Dokumentacja

Etap 3

Plik `.csv` z zebranymi danymi znajduje się w `output/times.csv`. Jest on generowany przy pomocy skryptu `output/times.sh`, który pobiera wyniki cząstkowe z Google Storage, a następnie *skleja* je w całość. W pliku `.csv` znajduje się 120 rekordów. Nie ma w nim danych dla modelu LR z biblioteki synapse-ml, ponieważ biblioteka go nie zawiera i nie był on też badany we wcześniejszym etapie. Wykresy są generowane przy użyciu skryptu `output/plot.py`.

W ramach jednego uruchomienia joba Sparkowego testowane były oba modele dla trzech bibliotek (z wyjątkiem LR z synapse-ml jak wspomniano wcześniej), po 3 powtórzenia każdy. Po każdym uruchonieniu należało manualnie zmienić liczbę egzekutorów w `src/k8s/pyspark_job.yaml.tpl` lub `modules/dataproc-pyspark-job/main.tf` (w zależności od testowanej platformy), oraz w `src/py/pyspark_job.py.tpl` (wpływało na ścieżkę do zapisania wyników).

Dataproc

W celu uzyskania zbliżonego środowiska testowego dla Dataproca (w porównaniu do GKE), w pliku `modules/dataproc-pyspark-job/main.tf` zostały dokonane następujące zmiany:

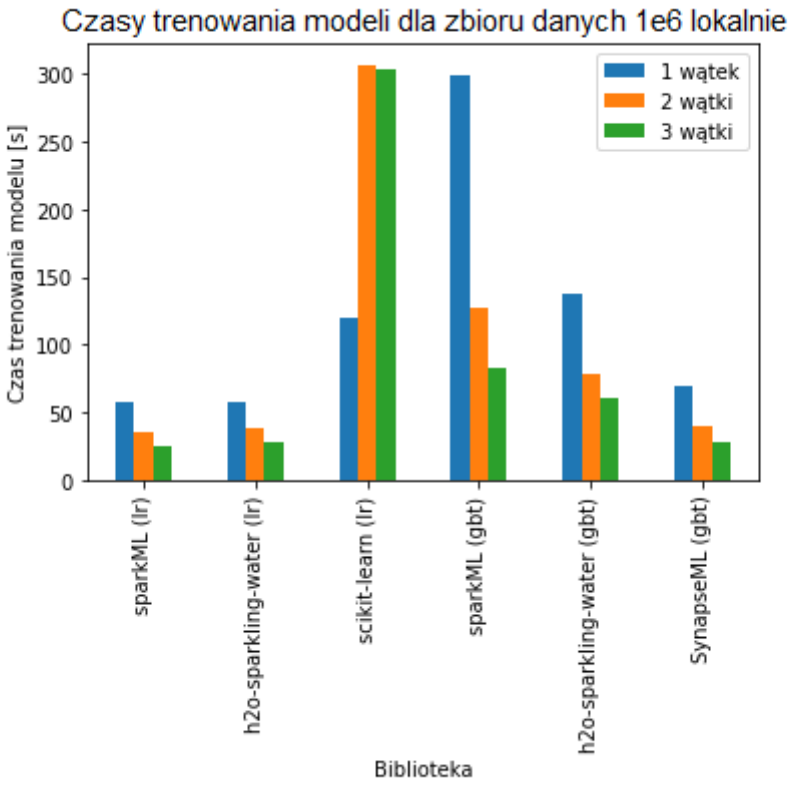
- `machine_type` dla master i worker nodów: `n1-standard-2` -> `e2-standard-2`
- `boot_disk_size_gb` dla master i worker nodów: `30` -> `50`
- `num_instances` dla worker nodów: `2` -> `6`
- do `jobs.pyspark_job.properties` zostały dodane następujące wpisy:

