Univerza v Ljubljani
Fakulteta za matematiko in fiziko
Finančna matematika

Johnson-Lindenstraussova lema

Projekt pri predmetu Finančni praktikum

Avtorici: Klara Doler, Eva Winkler
Ljubljana, 10.1.2022

Kazalo

1 Uvod

V projektni nalogi se bova ukvarjali z Johnson-Lindenstraussovo lemo, poimenovano po Williamu B. Johnsonu in Joramu Lindenstraussu.
Lema pravi, da je mogoče množico točk iz visokodimenzionalnega prostora projecirati v podprostor dimenzije $O(\frac{log(n)}{\varepsilon^2})$ tako, da se razdalja med poljubnima točkama z veliko verjetnostjo spremeni za največ ε.
Cilj najine naloge je preveriti veljavnost leme. V začetku poročila sledijo definicije in leme za lažje razumevanje problema. Nato nadaljujeva z razlago kode, ki sva jo uporabili za preverjanje veljavnosti ter na koncu še poskusi in sklep.

2 Teorija

Spodaj so opisane definicije in leme za lažje razumevanje Johnson-Lindenstraussova leme.

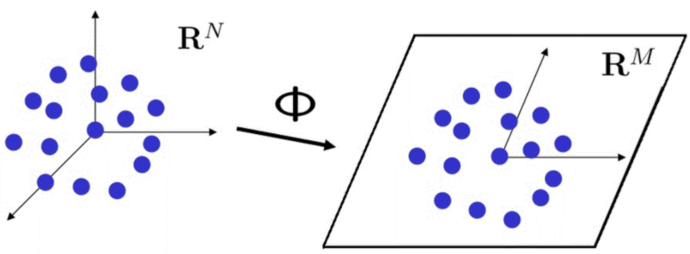
2.1 Johnson-Lindenstraussova lema

Naj bo $0 < \varepsilon < 1$ in $n \in \mathbb{N}$. Predspostavimo, da je $k > \frac{8\log(n)}{\varepsilon^2}$. Potem za vsak nabor n točk iz množice $X \subset \mathbb{R}^d$ obstaja taka preslikava $f : \mathbb{R}^d \to \mathbb{R}^k$, da za vse $u, v \in X$ velja:

$$(1-\varepsilon)\|u-v\|^2 \leqslant \|f(u)-f(v)\|^2 \leqslant (1+\varepsilon)\|u-v\|^2$$

Zgornje lahko zapišemo tudi kot:

$$(1+\varepsilon)^{-1}\|f(u)-f(v)\|^2 \leqslant \|u-v\|^2 \leqslant (1-\varepsilon)^{-1}\|f(u)-f(v)\|^2$$



Slika 1: Slika prikazuje preslikavo točk s $f : \mathbb{R}^N \to \mathbb{R}^M$, za $N > M$

2.2 Johnson-Lindenstraussova distribucija

Sorodna lema je distribucijska Johnson-Lindenstraussova lema. Ta lema pravi, da za katerakoli $0 < \varepsilon$, $\delta < \frac{1}{2}$ in pozitivno celo število d, obstaja porazdelitev na $\mathbb{R}^{k\times d}$, iz katere je sestavljena matrika A, tako da za $k = O(\varepsilon - 2\log(\frac{1}{\delta}))$ in za kateri koli enotski vektor x $\in \mathbb{R}^d$ velja spodnja trditev:

$$P(|\|Ax\|_2^2 - 1| > \varepsilon) < \delta$$

Vidimo lahko, da lahko lemo JL pridobimo iz distribucijske različice tako, da nastavimo $x = \frac{u-v}{\|u-v\|^2}$ in $\delta < \frac{1}{n^2}$ za nek par $u, v \in X$.

2.3 Dodatne definicije

Definicija 1.1 Končno razsežen vektorski prostor nad $\mathbb{R}$ opremljen s skalarnim produktom imenujemo evklidski prostor.

Definicija 1.2 V n-razsežnem evklidskem prostoru $\mathbb{R}^n$ je dolžina vektorja $\vec{x} = [x_1, x_2, \cdots x_n]$ določena z:

$\vec{x} = \sqrt{(x_1)^2 + (x_2)^2 + (x_3)^2 + \cdots + (x_n)^2}.$

Imenujemo jo evklidska norma.

