Universidade Federal de Santa Maria
Centro de Tecnologia
ELC139 - Programação Paralela

Profiling de algoritmos de machine learning do Weka / wekaPython

Maio / 2022

Introdução

Weka é uma ferramenta desenvolvida na Universidade de Waikato, na Nova Zelândia. É uma ferramenta consolidada e muito popular na área de data mining, oferecendo diversos algoritmos de machine learning e facilidades para aplicá-los via interface gráfica ou via código Java.

Weka permite a instalação de pacotes que estendem suas funcionalidades. Um desses pacotes é o wekaPython, que habilita o Weka a invocar bibliotecas em Python, tais como scikit-learn. Com isso, é possível combinar facilidades do Weka com estas bibliotecas.

Uma execução típica do Weka compreende a carga de um dataset e seu processamento com um algoritmo escolhido. Muitos algoritmos podem exigir um tempo de execução

significativo para datasets com muitas instâncias e atributos. Mas como esse tempo de execução se divide entre as diversas partes do Weka e das bibliotecas em Python? Será que é possível fazer profiling disso?

Ferramentas

JFR (Java Flight Recorder)

O Java Flight Recorder é uma ferramenta de diagnóstico do sistema operacional que pode ser usada para monitorar e registrar o comportamento do sistema em tempo real. Ele pode ser usado para ajudar a solucionar problemas de desempenho e estabilidade, bem como para investigar causas de falhas do sistema.

JMC (Java Mission Control)

Java Mission Control é um conjunto de ferramentas de monitoramento e diagnóstico para aplicativos Java. Ele fornece um painel de controle para monitorar o desempenho de aplicativos Java em tempo real, bem como um conjunto de ferramentas para diagnosticar problemas de performance.

Procedimentos

O Weka em https://prdownloads.sourceforge.net/weka/weka-3-8-6-azul-zulu-linux.zip inclui um JRE na pasta weka-3-8-6/jre/zulu17.32.13-ca-fx-jre17.0.2-linux_x64. O script weka.sh invoca a JVM localizada nesta pasta, portanto não é usada nenhuma outra JVM que porventura esteja instalada no sistema.

Os seguintes arquivos são usados para uniformizar e automatizar as execuções com/sem profiling:

Arquivo	Descrição
run-exps.sh	Script criado para automatizar rodadas
exp-vars.csv	Arquivo com variáveis dos experimentos, lido pelo script run-exps.sh
wekajfr.sh	Arquivo weka.sh modificado, com flag para ativar profiling
profile.jfc	Arquivo com configurações para o Java Flight Recorder (cópia do arquivo weka-3-8-6/jre/zulu17.32.13-ca-fx-jre17.0.2-linux_x64/lib/jfr/profile.jfc distribuído com o Weka).

Resultados

Instruções para preenchimento dos resultados

Cada estudante deve preencher uma seção com:

- Nome
- URL do repositório criado pelo GitHub Classroom, contendo dados brutos das rodadas. Por exemplo: https://github.com/elc139/t3-2022a-mazuimmiguel
- Hardware utilizado: processador e memória, por exemplo: Intel Core i7-7700T 2.90GHz x 8, 15,5 GB
- SO/plataforma utilizados: no Linux, execute "cat /proc/version" para obter detalhes. Informe também se usou máquina virtual, WSL (Windows Subsystem for Linux), etc.
- Respostas às questões, acompanhada de elementos explicativos (números, gráficos, screenshots, etc.):
    - Qual o efeito do parâmetro n_jobs sobre o tempo de execução de cada algoritmo e dataset?
    - Usando o JMC, como o tempo de execução se divide entre os diversos métodos invocados em cada caso? (veja Method Profiling no JMC)

Resultados de Eduardo Lima

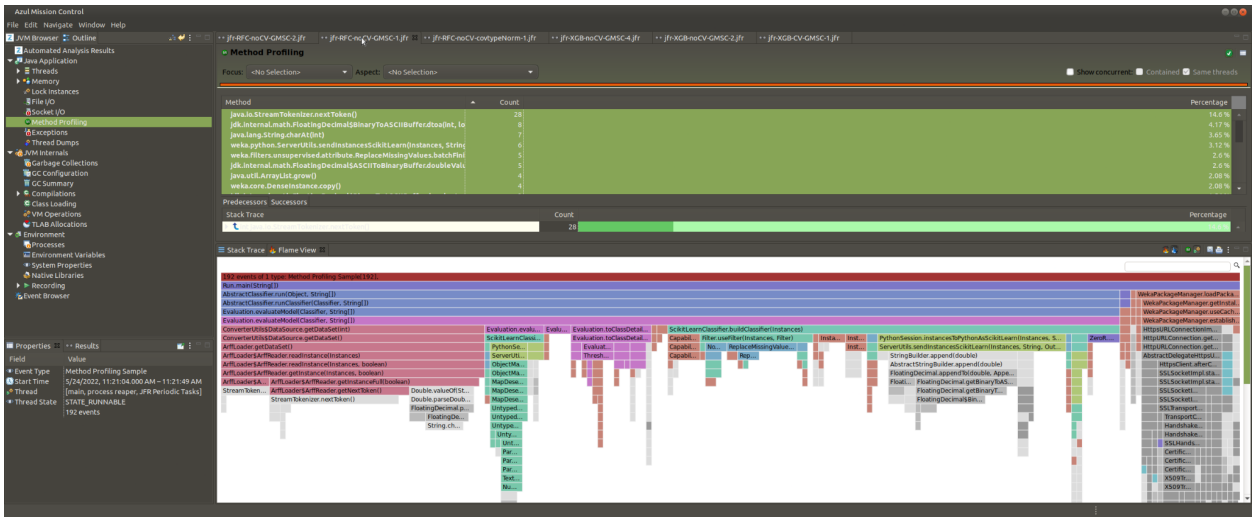
URL:
Hardware: Intel Core i7-5500U CPU 2.4GHz, 8GB RAM
SO/plataforma: Ubuntu 20.04.4 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

