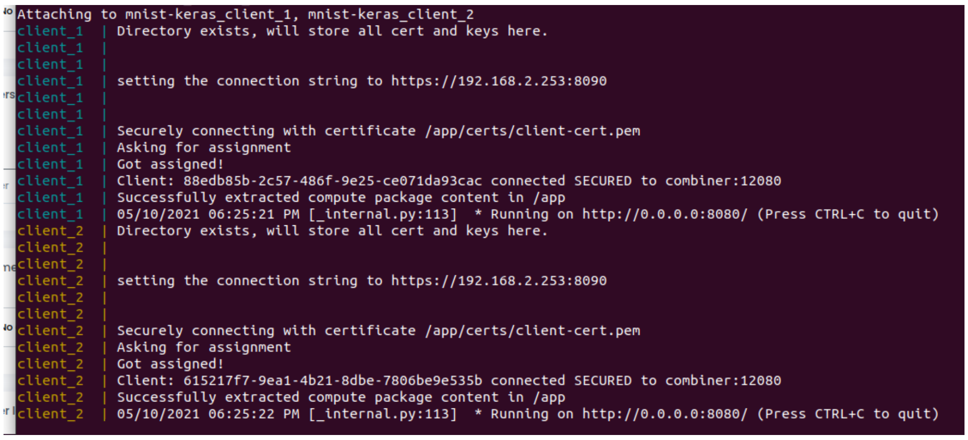
Basic service and Reducer run on 192.168.2.253. Combiner runs on 192.168.2.37. Clients run on 192.168.2.175. The situation of them could be seen in the following picture.



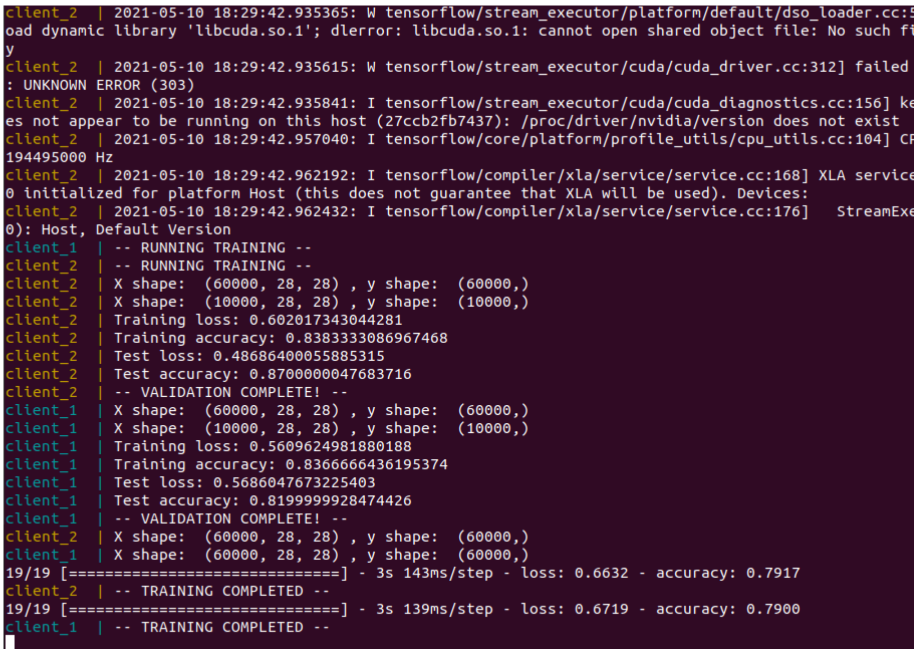
Combiners

Name	Active clients	IP	Country	Region	City
combiner	2	130.238.28.78	SE	Uppsala	Uppsala

The logs in clients:

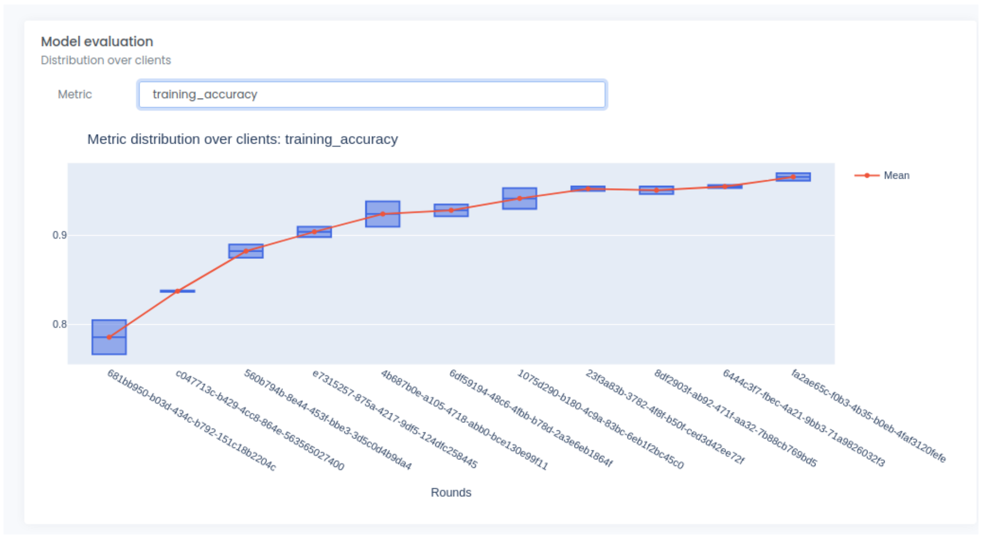


```
Attaching to mnist-keras_client_1, mnist-keras_client_2
client_1  | Directory exists, will store all cert and keys here.
client_1  |
client_1  |
client_1  | setting the connection string to https://192.168.2.253:8090
client_1  |
client_1  |
client_1  | Securely connecting with certificate /app/certs/client-cert.pem
client_1  | Asking for assignment
client_1  | Got assigned!
client_1  | Client: 88edb85b-2c57-486f-9e25-ce071da93cac connected SECURED to combiner:12080
client_1  | Successfully extracted compute package content in /app
client_1  | 05/10/2021 06:25:21 PM [_internal.py:113]  * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
client_2  | Directory exists, will store all cert and keys here.
client_2  |
client_2  |
client_2  | setting the connection string to https://192.168.2.253:8090
client_2  |
client_2  |
client_2  | Securely connecting with certificate /app/certs/client-cert.pem
client_2  | Asking for assignment
client_2  | Got assigned!
client_2  | Client: 615217f7-9ea1-4b21-8dbe-7806be9e535b connected SECURED to combiner:12080
client_2  | Successfully extracted compute package content in /app
client_2  | 05/10/2021 06:25:22 PM [_internal.py:113]  * Running on http://0.0.0.0:8080/ (Press CTRL+C to quit)
```



```
client_2  | 2021-05-10 18:29:42.935365: W tensorflow/stream_executor/platform/default/dso_loader.cc:5
oad dynamic library 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such fi
y
client_2  | 2021-05-10 18:29:42.935615: W tensorflow/stream_executor/cuda/cuda_driver.cc:312] failed
: UNKNOWN ERROR (303)
client_2  | 2021-05-10 18:29:42.935841: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] ke
es not appear to be running on this host (27ccb2fb7437): /proc/driver/nvidia/version does not exist
client_2  | 2021-05-10 18:29:42.957040: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104] CP
194495000 Hz
client_2  | 2021-05-10 18:29:42.962192: I tensorflow/compiler/xla/service/service.cc:168] XLA service
0 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
client_2  | 2021-05-10 18:29:42.962432: I tensorflow/compiler/xla/service/service.cc:176]   StreamExe
0): Host, Default Version
client_1  | -- RUNNING TRAINING --
client_2  | -- RUNNING TRAINING --
client_2  | X shape:  (60000, 28, 28) , y shape:  (60000,)
client_2  | X shape:  (10000, 28, 28) , y shape:  (10000,)
client_2  | Training loss: 0.602017343044281
client_2  | Training accuracy: 0.8383333086967468
client_2  | Test loss: 0.48686400055885315
client_2  | Test accuracy: 0.8700000047683716
client_2  | -- VALIDATION COMPLETE! --
client_1  | X shape:  (60000, 28, 28) , y shape:  (60000,)
client_1  | X shape:  (10000, 28, 28) , y shape:  (10000,)
client_1  | Training loss: 0.5609624981880188
client_1  | Training accuracy: 0.8366666436195374
client_1  | Test loss: 0.5686047673225403
client_1  | Test accuracy: 0.8199999928474426
client_1  | -- VALIDATION COMPLETE! --
client_2  | X shape:  (60000, 28, 28) , y shape:  (60000,)
client_1  | X shape:  (60000, 28, 28) , y shape:  (60000,)
19/19 [==============================] - 3s 143ms/step - loss: 0.6632 - accuracy: 0.7917
client_2  | -- TRAINING COMPLETED --
19/19 [==============================] - 3s 139ms/step - loss: 0.6719 - accuracy: 0.7900
client_1  | -- TRAINING COMPLETED --
```

