

1 Nima uchun Machine Learning strategiyasi

Mashinali o‘rganish veb-qidiruv, elektron pochta spamga qarshi, nutqni aniqlash, mahsulot tavsiyalari va boshqalarni o‘z ichiga olgan son-sanoqsiz muhim ilovalarning asosidir. Taxminimcha, siz yoki sizning jamoangiz mashinali o‘rganish ilovasi ustida ishlayapsiz va tezda rivojlanishni xohlaysiz. Bu kitob sizga yordam beradi.

Misol: Mushuk rasmini ishga tushirish:

Aytaylik, siz mushuksevarlarga mushuk rasmlarining cheksiz oqimini taqdim etadigan startap yaratmoqdasiz.



Siz rasmlardagi mushuklarni aniqlash uchun kompyuter ko‘rish tizimini qurish uchun neyron tarmog‘idan foydalanasiz.

Ammo, afsuski, sizning o‘rganish algoritmingizning aniqligi hali yetarli darajada emas. Siz ostida mushuk detektorining yaxshilash uchun juda katta bosim. Nima bilan shug‘ullanasis?

Jamoangizda juda ko‘p g‘oyalar bor, masalan:

- Ko‘proq ma’lumot oling: Mushuklarning ko‘proq suratlarini to‘plang.
- Yanada xilma-xil mashg‘ulot to‘plamini to‘plang. Masalan, mushuklarning g‘ayrioddiy holatdagi rasmlari; mushuklar g‘ayrioddiy ranglar bilan; turli xil kamera sozlamalari bilan suratga olingen rasmlar;....
- Algoritmni uzoqroq mashq qildirish, ko‘proq gradiyentli pasayish iteratsiyalarini bajarish orqali.
- Ko‘proq qatlamlar/yashirin birliklar/parametrlarga ega kattaroq neyron tarmog‘ini sinab ko‘ring. (layers/hidden units/parameters.)
- Kichikroq neyron tarmoqni sinab ko‘ring.
- Regulyarizatsiya qo‘sishga harakat qiling (masalan, L2 regulyarizatsiyasi).
- Neyron tarmoq arxitekturasini o‘zgartirish (faollashtirish funksiyasi, yashirin birliklar soni va h.k.)

Agar siz ushbu mumkin bo‘lgan yo‘nalishlar orasidan yaxshi tanlasangiz, yetakchi mushuk rasmini yaratasziz

platformasini yarating va kompaniyangizni muvaffaqiyatga olib chiqing. Agar noto‘g‘ri tanlasangiz, oylarni behuda sarflashingiz mumkin.

Qanday davom etasiz? Bu kitob sizga qanday qilish kerakligini ko‘rsatadi. Mashinaviy o‘rganish muammolarining aksariyati sizga nima ekanligini ko‘rsatuvchi belgilarni qoldiradi sinab ko‘rish foydali, sinab ko‘rish esa foydasiz. Bu ma’lumotlarni o‘qishni o‘rganish oylarni tejaydi yoki yillar davomida ishlab chiqilgan.

2 Jamoangizga yordam berish uchun bu kitobdan qanday foydalanish mumkin

Ushbu kitobni tugatganingizdan so‘ng, mashina o‘rganish (machine learning) loyihasi uchun texnik yo‘nalishni qanday belgilash haqida chuqur tushunchaga ega bo‘lasiz.

Lekin jamoangiz nima uchun aynan shu yo‘nalishni tavsiya qilayotganingizni tushunmasligi mumkin.

Masalan, siz jamoangizdan bitta raqamli baholash mezonini (single-number evaluation metric) belgilashni xohlaysiz, lekin ular bunga ishonmayapti.

Ularni qanday qilib ishontirasiz?

Shu sababli men boblarni qisqa qildim: shunda siz ularni chop etib, jamaa a’zolaringizga faqat sizga kerak bo‘lgan 1–2 sahifani o‘qitishingiz mumkin bo‘ladi.

Ustuvorliklarni (prioritizatsiyani) biroz o‘zgartirish jamoangiz unumdarligiga juda katta ta’sir ko‘rsatishi mumkin. Jamoangizga shunday bir nechta o‘zgarishlar bilan yordam berib, siz jamoangizning haqiqiy qahramoniga aylanasisiz degan umiddaman jamoangizning haqiqiy qahramoniga aylanasisiz degan umiddaman!

