KLASTEROVANJE

Šta je klasterovanje?

● Nalaženje grupa objekata takvih da su objekti iz grupe međusobno slični (ili povezani) i da su različiti (nepovezani) od objekata u drugim grupama

Primene klasterovanja:

● Razumevanje

● Smanjenje veličine velikih skupova podataka

Tipovi klasterovanja – terminologija 1/2

● Partitivno

    ○ Podela objekata u nepreklapajuće podskupove (klastere) takva da je svaki objekat u tačno jednom podskupu

● Hijerarhijsko

    ○ Skup ugnježdenih klastera organizovanih kao hijerarhijsko stablo

Tipovi klasterovanja – terminologija 2/2

● Tvrdo (Hard)

    ○ Binarana pripadnost klasteru

● Meko (Soft)

    ○ Pripadnost klasteru je kontinualna vrednost (najviše ima smisla da je u intervalu [0,1]

K-sredina

● Partitivni pristup klasterovanju

● Svakom klasteru se dodeljuje centroid (centar)

● Svaka tačka se svrstava u klaster sa najbližim centroidom

● Broj klastera K mora biti zadat

● Osnovni algoritam je vrlo jednostavan:

1. Selektovati K tačaka za početne centroide

2. repeat

3. Formirati K klastera svrstavanjem tačaka u najbliži centroid

4. Sračunati novi centroid za svaku klasu (na bazi svrstanih tačaka)

5. until centroid se ne menja

K-sredina - Detalji

● Inicijalni centroidi se često slučajno biraju.

● Dobijaju se različiti klasteri za različite slučajne sekvence.

● Centroid je (obično) srednja vrednost tačaka iz klastera.

● ‘Blizina’ se meri Euklidskim rastojanjem, kosinusnom sličnošću, korelacijom, itd.

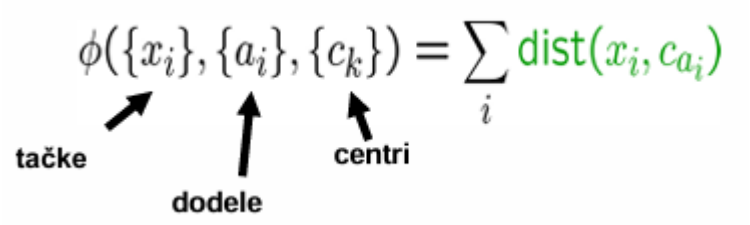
● K-sredine konvergiraju za uobičajene (pomenute) mere sličnosti.

● Najbrža je konvergencija u prvih nekoliko iteracija.

● Kriterijum zaustavljanja u praksi je najčešće ‘dok relativno malo podataka menja klaster

K-Means kao optimizacioni problem

Pogledajmo ukupan zbir rastojanja tačaka do centara:



Svaka iteracija smanjuje fukciju ϕ

Dve faze u svakoj iteraciji:

– Dodela klasterima: fiksiramo centre c, menjamo dodele a

– Promena centara: fiksiramo a, menjamo centre c

Faza I: Dodela Klasterima

Dodaj svaku tačku centru koji joj je najbliži: $a_i = \underset{k}{\text{argmin}}\, \text{dist}(x_i, c_k)$

$$\phi(\{x_i\},\{a_i\},\{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

Ova faza može samo da smanji fukciju ϕ!

Faza II: Promena Centara

Pomeramo svaki centar ka proseku tačaka koje su mu dodeljene: $$c_k = \frac{1}{|\{i : a_i = k\}|} \sum_{i:a_i=k} x_i$$

Takođe samo smanjuje fukciju ϕ.

Uzećemo bez dokaza: tačka koja ima najmanju kvadratnu euklidsku udaljenost ka tačkama {x} u nekom skupu je baš centar tih tačaka

Inicijalizacija

K-means ne daje uvek isti rezultat za više pokretanja

– Zahteva inicijalne centre

– Vrlo je značajno kako su odabrani!

– Postoji puno metoda za rešavanje ovog problema. Jedan od njih ćemo raditi danas

K-Means može da se zaglavi

Problemi pri izboru inicijalnih tačaka

● Ako postoji K‘stvarnih’ klastera, verovatnoća da se izabere jedan centroid za svaki klaster je mala.

