S.KOMOLOV
SH.RAXMATOV

SUN'IY INTELLEKT ASOSLARI

MASHINAVIY O'QITISH

S.KOMOLOV
SH.RAXMATOV

SUN'IY INTELLEKT ASOSLARI

MASHINAVIY O'QITISH

821.004.8
32-813
K 64

K 64 Sirojiddin Komolov, Sherzod Raxmatov.
Sun'iy intellekt asoslari. Mashinaviy o'qitish. [Matn]:
/ S.Komolov, Sh.Raxmatov – Toshkent: "Ijod NASHR" nashriyoti,
2022. – 104 b.

ISBN 978-9943-6561-8-5

Innopolis University

2022

ISBN 978-9943-6561-8-5

Mundarija

Kirish

Ushbu kitob sun'iy intellektning mashinaviy o'qitish yo'nalishi haqidagi qo'llanma bo'lib, mashinaviy o'qitish asoslarini o'zida qamrab oladi. Qo'llanma o'zida bir qancha mashinaviy o'qitish algoritmlarini jamlagan bo'lib, har bir algoritmni yoritishda nazariy bilim bilan birgalikda, amaliy mashg'ulot ham berib boriladi. Kitobda keltirilgan bilimlarni egallashda oliy matematika asoslari hamda Pyton dasturlash tili ko'nikmalari talab etiladi.

Shuni alohida qayd etish joizki, qo'llanmadagi barcha amaliy mashg'ulotlarda mashinaviy o'qitish algoritmlari hech qanday tayyor mashinaviy o'qitish kutubxonalarisiz noldan yoziladi. Ushbu uslub garchi tayyor mashinaviy o'qitish kutubxonalariga nisbatan qiyin va ko'p vaqt talab etsa-da, ammo o'quvchining mashinaviy o'qitish sohasidagi bilim va ko'nikmalariga kuchli poydevor qo'yadi. Sababi ko'p hollarda sun'iy intellektning real hayotga qo'llanmasi dasturchidan mashinaviy o'qitish algoritmlarining ma'nosini tushunishni va muammoga qarab moslashtirishni talab qiladi. Shu sababli nazariy bo'limlarda ham algoritmlarga matematik nuqtayi nazardan ahamiyat beriladi. Albatta, keng tarqalgan oddiy masalalarni yechishda ma'lumotlar to'plami bilan qulay va effektivroq ishlash uchun Pyton dasturlash tilidagi ma'lum kutubxonalardan foydalaniladi. Xususan, *numpy* – Pytonda massivlar bilan ishlash uchun, *matplotlib* – Pytonda vizualizatsiya, diagrammalar bilan ishlash uchun va hokazolar shular jumlasidandir.

Kitobda yoritilgan mavzulardagi algoritmlarning amaliy mashg'ulotini o'quvchi o'zi mustaqil ravishda yechishi tavsiya etiladi. Shu tariqa o'quvchi algoritm xususidagi bilim va ko'nikmalarini yanada mustahkamlaydi hamda real hayotdagi masalalarga mustaqil ravishda tatbiq eta oladi. Amaliy mashg'ulotlardagi kodlar *github.com* saytidagi repozitoriyga joylashtirilgan bo'lib, ushbu repozitoriyga `https://bit.ly/3qRgBsb` ssilkasi orqali yoki quyidagi QR-kodni skaner qilish orqali o'tishingiz mumkin.

Mualliflar ushbu kitobda xatolarni imkon qadar kamaytirishga harakat qilishdi. Yo'l qo'yilgan xato va kamchiliklar uchun oldindan uzr so'raydi. Ushbu

kitob bo'yicha barcha fikr va mulohazalaringizni `s.komolov01@gmail.com` e-mail manziliga yuborishingizni so'rab qolamiz.

Tashakkurnomalar

Mualliflar ushbu kitob yaratilishida bevosita va bilvosita o'z hissasini qo'shgan barcha insonlarga o'z minnatdorligini bildirib qoladi. Xususan, mashinaviy o'qitish sohasida ekspertlar bo'lmish *Yusuf Yusriy Nasr Ibrohim* va *Imod Eddina Ibrohim Bekkouch*larning yo'l-yo'riqlarini qadrlaydi. Ushbu qo'llanma mualliflarning ilk kitobi bo'lgani uchun *Professor Manuel Mazzara* hamda *Nursulton Askarbekuli*larning ilmiy sohadagi rahbarlik va yo'riqnomalari uchun minnatdorlik bildirib o'tadi. Qadrli ustozimiz *Asad Karimov*ga matematika fanidan bergan saboqlari uchun alohida rahmat aytib qolamiz. Va albatta, bizni qo'llab turgan oila a'zolarimizga, jumladan, ota-onalarimizga ham o'z minnatdorligimiz hamda cheksiz mehr-muhabbatimizni izhor qilamiz.

Acknowledgements

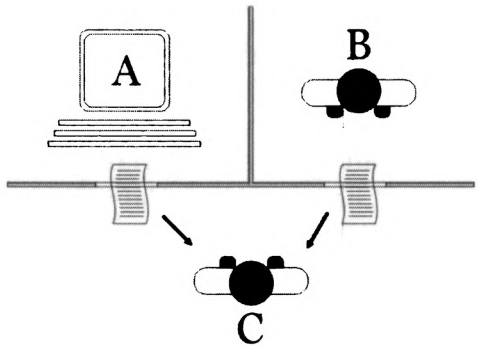
The authors would like to thank all those who have made explicit and implicit contributions to this book. The authors especially appreciate the recommendations from *Youssef Youssry Nasr Ibrahim* and *Imad Eddine Ibrahim Bekkouch*, who are experts in the machine learning field. Since this is the first book of the authors, we also express our gratitude to *Professor Manuel Mazzara* and *Nursultan Askarbekuly* for their teachings and instructions in the academic and research field. Special thanks to our esteemed teacher *Asad Karimov* for his guidance on mathematics. Lastly, we would like to thank our family and our parents for their great support.

1 Sun'iy intellekt o'zi nima?

1950-yilda amerikalik matematik Alan Turing butun dunyoni o'zgartirib yuborgan oddiy savolni o'rtaga tashladi:

- Mashinalar o'ylay oladimi?

Turingning 1950-yildagi "Hisoblash texnikasi va intellekt" ("Computing machinery and intelligence") deb nomlangan maqolasi hamda unda yoritilgan "Turing testi" bugungi kundagi sun'iy intellektga asos bo'ldi, desak to'g'ri bo'ladi.



Rasm 1: Turing testi

Manba: [12]

1-rasmda "Turing testi"ning interpretatsiyasi keltirilgan. Bunda C (baholovchi), A (kompyuter) hamda B (inson)lardan ularni ko'rmasdan, savollarga matn formatida javob oladi. Bunda baholovchi qaysi javob kompyuter tomonidan, qaysinisi inson tomonidan yozilganini topishi kerak. Kompyuterning javobi inson javobiga qanchalik yaqinlashsa, kompyuterning o'ylashi (sun'iy intellekti) shuncha yuqori bo'ladi.

1.1 Mashinaviy o'qitish haqida

Sun'iy intellekt amallari turli xil usullar yordamida amalga oshiriladi, hamda mashinaviy o'qitishning eng keng tarqalgan usullardan biridir. Ushbu qo'llanma sun'iy intellektning mashinaviy o'qitish usulini yoritishga harakat qiladi. 1959-yilda Artur Samuel (mashinaviy o'qitishning eng birinchi mutaxassislaridan biri) mashinaviy o'qitishga shunday ta'rif bergan edi:

"Mashinaviy o'qitish shunday sohaki, bunda kompyuterlar to'g'ridan-to'g'ri dasturlanmasdan o'rganish qobiliyatiga ega bo'ladi".

Albatta, bu juda keng ko'lamdagi tavsif bo'lib, Tom Mitchel ancha formal xarakterga ega bo'lgan tavsif bera oldi:

"Kompyuter dasturi T vazifadan E tajribani o'rganadi, bunda dastur effektivligi P orqali o'lchanadi, agar T vazifani yechishdagi effektivlik (ya'ni P orqali o'lchanuvchi) E tajriba orqali oshsa".

1.2 Sun'iy intellekt hayotimizda

Bugungi kunda juda ko'p dasturiy mahsulotlar sun'iy intellektdan foydalanadi. Xususan, quyida sun'iy intellektdan samarali tarzda foydalanayotgan va hayotimizda keng tarqalgan dasturiy xizmatlarning qisqacha ro'yxati keltirilgan:

- **Gmail xizmati.** Gmail dasturi xabarlarni kategoriyalarga bo'lishda (Asosiy, ijtimoiy va h.k.) yoki spam xabarlarni aniqlashda sun'iy intellektdan foydalanadi.
- **Google xizmati.** Bu qidiruvda harflarni terganingiz sari Google sizga mos keluvchi eng yaxshi tavsiyalar bera boshlaydi. Ushbu funksional ham sun'iy intellekt yordamida mukammallashtirilgan.
- **Tovarlar tavsiyasi.** Amazon va shunga o'xshash onlayn xarid xizmatlari sizning ta'bingizdan kelib chiqib, sizga tovarlarni tavsiya eta boshlaydi.
- Avtomobillardagi o'zini o'zi boshqarish (avtoboshqaruv) xususiyati ham sun'iy intellekt orqali amalga oshiriladi.

1.3 Mashinaviy o'qitish turlari

Oldingi bo'limda ta'kidlab o'tganimizdek, sun'iy intellektning asosiy tarmog'i mashinaviy o'qitish hisoblanadi. Mashinaviy o'qitishning ham turli toifalari mavjud. Ma'lumotlarning qay tarzda foydalanilishiga nisbatan mashinaviy o'qitishni quyidagi toifalarga ajratishimiz mumkin:

- nazoratli o'qitish (supervised learning);
- nazoratsiz o'qitish (unsepervised learning);
- kuchaytirilgan o'qitish (reinforcement learning).

1.3.1 Nazoratli o‘qitish

Ushbu turdagi o‘qitish eng keng tarqalgan usul bo‘lib, belgilangan ma’lumotlardan foydalangan holda kompyuter dasturi o‘qitiladi (training). Nazoratli o‘qitishning o‘zini ham 2 toifaga ajratish mumkin:

- Regressiya (Regression);
- Klassifikatsiyalash (Tasniflash) (Classification).

Regressiyaga misol tariqasida berilgan uy maydoni va narxlari asosida mashinali o‘qitishni keltirishimiz mumkin (bunda belgi sifatida, inglizcha *label* – narx ishtirok etadi). Tasavvur qilaylik, bizda 1-jadvaldagi ma’lumotlar to‘plami mavjud.

Uy maydoni	**Uy narxi**
$50\,m^2$	20 000 $
$60\,m^2$	22 000 $
$100\,m^2$	40 000 $

Jadval 1: Uy narxlari

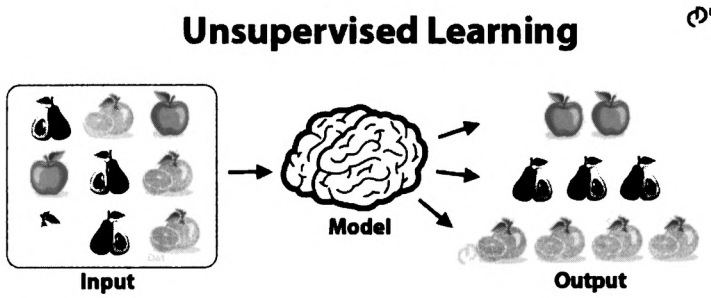
1-jadvalda keltirilgan ma’lumotlardan foydalanib, mashinaviy o‘qitish amalga oshiriladi va o‘qitilgan model uyning narxini uning maydoniga qarab topa oladi. Ya’ni ushbu modelga ma’lum uy maydoni kiritiladi va model (mashinaviy o‘qitishda “model" deganda mashinaviy o‘qitishni amalga oshirish natijasida hosil bo‘lgan faylga aytiladi va ushbu fayldan foydalangan holda prognoz (oldindan xulosa) amalga oshiriladi) uyning narxini prognoz qila oladi. Ko‘rib turibmizki, regressiyada model qiymati davomiy bo‘lgan belgi bilan ishlaydi. Xususan, yuqorida ko‘rib o‘tgan misoldagidek uy narxi qiymati davomiy ko‘rinishga ega bo‘lib, ma’lum bir aniq belgilangan qiymatlari yo‘q.

Klassifikatsiyalash (Tasniflash)ga misol tariqasida esa berilgan rasmda mushuk bor yoki yo‘qligini aniqlaydigan modelni keltirishimiz mumkin. Tasavvur qiling, bizda belgilangan bir qancha mushuk rasmlari bor. Ular asosida biz modelni mushukni tanitishga o‘rgatamiz. Natijada modelga belgilanmagan rasm berilganda ushbu rasmda mushuk bor yoki yo‘qligini aniqlay oladi.

1.3.2 Nazoratsiz o‘qitish

Nazoratsiz o‘qitish usulidan maqsad belgilanmagan ma’lumotlar orqali modelni o‘rgatish hisoblanadi. Masalan, bizda bir qancha meva rasmlari bor

va ular belgilanmagan (ya'ni qaysi rasmda qanday meva borligi haqida bizda ma'lumot yo'q). Lekin nazoratsiz o'qitish usullaridan foydalanib modelni ushbu belgilanmagan rasmlar orqali o'qitamiz va bunda model o'rganish jarayonida ma'lumot birliklarining o'xshashlik alomatlariga qarab (rangi, shakli, mazasi va h.k.) tasniflay oladi hamda bizga 2-rasmda ko'rsatilgan natijani berishi mumkin.



Rasm 2: Nazoratsiz o'qitish

Manba: [9]

1.3.3 Kuchaytirilgan o'qitish

Kuchaytirilgan o'qitish metodi davom etayotgan jarayondagi maqbul harakatlarni mukofotlash va aksincha, nomaqbul harakatlarni jarimaga tortishga asoslanadi. Bu metodda aqlli agentdan (o'zi mavjud bo'lgan muhitda tajriba, foydalanuvchi kiritgan ma'lumotlar va turli shartlar asosida qaror qabul qila oladigan kompyuter dasturi elementi) foydalanib, kerakli natijaga erishiladi. Jarayon davomidagi mukofot va jarima sifatida ball sistemasidan foydalaniladi. Ya'ni maqbul harakatlar bajarilganda ball beriladi va nomaqbul harakatlar uchun ball olib tashlanadi. Shu tariqa agent faqat maqbul harakatlarni bajarishga rag'batlantiriladi.

Hozirga kunda mashinaviy o'qitishning ushbu usulini qo'llash barcha sohalarda ham keng tarqalmagan. Asosan, quyidagi sohalar kuchaytirilgan o'qitish metodidan keng foydalanishadi:

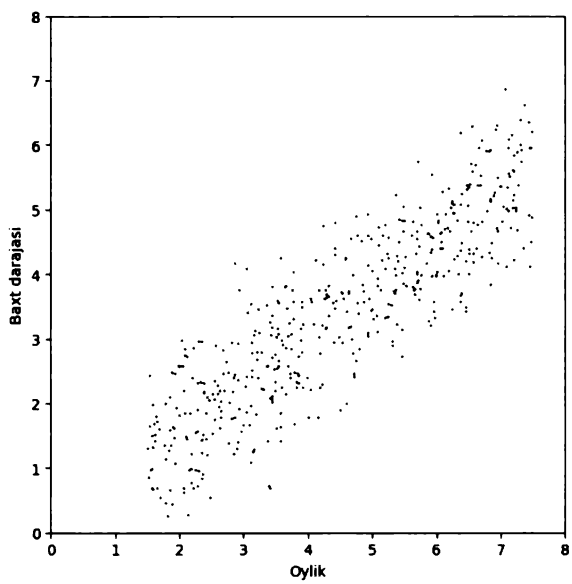
- o'yin dasturlari;
- resurslarni boshqarish;
- shaxsga moslashtirilgan tavsiyalar;
- robototexnika.

Bundan tashqari, kuchaytirilgan o'qitish ilmiy sohada ham keng tarqalgan bo'lib, hozirgi kunda ushbu metodning turli xil sohalarga qo'llanilishi xususidagi teoriyalar ishlab chiqilmoqda.

2 Bir o'zgaruvchili chiziqli regressiya

2.1 Umumiy tushunchalar

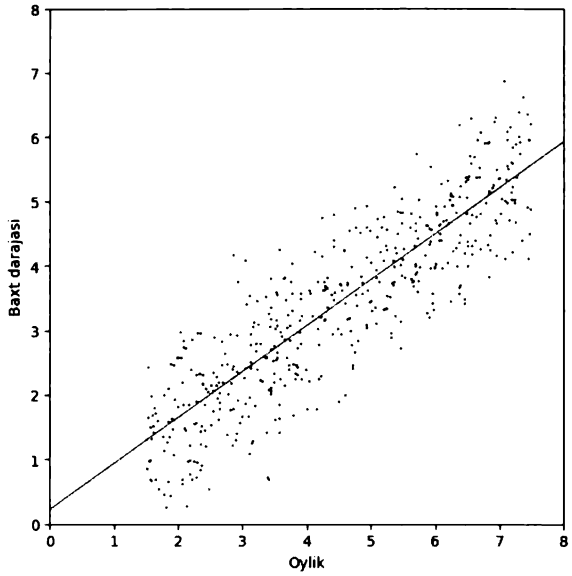
Chiziqli regressiya mashinaviy o'qitish sohasining eng boshlang'ich va oson usullaridan biri hisoblanadi. Ushbu usul yordamida davomiy bo'lgan qiymatlar prognoz qilinadi. Chiziqli regressiyaga misol qilib 3-rasmda keltirilgan oylik maosh hamda baxt darajasi o'rtasidagi munosabatni ko'rsatuvchi ma'lumotlar to'plamini misol qilib keltirishimiz mumkin. Ma'lumotlar to'plami bizga umumiy shunday ko'rinish beryaptiki, *Oylik* o'zgaruvchisi oshgan sari *Baxt darajasi* qiymati ham ortib bormoqda.



Rasm 3: Chiziqli regressiya. Baxt darajasining Oylikka nisbati.

3-rasmda keltirilgan ma'lumotlar to'plami elementlari o'rtasidagi munosabatni aniq bir formula orqali ifoda qilish qiyin, ya'ni *Oylik* va *Baxt darajasi* o'zgaruvchilari o'rtasidagi munosabatda ma'lum bir shovqin (noaniqlik) bor. Biroq umumiy ko'rinishda baxt darajasining oylikka bog'liqligining ma'lum

bir yo'nalishini ko'ramiz va bu yo'nalishni chiziqli regressiya orqali ifodalash mumkin (4-rasm).



Rasm 4: Chiziqli regressiya

4-rasmda ko'rsatilgan qizil chiziq ushbu ma'lumotlar to'plamidagi munosabatni ifodalovchi eng aniq chiziq hisoblanadi hamda ushbu chiziqni quyidagi 1-formula orqali ifodalash mumkin:

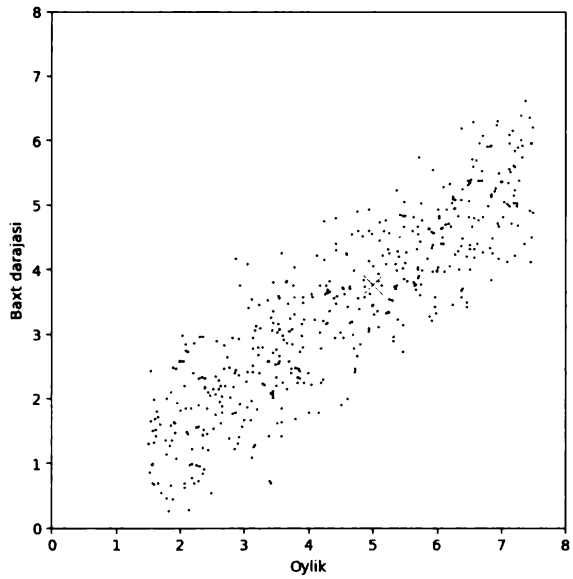
$$f(x) = a_0 + a_1 \times x \tag{1}$$

4-rasmda keltirilgan ma'lumotlar to'plamida 1-formuladagi x – oylikni, $f(x)$ ning qiymati esa baxt darajasini ifoda etadi. a_0 hamda a_1 konstantalar esa mana shu chiziqning yo'nalishini belgilab beradi va biz bu konstantalar yordamida oylik qiymat berilganda baxt darajasini topa olamiz. Xususan, yuqoridagi qizil chiziqning konstantalar qiymatlari aniqlangan holdagi formulasi quyidagichadir:

$$f(x) = 0.2 + 0.713 \times x \tag{2}$$

Tasavvur qiling, **oylik** o'zgaruvchisining shartli qiymati **5** bo'lganda, **baxt darajasi 3.765** shartli qiymatni tashkil etishini 2-formula orqali topa olamiz. Ya'ni biz ma'lum bir a_0 hamda a_1 konstantalar yordamida berilgan oylik orqali baxt darajasini prognoz qila oldik.

5-rasmda biz prognoz qilgan baxt darajasi bizga berilgan ma'lumotlar to'plamida qanday joylashishini ko'rishimiz mumkin va ta'kidlash joizki, bizning prognozimiz qoniqarli joylashuvga ega.



Rasm 5: Prognoz qilingan Baxt darajasining ma'lumotlar to'plamida joylashuvi

Xulosa qilishimiz mumkinki, prognoz qilishimiz uchun biz a_0 hamda a_1 konstantalarni topishimiz kerak. Ushbu konstantalarning qay darajada qonirqali ekanini topishda esa qiymat funksiyasi yordam beradi.

2.2 Qiymat funksiyasi

Aynan chiziqli regressiya bilan ishlaganda bizning maqsadimiz – ma'lumotlarni ifoda etuvchi eng yaxshi regressiya chizig'ini topish. Agarda ushbu regressiya chizig'i 1-formula orqali ifoda etiladigan bo'lsa, u holda a_0 hamda a_1 koeffitsiyentlarni topishimiz kerak. Bunda ushbu konstantalar orqali prognoz qilingan natija ma'lumotlar to'plamida keltirilgan haqiqiy qiymatga imkon qadar yaqin bo'lishi, ya'ni prognoz qilingan qiymat va haqiqiy qiymat o'rtasidagi tafovut imkon qadar kam bo'lishi kerak. Ushbu tafovutni **xatolik** deb ataymiz. Qiymat funksiyasining asosiy maqsadi ushbu xatolikni aniqlashdir.

2.2.1 O‘rta kvadratik xatolik funksiyasi

Qiymat funksiyasi orqali biz regressiya koeffitsiyentlari (a_0 va a_1) qanchalik to‘g‘ri topilganini aniqlashimiz, ya’ni ushbu koeffitsiyentlar orqali prognoz qilingan va haqiqiy qiymatlar o‘rtasidagi xatolikni (tafovutni) topishimiz mumkin. Buning uchun qiymat funksiyasi sifatida biz quyidagi o‘rta kvadratik xatolik funksiyasidan foydalanamiz:

$$J = \frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2 \tag{3}$$

3-formulada m – ma’lumotlar to‘plamidagi elementlar soni, y_i – ma’lumotlar to‘plami i-qatorining haqiqiy qiymati (belgi qiymati), $f(x_i)$ – i-qatordagi prognoz qilingan qiymat. Ushbu formuladagi $f(x_i)$ funksiyani bir chiziqli regressiya sifatida yoysak, quyidagi formula kelib chiqadi:

$$J = \frac{1}{m}\sum_{i=1}^{m}((a_0 + a_1 \times x_i) - y_i)^2 \tag{4}$$

Prognoz qilingan qiymat haqiqiy qiymatdan qanchalik uzoqda bo‘lsa, o‘rta kvadratik xatolik funksiyasi ham shuncha katta bo‘ladi. O‘rta kvadratik xatolik funksiyasi hisoblagan xatolikka qarab biz regressiya koeffitsiyentlarini (a_0 va a_1) yangilashimiz, ya’ni yanada yaxshiroq prognoz qilish va xatolikni kamaytirishimiz kerak.

2.2.2 Gradient tushish

Gradient tushish usuli bizga regressiya koeffitsiyentlarini yangilash orqali xatolikni kamaytirishga imkon beradi. Odatda o‘qitish jarayoni avvalida koeffitsiyentlarga (a_0 va a_1) tasodifiy qiymat beriladi. Keyin qiymat funksiyasini hisoblaymiz (J) va xatolikni topamiz. Bizning asosiy maqsadimiz xatolikning qiymatiga qarab qiymat funksiyasini (J), chiziqli regressiya misolida esa o‘rta kvadratik funksiyasini (3-formula) minimallashtirishdir:

$$minimallashtirish\frac{1}{m}\sum_{i=1}^{m}(f(x_i) - y_i)^2 \tag{5}$$

Yuqorida keltirilgan formuladagi prognoz qiymatini $f(x_i)$ formulasi orqali yoysak, quyidagi formula kelib chiqadi:

$$minimallashtirish\frac{1}{m}\sum_{i=1}^{m}((a_0 + a_1 \times x_i) - y_i)^2 \tag{6}$$

Bunda koeffitsiyentlarga (a_0 va a_1) optimal qiymat berilishi orqali xatolikni kamaytirishimiz mumkin. Xo'sh, ushbu optimal qiymatlarni qanday aniqlaymiz? Buning uchun qiymat funksiyasidan koeffitsiyentlarga (a_0 va a_1) nisbatan hosila olamiz.

$$J = \frac{1}{m}\sum_{i=1}^{m}(a_0 + a_1 \times x_i - y_i)^2 \tag{7}$$

$$\Downarrow$$

$$\begin{cases} \frac{\delta J}{\delta a_0} = \frac{2}{m}\sum_{i=1}^{m}(a_0 + a_1 \times x_i - y_i) \Rightarrow \frac{2}{m}\sum_{i=1}^{m}(f(x_i) - y_i) \\ \frac{\delta J}{\delta a_1} = \frac{2}{m}\sum_{i=1}^{m}(a_0 + a_1 \times x_i - y_i) \times x_i \Rightarrow \frac{2}{m}\sum_{i=1}^{m}(f(x_i) - y_i) \times x_i \end{cases} \tag{8}$$

Hosilalarni hisoblab olganimizdan keyin (a_0 va a_1) koeffitsiyentlarni yangilashga o'tishimiz mumkin va bunda quyidagi formuladan foydalaniladi:

$$\begin{cases} a_0 = a_0 - \alpha \times \frac{2}{m}\sum_{i=1}^{m}(f(x_i) - y_i) \\ a_1 = a_1 - \alpha \times \frac{2}{m}\sum_{i=1}^{m}(f(x_i) - y_i) \times x_i \end{cases} \tag{9}$$

$$\Downarrow$$

$$\begin{cases} a_0 = a_0 - \alpha \times \frac{2}{m}\sum_{i=1}^{m}((a_0 + a_1 \times x_i - y_i) - y_i) \\ a_1 = a_1 - \alpha \times \frac{2}{m}\sum_{i=1}^{m}((a_0 + a_1 \times x_i - y_i) - y_i) \times x_i \end{cases} \tag{10}$$

Yuqorida ko'rsatilgan formuladagi α – o'rganish darajasi bo'lib, koeffitsiyent yangilanish jarayonining o'zgarish kattaligini belgilaydi, ya'ni o'rganish darajasi qancha katta bo'lsa, koeffitsiyent shuncha katta miqdorda yangilanadi. Lekin bunda optimal qiymatni yo'qotib qo'yish ehtimoli ham shuncha katta bo'ladi. O'rganish darajasi qancha kichik bo'lsa, optimal qiymatni yo'qotish ehtimoli ham shuncha kichiklashadi, biroq optimal qiymatgacha juda ko'p qadam tashlash kerak bo'ladi. Shuning uchun ushbu qiymat optimizatsiya qilinadigan (muvofiqlashtiriladigan) parametr hisoblanadi. O'rganish darajasi haqida esa keyingi bo'limlarda batafsil ma'lumot beramiz.

10-formulada koeffitsiyentlarning qiymatlari parallel tarzda yangilanadi. Masalan, tasavvur qiling, hozir 10-qadam ketmoqda. Biz a_0 koeffitsiyentni

yangilaganimizda formuladagi a_0 va a_1 koeffitsiyentlarning qiymatlari – 9-qadamdagi qiymatlardan foydalanamiz. Hatto a_0 koeffitsiyentni yangilashdan so'ng a_1 koeffitsiyentni yangilashga o'tganimizda ham 10-formuladagi a_0 va a_1 koeffitsiyentlarning qiymatlarida 9-qadamdagi qiymatlardan foydalanamiz.

Xulosa qilib aytish mumkinki, gradient tushishni amalga oshirishda 10-formulada ifoda etilgan amallarni bir necha marta takrorlashimiz kerak. Bunda har safar biz yangi koeffitsiyentlarga ega bo'lamiz va ushbu koeffitsiyentlar orqali xatolikni hisoblab, ularning qanchalik optimal ekanligi haqida tasavvurga ega bo'lamiz.

2.3 Qisqa takrorlash

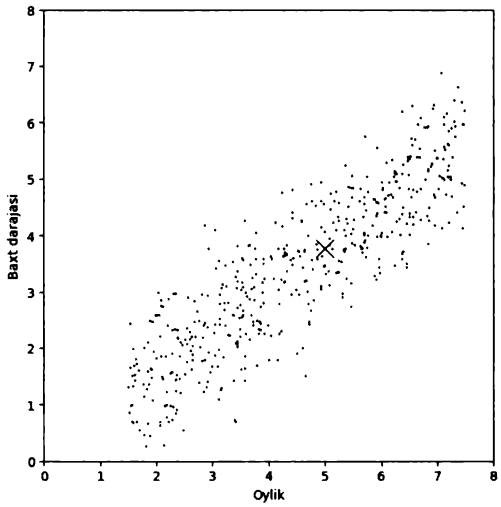
Yuqoridagilarni umumiylashtirgan holda bir o'zgaruvchili chiziqli regressiyani yechishda quyidagi algoritmni keltirishimiz mumkin:

1. Chiziqli funksiya regressiya formulasini aniqlab olish;
2. Qiymat funksiyasi formulasini aniqlab olish;
3. Gradient tushish formulasini aniqlab olish;
4. Gradient tushish uchun parametrlar (o'rganish darajasi – α, qadamlar soni)ni aniqlab olish;
5. Gradient tushishni ma'lum marta (oldindan belgilangan qadamlar sonicha) amalga oshirish. Bunda har qadamda koeffitsiyentlarni yangilab, xatolikni qiymat funksiyasi orqali hisoblab borish.

3 Ko'p o'zgaruvchili chiziqli regressiya

Oldingi bo'limda 6-rasmda keltirilgan ma'lumotlar to'plamiga bir o'zgaruvchili chiziqli regressiyani tatbiq qilgan edik:

Ya'ni bir o'zgaruvchili chiziqli regressiya Oylik va Baxt darajasi haqidagi ma'lumotlar to'plamiga to'g'ri keldi. Sababi bizda faqat birgina o'zgaruvchi bor edi – Oylik ustuni (bunda Baxt darajasi o'zgaruvchisidan belgi sifatida foydalanib, prognoz qilinuvchi ustun sifatida qabul qilingan). Endi tasavvur qiling, bizda shunday ma'lumotlar to'plami mavjudki, bunda prognoz bir necha o'zgaruvchiga bog'liq. Masalan, 2-jadvalda keltirilgan ma'lumotlar to'plamini [3] olaylik.



Rasm 6: Prognoz qilingan Baxt darajasining ma'lumotlar to'plamida joylashuvi

№	Mashina og'irligi	Dvigatel hajmi	Ot kuchi	Narxi
1	2548	130	111	**13495**
2	2548	130	111	**16500**
3	2823	152	154	**16500**
4	2337	109	102	**13950**

Jadval 2: Mashina narxlari

Mashina narxlari ko'rsatilgan jadvalda mashina narxi bir necha o'zgaruvchi, ya'ni mashina og'irligi, dvigatel hajmi va ot kuchiga bog'liq.

Shu o'rinda biz foydalaniladigan ba'zi bir tushunchalarni aniqlashtirib olaylik:

- m — ma'lumotlar to'plami soni (qatorlar soni);
- n — o'zgaruvchilar soni (ustunlar soni);
- $x^{(i)}$ — i-qatordagi o'zgaruvchilar;
- $x_j^{(i)}$ — i-qatordagi j-o'zgaruvchining qiymati;
- X — ma'lumotlar to'plamidagi o'zgaruvchilarni ifoda etuvchi matritsa. Bunda ushbu matritsaning o'lchami $(m \times n)$;
- a_j — j-koeffitsiyent;

- A — barcha koeffitsiyentlarni ifodalovchi matritsa yoki vektor. Uning o'lchami $(n \times 1)$;
- $y^{(i)}$ — i-qatordagi prognoz qilinadigan belgining (ba'zida nishon deb ham ataymiz) ma'lumotlar to'plamidagi asl qiymati;
- $\hat{y}^{(i)}$ — i-qatordagi prognoz qilingan nishon qiymati;
- $\hat{Y}$ — prognoz qilingan barcha nishonlarni ifodalovchi birlik matritsa yoki vektor. Uning o'lchami $(1 \times m)$;
- $f(x)$ — prognozni hisoblash funksiyasi.