3 Talablar va Belgilanishlar (Notation)

Men siz nazoratli o‘rganish tushunchasi bilan tanish deb hisoblayman: bu – belgilangan (yorliqlangan) o‘quv namunalariga asoslanib, **x dan y ga** xaritalovchi funksiyani o‘rganishdir. Nazoratli o‘rganish algoritmlariga chiziqli regressiya (*linear regression*), logistik regressiya (*logistic regression*) va neyron tarmoqlar (*neural networks*) kiradi. Mashina o‘rganishning ko‘plab shakllari mavjud, ammo hozirgi kunda uning amaliy qiymatining katta qismi aynan nazoratli o‘rganishdan kelib chiqadi.

Men tez-tez neyron tarmoqlar (yoki *chuqur o‘rganish, deep learning*) haqida eslatib o‘taman. Ushbu matnni tushunish uchun sizda ular haqida asosiy tushuncha bo‘lishi kifoya.

4 Masshtab (Scale) — machine learning dagi taraqqiyotning harakatlantiruvchi kuchidir.

Chuqur o‘rganish (*deep learning*) — **neyron tarmoqlar** (*neural networks*) g‘oyalari o‘n yillardan beri mavjud (*have been around for decades*). Unda nega endi bu g‘oyalar **keskin rivojlanib ketmoqda** (*taking off now*)?

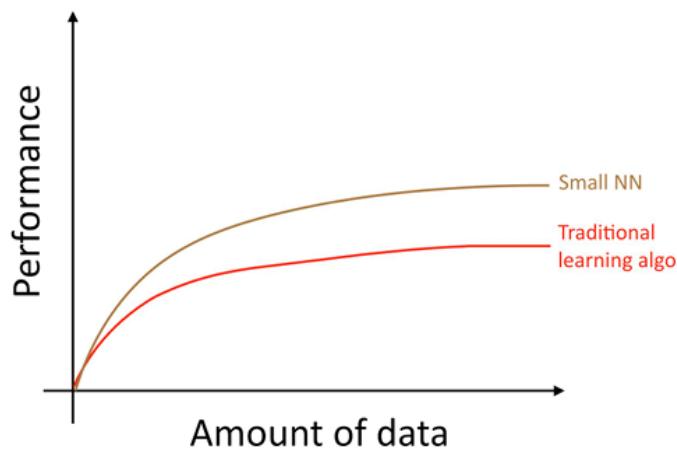
So‘nggi yillardagi yutuqlarning **ikkita asosiy omili** (*two of the biggest drivers*) bor:

- **Ma’lumotlarning mavjudligi** (*data availability*). Odamlar hozirda **raqamli qurilmalarda** (*digital devices*) — **noutbuk, mobil qurilmalar** — ko‘proq vaqt o‘tkazmoqda. Ularning **raqamli faoliyati** (*digital activities*) esa o‘rganish algoritmlarimizga beriladigan **juda katta hajmdagi ma’lumotlarni** (*huge amounts of data*) yaratmoqda.
- **Hisoblash masshtabi** (*computational scale*). Faqat so‘nggi yillarda biz **neyron tarmoqlarni shunchalik katta hajmda** (*neural networks that are big enough*) o‘qitishni boshladikki, ular mavjud **ulkan datasetlardan** (*huge datasets*) foyda olish darajasiga yetdi.

Batafsil aytganda (*in detail*), siz **ko‘proq ma’lumot** (*more data*) to‘plaganingizda ham, odatdagi eski o‘rganish algoritmlari, masalan, **logistik regressiya** (*logistic regression*), odatda ma’lum bir nuqtada **to‘xtab qoladi** (*plateaus*) – ya’ni uning **o‘rganish egri chizig‘i** (*learning curve*) **tekislanadi** (*flattens out*), va siz unga yana ma’lumot qo‘sghaningizda ham, algoritm **yaxshilanmaydi** (*stops improving*).

