Πολυτεχνική Σχολή
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Δ.Π.Μ.Σ. «ΥΠΟΛΟΓΙΣΤΙΚΗ ΔΕΔΟΜΕΝΩΝ ΚΑΙ ΑΠΟΦΑΣΕΩΝ – Υ.Δ.Α.»

Υπολογιστική Υψηλών Επιδόσεων Επιστήμης Δεδομένων

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΑΚΗΣ ΑΣΚΗΣΗΣ

Παράλληλη εφαρμογή sklearn scalers σε h5 αρχεία

Σκόνδρας Γεώργιος
Α.Μ. 1020408
Πάτρα, 2023

Περιεχόμενα

1. Περιβάλλον Υλοποίησης **2**
2. Δημιουργία Δεδομένων **2**
3. Standard Scaler **2**
4. MinMax Scaler **4**
5. Πειράματα **6**

1. Περιβάλλον Υλοποίησης

Η υλοποίηση και τα πειράματα έγιναν σε υπολογιστικό σύστημα με επεξεργαστή Intel Core i7-10700 με 8 cores, 16 threads και 16GB RAM.

2. Δημιουργία Δεδομένων

Με το script create_data.py δημιουργώ ένα μεγάλο αρχείο h5 με 30 εκατομμύρια γραμμές και 30 στήλες με τυχαίους αριθμούς στο διάστημα [0,1) όπως φαίνεται παρακάτω.

```
import numpy as np
import h5py

np.random.seed(0)
data = np.random.rand(30000000, 30)

hf = h5py.File("data.h5", "w")
hf.create_dataset('data', data=data)
hf.close()
```

Το αρχείο αυτό έχει μέγεθος 7GB.

3. Standard Scaler

Στο script std_serial.py εφαρμόζω τον `StandardScaler` της sklearn, μέσω της συνάρτησης fit, όπου σειριακά εφαρμόζεται σε όλα τα δεδομένα και βλέπω τα αποτελέσματα.

```
(.venv) up1020408@preveza:~/HPC-course/Python_Project$ python3 std_serial.py
========================================
==> StandardScaler <==
scaler.with_mean = True
scaler.with_std = True
scaler.copy = True
scaler.n_features_in_ = 30
scaler.n_samples_seen_ = 30000000
scaler.mean_ = [0.49993414 0.5000027  0.49995984 0.49993439 0.50003678 0.50010085
 0.50002603 0.49997509 0.49999886 0.49990988 0.50002849 0.49992265
 0.49994004 0.49998343 0.50002264 0.4999773  0.49995751 0.49996623
 0.50007075 0.50002421 0.49998467 0.5000447  0.49995382 0.49999051
 0.50004799 0.50001355 0.49991992 0.49996398 0.50006247 0.50006267]
scaler.var_ = [0.08333453 0.08335434 0.08332879 0.08332436 0.08332719 0.08333688
 0.08334524 0.08333801 0.08332206 0.08333113 0.08333486 0.08333581
 0.08333384 0.08333173 0.08331552 0.08333427 0.08333015 0.08332517
 0.08334942 0.08334127 0.08333474 0.0833275  0.08334017 0.08333015
 0.08332401 0.08332324 0.08331502 0.08332242 0.08335184 0.08335361]
scaler.scale_ = [0.2886772  0.28871152 0.28866726 0.28865959 0.28866449 0.28868128
 0.28869576 0.28868323 0.2886556  0.28867132 0.28867778 0.28867942
 0.28867601 0.28867235 0.28864428 0.28867675 0.28866962 0.288661
 0.288703   0.28868889 0.28867757 0.28866503 0.28868698 0.28866963
 0.28865898 0.28865766 0.28864342 0.28865623 0.28870718 0.28871025]
```