Xususan, 2-jadvalda keltirilgan ma'lumotlar to'plamida:

- $m = 4$, ya'ni ma'lumotlar to'plami soni (qatorlar soni) 4 ga teng;
- $n = 3$, o'zgaruvchilar (ustunlar) soni 3 ga teng. Bunda o'zgaruvchilarni *mashina og'irligi*, *dvigatel hajmi* va *ot kuchi* ustunlari tashkil etadi;
- $x^{(1)} = [2548, 130, 111]$, ya'ni 1-qatorda keltirilgan *mashina og'irligi*, *dvigatel hajmi* va *ot kuchi* ustunlarining qiymatlar to'plami;
- $x_2^{(1)} = 130$, ya'ni 1-qatordagi 2-o'zgaruvchining (*dvigatel hajmi*) qiymati;
- $y^{(1)} = 13495$, ya'ni 1-qatorda prognoz qilinadigan nishonning ma'lumotlar to'plamidagi asl qiymati;
- α — o'rganish darajasi (inglizchada *learning rate*);
- *davr* — gradient tushishdagi qadamlar soni (inglizchada *epochs*);

Ushbu ma'lumotlar to'plamiga bir o'zgaruvchili chiziqli regressiya to'g'ri kelmaydi. Sababi bu yerda bir qancha o'zgaruvchilar ishtirok etmoqda. Shuning uchun bu ma'lumotlar to'plamiga ko'p o'zgaruvchili chiziqli regressiyani qo'llaymiz va bunda formula quyidagicha ko'rinish oladi:

$$f(x) = a_0 + a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 + \cdots + a_n \times x_n \tag{11}$$

Yuqoridagi formulada $x_1, x_2 \ldots x_n$ lar o'zgaruvchilar, $a_0, a_1, a_2 \ldots a_n$ lar esa koeffitsiyentlardir. Ushbu formulani 2-jadvalda ko'rsatilgan Mashinalar narxi ma'lumotlar to'plamiga qo'llaydigan bo'lsak, quyidagi 12-formulani olamiz:

$$f(x) = a_0 + a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 \tag{12}$$

E'tibor bering, endi bizda 3 ta o'zgaruvchi ishtirok etmoqda. Bunda x_1 – *Mashina og'irligi*, x_2 – *Dvigatel hajmi* va x_3 – *Ot kuchi* ustunini ifoda etadi. $f(x)$ funksiyaning qiymati esa mashina narxini prognoz qiladi. 12-formulada x_1, x_2, x_3 o'zgaruvchilarning qiymatlari bizga berilgan ma'lumotlar to'plamida mavjud (2-jadval), lekin a_0, a_1, a_2 hamda a_3 koeffitsiyentlarning qiymatlari noma'lum. Koeffitsiyentlarning qiymatini gradient tushish orqali topamiz. Ko'p o'zgaruvchili chiziqli regressiya uchun qiymat funksiyasi hamda gradient tushishni hisoblashdagi amallarda matritsaviy ko'paytmalarni qo'llash orqali yuqori samaradorlikka erishishimiz mumkin. Bunday matritsaviy ko'paytma amallar mashinaviy o'qitish dasturlarida **vektorlangan hisoblash** deyiladi va keyingi bo'limda shu haqida batafsil suhbat quramiz.

3.1 Vektorlangan hisoblash

Ko'p o'zgaruvchili chiziqli regressiya uchun gradient tushish funskiyasini qulaylashtirish maqsadida vektorlangan hisoblashni ko'rib chiqaylik. Chiziqli regressiyaning ushbu turida juda ko'p o'zgaruvchilar qatnashishi mumkin. Bunda vektor hisob-kitoblar ancha qulaylik tug'diradi hamda hisoblash samaradorligining sur'atini oshiradi. Shu sababdan vektor hisoblarda qulaylik uchun ko'p o'zgaruvchili chiziqli regressiya formulasiga biroz o'zgartirish kiritamiz:

$$f(x) = a_0 \times x_0 + a_1 \times x_1 + a_2 \times x_2 + \cdots + a_n \times x_n \tag{13}$$

Bunda $x_0 = 1$, yangi x_0 o'zgaruvchi kiritildi. Aslida uning qiymati doim 1 ga teng bo'lib, vektor hisoblarda qulay bo'ladi.

13-formulani ma'lum bir ma'lumotlar to'plami qatorlariga moslashtirsak, quyidagi tenglikni olamiz:

$$\begin{aligned}
\hat{y}^{(1)} &= a_0 \times x_0^{(1)} + a_1 \times x_0^{(1)} + a_2 \times x_2^{(1)} + \cdots + a_n \times x_n^{(1)}, \\
\hat{y}^{(2)} &= a_0 \times x_0^{(2)} + a_1 \times x_0^{(2)} + a_2 \times x_2^{(2)} + \cdots + a_n \times x_n^{(2)}, \\
\hat{y}^{(3)} &= a_0 \times x_0^{(3)} + a_1 \times x_0^{(3)} + a_2 \times x_2^{(3)} + \cdots + a_n \times x_n^{(3)}, \\
&\vdots \\
\hat{y}^{(m)} &= a_0 \times x_0^{(m)} + a_1 \times x_0^{(m)} + a_2 \times x_2^{(m)} + \cdots + a_n \times x_n^{(m)}
\end{aligned} \tag{14}$$

14-formulani biz ma'lumotlar qatorining har bir qatori uchun tatbiq qildik va bunda $\hat{y}^{(1)}$ – 1-qatorning prognoz qilingan qiymati, $\hat{y}^{(2)}$ – 2-qatorning prognoz qiymati va shu tarzda $\hat{y}^{(m)}$ – m-qatorning prognoz qiymatidir. Endi ushbu

formula elementlarini matritsalar yordamida ifodalaymiz. Avval prognozlar matritsasini tuzib olaylik:

$$\hat{Y} = \begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(m)} \end{bmatrix} \tag{15}$$

E'tibor bering, bizda o'lchami $(m \times 1)$ bo'lgan $\hat{Y}$ matritsa hosil bo'ldi. Endi o'zgaruvchilar matritsasini tuzib olamiz:

$$X = \begin{bmatrix} x_0^{(1)}, & x_1^{(1)}, & \cdots & x_n^{(1)} \\ x_0^{(2)}, & x_1^{(2)}, & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{(m)}, & x_1^{(m)}, & \cdots & x_n^{(m)} \end{bmatrix} \tag{16}$$

Bizda $(m \times (n+1))$ o'lchamli matritsa hosil bo'ldi. Bunda m – qatorlar soni, $(n+1)$ – o'zgaruvchilar soni (x_0 ni ham hisobga olgan holda). Quyidagi formula esa koeffitsiyentlar matritsasini ifoda etadi:

$$A = \begin{bmatrix} a_0, & a_1, & \cdots & a_n \end{bmatrix} \tag{17}$$

Koeffitsiyentlar matritsasi o'zgaruvchilarning har bir ustuniga mos ravishda $(1 \times (n+1))$ o'lchamini oldi (Ya'ni qatorlar soni 1 ga teng va ustunlar soni $(n+1)$ qiymatiga teng) va buni vektor deb atasak ham bo'ladi. Vektor ko'paytirishlar uchun A vektorni transpozitsiya (qatorlarni ustunlar bilan almashtirish) qilamiz va ushbu matritsani 18-formulada ifodalaymiz.

$$A^T = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{18}$$

Natijada bizda o'lchami $((n+1) \times 1)$ bo'lgan matritsa, ya'ni vektor hosil bo'ldi. Endi biz X matritsani A^T vektorga ko'paytirsak bo'ladi.

$$\begin{bmatrix} \hat{y}^{(1)} \\ \hat{y}^{(2)} \\ \vdots \\ \hat{y}^{(n)} \end{bmatrix} = \begin{bmatrix} x_0^{(1)}, & x_1^{(1)}, & x_2^{(1)}, & \cdots & x_n^{(1)} \\ x_0^{(2)}, & x_1^{(2)}, & x_2^{(2)}, & \cdots & x_n^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_0^{(m)}, & x_1^{(m)}, & x_2^{(m)}, & \cdots & x_n^{(m)} \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{19}$$

Ko'p o'zgaruvchili chiziqli regressiyaning formulasini 19-formulada keltirilgan matritsalar orqali ifodalab oldik. Uning qisqa ko'rinishi quyidagicha ifodlanadi.

$$\hat{Y} = X \times A^T \tag{20}$$

20-formulada, $\hat{Y}$ – o'lchami $(m \times 1)$ bo'lgan vektor, X – o'lchami $(m \times (n+1))$ bo'lgan matritsa hamda A^T – o'lchami $((n+1) \times 1)$ bo'lgan vektordir.

3.2 Kirish o'zgaruvchilarining sayqallanishi

Gradient tushish tezligi o'zgaruvchilar qiymatlarining bir-biriga qanchalik yaqinligiga bog'liq. Ushbu jarayonni amalda ko'rish uchun mashina narxlari xususidagi jadvalni ko'rib chiqaylik. Ortiqcha murakkablashtirishdan xalos bo'lish uchun ushbu jadvalning faqatgina ikki ustuniga ahamiyat beraylik (3-jadval).

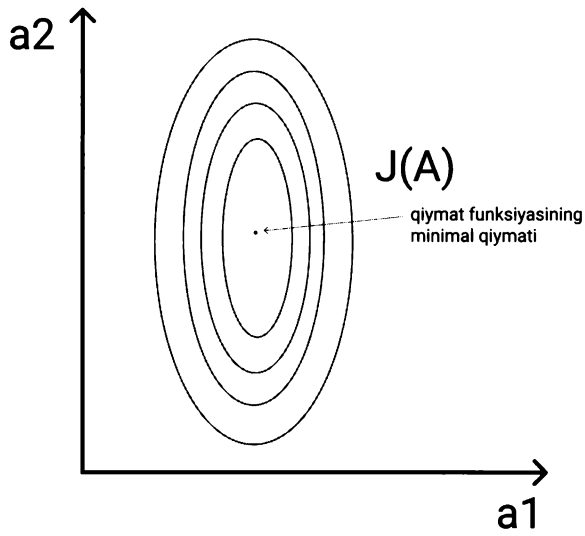
№	Mashina og'irligi (x_1)	Eshiklar soni (x_2)
1	2548	4
2	2548	4
3	2823	2
4	2337	4

Jadval 3: Mashina narxlari

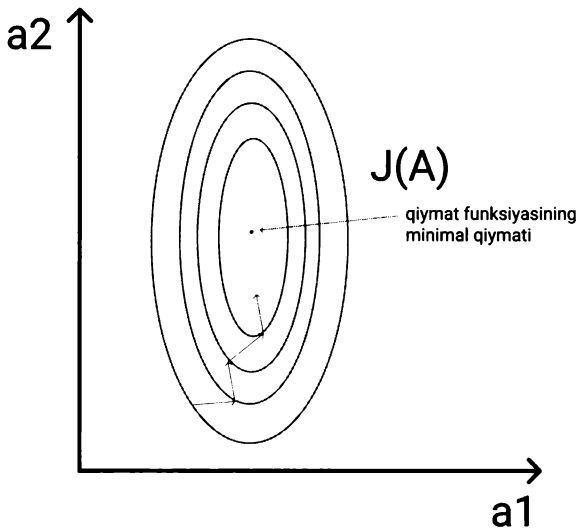
Mashina og'irligi hamda eshiklar soni qiymatlari bir-biridan keskin farq qiladi. Agarda biz shunday xususiyatga ega ma'lumotlar to'plami ustida gradient tushishni amalga oshirsak, unda turli qadamlardagi qiymat funksiyasi natijalari bir-biridan keskin farq qiluvchi qiymatlarga ega bo'ladi. Bunday holatda qiymat funksiyasining minimal qiymatini topish bizdan juda ko'p sonli qadamlarni talab qilishi mumkin. Bu esa, o'z navbatida, gradient tushishning sur'atini sekinlashtirib yuboradi. Ushbu jarayonni vizual tarzda ifodalash uchun 7-rasmda shartli ravishda ifodalangan qiymat funksiyasini ko'rib chiqaylik:

7-rasmda a_1 va a_2 (soddalashtirish uchun a_0 tashlab yuborildi) koeffitsiyentlarining turli kombinatsiyasida xatolik (qiymat) funksiyaning turli qiymatlari keltirilgan. Bunda har bir oval-aylana ma'lum bir qadamdagi xatolik funksiyasi qiymatlaridan iborat.

Gradient tushish esa, albatta, 7-rasmda ifodalangan qiymat funksiyasining minimal qiymatiga intiladi. 8-rasmdagi gradient tushishda qadamlarning ketma-ketligi vizual ko'rinib turibdi (gradient tushish qadamlari ketma-ketligi shartli ravishda ko'k chiziqlar orqali ifodalangan).



Rasm 7: Qiymat funksiyasining 2D o'lchamdagi shartli ko'rinishi



Rasm 8: Gradient tushishning 2D o'lchamdagi shartli ko'rinishi

8-rasmdan ko'rishimiz mumkinki, gradient tushish qadamlari "turli yo'nalishlarga ko'p burilib", o'z tezligini yo'qotmoqda. Buning sababi esa a_1 hamda a_2 koeffitsiyentlarning optimal qiymatini topish jarayonida, qiymatlari

orasida katta farq bo'lgan x_1 hamda x_2 o'zgaruvchilardan (3-jadvaldagi ma'lumotlar to'plami misolida *Mashina og'irligi* va *Eshiklar soni* ustunlaridan) foydalaniladi. Aynan mana shu katta farq tufayli optimal kombinatsiyani topish ko'p qadamlarni yuzaga keltiradi.

Ushbu muammoni yechishda kiruvchi o'zgaruvchilar qiymatlarini o'zgartirib, odatda 21-formulada ko'rsatilgan oraliqqa tushiriladi (albatta, boshqa kichik oraliqqa ham tushirsa bo'ladi, asosiy maqsad – o'zgaruvchilar o'rtasidagi tafovutni kamaytirish).

$$-1 \leq x_i \leq 1 \tag{21}$$

Ushbu kichik oraliqqa tushirishni *kirish o'zgaruvchilari sayqallanilishi* deb ataylik. Odatda Normallashtirish yoki Standartlashtirish usullari orqali amalga oshirish mumkin.

3.2.1 Normallashtirish

Kiruvchi o'zgaruvchilarni normallashtirish usulida, 22-formuladan foydalaniladi.

$$x_j^{(i)} := \frac{x_j^{(i)} - X_{j,min}}{X_{j,max} - X_{j,min}} \tag{22}$$

22-formulada $x_j^{(i)}$ – ma'lum i-qatordagi va j-ustundagi o'zgaruvchi, $X_{j,min}$ – j-ustundagi o'zgaruvchilarning eng kichik qiymati, $X_{j,max}$ – j-ustundagi o'zgaruvchilarning eng katta qiymati. Ushbu usulni 7-jadvalda ko'rsatilgan ma'lumotlar to'plamiga tatbiq qilsak (bunda biz normalizatsiyani ma'lumotlar to'plamining har bir ustuni uchun alohida qo'llaymiz):

Mashina og'irligi ustunidan boshlasak. Avvalo, eng kichik va eng katta qiymatlarni aniqlab olamiz. $x_{1,min} = 2337$, $x_{1,max} = 2823$.

Endi, mashina og'irligi ustunidagi har bir o'zgaruvchi uchun 22-formulani qo'llaymiz:

$$\begin{aligned} x_1^{(1)} &:= \frac{2548 - 2337}{2823 - 2337} \approx 0.434 \\ x_1^{(2)} &:= \frac{2548 - 2337}{2823 - 2337} \approx 0.434 \\ x_1^{(3)} &:= \frac{2823 - 2337}{2823 - 2337} = 1 \\ x_1^{(4)} &:= \frac{2337 - 2337}{2823 - 2337} = 0 \end{aligned} \tag{23}$$

23-formuladan ko‘rinib turibdiki, mashina og‘irligi ustunidagi o‘zgaruvchilarning qiymati $[0,1]$ oraliqqa tushirildi. Xuddi shunday tarzda eshiklar sonini ham $[0,1]$ oraliqqa tushirilsa, ikki ustun o‘zgaruvchilari o‘rtasidagi tafovut kamayadi hamda gradient tushish tezligi ortadi.

3.2.2 Standartlashtirish

Kirish o‘zgaruvchilarini standartlashtirishda 24-formuladan foydalaniladi:

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{S_j} \tag{24}$$

24-formulada $x_j^{(i)}$ – ma’lum i-qatordagi va j-ustundagi o‘zgaruvchi, μ_j – j-ustundagi o‘zgaruvchilarning o‘rta arifmetik qiymati, S_j – j-ustundagi o‘zgaruvchilarning o‘rta kvadratik og‘ishi (25-formula)

$$S_j = \sqrt{\frac{\sum (x_j^{(i)} - \mu_j)^2}{N_j}} \tag{25}$$

25-formula j-ustun uchun o‘rta kvadratik og‘ishni hisoblaydi va bunda $x_j^{(i)}$ – ma’lum i-qatordagi va j-ustundagi o‘zgaruvchi, μ_j – j-ustundagi o‘zgaruvchilarning o‘rta arifmetik qiymati, N_j – j-ustundagi o‘zgaruvchilar soni.

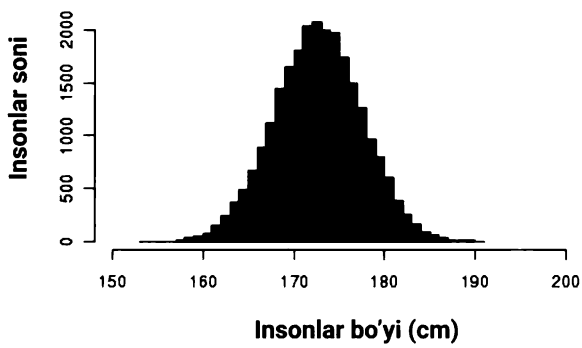
$$\mu_j = \frac{\sum_{j=1}^{N_j} x_j^{(i)}}{N_j} \tag{26}$$

Yuqorida keltirilgan formulalarni 7-jadvaldagi ma’lumotlar to‘plamiga, xususan, *mashina og‘irligi* ustuniga qo‘llaylik. Qulaylik uchun o‘rta arifmetik qiymat hamda o‘rta kvadratik og‘ishlarni hisoblab oldik: $\mu_1 = 2564$, $S_1 \approx 172.57$ (*Mashina og‘irligi* ustuni uchun) va $\mu_2 = 2564$, $S_2 \approx 172.57$ (*Eshiklar soni* ustuni uchun).

$$\begin{aligned} x_1^{(1)} &:= \frac{2548 - 2564}{172.57} \approx -0.092 \\ x_2^{(1)} &:= \frac{2548 - 2564}{172.57} \approx -0.092 \\ x_3^{(1)} &:= \frac{2823 - 2564}{172.57} \approx 1.5 \\ x_4^{(1)} &:= \frac{2337 - 2564}{172.57} \approx -1.315 \end{aligned} \tag{27}$$

27-tenglik bizga ko‘rsatib turibdiki, *mashina og‘irligi* ustunidagi o‘zgaruvchilar kichik tafovutli (eng katta qiymat 1.5 va eng kichik qiymat -1.315) oraliqqa tushirildi. Xuddi shunday qilib *eshiklar soni* ustunidagi qiymatlar ham standartlashtiriladi va kichik tafovutli oraliqqa tushiriladi hamda mashinalar og‘irligi va eshiklar soni o‘rtasidagi katta tafovut bartaraf etiladi.

Albatta, kiruvchi o‘zgaruvchilarni sayqallashda qaysi usulni tanlash kerak, degan tabiiy savol tug‘ilishi mumkin. Bunda ma’lumotlar to‘plamining xususiyatidan kelib chiqib qaror qabul qilish kerak. Ya’ni ma’lumotlar to‘plami normal taqsimlangan (**Gauss taqsimoti**) yoki yo‘qligiga qarab qaror qabul qilinishi mumkin. Shu o‘rinda Gauss taqsimoti haqida qisqacha ma’lumot berib ketsak. Ma’lumotlar to‘plami turlicha taqsimlangan bo‘lishi mumkin. Xususan, 9-rasmda insonlarning bo‘yi haqidagi ma’lumotlar to‘plamining taqsimoti aks ettirilgan. Bunda o‘rtacha bo‘yli (172 cm) insonlar soni eng ko‘p tarqalgan bo‘lib, insonlar soni eng past hamda eng baland bo‘ylilarga nisbatan tekis kamayib bormoqda. Ushbu turdagi ma’lumotlar taqsimoti normal taqsimot yoki Gauss taqsimoti deb ataladi.



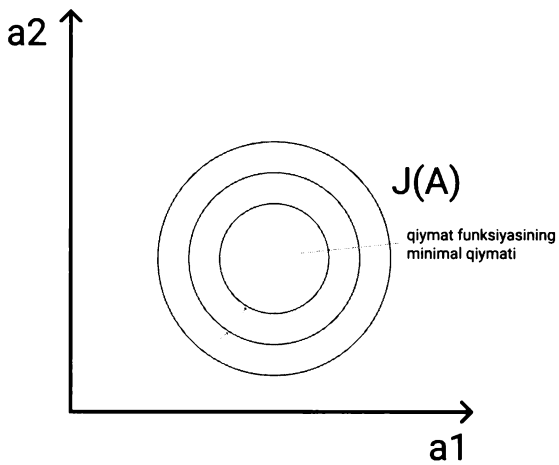
Rasm 9: Insonlar bo‘yi haqidagi ma’lumotlar to‘plamining taqsimoti [7]

Demak, ma’lumotlar to‘plamining taqsimotiga qarab sayqallash usulini tanlashimiz mumkun.

- **Standartlashtirish.** Agarda ma’lumotlar to‘plami normal taqsimlangan bo‘lsa (Gauss taqsimoti), unda standartlashtirishni tanlagan ma’qul.
- **Normallashtirish.** Agarda ma’lumotlar to‘plami normal taqsimlanmagan bo‘lsa (Gauss taqsimotiga bo‘ysunmasa), normallashtirishni tanlagan ma’qul.

Bundan tashqari, ma'lumotlar to'plamining boshqa xususiyatlari ham bo'lishi mumkin va eng ishonchli usuli har bir sayqallash usulini sinab ko'rib, eng yaxshi natijaga ega bo'lgan usulni tanlashdir.

Umuman olganda, sayqallash usulini qo'llab, o'zgaruvchilar qiymatlari o'rtasidagi tafovutni kamaytirgandan keyin gradient tushishni amalga oshirsak, har qadamda 10-rasmda shartli ravishda ko'rsatilgan qiymat funksiyasiga ega bo'lamiz. E'tibor bering, bunda gradient tushish qadamlari ortiqcha burilishlarsiz, qiymat funksiyasini minimallashtiruvchi koeffitsiyentlarning optimal qiymatlariga qarab harakatlanmoqda.



Rasm 10: Gradient tushishning 2D o'lchamdagi shartli ko'rinishi

3.3 Qiymat funksiyasi va gradient tushish

Gradient tushish funksiyasining asosiy vazifasi bu qiymat funksiyasini minimallashtiruvchi koeffitsiyentlarni topishdir. Shuning uchun ko'p o'zgaruvchili chiziqli regressiya uchun qiymat funksiyasini aniqlab olaylik:

$$J = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2 \tag{28}$$

Ya'ni ko'p o'zgaruvchili chiziqli regressiyaning qiymat funksiyasi bir o'zgaruvchili chiziqli regressiyaniki bilan deyarli bir xil, faqatgina hisob-kitoblarda yanada qulay bo'lishi uchun $\frac{1}{m}$ koeffitsiyentning o'rniga

$\frac{1}{2m}$ koeffitsiyentni kiritdik. Gradient tushishning asosiy maqsadi qiymat funksiyasini minimallashtirish bo'lgani uchun kiritilgan yangi koeffitsiyent umumiy maqsadni o'zgartirmaydi.

$$min\frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)})-y^{(i)})^2 \Leftrightarrow min\frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)})-y^{(i)})^2 \qquad (29)$$

Biz qiymat funksiyasini aniqlab oldik. Endi uni minimallashtirish uchun gradient tushish formulalariga o'tamiz. Bir o'zgaruvchili chiziqli regressiyadan bilamizki, funksiyani minimallashtirish uchun ushbu funksiyaning eng minimal qiymatga yetaklovchi koeffitsiyentlarini topishimiz kerak. Qiymat funksiyasini koeffitsiyentlar yordamida ifodalasak:

$$J=\frac{1}{2m}\sum_{i=1}^{m}((a_0\times x_0^{(i)}+a_1\times x_0^{(i)}+a_2\times x_2^{(i)}+\cdots+a_n\times x_n^{(i)})-y^{(i)})^2 \qquad (30)$$

30-formulada ifoda etilgan qiymat funksiyasida bizda $(n+1)$ ta koeffitsiyent qatnashmoqda va bular:

$$A=\left[a_0,\ \ a_1,\ \ \cdots\ \ a_n\right] \qquad (31)$$

Boshqacha aytganda, biz qiymat funksiyasini eng minimal qiymatini topish uchun 31-formulada berilgan koeffitsiyentlarning optimal qiymatini topishimiz kerak. Buning uchun qiymat funksiyasidan har bir koeffitsiyentga nisbatan hosila olamiz. Shunda bizda quyidagi ifoda paydo bo'ladi:

$$\begin{cases} \frac{\delta J}{\delta a_0} = \frac{1}{m}\sum_{i=1}^{m}((a_0 \times x_0^{(i)} + a_1 \times x_1^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_0^{(i)} \\ \frac{\delta J}{\delta a_1} = \frac{1}{m}\sum_{i=1}^{m}((a_0 \times x_0^{(i)} + a_1 \times x_1^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_1^{(i)} \\ \frac{\delta J}{\delta a_2} = \frac{1}{m}\sum_{i=1}^{m}((a_0 \times x_0^{(i)} + a_1 \times x_1^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_2^{(i)} \\ \vdots \\ \frac{\delta J}{\delta a_n} = \frac{1}{m}\sum_{i=1}^{m}((a_0 \times x_0^{(i)} + a_1 \times x_1^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_n^{(i)} \end{cases} \tag{32}$$

$$\Downarrow$$

$$\begin{cases} \frac{\delta J}{\delta a_0} = \frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)}) \times x_0^{(i)} \\ \frac{\delta J}{\delta a_1} = \frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)}) \times x_1^{(i)} \\ \frac{\delta J}{\delta a_2} = \frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)}) \times x_2^{(i)} \\ \vdots \\ \frac{\delta J}{\delta a_n} = \frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)}) \times x_n^{(i)} \end{cases} \tag{33}$$

Koeffitsiyentlarning optimal qiymatini topishda olingan hosilalardan foydalanamiz va ko‘p o‘zgaruvchili chiziqli regressiyaning gradient tushish formulasini yozamiz:

$$\begin{cases} a_0 = a_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_0^{(i)} \\ a_1 = a_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_1^{(i)} \\ a_2 = a_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_2^{(i)} \\ \\ a_n = a_n - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_n^{(i)} \end{cases} \quad (34)$$

34-formuladagi amallarni boshlashdan avval koeffitsiyentlarning qiymatlariga tasodifiy sonlar beriladi (α – o'rganish darajasi). Ushbu formuladagi amallar koeffitsiyentlarning optimal qiymati topilguncha qaytariladi (davr tugaguncha). Buning uchun davrning har qadamida (qadamlar soni juda ko'p bo'lsa, har 10 qadamda yoki har 100 qadamda va h.k.) qiymat funksiyasi hisoblab boriladi. Shuni ta'kidlab o'tish joizki, 34-formulada amallardagi koeffitsiyentlar qiymatlari parallel ravishda yangilanadi. Tasavvur qiling, biz gradient tushishning 10-qadamidamiz. Avvalo, a_0 ni hisoblaymiz:

$$a_0 = a_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((a_0 \times x_0^{(i)} + a_1 \times x_0^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_0^{(i)} \quad (35)$$

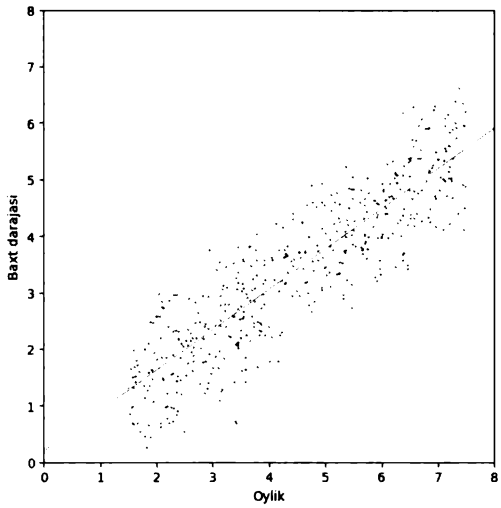
Bizda a_0 koeffitsiyentning yangilangan qiymati mavjud. Endi a_1 koeffitsiyentning yangi qiymatini hisoblaymiz:

$$a_1 = a_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} ((a_0 \times x_0^{(i)} + a_1 \times x_0^{(i)} + a_2 \times x_2^{(i)} + \cdots + a_n \times x_n^{(i)}) - y^{(i)}) \times x_1^{(i)} \quad (36)$$

E'tibor bering, 36-formuladagi a_0 koeffitsiyentning qiymati sifatida 10-qadamda yangilangan qiymatni emas, balki 9-qadamda hisoblangan qiymatni olamiz va shu tariqa 10-qadamda $a_2, a_2, \cdots, a_n$ koeffitsiyentlarning qiymati sifatida 9-qadamda hisoblangan qiymat orqali ifodalaymiz.

3.4 Polinomial regressiya

Biz bir o'zgaruvchili chiziqli regressiya bilan tanishganimizda keltirilgan ma'lumotlar to'plamini to'g'ri chiziq funksiya orqali ifodalasak bo'lar edi (11-rasmda keltirilganidek).



Rasm 11: Chiziqli regressiya

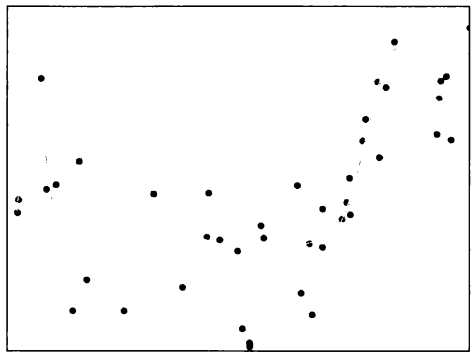
Ammo bizda doim ham shunday "qulay" ma'lumotlar to'plami uchrayvermaydi. Odatda ma'lumotlar to'plami elementlari juda xaotik, ya'ni tartibsiz joylashadi hamda ular o'rtasidagi qonuniyatni topish ham qiyinchilik tug'diradi. Masalan, 12-rasmda ko'rsatilgan ma'lumotlar to'plamini olaylik.