Rešenje problema inicijalnih centroida– 1/2

● Višestruka izvršavanja

● Korišćenje hijerarhijskog klasteringa za određivanje inicijalnih centroida

● Generisanje više od k inicijalnih centroida i zatim izbor među tim centroidima

● Biraju se oni koji su najbolje razdvojeni

● Post-procesing (spajanje ili razbijanje dobijenih klastera)

● Bisekcija K-sredina (biće prikazan na nekom drugih mojih kurseva)

K-means++

-Predlog rešenja problema inicijalizacije centara

- Ideja algoritma: raširiti centre što više

- Algoritam je stohatički.

- Verovatnoće za odabir centara podešene

K-means kao metod optimizuje sumu kvadrata grešaka (SSE):

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist^2 (m_i, x)$$

● x je podatak (tačka) iz klastera Ci, a mi odgovara centru klastera.

Kako odabrati broj klastera?–„Lakat“ metod

● Iteriramo po broju klastera i prikazujemo SSE.

●Tražimo nagli prelaz (“lakat”) u grafiku SSE po broju klastera

Kako odabrati broj klastera?– Gap statistika

● Zasniva se na razlici (Gap) disperzije klastera dobijenih pomoću K-sredina za dati skup podataka i disperzije klastera slučajno generisanih skupova podataka

● Razlika se meri iterativno od nekog datog broja klastera

● Broj klastera koji proizvede najveći razmak je predlog za broj klastera za K-sredina

Ograničenja K-sredina

● K-sredina ima probleme kada se razlikuju klasteri

● K-sredina ima problem u slučaju prisustva stranih podataka.

DBSCAN

● Algoritam baziran na gustini.

● Gustina = broj tačaka unutar zadatog prečnika (Eps)

● Tačka je tačka jezgra (core point) ako ima više od specificiranog broja tačaka (MinPts) unutar Eps

● Ivična tačka (border point) ima manje od MinPts tačaka na rastojanju Eps, ali je susedna sa tačkom jezgra (nalazi se u Eps “krugu” neke tačke jezgra)

● Tačka šuma (noise point) je svaka tačka koja nije ni tačka jezgra ni ivična tačka.

DBSCAN funkcioniše tako što markira guste komšiluke tačaka kao zasebne klastere

MASINSKO UCENJE

Mašinsko učenje je proučavanje algoritama koji:

• poboljšavaju svoje performanse P

• za neki zadatak T

• kroz iskustvo E.

Dobro definisan zadatak mašinskog učenja dat je sa <P,T,E>

Mašinsko Učenje je Grana veštačke inteligencije, koja se bavi konstrukcijom i proučavanjem sistema koji uče iz podataka.

ML se koristi kad:

• Ljudska ekspertiza ne postoji

• Ljudi ne mogu da objasne ekspertizu

• Nivo kvaliteta, detaljnosti za koji ljudi nemaju vremena

• Količine podataka koje ljudi ne mogu da savladaju

Još neki zadaci za koje je prikladna i uspešna primena ML: Prepoznavanje šablona,Generisanje šablona, Detekcija anomalija, Predikcija

Tipovi Učenja

• Nadgledano (induktivno) učenje – Dato: skup podataka + oznake (npr. oznake klasa)

• Nenadgledano učenje – Dato: skup podataka (bez oznaka)

• Polu-nadgledano učenje – Dato: skup podataka + vrlo malo označenih podatka

• Učenje Uslovljavanjem – Nagrade ili kazne nakon niza akcija

Nadgledano učenje: klasifikacija

• Dato je (x1, y1), (x2, y2), …, (xn, yn)

• Cilj je naučiti funkciju f(x) koja predviđa y za dato x

– y je diskretna vrednost

x može biti više-dimezioni – Svaka dimenzija je jedan atribut

Nadgledano Učenje: Regresija

• Dato je (x1, y1), (x2, y2), …, (xn, yn)

• Cilj je naučiti funkciju f(x) koja predviđa y za dato x

– y je kontinunalna vrednost

Nadgledano učenje: rangiranje

Rangiranje : oznaka je rang (npr. 1 je bolje od 2, a 2 od 3 itd.)