Nos algoritmos em que variamos o parâmetro n_jobs pudemos perceber uma diminuição no tempo de execução, em alguns podendo chegar em média 70% do tempo, em outros apenas um pouco a menos do que a execução com um parâmetro menor.

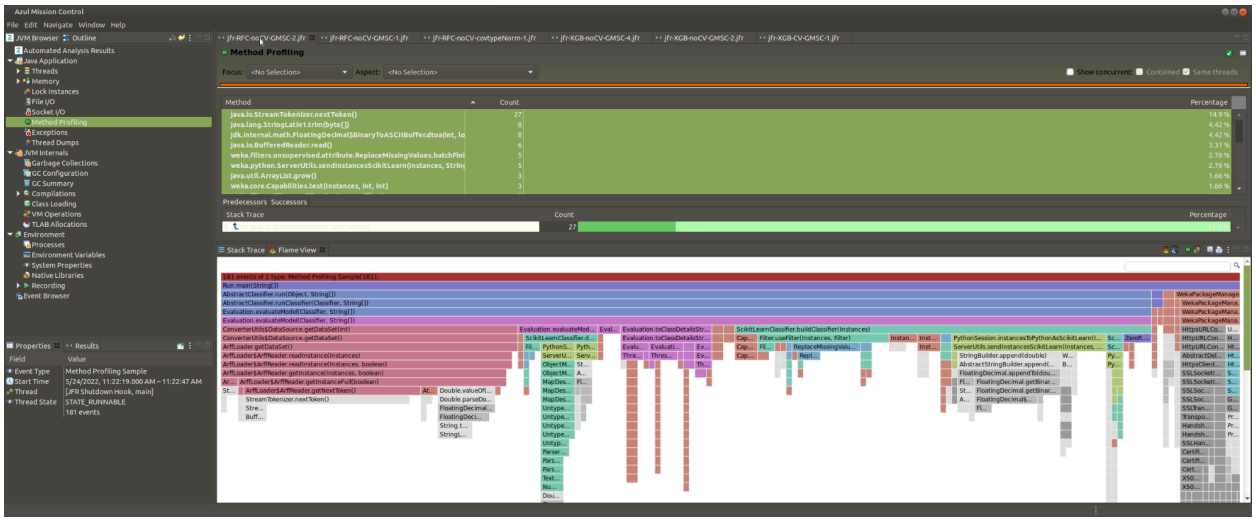
Alguns exemplos obtidos do arquivo times.csv:

XGB-noCV,GMSC,2,12.397563800
XGB-noCV,GMSC,4,11.504107400
jfr-XGB-noCV,GMSC,2,12.956396800
jfr-XGB-noCV,GMSC,4,12.470948900
RFC-noCV,covtypeNorm,1,241.463785900
RFC-noCV,covtypeNorm,2,167.568753600
jfr-RFC-noCV,covtypeNorm,1,239.841443500
jfr-RFC-noCV,covtypeNorm,2,172.975340100
RFC-noCV,GMSC,1,53.241103900
RFC-noCV,GMSC,2,37.580396600
jfr-RFC-noCV,GMSC,1,52.124864400
jfr-RFC-noCV,GMSC,2,38.049184100

Analisando o Flame View do Method Profiling no JMC utilizando o Azul Mission Control, pudemos perceber que algumas tarefas estão sendo executadas paralelamente onde o n_jobs é igual a 2 isso reduziu o tempo de execução consideravelmente. Invocando mais métodos ao mesmo tempo o programa tem um potencial de diminuição inversamente proporcional ao aumento do n_jobs, porém, como nem todos os métodos são paralelizáveis, por haver interdependências e outros empecilhos, os programas que rodamos obtiveram uma redução de cerca de 30% ao dobrarmos o parâmetro n_jobs. Abaixo, como exemplo duas chamadas do mesmo algoritmo, primeiro com n_jobs igual a um e depois com n_jobs igual a dois.



RFC-noCV,GMSC,1,53.241103900



RFC-noCV,GMSC,2,37.580396600