Statystyka w Haskellu

Programowanie funkcyjne

Aleksandra Hobot, Izabela Czarny, Maciej Kucharski

26 stycznia 2022

26 stycznia 2022

Spis treści

1 Wstęp teoretyczny

1.1 Statystyka

W naszym projekcie skupiliśmy się głownie na użyciu jęzka programowania Haskell w celu rozwiązywania zagadnień związanych ze statystyką. Statystyka jest nauką zajmującą się różnymi metodami pozyskiwania, prezentacji, analizy oraz interpretacji danych.

1.2 Średnia ruchoma

Pierwszym zagadnieniem przez nas poruszonym była średnia ruchoma. Jest ona działaniem na zbiorze danych, która tworzy serie średnich arytmetycznych dla kojenych następujących dla siebie podzbiorów. Jej główną różnicą od standardowej średniej arytmetycznej jest to, że średnia ruchoma bazuje na wcześniejszych wynikach.

1.3 Regresja liniowa

Regresja polega na estymacji warunkowej wartości oczekiwanej. W przypadku regresji liniowej modelem zależności między zmiennymi jest funkcja liniowa. Innymi słowy metoda regresji liniowej dopasowuje to zbioru x-ów i y-ów funkcje, która przewiduje jak mogą wyglądać następne wyniki.

1.4 Znajdowanie unikalnych par na liście

Znajdowanie unikalnych par na liście to tzw. szukanie kombinacji bez powtórzeń – w danym n-zbiorze skończonym szukamy podzbioru k-elementowego.

1.5 Używanie łańcucha Markowa do generowania tekstu

Łańcuch Markowa to ciąg wykorzystujący Proces Markowa z czasem dyskretnym. Ten proces to ciąg zdarzeń, w którym prawdopodobieństwo każdego zdarzenia zależy od wyniku poprzedniego (lub kilku poprzednich).

2 Metodologia

2.1 Implementacja algorytmu liczącego średnią ruchomą dla zbioru liczb

Celem kodu jest wypisanie zbioru liczb, obliczenie średniej arytmetycznej oraz ruchomej dla tego zbioru. Zbiór liczb importowany jest z pliku tekstowego "input.txt" w którym każda liczba znajduje się w innej linii pliku.

```
clean raw = map (\s -> read s :: Double) (lines raw)

avg :: [Double] -> Double
avg (x:xs) = a*x + (1-a)*(avg xs)
    where a = 0.95
avg [] = 0

mean xs = (sum xs) / (fromIntegral (length xs))

main = do
    rawInput <- readFile "input.txt"
    let input = clean rawInput
    print input
    putStrLn $ "srednia jest rowna: " ++ (show.mean) input
    putStrLn $ "srednia ruchoma jest rowna " ++ (show.avg) input
```

Kod 1: Liczenie średniej ruchomej

Działanie kodu:

- Funckja "clean" - konwersja wprowadzanego pliku na listę danych w formacie Double,
- Funkcja "avg" - liczenie średniej ruchomej dla listy liczb,
- Funkcja "mean" - liczenie średniej ruchomej dla listy liczb,
- Główna funkcja "main" - obróbka danych z pliku za pomocą ww. funkcji oraz wyświetlanie wyników.

2.2 Implementacja algorytmu liczącego regresje liniową dla zbioru punktów

Celem kodu jest wyświetlenie równania funkcji liniowej dopasowanej do zbioru x-ów i y-ów (wprowadzonych bezpośrednio w kodzie) za pomocą regresji liniowej.

```
import Statistics.LinearRegression
import qualified Data.Vector.Unboxed as U

main = do
    let xs = U.fromList [1.0, 2.0, 3.0, 4.0, 5.0] :: U.Vector Double
    let ys = U.fromList [1.0, 2.0, 1.3, 3.75, 2.25]::U.Vector Double
    let (b, m) = linearRegression xs ys
    print $ concat ["y = ", show m, " x + ", show b]
```

Kod 2: Liczenie regresji liniowej

Działanie kodu:

- Zaimportowanie bibliotek "Statistics.Linear.Regression" oraz "Data.Vector.Unboxed" - Pierwsza biblioteka pozwala nam na obliczanie regresji liniowej, a druga na odpowiednią obróbkę danych.
- Głowna funkcja "main":
  - Stworzenie serii punktów ze współrzędnych,
  - Wprowadzenie tych serii do funkcji "linearRegression",
  - Wyświetlenie wyliczonego równania funkcji.

2.3 Implementacja algorytmu znajdującego unikalne pary na liście

Celem kodu jest znalezienie i obliczenie wszystkich możliwych, niepowtarzajacych się par z listy liczb (wprowadzonej bezpośrednio w kodzie).

```
import Data.List (tails, nub, sort)

pairs xs = [(x, y) | (x:ys) <- tails (nub xs), y <- ys]
main = print $ pairs [1,2,3,3,4]
```

Kod 3: Szukanie unikalnych par

Działanie kodu:

- Zaimportowanie biblioteki "Data.List" - pozwala na korzystanie z funkcji operowania na listach (w naszym przypadku: "tail", "nub" i "sort"),
- Funckja "pairs" - tworzy wszystkie unikalne pary dla listy liczb,
- Główna funkcja "main" - wyświetla znalezione pary z listy wprowadzonych w tym miejscu liczb.