Nenadgledno Učenje

• Dato je x1, x2, …, xn (bez oznaka)

• Izlaz je neka struktura (šablon) koji važi za x-ove

– Npr. automatsko grupisanje (klasterovanje)

Učenje uslovljavanjem

• Učenje uslovljavanjem je vrsta mašinskog učenja

Svaki algoritam mašinskog učenja ima tri komponente:

– Reprezentacija

– Optimizacija

– Evaluacija

Različiti načini reprezentacije ciljne funkcije: Numeričke funkcije, Simboličke funkcije, Funkcije zasnovane na instancama (podacima), Probabilistički modeli

Različiti algoritmi za optimizaciju i pretragu: Gradijentni spust, Dinamičko programiranje, „Podeli pa Vladaj“, Evolutivni Algoritmi

Evaluacija:

• Tačnost (Accuracy)

• Preciznost i Odziv (Precision and Recall)

• Kvadrat greške (Squared error)

• Verovatnost (Verodostojnost) - Likelihood

• Posteriorna verovatnoda (Posterior probability)

• Trošak / Korisnot (Cost / Utility)

• Entropija

• K-L divergencija

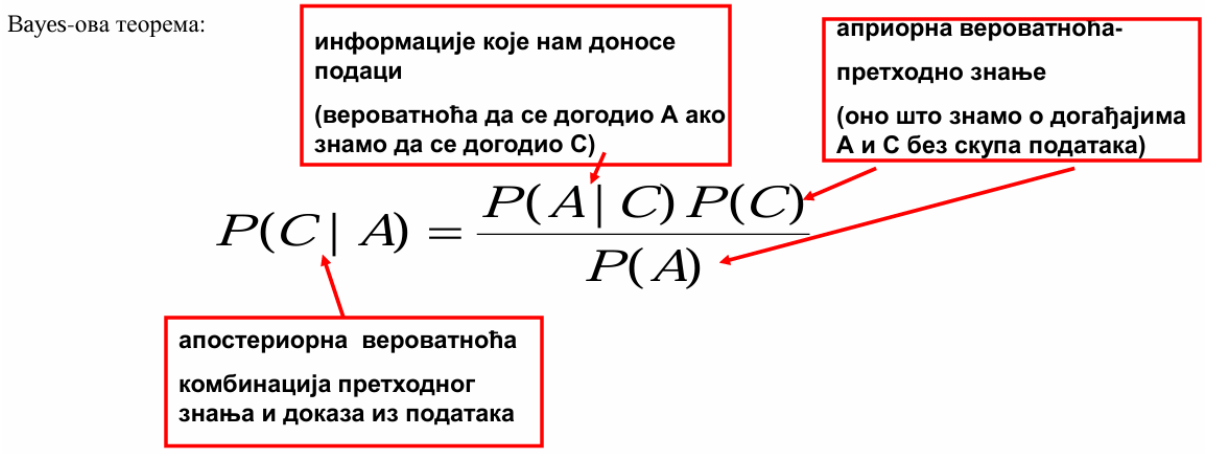
NAIVNI BAJESOV KLASIFIKATOR

Naivni Bajes Intuicija - Jednostavan (“naivan”) klasfikacioni metod zasnovan na Bajesovoj teoremi, Oslanja se na jako jednostavnu reprezentaciju dokumenata - Vreca reči (Bag of words)

Bajesova teorema primenjena na dokumente i klase:

$$P(c \mid d) = \frac{P(d \mid c) P(c)}{P(d)}$$

Za dokument d i klasu c:



$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(x_1, x_2, \ldots, x_n \mid c) P(c)$$

MAP - “maksimalna aposterirorna verovatnoća” = najverovatnija klasa

Multinomialni Naivni Bajes

Pretpostavka Bag-of-Words reprezentacije:
Pretpostavljamo da pozicija reči u tekstu nije važna

Uslovna nezavisnost: Pretpostavljamo da su verovatnoće osobina tj. $P(x_i \mid c_j)$ nezavisne u odnosu na klasu *c.* Na taj način verovatnoću $P(x_1,..,x_n \mid C)$ računamo kao:

$$P(x_1, \ldots, x_n \mid c) = P(x_1 \mid c) \bullet P(x_2 \mid c) \bullet P(x_3 \mid c) \bullet \ldots \bullet P(x_n \mid c)$$

$$c_{NB} = \underset{c \in C}{\operatorname{argmax}}\, P(c_j) \prod_{x \in X} P(x \mid c)$$