Για τον παράλληλο `StandardScaler` στο φάκελο modules.parallel_scalers στο script parallelScaler.py δημιουργώ την κλάση `ParStandardScaler` που κληρονομεί την `StandardScaler` της sklearn. Εκεί ορίζω τη συνάρτηση `parallel_fit` η οποία ανοίγει το αρχείο που της δίνουμε σαν όρισμα και αφού διαβάσει το μέγεθος του αρχείου, υπολογίζει πόσο είναι το batch_size που θα επεξεργαστεί ο κάθε worker(ο αριθμός των workers δίνεται σαν όρισμα στην parallel_fit) και πόσες θα είναι οι εργασίες που θα πρέπει να κάνουν οι workers. Στη συνέχεια υπολογίζω τους δείκτες στο αρχείο που θα πρέπει να επεξεργαστεί ο κάθε worker και με τη βοήθεια του `ProcessPoolExecutor` της `concurrent.futures` μοιράζω τις εργασίες στα processes.

```
attributs = [(start*batch_size,(start*batch_size)+batch_size, data_file) for start in
range(iterations)]

        with ProcessPoolExecutor(max_workers=num_workers) as executor:
            scalers = executor.map(work_min_max, attributs)
```

Η συνάρτηση `work_std` αποτελεί τη δουλειά που μοιραζεται στα processes, όπου κάθε process ανοίγει το αρχείο, εφαρμόζει τον standard scaler με `partial_fit` στο κομμάτι των δεδομένων που του δίνουν οι παράμετροι start και end και επιστρέφει τον scaler, όπως βλέπουμε παρακάτω:

```
def work_std(attributs):
    start = attributs[0]
    end = attributs[1]
    data_file = attributs[2]
    data_file = "./" + data_file
    hf = h5py.File(data_file, "r")
    data = hf["data"]
    partial_data = data[start:end]
    scaler = preprocessing.StandardScaler()
    scaler.partial_fit(partial_data)

    return scaler
```

Μόλις ολοκληρώσουν τη δουλειά οι workers, ο αρχικός συναθροίζει τη λίστα με όλους τους scalers σε έναν με τη συνάρτηση `reduce_min_max` που βρίσκεται στο φάκελο `modules.utils.utils_StandardScaler`

```
scalers = list(scalers)
final_scaler = reduce_min_max(scalers)
```

Ελέγχω ότι τα αποτελέσματα τρέχοντας το script std_parallel.py και βλέπω πως είναι ίδια με το σειριακό.

```
(.venv) up1020408@preveza:~/HPC-course/Python_Project$ python3 std_parallel.py
Parallel fitting Standard Scaler using
Workers:8
batch_size:3750000

==> StandardScaler <==
scaler.with_mean = True
scaler.with_std = True
scaler.copy = True
scaler.n_features_in_ = 30
scaler.n_samples_seen_ = 30000000
scaler.mean_ = [0.49993414 0.5000027  0.49995984 0.49993439 0.50003678 0.50010085
 0.50002603 0.49997509 0.49999886 0.49990988 0.50002849 0.49992265
 0.49994004 0.49998343 0.50002264 0.4999773  0.49995751 0.49996623
 0.50007075 0.50002421 0.49998467 0.5000447  0.49995382 0.49999051
 0.50004799 0.50001355 0.49991992 0.49996398 0.50006247 0.50006267]
scaler.var_ = [0.08333453 0.08335434 0.08332879 0.08332436 0.08332719 0.08333688
 0.08334524 0.08333801 0.08332206 0.08333113 0.08333486 0.08333581
 0.08333384 0.08333173 0.08331552 0.08333427 0.08333015 0.08332517
 0.08334942 0.08334127 0.08333474 0.0833275  0.08334017 0.08333015
 0.08332401 0.08332324 0.08331502 0.08332242 0.08335184 0.08335361]
scaler.scale_ = [0.2886772  0.28871152 0.28866726 0.28865959 0.28866449 0.28868128
 0.28869576 0.28868323 0.2886556  0.28867132 0.28867778 0.28867942
 0.28867601 0.28867235 0.28864428 0.28867675 0.28866962 0.288661
 0.288703   0.28868889 0.28867757 0.28866503 0.28868698 0.28866963
 0.28865898 0.28865766 0.28864342 0.28865623 0.28870718 0.28871025]
```