Ushbu ma'lumotlar to'plamini taxminan ifoda etadigan chiziq o'z-o'zidan to'g'ri chiziq bo'la olmaydi. Biz bunday murakkab ma'lumotlar to'plamini ifoda etish uchun koeffitsiyentlarning murakkab kombinatsiyasiga ega funksiyadan foydalanishimiz mumkin. Ya'ni bizda x_1 hamda x_2 o'zgaruvchilar mavjud bo'lsa, biz ularning ko'paytmasidan iborat bo'lgan yangi x_3 o'zgaruvchini kiritishimiz mumkin:

$$x_3 = x_1 \times x_2 \tag{37}$$

Polinomial regressiyani chuqurroq tushinish uchun bir o'zgaruvchili chiziqli regressiya misolida ko'rib chiqaylik. Quyidagi formula bir o'zgaruvchili chiziqli regressiyani ifoda etadi.

$$f(x) = a_0 + a_1 \times x_1 \tag{38}$$



Rasm 12: Xaotik (tartibsiz) ko‘rinishdagi ma’lumotlar to‘plami [11]

38-formula orqali faqatgina to‘g‘ri chiziqni ifodalash mumkin hamda agarda bizda murakkab xususiyatga ega ma’lumotlar to‘plami bo‘lsa (12-rasmda ko‘rsatilgan kabi), unda biz o‘zgaruvchinining kvadrati, kubi yoki 4-darajasi va boshqa darajalaridan foydalanib, murakkab formula tuza olamiz:

$$f(x) = a_0 + a_1 \times x_1 + a_2 \times x_1^2 + a_3 \times x_1^3 \cdots + a_n \times x_1^n \quad (39)$$

39-formulaga e’tibor beradigan bo‘lsak, bizda yangi koeffitsiyentlar paydo bo‘ldi:

$$a_2, a_3, \cdots, a_n$$

Agarda biz x_1 o‘zgaruvchini turli darajaga oshirib, yaratilgan yangi o‘zgaruvchilarimizni quyidagicha ifodalasak

$$x_2 = x_1^2;\ x_3 = x_1^3;\ \cdots\ x_n = x_1^n \quad (40)$$

hamda yangicha ko‘rinish olgan o‘zgaruvchilar orqali 39-formulani yangilasak, quyidagi formulani olamiz:

$$f(x) = a_0 + a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 \cdots + a_n \times x_n \quad (41)$$

41-formula ko‘p o‘zgaruvchili chiziqli regressiya formulasiga mos tushadi va shu kabi yechiladi.

3.5 O‘rganish darajasi

Mashinaviy o‘qitishda biz ikki xil parametrlar bilan ishlaymiz:

- mashina o‘rganadigan parametrlar (machine learnable parameters);
- giperparametrlar (hyperparameters);

Mashina o‘rganadigan parametrlar haqida o‘tgan bo‘limlarda ko‘p suhbat qurdik, ya’ni bular nishon qiymatini topishda foydalaniladigan formuladagi koeffitsiyentlardir:

$$a_0, a_1, \cdots, a_n$$

Ushbu koeffitsiyentlarning optimal qiymatini mashina berilgan ma’lumotlar to‘plami ustida o‘qitish jarayonida (gradient tushish jarayonida) o‘zi topa oladi.

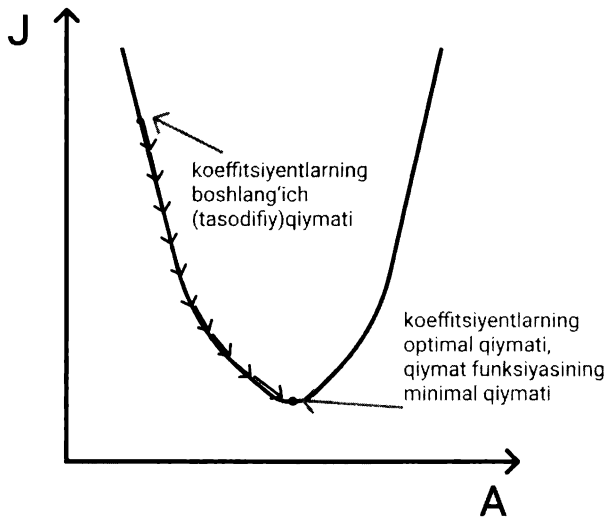
Giperparametrlar esa ma’lumotlar to‘plami injineri yoki dasturchining o‘zi optimal qiymat berishi kerak bo‘lgan parametrlardir. Dasturchi o‘qitish jarayonini optimallashtirishi uchun ushbu giperparametrlarni sozlashi, ya’ni yanada optimalroq qiymat berishi kerak bo‘ladi. Ushbu giperparametrlarga bir qancha parametrlar, xususan, o‘rganish darajasi kiradi. O‘rganish darajasi α orqali belgilanib, gradient tushish jarayonida ushbu parametrni qo‘llagan edik:

$$a_j := a_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_j^{(i)} \tag{42}$$

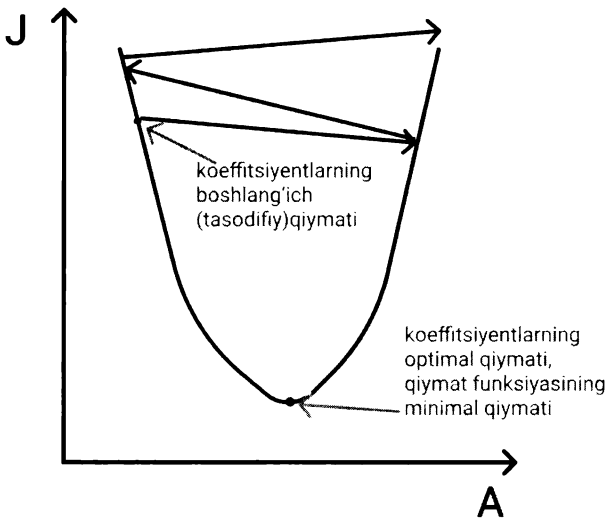
O‘rganish darajasi gradient tushish jarayonida muhim vazifani o‘taydi va u o‘rganishning tezligini belgilab beradi. Ya’ni o‘rganish darajasi qancha katta bo‘lsa, gradient tushish shuncha jadallashadi va koeffitsiyentlar yangilanishi shuncha katta bo‘ladi va aksincha, o‘rganish jarayoni qancha kichik bo‘lsa, o‘rganish jarayoni shuncha sekin bo‘ladi, koeffitsiyentlarning yangi qiymati kichik darajaga ortadi. 13-rasmda o‘rganish darajasi juda kichik bo‘lganda gradient tushishning shartli ko‘rinishi tasvirlangan. Bunda qizil ko‘rsatkichlar orqali gradient tushish qadamlari ko‘rsatilgan.

Albatta, biz tezroq optimal qiymatga erishgimiz keladi va bunda o‘rganish darajasiga kattaroq qiymat berib, gradient tushishni tezlashtirsak, go‘yoki maqsadga yetadigandekmiz. Biroq o‘rganish darajasining katta qiymati, koeffitsiyentlarning optimal qiymatidan sakrab" o‘tib ketish ehtimolini oshiradi. Xususan, 14-rasm qanday qilib katta o‘rganish darajasi gradient tushish qadamlarining optimal qiymatdan o‘tib ketishi tasvirlangan.

O‘rganish darajasiga normal qiymat berilganda, gradient tushish tezligi qoniqarli bo‘ladi hamda koeffitsiyentlarning optimal qiymatini topish ehtimoli



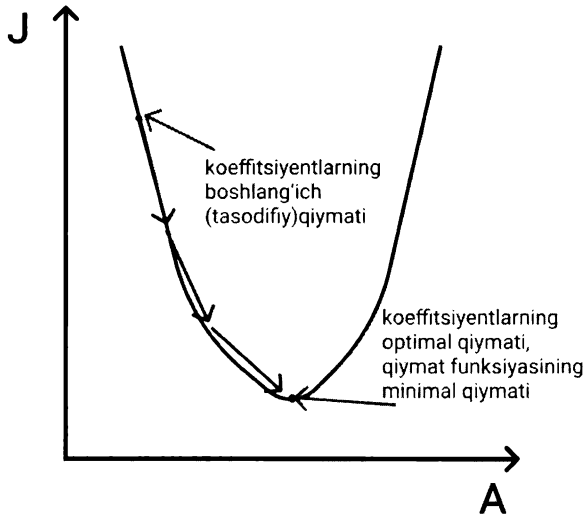
Rasm 13: Juda kichik o'rganish darajasi



Rasm 14: Juda katta o'rganish darajasi

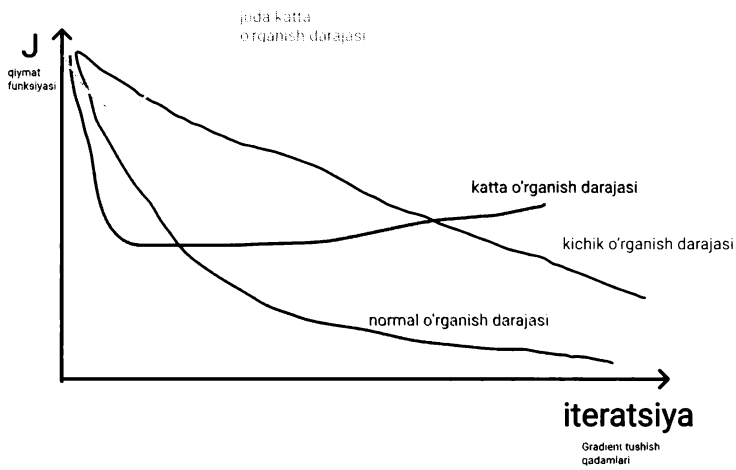
ham oshadi. 15-rasmda o'rganish darajasiga normal qiymat berilganda, gradient tushish shartli ravishda tavsirlangan. Ya'ni qiymat funksiyasini minimallashtiradigan koeffitsiyentlarning optimal qiymati bir necha qadamda topilmoqda.

O'z-o'zidan tabiiy savol tug'iladi. Qanday qilib o'rganish darajasiga optimal



Rasm 15: Normal o‘rganish darajasi

qiymat bera olamiz? Ushbu savolga javob topish uchun qiymat funksiyasi ustida bir qator izlanishlar olib borishga to‘g‘ri keladi. Ya’ni gradient tushishning ma’lum bir qadamida (yoki har qadamida) qiymat funksiyasini hisoblab boramiz va qiymat funksiyasining o‘zgarishiga qarab o‘rganish darajasi haqida xulosa qilishimiz mumkin.



Rasm 16: O‘rganish darajasining qiymat funksiyasiga ta’siri

Xususan, 16-rasm turlicha o'rganish darajalarining qiymat funksiyasiga ta'siri keltirilgan. Keling, ularni birma-bir tahlil qilib chiqaylik. Agar o'rganish darajasiga juda katta qiymat berilsa, u holda qiymat funksiyasi avval minimallashadi, biroq bir necha qadamlar ichida qiymat funksiyasi kattalashib boradi. Ushbu holat 16-rasmda sariq chiziq orqali keltirilgan. O'rganish darajasiga nisbatan katta qiymat berish ham qiymat funksiyasini minimallashtirishga olib kelmaydi va buni 16-rasmdagi yashil chiziq orqali namoyish etilgan qiymat funksiyasidan ko'rishimiz mumkin. Agar o'rganish darajasiga kichik qiymat berilsa, u holda 16-rasmdan ko'rishimiz mumkinki, qiymat funksiyasini minimallashtirish uchun juda ko'p **iteratsiya**, ya'ni gradient tushish qadamlari talab etiladi. Ushbu rasmdagi yashil chiziq esa o'rganish darajasiga normal qiymat berilgandagi holatni aks ettirmoqda. Bunda qiymat funksiyasi har qadamda tushib borayotganini ko'rishimiz mumkin.

Demak, o'rganish darajasining optimal qiymatini topishda har qadamda (yoki ma'lum bir qadamlarda) qiymat funksiyasini hisoblab boramiz. Shu bilan birga qiymat funksiyasining o'zgarishini ham tahlil qilib borishimiz kerak. Ushbu o'zgarish tendensiyasiga qarab o'zgarish darajasini kattalashtirish yoki kichiklashtirish kerakligi haqida ma'lumot olishimiz mumkin. Albatta, o'zgarish darajasining optimal qiymatini tanlashda yordam beradigan usullar ham mavjud va ular haqida keyingi bo'limlarda batafsil ma'lumot beramiz.

3.6 Qisqa takrorlash

Ko'p o'zgaruvchili chiziqli regressiyani amalga oshirishda quyidagi algoritmga amal qilishimiz mumkin:

1. Ko'p chiziqli funksiya regressiya formulasini aniqlab olish;
2. Vektorlangan hisoblashlarni qulaylashtiradigan amallarni bajarish (xususan, $x_0 = 1$ ustunni qo'shish, koeffitsiyentlarni transpozitsiyalash – A^T, chiziqli funksiyaning vektorlangan ko'rinishini belgilab olish);
3. Kiruvchi o'zgaruvchilar qiymatlari orasida tafovut juda katta bo'lsa, sayqallash (normallashtirish yoki standartlashtirish) amalini bajarish;
4. Qiymat funksiyasi formulasini aniqlab olish, vektorlangan ko'rinishini aniqlash;
5. Gradient tushish formulasini aniqlab olish, vektorlangan ko'rinishini aniqlash;

6. Gradient tushish uchun giperparametrlar (o‘rganish darajasi – α, qadamlar soni)ni aniqlab olish;

7. Gradient tushishni belgilangan davr ichida (ma’lum bir qadamlar soni qadar) amalga oshirish. Bunda har qadamda koeffitsiyentlarni yangilab borish va har bir qadamda (yoki ma’lum bir qadamlarda) xatolikni qiymat funksiyasi orqali hisoblab borish.

3.7 Amaliy mashg‘ulot

Ushbu amaliy mashg‘ulotning *github.com* saytidagi repozitoriysiga `https://bit.ly/3qRgBsb` ssilkasi orqali yoki quyidagi QR-kodni skanerlash orqali o‘tishingiz mumkin. Aynan shu amaliy mashg‘ulotning **.ipynb* fayli uchun `practice_session/Multivariate_linear_regression_4_6/` papkasini tekshiring.

Ushbu amaliy mashg‘ulotda *Jupyter Notebook* dasturi orqali Pyton dasturlash tilida Chiziqli regressiya muammosini, xususan, ko‘p o‘zgaruvchili chiziqli regressiya muammosini o‘rganamiz. *Jupyter Notebook* dasturi orqali ma’lum bir katalogga yangi “*.ipynb*" faylni yaratamiz va quyidagi kodlarni navbatma-navbat yozib boramiz.

Avvaliga kerakli kutubxonalarni yuklab olaylik:

- **numpy** – Pytonda massivlar bilan ishlash uchun;
- **pandas** – Pytonda ma’lumotlarni qayta ishlash uchun biz CSV hamda jadval ko‘rinishidagi ma’lumotlar bilan ishlash uchun yuklaymiz;
- **matplotlib** – Pytonda chizish amallarini bajarish uchun biz grafiklarni chizishda foydalanamiz;
- **math** – Pytonda matematik amallar bilan ishlash uchun.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from matplotlib.pyplot import figure
5 import math
```

Ko‘p o‘zgaruvchili chiziqli regressiya uchun **“Kaggle"** ilovasida keltirilgan ma’lumotlar to‘plamlarining biridan (`https://www.kaggle.com/mohansacharya/graduate-admissions` havolada joylashgan) foydalanamiz. Ushbu ma’lumotlar to‘plamini yuklab olamiz hamda ushbu *“.ipynb"* fayl joylashgan katalogga joylaymiz va arxivdan chiqaramiz. So‘ngra *pandas* kutubxonasi yordamida yuklaymiz:

```
1 df = pd.read_csv('Admission_Predict.csv')
```

Ma’lumotlar ba’zasi bilan tanishib chiqaylik. Ya’ni qanday ustunlar bor? Ulardagi ma’lumotlar qanday ko‘rinishda va h.k.

```
1 df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

Biz foydalaniladigan ma’lumotlar to‘plami abituriyentlarning ko‘rsatkichlariga qarab dunyoning yetakchi universitetlariga kirish ehtimoli to‘g‘risidagi ma’lumotlarni o‘z ichiga oladi. Ushbu ma’lumotlar to‘plami bilan kengroq tanishsak:

- **Serial No.** – qatorning tartib raqami (hech qanday ahamiyatga ega emas)
- **GRE Score** – abituriyentning GRE (Graduate Record Examinations) imtihondan olgan balli (eng yuqori qiymat: 340)
- **TOEFL Score** – abituriyentning TOEFL ingliz tili imtihonidan olgan balli (eng yuqori qiymat: 120)

- **University Rating** – abituriyent o'qimoqchi bo'lgan universitetning reytingi (eng yuqori qiymat: 5)
- **SOP** – abituriyent maqsadi ifodalangan arizasi baholangani (eng yuqori qiymat: 5)
- **LOR** – abituriyent tavsiyanomasining kuchi (eng yuqori qiymat: 5)
- **CGPA** – abituriyentning oldingi bilim yurtidagi o'rtacha bahosi (eng yuqori qiymat: 10)
- **Research** – ilmiy tadqiqot tajribasi bor yoki yo'qligi (0 yoki 1)
- **Chance of Admit** – abituriyent tanlagan universitetga kirish ehtimolligi ([0, 1] oraliqda)

Endi ma'lumotlar to'plamining texnik jihatlari bilan tanishsak, ya'ni qanday formatda, qatorlar soni va h.k.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 9 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Serial No.         400 non-null    int64
 1   GRE Score          400 non-null    int64
 2   TOEFL Score        400 non-null    int64
 3   University Rating  400 non-null    int64
 4   SOP                400 non-null    float64
 5   LOR                400 non-null    float64
 6   CGPA               400 non-null    float64
 7   Research           400 non-null    int64
 8   Chance of Admit    400 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 28.2 KB
```

Ko'rib turganimizdek, ma'lumotlar to'plami 400 ta qatordan iborat hamda har bir ustun turli xil formatlarda (*int64* va *float64*). Ushbu ma'lumotlar to'plamini matematik amallarga, xususan, chiziqli regressiya jarayoniga moslaylik. Bunda, avvalo, hisob-kitoblarda qulay bo'lishi uchun *numpy* massiviga o'giramiz, bunda ustunlarning toifasi (formati) sifatida *float64* tanlaymiz.

```
df = np.array(df, dtype = float)
```

Ma'lumotlar to'plamidan o'rganish jarayoni uchun ustunlarni ajratib olamiz va **X** o'zgaruvchisiga massiv sifatida yuklaymiz. Bunda keraksiz ustunlarni, xususan, **Serial No** ustunini tashlab yuboramiz. E'tibor bering, oxirgi ustun **Chance of Admit** ustuni ham yuklanmayapti, sababi bu ustunni alohida o'zgaruvchiga nishonlar massivi sifatida yuklaymiz.

```
X = df[:,1:8]
```

Ma'lumotlar to'plamidan nishonlarni ajratib olamiz va Y o'zgaruvchisiga yuklaymiz.

```
Y = df[:,8:]
Y[:10]
```

```
array([[0.92],
       [0.76],
       [0.72],
       [0.8 ],
       [0.65],
       [0.9 ],
       [0.75],
       [0.68],
       [0.5 ],
       [0.45]])
```

Yuqorida ko'rib turganimizdek, nishonlar massivi $[0,1]$ oraliqdagi sonlardan iborat. Hisob-kitoblarda qulaylik yaratish uchun ularni $[0,100]$ oraliqqa tushiraylik.

```
Y = Y  100
Y[:10]
```

Kirish o'zgaruvchilari sayqallanishi – standartlashtirish

Kirish o'zgaruvchilarini sayqallashtirishda standartlashtirish usulidan foydalanamiz, bunda quyidagi formuladan foydalaniladi:

$$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{S_j}$$

```
array([[92.],
       [76.],
       [72.],
       [80.],
       [65.],
       [90.],
       [75.],
       [68.],
       [50.],
       [45.]])
```

Yuqoridagi formulada o'rta arifmetik qiymat (μ_j) hamda o'rta kvadratik og'ishni (S_j) topishda **numpy** kutubxonasidagi **mean()** va **std()** funksiyalaridan foydalanamiz va ushbu formulani ifoda etuvchi **feature_scaling** funksiyasini yarataylik.

```
1 def feature_scaling(X):
2     avg_array = np.mean(X, 0)
3     std_array = np.std(X, 0)
4     return np.divide(X - avg_array, std_array)
```

```
1 X = feature_scaling(X)
```

Vertikal hisoblarda qulay bo'lishi uchun qiymati 1 dan iborat bo'lgan X_0 vektorni X massiviga qo'shaylik.

```
1 def add_bias(X):
2     A_0 = np.ones((X.shape[0], 1))
3     return np.hstack((A_0, X))
```

```
1 X = add_bias(X)
```

Ma'lumotlarni mashq/test to'plamlariga bo'lish

Bizda mavjud bo'lgan ma'lumotlar to'plamini 2 guruhga bo'laylik va 1-guruhni mashq to'plami, 2-guruhni esa test to'plami

deb ataylik. Bunda biz yaratadigan chiziqli regressiya mashq ma'lumotlar to'plamidan o'rganadi, biz esa test ma'lumotlar to'plamidan uning effektivligini tekshiramiz. Mashq/test ma'lumotlar to'plamiga bo'lishda 80/20 nisbatdan foydalanamiz, ya'ni ma'lumotlar to'plamining 80% mashq va 20% esa test ma'lumotlar to'plamiga bo'lamiz.

```
1 def split(X, Y):
2     rows, _ = X.shape
3     train_rows = round(rows 0.8)
4     test_rows = rows − train_rows
5     return X[0:train_rows,:], X[train_rows:, :],
6         Y[0:train_rows,:], Y[train_rows:, :]
```

```
1 X_train, X_test, Y_train, Y_test = split(X, Y)
```

Koeffitsiyentlarni inisializatsiya qilish

Ko'p o'zgaruvchili Chiziqli regressiyaning formulasini eslaylik:

$$f(x) = a_0 \times x_0 + a_1 \times x_1 + a_2 \times x_2 + \cdots + a_n \times x_n$$

Bizda $n = 7$, chunki ma'lumotlar to'plamida 7 ta ustun mavjud (8-ustun bu qiymati 1 dan iborat bo'lgan a_0 vektor). Demak, bizda a_0 koeffitsiyentni ham hisobga olganda 8 ta koeffitsiyent bo'ladi va ularni vektorlangan ko'rinishda 1 o'lchamli massivda ifodalashimiz mumkin. Koeffitsiyentlarni inisializatsiya qilish (ya'ni boshlang'ich qiymat berish)da *numpy* kutubxonasining tasodifiy sonlarni generatsiya qiluvchi *rand* funksiyasidan foydalanamiz.

```
1 A = np.random.rand(X.shape[1], 1)
```

Chiziqli regressiya funksiyasi

Koeffitsiyentlar uchun o'zgaruvchilarni aniqlab oldik, endi

chiziqli regressiyaning asosiy funksiyasini aniqlasak bo'ladi. Ushbu funksiyani *f_x()* deb ataylik.

```
def f_x(X, A):
    return np.dot(X,A)
```

Qiymat funskiyasi va gradient tushish

Chiziqli regressiyaning qiymat funksiyasini eslaylik:

$$J = \frac{1}{2m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)})^2$$

Ushbu qiymat funksiyasini dasturda ifodalaylik:

```
def cost(X, A, Y, m):
    diff = f_x(X, A) - Y
    diff_2 = np.power(diff, 2)
    sum_ = np.sum(diff_2)
    cost_ = sum_ / (2 m)
    return cost_
```

Gradient tushishda esa biz yuqoridagi qiymat funksiyasini minimallashtiruvchi parametrlarni topamiz. Bunda biz har bir parametr uchun ma'lum bir qator davomida quyidagi ifodani takrorlashimiz kerak:

$$takrorla, k = 1 \ldots epochs \{$$
$$a_j := a_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(f(x^{(i)}) - y^{(i)}) \times x_j^{(i)}$$
$$\}$$

Yuqoridagi siklni amalga oshirishda bir qator giperparametrlarni aniqlab olaylik:

- Qadamlar sonining qiymatini 500 deb belgilaymiz va *epochs* o'zgaruvchisida saqlaymiz;

- O‘rganish darajasini 0.1 deb belgilab, *learning_rate* o‘zgaruvchisida saqlaymiz.

Har bir qadamda mashq va test to‘plami uchun qiymatni hisoblaymiz va mos ravishda *train_costs* hamda *test_costs* massivlarida saqlaymiz.

```
epochs = 100
learning_rate = 0.05
```

Gradient tushishni amalga oshiruvchi funksiyani yaratamiz va *gradient_descent* deb nomlaymiz.

```
def gradient_descent(X_train, Y_train,
    X_test, Y_test, A, learning_rate, epochs):

    m_train = Y_train.shape[0]
    m_test = Y_test.shape[0]
    train_costs = []
    test_costs = []
    for k in range(epochs):
        fx = f_x(X_train, A)
        sum_diff = np.dot(X_train.T, np.subtract(fx, Y_train)) / m_train
        A = A - learning_rate sum_diff
        cost_train = cost(X_train, A, Y_train, m_train)
        cost_test = cost(X_test, A, Y_test, m_test)
        if k % 10 == 0:
            print('epoch: %d, %f% (k, cost_train))
            print('epoch: %d, %f% (k, cost_test))
        train_costs.append(cost_train)
        test_costs.append(cost_test)
    return A, train_costs, test_costs
```

Gradient tushishni amalga oshiraylik. Bunda biz yaratgan *gradient_descent* funksiyasidan foydalanamiz va ushbu funksiya gradient tushishni amalga oshirib, topilgan optimal qiymatni *A* o‘zgaruvchisi va har bir qadamdagi *mashq* hamda *test* to‘plamlar uchun qiymat funksiyani ifoda etuvchi *train_costs*, *test_costs*

massivlarini qaytaradi.

```
A, train_costs, test_costs = gradient_descent(X_train, Y_train,
    X_test, Y_test, A, learning_rate, epochs)
```

```
epoch: 0, 2345.102347
epoch: 0, 2342.961236
epoch: 10, 833.895532
epoch: 10, 982.567122
epoch: 20, 315.547529
epoch: 20, 401.266495
epoch: 30, 128.295151
epoch: 30, 175.199474
epoch: 40, 60.274142
epoch: 40, 86.605985
epoch: 50, 35.381449
epoch: 50, 50.669755
epoch: 60, 26.160050
epoch: 60, 35.368646
epoch: 70, 22.670442
epoch: 70, 28.452940
epoch: 80, 21.298593
epoch: 80, 25.113725
epoch: 90, 20.722400
epoch: 90, 23.392565
```

Gradient tushishda qiymat funksiyasining qay darajada o‘zgarganini vizual ko‘rish uchun grafika *plot_cost* funksiyasini yarataylik. Ushbu funksiya *matplotlib* kutubxonasidan foydalanib grafika chizadi.

```
def plot_cost(train_costs,test_costs, epochs):
    plt.xlabel('Epochs')
    plt.ylabel('Cost')
    plt.plot(epochs, train_costs, 'm', linewidth = "1",
        color='r', label='mashqdagi xatolik')
    plt.plot(epochs, test_costs, 'm', linewidth = "1",
        color='g', label='testdagi xatolik')
    plt.legend(loc="upper right")
```

```
9    plt.show()
```

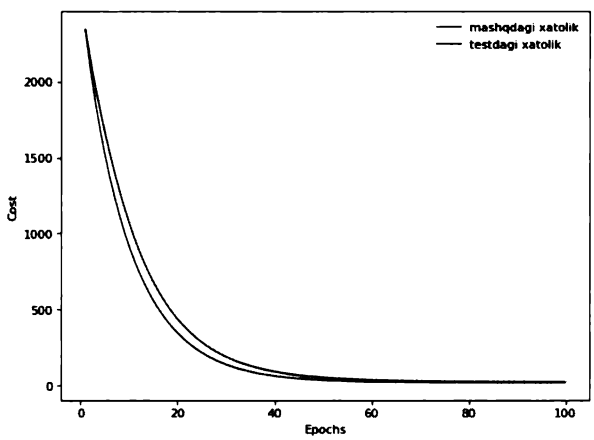
plot_cost funksiyasi parametr sifatida 3 ta massiv o‘zgaruvchisini qabul qiladi. Bular:

- Qiymat funksiyasining natijasi ifodalangan massiv, *train_costs* o‘zgaruvchisi;
- Test to‘plami uchun qiymat funksiyasining natijasi ifodalangan massiv, *test_costs* o‘zgaruvchisi;
- Qadamlarni ifoda etuvchi massiv, *epochs* o‘zgaruvchisi.

```
1 n_epochs = np.arange(1, epochs + 1)
2 n_train_costs = np.array(train_costs)
3 n_test_costs = np.array(test_costs)
```

Qiymat funksiyasini har qadamda o‘zgarishini *plot_cost* funksiyasi orqali grafika tarzda ifodalaylik:

```
1 plot_cost(train_costs, test_costs, n_epochs)
```



Yuqoridagi grafikadan ko‘rib turibmizki, xatolik mashq to‘plamida ham, test to‘plamida ham har qadamda tushib

boryapti. Bundan xulosa qilinsa, biz yaratgan chiziqli regressiya to‘g‘ri ishlayapti.

Endi, amalda test to‘plamdagi birinchi 10 qatorni chiziqli regressiya topgan koeffitsiyentlar orqali bashorat qilamiz va haqiqiy nishonlar bilan tekshiramiz:

```
m_to_predict = 10

predicted = f_x(X_test[:m_to_predict,:], A)

for i in range(m_to_predict):
    print('bashorat: %f, haqiqiy: %f'%(predicted[i], Y_test[i]))
```

```
bashorat: 71.051205, haqiqiy: 75.000000
bashorat: 72.069340, haqiqiy: 73.000000
bashorat: 67.168504, haqiqiy: 72.000000
bashorat: 58.397359, haqiqiy: 62.000000
bashorat: 63.817418, haqiqiy: 67.000000
bashorat: 83.984315, haqiqiy: 81.000000
bashorat: 54.675788, haqiqiy: 63.000000
bashorat: 52.106927, haqiqiy: 69.000000
bashorat: 78.453317, haqiqiy: 80.000000
bashorat: 49.207527, haqiqiy: 43.000000
```

Ko‘rib turibmizki, biroz xatolik bilan bo‘lsa-da, bashorat qilingan nishonlar haqiqiy nishonlar qiymatlariga yaqin. Gradient tushish orqali topilgan optimal koeffitsiyentlar qiymatlari bilan ham tanishishimiz mumkin:

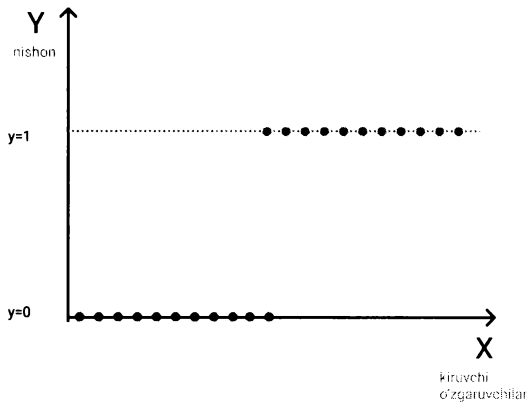
```
A
```

4 Logistik regressiya

Ushbu turdagi regressiya ham mashinaviy o‘qitishning nazoratli o‘qitish turiga kiradi hamda chiziqli regressiyadan farq qiladi. Chiziqli regressiyadan maqsad davomiy nishonni topish bo‘lsa, logistik regressiya **diskret** (ya’ni