Obučavanje NB klasifikatora

Koristimo metod maksimalne verovatnosti (maximum likelihood estimates) • konkretno koristimo frekvencije dobijene iz korpusa

Izračuvanje verovatnoca

$$\hat{P}(w_i|\, c_j) = \frac{count(w_i, c_j)}{\sum_{w \in V} count(w,\, c_j)}$$

broj pojavljivanja reči wi u dokumentima koji imaju klasu cj podeljen sa brojem pojavljivanja svih reči iz rečnika V u svim dokumentima koji imaju klasu cj

• Kreiramo mega-dokument za klasu j tako što sve dokumente ove klase spojimo u jedan dokument

• Računamo frekvenciju reču wi u tom mega-dokumentu

Bez obzira na vrednosti drugih verovatnoda ako je jedna od verovatnoda 0 verovatnoda klase c je 0!

Laplasovo (dodaj-1) poravnavanje za NB

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c) + 1}{\sum_{w \in V} \left(count(w, c) + 1 \right)}$$

$$= \frac{count(w_i, c) + 1}{\left(\sum_{w \in V} count(w, c) \right) + |V|}$$

Naivni Bajes nije baš tako naivan

• Vrlo brz, ne treba mu puno memorije

• Robusan na beznačajne osobine Osobine koje su jako slične za sve primere ne menjaju verovatnode

• Ako pretpostavka o nezavisnosti stvarno važi NB je optimalan model

• Čak iako ne važi praksa je pokazala da je NB dobar klasifikator

• U trenutnom stajnu ML oblasti NB je pozdan model i dobar osnonovni model

Matrica Konfuzije, Odlučimo koja je klasa za nas pozitivna, koja negativna i formiramo matricu.

Preciznost (Precision) : % predikcija poz koji su stvarno poz p$recision$ = tp / tp +fp

Odziv (Recall): % stvarno poz od predikcija poz. r$ecall$ = tp / tp +fn

F-mera • Mera koja procenjuje koji je balans između P i R: F= 2PR/ P+R

Makro prosek: Izračunamo meru za svaku klasu i onda uzmemo prosek (npr. prosek F-mera)

Mikro prosek: Formiramo matricu konfuzije i onda iz nje računamo mere na nivou cele matrice. Npr. za Odziv uradimo zbir svih tp u celoj matrici i podelimo sa svim (tp+fn)

Validacioni skup – koristimo ga tokom razvoja modela za podešavanje parametara i selekciju osobina.

• Test skup – koristimo ga za krajnju procenu kvaliteta modela • može da bude varljiv (previše lak ili previše težak test skup)

• Unakrsna validacija (objašjeno na slededem slajdu) – može da se koristi i tokom razvoja ili za krajnju procenu. Nikako za oba u isto vreme. Ako je koristimo u toku razvoja, moramo da imamo odvojen test skup.

Jako mali obučavajući skup?

• Korisiti Naivni Bajesov model, NB ima veliki bijas i malu varijansu

• Povecati obučavajuci skup, „Ubediti“ ljude da vam označe podatke

• Probati metode polu-nadgledanog učenja

Relativno veliki obučavajući skup?

• Savršena postavka za kompleksnije modele: SVM, Regularized Logistic Regression

• Možete koristiti i stabla odlučivanja: Interpretabilan model

Jako veliki obučavajući skup?

• Možemo dobiti jako veliku tačnost!

• Cena je naravno brzina: • SVM (sporo obučavanje) ili kNN (spora primena na test skupu) •

Naivni Bajes je ovde odlična opcija jer je brz!

• Naravno Deep Learning je uvek odlična opcija kada imate puno podataka.

UVOD U KLASIFIKACIJU SLIKA

Izazovi: Pomeranje tačke iz koje gledamo, Osvetljenost, Deformacija, Okluzija, Pretrpana pozadina, Varijabilnost unutar iste klase

Za razliku od npr. sortiranja liste, ne postoji jednostavan način da hard-kodujemo algoritam za prepoznavanje klasa.