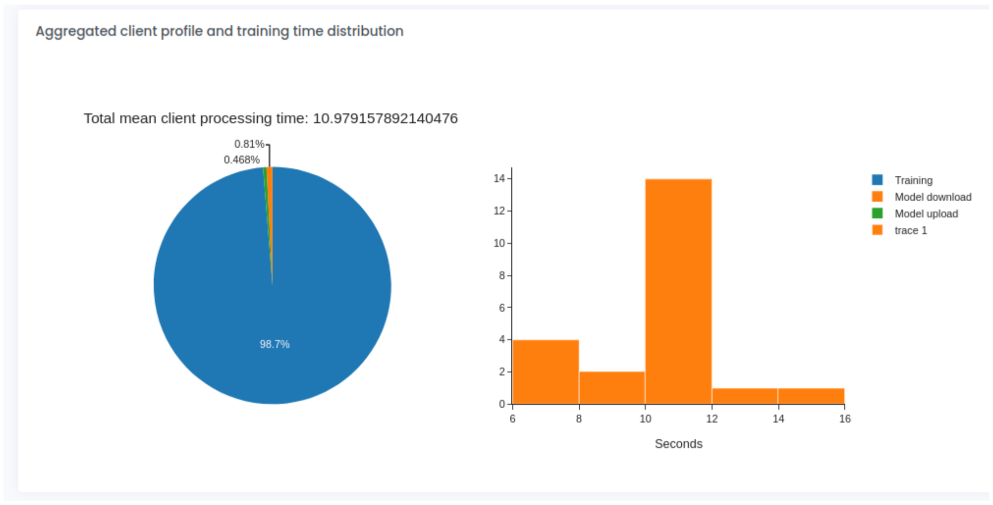
The results in dashboard(10 rounds):