3 Potek dela

Za generiranje najinega problema sva uporabili programski jezik *Python*. Izvedli sva poskuse s katerimi sva preverili, ali distribucijska JL lema drži:

- izbrali sva vzorec n točk v prostoru določene dimenzije, ki je odvisen od ε,
- naredili sva projekcijo izbranih točk na podprostor manjše dimenzije,
- primerjale sva razdaljo med točkami iz prostora nižje in višje dimenzije,
- za konec sva si še pogledali, kakšna je verjetnost, da je razdalja med točkami spremenjena za največ ε.

4 Opis programa za preverjanje leme

Začeli sva z izborom oznak, ki sva jih uporabljali tekom programiranje.

$n \ldots$ velikost vzorca točk
$d \ldots$ velikost dimenzije prostora
$k \ldots$ velikost dimenzije podprostora

Ker imamo opravka z vzorcem n točk v večdimenzionalnem prostoru d, so ti pravzaprav vektorji velikosti $1 \times d$. Za lažje generiranje leme sva vektorje združile v matriko velikosti $d \times n$, torej d vrstic in n stolpecv.

Dodatno sva defirnirali še oznaki za matriki:
$M \ldots$ naključno generirana matrike velikosti $d \times n$
$G \ldots$ transformirana matrika velikosti $k \times d$

4.1 Program za transformacijo matrike

Za pravilno delovanje programa, sva dodatni naložili še spodnje knjižnice:

```
import random
import math
import numpy
import numpy as np
```

Najprej sva definirali funkcijo `spodnjameja_k`, ki nam za izbrano število n točk in izbran ε izračuna spodnjo mejo dimenzije podprostora.

```
def spodnjameja_k(n, epsilon):
   return math.ceil(8*math.log(n) / (epsilon**2))
```

Generirale sva naključno matriko M s funkcijo `randomMatrika(d,n)` velikosti $d \times n$. Vrednosti v matriki so generirane naključno z normalno porazdelitvijo. Matrika ima d vrstic in n stolpcev. Stolpci predstavljajo izbrano število n točk v prostoru dimenzije d.

```
def randomMatrika(d,n):
   return numpy.random.normal(0, 1, size=(d,n))
```

Nadaljevali sva z generiranjem Johnson-Lindenstraussove transformacije preko funkcije `jltransformacija`. Funkcije sprejme naključno matriko M velikosti $d \times n$, nato znotraj kode definira novo matriko A velikosti $k \times d$, s pomočjo katere poteka transformacija. Za tranformacijo sva uporabili spodnjo preslikavo:
$f(x) = \frac{1}{\sqrt{k}} * Ax$
Preslikava poteka po stolpcih x matrike M. Kot rezultat dobimo transformirano matriko G iz podprostora k, ki ima k vrstic in n stolpecv.

```
def randomPodprostor(podprostorDimenzija, prostorDimenzija):
   return numpy.random.normal(0, 1, size=(podprostorDimenzija, prostorDimenzija))

def jltransformacija(podatki, podprostorDimenzija):
   prostorDimenzija = len(podatki)  # dimenzija
   A = randomPodprostor(podprostorDimenzija, prostorDimenzija)
   return (1 / math.sqrt(podprostorDimenzija)) * A.dot(podatki)
```

4.2 Ohranjanje razdalje

Za pravilo delovanje Johnson-Lindenstraussova leme je ključno ohranjanje razdalj med točkami v podprostoru, torej ohranjanje evklidske norme med vektorji.

Za lažje preverjanje norm sva pred transformacijo poskrbeli, da imajo vsi stolpci v matriki M normo 1. To sva naredili s spodnjo kodo, ki nam vrne novo matriko, katero sva nato uporabili za tranformacijo. Označimo novo matriko z oznako D.

```
M / numpy.linalg.norm(M, axis=0)
```

Računanje norme se je tako poenostavilo. Primerjati sva morale le absolutne vrednosti razlik med stolpci matrike D in stoplpci matrike G. Stoplci matrike D imajo zaradi zgornjega normiranja normo 1. To nam olajša računanje.
V nadaljevanju naju je zanimala verjetnost, da se je razdalja med točkami spremenila za manj kot ε.

Definicija Johnson-Lindenstraussova distribucije pravi, da za katerakoli $0 < \varepsilon$, $\delta < \frac{1}{2}$ in pozitivno celo število d obstaja porazdelitev na $\mathbb{R}^{k\times d}$, iz katere je sestavljena matrika A, tako da za $k = O(\varepsilon - 2\log(\frac{1}{\delta}))$ in za kateri koli enotski vektor $\mathrm{x} \in \mathbb{R}^d$ velja spodnja trditev:

$$P(|\|Ax\|_2^2 - 1| > \varepsilon) < \delta$$

Poleg tega lahko vzamemo, da je $\delta < \frac{1}{n^2}$.
V zgornji formuli $\|Ax\|$ označuje normo matrike G, 1 pa je norma matrike D.