Upotreba Nadgledanog Učenja

1. Prikupiti skup slika i oznaka klasa

2. Upotrebiti nadgledano učenje za obučavanje klasifikatora

3. Evaluirati klasifikator na novim slikama

Klasifikator K-Najbližih Komšija, KNN

- Čuvamo (pamtimo) obučavajući skup

- Za test sliku: Naći nasličniju (najbližu) sliku iz obučavajućeg skupa Vratiti oznaku klase te slike kao predikciju klase za test sliku

Mera sličnost (udaljenosti) za poredjenje slika

L1 (Menhetn) mera:

$$d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$$

L2 (Euklidsko) mera:

$$d_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

Sa N slika u obučavajućem skupu, koliko traje obučavanje i primena na test sliku? Odgovor: Obučavanje O(1), Primena na test sliku O(N) To nije ono što želimo: hoćemo klasifikatore koji se brzo primenjuju, dok je sporo obučavanje prihvatljivo

hiper-parametri: odluke koje moramo da donesemo o modelu pre procesa učenja.

Kako odredjujemo hiper-parametre?

Način #1: Biramo vrednosti koje daju najbolje rezultate na obučavajućem skupu, Loš način: Za K = 1 uvek imamo savršene rezultate na ob. Skupu

Način #2: Delimo ob. skup na obučavajući i test, biramo vrednosti koje daju najbolje rezultate na test skupu, Loš način: Nećamo imati nikakvu realnu predstavu o tome kako će se model ponašati na stvarno nepozatim podacima

Način #3: Delimo ob. skup na obučavajući, validacioni, i test; biramo vrednosti koje daju najbolje rezultate na validacionom, konačnu evalaciju radimo na test skupu, Sad je sve korektno!

Način #4: Unakrsna Validacija: Delimo podatke u delove (fold), svaki deo koristimo kao validacioni skup, pa uzimamo prosek rezultata, Korisno kad imamo manji obučavajući skup, ali retko upotrebljeno u deep learning oblasti

KNN se u današnje vreme nikada ne koristi za klasifikaciju slika - Vrlo spor kad se primenjuje na test - Mere sličnosti koje smo pokazali nisu dovoljno dobre kada se primene na piksele

LINEARNI KLASIFIKATORI

(linearna) skor funkcija f(x,W) = Wx + b

funkcija greške za SVM je oblika:

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Greška za sve tri slike je prosek grešaka:

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i$$

Naša funkcija greške ima bag:

$$f(x, W) = Wx$$

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1)$$

Regularizacija težina
lambda = jačina regularizacije (hiper-parametar)

$$L = \frac{1}{N} \sum_{i=1}^{N} \sum_{j \neq y_i} \max(0, f(x_i; W)_j - f(x_i; W)_{y_i} + 1) + \lambda R(W)$$

Tipično se koristi:

L2 regularizacija

L1 regularizacija

Elastic net (L1 + L2)

Dropout

$$R(W) = \sum_k \sum_l W_{k,l}^2$$

$$R(W) = \sum_k \sum_l |W_{k,l}|$$

$$R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$$

Softmax Klasifikator (Multinominalna Logistička Regresija)

skorovi = nenormalizovane log verovatnoće klasa (nenormalizovane znači da se ne sabiraju na 1)

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad s = f(x_i; W)$$

Maksimizujemo log verovatnost (log likelihood) – obično se minimizuje negativna log verovatnost:

$$L_i = -\log P(Y = y_i|X = x_i)$$

$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

Zašto koristimo log od funkcije greške?

Zato što pomocu ove funkcije greške, vrlo pouzdane pogrešne predikcije proizvode značajnu grešku, pa ce samim tim optimzacioni algoritam da se trudi da izbegne takve predikcije. To nam je i cilj! *(verovatnoda prave klase treba da je bilzu 1, a verovatnoda koju je dao model je blizu 0)

nenormalizovane log verovatnoće ---(exp)--> nenormalizovane verovatnoće --(normalizacija)--> verovatnoce

Optimizacija - Strategija #1: Random pretraga

Strategija #2: Pratimo nagib Kod funkcije jedne promenljive, izvod funkcije dat je sa:

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Kad imamo više promenljivih (atributa), gradijent je vektor parcijalnih izvoda po svim atributima.

Numerički izvod nije dobar pristup ovde. Greška je funkcija od W, pa možemo da koristimo analitički izvod.

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$s = f(x; W) = Wx$$

treba nam $\nabla_W L$

Rezime:

- Numerički izvod: aproksimacija, spor, ali jednostavna formula

- Analitički izvod: tačan, brz, nije ga uvek lako naći, pa postoji mogućnost greške

U praksi: Uvek koristimo analitički izvod, ali ga proverimo numerički. Ovo se zove provera gradijenta (gradient check).