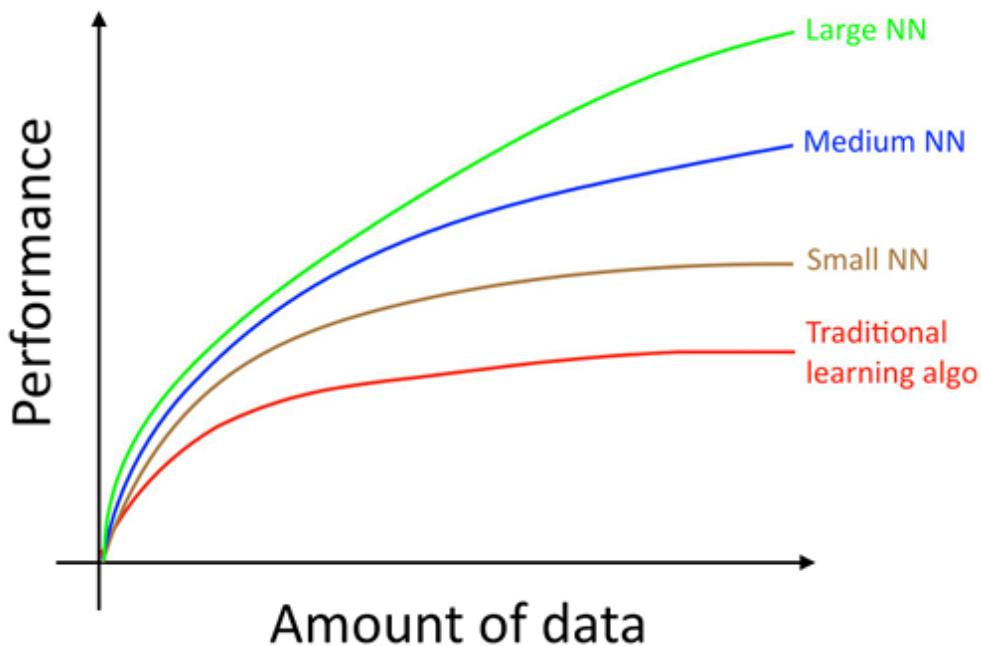
Go‘yoki eski algoritmlar hozir bizda mavjud bo‘lgan barcha **ma’lumotlar** (*data*) bilan nima qilishni **bilmagandek edi** (*didn’t know what to do*).

Agar siz **bir xil nazoratli o‘rganish vazifasida** (*same supervised learning task*) **kichik bir neyron tarmoqni** (*small neural network, NN*) o‘qitsangiz, u holda **biroz yaxshiroq natija** (*slightly better performance*) olishingiz mumkin.



Bu yerda “Small NN” deganda biz **kam sonli yashirin qatlamlar / birliklar / parametrlarga ega bo‘lgan neural networkni** nazarda tutyapmiz.

Agar siz yanada katta va murakkab neyron tarmoqlarni (*larger and larger neural networks*) o‘qitsangiz, yanada yaxshi performance (ishlash samaradorligi) ga erishishingiz mumkin.



Shunday qilib, **eng yaxshi natijani** quyidagilarga amal qilgan holda olasiz:

- (i) **Juda katta neyron tarmoqni o'qitish** (*train a very large neural network*), bu sizni yuqoridagi **yashil egri chiziqqa** (*green curve*) olib chiqadi;
 - (ii) **Juda katta hajmdagi ma'lumotga ega bo'lish** (*have a huge amount of data*). Albatta, **neyron tarmoq arxitekturasi** (*neural network architecture*) kabi boshqa ko'plab tafsilotlar ham muhim va bu sohada ko'p yangiliklar bo'lgan.
- Ammo bugungi kunda **algoritm samaradorligini yaxshilashning eng ishonchli usullaridan biri** hamon quyidagilardir:

- (i) **katta tarmoqni o'qitish**, va
- (ii) **ko'proq ma'lumot to'plash**.

Eslatma: Bu diagrammada **kichik datasetlar** holatida ham neyron tarmoqlar yaxshi ishlayotgani ko'rsatilgan.

Ammo bu holat **har doim ham izchil bo'lmaydi** (*less consistent*) — ya'ni, kichik ma'lumotlarda ba'zida **an'anaviy algoritmlar** (*traditional algorithms*) yaxshi ishlashi ham mumkin.

Masalan, agar sizda atigi **20 ta o'quv namunasi** bo'lsa, **logistik regressiya** yoki **neyron tarmoq ishlatganingiz** muhim emas — bu yerda **qo'lda yaratilgan xususiyatlar** (*hand-engineered features*) **algoritm tanlovidan muhimroq** bo'ladi.

Ammo agar sizda **1 millionta o'quv namunasi** bo'lsa, unda men **neyron tarmoqni tanlagan bo'lardim**.

Ushbu kitobda yuqoridagi (i) va (ii) bandlarni qanday amalga oshirish haqida **batafsil** tushuntiriladi.

Biz **an'anaviy o'rganish algoritmlari** va **neyron tarmoqlar** uchun foydali bo'lgan **umumiyy strategiyalardan** boshlaymiz, so'ngra **chuqur o'rganish tizimlarini yaratish bo'yicha eng zamonaviy strategiyalarga** o'tamiz.

Rivojlantirish (development) va test to‘plamlarini sozlash *(Setting up development and test sets)*

5 Sizning development va test to‘plamlaringiz

Keling, oldingi **mushuk rasmi misoliga** qaytaylik: Siz bir mobil ilovani boshqarasiz va foydalanuvchilar turli xil narsalarning rasmlarini yuklashmoqda. Siz ushbu ilovada **mushuk rasmlarini avtomatik aniqlamoqchisiz**.