Ponovno sva normirali stolpce, a tokrat v trasformirani matriki G. Kot rezultat sva dobili vektor dimenzije $n \times 1$. Ker pri izračunu verjetnosti potrebujemo kvadrat norme, sva dobljeni vektor kvadrirali. Označimo vektor s črko X.

```
x = numpy.linalg.norm(G, axis=0)
X = x*x
```

Nato sva izračunali absolutne razlike med 1 in vrednostmi v vektorju X. Dobljene razlike so točno razlike razdalj med vektorji. Ker lema drži z veliko verjetnostjo je veliko več razlik, ki so manjše od ε, kot tistih večjih. V ta namen sva s funkcijo `preštejemo` dobile število vektorjev, ki imajo razliko večjo od ε. To število sva nato delili z številom vseh vektorjev oz. stolpcev in dobili delež odstopanj, ko se razdalja ni ohranila za manj kot ε.

```
preštejemo = abs(1-X)> epsilon
število_primerov = preštejemo.sum()
delež = število_primerov/(stevilo stolpcev)
```

4.3 Program

Za bolj resno obdelavo pa ni dovolj, da preverimo lemo le na enem primeru.

Napisali sva funkcijo `program(epsilon, n, C, i, j, t=10)`, ki nam za določene parametere sama večkrat zažene program.

Ker imava pri verjetnosti opravka z $\delta < \frac{1}{n^2}$ lahko pričakujeva, da bo do ena pojavitev odstopanja na n^2 točk. Zato sva vzeli matriko M z $C * n^2$ stolpci (npr. pri n = 50 in C = 40 bo imela 100000 stolpcev).

Dodatno sva na matriki M večkrat klicali `jltransformacija`. V ta namen sva v funkciji `program(epsilon, n, C, i, j, t)` definirali parameter t, s katerim lahko določimo število ponovitev transformcije.

Spodaj je definirana funkcija `program(epsilon, n, C, i, j, t=10)` .

```
def program(epsilon, n, C, i, j, t=10):
    število_vseh_primerov = []
    #deleži_odstotek = []
    deleži = []
    delta_drži = []
    for m in range(i,j+1):
        k = spodnjameja_k(n, epsilon)
        d = k + 100*m
        M = randomMatrika(d,C*n*n)
        D = M / numpy.linalg.norm(M, axis=0)
        število_primerov = 0
        for _ in range(t):
            G = jltransformacija(D, k)
            x = numpy.linalg.norm(G, axis=0)
            X = x*x
            preštejemo = abs(1-X)>epsilon
            število_primerov += preštejemo.sum()
        delež = število_primerov/(C*n*n*t)
        preštejemo_delta = delež < (1 /(n*n))
        #delež_odstotek = delež * 100
        število_vseh_primerov.append(število_primerov)
        deleži.append(delež)
        #deleži_odstotek.append(delež_odstotek)
        delta_drži.append(preštejemo_delta)
    return število_vseh_primerov, deleži, delta_drži
```

V sami funkciji sva definirali tudi parametra i in j preko katerih lahko izračunamo tranformacijo za matrike iz različnih dimenzij. Izbrani ε nam določi spodnjo mejo za dimenzijo k podprostora. Za lažje definiranje programa sva se odločili, da za dimenzijo k vzamemeva kar to spodnjo mejo. Nato pa sva za dimenzijo d vzeli $d = k+100*m$, kjer vrednost m teče med parametrom i in $j+1$.

Funkcija nam kot odgovor vrne 3 sezname.
Prvi seznam `število_vseh_primerov` nam vrne število primerov odstopanj za posamezno matriko. Naslednji seznam `deleži` vrne deleže odstopanj v posamezni matriki. Za konec pa še seznam `delta_drži`, ki vrne vrednost $True$, če je delež manjši od delte. Znotraj programa sva namreč dodatno privzeli, da je $\delta = \frac{1}{n^2}$. Posledično, če je delež manjši od $\frac{1}{n^2}$ je tudi manjši od neke δ, ki je manjša od $\frac{1}{n^2}$.

5 Poskus in ugotovitve

5.1 Poskus 1

Pogledali sva si kakšne rezultate dobiva, če spreminjava ε.