2.4 Implementacja algorytmu wykorzystującego łańcuch Markowa do generowania tekstu

Celem kodu jest wyświetlenie dwóch tekstów wygenerowanych za pomocą łańcucha Markowa, wytrenowanego w oparciu o zaimportowany przez nas plik tekstowy. Pierwszy tekst generowany jest litera po literze, w oparciu o trzy ostatnie litery. Drugi tekst generowany jest słowo po słowie, w oparciu o dwa ostatnie słowa.

```
import Data.MarkovChain
import System.Random (mkStdGen)

main = do
    rawText <- readFile "big.txt"
    let g = mkStdGen 100
    putStrLn $ "Litera po literze: \n"
    putStrLn $ take 100 $ run 3 rawText 0 g
    putStrLn $ "\nSlowo po slowie: \n"
    putStrLn $ unwords $ take 100 $ run 2 (words rawText)0 g
```

Kod 4: Generacja tekstu za pomocą łańcucha Markowa

Działanie kodu:

- Zaimportowanie bibliotek "Data.MarkovChain" oraz "System.Random" - Pierwsza biblioteka pozwala nam na korzystanie z łańcucha Markowa, a druga do generowania liczb losowych.
- Główna funkcja "main":
  - Importuje tekst z pliku "big.txt",
  - Tworzy zmienną losową "g", potrzebną do odpalenia funkcji łańcucha Markowa,
  - Wyświetla tekst wygenerowany litera po literze w oparciu o funkcje "run" z biblioteki "MarkovChain". Rozmiar wygenerowanego tekstu to 100 i generuje on na podstawie 3 poprzednich liter.
  - Wyświetla tekst wygenerowany słowo po słowie, analogicznie jak poprzednio, ale na podstawie 2 ostatnich słów, a nie 3.

3 Wyniki

3.1 Średnia ruchoma

Po odpaleniu programu uzyskaliśmy następujący wynik:

```
ghci> main
[4.0,3.0,2.0,5.0,3.0,4.0,1.0,3.0,12.0,3.0]
srednia jest rowna: 4.0
srednia ruchoma jest rowna 3.9478627675211913
```

Kod działa poprawnie. Została wyświetlona lista zaimportowanych liczb, ich średnia arytmetyczna oraz ważona.

3.2 Regresja liniowa

Po odpaleniu programu uzyskaliśmy następujący wynik:

```
"y = 0.425 x + 0.7850000000000001"
```

Kod działa poprawnie. Wyświetlone zostało równanie funkcji liniowej przypisanej do zbioru współrzędnych za pomocą regresji liniowej

3.3 Znajdowanie unikalnych par na liście

Po odpaleniu programu uzyskaliśmy następujący wynik:

```
ghci> main
[(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)]
```

Kod działa poprawnie. Wyświetlone zostały wszystkie możliwe pary bez powtórzeń dla zbioru liczb od 1 do 4.

3.4 Generowanie tekstu za pomocą łańcucha Markowa

W związku z problemami z importowaniem biblioteki "markov-chain" do środowiska w którym pracowaliśmy nie mogliśmy osobiście przetestować działania kodu. Problem z importem biblioteki najprawdopodobniej spowodowany był niezgodnością wersji jednego lub więcej elementów potrzebnych do zainstalowania tej biblioteki. Po wielu próbach aktualizacji i szukania wszelakich rozwiązań niestety nie udało nam się rozwiązać problemu.

Poprawny wynik działania programu zaprezentowany w materiałach źródłowych wyglądał następująco:

```
Generated character by character:

The evaturn bring everice Ana Paciously skuling from to was
fing, of rant of and sway.

5. Whendent

Generated word by word:

The Project gratefully accepts contributions of money, though
there was a brief word, showing that he would do so. He could
hear all that she had to reply; the room scanned Princess Mary's
heartbeat so violently at this age, so dangerous to life, by the
friends of the Russians, was trying to free his serfs--and that
till the eggs mature, when by their Creator with certain small
vessels but no me...." And the cavalry, Colonel, but I don't wish
to know which it has a fit, and there was a very large measure,
attributed to eating this root. But
```

Rysunek 5: Caption

4 Dyskusja

Zaprezentowaliśmy trzy znane i popularne zagadnienia statystki oraz jedno bardziej interesujące i zaawansowane. We wszystkich przypadkach kody użyte do rozwiązywania zagadnień były trywialne, co w większości zasługą było wielu doskonałych bibliotek wykonujących większość ciężkiej pracy za nas. Wykorzystywanie języka Haskell w celu rozwiązywania zagadnień statystyki jest więc proste, jednak przy obecnym poziomie rozwoju bibliotek do języka Python trzeba się zastanowić, czy w Pythonie nie byłoby to jednak jeszcze łatwiejsze.

5 Bibliografia

Literatura

[1] plik 3Statistics_Haskell

[2] http://en.wikipedia.org/wiki/File:Correlation_coefficient.gif

[3] http://kfe.fjfi.cvut.cz/ kucharik/edu/PF/1/lit/Landau_Paez-CP_Python-2018.pdf

[4] https://pl.wikipedia.org/wiki/Statystyka

[5] https://zerodha.com/varsity/chapter/moving-averages/

[6] http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

[7] https://brilliant.org/wiki/markov-chains/

[8] https://hackage.haskell.org/package/markov-chain-0.0.3.4/docs/Data-MarkovChain.html