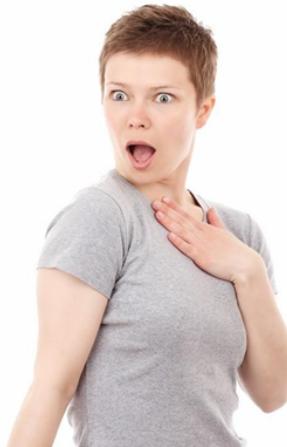
Jamoangiz katta o‘quv to‘plami (*large training set*) yaratish uchun internetdan **mushuklar** (ijobiy misollar — *positive examples*) va **mushuk bo‘Imagan** (salbiy misollar — *negative examples*) rasmlarni yuklab olishadi. Keyin ular bu datasetni **70% trening** va **30% test** to‘plamlariga bo‘lishadi.

Ular shu ma’lumotlar asosida **mushuk aniqlovchi** (*cat detector*) model yaratishadi va bu model **trening** hamda **test to‘plamlarida** yaxshi ishlaydi.

Test natijalari yaxshi ko‘ringan bo‘lishi mumkin.

Lekin siz bu **klassifikatorni mobil ilovaga joylaganingizda**, **model ish faoliyati juda sust bo‘lib chiqadi!**

(*you find that the performance is really poor!*)



Nima bo‘ldi?

Siz aniqlaysizki, foydalanuvchilar yuklayotgan rasmlar **ko‘rinish jihatdan boshqacha**: ularning ilovaga yuklayotgan suratlari **mobil telefonlar** bilan olingan, bu esa odatda:

- **pastroq aniqlikdagi** (*lower resolution*),
- **xiralashgan** (*blurrier*),
- va **yomon yoritilgan** (*poorly lit*) bo‘ladi.

Sizning **trening va test to‘plamingiz** esa **veb-saytlardan** yuklab olingan aniq, sifatli **rasmlardan** tashkil topgan edi.

Shu sababli, sizning algoritmingiz **aslida siz uchun muhim bo‘lgan taqsimotga** (*actual distribution you care about*) – ya’ni **mobil telefon suratlariga** – **umumlashtira olmadi** (*did not generalize well*).

Zamonaviy **katta ma’lumotlar davrigacha**, mashina o‘rganishda **70%/30% tasodifiy ajratish** (*random 70%/30% split*) orqali trening va test to‘plamlarini shakllantirish **odatiy qoida** edi.

Bu usul **ba’zi holatlarda ishlashi mumkin**,

lekin ko‘pgina zamonaviy ilovalarda bu **yomon fikrga aylanmoqda**.

Nega?

Chunki ko‘pincha **trening to‘plami taqsimoti** (*training distribution*) — masalan, yuqoridagi misolda **veb-saytdan olingan rasmlar** —

siz uchun **asosiy ahamiyatga ega bo‘lgan real taqsimotdan** (*distribution you ultimately care about*) — ya’ni **mobil telefon rasmlaridan** — farq qiladi.

Training set (*o‘quv to‘plami*) — Siz o‘rganish algoritmini aynan shu to‘plamda ishlatasiz. Model shu ma’lumotlar asosida o‘qitiladi.

Dev set (*development set, ba’zida hold-out cross validation set* deb ataladi) — Siz gipermetrlarni sozlash, xususiyatlarni tanlash va boshqa **algoritm bo‘yicha qarorlar qabul qilish** uchun foydalanasiz.

Test set — Siz bu to‘plamni faqatgina **modelning yakuniy ishlash sifatini baholash** uchun ishlatasiz. Bu yerda siz **algoritmni yoki parametrlarni tanlamaysiz**, faqat baholaysiz.

Nega bu muhim?

Dev va test setlar orqali siz jamoa bilan turli fikrlarni — algoritm turlari, parametrlar, arxitekturalarni — sinab ko‘rib, **qaysi variant eng yaxshi ishlashini tezda ko‘rishingiz mumkin**.

Dev va test to‘plamlarining maqsadi:

Jamoani to‘g‘ri o‘zgarishlar sari yo‘naltirish — ya’ni mashina o‘rganish tizimini yaxshilash uchun **eng muhim qadamlarni aniqlash**.

6 Nima qilish kerak?

Siz dev va test to‘plamlarini kelajakda olishni kutayotgan ma’lumotlarga mos tarzda tanlashingiz kerak.

Ya’ni, **test set** bu shunchaki mayjud ma’lumotlarning 30% qismi bo‘lmasi kerak — ayniqsa agar **kelajakdagi ma’lumotlar** (masalan, mobil telefon rasmlari) **treningdagi ma’lumotlardan** (masalan, sayt rasmlaridan) **farq qilsa**.