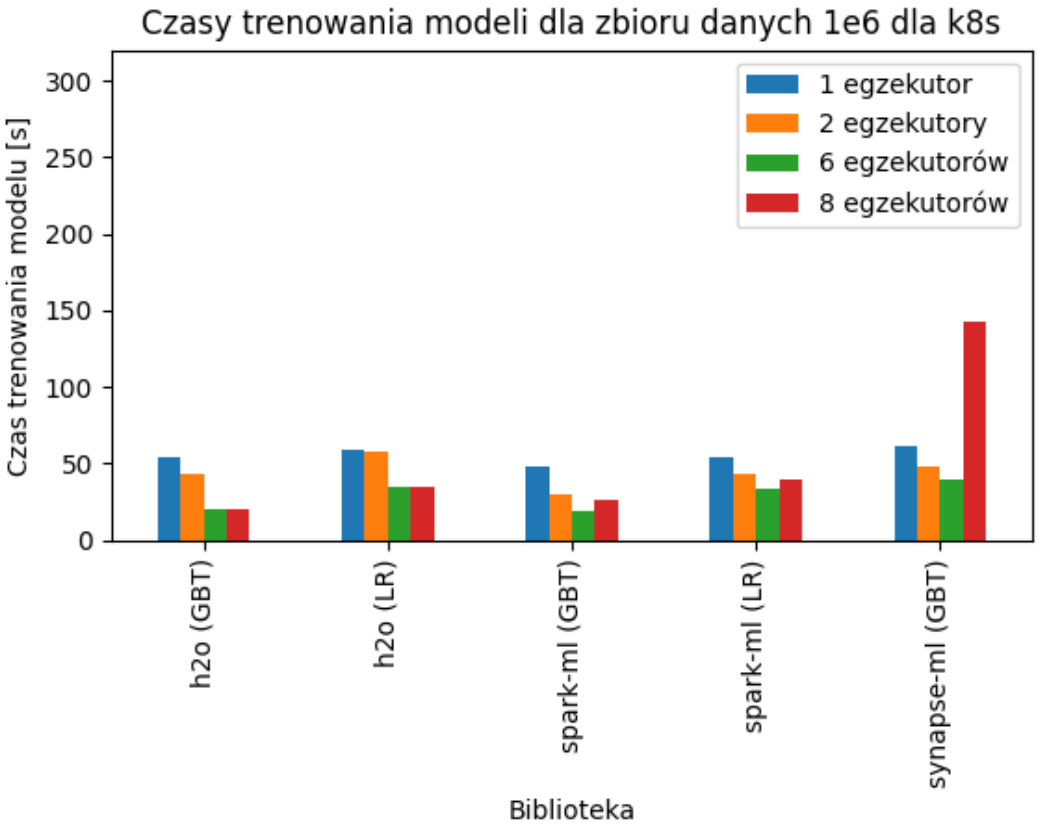
```
"spark.executor.cores" : 1
"spark.executor.instances" : 2
"spark.executor.memory" : "2000m"
```

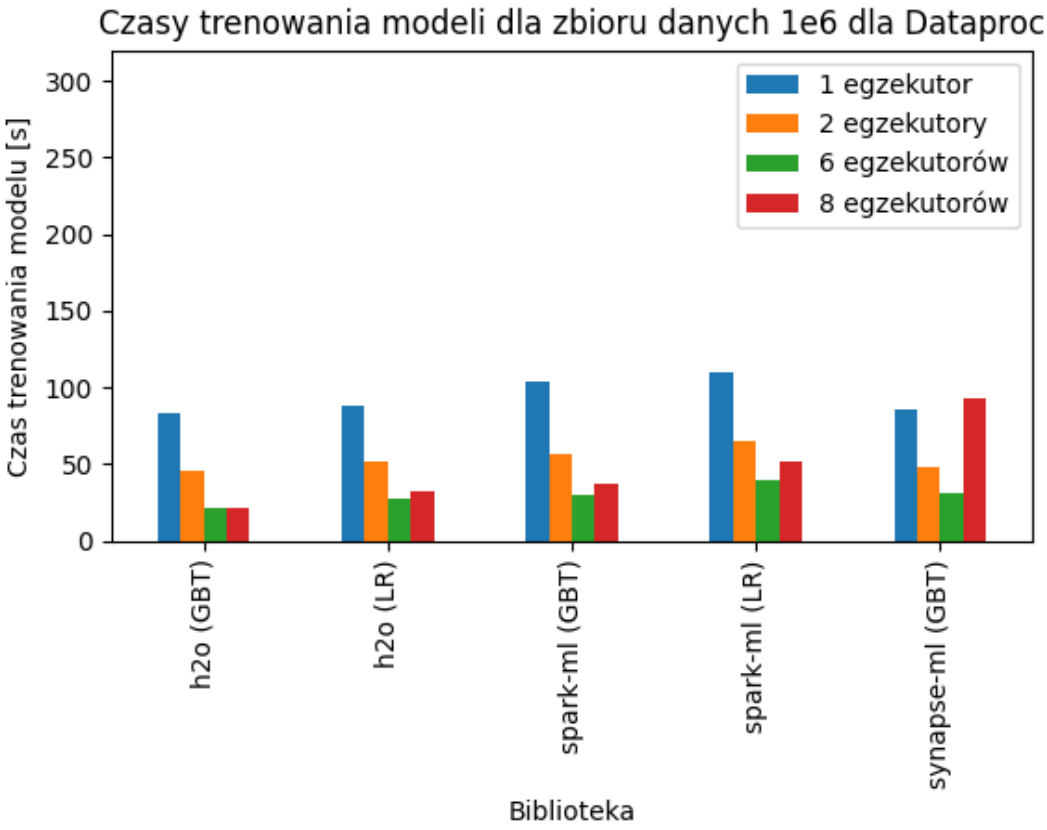
W celu modyfikowania liczby egzekutorów zmieniany był parametr `spark.executor.instances`.