```
array([[71.5714992 ],
       [ 2.20428872],
       [ 2.73675912],
       [ 1.15151367],
       [ 0.28778393],
       [ 2.30914182],
       [ 5.15882405],
       [ 0.80342097]])
```

faqat ma'lum bir turdagi) nishonlarni prognoz qiladi. Masalan, bizga kelgan e-mail xabarning spam yoki spam emasligini aniqlashda logistik regressiya algoritmidan foydalaniladi. Chiziqli regressiya hamda logistik regressiya o'rtasidagi farqni yanada chuqurroq anglash uchun 17-rasmga qarang.

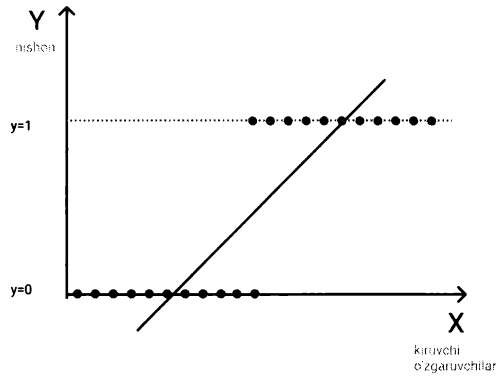


Rasm 17: Diskret qiymatli ma'lumotlar to'plami

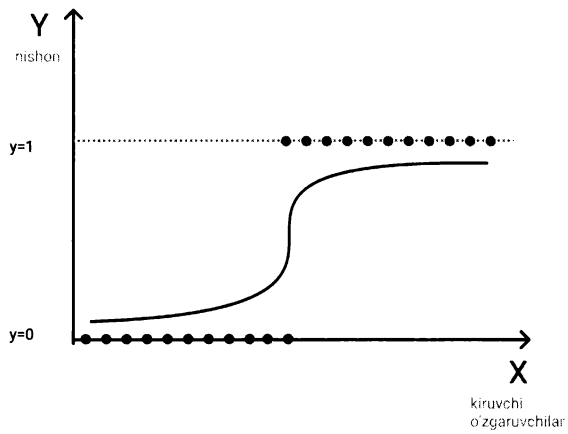
17-rasmda ifoda etilgan ma'lumotlar to'plami Y o'qida (ya'ni nishon qiymati) 0 (ko'k nuqtalar) yoki 1 (yashil nuqtalar) qiymatiga ega bo'ladi. Keling, endi shu ma'lumotlar to'plamiga chiziqli regressiya tatbiq etilganda qanday ko'rinish olishini ko'raylik.

18-rasmdan ko'rinib turibdiki, chiziqli regressiya chizig'i $(0, 1)$ oraliqdan tashqarida ham qiymatlarga ega (Y o'qiga qarang). Ammo ma'lumotlar to'plamidagi nishon qiymatlari (ya'ni Y o'qidagi qiymatlar) faqatgina 0 yoki 1 qiymat qabul qilishi mumkin. Demak, chiziqli regressiya ushbu turdagi vazifalarga eng yaxshi usul emas. Keling, logistik regressiyaga mos tushuvchi funksiyani ko'rib chiqaylik.

Xususan, 19-rasmda sigmoid funksiyaning (qizil egri chiziq) diskret qiymatlarga ega ma'lumotlar to'plamiga tatbiqi ko'rsatilgan va ushbu funksiya $(0, 1)$ oraliqda yotadi va biz prognoz qilgan qiymat ma'lum bir diskret qiymatga



Rasm 18: Diskret qiymatlar uchun chiziqli regressiya tatbiq etilganda



Rasm 19: Diskret qiymatlar uchun logistik regressiya tatbiq etilganda

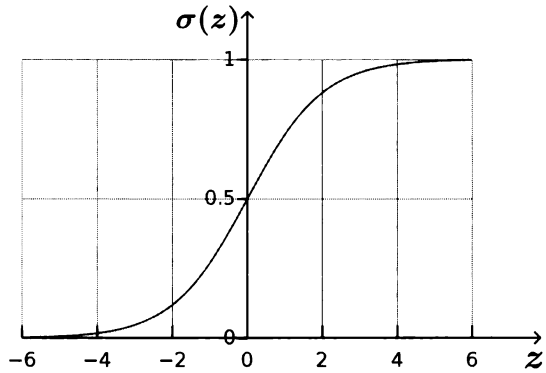
(0 yoki 1) qanchalik yaqinligini ko'rsatib beradi. Shuning uchun klassifikatsiya muammolarida sigmoid funksiya keng qo'llaniladi.

4.1 Sigmoid funksiya

Sigmoid funksiya **logistik funksiya** deb ham ataladi va uning asosiy vazifasi berilgan sonning $(0, 1)$ oraliqdagi qiymatini aniqlashdir. Mashinaviy o'qitishda esa sigmoid funksiyadan biz prognoz qilgan qiymatni ehtimoliysini aniqlashda foydalanamiz.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{43}$$

43-formulada sigmoid funksiyasining formulasi berilgan. Unda z – sigmoid funksiyaga kiruvchi o'zgaruvchi bo'lsa, $\sigma(z)$ sigmoid funksiyadir. Sigmoid funksiyanining formulasini chuqurroq tushunish uchun ushbu funksiyani koordinatalar o'qida tasvirlaylik (20-rasmga qarang).



Rasm 20: Sigmoid funksiya grafigi

20-rasmdan ko'rinib turibdiki, sigmoid funksiyaning minimal qiymati 0 bo'lsa, maksimal qiymati 1 dir. Ya'ni ushbu funksiya $(0, 1)$ oraliqda yotadi. Ushbu funksiyaga qancha katta musbat qiymat berilsa (ya'ni z ning qiymat qancha katta musbat bo'lsa), sigmoid funksiya shuncha 1 ga yaqinlashadi va aksincha, sigmoid funksiyaga qancha katta manfiy qiymat berilsa (ya'ni z ning qiymat qancha katta manfiy bo'lsa), sigmoid funksiya 0 ga yaqinlashadi.

Endi sigmoid funksiyaning logistik regressiyaga tatbiqini ko'rib chiqaylik. Buning uchun esa chiziqli regressiyaning formulasini (vektorlangan ko'rinishini) eslaylik (44-formula).

$$f(x) = XA^T \tag{44}$$

44-formulada ikki matritsa ko'paytmasi (X va A^T) bizga davomli, ya'ni chiziqli qiymatlarni beradi. Undan diskret qiymatlarni olish uchun esa sigmoid funksiyasini tatbiq qilish kerak.

$$\sigma(Z) = \sigma(XA^T) \tag{45}$$

Sigmoid funksiyaning logistik regressiyadagi to'liq formulasini quyidagicha yozishimiz mumkin:

$$f(X) = \frac{1}{1 + e^{-(XA^T)}} \tag{46}$$

Sigmoid funksiyasining qisqaroq ko'rinishini quyidagicha ifodalaymiz:

$$f(Z) = \frac{1}{1 + e^{-Z}} \tag{47}$$

47-formulada $Z = XA^T$ deb hisoblaymiz.

4.2 Qaror qilish chegarasi

Sigmoid funksiya, ya'ni logistik funksiya $(0, 1)$ oraliqda yotishini ko'rib chiqdik. Klassifikatsiya muammosida esa logistik funksiya qiymatini prognoz ehtimolligi deb olamiz. Tasavvur qiling, biz berilgan rasmda kuchuk tasvirlanganini topishimiz kerak, ya'ni bizda 2 xil nishon bor: 1)kuchuk bor; 2) kuchuk yo'q. Ularni mos ravishda 1 – kuchuk mavjud, 0 – kuchuk mavjud emas, deb belgilab olaylik. Shundan so'ng qaror qilish chegarasini belgilab olamiz. Ya'ni 0 hamda 1 orasidagi sonni tanlaymiz. Misol uchun 0.5 deb. Bunda logistik funksiyaning 0.5 dan katta qiymatini kuchuk rasmi mavjud deb, aksincha, 0.5 dan kichik bo'lsa, kuchuk rasmi yo'q sifatida tan olishimiz mumkin. Aytaylik, berilgan rasm ustida logistik funksiyani ishga tushirdik va 0.7 qiymatni oldik. Bunda ushbu rasmning kuchuk bo'lish ehtimoli 70% ni tashkil etadi. Agarda logistik funksiya 0.2 qiymat qaytarsa, biz ushbu rasmda kuchuk yo'q, deb aytishimiz mumkin.

4.3 Qiymat funksiyasi

O'tgan bo'limlarda ko'rib chiqqanimizdek, qiymat funksiyasining asosiy maqsadi – bu prognoz qilingan qiymat va haqiqiy qiymat o'rtasidagi tafovut (xatolik)ni aniqlash.

$$J = \frac{1}{2m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)})^2 \tag{48}$$

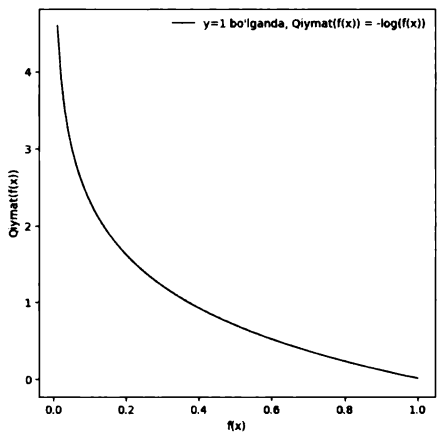
48-formula chiziqli regressiyaning qiymat funksiyasi, ammo biz ushbu funksiyani logistik regressiya uchun qo'llay olmaymiz. Logistik regressiya uchun qiymat funksiyasining umumiy ko'rinishini esa quyidagicha ifodalaymiz:

$$J = \frac{1}{m} \sum_{i=1}^{m} (Qiymat(f(x)^{(i)}, y^{(i)})) \tag{49}$$

Logistik regressiyada qiymat funksiyasining aniq ko'rinishi nishonning haqiqiy qiymatiga bog'liq. Ya'ni nishonga qarab qiymat funksiyasining ko'rinishi o'zgaradi.

$$\begin{cases} Qiymat(f(x), y) = -\log(f(x)), \text{ agar } y = 1 \\ Qiymat(f(x), y) = -\log(1 - f(x)), \text{ agar } y = 0 \end{cases} \tag{50}$$

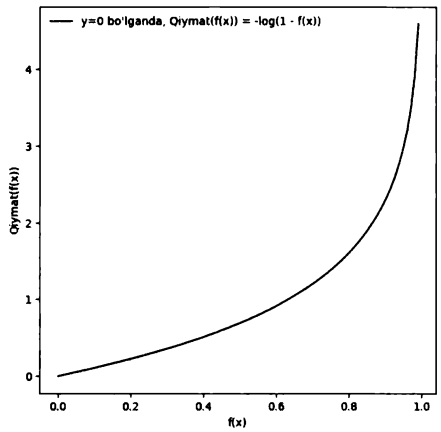
Xususan, 50-formulada nishonning qiymatiga ko'ra 2-xil qiymat funksiyasiga ega bo'lamiz. Ushbu qiymat funksiyasini chuqurroq tushunish uchun ularni koordinatalar o'qida ifodalaylik.



Rasm 21: $y = 1$ bo'lganda qiymat funksiya grafigi

21-rasmda $y = 1$ bo'lganda qiymat funksiyasi ifodalangan. Ko'rinib turibdiki, $f(x)$ ning qiymati 1 ga yaqinlashgan sari qiymat funksiyasi minimallashyapti. Chunki biz nishonning asl qiymatiga ($y = 1$) yaqin qiymatni prognoz qilganimiz sari xatolik kamaymoqda va aksincha, $f(x)$ ning qiymati 0 ga yaqinlashgan sari qiymat funksiyasi ham o'sib boryapti. Bunda nishonning asl qiymati ($y = 1$)ni topa olmasak, $f(x)$ning xatoligini yuqori baholashimiz kerak.

22-rasmda $y = 0$ bo'lganda qiymat funksiyasi ifodalangan. Funksiya grafigidan ko'rishimiz mumkinki, $f(x)$ ning qiymati 0 ga yaqinlashgan sari qiymat funksiyasi minimallashyapti. Chunki biz nishonning asl qiymatiga ($y = 0$) yaqin qiymatni prognoz qilganimiz sari xatolik kamaymoqda va aksincha, $f(x)$ ning qiymati 1 ga yaqinlashgan sari qiymat funksiyasi ham o'sib boryapti. Bunda nishonning asl qiymatini ($y = 0$) topa olmasak, $f(x)$ning xatoligini yuqori baholashimiz kerak.



Rasm 22: $y = 0$ bo'lganda qiymat funksiya grafigi

50-formulada ifodalangan logistik regressiyaning formulalarini birlashtirib yozish mumkin. Bunda logistik regressiyaning qiymat funksiyasi quyidagicha ko'rinish oladi:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f(x^{(i)})) + (1-y^{(i)})\log(1-f(x^{(i)}))\right] \tag{51}$$

4.4 Gradient tushish

Logistik regressiyada ham qiymat funksiyasini minimallashtirishda gradient tushish usulidan foydalanamiz. Ya'ni gradient tushish orqali qiymat funksiyasining minimallashtiruvchi parametrlar topiladi. Gradient tushishning har bir parametri uchun umumiy formulasini eslayslik.

$$\begin{aligned} &takrorlash, j = 1 \ldots n \ \{ \\ &a_j := a_j - \alpha\frac{\delta J}{\delta a_j} \\ &\} \end{aligned} \tag{52}$$

52-formulada keltirilganlarni logistik regressiyaga tatbiq etsak, quyidagi formulani olamiz:

$$takrorlash, j = 1 \ldots n \ \{$$
$$a_j := a_j - \alpha \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) x_j^{(i)} \tag{53}$$
$$\}$$

53-formulada ko'rinib turibdiki, biz gradient tushishni har bir parametr uchun takrorlashimiz kerak. Ya'ni bizda n ta parametr bo'lsa, 1-parametrdan boshlab (a_1) n-parametrgacha (a_n) parallel ravishda takrorlashimiz kerak.

Bunda $f(x)$ – sigmoid funksiyaligini esdan chiqarmasligimiz kerak.

4.5 Model aniqligi

	Prognoz qilingan klass		
Haqiqiy klass		Klass = pozitiv	Klass = negativ
	Klass = pozitiv	Haqiqiy pozitiv	Yolg'on negativ
	Klass = negativ	Yolg'on pozitiv	Haqiqiy negativ

Jadval 4: Haqiqiy va yolg'on pozitiv-negativ klasslar

Mashinaviy o'qitishning klassifikatsiyalash turida modellarning qanchalik to'g'ri hisoblashini tekshirishning bir necha o'lchovlari bor. Eng keng tarqalgan o'lchovlarga **F1-Score** va **Accuracy** kiradi. Ushbu o'lchovlarni hisoblashda ham bir qancha parametrlar ishtirok etadi va ular quyidagilardir [2]:

- **Haqiqiy pozitiv – TP (True Positives)** – to'g'ri prognoz qilingan pozitiv qiymat. Masalan, ma'lumotlar to'plamida ma'lum bir nishon 1 (yoki *ha*, umuman olganda, pozitiv nishon) deya belgilangan bo'lsa va model ham 1 deb prognoz qilsa, ushbu prognoz haqiqiy pozitiv deyiladi.

- **Haqiqiy negativ – TN (True Negatives)** – to'g'ri prognoz qilingan negativ qiymat. Masalan, ma'lumotlar to'plamida ma'lum bir nishon 0 (yoki *yo'q*, umuman olganda, negativ nishon) deya belgilangan bo'lsa va model ham 0 deb prognoz qilsa, ushbu prognoz haqiqiy negativ deyiladi.

- **Yolg'on pozitiv – FP (False Positives)** – ma'lumotlar to'plamidagi haqiqiy qiymati negativ bo'lgan, ammo model pozitiv deya noto'g'ri prognoz

qilingan qiymat. Masalan, ma'lumotlar to'plamida ma'lum bir nishon 0 (yoki *yo'q*, umuman olganda, negativ nishon) deya belgilangan bo'lsa-yu, biroq model 1 deb noto'g'ri prognoz qilsa, ushbu prognoz yolg'on pozitiv deyiladi.

- **Yolg'on negativ – FN (False Negatives)** – ma'lumotlar to'plamidagi haqiqiy qiymati pozitiv bo'lgan, ammo model negativ deya noto'g'ri prognoz qilgan qiymat. Masalan, ma'lumotlar to'plamida ma'lum bir nishon 1 (yoki *ha*, umuman olganda, pozitiv nishon) deya belgilangan bo'lsa-yu, biroq model 0 deb noto'g'ri prognoz qilsa, ushbu prognoz yolg'on negativ deyiladi.

- **To'plamdagi elementlar soni – N** – modelning aniqligi hisoblanayotgan to'plamdagi elementlar soni. Bunda $N = TP + TN + FP + FN$

Kerakli parametrlarni aniqlashtirib olganimizdan keyin, *F1-Score* hamda *Accuracy*larni aniqlashga o'tishimiz mumkin.

Accuracy – modelning aniqligini o'lchaydigan eng sodda usul bo'lib, to'g'ri topilgan nishonlarning sonini (TP + TN) nishonlarning umumiy soniga (N) bo'lishdan kelib chiqadi. Ushbu o'lchov ma'lumotlar to'plamidagi pozitiv va negativ nishonlarning qiymatlari teng bo'lganda o'rinlidir. Masalan, ma'lumotlar to'plami 100 ta qatordan iborat hamda ularning 50 tasi 1 ("ha" yoki har qanday pozitiv qiymat) va 50 tasi 0 ("yo'q" yoki har qanday negativ qiymat) deya belgilangan bo'lsa, **accuracy** o'lchovi o'rinlidir.

$$Accuracy = \frac{TP + TN}{N} \tag{54}$$

Agarda ma'lumotlar to'plamida pozitiv hamda negativ qiymatlar nisbati teng bo'lmasa, u holda accuracy o'lchovidan ko'ra aniqroq F1-Score o'lchovidan foydalaniladi. F1-Score'ni o'lchashda *Precision* hamda *Recall* parametrlarini aniqlab olishimiz kerak.

Precision – to'g'ri prognoz qilingan pozitiv qiymatlarning (TP) pozitiv deya prognoz qilingan qiymatlarning umumiy soniga (TP + FP) nisbatiga aytiladi. Masalan, mashinaviy o'qitish modeli 50 ta pozitiv nishonni prognoz qildi va ulardan 25 tasi to'g'ri prognoz qilingan edi. Demak, $precision = 0.5$ ga teng.

$$Precision = \frac{TP}{TP + FP} \tag{55}$$

Recall (Sensitivity) – to'g'ri prognoz qilingan pozitiv qiymatlarning (TP) ma'lumotlar to'plamidagi barcha haqiqiy pozitiv nishonlarga (TP + FN) nisbatidir. Masalan, ma'lumotlar to'plamida pozitiv deya beliglangan 50 ta

nishon bor. Mashinaviy o'qitish modeli esa shulardan 25 tasini to'g'ri prognoz qila oldi, xolos. Bunda $Recall = 0.5$ ga teng bo'ladi.

$$Recall = \frac{TP}{TP + FN} \tag{56}$$

F1-Score – *Precision* hamda *Recall*'lar orqali ifodalanadigan o'lchov bo'lib, uning formulasi quyidagichadir:

$$F1 - Score = \frac{2 * (Recall * Precision)}{Recall + Precision} \tag{57}$$

Ko'rib turibmizki, *F1-Score* modelning aniqligini hisoblashda, *accuracy*'dan farqli o'laroq, yolg'on pozitiv hamda yolg'on negativni ham hisobga oladi va modelning aniqligi haqida yanada xatosiz ma'lumot beradi. Ayniqsa, ma'lumotlar to'plamida pozitiv va negativ nishonlar notekis taqsimlanganda *F1-Score* eng aniq o'lchovlardan biri hisoblanadi.

4.6 Amaliy mashg'ulot

Ushbu amaliy mashg'ulotning *github.com* saytidagi repozitoriysiga `https://bit.ly/3qRgBsb` ssilkasi orqali yoki quyidagi QR-kodni skaner qilish orqali o'tishingiz mumkin. Aynan shu amaliy mashg'ulotning **.ipynb* fayli uchun `practice_session/Logistic_regression_5_6/` papkasini tekshiring.

Ushbu amaliy mashg'ulotda *Jupyter Notebook* dasturi orqali Pyton dasturlash tilida Logistik regressiya muammosini o'rganamiz. *Jupyter Notebook* dasturi orqali ma'lum bir katalogga yangi "*.ipynb*" faylni yaratamiz va quyidagi kodlarni navbatma-navbat yozib boramiz.

Avvaliga kerakli kutubxonalarni yuklab olaylik:

- **numpy** – Pytonda massivlar bilan ishlash uchun;
- **pandas** – Pytonda ma'lumotlarni qayta ishlash uchun biz CSV hamda jadval ko'rinishidagi ma'lumotlar bilan ishlash uchun yuklaymiz;

- **matplotlib** – Pytonda chizish amallarini bajarish uchun biz grafiklarni chizishda foydalanamiz;
- **math** – Pytonda matematik amallar bilan ishlash uchun.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from matplotlib.pyplot import figure
5 import math
```

Logistik regressiya uchun **"Kaggle"** ilovasida keltirilgan ma'lumotlar to'plamlaridan biridan (`https://www.kaggle.com/uciml/pima-indians-diabetes-database` havolada joylashgan) foydalanamiz. Ushbu ma'lumotlar to'plamini yuklab olamiz hamda ushbu *".ipynb"* fayl joylashgan katalogga joylaymiz va arxivdan chiqaramiz. So'ngra *pandas* kutubxonasi yordamida yuklaymiz:

```
1 df = pd.read_csv('diabetes.csv')
```

Ma'lumotlar ba'zasi bilan tanishib chiqaylik, ya'ni qanday ustunlar bor, ulardagi ma'lumotlar qanday ko'rinishda va h.k.

```
1 df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Ushbu ma'lumotlar to'plami AQSHning Diabet, oshqozon-ichak va buyrak kasalliklari milliy instituti (National Institute of Diabetes and Digestive and Kidney Diseases) ma'lumotlariga asoslangan bo'lib, AQSh tub aholisining 21 yoshdan katta bo'lgan ayollardagi turli analizlar, xususan, diabet kasalligi bor yoki yo'qligini o'zida ifodalaydi. Ma'lumotlar to'plamida 9 ta ustun mavjud bo'lib, ularni quyidagicha tavsif etish mumkin:

- **Pregnancies** – ayol necha marta homilador bo'lgani;

- **Glucose** – glyukoza o'lchovi;
- **BloodPressure** – qon bosimi;
- **SkinThickness** – teri qalinligi;
- **Insulin** – insulin o'lchovi;
- **BMI** – bo'y va vazn o'lchovi (kg/m^2)
- **Diabetes Pedigree Function** – diabet kasalligining nasldagi koeffitsiyenti;
- **Age** – yosh
- **Outcome** – diabet kasalligi mavjud (1) yoki yo'q (0).

Endi ma'lumotlar to'plamining texnik jihatlari bilan tanishsak, ya'ni qanday formatdaligi, qatorlar soni va h.k.

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Ko'rib turganimizdek, ma'lumotlar to'plami 768 ta qatordan iborat hamda har bir ustun turli xil formatlarda (*int64* va *float64*). Ushbu ma'lumotlar to'plamini matematik amallarga, xususan, logistik regressiya jarayoniga moslaylik. Bunda, avvalo, hisob-kitoblarda qulay bo'lishi uchun *numpy* massiviga o'giramiz. Bunda ustunlar formati sifatida *float64* tanlaymiz.

```
1 df = np.array(df, dtype = float)
```

Ma'lumotlar to'plamidan o'rganish jarayoni uchun ustunlarni ajratib olamiz va X o'zgaruvchisiga massiv sifatida yuklaymiz. E'tibor bering, oxirgi ustun *Outcome* ustuni mashq to'plamiga yuklanmayapti. Sababi bu ustunni alohida o'zgaruvchiga nishonlar massivi sifatida yuklaymiz.

```
1 X = df[:, :8]
```

Ma'lumotlar to'plamidan nishonlarni ajratib olamiz va Y o'zgaruvchisiga yuklaymiz.

```
1 Y = df[:,8:]
```

Kirish o'zgaruvchilarining sayqallanishi – standartlashtirish

Kirish o'zgaruvchilarini sayqallashtirishda standartlashtirish usulidan foydalanaylik. Bunda quyidagi formuladan foydalanamiz:

$x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{S_j}$

Yuqoridagi formuladagi o'rta arifmetik qiymat (μ_j) hamda o'rta kvadratik og'ishni (S_j) topishda **numpy** kutubxonasidagi **mean()** va **std()** funksiyalaridan foydalanamiz. Endi ushbu formulani ifoda etuvchi **feature_scaling** funksiyasini yarataylik:

```
1 def feature_scaling(X):
2     avg_array = np.mean(X, 0)
3     std_array = np.std(X, 0)
4     return np.divide(X - avg_array, std_array)
```

feature_scaling funksiyasi yordamida kiruvchi o'zgaruvchilarni sayqallashtiramiz:

```
1 X = feature_scaling(X)
```

Vertikal hisoblarda qulay bo'lish uchun qiymati 1 dan iborat bo'lgan X_0 vektorni X massiviga qo'shamiz:

```
def add_bias(X):
    B = np.ones((X.shape[0], 1))
    return np.hstack((B, X))

X = add_bias(X)
```

Ma'lumotlarni mashq/test to'plamlariga bo'lish

Bizda mavjud bo'lgan ma'lumotlar to'plamini 2 guruhga bo'laylik va 1-guruhni mashq to'plami, 2-guruhni esa test to'plami deb ataylik. Bunda biz yaratadigan logistik regressiya modeli mashq ma'lumotlar to'plamidan o'rganadi va test ma'lumotlar to'plamidan uning effektivligini tekshiramiz. Mashq/test ma'lumotlar to'plamiga bo'lishda 80/20 nisbatdan foydalanamiz. Ya'ni ma'lumotlar to'plamining 80% mashq va 20% test ma'lumotlar to'plamiga bo'lamiz.

```
def split(X, Y):
    rows, _ = X.shape
    train_rows = round(rows 0.8)
    test_rows = rows − train_rows
    return X[0:train_rows,:], X[train_rows:, :],
        Y[0:train_rows,:], Y[train_rows:, :]

X_train, X_test, Y_train, Y_test = split(X, Y)
```

Koeffitsiyentlarni inisializatsiya qilish

Logistik regressiyaning formulasini eslaylik: $f(Z) = \frac{1}{1+e^{-Z}}$

Bunda **Z:** $Z = a_0 \times x_0 + a_1 \times x_1 + a_2 \times x_2 + \cdots + a_n \times x_n$

Demak, bizda $n = 8$. Chunki bizda 8 ta ustun mavjud. Lekin biz 9 ta parametrni inisializatsiya qilamiz (a_0 ni ham hisobga olganda).

**Eslatib o'tamiz, Z funksiyani vektorlangan ko'rinishi:* $Z = X * A^T$

```
A = np.random.rand(X.shape[1], 1)
```

Logistik regressiya funksiyasi

Biz koeffitsiyentlar uchun o'zgaruvchilarni aniqlab oldik. Endi funksiyani aniqlasak bo'ladi. Ushbu funksiyani *logistic_regression_function()* deb ataymiz va u quyidagi formula orqali ifodalanadi:

$$f(Z) = \frac{1}{1+e^{-Z}}$$

Bunda Z:

$$Z = a_0 \times x_0 + a_1 \times x_1 + a_2 \times x_2 + \cdots + a_n \times x_n$$

```
1 def logistic_regression_function(X, A):
2     Z = np.dot(X,A)
3     return 1 / (1 + np.exp(−Z))
```

Qiymat funskiyasi va gradient tushish

Chiziqli regressiyaning qiymat funksiyasini eslaylik:

$$J = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log(f(x^{(i)})) + (1-y^{(i)})\log(1-f(x^{(i)}))\right]$$

Ushbu qiymat funksiyasini dasturda ifodalaylik:

```
1 def cost(X, A, Y):
2     m = Y.shape[0]
3     F_x = logistic_regression_function(X, A)
4     cost_ = − np.sum(Y np.log(F_x) + (1−Y) np.log(1 − F_x)) / m
5     return cost_
```

Gradient tushishda esa biz yuqoridagi qiymat funksiyasini minimallashtiruvchi parametrlarni topamiz. Bunda biz har bir parametr uchun ma'lum bir qator davomida quyidagi ifodani takrorlashimiz kerak:

$$takrorla, k = 1 \ldots epochs \{$$

$$a_j := a_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)}) \times x_j^{(i)}$$

$$\}$$

Yuqoridagi siklni amalga oshirishda bir qator giperparametrlarni aniqlab olaylik:

- Qadamlar sonining qiymatini 500 deb belgilaymiz va *epochs* o'zgaruvchisida saqlaymiz;
- O'rganish darajasini 0.1 deb belgilaylik va *learning_rate* o'zgaruvchisida saqlaymiz.

Har bir qadamda mashq va test to'plami uchun qiymatni hisoblaymiz va mos ravishda *train_costs* va *test_costs* massivlarida saqlaymiz.

```
epochs = 500
learning_rate = 0.1
```

Gradient tushishni amalga oshiruvchi funksiyani yaratamiz va *gradient_descent* deb nomlaymiz.

```
def gradient_descent(X_train, Y_train,
    X_test, Y_test, A, learning_rate, epochs):
    train_costs = []
    test_costs = []
    m_train = Y_train.shape[0]
    m_test = Y_test.shape[0]
    for k in range(epochs):
        F_x = logistic_regression_function(X_train, A)
        A = A - learning_rate np.dot(X_train.T, (F_x - Y_train)) / m_train
        cost_train_ = cost(X_train, A, Y_train)
```

```
        cost_test_ = cost(X_test, A, Y_test)
        train_costs.append(cost_train_)
        test_costs.append(cost_test_)
        if k % 20 == 0:
            print('epoch: %d, %f' % (k, cost_train_))
    return A, train_costs, test_costs
```

Gradient tushishni amalga oshiraylik, bunda biz yaratgan *gradient_descent* funksiyasidan foydalanamiz. Ushbu funksiya gradient tushishni amalga oshirib, topilgan optimal qiymatni *A* o'zgaruvchisi va har bir qadamdagi *mashq* va *test* to'plamlar uchun qiymat funksiyani ifoda etuvchi *train_costs*, *test_costs* massivlarini qaytaradi.

```
A, train_costs, test_costs = gradient_descent(X_train, Y_train,
    X_test, Y_test, A, learning_rate, epochs)
```

```
epoch: 0, 0.761052
epoch: 20, 0.622478
epoch: 40, 0.544009
epoch: 60, 0.505538
epoch: 80, 0.487861
epoch: 100, 0.479525
epoch: 120, 0.475272
epoch: 140, 0.472899
epoch: 160, 0.471471
epoch: 180, 0.470563
epoch: 200, 0.469962
epoch: 220, 0.469553
epoch: 240, 0.469269
epoch: 260, 0.469068
epoch: 280, 0.468926
epoch: 300, 0.468823
epoch: 320, 0.468749
epoch: 340, 0.468694
epoch: 360, 0.468655
epoch: 380, 0.468625
epoch: 400, 0.468604
epoch: 420, 0.468588
epoch: 440, 0.468576
epoch: 460, 0.468568
epoch: 480, 0.468561
```

Gradient tushishda qiymat funksiyasining qay darajada o'zgarganini vizual ko'rish uchun grafik *plot_cost* funksiyasini

yaratamiz. Ushbu funksiya *matplotlib* kutubxonasidan foydalanib grafika chizadi.

```
def plot_cost(train_costs,test_costs, epochs):
    plt.xlabel('Epochs')
    plt.ylabel('Cost')
    plt.plot(epochs, train_costs, 'm', linewidth = "1",
        color='r', label='mashqdagi xatolik')
    plt.plot(epochs, test_costs, 'm', linewidth = "1",
        color='g', label='testdagi xatolik')
    plt.legend(loc="upper right")
    plt.show()
```

plot_cost funksiyasi parametr sifatida 3 ta massiv o'zgaruvchisini qabul qiladi va bular:

– Mashq to'plami uchun qiymat funksiyasining natijasi ifodalangan massiv, *train_costs* o'zgaruvchisi;

– Test to'plami uchun qiymat funksiyasining natijasi ifodalangan massiv, *test_costs* o'zgaruvchisi;

– Qadamlarni ifoda etuvchi massiv, *epochs* o'zgaruvchisi.

```
n_epochs = np.arange(1, epochs + 1)
```

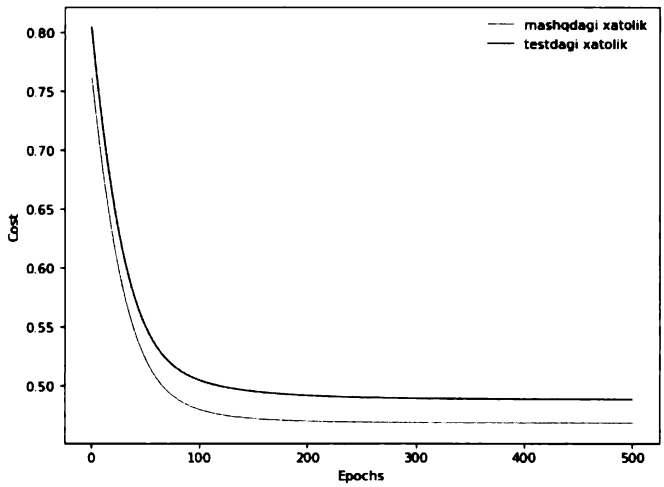
Qiymat funksiyasining har qadamda o'zgarishini *plot_cost* funksiyasi orqali grafik tarzda ifodalaylik.

```
figure(figsize=(8, 6), dpi=80)

plot_cost(train_costs, test_costs, n_epochs)
```

Yuqoridagi grafikadan ko'rib turibmizki, xatolik mashq to'plamida ham, test to'plamida ham har qadamda tushib boryapti. Bundan xulosa qilishimiz mumkinki, biz yaratgan logistik regressiya to'g'ri ishlayapti.

Qaror qabul qilish chegarasi (Decision boundary)



Biz yaratgan logistik regressiyaning natijasi $[0,1]$ oraliqda yotadi. Ya'ni butun bo'lmagan sonlarni chiqaradi. Bizning nishonlarimiz esa qiymatlari 0 yoki 1 bo'lgan butun sonlardan iborat. Xususan, test to'plami uchun logistik regressiyani hisoblaylik va dastlabki 10 ta elementni chop etaylik:

```
1 test_predictions = logistic_regression_function(X_test, A)
2 test_predictions[:10,:]
```

```
array([[0.74926893],
       [0.08762216],
       [0.16827159],
       [0.01457882],
       [0.50125461],
       [0.2747579 ],
       [0.20935115],
       [0.16839148],
       [0.96523638],
       [0.20502793]])
```

Yuqoridagi sonlarni 0 va 1 butun sonlarga aylantirish uchun qaror qabul qilish chegarasini amalga oshiruvchi funksiyani yarataylik. Bunda chegara sifatida 0.5 ni tanlaymiz. Ya'ni 0.5 dan katta sonlarni 1, 0.5 ga teng va undan kichiklarini 0 sifatida qabul qilamiz. Biz yaratgan funksiya massivlar bilan ishlay

olishi uchun ushbu funksiyaga *numpy* kutubxonasining *vectorize* funksiyasi yordamida ishlov beramiz.