4. MinMax Scaler

Για τον MinMax Scaler η διαδικασία είναι η αντίστοιχη με τον StandardScaler.
Στο script min_max_serial.py εφαρμόζω τον `MinMaxScaler` της sklearn, μέσω της συνάρτησης fit, οπου σειριακά εφαρμόζεται σε όλα τα δεδομένα και βλέπω τα αποτελέσματα.

```
==> MinMaxScaler <==
scaler.feature_range = (0, 1)
scaler.copy = True
scaler.clip = False
scaler.n_features_in_ = 30
scaler.n_samples_seen_ = 30000000
scaler.scale_ = [1.00000005 1.00000006 1.00000009 1.00000007 1.00000014 1.00000009
 1.00000011 1.00000009 1.00000007 1.00000001 1.0000001  1.
 1.00000013 1.00000008 1.00000006 1.00000009 1.00000012 1.00000003
 1.00000002 1.00000009 1.00000009 1.00000016 1.00000007 1.0000001
 1.0000001  1.00000004 1.00000007 1.00000016 1.00000008 1.00000007]
scaler.min_ = [-1.63184451e-08 -1.47698340e-09 -8.26149250e-08 -5.06891744e-08
 -9.78644490e-08 -5.62248088e-09 -4.22000197e-08 -1.00064160e-08
 -6.09535723e-08 -4.07807138e-09 -1.24197856e-08 -3.77923272e-09
 -8.16063812e-08 -2.99144962e-08 -1.03619218e-08 -1.98050513e-08
 -5.63027899e-08 -2.74123252e-08 -1.18771972e-08 -8.58136920e-08
 -3.91395431e-09 -1.41618170e-07 -6.90709307e-08 -7.84695300e-08
 -8.03827416e-08 -1.06329557e-08 -4.34283192e-08 -1.12734128e-07
 -5.21057684e-08 -5.47858469e-08]
scaler.data_min_ = [1.63184443e-08 1.47698331e-09 8.26149171e-08 5.06891707e-08
 9.78644354e-08 5.62248037e-09 4.22000150e-08 1.00064151e-08
 6.09535681e-08 4.07807133e-09 1.24197844e-08 3.77923270e-09
 8.16063708e-08 2.99144938e-08 1.03619212e-08 1.98050495e-08
 5.63027832e-08 2.74123243e-08 1.18771970e-08 8.58136842e-08
 3.91395394e-09 1.41618148e-07 6.90709255e-08 7.84695223e-08
 8.03827339e-08 1.06329553e-08 4.34283163e-08 1.12734110e-07
 5.21057643e-08 5.47858430e-08]
scaler.data_max_ = [0.99999996 0.99999994 0.99999999 0.99999998 0.99999996 0.99999991
 0.99999993 0.99999992 0.99999999 0.99999999 0.99999992 1.
 0.99999995 0.99999995 0.99999995 0.99999993 0.99999994 0.99999999
 0.99999999 0.99999999 0.99999991 0.99999999 0.99999999 0.99999998
 0.99999998 0.99999997 0.99999998 0.99999995 0.99999997 0.99999998]
scaler.data_range_ = [0.99999995 0.99999994 0.99999991 0.99999993 0.99999986 0.99999991
 0.99999989 0.99999991 0.99999993 0.99999999 0.9999999  1.
 0.99999987 0.99999992 0.99999994 0.99999991 0.99999988 0.99999997
 0.99999998 0.99999991 0.99999991 0.99999984 0.99999993 0.9999999
 0.9999999  0.99999996 0.99999993 0.99999984 0.99999992 0.99999993]
```