Agar siz hali ilovani ishga tushirmagan bo‘lsangiz va foydalanuvchilaringiz bo‘lmasa, unda sizda **real ma’lumotlar yo‘q** bo‘lishi mumkin.

Bunday holatda siz **real holatni taxminan aks ettiruvchi ma’lumotlar** to‘plashga harakat qilishingiz kerak.

Misol: Do‘stlarining **mobil telefon bilan mushuklarning rasmlarini** olib, sizga yuborishlarini so‘rang.

Shunda **real foydalanuvchi ma’lumotlarini taxminan aks ettiruvchi** dev/test setga ega bo‘lasiz.

Ogohlantirish:

Agar sizda boshqa iloj bo‘lmasa va faqat **sayt rasmlarini** ishlashiga majbur bo‘lsangiz, bu bilan boshlashningiz mumkin.

Ammo **bu yondashuv oxir-oqibat umumlashtirilmaydigan tizimga** olib kelishi mumkinligini **tushunib turing**.

(risk of this leading to a system that doesn’t generalize well)

Zo‘r dev va test to‘plamlarini yaratishga qancha resurs sarflash kerakligini aniqlash — bu tajriba va fikr yuritishni talab qiladi.

(It requires judgment to decide how much to invest in developing great dev and test sets.)

Lekin sizning trening to‘plamingizdagi taqsimot — ya’ni **ma’lumotlar tarqatilishi** — test to‘plamingizdagi taqsimotga aynan teng deb o‘ylamang.

(Don’t assume your training distribution is the same as your test distribution.)

Shunchaki “nimadir bor ekan” deb test qilish o‘rniga:

Siz test to‘plamini aynan o‘zingiz yaxshi ishlashni xohlayotgan real vaziyatlarni aks ettiruvchi misollardan tanlang.

(Try to pick test examples that reflect what you ultimately want to perform well on, rather than whatever data you happen to have for training.)

Sizning dev (development) va test to‘plamlaringiz bir xil taqsimotdan bo‘lishi kerak
(Your dev and test sets should come from the same distribution)



Siz mushuklarni aniqlovchi ilova ustida ishlayapsiz va rasm ma'lumotlaringizni **to'rt mintaqaga** ajratgansiz:

- (i) AQSH,
- (ii) Xitoy,
- (iii) Hindiston,
- (iv) Boshqa.

Siz dev va test to'plamlarini quyidagicha tuzmoqchisiz:

- AQSH va Hindiston — **dev set**,
- Xitoy va Boshqa — **test set**.

Siz o'ylayapsiz: "Axir shunchaki to'rt segmentdan ikkitani dev setga, qolgan ikkitani test setga **tasodifiy taqsimladik-ku**, to'g'rimi?"

Bu yondashuv muammo tug'diradi. Nega?

1. **Dev set — bu siz yaxshilashni xohlaydigan asosiy vazifani aks ettirishi kerak.**

Agar siz faqat AQSH va Hindistondagi ishlashni nazorat qilsangiz, jamoa **shu ikkitasiga moslashgan model** tuzadi.

Ammo siz **to'rt mintaqada ham** yaxshi ishlashni xohlaysiz, to'g'rimi?

2. **Agar dev va test to'plamlarining taqsimoti bir xil bo'limasa**, sizda quyidagi xatoliklar yuz beradi:

- a. Dev setda ishlagan yechim test setda ishlamaydi.
- b. Nima noto'g'ri ketganini **aniqlash qiyinlashadi**.
- c. Bu esa **vaqt va resurslarning behuda sarflanishiga** olib keladi.

Misol: Tasavvur qiling, jamoangiz dev setda (AQSH + Hindiston) yaxshi ishlaydigan model yaratdi.

Lekin test setda (Xitoy + Boshqa) model **umuman ishlamaydi**.

Agar dev va test set **bir xil taqsimotdan** bo'lganida:

- Siz bu holatni **overfitting** deb tushungan bo'lardingiz
- Yechim: **ko'proq dev set ma'lumot** olish

Ammo ular **turli taqsimotdan** bo'lsa, unda:

Muammo nimada?

1. Overfittingmi?
2. Test set shunchaki qiyinmi?
3. Yoki test set boshqa, shunchaki **turli**, dev setda ishlagan narsa bu yerda ishlamayaptimi?

Bu holatlarda **tahlil qilish, strategiya tanlash, ustuvorlik belgilash qiyinlashadi**.

Xulosa va tavsiya:

Agar siz ilmiy tadqiqot bilan emas, balki **aniq bir ML amaliyoti** ustida ishlayotgan bo'lsangiz

Dev va test to‘plamlaringizni bir xil taqsimotdan tanlang.