```
def db_function(prediction):
    if prediction > 0.5:
        return 1
    else:
        return 0
db_function = np.vectorize(db_function)
```

Biz yaratgan *db_function* funksiyasini amalga oshiraylik va *test_predictions* massividagi sonlarni butun sonlarga o'girib olaylik:

```
test_predictions = db_function(test_predictions)
test_predictions[:10,:]
```

```
array([[1],
       [0],
       [0],
       [0],
       [1],
       [0],
       [0],
       [0],
       [1],
       [0]])
```

Model aniqligi (Model accuracy)

Biz yaratgan modelning aniqligini baholash uchun *Accuracy* va *F1-Score* o'lchovlarini hisoblaylik. Ushbu o'lchovlarni hisoblashdan oldin bir nechta yordamchi o'lchovlarni aniqlab olaylik. Xususan:

- **Haqiqiy pozitiv** – TP (True Positives)
- **Haqiqiy negativ** – TN (True Negatives)

- **Yolg‘on pozitiv** – FP (False Positives)
- **Yolg‘on negativ** – FN (False Negatives)
- **To‘plamdagi elementlar soni** – N

Yuqoridagi yordamchi o‘lchovlarni hisoblovchi funksiyani yaratamiz:

```
def calculate_helper_vars(predicted, actual):
    n = actual.shape[0]
    predicted = predicted.reshape(-1, actual.shape[1])
    tp = np.count_nonzero((predicted == actual) & (predicted == 1))
    tn = np.count_nonzero((predicted == actual) & (predicted == 0))
    fp = np.count_nonzero((predicted != actual) & (predicted == 1))
    fn = np.count_nonzero((predicted != actual) & (predicted == 0))
    return tp, tn, fp, fn, n
```

calculate_helper_vars funksiyasi yordamida yordamchi o‘lchovlarni hisoblaylik:

```
tp, tn, fp, fn, n = calculate_helper_vars(test_predictions, Y_test)
```

Yordamchi o‘lchovlar yordamida modelni baholovchi quyidagi o‘lchovlarni hisoblaylik:

- *Accuracy* – model aniqligi haqidagi eng sodda usul;
- *Precision* – to‘g‘ri prognoz qilingan pozitiv qiymatlarning pozitiv deya prognoz qilingan qiymatlarining umumiy soni nisbati;
- *Recall* – to‘g‘ri prognoz qilingan pozitiv qiymatlarning ma’lumotlar to‘plamidagi barcha haqiqiy pozitiv nishonlarga nisbati;
- *F1-Score* – *Precision* hamda *Recall* orqali ifodalanadigan o‘lchov bo‘lib, model aniqligi haqida yanada aniqroq tasavvur beradi

Hisoblash uchun *calc_metrics* nomli funksiya yaratamiz:

```
def calc_metrics(tp, tn, fp, fn, n):
    accuracy = (tp + tn) / n
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1_score = 2 (recall precision) / (recall + precision)
    return accuracy, precision, recall, f1_score
```

calc_metrics funksiyasini ishga tushiramiz:

```
accuracy, precision, recall, f1_score = calc_metrics(tp, tn, fp, fn, n)
print('accuracy: %f, precision: %f, recall: %f, f1_score: %f'%
    (accuracy, precision, recall, f1_score))
```

```
accuracy: 0.772727, precision: 0.750000, recall: 0.545455, f1_score: 0.631579
```

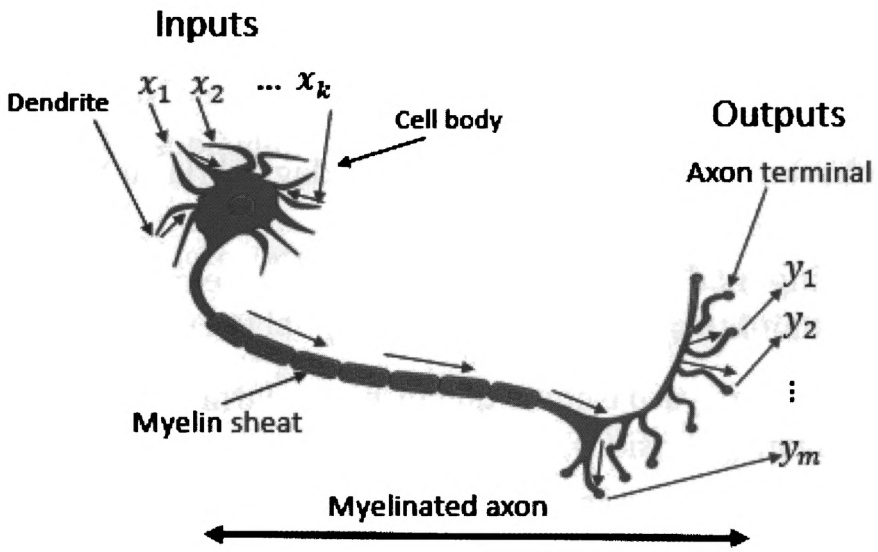
O‘lchovlar orqali model haqida quyidagi fikrlarni yuritishimiz mumkin:

- bizning model test to‘plamidagi 154 ta elementning 77 foizi nishonini to‘g‘ri topgan (accuracy);
- model *1* deya bashorat qilgan nishonlarning 75 foizi to‘g‘ri topilgan (precision)
- test to‘plamdagi *1* nishonlarning 54 foizi model tomonidan to‘g‘ri topilgan
- *F1-Score* 0.63 ni tashkil etmoqda. Ushbu o‘lchovni qiyosiy o‘lchov deb atasak ham bo‘ladi. Chunki, odatda, ushbu o‘lchovlar orqali turli modellarning aniqligi bir-biriga solishtiriladi.

O‘lchov natijalaridan xulosa qilib aytishimiz mumkinki, modelning aniqligi yaxshi. Albatta, buni yanada yaxshilash mumkin.

5 Neyron tarmoqlari

Neyron tarmoqlar mashinaviy o‘qitish algoritmlarining bir turi bo‘lib, kiruvchi qiymatlar ustida turli xil funksiyalar kombinatsiyasini qo‘llab, kerakli natijani hisoblaydi. Neyron tarmoq algoritmini amalga oshirish jarayonida xuddi chiziqli regressiya va logistik regressiyadagi kabi funksiyalardagi koeffitsiyentlar mashq jarayonida optimal qiymatga intiladi (albatta, bunda neyron tarmoq algoritmi to‘g‘ri amalga oshirilsa). Ushbu usul inson miyasidagi asab tolalarining ishlashidan ilhomlanib ishlab chiqilgan bo‘lib [10], neyron tarmoqlari bir necha qatlamdan iborat bo‘lishi mumkin. Xususan, 23-rasmda inson miyasidagi asab tolalarining bir-biriga ma’lumot almashishi ifodalangan. Mashinaviy o‘qitishda ham neyronlar xuddi shu kabi bir-biriga ma’lumot uzatib, neyron tarmoqlarini tashkil etadi va juda murakkab bo‘lgan hisoblarni amalga oshirishi mumkin.

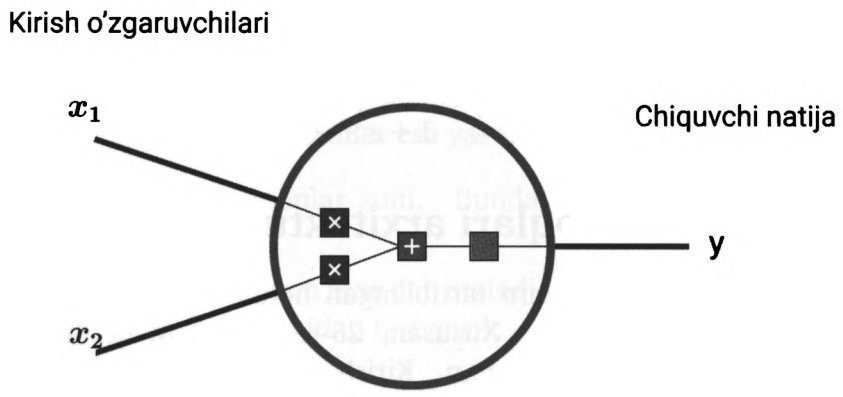


Rasm 23: Inson asab tolalarining ishlashi
[6]

5.1 Neyron o‘zi nima?

Neyron tarmoqlarini o‘rganish jarayonida “Neyron o‘zi nima?" degan tabiiy savol tug‘lishi mumkin. Neyron kirish o‘zgaruvchilarining ustida matematik

amallar bajaradi va natijani chiqaradi. Xususan, 24-rasm bir neyron elementini ko'rsatib beradi.



Rasm 24: Neyron elementi

Ko'rib turibmizki, neyronga o'zgaruvchilar kiritiladi (x_1 va x_2) va ular ustida turli matematik amallar bajariladi. Keyin esa natija neyron tarmog'ining boshqa qatnashchilariga uzatiladi (ushbu element oxirgi qatlam bo'lsa, u holda natija prognoz qilingan nishon sifatida qo'llanadi).

Kirish o'zgaruvchilari ustidagi matematik amallarga to'xtaladigan bo'lsak, avvaliga, xuddi chiziqli regressiyada bo'lganidek $f(x)$ funksiyani hisoblab olamiz:

$$f(x) = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 \cdots + a_n \times x_n + b \tag{58}$$

E'tibor bering, 58-formulada $a_0 \times x_0$ ifodaning o'rniga b koeffitsiyenti, ya'ni *bias* qo'llanilmoqda. Sababi hisob-kitoblarda, ayniqsa, gradient tushish amalida ushbu usul qulayroq hisoblanadi. 58-formulaning vektor ko'rinishi esa quyidagicha bo'ladi:

$$Z = f(X) = X * A^T \tag{59}$$

58-formulada, $a_1, a_2, \ldots a_n$ va b – neyronning parametrlari bo'lib, n – kiruvchi elementlar soniga bog'liqdir. $f(x)$ funksiyani hisoblab olganimizdan so'ng aktivlashtirish ($h_a(x)$) funksiyasi hisoblanadi va bu funksining qiymati ushbu neyronning eng so'nggi natijasi hisoblanadi va keyingi elementlarga uzatiladi. Aktivlashtirish funksiyasi sifatida, odatda, sigmoid funksiya ishlatiladi, ya'ni:

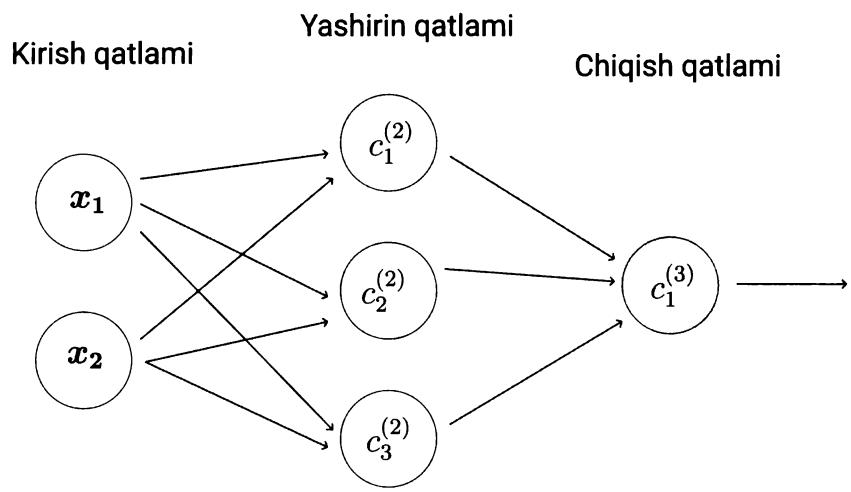
$$h_a(Z) = \frac{1}{1 + e^{-Z}} \tag{60}$$

60-formuladagi Z – 58-formulada hisoblangan $f(x)$ funksiyasining qiymatidir, ya'ni to'liq funksiyani quyidagicha yozishimiz mumkin:

$$f(X) = \frac{1}{1 + e^{-X*A^T}} \tag{61}$$

5.2 Neyron tarmoqlari arxitekturasi

Bir nechta neyronning o'zaro tartiblangan holda bir-biri bilan bog'lanishi **neyron tarmog'i** deb ataladi. Xususan, 25-rasmda 3 ta qatlamdan iborat bo'lgan neyron tarmog'i tasvirlangan. Kirish qatlamida kirish o'zgaruvchilari joylashadi, yashirin qatlamda esa bir qancha neyronlar joylashgan bo'ladi. Yashirin qatlamning o'zi ham birdan ortiq qatlamlardan iborat bo'lishi mumkin. Chiqish qatlamida esa bevosita ushbu neyron tarmoqdan chiquvchi oxirgi natijani hisoblovchi neyronlar joylashadi.



Rasm 25: Neyron tarmoqlari

Shu o'rinda biz foydalanadigan belgilar bilan ham tanishib ketishimiz joizdir. 25-rasmda keltirilgan neyron tarmoqni misol qilib oladigan bo'lsak, quyidagi belgilar bilan ishlaymiz:

- m – ma'lumotlar to'plami soni (qatorlar soni);
- n – kiruvchi o'zgaruvchilar soni (ustunlar soni);
- L – neyron tarmoqdagi qatlamlar soni. Bunda qatlamlarning tartib raqami quyidagicha bo'ladi:

  $1, 2, \cdots L$. 1-qatlam kirish qatlami deyiladi, L-qatlam chiqish qatlami deyiladi, $2, \cdots (L-1)$ qatlamlar esa yashirin qatlamlar deyiladi;
- s_l – l-qatlamdagi neyronlar soni. Bunda neyronlarning tartib raqami quyidagicha bo'ladi:

  $1, 2, \cdots l$ hamda 0 – neyron **bias** deb ataladi va uning chiqish qiymati doim 1 ga teng (chiqish qatlamidan tashqari);
- $s_{l,i}$ – l-qatlamdagi i-neyronning koeffitsiyentlari soni;
- K – chiqish qatlamidagi neyronlar soni. Bunda neyronlarning tartib raqami quyidagicha bo'ladi:

  $1, 2, \cdots K$, chiqish qatlamida bias qo'llanilmaydi;
- $x^{(i)}$ – i-qatordagi o'zgaruvchilar;
- $x_j^{(i)}$ – i-qatordagi j-o'zgaruvchining qiymati;
- X – ma'lumotlar to'plamidagi o'zgaruvchilarni ifoda etuvchi matritsa. Bunda ushbu matritsaning o'lchami $(m \times n)$;
- $A^{(l)}$ – neyron tarmoqning $(l+1)$-qatlamda yotuvchi koeffitsiyentlarini ifoda etuvchi matritsa. Ushbu koeffitsiyentlar l-qatlamining chiquvchi qiymatlari bilan matematik amallarga kirishadi (ya'ni ushbu koeffitsiyentlar l-qatlamning chiquvchi qiymatlariga ko'paytiriladi). Ushbu matritsaning o'lchami $(s_l \times n_i)$. Bunda s_l – l-qatlamdagi neyronlar (agarda ushbu qatlam kirish qatlami bo'lsa, kiruvchi o'zgaruvchilar) soni va n_i;
- $A_i^{(l)}$ – neyron tarmoqning $(l+1)$-qatlamda yotuvchi i – neyronining koeffitsiyentlarni ifoda etuvchi matritsa. Ushbu koeffitsiyentlar l-qatlamining chiquvchi qiymatlari bilan matematik amallarga kirishadi (ya'ni ushbu koeffitsiyentlar l-qatlamning chiquvchi qiymatlariga ko'paytiriladi). Ushbu matritsaning o'lchami $(s_l \times 1)$, s_l – l-qatlamdagi neyronlar (agarda ushbu qatlam kirish qatlami bo'lsa, kiruvchi o'zgaruvchilar) soni;

- $B^{(l)}$ – neyron tarmoqning $(l+1)$-qatlamda yotuvchi bias koeffitsiyentlarini ifoda etuvchi matritsa;
- $a_{i,j}^{(l)}$ – neyron tarmoqning $l+1$-qatlamda yotuvchi i-neyronning j-koeffitsiyenti;
- $b_i^{(l)}$ – neyron tarmoqning $l+1$-qatlamda yotuvchi i-neyronning bias koeffitsiyenti;
- $c_i^{(l)}$ – l-qatlamdagi i-neyronning chiqish qiymati;
- $c^{(l)}$ – l-qatlamdagi neyronlarning chiqish qiymatlarini ifoda etuvchi matritsa. Bunda ushbu matritsaning o'lchami $(s_{l-1} \times s_l)$;
- $g()$ – neyronlarning chiqish funksiyasi;
- $h_a(x)$ – neyron tarmoqning oxirgi natijasini (agarda $K > 1$ bo'lsa, natija $(K \times 1)$ o'lchamdagi matritsa bo'ladi) ifoda etuvchi funksiya;
- $y^{(i)}$ – i-qatordagi prognoz qilinadigan nishonning ma'lumotlar to'plamidagi asl qiymati;
- $\hat{y}^{(i)}$ – i-qatordagi prognoz qilingan nishon qiymati;
- $\hat{Y}$ – prognoz qilingan barcha nishonlarni ifodalovchi birlik, matritsa yoki vektor o'lchami $(1 \times m)$;
- $f(x)$ – prognozni hisoblash funksiyasi;

5.2.1 Multiklass neyron tarmoqlari

25-rasmda tasvirlangan neyron tarmoq binar klassifikatsiyani amalga oshiradi. Ya'ni ushbu neyron tarmoqdan quyidagi maqsadlarda foydalanish mumkin:

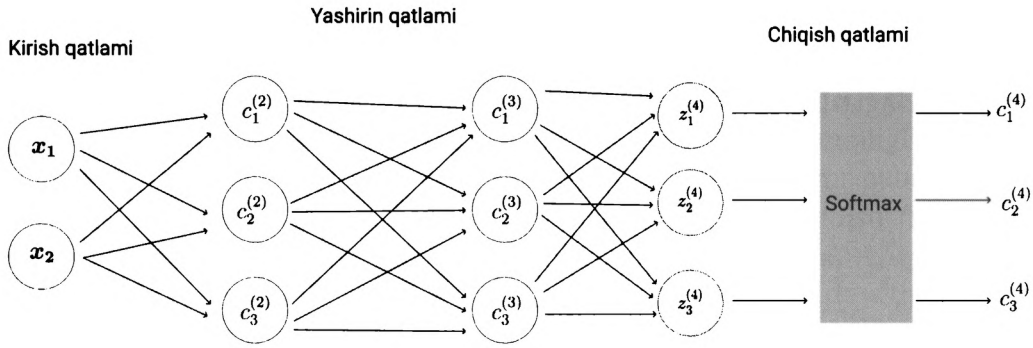
- E-mail xabarlarni spam yoki spam emasligini aniqlash;
- Berilgan rasmda mushuk bor yoki yo'qligini aniqlash va hokazo.

Lekin hayotda murakkabroq masalalar ham uchrab turadi. Aytaylik, e-mail xabarni bir necha kategoriyalarga ajratish kerak bo'lsin, xususan:

- asosiy;
- ijtimoiy tarmoqlar;

- reklama.

Yuqoridagi misolda ko'rib turibmizki, 25-rasmda tasvirlangan neyron tarmoq arxitekturasi to'g'ri kelmaydi. Sababi bizga chiqish qatlamida 3 ta neyrondan iborat bo'lgan arxitekturali neyron tarmoq kerak va ushbu masalani yechishda 26-rasmda tasvirlangan neyron tarmoqning arxitekturasi mos tushadi.



Rasm 26: Multiklass neyron tarmoqlari

26-rasmda tasvirlangan arxitekturaning chiqish qatlami 3 ta neyrondan iborat va ularning chiqish funksiyasi sifatida *softmax* funksiyasidan foydalanamiz. Ushbu turdagi neyron tarmoqdan e-mail xabarlarni asosiy, ijtimoiy tarmoqlar va reklama kategoriyalariga ajratish masalasini yechishda foydalanish uchun chiqish qatlamidagi har bir neyronni ma'lum bir kategoriyaga moslaymiz. Ya'ni $c_1^{(4)}$ – asosiy, $c_2^{(4)}$ – ijtimoiy tarmoqlar, $c_3^{(4)}$ – reklama kategoriyasini ifoda etsin. Bunda biz quyidagi formatda chiqish qiymatlarini olishimiz mumkin:

$$h_x(a) = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{62}$$

62-formulada neyron tarmoqning oxirgi natijasini ifoda etuvchi (3×1) matritsa ifodalangan va ushbu natijadan ko'rinib turibdiki, prognoz qilinayotgan e-mail xabar asosiy kategoriya sifatida klassifikatsiya qilindi. Bunga sabab matritsaning 1-elementi, ya'ni $c_1^{(4)}$-neyronning qiymatini 1 ga teng, qolgan 2- va 3-elementlar qiymati 0 ga teng deb olindi. E-mail xabarning kategoriyasi

sifatida ijtimoiy tarmoqlar klassifikatsiya qilinganda 63-formulada ifodalangan chiqish qiymatlarini olishimiz mumkin.

$$h_x(a) = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{63}$$

Shu o'rinda ta'kidlab o'tish joizki, neyron tarmoqlar o'qitilayotgan ma'lumotlar to'plamidagi haqiqiy nishonlar ham 62 va 63-ifodalarda keltirilgan formatda bo'lishi kerak. Misol tariqasida, 64-ifodada ko'rsatilgan ma'lumotlar to'plamidagi nishonning asl qiymatini keltirishimiz mumkin.

$$y^{(i)} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{64}$$

Shu tariqa, chiqish qatlamida birdan ko'p neyronlarni joylashtirish orqali multiklass neyron tarmoqlarini yaratib, murakkabroq masalalar, xususan, ko'p kategoriyali klassifikatsiyalash muammosini yechish mumkin. Albatta, ko'p kategoriyali klassifikatsiyalash muammosini yechishda so'nggi qatlamda chiquvchi funksiya rolini bajaruvchi – *softmax* funksiya haqida batafsil ma'lumot berishimiz joizdir.

5.2.2 Softmax funksiyasi

Softmax funksiyasi berilgan sonlar to'plamini ularning bir-biriga qiyosiy qiymatiga qarab har bir elementining qiymati $[0, 1]$ oraliqda yotuvchi ehtimolliklarni ifoda etuvchi sonlar to'plamiga aylantiradi. Masalan, tasavvur qilaylik, bizda quyidagicha sonlar to'plami mavjud:

$$y = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \tag{65}$$

Yuqoridagi to'plamni ma'lum bir elementlar bilan bog'laylik. Xususan, e-mail xabarlarni kategoriyalarga ajratish muammosini olsak, **asosiy** kategoriyaning klassi 1-element (qiymati 2), **ijtimoiy tarmoqlar** kategoriyasining klassi 2-element (qiymati 1) va nihoyat, **reklama**

kategoriyasining klassi sifatida 3-element (qiymati 0) bo'lsin. Ushbu sonlar to'plamiga *softmax* funksiyasini tatbiq qilganimizdan so'ng natija sifatida biz quyidagi ehtimolliklar to'plamini olamiz:

$$y = \begin{bmatrix} 0.7 \\ 0.2 \\ 0.1 \end{bmatrix} \tag{66}$$

66-ifodadagi sonlar to'plami ustida matematik amallarni bajarish 65-ifodadagi sonlar to'plamidan ancha qulay. Sababi softmax funksiyasida tatbiq etilgan sonlar to'plami ma'lum bir oraliqda yotadi (ya'ni $[0, 1]$). Bu esa bizga ushbu to'plamning elementlari qiymatlarini klasslar ehtimolligi sifatida qabul qilish imkonini beradi. Bundan tashqari, ma'lum bir qaror qabul qilish chegarasini tanlashda qulaylik yaratadi. Masalan, qaror qabul qilish chegarasi sifatida 0.5 qiymat olingan bo'lsa, u holda ushbu qabul qilingan chegaradan yuqori qiymatli elementni ushbu to'plamning asosiy klassi sifatida e'tirof etish mumkin. Masalan, 66-ifodadagi elementlarning qiymatlari orqali quyidagi fikrlarni e'tirof etish mumkin:

– Ushbu to'plam 70% ehtimollik bilan **asosiy** kategoriyani, 20% ehtimollik bilan **ijtimoiy tarmoqlar** kategoriyasini va 10% ehtimollik bilan **reklama** kategoriyasini anglatadi. Qaror qabul qilish chegarasini (ya'ni 0.5) inobatga olib aytganda, ushbu to'plam **asosiy** kategoriyasini anglatadi.

Endi **softmax** funksiyasining formulasiga to'xtalsak:

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{67}$$

Yuqoridagi formulada $i = [1...K]$ hamda $z = [z_1...z_K]$ oraliqlarida joylashadi va K klasslar sonini, z_i (yoki z_j) esa oxirgi qatlamning neyronlaridan chiquvchi qiymatlarini ifoda etadi (bunda $z = a_1 \times c_1 + a_2 \times c_2 + \ldots + a_n \times c_n$ ifodasining natijasidir).

5.2.3 Oldinga siljish algoritmi

Neyron tarmoqdagi har bir qatlam neyronlarining chiqish qiymatlarini hisoblab, ushbu qiymatlarni keyingi qatlamga uzatish jarayoni – **oldinga siljish algoritmi** deyiladi. Hisob-kitoblar 2-qatlamdan boshlanadi: 1-qatlam – kirish qatlami, ya'ni kiruvchi o'zgaruvchilar qatlami bo'lgani uchun 1-qatlamning qiymatlari bevosita 2-qatlamga uzatiladi.

Shu o'rinda kiruvchi o'zgaruvchilarning qay tariqa neyron tarmoqlariga uzatilishi xususida so'z yuritsak. Sababi hisob-kitoblarni tezlashtirish uchun odatda vektorlangan hisob-kitoblardan foydalaniladi, ya'ni bir qadamning o'zida mashq to'plamining bir-necha qatori birdaniga neyron tarmoqqa kiritiladi. Biroq odatda mashq to'plamlari juda katta hajmdagi ma'lumotlardan iborat bo'lgani sababli hisob-kitoblar juda katta xotira talab qilishi mumkin. Shuning uchun ma'lumotlar to'plami qatorlarini bir necha bo'lakka bo'lib, neyron tarmoqqa uzatiladi. Ushbu bo'laklar *batch*, deb ataladi hamda giperparametr hisoblanib, neyron tarmoq amallari boshlanishidan oldin tanlanadi. Aytaylik, mashq to'plamida 1250 ta qator mavjud va biz *batch* hajmi sifatida (*batch_size*) 100 ni tanladik (Ya'ni $batch\_size = 100$). Bunda neyron tarmoqqa birinchi 100 qatorni (1–100-qatorlarni) kiritamiz va neyron tarmoqni mashq qildiramiz. Keyin ikkinchi 100 talikni (101–200-qatorlarni) neyron tarmoqqa kiritamiz va mashq jarayonini amalga oshiramiz. Ushbu sikl oxirida ma'lumotlar to'plamining oxirgi 50 ta qatorini (1201–1250-qatorlarni) neyron tarmoqqa kiritamiz va mashqni amalga oshiramiz. Ushbu jarayon o'z navbatida davrlar sonicha (ya'ni *epochs*) takrorlanadi. Shunda biz har bir davrda 13 ta *batches* yoki bo'laklardan (e'tibor bering 13 – *batch* 50 ta qatorni o'z ichiga oladi) iborat bo'lgan ichki siklga ega bo'lamiz. Ma'lumotlar to'plamini bo'laklarga bo'lib, ya'ni *batch*lar orqali neyron tarmoqni mashq qildirish quyidagi afzalliklarga ega [1]:

- Kamroq xotira talab qilinadi, chunki ma'lumotlar to'plami butunligicha emas, bo'laklarga bo'lingan holda (*batch*) neyron tarmoqqa kiritiladi;
- Neyron tarmoq optimal parametrlarga tezroq ega bo'ladi, chunki har bir *batch* siklida neyron tarmoqning parametrlarni o'zgartirish amali bajariladi.

Albatta, ushbu *batch_size* parametri, ya'ni har bir bo'lak soni **giperparametr** hisoblanadi va uning qiymatini vaziyatdan kelib chiqib to'g'ri tanlash talab etiladi. Sababi qiymat sifatida juda kichik son tanlansa, neyron tarmoqning tezligini va aniqligini tushirib yuboradi.

Endi bevosita oldinga siljish algoritmiga qaytsak. 25-rasmdagi neyron tarmoq misolida ushbu algoritmni ko'rib chiqamiz, bunda hisob-kitoblar quyidagicha amalga oshiriladi:

Avvaliga, yashirin qatlamning chiqish qiymatlarini hisoblab olamiz:

$$\begin{aligned} c_1^{(2)} &:= g(a_{1,1}^{(2)} \times x_1 + a_{1,2}^{(2)} \times x_2 + b_1^{(2)}) \\ c_2^{(2)} &:= g(a_{2,1}^{(2)} \times x_1 + a_{2,2}^{(2)} \times x_2 + b_2^{(2)}) \\ c_3^{(2)} &:= g(a_{3,1}^{(2)} \times x_1 + a_{3,2}^{(2)} \times x_2 + b_3^{(2)}) \end{aligned} \tag{68}$$

68-formulada $g()$ funksiyasi aktivlashtirish funksiyasi bo'lib, *sigmoid* funksiyasini qo'llash mumkin. So'ngra 3-qatlamdagi neyronlarning chiqish qiymatini hisoblaymiz va bu oxirgi natija hisoblanadi:

$$\begin{aligned} z_1^{(3)} &:= a_{1,1}^{(3)} \times c_1^{(2)} + a_{1,2}^{(3)} \times c_2^{(2)} + a_{1,3}^{(3)} \times c_3^{(2)} + b_1^{(3)} \\ z_2^{(3)} &:= a_{2,1}^{(3)} \times c_1^{(2)} + a_{2,2}^{(3)} \times c_2^{(2)} + a_{2,3}^{(3)} \times c_3^{(2)} + b_2^{(3)} \\ z_3^{(3)} &:= a_{3,1}^{(3)} \times c_1^{(2)} + a_{3,2}^{(3)} \times c_2^{(2)} + a_{3,3}^{(3)} \times c_3^{(2)} + b_3^{(3)} \end{aligned} \tag{69}$$

3-qatlam uchun z_i qiymatlarni hisoblab olganimizdan so'ng ularni *softmax* funksiyasi orqali klasslar ehtimolliklariga aylantiramiz:

$$c_1^{(3)}, c_2^{(3)}, c_3^{(3)} := softmax(z_1^{(3)}, z_2^{(3)}, z_3^{(3)}) \tag{70}$$

70-formuladagi *softmax* funksiyasini quyidagi ifoda orqali ta'riflash mumkin:

$$\begin{aligned} c_1^{(3)} &:= \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}} \\ c_2^{(3)} &:= \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}} \\ c_3^{(3)} &:= \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}} \end{aligned} \tag{71}$$

71-formuladagi ifoda bizga qiymatlari $[0, 1]$ oraliqda joylashgan va har bir klass uchun ehtimolliklarni beradi. Ushbu ehtimolliklar orqali biz natijalarni haqiqiy klass qiymatlariga tekshirishimiz, ya'ni qiymat funksiyasini amalga oshirishimiz mumkin.

5.2.4 Qiymat funksiyasi

Neyron tarmoqlarda qiymat funksiyasi eng oxirgi qatlamning, ya'ni chiqish qatlamining neyronlari qanchalik to'g'ri natija berganini baholovchi funksiyadir. Qiymat funksiyasi neyron tarmoq qo'llanilayotgan muammoga qarab tanlanadi. Masalan, neyron tarmoq chiziqli regressiya muammosiga o'xshab davomiy qiymatlarni (uy narxlarini) bashorat qilsa, unda o'rta kvadratik xatolik funksiyasidan (ing. *mean squared error*) foydalanishimiz mumkin:

$$J = \frac{1}{2m} \sum_{i=1}^{m} (f(x^{(i)}) - y^{(i)})^2 \tag{72}$$

Neyron tarmoqlari 2 ta klassdan iborat bo'lgan klassifikatsiyalash muammosiga (xususan, berilgan rasmda kuchuk bor ($klass = 1$) yoki yo'qligi

($klass = 0$)) tatbiq etilayotgan bo'lsa, binar-logorifmik xatolik funksiyasini (yoki ing. *binary cross-entropy loss*) tatbiq etishimiz mumkin (bunda K – klasslar soni):

$$J = -\left[y^{(i)} \log(f(x^{(i)})) + (1 - y^{(i)}) \log(1 - f(x^{(i)}))\right] \tag{73}$$

Agarda neyron tarmoqlari 2 tadan ortiq klassdan iborat bo'lgan klassifikatsiyalash (masalan, e-mail xabarlarni *asosiy*, *ijtimoiy tarmoqlar* va *reklama* kategoriyalariga ajratish) muammosiga qo'llanilayotgan bo'lsa, multiklass logorifmik xatolik funksiyasini (yoki ing. *cross-entropy loss*) qo'llashimiz mumkin:

$$J = -\sum_{j=1}^{K} \left[y_j \log(f(x_j))\right] \tag{74}$$

Yuqoridagi formulada y_j – klassning haqiqiy qiymati va $\log(f(x_j))$ – neyron tarmoqning oxirgi qatlami neyronlarining chiqish qiymati (albatta, bu yerda oxirgi qatlam chiquvchi funksiyasi sifatida *softmax* funksiyasi qo'llanilyapti).

Biz neyron tarmoqlar e-mail xabarlarni 3 xil kategoriyaga (*asosiy*, *ijtimoiy tarmoqlar* va *reklama*) klassifikatsiyalash muammosi ustida ko'rib chiqayotganimiz uchun 74-formulada ifodalangan multiklass logorifmik xatolik funksiyasini qiymat funksiyasi sifatida olamiz. Keyingi amallar ushbu qiymat funksiyasiga asoslanadi.

Xususan, 26-rasmda ifoda etilgan neyron tarmoq arxitekturasiga moslashtirsak, qiymat funksiyasini quyidagicha ifoalasak bo'ladi:

$$J = -\sum_{j=1}^{K} \left[y_j \log(c_j^{(L)})\right] \tag{75}$$

Bunda 75-formuladagi y_j nishon qiymatlarining maxsus ko'rinishga keltirilgan vektor elementi bo'lib, ushbu maxsus vektorni *one-hot vektor* deb ataylik. *One-hot vektor*ni tushunish uchun e-mail xabarlarni kategoriyalarga (*asosiy*, *ijtimoiy tarmoqlar* va *reklama*) ajratish muammosi asosida misol ko'rib chiqaylik. Har bir kategoriyani ma'lum bir son bilan belgilaylik. Xususan, *asosiy* – 1; *ijtimoiy tarmoqlar* – 2; *reklama* – 3. Endi tasavvur qiling, bizda quyidagi nishonlar jadvali mavjud:

5-jadvalda 3 ta qator (albatta, osonlik uchun katta bo'lmagan ma'lumotlar to'plamidan foydalanyapmiz) va kategoriyani ifoda etuvchi birgina ustun mavjud (bunda birinchi qatordagi qiymati 2 bo'lgan element *ijtimoiy tarmoqlar*ni bildiradi). Ushbu nishonlar to'plamini *one-hot vektor* ko'rinishiga keltirish uchun har bir kategoriya uchun yangi ustun ajratamiz va kerakli

Kategoriya
2
3
1

Jadval 5: E-mail xabarlar kategoriyalarning nishonlari jadvali

kategoriya ustuniga 1, qolganlariga esa 0 yozib chiqamiz. Xususan, 5-jadvalda aks ettirilgan ma'lumotlar to'plamining *one-hot vektor* ko'rinishi quyidagicha bo'ladi:

Asosiy	ijtimoiy tarmoqlar	Reklama
0	1	0
0	0	1
1	0	0

Jadval 6: E-mail xabarlar kategoriyalaring nishonlari *one-hot vektor* jadvali

6-jadval ifoda etgan to'plamga ahamiyat bersak, har bir qatorda qiymati 1 bo'lgan ustundagi kategoriya aktiv hisoblanadi. Xususan, 1-qatorga nazar solsak, *ijtimoiy tarmoqlar* ustunidagi qiymat 1 va boshqa ustunlar 0 qiymatga ega. Bu esa 5-jadvaldagi 1-qator elementi kategoriyasi bilan to'la mos keladi. Ushbu *one-hot vektor* ko'rinishidagi nishonlar to'plami algoritmlar ishlashida qulaylik tug'diradi va effektivligini oshiradi.

Endi 75-formulaga qaytamiz. Demak, ushbu formuladagi y_j elementini tushundik, ya'ni nishon qiymatlarining *one-hot vektor* ko'rinishidagi matritsasining j-indeksdagi qiymati. $c_j^{(L)}$ – neyron tarmoqning L-qatlamdagi j-neyron chiqish qiymati (albatta, bu yerda oxirgi qatlam chiquvchi funksiyasi sifatida *softmax* funksiyasi qo'llanilyapti). Qiymat funksiyasini aniqlab oldik. Endi keyingi qadam – ortga siljish algoritmini amalga oshirishdir.

5.2.5 Ortga siljish algoritmi

Neyron tarmoqning samaradorligini oshirish uchun qiymat funksiyasini minimallashtirish, ya'ni optimal koeffitsiyentlarni topishimiz kerak. Bunda biz ortga siljish algoritmidan foydalanamiz. Ortga siljish algoritmining asosiy maqsadi – bu neyron tarmoqdagi har bir koeffitsiyentning o'zgarishi qiymat funksiyasiga qanday ta'sir qilishini topishdir. Ushbu ta'sirni topganimizdan so'ng koeffitsiyentni qiymat funksiyasini kamaytiradigan tarzda yangilashimiz mumkin. Ushbu ta'sirni topish uchun avvaliga oldinga siljish algoritmini

amalga oshiramiz, ya'ni $l = 2, 3, \ldots L$ qatlamlar uchun $z^{(l)}$ va $c^{(l)}$ o'zgaruvchilar qiymatlarini hisoblab olamiz. So'ngra ortga siljish algoritmini hisoblaymiz. Bunda harakat oldinga siljishga nisbatan teskari, ya'ni oxirgi qatlamdan boshlanadi: $L, L-1, \ldots 3, 2$. Yuqorida ta'kidlab o'tganimizdek, ortga siljish algoritmi koeffitsiyentlarning qiymat funksiyasiga ta'sirini o'rganadi, ya'ni barcha qatlamlardagi neyron koeffitsiyentlari uchun $\frac{\partial J}{\partial a_{i,j}^{(l)}}$ va $\frac{\partial J}{\partial b_i^{(l)}}$ ifodalarni hisoblashdir. Ushbu ifodalarni hisoblashdan oldin har bir qatlamdagi neyron uchun yordamchi parametr – $\delta_j^{(l)}$ ni hisoblab olishimiz kerak bo'ladi. Ushbu parametr odatda neyron xatoligi, deb ataladi va chiqish qatlami uchun quyidagi formula o'rinlidir:

$$\delta_j^{(L)} = \frac{\partial J}{\partial z_j^{(L)}} \tag{76}$$

76-formulada $\delta_j^{(L)}$ qiymati sifatida chiqish qatlamdagi $z_j^{(L)}$ o'zgaruvchisining qiymat (J) funksiyasiga ta'siri hisoblanmoqda. Ushbu ta'sirni topish, albatta, oxirgi qatlamdagi chiqish funksiyasiga hamda foydalanilayotgan qiymat funksiyasiga bevosita bog'liqdir. Bizda so'nggi qatlamdagi chiqish funksiyasi sifatida $softmax()$ hamda qiymat funksiyasi sifatida ishlatilayotgani sababli 76-formulani quyidagicha soddalashtirishimiz mumkin (quyidagi formulada z va c o'zgaruvchilari oxirgi qatlamning o'zgaruvchilaridir, ya'ni $z^{(L)}$ va $c^{(L)}$):

$$\delta_j^{(L)} = \frac{\partial J}{\partial z_j}$$

$$= \frac{\partial}{\partial z_j}\Big[-\sum_{k=1}^{K} \big(y_k \log(c_k)\big)\Big]$$

$$= -\sum_{k=1}^{K} y_k \frac{\partial(\log(c_k))}{\partial z_j}$$

$$= -\sum_{k=1}^{K} y_k \frac{\partial(\log(c_k))}{\partial c_k} \times \frac{\partial c_k}{\partial z_j}$$

$$= -\sum_{k=1}^{K} \frac{y_k}{c_k} \times \frac{\partial c_k}{\partial z_j}$$

$$= -\Big[\frac{y_j}{c_j} \times \frac{\partial c_j}{\partial z_j} + \sum_{k=1,k\neq j}^{K} \frac{y_k}{c_k} \times \frac{\partial c_k}{\partial z_j}\Big] \tag{77}$$

$$= -\frac{y_j}{\cancel{c_j}} \times \cancel{c_j}(1-c_j) - \sum_{k=1,k\neq j}^{K} \frac{y_k}{\cancel{c_k}}(\cancel{c_k} c_j)$$

$$= -y_j + y_j c_j + \sum_{k=1,k\neq j}^{K} y_k c_j$$

$$= c_j(y_j + \sum_{k=1,k\neq j}^{K} y_k) - y_j$$

$$= c_j \times \sum_{k=1}^{K} y_k - y_j$$

$$= c_j \times 1 - y_j, \text{ chunki} \sum_{k=1}^{K} y_k = 1$$

$$= c_j - y_j$$

77-formuladan ko‘rinib turibdiku, chiqish funksiya sifatida $softmax()$ ishlatilganda $\delta_j^{(L)}$ ning qiymati $c_j - y_j$ ga teng ekan. Chiqish qatlami uchun xatolikni (δ^L) hisoblab olganimizdan so‘ng keyingi qatlamdagi, aniqrog‘i, chiqish qatlamdan oldingi qatlamlardagi xatolikka ($\delta^{(l)}$) to‘xtalamiz.

$$\delta^l = ((c^{(l)})^T \delta^{(l+1)}) \odot g'(z^{(l)}) \tag{78}$$

78-formulani tavsiflashdan oldin bizga notanish bo‘lgan belgiga ta’rif berib

ketishimiz joizdir. $\odot$ – o'lchamlari bir xil bo'lgan ikki vektorning (yoki matritsaning) indekslari mos tushuvchi elementlari ko'paytmasidir. Ushbu ko'paytmani chuqurroq tushunish uchun quyidagi amalga e'tibor bering:

$$\begin{bmatrix}1\\2\end{bmatrix}\odot\begin{bmatrix}3\\4\end{bmatrix}=\begin{bmatrix}1\times 3\\2\times 4\end{bmatrix}=\begin{bmatrix}3\\6\end{bmatrix} \tag{79}$$

79-amaldan ko'rib turibmizki, indekslari mos tushuvchi elementlar o'zaro ko'paytirilmoqda.

Endi bevosita 78-formula tavsifiga o'tadigan bo'lsak, ushbu tenglikning o'ng tomonidagi ko'paytmasining ($\odot$) birinchi ishtirokchisi, ya'ni $((c^{(l)})^T\delta^{(l+1)})$ ga e'tibor beraylik. Bunda ko'rib turibmizki, l-qatlamning chiqish qiymatlari transpozitsiyasining $(l+1)$-qatlam xatoligiga matritsaviy ko'paytmasi aks etgan. Qatlam xatoliklarini topishda oxirgi qatlamdan boshlanishini hisobga olsak, l-qatlamdagi xatolikni hisoblayotgan paytda $(l+1)$-qatlam xatoligi allaqachon hisoblangan bo'ladi. Shuning uchun bu yerda bizda mavjud qiymatlar o'zaro hisob-kitobga kirishmoqda. Endi 78-formuladagi ko'paytmaning 2-ishtirokchisini, ya'ni $g'(z^{(l)})$ ifodani tavsiflasak. Ushbu ifodada l-qatlam chiqish funksiyasining hosilasi tasvirlangan. Ya'ni l-qatlamdagi $z^{(l)}$ parametri o'zgarishining chiqish funksiyasiga ta'siri ifoda etilgan. Albatta, ushbu ifoda neyron tarmoq qatlamining chiqish funksiyasiga bevosita bog'liqdir. Xususan, $z^{(l)}$ parametrining chiqish funksiyasiga ta'sirini, boshqacha aytganda, chiqish funksiyasining $z^{(l)}$ parametrga nisbatan hosilasini topish talab etiladi. Ushbu hosilani $(g'(z^{(l)}))$ *sigmoid* funksiyasi misolida ko'rib chiqsak:

$$\sigma'(x)=\sigma(x)(1-\sigma(x)),\ \text{bunda}\ \sigma(x)=\frac{1}{1+e^{-x}} \tag{80}$$

80-formulada $sigmoid$ funksiyasining hosilasi aks ettirilgan. Ushbu ifodani 78-formulaga tatbiq etaylik:

$$\delta^l=((c^{(l)})^T\delta^{(l+1)})\odot(\sigma(z^{(l)})(1-\sigma(z^{(l)}))) \tag{81}$$

81-formula orqali barcha qatlamlardagi xatoliklarni $(\delta^{(l)})$ aniqlab olganimizdan so'ng neyron koeffitsiyentlari uchun $\frac{\partial J}{\partial a_{i,j}^{(l)}}$ va $\frac{\partial J}{\partial b_i^{(l)}}$ ifodalarni hisoblashga o'tsak bo'ladi. Ushbu ifodalarni quyidagi formula orqali hisoblaymiz:

$$\begin{aligned}\frac{\partial J}{\partial a_{i,j}^{(l)}}&=c_i^{(l)}\delta^{(l)}\\ \frac{\partial J}{\partial b_i^{(l)}}&=\delta^{(l)}\end{aligned} \tag{82}$$

82-formuladan ko‘rib turibmizki (ushbu formula, albatta, oxirgi qatlam, ya’ni L-qatlamga ham tatbiq qilinadi), l-qatlamdagi asosiy parametrning $(a_{i,j})$ qiymat funksiyasiga ta’sirini aniqlash uchun ushbu parametr joylashgan i-neyronning chiqish qiymatini $(c_i^{(l)})$ joriy qatlam xatoligiga ko‘paytirish kerak. l-qatlamdagi bias parametrlarning (b) qiymat funksiyasiga ta’sirini aniqlash uchun $(\delta^{(l)})$ foydalaniladi. 82-formuladagi ifodalarni hisoblab olganimizdan so‘ng parametrlarni yangilashga o‘tsak bo‘ladi:

$$\begin{aligned} a_{i,j}^{(l)} &= a_{i,j}^{(l)} - \alpha \frac{\partial J}{\partial a_{i,j}^{(l)}} \\ b_i^{(l)} &= b_i^{(l)} - \alpha \frac{\partial J}{\partial b_i^{(l)}} \end{aligned} \tag{83}$$

83-formulada ko‘rinib turibdiki, parametrlarni oldingi boblardagi (masalan, chiziqli regressiya, logistik regressiyalardagi) kabi yangilanadi. Bunda α – o‘rganish darajasi.

Umuman olganda, ortga siljish algoritmini quyidagi qadamlarda amalga oshirish mumkin:

1. **Oldinga siljish algoritmini amalga oshiramiz.** $l = [2, 3, \dots L]$ qatlamlar uchun $z^{(l)} = a^{(l-1)}c^{(l)} + b^l$ va chiqish qiymatlarini hisoblab olamiz: $c^{(l)} = \sigma(z^{(l)})$. Bunda $\sigma(z^{(l)})$ – l-qatlam chiqish funksiyasi;

2. **Chiqish qatlam xatoligini hisoblaymiz.** $\delta^{(L)} = \frac{\partial J}{\partial z^{(L)}}$. Bunda agar chiqish funksiyasi sifatida $softmax()$ hamda qiymat funksiyasi sifatida multiklass logorifmik xatolik funksiyasi $(\sum_{j=1}^{K}\left[y_j \log(f(x_j))\right])$ ishlatilayotgan bo‘lsa, yakuniy ifoda quyidagicha o‘rin oladi: $c_j - y_j$;

3. $l = [L-1, L-2, \dots 3, 2]$ **qatlamlar uchun xatolikni hisoblaymiz.** $\delta^l = ((c^{(l)})^T \delta^{(l+1)}) \odot g'(z^{(l)})$;

4. **Barcha qatlamlar uchun hisoblangan xatoliklar orqali (δ) parametrlarining qiymat funksiyasiga ta’sirini hisoblaymiz.** $\frac{\partial J}{\partial a_{i,j}^{(l)}} = c_i^{(l)}\delta^{(l)}$ – asosiy parametrlar uchun va $\frac{\partial J}{\partial b_i^{(l)}} = \delta^{(l)}$ – bias parametrlar uchun;

5. **Parametrlarni yangilaymiz.** $a_{i,j}^{(l)} = a_{i,j}^{(l)} - \alpha \frac{\partial J}{\partial a_{i,j}^{(l)}}$ – asosiy parametrlar uchun $b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial J}{\partial b_i^{(l)}}$ – bias parametrlar uchun.

Yuqoridagi qadamlar oldindan belgilangan ma’lum bir davr (epochs) mobaynida takrorlanadi va shu tariqa parametrlarning optimal qiymatiga ega bo‘linadi.

5.3 Amaliy mashg'ulot

Ushbu amaliy mashg'ulotning *github.com* saytidagi repozitoriysiga `https://bit.ly/3qRgBsb` ssilkasi orqali yoki quyidagi QR-kodni skaner qilish orqali o'tishingiz mumkin. Aynan shu amaliy mashg'ulotning **.ipynb* fayli uchun `practice_session/Neural_networks_6_3/` papkasini tekshiring.

Ushbu amaliy mashg'ulotda *Jupyter Notebook* dasturi orqali Pyton dasturlash tilida Neyron tarmoq muammosini o'rganamiz. *Jupyter Notebook* dasturi orqali ma'lum bir katalogga yangi "*.ipynb*" faylni yaratamiz va quyidagi kodlarni navbatma-navbat yozib boramiz.

Quyidagi Pyton kutubxonalarini yuklab olamiz:

- numpy – Pytonda massivlar bilan ishlash uchun;
- matplotlib – Pytonda vizualizatsiya, diagrammalar bilan ishlash;
- math – Pytonda matematik amallar bilan ishlash;
- h5py – HDF5 binar formatdagi fayllar bilan ishlovchi kutubxona (Ko'proq ma'lumot uchun: https://docs.h5py.org/en/stable/);
- sklearn – Pytonda mashinaviy o'qitish algoritmlari bilan ishlash uchun *Scikit-learn* kutubxonasi. Biz ushbu kutubxonadan modelning aniqligini o'lchashda foydalanamiz.

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from matplotlib.pyplot import figure
4 import h5py
5 from sklearn import metrics
```