Gradijetni Spust sa Mini-Podskupovima

- koristimo samo deo obučavajućeg skupa da izračunamo gradijent.

Tipične veličine podskupova su 32/64/128/256 primera

Funkcija greške opada kroz iteracije - epohe.

Kako pronaći minimum funkcije?

• Izvod funkcije predstavlja nagib tangente na krivu funkcije

• Ideja: iterativno demo se pomerati ka minimumu

• levo od minimuma: gradijent je negativan – pomeramo se u desno

• desno od minimuma: gradijent je pozitivan – pomeramo se u levo

Gradient descent

- **Optimizaciona tehnika**: data je (proizvoljna) funkcija $J(\theta)$. Želimo da pronađemo $\min_{\theta} J(\theta)$

Ulaz	• $J(\theta)$ – funkcija koja se optimizuje • θ_0 – početno rešenje • α – *learning rate* (veličina koraka) • *maxIters* – maksimalan broj iteracija
Postupak	for t = 1, 2,..., maxIters: for d = 1,2,...,D • $\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha \frac{\partial}{\partial \theta_d} J(\theta)$
Izlaz	θ – tačka u kojoj funkcija $J(\theta)$ ima minimum

Gradijent – određuje smer pretrage

Primena na linearnu regresiju

- Fitovanje modela: želimo da pronađemo parametre θ za koje funkcija greške ima najmanju vrednost

$$J(\theta) = \frac{1}{2N} \sum_{i=1}^{N} \left(\theta_1 x^{(i)} + \theta_0 - y^{(i)}\right)^2$$

- Ovo možemo uraditi primenom *gradient descent* algoritma:
  - Ponavljati do konvergencije:

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha \frac{\partial}{\partial \theta_d} J(\theta) \text{ za } d \in \{0,1\}$$

- Ako bismo imali samo jedan primer u skupu podataka, pravilo za ažuriranje parametara je:

$$\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha (h_\theta(x) - y) x_d$$

- Ovo pravilo se naziva LMS (Least Mean Squares) update rule ili Widrow-Hoff learning rule
- Magnituda promene je proporcionalna greški $h_\theta(x) - y$

Batch GD vs. Stohastic GD

- *Batch GD*: u svakom koraku istovremeno ažuriramo parametre modela koristeći *sve* trening podatke

for t = 1, 2,..., maxIters:

for d = 1,2,...,D

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \frac{\alpha}{N}\sum_{i=1}^{N}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_d{}^{(i)}$

moramo da izračunamo grešku za sve primere pa tek onda radimo promenu

- *Stohastic GD* (ili *Incremental GD*): više puta prolazimo kroz skup podataka. Kad god naiđemo na trening podatak, ažuriramo gradijent na osnovu tog (jednog) trening podatka

for t = 1, 2,..., maxIters:

for d = 1,2,...,D

for i = 1,2,..., N

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \alpha\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_d{}^{(i)}$

nema sume ovde, promenu radimo odmah za svaki primer

Mini-Batch Stohastic GD

- *Batch GD*: u svakom koraku istovremeno ažuriramo parametre modela koristeći *sve* trening podatke

batch_size = 32

for t = 1, 2,..., maxIters:

mini_batch = slučajno odabrana 32 primera iz skupa podataka

for d = 1,2,...,D

- $\theta_d^{(t+1)} = \theta_d^{(t)} - \frac{\alpha}{batch\_size}\sum_{i=1}^{batch\_size}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)x_d{}^{(i)}$

ovde koristimo samo primere iz skupa mini_batch

Mini-Batch Stohastic GD

- Predstavlja ravnotežu koja ispravlja mane prethodna dva algoritma
- Najčešće se koristi u praksi

Gradient descent može da konvergira u minimum za fiksiranu vrednost α: kako se približavamo minimumu koraci su sve manji jer je gradijent (nagib) sve manji

Ako se nalazimo levo od minimuma vrednost gradijenta je negativna: θ de da raste

Ako se nalazimo desno od minimuma vrednost gradijenta je pozitivna: θ se smanjuje

Za male vrednosti α gradient descent je spor

Za preveliko α gradient descent ne konvergira, a može čak i da divergira