Bu sizga quyidagilarni beradi:

- Aniqroq tahlil
- Yaxshiroq prioritetlar
- Kamroq vaqt isrof
- Barqaror yondashuv
- Eslatma: Ilmiy tadqiqotlarda har xil taqsimotlarda ishlaydigan algoritm yaratish dolzARB masala.

Lekin **amaliy ML tizimi qurishda** eng samarali yo‘l — **bir xil taqsimotli dev/test to‘plamlar** bilan ishlashdir.

7 Dev va test to‘plamlari qanchalik katta bo‘lishi kerak?

(How large do the dev/test sets need to be?)

Dev set (development set):

Dev set — bu siz turli algoritmlar yoki parametrlarni sinab ko‘rganiningizda, **ular orasidagi farqni sezishingiz uchun yetarlicha katta bo‘lishi kerak.**

Misol:

Agar:

- **Classifier A** — 90.0% aniqlikka ega bo‘lsa,
- **Classifier B** — 90.1% bo‘lsa,
va sizda **faqat 100 ta misol bo‘lsa**, bu **0.1% farqni aniqlay olmaysiz.**

Chunki bu juda kichik dev set hisoblanadi.

Amalda esa:

- **1,000 dan 10,000 gacha** bo‘lgan dev setlar **odatiy holat**
- 10,000 misollik dev set bilan **0.1% farqni aniqlash ehtimoli ancha yuqori**

Yirik va muhim ilovalar (masalan, reklamalar, web qidiruv, mahsulot tavsiyalari) uchun, ba’zi jamoalar hatto **0.01%** yaxshilanishni ham aniqlashga harakat qiladi, chunki bu **kompaniya foydasiga bevosita ta’sir qiladi.**

Shunday holatda dev setlar **10,000+** misoldan ham katta bo‘ladi.

Test set:

Test set esa **butun tizim samaradorligini ishonchli baholash** uchun yetarlicha katta bo‘lishi kerak.

Odatda ishlatilgan usul:

- Ma’lumotlarning **30% qismini** test setga ajratish (100 – 10,000 misol bo‘lsa, bu hali ham mantiqli)

Ammo katta hajmli ma’lumotlar davrida (ba’zan **1 milliarddan ortiq misol** mavjud bo‘ladi),

- Dev/test setlar uchun **ajratilgan foiz kamaymoqda**,
- Lekin **ularning mutlaq soni ortmoqda**.

Muhim: Dev/test setlar **ortiqcha katta bo‘lishi shart emas** — ular **algoritmlarni baholash uchun yetarli bo‘lsa bas.**

Izoh:

Nazariy jihatdan, siz **algoritmdagi o‘zgarish statistik jihatdan sezilarli yoki yo‘qligini test qilish** mumkin.

Lekin amaliyotda (ilmijy maqola yozish holatlari bundan mustasno), jamoalar buni kamdan-kam qiladi.

Shaxsan Andrew Ng ham **interim (vaqtinchalik) o‘sishni baholashda statistik testlarni unchalik foydali deb hisoblamaydi.**

8 Jamoa uchun yagona raqamli baholash mezonini tanlang

Mashina o‘rganish loyihibarida jamoaning harakatini to‘g‘ri yo‘naltirish uchun **bitta aniq raqamli (single-number) baholovchi ko‘rsatkich** tanlash juda muhim. Bunday ko‘rsatkich yordamida siz turli algoritmlar yoki yondashuvlarni aniq taqqoslab, qaysi biri yaxshiligi haqida tezroq qaror qilishingiz mumkin.

Masalan, **aniqlik (accuracy)** shunday yagona metrikaga misol bo‘ladi. Siz dev yoki test to‘plamida modelni sinab ko‘rasiz va u nechta namunani to‘g‘ri tasniflaganini foizda ko‘rsatadi. Biroq **Precision** va **Recall** kabi ko‘rsatkichlar ikki xil natija beradi: biri to‘g‘ri tasniflangan ijobjiy javoblar ulushini bildirsa, ikkinchisi haqiqiy ijobjiy holatlardan qanchasi to‘g‘ri topilganini ko‘rsatadi. Bu esa algoritmlarni taqqoslashni murakkablashtiradi.

Bu yerda qaysi model yaxshiligi haqida aniq qaror qilish qiyinlashadi, chunki har biri boshqa jihatdan ustunlikka ega.