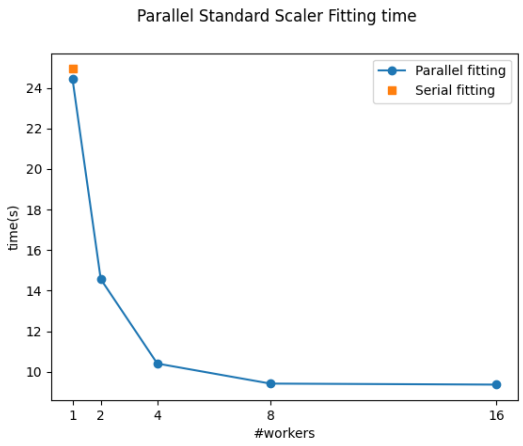
Για τον παράλληλο `MinMaxScaler` στο φάκελο modules.parallel_scalers στο script parallelScaler.py δημιουργώ την κλάση `ParMinMaxScaler` που κληρονομεί την `StandardScaler` της sklearn. Εκεί ορίζω τη συνάρτηση `parallel_fit` με παρόμοια λειτουργία με αυτή του παράλληλου standard scaler. Τώρα η συνάρτηση `work_min_max` αποτελεί τη δουλειά που μοιραζεται στα processes. Με παρόμοιο τρόπο όπως και πριν συναθροίζονται όλοι οι scalers.

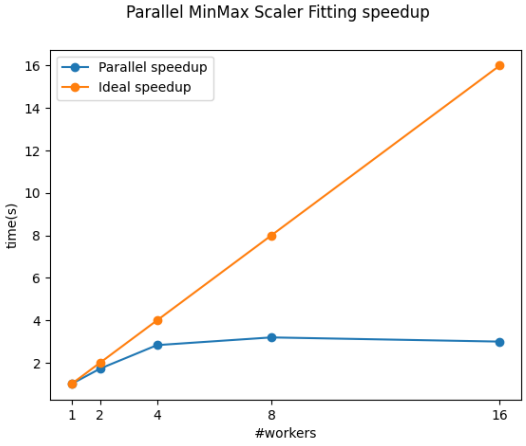
Ελέγχω ότι τα αποτελέσματα τρέχοντας το script min_max_parallel.py και βλέπω πως είναι ίδια με το σειριακό.