Neyron tarmoq uchun **"Kaggle"** ilovasida keltirilgan ma'lumotlar to'plamlarining biridan (`http://yann.lecun.com/exdb/mnist/` havolada joylashgan) foydalanamiz. Ushbu ma'lumotlar to'plamini yuklab olamiz hamda ushbu *".ipynb"* fayl joylashgan katalogga joylaymiz va arxivdan chiqaramiz. So'ngra *pandas* kutubxonasi yordamida yuklaymiz:

pip install python-mnist

Hisoblarda qulayroq bo'lishi uchun ushbu ma'lumotlar to'plamining *h5py* formatdagi ko'rinishidan foydalanamiz. Ma'lumotlar to'plamini quyidagi havoladan yuklab olib, ushbu *.ipynb* fayl joylashgan katalogga joylash kerak bo'ladi.

`https://github.com/Zhenye-Na/cs598/blob/master/assignments/mp1/MNISTdata.hdf5`

h5py kutubxonasi yordamida ma'lumotlar to'plamini yuklaymiz:

```
MNIST_data = h5py.File("./MNISTdata.hdf5", 'r')
```

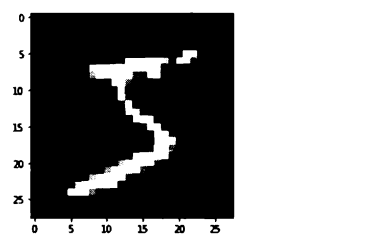
Ma'lumotlar to'plamini *numpy* formatidagi massivga yuklaymiz. Bunda kerakli formatni ko'rsatamiz, ya'ni kiruvchi to'plam uchun *float32* va nishonlar to'plami uchun *int32*.

```
x_train = np.float32(MNIST_data['x_train'][:])
y_train = np.int32(np.array(MNIST_data['y_train'][:, 0])).reshape(-1, 1)
x_test = np.float32(MNIST_data['x_test'][:])
y_test = np.int32(np.array(MNIST_data['y_test'][:, 0])).reshape(-1, 1)
MNIST_data.close()
```

Ma'lumotlar to'plami to'g'ri yuklanganini tekshirish uchun **matplotlib** kutubxonasi orqali ma'lumotlar to'plamining biron elementini chizib ko'ramiz.

```
plt.imshow(x_train[0,:].reshape(28,28), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7fa44905f190>
```



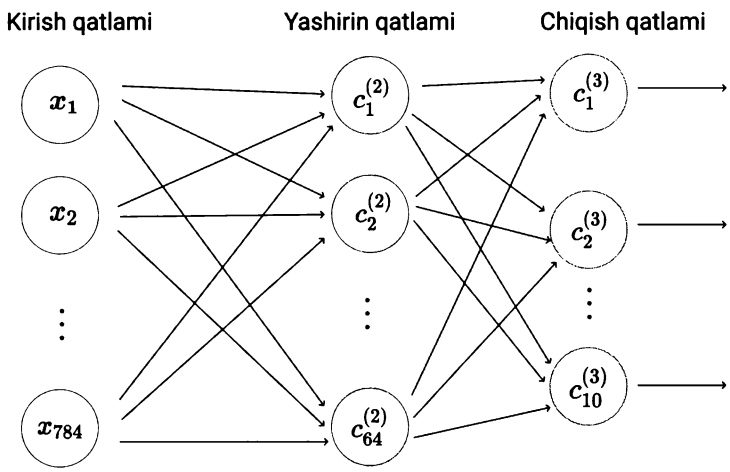
Ko‘rib turganimizdek, 5 raqamini tasvirlovchi rasm chizildi va bu shundan dalolat beradiki, ma’lumotlar to‘plami to‘g‘ri yuklangan.

Ma’lumotlar to‘plami bilan kengroq tanishib chiqaylik. Avvalo, to‘plamlar qanday hajmda ekanini aniqlaylik:

```
1 print(x_train.shape)
2 print(y_train.shape)
3 print(x_test.shape)
4 print(y_test.shape)
```

```
(60000, 784)
(60000, 1)
(10000, 784)
(10000, 1)
```

Biz arxitekturasi quyida keltirilgan neyron tarmoqni qo‘llaymiz:



Neyron tarmoqning ushbu arxitekturasida 3 ta qatlam ishtirok etmoqda:

- Kirish qatlami o‘lchami: 784. Sababi kirish o‘zgaruvchilarining har bir qatlami 784 ustundan iborat.
- Yashirin qatlam yoki 2-qatlam 64 ta neyrondan iborat. Yashirin qatlamlar uchun neyronlar sonini topishning aniq bir qoidasi yo‘q, lekin, asosan, kirish va chiqish qatlamlariga yaqinroq va mos ravishdagi son tanlanadi.
- Chiquvchi qatlam yoki 3-qatlam 10 ta neyrondan iborat. Bunda har bir neyron ma’lum bir sonni bildiradi. Masalan, 1-neyron 0 raqamini ifodalaydi va h.k.

Ushbu arxitekturaga moslash uchun kiruvchi to'plamning shaklini o'zgartiraylik, ya'ni transpozitsiyalaylik:

```
x_train = x_train.T
x_test = x_test.T
```

Kiruvchi to'plam, aslida, har biri 0 dan 9 gacha raqamlardan birining tasvirini ifoda etuvchi 28×28 o'lchamdagi (shunda 28×28 matritsani bir o'lchamli vektorga aylantirsak, o'lchami 784 bo'lgan vektor hosil bo'ladi) matritsadan iborat. Umuman olganda, kiruvchi to'plam 60000 ta mashq hamda 10000 ta test elementlaridan iborat. Kiruvchi matritsa elementlari $[0,1]$ oraliqda joylashgani sababli kirish sayqallanishi jarayoniga hojat yo'q.

Neyron tarmoqqa kirishda qulay bo'lishi uchun mashq to'plamining shaklini 784×60000 va test to'plamini 784×10000 ko'rinishga keltirib oldik.

Nishonlar to'g'risida so'z yuritsak. Ular qanday ko'rinishda ekanini ko'rish uchun ularning ba'zi elementlarini chop etaylik.

```
y_train[:, :10]
```

```
array([[5],
       [0],
       [4],
       ...,
       [5],
       [6],
       [8]], dtype=int32)
```

Ko'rib turganimizdek, nishon to'plami o'zida $[0,9]$ oraliqda joylashgan butun sonlarni tashkil etadi. Nishon to'plamini qiymati 0 va 1 dan iborat bo'lgan *one-hot* vektor to'plamiga aylantiramiz. Buning uchun bir qancha yordamchi o'zgaruvchilarni aniqlab olaylik. Xususan:

- $digits$ – one-hot vektor elementining uzunligi;
- m_train – mashq to'plami uzunligi;
- m_test – test to'plami uzunligi;

```
digits = 10
m_train = y_train.shape[0] #60000
m_test = y_test.shape[0] #60000
```

Endi mashq hamda test to'plamini *one-hot* vektor massivga aylantiramiz. Bunda *numpy* kutubxonasining *eye* funksiyasidan foydalanamiz.

```
y_train_old = y_train
y_test_old = y_test
y_train = np.eye(digits)[y_train.astype('int32')]
y_train = y_train.T.reshape(digits, m_train)

y_test = np.eye(digits)[y_test.astype('int32')]
y_test = y_test.T.reshape(digits, m_test)
```

Nishonlarni *one-hot* vektor massivlarga aylanganligini tekshirib ko'raylik:

```
print(y_train_old[2,:])
y_train[:,2]
```

```
[4]

array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.])
```

Neyron tarmoqning quyidagi giperparametrlarini aniqlab olaylik:

- *lr* (*learning rate*) – o'rganish darajasi;
- *epochs* – neyron tarmoqning takrorlanish davri;
- *n_x* – kirish qatlami uzunligi;
- *n_h* – yashirin, ya'ni 2-qatlamdagi neyronlar soni;
- *batch_size* – batch uzunligi;
- *batches* – batchlar soni.

```
lr = 0.5 #learning rate
epochs = 50 #number of epochs to train
n_x = 784 #number of inputs
n_h = 64 #number of hidden units
batch_size = 64 #input batch size
batches = m_train // batch_size # number of batches
```

Neyron tarmoqning koeffitsiyentlarini *numpy* kutubxonasidan foydalangan holda inisializatsiya qilaylik, ya'ni boshlang'ich qiymat beraylik:

- *A1* – 2-qatlamning asosiy koeffitsiyentlari massivi;
- *b1* – 2-qatlamning bias koeffitsiyentlari massivi;
- *A2* – 3-qatlamning asosiy koeffitsiyentlari massivi;
- *b2* – 3-qatlamning bias koeffitsiyentlari massivi.

Bias koeffitsiyentlarning ilk qiymati sifatida 0 tanlangan bo'lsa, asosiy koeffitsiyentlarga esa **Xavier** inisializatsiya yo'li yordamida ilk qiymatlar berilmoqda.

```
# initialization
params = {"A1": np.random.randn(n_h, n_x)  np.sqrt(1. / n_x),
          "b1": np.zeros((n_h, 1)),
          "A2": np.random.randn(digits, n_h) np.sqrt(1. / n_h),
          "b2": np.zeros((digits, 1))
         }
```

Shu koeffitsiyentlarning qiymat funksiyasiga ta'sirini ifodalovchi qiymatlarni o'zida saqlovchi o'zgaruvchilarni ham e'lon qilaylik va inisializatsiya qilaylik:

- *dA1* – *A1* koeffitsiyentlarning qiymat funksiyasiga ta'siri;
- *db1* – *b1* koeffitsiyentlarning qiymat funksiyasiga ta'siri;
- *dA2* – *A2* koeffitsiyentlarning qiymat funksiyasiga ta'siri;
- *db2* – *b2* koeffitsiyentlarning qiymat funksiyasiga ta'siri.

```
dA1 = np.zeros(params["A1"].shape)
db1 = np.zeros(params["b1"].shape)
dA2 = np.zeros(params["A2"].shape)
db2 = np.zeros(params["b2"].shape)
```

Koeffitsiyentlarning o'zgaruvchilar bilan ko'paytmasini amalga oshiruvchi funksiyani ifodalaylik. Ushbu funksiya quyidagi formulaga asoslangan:

$Z = a_1 \times c_1 + a_2 \times c_2 + ... + a_n \times c_n + b$

Bu formulada a_i – ma'lum qatlam koeffitsiyentlari, c_i – ushbu qatlamga kiruvchi o'zgaruvchilar va b – bias.

```
def layer_multiplication(A, C, B):
    return np.matmul(A, C) + B
```

Ikkinchi qatlam uchun chiquvchi funksiya, ya'ni sigmoid funksiyasini e'lon qilaylik. Ushbu funksiya quyidagi formulaga asoslangan:

$\sigma(z) = \frac{1}{1+e^{-z}}$

```
def sigmoid(Z):
    s = 1. / (1. + np.exp(−Z))
    return s
```

Oxirgi qatlamning chiqish funksiyasi sifatida ishtirok etuvchi softmax funksiyasini ham Pytonda aniqlab olaylik. Uning formulasi quyidacha:

$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$, $i = [1...K]$ hamda $z = [z_1...,z_K]$ oraliqlarida.