Vzeli sva 50 točk ($n = 50$), $C = 40$, $d = k + 100 * l$, $l = 1, 2, 3$ in $t = 10$.

Za ε sva vzeli 0.2, 0.3, 0.4, 0.5 in 0.8.

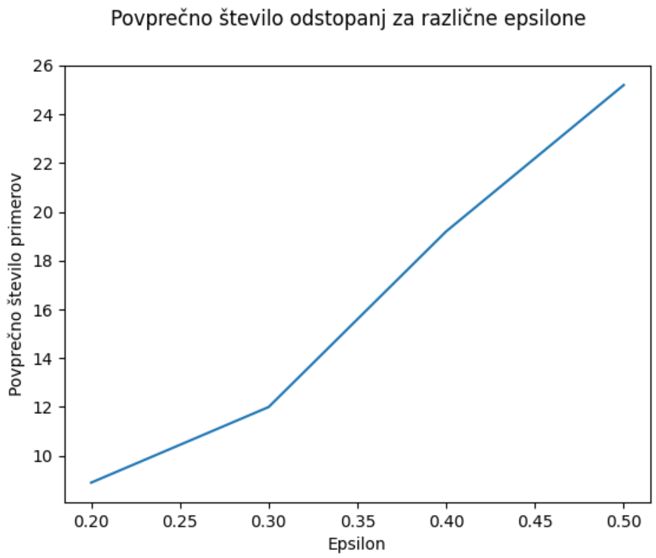
Dobili sva sledeče rezultate:

```
program(0.2,50,40,1,3,10)
([89, 101, 111], [8.9e-05, 0.000101, 0.000111], [True, True, True])

program(0.3,50,40,1,3,10)
([120, 145, 140], [0.00012, 0.000145, 0.00014], [True, True, True])

program(0.4,50,40,1,3,10)
([192, 196, 182], [0.000192, 0.000196, 0.000182], [True, True, True])

program(0.5,50,40,1,3,10)
([252, 305, 235], [0.000252, 0.000305, 0.000235], [True, True, True])

program(0.8,50,40,1,3,10)
([525, 429, 457], [0.000525, 0.000429, 0.000457], [False, False, False])
```

Iz rezultatov sva ugotovili:

- Pri istem ε in različih dimenzijah d so prišle podobne vrednosti števila odstopanj.
- Z večjim ε so tudi večja števila odstopanj.
- Pri $\varepsilon = 0.8$ so prišli deleži večji od δ ($\delta = \frac{1}{50^2}$). S tem lahko potrdiva, da Johnson-Lindenstraussova distribucija deluje za $0 < \varepsilon < 0.5$

Rezultate sva predstavili tudi grafično.
Na x-osi so vrednosti ε, na y-osi pa povprečna vrednost števila odstopanj za $d = k + 100$.



Iz grafa lahko opazimo, da je pri večjem ε število primerov napak večje. Za $\varepsilon = 0.2$ je bilo povprečno število primerov odstopanj 8.9, za $\varepsilon = 0.3$ je bilo 12.0, za $\varepsilon = 0.5$ pa je bilo 25.2.

5.2 Poskus 2

V drugem poskusu sva si pogledali kakšne rezultate dobiva, če spreminjava število izbranih točk v prostoru d pri $\varepsilon = 0.2$.

Za n sva vzeli 5, 10 in 50. Ostali parametri so ostali enaki.

```
program(0.2,5,40,1,3,10)
([98, 117, 114], [0.0098, 0.0117, 0.0114], [True, True, True])

program(0.2,10,40,1,3,10)
([107, 92, 109], [0.002675, 0.0023, 0.002725], [True, True, True])

program(0.2,50,40,1,3,10)
([89, 101, 111], [8.9e-05, 0.000101, 0.000111], [True, True, True])
```

Iz rezultatov lahko vidimo, da se delež primerov odstopanj manjša z večanjem števila točk in število primerov odstopanj ostaja enako.

6 Sklep

S projektno nalogo lahko potrdiva veljavnost Johnson-Lindenstraussove leme.

7 Viri

- https://cs.stanford.edu/people/mmahoney/cs369m/Lectures/lecture1.pdf, dostopano dne 6. 12. 2021
- https://en.wikipedia.org/wiki/Johnson%E2%80%93Lindenstrauss_lemma, dostopano dne 6. 12. 2021
- https://jeremykun.com/2016/02/08/big-dimensions-and-what-you-can-do-about-it/, dostopano dne 10. 12. 2021