Dla testów z jednym egzekutorem (zarówno dla Dataproca jak i GKE), ilość pamięci `spark.executor.memory` była zwiększana do `4000m`, ponieważ `2000m` to było za mało i job był kończony niepowodzeniem.

Wyniki


Czasy trenowania modeli dla zbioru danych 1e6 lokalnie
1 wątek
2 wątki
3 wątki
Czas trenowania modelu [s]
0
50
100
150
200
250
300
sparkML (lr)
h2o-sparkling-water (lr)
scikit-learn (lr)
sparkML (gbt)
h2o-sparkling-water (gbt)
SynapseML (gbt)
Biblioteka


Czasy trenowania modeli dla zbioru danych 1e6 dla k8s
1 egzekutor
2 egzekutory
6 egzekutorów
8 egzekutorów
Czas trenowania modelu [s]
0
50
100
150
200
250
300
h2o (GBT)
h2o (LR)
spark-ml (GBT)
spark-ml (LR)
synapse-ml (GBT)
Biblioteka



Na pierwszym wykresie przedstawiono czasy dla testów lokalnych (z poprzedniego etapu). Pozostałe wykresy mają skalę zbliżoną do pierwszego wykresu dla łatwiejszego porównywania (jednak kolejność kolumn nie została zachowana).

Na wykresach można zauważyć, że zarówno dla GKE jak i Dataproca, czasy dla modelu GBT z bibliotek spark-ml oraz h2o były zbliżone do czasów pozostałych bibliotek / modeli. Nie występował w nich *spike* jak w przypadku środowiska lokalnego.

Należy pamiętać, że bezpośrednie porównywanie czasów między środowiskiem lokalnym a chmurowym może nie mieć dużego sensu ze względu na różniącą się specyfikację i wydajność maszyn, na których te testy były uruchamiane.

Dla Dataproca czasy są ogólnie większe. W przypadku 6 egzekutorów są zbliżone do GKE, jednak dla 1 egzekutora potrafią być (w przypadku Dataproca) prawie 2 razy większe. Jedynym odstępstwem jest czas dla 8 egzekutorów dla synapse-ml (GBT), który *wyskoczył* na GKE 2 razy bardziej (względem 1 egzekutora), jednak nawet dla Dataproca był bardzo wysoki (względem innych modeli).

Na wykresach widać pewną *bardzo ciekawą* rzecz. Dla 8 egzekutorów czasy są większe niż dla 6. Wynika to z tego, że zarówno dla GKE i Dataproca został ustawiony sztywny limit 6 worker nodów. Dla liczby egzekutorów <6 istniały node'y, które nic nie robiły, ale dla liczby egzekutorów >6, istniały node'y, które musiały obsłużyć więcej niż jednego egzekutora. Prowadziło to do walki o zasoby między egzekutorami znajdującymi się na tym samym node'zie. Dla przypadków, w których czas dla 6 i 8 egzekutorów jest identyczny, można wnioskować, że zasobów było na tyle dużo (a raczej zapotrzebowanie na nie tak małe), że każdy egzekutor dostawał to, co potrzebował. Jednak GBT z synapse-ml musiał być bardzo zasobo-żerny, ponieważ czas wykonania dla 8 egzekutorów *wystrzelił w kosmos*.

Wniosek z tego jest taki, że najbezpieczniej jest, aby liczba egzekutorów była równa liczbie worker node'ów, tzn. po jednym egzekutorze na jeden worker node. Większy stosunek egzekutorów do worker node'ów może skutkować dłuższymi czasami wykonania, jeśli node nie będzie w stanie sprostać potrzebom egzekutorów (cpu, ram, itp.). Z kolei niższy stosunek spowoduje, że część node'ów będzie bezczynna, a inne node'y będą musiały same wykonać całą pracę zamiast się nią podzielić (zrównoleglić obliczenia).