Shuning uchun, **F1 score** kabi bitta raqamda ikkala ko‘rsatkichni birlashtiruvchi metrikadan foydalanish tavsiya etiladi. Bu jamoangizni bir yo‘nalishga jamlaydi va ishlash samaradorligini oshiradi.

| Classifier | Precision | Recall |
|------------|-----------|--------|
| A | 95% | 90% |
| B | 98% | 85% |

Bu yerda **ikkala klassifikator ham aniq ustun emas**, shuning uchun bu sizga **darhol qaysi birini tanlash kerakligini ko‘rsatmaydi**.

Loyihani ishlab chiqish davomida sizning jamoangiz ko‘plab g‘oyalarni sinab ko‘radi:

- algoritm arxitekturasi,
- model parametrlari,
- xususiyatlar (features) tanlovi va hokazo.

Shu jarayonda **bitta raqamli baholash mezonini** (masalan, **aniqlik – accuracy**) bo‘lishi, model natijalarini tartiblash va eng yaxshisini tezda tanlash imkonini beradi.

Agar siz Precision va Recall ikkalasiga ham ahamiyat qaratayotgan bo‘lsangiz, unda ularni **bitta raqamga birlashtirish** tavsiya etiladi.

Buni amalga oshirishning ba’zi usullari:

1. **Precision va Recall o‘rtacha qiymatini** olish (masalan, arifmetik o‘rtacha)
2. Yoki **F1 score** ni hisoblash:

Bu metrika Precision va Recall o‘rtasidagi muvozanatni hisobga oladi, va **ikkisidan biri past bo‘lsa, F1 ham past bo‘ladi** – bu esa yanada realistik baholash beradi. Yagona raqamli baholash mezoniga ega bo‘lish, ko‘plab klassifikatorlar orasidan tanlov qilish jarayonini tezlashtiradi. Bu har bir model uchun aniq ustuvorlik (ranking) beradi va shuning orqali jamoangizga qaysi yo‘nalishda harakat qilish kerakligini aniqlashtirib beradi.

Masalan, agar siz mushuklarni aniqlovchi modelning aniqligini to‘rtta asosiy bozor bo‘yicha kuzatayotgan bo‘lsangiz — (i) AQSH, (ii) Xitoy, (iii) Hindiston va (iv) Boshqa — bu sizga to‘rtta alohida ko‘rsatkich beradi. Bunday holatda ham, har xil natijalarni solishtirish murakkab bo‘lishi mumkin.

Shunday vaziyatda bu to‘rtta ko‘rsatkichning oddiy o‘rtachasini yoki vaznli o‘rtachasini olish orqali ularni bitta yagona raqamga birlashtirish mumkin. O‘rtacha yoki vaznli o‘rtacha hisoblash — bir nechta baholash mezonlarini yagona ko‘rsatkichga birlashtirishning eng keng tarqalgan usullaridan biridir.

9 Optimallashtiriluvchi va yetarli (satisficing) metrikalar

Bu bo‘limda bir nechta baholash mezonlarini birlashtirishning yana bir usuli bilan tanishamiz. Tasavvur qiling, siz mashina o‘rganish algoritmini baholayotganda **ikkita muhim jihatga** e’tibor qaratayapsiz:

- **Aniqlik (accuracy)**
- **va ishslash vaqt (running time)**

Endi siz quyidagi uchta klassifikatordan birini tanlashingiz kerak:

| Classifier | Accuracy | Running time |
|------------|----------|--------------|
| A | 90% | 80ms |
| B | 92% | 95ms |
| C | 95% | 1,500ms |

Aniqlik (accuracy) va ishslash vaqt (running time) kabi ikki turli metrikani bitta formulaga birlashtirish, masalan:

Accuracy – $0.5 \times \text{Running Time}$

kabi shaklda ifodalash tabiiy emasdek tuyuladi.

Buning o‘rniga, quyidagicha yondashuvni qo‘llashingiz mumkin:

Avval, **qoniqarli ishslash vaqt** qanday bo‘lishi kerakligini belgilab oling. Masalan, 100 millisekundda ishlaydigan har qanday model **qoniqarli** deb hisoblanadi. Keyin esa, **shu shartni qanoatlantiruvchi modellar orasidan aniqligi eng yuqori bo‘lganini tanlang**.

Bu yerda ishslash vaqt — **yetarli (satisficing) metrika** bo‘lib xizmat qiladi: ya’ni model bu metrika bo‘yicha “etarlicha yaxshi” bo‘lishi kerak — 100ms dan ortiq bo‘lmasi lozim.

Aniqlik esa — **optimallashtiriluvchi metrika (optimizing metric)** bo‘ladi: siz aynan shuni maksimal darajaga yetkazishga harakat qilasiz.