Model evaluation

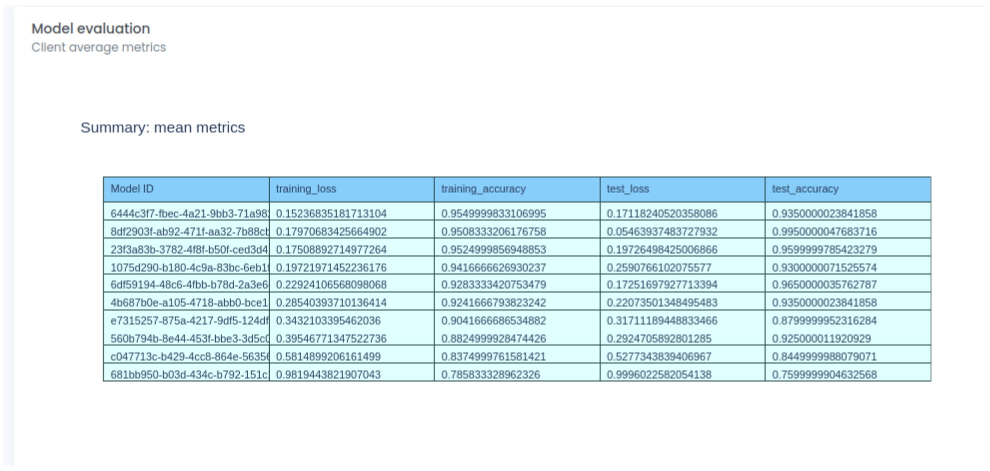
Distribution over clients

Metric: training_accuracy



Aggregated client profile and training time distribution

Total mean client processing time: 10.979157892140476


Model evaluation
Client average metrics

Summary: mean metrics

Model ID	training_loss	training_accuracy	test_loss	test_accuracy
6444c3f7-fbec-4a21-9bb3-71a98	0.15236835181713104	0.9549999833106995	0.17118240520358086	0.9350000023841858
8df2903f-ab92-471f-aa32-7b88c	0.17970683425664902	0.9508333206176758	0.05463937483727932	0.9950000047683716
23f3a83b-3782-4f8f-b50f-ced3d4	0.17508892714977264	0.9524999856948853	0.19726498425006866	0.9599999785423279
1075d290-b180-4c9a-83bc-6eb1	0.19721971452236176	0.9416666626930237	0.2590766102075577	0.9300000071525574
6df59194-48c6-4fbb-b78d-2a3e6	0.22924106568098068	0.9283333420753479	0.17251697927713394	0.9650000035762787
4b687b0e-a105-4718-abb0-bce1	0.28540393710136414	0.9241666793823242	0.22073501348495483	0.9350000023841858
e7315257-875a-4217-9df5-124df 560b794b-8e44-453f-bbe3-3d5c0	0.3432103395462036 0.39546771347522736	0.9041666686534882 0.8824999928474426	0.31711189448833466 0.2924705892801285	0.8799999952316284 0.925000011920929
c047713c-b429-4cc8-864e-5635	0.5814899206161499	0.8374999761581421	0.5277343839406967	0.8449999988079071
681bb950-b03d-434c-b792-151c	0.9819443821907043	0.785833328962326	0.9996022582054138	0.7599999904632568

1. **Write a short (max 0.5 page Arial 11pt) reflection on how federated learning differs from distributed machine learning. Focus on statistical and system heterogeneity.**

Different types of data distribution on computing nodes. In distributed learning, the division of data on different computing nodes is usually uniform and random. They have the statistical characteristics of independent and identical distribution. Such characteristics are very suitable for designing efficient training algorithms. In federated learning, it is not simply assumed that the data are independent and identically distributed. Since the data in the computing nodes are generated independently, they often show different distribution characteristics.

The central server has different control over data and computing nodes. The central server of distributed learning has high control over the computing nodes and the data therein. The computing nodes are completely controlled by the central server and receive instructions from the central server. But in federated learning, the central server cannot directly or indirectly manipulate the data on the computing node; the computing node can stop computing and communication at any time, and exit the learning process.

The magnitude of the data on the computing node is different. Distributed learning usually distributes training data evenly on each computing node to achieve load balancing. However, under the condition of federated learning, the amount of data owned by each computing node is related to the device itself.

The stability of computing nodes is different. Computing nodes in distributed learning scenarios are usually stable (the same or similar hardware, software and environment). But those in federated learning sometimes are just mobile phones and not stable (different hardware, software and environment). The cost of communication is more expensive in federated learning.

2. **Write a short (max 0.5 page Arial 11pt) discussion on how federated learning relates to the Parameter Server strategy for distributed machine learning.**

Relations:

The most important relation between them is that in both of them, only the model parameters are stored in the servers and there is no need to exchange data. Another similarity is that the computing nodes could get models from the servers. And the servers could help us to integrate global models. Their architecture and framework are similar, too. In parameter server strategy, there are master, parameter servers and workers. In federated learning, for example in *fedn,* there are reducer, combiners and clients. Their functions are very similar. And their pipeline of working is also similar. The clients(worker) train the model locally and upload the updates to the parameter servers(combiner), and then the parameter servers(combiner) update the global model. The master(reducer) manages the system.

The difference is that in typical federated learning, the server will average all the updates to get a global model after all clients upload their own updates(allow some clients to fail or exit). In addition to transferring model updates to the cloud, the updated model in each client can also be used immediately on itself. But in the typical parameter server strategy, the overall weight can be updated immediately after one computing node uploads its update (asynchronous). But in some cases of parameter server strategy, the update of global models can also be synchronous. And recent federated learning developments introduced novel techniques to tackle asynchronicity during the training process, or training with dynamically varying models. So, the difference does not always exist and the parameter server strategy could be used in the federated learning in some cases. In addition, there are some other differences like types of data sets they use and computing nodes in the framework, which are included in question 1. We could implement a federated learning framework by using a modified parameter server strategy.