Spark: pierwszy kontakt

(czyli mały workshop z Apache Spark)

Jak to skonfigurować (na maszynach w pracowni)?

- pobierz paczkę zawierającą gotowe, zbudowane binarki z https://drive.google.com/open?id=1S3LUzLP19Owv_8wBEGrszmuxsrC7jkqi
- przenieś wszystkie znajdujące się w niej foldery w wygodne miejsce do którego masz dostęp
- ustaw wartość odpowiednich zmiennych środowiskowych
    - do PATH dodaj lokalizacje wszystkich plików wykonywalnych (np. takie jak poniżej)
        - C:\Program Files\Java\jdk1.8.0_162\bin
        - C:\scala-2.11.7\bin
        - C:\spark-2.3.0-bin-hadoop2.7\bin
        - C:\hadoop-2.8.3\bin
    - ustaw JAVA_HOME na C:\Program Files\Java\jdk1.8.0_162 (lub podobne)
    - ustaw SCALA_HOME na C:\scala-2.11.7 (lub podobne)
    - ustaw SPARK_HOME na C:\spark-2.3.0-bin-hadoop2.7 (lub podobne)
    - ustaw HADOOP_HOME na C:\hadoop-2.8.3 (lub podobne)
- sprawdź działanie komend spark-shell, pyspark, spark-submit (na tym etapie powinny być w pełni używalne)
- w folderze C:\ spark-2.3.0-bin-hadoop2.7\conf (lub podobnym) skopiuj log4j.properties.template i zapisz jako log4j.properties
- [opcjonalne] następnie we wnętrzu tego pliku zmień poziom logowania z INFO na WARN (oszczędzi nam to ściany tekstu przy okazji wykonywania skryptów)

Nie wiem co mam zrobić – co teraz?

Do zrobienia są trzy zadania opisane poniżej. Najprościej wykonać je startując z szablonu dostępnego na UPELu, a sporządzony skrypt przesyłać do wykonania poleceniem spark-submit. Powłokę dostępną pod poleceniem pyspark można wykorzystać do szybkiego prototypowania i testowania pomysłów.

Nie wiem jak mam coś zrobić – jak żyć?

Prowadzący powinien rozpocząć zajęcia od krótkiego wstępu. Będzie on pobieżnym streszczenie oficjalnych materiałów szkoleniowych od Apache. W szczególności zaś:

- https://spark.apache.org/docs/latest/quick-start.html (szybki start ze Sparkiem)
- https://spark.apache.org/docs/latest/rdd-programming-guide.html (wprowadzenie do RDD)
- https://spark.apache.org/docs/latest/sql-programming-guide.html (wprowadzenie do DataFrames)
- https://spark.apache.org/docs/latest/ml-guide.html (wprowadzenie do MLliba)
- oraz najważniejszego, czyli dokumentacji samego API - https://spark.apache.org/docs/latest/api/python/pyspark.html

P.S. Nie martw się, jeżeli nie zdążysz zrobić wszystkich zadań – zostały przygotowane „na zapas”, na wypadek gdyby ich rozwiązywanie szło bardzo szybko.

Zadanie 1 – całkowanie Monte Carlo

Korzystając z API RDD napisz krótki skrypt, który policzy całkę oznaczoną z x^2+1 w zakresie od 0 do 2 korzystając z metody Monte Carlo (czyli w praktyce wylosuje dużo punktów w pewnym prostokącie, sprawdzi ile z nich znalazło się pod wykresem funkcji, oraz przemnoży ten ułamek przez pole prostokąta). Porównaj uzyskany wynik z rozwiązaniem analitycznym (jeżeli nie pamiętamy jak się całkowało – to można je dostać np. na WolframAlpha…). Można inspirować się podobnym przykładem dla obliczania liczby pi (https://spark.apache.org/examples.html).

Zadanie 2 – statystyka porównawcza tekstów

Wraz z instrukcją dostarczono również spore pliki tekstowe, zawierające dwa dzieła literackie które na zawsze zmieniły oblicze Europy. Porównaj statystykę występujących w nich charakterystycznych słów. Dla każdej z ksiąg wypisz 20 najczęściej występujących w niej słów, spośród tych, których częstotliwość występowania w danym dziele jest przynajmniej 5 razy większa niż w drugim z nich (bierzemy pod uwagę tylko słowa występujące w obu księgach).

Konieczne będzie wczytanie tekstów z plików, ich podział na słowa, sprowadzenie wszystkich do lowercase, wyczyszczenie (usunięcie cyfr czy znaków przestankowych), policzenie częstotliwości wystąpień, odfiltrowanie tych spełniających zadany warunek, a na koniec wypisanie wyniku. Korzystamy z API RDD.

Zadanie 3 – wizualizacja rozkładu słów w tekstach

Na koniec prześledzimy rozkład chmury słów w obu dokumentach, korzystając z API DataFrame i gotowych rozwiązań z MLlib. Zrobimy to w następujący sposób:

- wczytamy oba teksty dzieląc je po kropkach na zdania
- scalimy je następnie do jednego DF
- użyjemy RegexTokenizera by podzielić zdania na słowa
- użyjemy StopWordsRemovera by usunąć słowa nieistotne semantycznie
- użyjemy algorytmu Word2Vec by każdemu ze słów przypisać (na podstawie kontekstu w którym występowało) dwuwymiarowy wektor je reprezentujący
- wyniki zapiszemy do pliku w postaci trzech kolumn (z której książki pochodziło słowo – wartość pierwszej współrzędnej wektora – wartość drugiej współrzędnej wektora)
- zwizualizujemy dowolnym narzędziem pozwalającym na rysowanie scatter plots
- efekt powinien być podobny jak poniżej