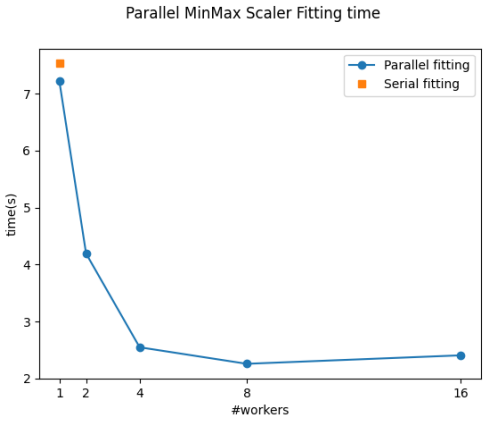
```
==> MinMaxScaler <==
scaler.feature_range = (0, 1)
scaler.copy = True
scaler.clip = False
scaler.n_features_in_ = 30
scaler.n_samples_seen_ = 30000000
scaler.scale_ = [1.00000005 1.00000006 1.00000009 1.00000007 1.00000014 1.00000009
 1.00000011 1.00000009 1.00000007 1.00000001 1.0000001  1.
 1.00000013 1.00000008 1.00000006 1.00000009 1.00000012 1.00000003
 1.00000002 1.00000009 1.00000009 1.00000016 1.00000007 1.0000001
 1.0000001  1.00000004 1.00000007 1.00000016 1.00000008 1.00000007]
scaler.min_ = [-1.63184451e-08 -1.47698340e-09 -8.26149250e-08 -5.06891744e-08
 -9.78644490e-08 -5.62248088e-09 -4.22000197e-08 -1.00064160e-08
 -6.09535723e-08 -4.07807138e-09 -1.24197856e-08 -3.77923272e-09
 -8.16063812e-08 -2.99144962e-08 -1.03619218e-08 -1.98050513e-08
 -5.63027899e-08 -2.74123252e-08 -1.18771972e-08 -8.58136920e-08
 -3.91395431e-09 -1.41618170e-07 -6.90709307e-08 -7.84695300e-08
 -8.03827416e-08 -1.06329557e-08 -4.34283192e-08 -1.12734128e-07
 -5.21057684e-08 -5.47858469e-08]
scaler.data_min_ = [1.63184443e-08 1.47698331e-09 8.26149171e-08 5.06891707e-08
 9.78644354e-08 5.62248037e-09 4.22000150e-08 1.00064151e-08
 6.09535681e-08 4.07807133e-09 1.24197844e-08 3.77923270e-09
 8.16063708e-08 2.99144938e-08 1.03619212e-08 1.98050495e-08
 5.63027832e-08 2.74123243e-08 1.18771970e-08 8.58136842e-08
 3.91395394e-09 1.41618148e-07 6.90709255e-08 7.84695223e-08
 8.03827339e-08 1.06329553e-08 4.34283163e-08 1.12734110e-07
 5.21057643e-08 5.47858430e-08]
scaler.data_max_ = [0.99999996 0.99999994 0.99999999 0.99999998 0.99999996 0.99999991
 0.99999993 0.99999992 0.99999999 0.99999999 0.99999992 1.
 0.99999995 0.99999995 0.99999995 0.99999993 0.99999994 0.99999999
 0.99999999 0.99999999 0.99999991 0.99999999 0.99999999 0.99999998
 0.99999998 0.99999997 0.99999998 0.99999995 0.99999997 0.99999998]
scaler.data_range_ = [0.99999995 0.99999994 0.99999991 0.99999993 0.99999986 0.99999991
 0.99999989 0.99999991 0.99999993 0.99999999 0.9999999  1.
 0.99999987 0.99999992 0.99999994 0.99999991 0.99999988 0.99999997
 0.99999998 0.99999991 0.99999991 0.99999984 0.99999993 0.9999999
 0.9999999  0.99999996 0.99999993 0.99999984 0.99999992 0.99999993]
```

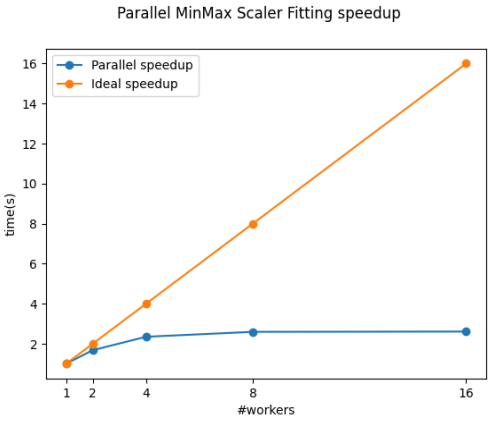
5. Πειράματα

Τα πειράματα γίνονται με το script experiments.py. Εκεί εκτελώ χρονομετρήσεις για την εφαρμογή των παράλληλων scalers για αριθμό workers 1, 2 ,4 ,8, 16. Κάθε πείραμα επαναλαμβάνεται 10 φορές και λαμβάνεται ο μέσος όρος της κάθε χρονομέτρησης. Παρακάτω βλέπουμε τις γραφικές απεικονίσεις των χρόνων εκτέλεσης και του speedup για τον παράλληλο MinMaxScaler και StandardScaler.









Παρατηρούμε και στις 2 περιπτώσεις καλή επιτάχυνση για 2 και 4 workers. Για περισσότερους workers δεν παρατηρούμε γρηγορότερη εκτέλεση.