Bunda bizda $K = 10$, ya'ni 10 ta klass mavjud $(0, 1 \ldots 9)$.

```
def softmax(Z):
    return np.exp(Z) / np.sum(np.exp(Z), axis=0)
```

Qiymat funksiyasini ifodalaylik. Bunda biz multiklass muammosi uchun qiymat funksiyasini quyidagi formulaga asosan ifodalaymiz:

$J(a) = -\frac{1}{m} \sum_{i=1}^{m} y_k^{(i)} log(h_a(x^{(i)}))$

```
def compute_loss(Y, Y_hat):
    """
    compute loss function
    """
    L_sum = np.sum(np.multiply(Y, np.log(Y_hat)))
    m = Y.shape[1]
    L = −(1./m) L_sum

    return L
```

Oldinga siljish algoritmini aniqlaylik. Bunda biz quyidagi ketma-ketlikda algoritmni amalga oshirib, oxirgi natijani olamiz:

1. 2-qatlam uchun koeffitsiyentlar va kiruvchi qatlam ko'paytmasi yig'indisini hisoblab olamiz va $cache[\imath Z1"]$ Pyton lug'atiga (*dictionary*) saqlaymiz;

2. 2-qatlamning chiqish funksiyasini (*sigmoid*) hisoblaymiz. Bunda natijani *cache*[ıC1"] lug'atda (*dictionary*) saqlaymiz;

3. 3-qatlam uchun koeffitsiyentlar va kiruvchi qatlam ko'paytmasi yig'indisini hisoblab olamiz va *cache*[ıZ2"] Pyton lug'atiga (*dictionary*) saqlaymiz;

4. 3-qatlamning chiqish funksiyasini (*softmax*) hisoblaymiz. Bunda natijani *cache*[ıC1"] lug'atda (*dictionary*) saqlaymiz.

```
def feed_forward(X, params):
    cache = {}

    # Z1 = A1.dot(x) + b1
    cache["Z1"] = layer_multiplication(params["A1"], X, params["b1"])

    # C1 = sigmoid(Z1)
    cache["C1"] = sigmoid(cache["Z1"])

    # Z2 = A2.dot(A1) + b2
    cache["Z2"] = layer_multiplication(params["A2"],
        cache["C1"], params["b2"])

    # C2 = softmax(Z2)
    cache["C2"] = softmax(cache["Z2"])

    return cache
```

Ortga siljish algoritmini amalga oshiramiz. Ortga siljish algoritmini oxirgi qatlamdan boshlaymiz va qatlamlar bo'ylab teskari harakatlanamiz. Bunda har bir qatlamdagi koeffitsiyentlarning qiymat funksiyasiga ta'sirini aniqlaymiz va kerakli o'zgaruvchilarga saqlaymiz.

```
def back_propagate(X, Y, params, cache, m_batch):
    # error at last layer
    delta_2 = cache["C2"] - Y

    # gradients at last layer (Py2 need 1. to transform to float)
    dA2 = (1. / m_batch) np.matmul(delta_2, cache["C1"].T)
    db2 = (1. / m_batch) np.sum(delta_2, axis=1, keepdims=True)

```

```
    # back propgate through first layer
    delta_1 = np.matmul(params["A2"].T, delta_2) (sigmoid(cache["Z1"])
        (1 − sigmoid(cache["Z1"])))

    # gradients at first layer (Py2 need 1. to transform to float)
    dA1 = (1. / m_batch) np.matmul(delta_1, X.T)
    db1 = (1. / m_batch) np.sum(delta_1, axis=1, keepdims=True)

    grads = {"dA1": dA1, "db1": db1, "dA2": dA2, "db2": db2}

    return grads
```

Neyron tarmoqni amalga oshirish uchun barcha funksiyalarni aniqlab bo'ldik. Endi bevosita neyron tarmoqdagi hisob-kitoblarni boshlasak bo'ladi. E'tibor bering, bizda ikki sikl ishtirok etmoqda, ya'ni birinchi sikl qadamlarni ifoda etsa, ikkinchi sikl esa mashq to'plamining mayda bo'laklarga bo'lingan guruhlarni o'z ichiga oluvchi *batches*ni ifodalaydi. Har bir qadamda test va mashq to'plamlaridagi xatoliklar hisoblab boriladi va mos ravishda *test_costs* va *train_costs* massivlarida saqlanadi.

```
test_costs = []
train_costs = []
# training
for i in range(epochs):

    # shuffle training set
    permutation = np.random.permutation(x_train.shape[1])
    X_train_shuffled = x_train[:, permutation]
    Y_train_shuffled = y_train[:, permutation]

    for j in range(batches):

        # get mini−batch
        begin = j  batch_size
        end = min(begin + batch_size, x_train.shape[1] − 1)
        X = X_train_shuffled[:, begin:end]
        Y = Y_train_shuffled[:, begin:end]
        m_batch = end − begin

        # forward and backward
        cache = feed_forward(X, params)
        grads = back_propagate(X, Y, params, cache, m_batch)
```

```
23
24          dA1 = grads["dA1"]
25          db1 = grads["db1"]
26          dA2 = grads["dA2"]
27          db2 = grads["db2"]
28
29          # gradient descent
30          params["A1"] = params["A1"] - lr dA1
31          params["b1"] = params["b1"] - lr db1
32          params["A2"] = params["A2"] - lr dA2
33          params["b2"] = params["b2"] - lr db2
34
35      # forward pass on training set
36      cache = feed_forward(x_train, params)
37      train_loss = compute_loss(y_train, cache["C2"])
38      train_costs.append(train_loss)
39
40      # forward pass on test set
41      cache = feed_forward(x_test, params)
42      test_loss = compute_loss(y_test, cache["C2"])
43      test_costs.append(test_loss)
44      if i % 10 == 0 or i == epochs -1:
45          print("Epoch {}: training loss = {}, test loss = {}".format(
46              i + 1, train_loss, test_loss))
```

```
Epoch 1: training loss = 0.25363514063736736, test loss = 0.25081073811714294
Epoch 11: training loss = 0.05886456975993621, test loss = 0.08787725606448088
Epoch 21: training loss = 0.03170710019303508, test loss = 0.07914492305547167
Epoch 31: training loss = 0.018595611484698654, test loss = 0.07900983113358759
Epoch 41: training loss = 0.012576158887516923, test loss = 0.08000798691994716
Epoch 50: training loss = 0.009289235194238864, test loss = 0.08380877467053
```

Gradient tushishda qiymat funksiyasining qay darajada o'zgarganini vizual ko'rish uchun grafik *plot_cost* funksiyasini yarataylik, ushbu funksiya *matplotlib* kutubxonasidan foydalanib grafika chizadi.

```
1 def plot_cost(train_costs,test_costs, epochs):
2     plt.xlabel('Epochs')
3     plt.ylabel('Cost')
4     plt.plot(epochs, train_costs, 'm', linewidth = "1",
5         color='r', label='mashqdagi xatolik')
6     plt.plot(epochs, test_costs, 'm', linewidth = "1",
7         color='g', label='testdagi xatolik')
8     plt.legend(loc="upper right")
```

```
9    plt.show()
```

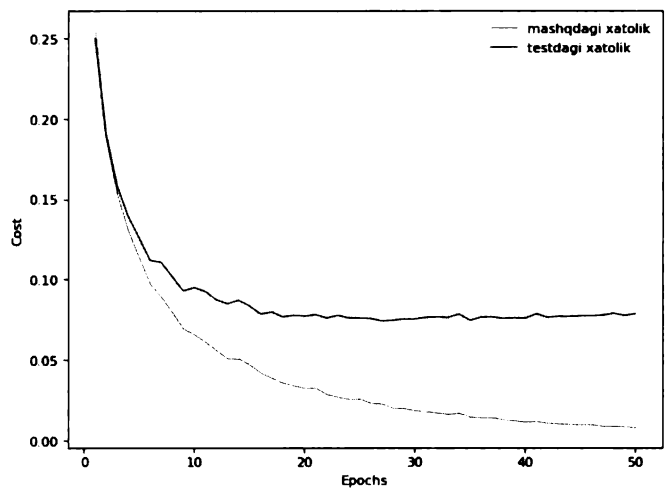
plot_cost funksiyasi parametr sifatida 3 ta massiv o'zgaruvchisini qabul qiladi va bular:

1. Mashq to'plami uchun qiymat funksiyasining natijasi ifodalangan massiv, *train_costs* o'zgaruvchisi;

2. Test to'plami uchun qiymat funksiyasining natijasi ifodalangan massiv, *test_costs* o'zgaruvchisi;

3. Qadamlarni ifoda etuvchi massiv, *epochs* o'zgaruvchisi.

```
1 n_epochs = np.arange(1, epochs + 1)
2 figure(figsize=(8, 6), dpi=80)
3 plot_cost(train_costs, test_costs, n_epochs)
```



Yuqoridagi diagrammadan ko'rib turibmizki, davrning har qadamida mashq va test to'plamlarda qiymat funksiyasi tushib boryapti. Shuni ta'kidlab o'tish kerakki, davrning, taxminan, 20-qadamidan keyin test to'plamida qiymat funksiyasi tushmayapti va hatto ko'rishimiz mumkinki, biroz bo'lsa-da ko'tarilmoqda ham. Bu shundan dalolat beryaptiki, bizning neyron tarmog'imiz 20-qadamdayoq o'zining optimal parametrlariga ega bo'lgan. Undan keyingi qadamlar, asosan, neyron tarmoq mashq to'plamida yaxshi natija berishi yoki, aksincha, test to'plamida xatoliklar ko'payishidan dalolat beradi.

Neyron tarmoqni endi test to'plamida sinab ko'raylik:

```
test_predictions = feed_forward(x_test, params)["C2"]
```

Neyron tarmoq test to'plamiga tatbiq qilindi va so'nggi chiqish funksiyasining natijasi *one-hot* vektor bo'lib, ularni $[0,9]$ oraqlidagi sonlarga aylantiraylik:

```
test_predictions_label = np.argmax(test_predictions, axis=0)
test_predictions_label
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

Model aniqligi (Model accuracy)

Biz yaratgan neyron tarmoq klassifikatsiyalash muammosini yechgani uchun ushbu modelning aniqligini o'lchashda *Accuracy* va *F1-Score* o'lchovlaridan foydalanishimiz mumkin. Logistik regressiya algoritmining amaliy mashg'ulotida ushbu o'lchovlarning binar-klassifikatsiyaga (ya'ni klasslar soni 2 ta) tatbiqini ko'rib o'tgan edik. Bizning hozirgi mashg'ulotimiz esa ko'p klassli muammoga bag'ishlangan, ya'ni klasslar soni 10 ta (0 dan 9 gacha butun sonlar) bo'lganlarga. Ko'p klassli modelga *Accuracy* va *F1-Score* o'lchovlarini tatbiq etishda chalkashlik matritsasini aniqlashimiz kerak. Ushbu matritsani aniqlaydigan funksiya yarataylik.

```
#from this answer: https://stackoverflow.com/a/53387268/4361020
def confusion_matrix(actual, predicted):
    classes = np.unique(np.concatenate((actual,predicted)))
    confusion_mtx = np.empty((len(classes),len(classes)),dtype=np.int)
    for i,a in enumerate(classes):
        for j,p in enumerate(classes):
            confusion_mtx[i,j] = np.where((actual==a)(predicted==p))[0].
                shape[0]
    return confusion_mtx
```

Ushbu funksiyani ishga tushiramiz va chalkashlik matritsasini olamiz.

```
cf = confusion_matrix(y_test_old_, test_predictions_label)
```

Ushbu chalkashlik matritsasini chop etaylik:

```
for i in range(10):
    #printing labels
    if i ==0:
```

```
        print('\t', end='')
        for j in range(10):
            print('b:', j, end='\t')
        print('\n')

    print('t:', i, end='\t')
    for j in range(10):
        print(cf[i][j], end='\t')
    print('\n')
```

	b: 0	b: 1	b: 2	b: 3	b: 4	b: 5	b: 6	b: 7	b: 8	b: 9
t: 0	968	0	2	0	0	1	5	1	2	1
t: 1	0	1126	1	1	0	1	2	1	3	0
t: 2	4	2	1001	5	3	1	3	9	4	0
t: 3	0	0	4	984	0	6	0	4	5	7
t: 4	2	0	1	1	955	1	4	2	1	15
t: 5	2	1	0	11	0	863	6	1	5	3
t: 6	4	2	1	1	4	6	939	0	1	0
t: 7	0	3	7	4	1	0	0	1002	1	10
t: 8	4	1	3	4	7	2	3	4	945	1
t: 9	2	3	0	5	8	3	0	8	0	980

Yuqoridagi matritsani tavsiflasak. Diagonal ravishda o'qlar qiymati bir xil bo'lgan (masalan, $(t:0)-(b:0)$, $(t:1)-(b:1)$ uyacha) uyachalar model tomonidan to'g'ri topilgan klasslarning klass qiymati bo'yicha kesishmasini anglatadi. Gorizontal o'q esa test to'plamidagi haqiqiy klasslarning model tomonidan bashorat qilingan klasslar soni kesishmasidir. Ya'ni $(t:0)$ qatorni olaylik. Ushbu qator test to'plamdagi haqiqiy qiymati 0 bo'lgan klassning model tomonidan qanday bashorat qilinganini anglatadi. Xususan, $(t:0)-(b:0)$ uyachaning qiymati 968, ya'ni 968 ta 0 klass model tomonidan to'g'ri topilgan. $(t:0)-(b:5)$ uyachaning qiymati esa 1. Bu degani test to'plamdagi haqiqiy qiymati 0 bo'lgan klass model tomonidan bir marta 5 deya noto'g'ri topilgan.

Biz ushbu chalkashlik matritsasidan foydalanib, har bir klass uchun *F1-Score*ni aniqlashimiz mumkin va quyidagi funksiyani yarataylik:

```
def calculate_f1_score_accuracy(cf, n):
    actual_sum = np.sum(cf, axis=1)
    predicted_sum = np.sum(cf, axis=0)
    print('raqam \t precision \t recall \t f1-score haqiqiy_klasslar soni')
```

```
    total_correct_sum = 0
    for i in range(10):
        correct_sum = cf[i][i]
        total_correct_sum += correct_sum
        precision = correct_sum / predicted_sum[i]
        recall = correct_sum / actual_sum[i]
        f1_score = 2 precision recall / (precision + recall)
        print(i, '\t', round(precision, 5), '\t', round(recall, 5),'\t',
            round(f1_score, 5),'\t', actual_sum[i])
    total_accuracy = total_correct_sum / n
    print('Accuracy: ', total_accuracy)
```

Ushbu funksiyadan foydalanib, modelning aniqligini o'lchaylik:

```
calculate_f1_score_accuracy(cf, m_test)
```

raqam	precision	recall	f1-score	haqiqiy_klasslar soni
0	0.98174	0.98776	0.98474	980
1	0.98946	0.99207	0.99076	1135
2	0.98137	0.96996	0.97563	1032
3	0.9685	0.97426	0.97137	1010
4	0.97648	0.97251	0.97449	982
5	0.97624	0.96749	0.97185	892
6	0.97609	0.98017	0.97812	958
7	0.97093	0.97471	0.97282	1028
8	0.97725	0.97023	0.97372	974
9	0.96362	0.97126	0.96742	1009

Accuracy: 0.9763

Yuqoridagi jadval har bir klass uchun *precision*, *recall*, *F1-Score*, test to'plamidagi haqiqiy klasslar sonini va modelning *accuracy* o'lchovi uchun qiymatlarni chop etadi. Ko'rib turibmizki, har bir klass uchun *F1-Score* ancha yuqori. Hamda umumiy aniqlik, ya'ni *accuracy* juda yuqori. Bu degani test to'plamdagi 97% element to'g'ri topilgan.

6 Qaror daraxti

Qaror daraxti mashinaviy o'qitishda eng keng tarqalgan usullardan bo'lib, qoidalar asosida prognoz qiladi. [5] Ushbu qoidalar, albatta, berilgan mashq to'plami asosida yaratiladi. Ushbu usulning qanday ishlashini tushunish uchun inson vazni me'yordan ortiq yoki yo'q ekanligi haqidagi ma'lumotlar to'plamidan foydalanaylik: [8]

7-jadvaldan ko'rib turibmizki, ma'lumotlar to'plami 4 ta ustundan iborat:

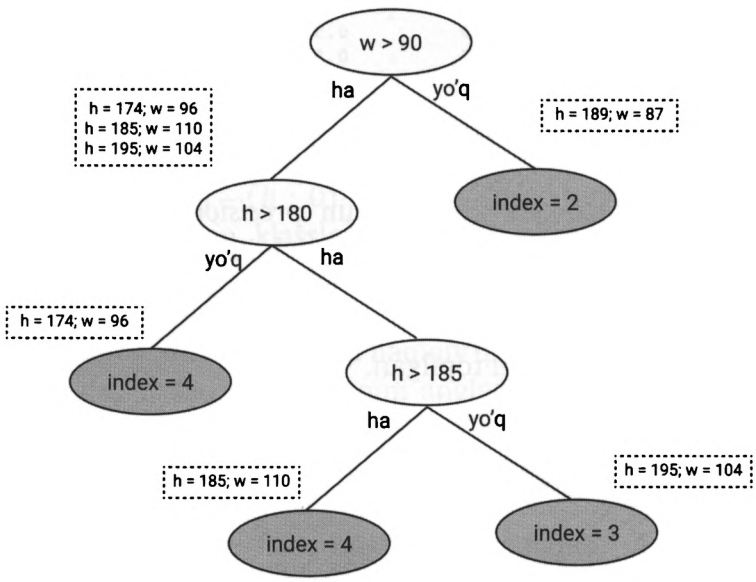
- *Gender* – inson jinsi. Bunda *Male* – erkak, *Female* – ayol;

Gender	Height	Weight	Index
Male	174	96	4
Male	189	87	2
Female	185	110	4
Female	195	104	3

Jadval 7: Inson vazni ma'lumotlar to'plami

- *Height* – inson bo'yi (cm)
- *Weight* – inson vazni (kg)
- *Index* – vazn ortiqchaligi indeksi (prognoz qilinishi kerak bo'lgan qiymatlar). Bunda 0 – juda ozg'in, 1 – ozg'in, 2 – normal, 3 – ortiqcha vazn, 4 – katta ortiqcha vazn, 5 – juda katta ortiqcha vazn.

Ushbu ma'lumotlar to'plamiga qaror daraxti algoritmini tatbiq qilganimizda 27-rasmda ko'rsatilgan tarzdagi yechimni olishimiz mumkin.



Rasm 27: Qaror daraxti

27-rasmdagi yechim xuddi daraxtga o'xshab, faqat teskarisiga o'sib borayapti, shuning uchun ham ushbu algoritm qaror daraxti deb ataladi (ushbu daraxt elementlarida ifodalangan *w* – *weight*, ya'ni inson vazni va *h* –

height, ya'ni inson bo'yi). Ushbu qaror daraxti algoritmidagi eng yuqoridagi element – ildiz element deb atalib, o'zida eng birinchi qoidani saqlaydi, ya'ni $w > 90$. Bu degani ushbu element 7-jadvaldagi elementlarni ikki to'plamga ajratadi. Bunda birinchi to'plamda o'zida vazni $90\,kg$ dan ortiq bo'lgan insonlar ro'yxati jamlangan bo'lsa, ikkinchi to'plamda vazni $90\,kg$ ga teng va undan kichik bo'lgan insonlar ro'yxati jamlangan. Qaror daraxti ko'rsatib turibdiki, vazni $90\,kg$ ga teng va undan kichik bo'lgan insonlar ro'yxatida faqatgina 1 yozuv mavjud ($h = 189, w = 87$) va ushbu yozuvning indeksi 2 ga teng. $index = 2$ deya ko'rsatayotgan element **daraxtning barg elementi** deyiladi va undan keyin daraxt o'sishdan to'xtaydi. Shu kabi daraxt grafikasida aks ettirilgan boshqa yashil rangdagi elementlar ham daraxt barglari, deya atalib, qoidalarning ma'lumotlar to'plamiga tatbiq qilinishidan olingan yozuvlarni o'zida jamlaydi. Endi bevosita ildiz elementga qaytsak. $w > 90$ qoidadan qaror daraxti shunday xulosaga keladiki, vazni $90\,kg$ ga teng va undan kichik bo'lgan insonlar indeksi 2 ga teng (normal). Albatta, real hayotda vazni 90 va undan kichik insonlar o'rtasida ham ortiqcha vazn kuzatiladi, ya'ni bo'yi $180\,cm$ dan past insonlarda. Bunday holatlarni qamrab olish uchun, albatta, ma'lumotlar to'plami turli xil kombinatsiyadagi yozuvlarni o'z ichiga olishi kerak. Biz esa tushunib olish uchun 7-jadvalda keltirilgan kamroq hajmdagi ma'lumotlar to'plamidan foydalanamiz.

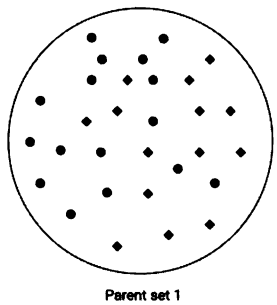
Endi e'tiborimizni $w > 90$ qoidadan qanoatlantirgan birinchi to'plamga qarataylik. Ushbu to'plam o'zida 3 yozuvni jamlaydi va ularni yanada aniqlashtirish uchun keyingi element – $h > 180$ qoidani o'zida aks ettirgan shox elementi tatbiq qilinadi. Ya'ni $h > 180$ qoidasi tatbiq qilinganidan so'ng to'plam yana ikki to'plamga bo'ladi. Bunda birinchi to'plam bo'yi $180\,cm$ dan katta insonlar guruhini o'zida mujassamlashtirsa, ikkinchi to'plam esa o'zida $180\,cm$ va undan kichik insonlar guruhini o'zida saqlaydi. Shu tariqa daraxt qoidalar asosida o'sib boradi va kerakli qoidalarni yaratib boradi. Albatta, shu o'rinda tabiiy savol tug'iladi: Qanday qilib ushbu qoidalar yaratiladi? Ya'ni nega biz aynan to'plamni vazni $90\,kg$ dan oshiq va undan kichik ($w > 90$) guruhlarga bo'lib olishdan boshladik? Ushbu savollarga javob berishdan oldin bir qator tushunchalar bilan tanishib olaylik.

6.1 Ma'lumotlar to'plami entropiyasi

Ma'lumotlar to'plami entropiyasi – berilgan to'plamdagi ma'lumotlarning o'rtacha stabilligi (barqarorligi)ni aniqlovchi o'lchov hisoblanadi. Ya'ni "Berilgan to'plamdagi ma'lumot turlari qay darajada ko'p?" degan savolga javob beradi. Ushbu daraja quyidagi formula bilan aniqlanadi:

$$Entropy = \sum_{i}^{n} -p_i log_2 p_i \tag{84}$$

84-formulada p_i – to‘plamdagi i-klassning ehtimolligi (boshqacha aytganda, i-klassdagi elementlar sonining to‘plamdagi barcha elementlar soniga bo‘linmasi). Ushbu formulaning qiymati $[0,1]$ oraliqda bo‘lib, qiymat 0 ga yaqinlashgan sari to‘plam shuncha stabil yoki bir klass dominant bo‘ladi. 0 qiymatida esa to‘plamda faqat bir turdagi elementlar joylashgan bo‘ladi. Entropiya 1 ga yaqinlashgan sari to‘plamning stabilligi shuncha past bo‘ladi va 1 qiymatda hech bir klass elementlari to‘plamda dominant bo‘lmaydi. Ushbu formulani yanada chuqurroq tushunish uchun 28-rasmda ifoda etilgan to‘plamga e’tibor beraylik [4]:



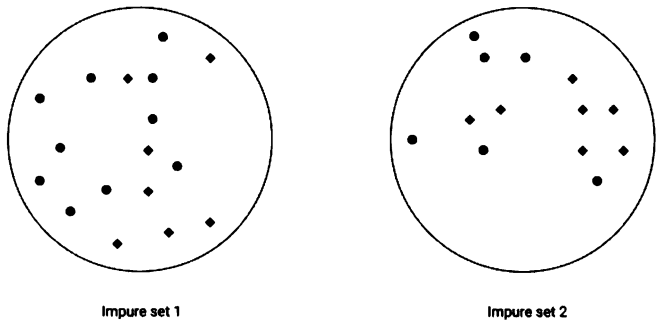
Rasm 28: To‘plam

28-rasmda ifoda etilgan to‘plamda 14 ta qizil rangdagi va 16 ta ko‘k rangdagi elementlar mavjud. Ushbu to‘plamning entropiyasini 84-formula orqali topaylik:

$$E_p = -(\frac{14}{30} \cdot log_2 \frac{14}{30}) - (\frac{16}{30} \cdot log_2 \frac{16}{30}) = 0.996 \tag{85}$$

85-formuladan ko‘rib turibmizki, to‘plamning entropiyasi deyarli 1 ga teng. Bu degani to‘plam stabil emas, ya’ni ma’lum bir klass dominantlik qilmaydi. Endi keyingi vazifani o‘z oldimizga qo‘yaylik, ya’ni ushbu to‘plamni stabillashtirishimiz kerak. Buning uchun to‘plamni ikki stabil to‘plamga bo‘lishga harakat qilamiz.

Tasavvur qilaylik, 29-rasm ushbu bo‘linma to‘plamlarini o‘zida ifoda etadi va bizning maqsadimiz ushbu bo‘linma qay darajada yaxshi ado etilganligini tekshirishdir. Rasmdan ko‘rib turibmizki, ikkala to‘plam ham stabil emas. Har bir to‘plamda har ikki klassdan (qizil va ko‘k) elementlar mavjud. Lekin ushbu



Rasm 29: Bo‘linma to‘plamlar (1)

bo‘lishni yanada aniqroq tekshirish uchun quyidagi formuladan foydalanishimiz mumkin:

$$E_U = E_p - (\frac{n_{b1}}{n_U} \cdot E_{b1} + \frac{n_{b2}}{n_U} \cdot E_{b2} + \ldots + \frac{n_{bn}}{n_U} \cdot E_{bn}) \tag{86}$$

86-formulada E_p – bo‘linuvchi, ya’ni barcha elementlarni o‘zida jamlovchi umumiy to‘plam (xususan, 28-rasmdagi to‘plam) entropiyasi; $E_{b1}, E_{b2}, \ldots, E_{bn}$ – bo‘linuvchi to‘plamni n bo‘lakka bo‘lishdan hosil bo‘lgan bo‘linma to‘plamlar (xususan, 29-rasmdagi to‘plamlar); n_U – umumiy to‘plamdagi elementlar soni, $n_{b1}, n_{b2}, \ldots n_{bn}$ – ma’lum to‘plamdagi elementlar soni; E_U – asosiy bo‘linuvchi to‘plamning naqadar to‘g‘ri bo‘linganini aniqlovchi entropiya (Ya’ni bo‘linma to‘planmalarining har birida bir xil klassdagi elementlarning to‘planishi). $E_U$ning qiymati $[0, 1]$ oraliqda yotib, 1 ga qancha yaqinlashgan sari, bo‘linma shuncha toza, faqat ma’lum klasslardan tashkil topgan to‘plamlardan iborat bo‘ladi. Aksincha, $E_U$ning qiymati 0 ga yaqinlashgan sari, shuncha shovqin ortadi, ya’ni to‘plamlarda hech qaysi klass dominant bo‘lmaydi.

86-formulani ba’zi hollarda bo‘linuvchi, ya’ni umumiy to‘plam entropiyasidan (E_p) foydalanmay ham aniqlasa bo‘ladi.

$$E_U = \frac{n_{b1}}{n_U} \cdot E_{b1} + \frac{n_{b2}}{n_U} \cdot E_{b2} + \ldots + \frac{n_{bn}}{n_U} \cdot E_{bn} \tag{87}$$

87-formuladagi ifodada umumiy bo‘linma entropiyasidan (E_p) foydalanilmayapti. Ushbu formula natijasi faqat bo‘linma to‘plamlardagi entropiya haqida ma’lumot beradi va bu ba’zi hollarda (masalan, sikl yoki rekursiya orqali barcha to‘plamni klasslar bo‘yicha bo‘lishning kombinatsiyalarini ko‘rib chiqqanda umumiy to‘plam entropiyasi oldingi

siklda yoki rekursiyada allaqachon hisoblangan bo'ladi) ko'proq natijaviy hisoblanadi.

Bo'linma to'plamlar qay darajada yaxshi bo'linganini o'lchovchi metrikani aniqlab oldik. Endi 86-formulani 29-rasmda ifodalangan bo'linma to'plamlarga tatbiq qilaylik. Shu o'rinda ushbu bo'linma to'plamlar to'g'risida biroz ma'lumot berib o'tsak. *Impure set 1* deya nomlangan chap tarafda joylashgan to'plamda 7 ta qizil va 10 ta ko'k elementlar mavjud. *Impure set 2* deya nomlangan o'ng tarafda joylashgan to'plamda esa 7 ta qizil va 6 ta ko'k elementlar mavjud. Avvaliga har bir bo'linma to'plamning entropiyasini aniqlab olamiz:

$$E_{b1} = -(\frac{7}{17} \cdot log_2 \frac{7}{17}) - (\frac{10}{17} \cdot log_2 \frac{10}{17}) = 0.977 \tag{88}$$

$$E_{b2} = -(\frac{7}{13} \cdot log_2 \frac{7}{13}) - (\frac{6}{13} \cdot log_2 \frac{6}{13}) = 0.995 \tag{89}$$

88-89-formulalarda bo'linma to'planmalardagi stabillik qanday hisoblanganini ifoda etadi. Aniq o'lchovlar ham shuni ko'rsatib turibdiki, ushbu to'plamlarda shovqin juda katta, ya'ni biror klass dominant emas. Endi ushbu to'plamlarning qanday bo'linganligini aniqlaylik, ya'ni 86-formulani tatbiq qilaylik:

$$E_{U1} = 0.996 - (\frac{17}{30} \cdot 0.977 + \frac{13}{30} \cdot 0.995) = 0.011 \tag{90}$$

90-formuladagi hisob-kitoblar natijasi 0 ga juda yaqin va bu shuni ko'rsatyaptiki, 29-rasmda ko'rsatilgan bo'linma katta shovqinga ega, ya'ni to'plamlarda hech qaysi klass dominant emas.

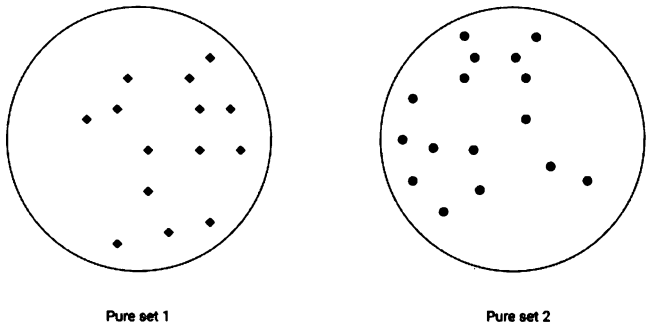
Endi, 28-rasmda ifodalangan to'plamning ideal tarzda bo'linishini ko'rib chiqaylik.

30-rasmda ideal tarzda bo'lingan to'plamlar ko'rsatilgan, ya'ni *Pure set 1* deya nomlangan to'plamda faqat qizil rangdagi 14 ta element mavjud, *Pure set 2* to'plamda esa 16 ta ko'k element bor. Albatta, vizual tarzda buni payqash oson, ammo buni isbotlash uchun entropiya formulalarini tatbiq qilaylik.

$$E_{b1} = -(\frac{14}{14} \cdot log_2 \frac{14}{14}) = 0 \tag{91}$$

$$E_{b2} = -(\frac{16}{16} \cdot log_2 \frac{16}{16}) = 0 \tag{92}$$

Yuqoridagi ifodalardan ko'rib turibmizki, 30-rasmda ifodalangan to'plamlar haqiqatan ham klasslar bo'yicha toza bo'lingan, ya'ni har bir to'plamda ma'lum bir klass dominant. Endi umumiy bo'linmaning entropiyasini aniqlaylik:



Rasm 30: Bo'linma to'plamlar 2

$$E_{U2} = 0.996 - (\frac{14}{30} \cdot 0 + \frac{16}{30} \cdot 0) = 0.996 \tag{93}$$

93-ifodaning qiymati 1 ga juda yaqin. Bu shuni ko'rsatadiki, 30-rasmda ifodalangan to'plamlar bo'linmasi haqiqatan deyarli ideal tarzda amalga oshirilgan.

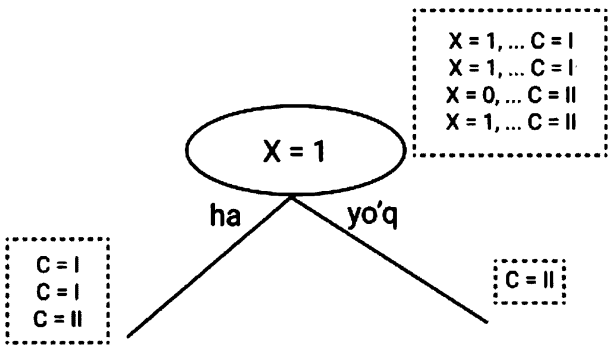
6.2 Ma'lumotlar entropiyasining Qaror daraxtiga tatbiqi

Ma'lumotlar to'plami entropiyasi bo'limidan oldin 27-rasmda ifodalangan Qaror daraxtining "Nega aynan $w > 90$ (vazni 90 kilogramdan katta insonlar) degan qoidani tanladik?" degan savolni o'rtaga qo'ygandik. Ushbu savolga javob esa oddiy, ya'ni Qaror daraxti har bir ustun bo'ylab turli xil qoidalarni sinab ko'radi va eng maqbulini tanlaydi. Eng maqbuli esa o'z-o'zidan qoidalar asosida yaratilgan bo'linma to'plamlar orasidan ma'lumotlar entropiyasini baholash orqali tanlab olinadi. Ushbu jarayonni osonroq tushunish uchun 8-jadvalda ko'rsatilgan oddiygina ma'lumotlar to'plamini ko'rib chiqaylik:

X	Y	Z	C
1	1	1	I
1	1	0	I
0	0	1	II
1	0	0	II

Jadval 8: Oddiy ma'lumotlar to'plami

8-jadvaldagi ma'lumotlar to'plamida C ustunida klasslar joylashgan va qolgan ustunlar orqali Qaror daraxti shartli qoidalar yaratadi hamda ma'lumotlarni quyi to'plamlarga bo'lib boradi. Natijada shu orqali ma'lumotlar to'plamini klasslarga ajratadi. Yuqorida ta'kidlab o'tganimizdek, barcha ustunlar asosidagi qoidalar bo'yicha to'plamlarga bo'linib, ularning eng yaxshi bo'linmasi tanlanadi, ya'ni ma'lumotlar entropiyasi baholanadi. Ushbu jarayonni 8-jadvali uchun X ustunidan boshlaylik.



Rasm 31: X ustun bo'yicha bo'linma

31-rasmda X ustuni asosida yaratilgan $X = 1$ qoida bo'yicha bo'linma to'plamlar aks ettirilgan. Keling, ularning entropiyasini baholaylik. Bunda, avvalo, asosiy to'plamning (Ya'ni 8-jadvaldagi to'plam) entropiyasini aniqlaymiz:

$$E_p = -(\frac{2}{4} \cdot log_2 \frac{2}{4}) - (\frac{2}{4} \cdot log_2 \frac{2}{4}) = 1 \tag{94}$$

Keyingi qadamda 31-rasmda ifodalangan bo'linma to'planmalarning entropiyasini aniqlaymiz:

$$E_{b1X} = -(\frac{2}{3} \cdot log_2 \frac{2}{3}) - (\frac{1}{3} \cdot log_2 \frac{1}{3}) = 0.918 \tag{95}$$

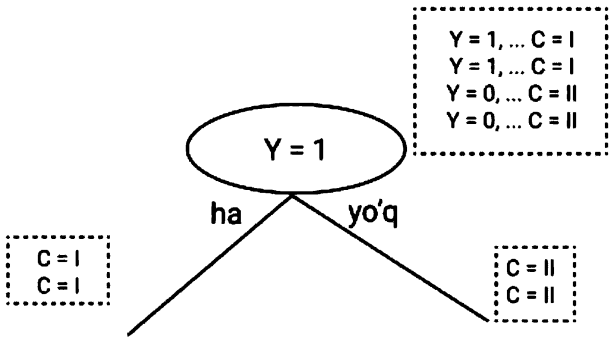
$$E_{b2X} = -(\frac{1}{1} \cdot log_2 \frac{1}{1}) = 0 \tag{96}$$

Yuqoridagi ifodalarda E_{b1X} qiymati 31-rasmdagi quyi chap to'plamning entropiyasini aks ettirsa, E_{b2X} ifoda esa o'ng ifodani aks ettiradi.

Bizda 31-rasmdagi bo'linma qay darajada effektivligini aniqlovchi umumiy entropiya uchun barcha qiymatlar mavjud va ushbu o'lchovni hisoblashga kirishamiz:

$$E_{UX} = 1 - (\frac{3}{4} \cdot 0.918 + \frac{1}{4} \cdot 0) = 0.311 \quad (97)$$

Demak, X ustuni bo'yicha bo'linmaning umumiy entropiyasini ifodalovchi $E_{UX} = 0.311$ ekan. Keyingi ustunlarga, xususan, Y ustuni bo'yicha to'plamni bo'lishga kirishamiz:



Rasm 32: Y ustun bo'yicha bo'linma

Xususan, 32-rasmda Y ustuni bo'yicha yaratilgan $Y = 1$ qoida asosidagi bo'linma tasvirlangan. Ushbu bo'linma to'plamlarning ham entropiyasini hisoblaymiz:

$$E_{b1Y} = -(\frac{2}{2} \cdot log_2 \frac{2}{2}) = 0 \quad (98)$$

$$E_{b2Y} = -(\frac{2}{2} \cdot log_2 \frac{2}{2}) = 0 \quad (99)$$

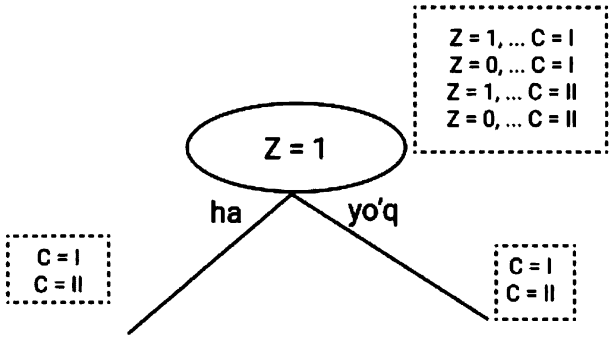
32-rasmdagi bo'linma uchun umumiy entropiyani hisoblaymiz:

$$E_{UY} = 1 - (\frac{2}{4} \cdot 0 + \frac{2}{4} \cdot 0) = 1 \quad (100)$$

Y ustuni bo'yicha umumiy entropiya 1 ga teng ekan, ya'ni $E_{UY} = 1$. Bu degani hozircha eng mukammal bo'linma, sababi har birinchi bo'linmada faqat I-klassdagi elementlar va ikkinchi bo'linmada II-klassdagi elementlar joylashgan. Buni vizual tarzda 32-rasmdagi to'plamlardan ham, 100-ifodada aks etgan bo'linmaning umumiy entropiyasidan ham ko'rishimiz mumkin.

Oxirgi ustun Z bo'yicha ham bo'linmani amalga oshiraylik.

Xususan, 33-rasmda Z ustuni asosidagi $Z = 1$ qoidasi bo'yicha bo'linmalar tasvirlangan. Ushbu quyi to'plamlarning entropiyasini ham hisoblaylik.



Rasm 33: Z ustun bo'yicha bo'linma

$$E_{b1Z} = -(\frac{1}{2} \cdot log_2 \frac{1}{2}) - (\frac{1}{2} \cdot log_2 \frac{1}{2}) = 1 \tag{101}$$

$$E_{b2Z} = -(\frac{1}{2} \cdot log_2 \frac{1}{2}) - (\frac{1}{2} \cdot log_2 \frac{1}{2}) = 1 \tag{102}$$

101-102-ifodalarda hisoblangan entropiyalar qiymati 1 ga teng. Bu esa ushbu to'plamlarda shovqin maksimal darajada, ya'ni bo'linma juda noeffiktiv tarzda amalga oshirilganini anglatadi. Endi ushbu bo'linmaning umumiy entropiyasini hisoblaymiz:

$$E_{UZ} = 1 - (\frac{2}{4} \cdot 1 + \frac{2}{4} \cdot 1) = 0 \tag{103}$$

103-ifodadan ko'rib turibmizki, Z ustuni bo'yicha amalga oshirilgan bo'linma eng nomaqbulidir. Buni $E_{UZ} = 0$ ifoda ham ko'rsatib turibdi.

X, Y, Z ustunlari bo'yicha amalga oshirilgan bo'linmalar umumiy entropiyasidan shunday xulosa qilish mumkinki, umumiy entropiyasi $E_{UY} = 1$ bo'lgan Y ustun orqali amalga oshirilgan bo'linma eng maqbul hisoblanadi. Demak, Qaror daraxti ushbu ustun bo'yicha qoidalar yaratib, to'plamni bo'lib oladi.

Albatta, biz juda sodda bo'lgan ma'lumotlar to'plamini ko'rib chiqdik. Bu to'plamdagi ustunlarda faqatgina 2 xil qiymat bor edi $(0, 1)$. Real hayotda esa to'plamlar murakkab qiymatlarga ega va Qaror daraxti ustunning deyarli boshqa qiymatlari asosida ham qoidalar, yanada ko'proq to'plamlar yaratishi mumkin. Albatta, bunda yakunda umumiy entropiyasi 1 ga eng yaqin bo'lgan qoida tanlanadi va to'plam shu qoida asosida bo'linadi. Biz qoida tanlash amalini Qaror daraxtining ildiz, ya'ni eng tepa elementiga tatbiqini ko'rib chiqdik.

Ushbu ildizdan quyi qatlamda ham xuddi shunday qoida tanlash amali orqali eng maqbul qoidalar tanlanadi va to'plamlar yanada quyi to'plamlarga bo'linadi.

Bunda tabiiy savol tug'ilishi mumkin: "Agarda ma'lumotlar to'plami juda katta bo'lsa, qanday qilib Qaror daraxtining juda chuqur o'sishi, quyi pog'onalarning kengayishi hisobiga vujudga keladigan bir qator muammolar, xususan, Qaror daraxti algoritmining mashq to'plamiga bog'lanib qolishi (*overfitting*) va uning vaqt bo'yicha effektivligini oldini olish mumkinmi? Bunday vaziyatlarning oldini olish uchun keng tarqalgan Qaror daraxti maksimal chuqurligini ifodalovchi *max_depth* giperparametri sozlanadi. Ya'ni Qaror daraxti qoidalar asosida o'sib borgan sari chuqurlik *max_depth* giperparametriga tekshiriladi va ushbu giperparametrga teng kelganda to'plamga shu yo'nalish bo'yicha qoidalar tatbiq qilinmay, oxirgi natija sifatida e'lon qilinadi.

6.3 Amaliy mashg'ulot

Ushbu amaliy mashg'ulotning *github.com* saytidagi repozitoriysiga `https://bit.ly/3qRgBsb` ssilkasi orqali yoki quyidagi QR-kodni skaner qilish orqali o'tishingiz mumkin. Aynan shu amaliy mashg'ulotning **.ipynb* fayli uchun `practice_session/Neural_networks_6_3/` papkasini tekshiring.

Ushbu amaliy mashg'ulotda *Jupyter Notebook* dasturi orqali Pyton dasturlash tilida Qaror daraxti muammosini o'rganamiz. *Jupyter Notebook* dasturi orqali ma'lum bir katalogga yangi "*.ipynb*" faylni yaratamiz va quyidagi kodlarni navbatma-navbat yozib boramiz.

Quyidagi Pyton kutubxonalarini yuklab olamiz:

- *sklearn* – Pytonda mashinaviy o'qitish algoritmlari bilan ishlash uchun *Scikit-learn* kutubxonasi;
- *load_iris* – *sklearn* kutubxonasining iris ma'lumotlar to'plami paketini yuklab olamiz;

- *train_test_split* – *sklearn* kutubxonasining ma'lumotlar to'plamini mashq-test to'plamlarga ajratuvchi paketi;
- *pprint* – Pytonda ma'lumotlar strukturasini o'qish uchun qulay chop etadigan kutubxona;
- *numpy* – Pytonda massivlar bilan ishlash uchun;
- *math* – Pytonda matematik amallar bilan ishlash uchun;

Ushbu amaliy mashg'ulotdagi yechim hamda kod quyida keltirilgan havoladagi yechimga asoslangan:

`https://medium.com/@penggongting/implementing-decision-tree-from-scratch-in-python-c732e7c69aea`

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 from pprint import pprint
4 import math
5 import numpy as np
```

Yuqorida ta'kidlaganimizdek, Qaror daraxti muammosini o'rganish uchun *iris* ma'lumotlar to'plamidan foydalanamiz. Ushbu to'plam bilan qulayroq ishlash uchun *sklearn* kutubxonasining *datasets* paketida *load_iris* nomli funksiya bor. Bu funksiya aynan mashinaviy o'qitish algoritmlari uchun moslangan formatdagi *iris* to'plamini taqdim etadi (Ya'ni to'plam *numpy* massiv formatida, mashq va nishonlar alohida massivlarga ajratilgan va h.k.). Ushbu to'plam bilan kengroq tanishish uchun quyidagi havolaga o'ting: `https://scikit-learn.org/stable/auto\_examples/datasets/plot\_iris\_dataset.html`

Avvaliga to'plam entropiyasini hisoblaydigan quyidagi formulani Pytonda ifodalab olamiz:

$Entropy = \sum_i^n -p_i log_2 p_i$

Yuqoridagi formulada n – to'plamdagi barcha elementlar soni, p_i – ma'lum bir i-klassning ehtimolligi bo'lib, quyidagi formula orqali ifodalasak bo'ladi:

$p_i = \frac{c_i}{n}$.

Bunda c_i – ma'lum i-klassdagi elementlar soni.

Biz yaratayotgan Qaror daraxti qoida asosida to'plamni ikkiga bo'linishini hisobga olgan holda entropiya funksiyasini to'plamda 2 ta klass mavjud deb faraz qilamiz (bunda $c1$ – birinchi klassdagi elementlar soni, $c2$ – ikkinchi klassdagi elementlar soni).

```
def entropy_cal(c1, c2):
    """
    Returns entropy of a group of data
    c1: count of one class
    c2: count of another class
    """
    if c1== 0 or c2 == 0: # when there is only one class in the group, entropy is 0
        return 0
    n = c1 + c2
    return −(c11.0/n)math.log(c11.0/n, 2) −(c21.0/n)math.log(c21.0/n, 2)
```

To'plam entropiya funksiyasini aniqlab oldik, ammo ushbu funksiya aniq klasslar sonini talab etadi. Amalda esa ma'lumotlar to'plamidan biz o'zimiz klasslar sonini aniqlab olishimiz kerak bo'ladi. Ushbu funksiyani yaratamiz va klasslar bo'yicha har bir bo'linmaning entropiyasini yuqorida yaratilgan *entropy_cal* funksiyasi yordamida topamiz. Nihoyat, bo'linmaning umumiy entropiyasini aniqlaymiz. Bo'linmaning umumiy entropiyasini aniqlashda quyidagi formuladan foydalanamiz:

$E_U = E_p - (\frac{n_{b1}}{n_U} \cdot E_{b1} + \frac{n_{b2}}{n_U} \cdot E_{b2} + ... + \frac{n_{bn}}{n_U} \cdot E_{bn})$

Yuqoridagi formuladan foydalanib, berilgan to'plam entropiyasini hisoblovchi funksiyani yaratamiz.

```
# get the entropy of one big circle showing above
def entropy_of_one_division(division):
    """
    Returns entropy of a divided group of data
    Data may have multiple classes
    """
    s = 0
    n = len(division)
    classes = set(division)
    for c in classes: # for each class, get entropy
        n_c = sum(division==c)
        e = n_c1.0/n entropy_cal(sum(division==c),
            sum(division!=c)) # weighted avg
        s += e
    return s, n
```

Entropiyani aniqlovchi funksiyalarni yaratib olgandan keyin ushbu funksiyalarni qanday qilib mashq to'plamlariga tatbiq qilish mumkinligini izlaymiz. Ushbu savolga javob izlashda ma'lum bir qadamda Qaror daraxti

qoida asosida to'plamni ikkiga bo'lishini eslab o'taylik, ya'ni ushbu qoidani qanoatlantiradigan to'plam (aytaylik, A to'plam) va qanoatlantirmaydigan to'plam (aytaylik, B to'plam). Bizning maqsad esa ushbu A va B to'plamlardagi klasslar qay darajada taqsimlanganini aniqlashdir. Albatta, bu aniqlash biz yuqorida yaratgan entropiya funksiyalari orqali aniqlanadi. Ya'ni biz ushbu funksiyalarga bo'linma A va B to'plamlari uchun nishonlarni uzatishimiz kerak. Undan oldin esa biz o'zida faqat nishon klass qiymatlarini saqlovchi A va B bo'linma to'plamlarining nishonlarini alohida ajratib olishimiz kerak. Bunda bizga ma'lum bir qoidaning to'plamga tatbiq, ya'ni o'zida $True$ (qanoatlantiradigan) hamda $False$ (qanoatlantirmaydigan) qiymatlarni saqlovchi $y\_predict$ nomli massiv kerak bo'ladi. Bundan tashqari, albatta, biz ustida ishlayotgan to'plamlarning (A, B) nishonlarini o'zida saqlovchi y_real nomli massiv ham talab etiladi. $Numpy$ kutubxonasining funksionalidan foydalangan holda y_real nishonlar massivini $y\_predict$ massivi orqali ikki to'plamga bo'lamiz. Bunda quyidagi ifodadan foydalangan holda A (Ya'ni ma'lum bir qoidani qanoatlantiradigan) to'plam uchun nishon qiymatlarini olsak:

$y\_real[y\_predict]$

Quyidagi ifodadan foydalangan holda B (Ya'ni ma'lum bir qoidani qanoatlantirmaydigan) to'plam uchun nishon qiymatlarini olamiz:

$y\_real[\sim y\_predict]$

Yuqorida keltirilgan fikrlarni amalga oshiruvchi $get\_entropy$ funksiyasini yarataylik:

```
# The whole entropy of two big circles combined
def get_entropy(y_predict, y_real):
    """
    Returns entropy of a split
    y_predict is the split decision, True/Fasle, and y_true can be multi class
    """
    if len(y_predict) != len(y_real):
        print('They have to be the same length')
        return None
    n = len(y_real)
    # left hand side entropy
    s_true, n_true = entropy_of_one_division(y_real[y_predict])
    # right hand side entropy
    s_false, n_false = entropy_of_one_division(y_real[~y_predict])
    # overall entropy, again weighted average
    s = n_true1.0/n s_true + n_false1.0/n s_false
```

```
17      return s
```

Entropiyaga oid kerakli barcha funksiyalarni yaratib oldik. Endi Qaror daraxtini o'zida ifodalovchi $DecisionTreeClassifier$ klassni yaratishga kirishamiz. Albatta, ushbu klass elementlarini birma-bir yaratib olamiz hamda so'ngida ushbu elementlardan iborat bo'lgan Qaror daraxti klassini tuzamiz.

Avvalambor, Qaror daraxtining maksimal chuqurligini belgilab beruvchi yordamchi o'zgaruvchilarni aniqlab olaylik. Eslatib o'tamiz, ushbu maksimal chuqurlikni belgilab olishdan maqsad Qaror daraxtining $overfitting$ muammosini hamda cheksiz chuqurlashib ketishining oldini olishdir. Demak, biz $depth$ – Qaror daraxtining ma'lum qadamdagi chuqurligini o'zida saqlab turuvchi va max_depth – Qaror daraxtining maksimal chuqurligi (Ya'ni Qaror daraxti ushbu darajadan chuqur bo'la olmaydi) qiymatlarini o'zida saqlovchi o'zgaruvchilarni yaratib olamiz.

```
1  class DecisionTreeClassifier(object):
2      def __init__(self, max_depth):
3          self.depth = 0
4          self.max_depth = max_depth
```

$DecisionTreeClassifier$ klassining keyingi funksiyasi $find\_best\_split(self, col, y)$ deb atalib, ushbu funksiya ma'lum bir ustun uchun eng minimal entropiyali to'plamlarni hosil qiluvchi qoidani topadi. Buning uchun esa qatorlar osha yangi sikl yaratilib, shu siklning har bir qadamida ustunning mos qatoridagi qiymat bo'yicha to'plam ikkiga bo'linadi va minimal entropiyaga solishtirib boriladi (Ya'ni ushbu qator qiymati orqali bo'lingan to'plamlar entropiyasi minimal entropiyadan kichik bo'lsa, ushbu qator qiymati minimal entropiya hosil qiluvchi sifatida qabul qilinadi va h.k.). Albatta, sikl so'ngida biz eng optimal qiymatni olamiz.

```
1  def find_best_split(self, col, y):
2      """
3      col: the column we split on
4      y: target var
5      """
6      min_entropy = 10
7      n = len(y)
8      for value in set(col): # iterating through each value in the column
9          y_predict = col < value # separate y into 2 groups
10         my_entropy = get_entropy(y_predict, y) # get entropy of this split
```

```
            if my_entropy <= min_entropy: # check if it's the best one so far
                min_entropy = my_entropy
                cutoff = value
        return min_entropy, cutoff
```

Ma'lum bir ustundagi minimal entropiyani beruvchi qatorni sikl orqali topishni aniqladik. Demak, xuddi shu funksiyani ustunlar bo'ylab tatbiq qilib, eng effektiv ustunni (albatta, shu ustunning minimal entropiya beradigan qatorini ham) aniqlashimiz mumkin. Ushbu amalni quyidagi funksiyada aniqlaylik:

```
def find_best_split_of_all(self, x, y):
    """
    Find the best split from all features
    returns: the column to split on, the cutoff value, and the actual entropy
    """
    col = None
    min_entropy = 1
    cutoff = None
    # iterating through each feature
    for i, c in enumerate(x.T):
        # find the best split of that feature
        entropy, cur_cutoff = self.find_best_split(c, y)
        # find the first perfect
        if entropy == 0:
            cutoff. Stop Iterating
            return i, cur_cutoff, entropy
        elif entropy <= min_entropy: # check if it's best so far
            min_entropy = entropy
            col = i
            cutoff = cur_cutoff
    return col, cutoff, min_entropy
```

Eng yaxshi to'plamlarni aniqlaydigan barcha funksiyalarni yaratib oldik. Endi bevosita mashq jarayonini amalga oshiruvchi $fit(...)$ funksiyasini yaratishga o'tsak bo'ladi. Ushbu funksiyaning asosiy maqsadi yuqoridagi eng yaxshi to'plamlarni aniqlaydigan funksiyalar yordamida berilgan mashq to'plamini quyi to'plamlarga bo'lish orqali qoidalarni Qaror daraxti shoxlariga "ilib" chiqishdir. Ya'ni ildiz element sifatida qaysi qoida, keyingi qadamlardagi qoidalar qanday bo'ladi va h.k.

Bundan tashqari, mashq jarayonida to'plamning Qaror daraxti yordamida ishlanganini vizual tarzda ko'rish uchun topilgan qoida va so'nggi yaproq

elementlarini Pytonning *dictionary* ma'lumotlar strukturasiga qayd qilib boramiz.

```
def fit(self, x, y, par_node={}, depth=0):
    """
    x: Feature set
    y: target variable
    par_node: will be the tree generated for this x and y.
    depth: the depth of the current layer
    """
    if par_node is None: # base case 1: tree stops at previous level
        return None
    elif len(y) == 0: # base case 2: no data in this group
        return None
    elif self.all_same(y): # base case 3: all y is the same in this group
        return {'val':y[0]}
    elif depth >= self.max_depth: # base case 4: max depth reached
        return None
    else: # Recursively generate trees!
        # find one split given an information gain
        col, cutoff, entropy = self.find_best_split_of_all(x, y)
        y_left = y[x[:, col] < cutoff] # left hand side data
        y_right = y[x[:, col] >= cutoff] # right hand side data
        par_node = {'col': iris.feature_names[col], 'index_col':col,
                    'cutoff':cutoff,
                    'val': np.round(np.mean(y))} # save the information
        # generate tree for the left hand side data
        par_node['left'] = self.fit(x[x[:, col] < cutoff], y_left, {}, depth+1)
        # right hand side trees
        par_node['right'] = self.fit(x[x[:, col] >= cutoff],
            y_right, {}, depth+1)
        self.depth += 1 # increase the depth since we call fit once
        self.trees = par_node
        return par_node

def all_same(self, items):
    return all(x == items[0] for x in items)
```

Qaror daraxtini amalga oshirish uchun deyarli barcha funksiyalar tayyor. Endi prognoz qilishni bevosita amalga oshiruvchi funksiyalarga o'tsak bo'ladi. Ushbu funksiyalar mashq jarayonida tanlanga qoidalar asosida to'plamni bo'laklarga bo'lib, prognozni amalga oshiradi.

```
    def predict(self, x):
        tree = self.trees
        results = np.array([0]len(x))
        for i, c in enumerate(x):
            results[i] = self._get_prediction(c)
        return results

    def _get_prediction(self, row):
        cur_layer = self.trees
        while cur_layer.get('cutoff'):
            if row[cur_layer['index_col']] < cur_layer['cutoff']:
                cur_layer = cur_layer['left']
            else:
                cur_layer = cur_layer['right']
        else:
            return cur_layer.get('val')
```

Yuqorida aniqlangan funksiyalar yordamida Qaror daraxtining klassini tuzib chiqamiz:

```
class DecisionTreeClassifier(object):
    def __init__(self, max_depth):
        self.depth = 0
        self.max_depth = max_depth

    def fit(self, x, y, par_node={}, depth=0):
        if par_node is None:
            return None
        elif len(y) == 0:
            return None
        elif self.all_same(y):
            return {'val':y[0]}
        elif depth >= self.max_depth:
            return None
        else:
            # find one split given an information gain
            col, cutoff, entropy = self.find_best_split_of_all(x, y)
            y_left = y[x[:, col] < cutoff]
            y_right = y[x[:, col] >= cutoff]
            par_node = {'col': iris.feature_names[col], 'index_col':col,
                        'cutoff':cutoff,
```

```
                    'val': np.round(np.mean(y))}
            par_node['left'] = self.fit(x[x[:, col] < cutoff],
                y_left, {}, depth+1)
            par_node['right'] = self.fit(x[x[:, col] >= cutoff], y_right, {},
                depth+1)
            self.depth += 1
            self.trees = par_node
            return par_node

    def find_best_split_of_all(self, x, y):
        col = None
        min_entropy = 1
        cutoff = None
        for i, c in enumerate(x.T):
            entropy, cur_cutoff = self.find_best_split(c, y)
            if entropy == 0: # find the first perfect cutoff. Stop Iterating
                return i, cur_cutoff, entropy
            elif entropy <= min_entropy:
                min_entropy = entropy
                col = i
                cutoff = cur_cutoff
        return col, cutoff, min_entropy

    def find_best_split(self, col, y):
        min_entropy = 10
        n = len(y)
        for value in set(col):
            y_predict = col < value
            my_entropy = get_entropy(y_predict, y)
            if my_entropy <= min_entropy:
                min_entropy = my_entropy
                cutoff = value
        return min_entropy, cutoff

    def all_same(self, items):
        return all(x == items[0] for x in items)

    def predict(self, x):
        tree = self.trees
        results = np.array([0]len(x))
        for i, c in enumerate(x):
            results[i] = self._get_prediction(c)
```

```
64          return results
65
66      def _get_prediction(self, row):
67          cur_layer = self.trees
68          while cur_layer.get('cutoff'):
69              if row[cur_layer['index_col']] < cur_layer['cutoff']:
70                  cur_layer = cur_layer['left']
71              else:
72                  cur_layer = cur_layer['right']
73          else:
74              return cur_layer.get('val')
```

Qaror daraxti tayyor. Endi ushbu qaror daraxtini *iris* ma'lumotlar to'plamiga tatbiq qilish orqali klassifikatsiya qilamiz. Avvaliga *iris* to'plamini yuklaymiz hamda mashq va nishonlar to'plamini alohida o'zgaruvchilar (x, y)ga ajratib olamiz.

```
1 iris = load_iris()
2
3 x = iris.data
4 y = iris.target
```

Sklearn kutubxonasining *train_test_split* funksiyasidan foydalanib, ma'lumotlar to'plamini mashq hamda test to'plamlariga ajratib olamiz.

```
1 x_train, x_test, y_train, y_test = train_test_split(x, y)
```

Qaror daraxtini inisializatsiya qilamiz va mashq jarayonini amalga oshiramiz. Bunda mashq jarayoni qaydnomasini *mashq_jarayoni* o'zgaruvchisiga saqlaymiz va chop etamiz.

```
1 qaror_daraxti = DecisionTreeClassifier(max_depth=7)
2 mashq_jarayoni = qaror_daraxti.fit(x_train, y_train)
3 pprint (mashq_jarayoni)
```

```
{'col': 'petal width (cm)',
 'cutoff': 1.0,
 'index_col': 3,
 'left': {'val': 0},
 'right': {'col': 'petal width (cm)',
           'cutoff': 1.6,
           'index_col': 3,
           'left': {'col': 'petal length (cm)',
                    'cutoff': 5.6,
                    'index_col': 2,
                    'left': {'val': 1},
                    'right': {'val': 2},
                    'val': 1.0},
           'right': {'col': 'petal width (cm)',
                     'cutoff': 1.8,
                     'index_col': 3,
                     'left': {'col': 'petal width (cm)',
                              'cutoff': 1.7,
                              'index_col': 3,
                              'left': {'val': 2},
                              'right': {'col': 'sepal length (cm)',
                                        'cutoff': 6.7,
                                        'index_col': 0,
                                        'left': {'val': 2},
                                        'right': {'val': 1},
                                        'val': 2.0},
                              'val': 2.0},
                     'right': {'val': 2},
                     'val': 2.0},
           'val': 2.0},
 'val': 1.0}
```

Qaror daraxtining vizual grafikasida ko'rsatilgan Pytonning *dictionary* qiymati bizga mashq jarayonida Qaror daraxti uchun topilgan qoidalar asosida to'plam qanday bo'linganini ko'rsatib bermoqda. E'tibor bering! Bunda:

- col – qoida sifatida tanlangan ustun;
- $cutoff$ – qoida uchun tanlangan qiymat;
- $left$ – chap bo'linma to'plam, ya'ni $[col < cutoff]$ qoidani qanoatlantiradigan bo'linma to'plam (bunda $value$ – prognoz qilinadigan ustun);
- $right$ – o'ng bo'linma to'plam, ya'ni $[col < cutoff]$ qoidani qanoatlantirmaydigan bo'linma to'plam (bunda to'plamda bir necha klasslar mavjud bo'lsa, bo'linma to'plamni yana bo'lish amalga oshiriladi).

Endi bevosita prognoz qilish jarayoniga o'tsak.

```
1 predicted_y_test = qaror_daraxti.predict(x_test)
```

Endi Qaror daraxtining aniqligini tekshiramiz. O'tgan bo'limlarda klassifikatsiya muammosi uchun o'lchovlarni (*precision, recall,* $f1-score$, *chalkashlik matritsasi*) yozishni o'rgangan edik va ko'rgan edikki, buning uchun bir qancha funksiyalar yozish talab etiladi. Baxtimizga *sklearn* kutubxonasida deyarli barcha ommabop o'lchovlar uchun paket mavjud. Chalkashlik matritsasi uchun *sklearn* kutubxonasidan *classification_report* funksiyasini yuklaymiz.

```
1 from sklearn.metrics import classification_report
```

classification_report funksiyasidan foydalangan holda Qaror daraxti prognoz qilgan *predicted_y_test* massivi hamda haqiqiy nishon qiymatlarini o'zida saqlovchi *y_test* o'rtasidagi chalkashlik matritsasini chop etamiz.

```
1 print(classification_report(y_test, predicted_y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       0.85      0.73      0.79        15
           2       0.75      0.86      0.80        14

    accuracy                           0.84        38
   macro avg       0.87      0.86      0.86        38
weighted avg       0.85      0.84      0.84        38
```

Chalkashlik matritsasidan ko'rib turibmizki, 84% nishonlar to'g'ri topilgan. $f1\_score$ o'lchovi qiymatlarini quyidagicha tavsiflashimiz mumkin:

- 0 nishonlarning barchasi to'g'ri topilgan;
- 1 nishonlarning 73% to'g'ri topilgan;
- 2 nishonlarning 86% to'g'ri topilgan.

Umuman olib aytganda, Qaror daraxtining aniqligini ancha yaxshi deb baholashimiz mumkin.

Foydalanilgan adabiyotlar

[1] itdxer (https://stats.stackexchange.com/users/64943/itdxer). *What is batch size in neural network?* Cross Validated. URL:https://stats.stackexchange.com/q/153535 (version: 2019-04-05). eprint: https://stats.stackexchange.com/q/153535. URL: https://stats.stackexchange.com/q/153535.

[2] *Accuracy, Precision, Recall and F1 Score: Interpretation of Performance Measures.* URL: https://blog.exsilio.com/all/accuracy-precision-recall-f1-score-interpretation-of-performance-measures/#.

[3] *Car data.* URL: https://www.kaggle.com/goyalshalini93/car-data.

[4] *Decision Tree.* URL: https://medium.com/@penggongting/implementing-decision-tree-from-scratch-in-python-c732e7c69aea.

[5] *Decision Tree 2.* URL: https://anderfernandez.com/en/blog/code-decision-tree-python-from-scratch/.

[6] *Design and Optimization of a Deep Neural Network Architecture for Traffic Light Detection.* URL: https://www.researchgate.net/figure/Biological-Neuron-Model_fig1_341241129.

[7] *Gaussian distribution.* URL: http://www.countbio.com/web_pages/left_object/R_for_biology/R_biostatistics_part-1/gaussian_distribution.html.

[8] *Inson vazni haqidagi ma'lumotlar to'plami.* URL: https://www.kaggle.com/yersever/500-person-gender-height-weight-bodymassindex.

[9] *Learn Types of Machine Learning Algorithms with Ultimate Use Cases.* URL: https://medium.com/@ferhat00/deep-learning-with-keras-classifying-cats-and-dogs-part-1-982067594856.

[10] *Neural Network definition.* URL: https://deepai.org/machine-learning-glossary-and-terms/neural-network.

[11] *Polynomial Regression.* URL: https://towardsdatascience.com/polynomial-regression-bbe8b9d97491.

[12] Pinar Saygin, A. Cicekli I., and Akman V. "Turing Test: 50 Years Later. Minds and Machines 10, 463–518 (2000)". In: (). DOI: https://doi.org/10.1023/A:1011288000451.

S.KOMOLOV
SH.RAXMATOV

SUN'IY INTELLEKT ASOSLARI

MASHINAVIY O'QITISH

Texnik muharrir **Ahmat To'ra**
Muharrir **Anvar Jabborov**
Muqova dizayni **Muhammadxon Yusupov**

«IJOD NASHR»
Nashriyot litsenziyasi AA № 0050.
Toshkent sh. Yunusubod tumani,
Sohibkor ko'chasi, 1-uy
Berilgan vaqti 15.06.2020

Bosishga 2022-yil 26-yanvarda ruxsat etildi.
Bichimi 70x100 $^{1}/_{16}$. Ofset bosma.
"Modern No.20" garniturasi.
Nashriyot bosma tabog'i 6,5.
Adadi 1 000. №42 buyurtma.

ijod nashr

Bog'lanish uchun:
+99897 765-56-86

«Azmir nashr print»
bosmaxonasida chop etildi.
100200, Тошкент ш.,
А.Рахмат кучаси, 10 уй.