Agar siz bir vaqtning o‘zida bir nechta (N ta) mezonlarni inobatga olayotgan bo‘lsangiz — masalan:

- modelning fayl o‘lchami (mobil ilovalar uchun muhim, chunki foydalanuvchilar katta ilovalarni yuklab olishni xohlamaydi),
- ishslash vaqt,
- aniqlik —

unda siz **N-1 ta metrikani satisficing** deb belgilashingiz mumkin. Ya’ni, bu metrikalar bo‘yicha model faqat belgilangan chegaraga yetishi kifoya. Keyin qolgan bitta metrikani **optimallashtiriluvchi metrika** sifatida tanlaysiz.

Masalan, fayl o‘lchami va ishlash vaqtining qabul qilinadigan chegaralarini belgilab olasiz, so‘ngra shu shartlar doirasida **aniqlikni optimallashtirishga** harakat qilasiz.

Yakuniy misol sifatida, siz ovozli signalni tanib olishga asoslangan apparat qurilma ishlab chiqayotganingizni tasavvur qiling. Bu qurilma mikrofon orqali foydalanuvchi tomonidan aytildigan “faollashtiruvchi so‘z” (wakeword) ni eshitadi va tizimni ishga tushiradi. Masalan:

- Amazon Echo — “Alexa”
- Apple Siri — “Hey Siri”
- Android — “Okay Google”
- Baidu — “Hello Baidu”

Bunday tizimda siz quyidagi ikkala xatolik turiga e’tibor berasiz:

- **False positive** — hech kim hech narsa demagan bo‘lsa ham, tizim noto‘g‘ri tarzda uyg‘onib ketadi
- **False negative** — foydalanuvchi “wakeword” ni aytgan bo‘lsa-da, tizim javob bermaydi

Bu holatda, **optimallashtiriluvchi metrika** — **false negative darajasini minimallashtirish** bo‘ladi, ya’ni tizim foydalanuvchi so‘z aytganda har doim uyg‘onishi kerak.

Shu bilan birga, siz **false positive holati** bo‘yicha **satisficing metrika** belgilaysiz — masalan, tizim bir sutkada bir martadan ortiq noto‘g‘ri uyg‘onmasligi kerak.

Jamoa qaysi metrikaning optimallashtirilayotganini aniq tushunib olgach, ular **tezroq va samaraliroq rivojlanish** yo‘liga kirishadi.

10 Dev to‘plami va baholash mezoniga ega bo‘lish iteratsiyalarni tezlashtiradi

Yangi muammoga qanday yondashuv eng yaxshi natija berishini **oldindan aniq bilish juda qiyin**.

Hatto tajribali mashina o‘rganish tadqiqotchilari ham **qoniqarli natijaga erishishdan oldin o‘nlab g‘oyalarni sinab ko‘rishadi**.

Mashina o‘rganish tizimini ishlab chiqayotganimda, odatda quyidagi bosqichlarni bajaraman:

1. Dastlab tizimi qanday qurish haqida biror g‘oyani tanlayman.
2. Ushbu g‘oyani kod shaklida amalga oshiraman.
3. Keyin esa, bu g‘oya **qanchalik yaxshi ishlaganini aniqlash uchun tajriba o‘tkazaman**.

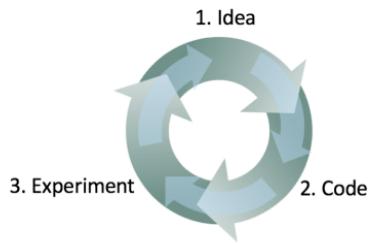
(Ko‘pincha dastlabki bir necha g‘oyam umuman ishlamaydi!)

O‘sha tajribalar asosida saboq olib, yangi g‘oyalar ishlab chiqaman va bu jarayonni doimiy ravishda davom ettiraman.

 Aynan shu **sinov–xulosa–takomillashtirish (iteration)** jarayonini tezlashtirish uchun **dev to‘plami** va **Aniq baholash mezoni** bo‘lishi muhimdir.

Bu vositalar sizga har bir yondashuvning qanchalik muvaffaqiyatlari ekanini tez baholashga yordam beradi,

va yangi g‘oyani yaratishdan ko‘ra, qaysi yo‘nalishda davom etish kerakligini tezda tushunishga imkon beradi.



Bu — **iterativ jarayon**. Ya’ni, siz g‘oyani ishlab chiqasiz, sinab ko‘rasiz, natijani baholaysiz va shu asosda yangi qarorlar qabul qilasiz. Siz bu jarayonni qanchalik tez bajarsangiz, shunchalik tez taraqqiyotga erishasiz.

Aynan shu sababli **dev/test to‘plamlari va aniq baholash mezoniga ega bo‘lish juda muhimdir**:

Har safar siz yangi g‘oyani sinaganingizda, uni **dev to‘plamida baholash orqali**, bu g‘oya to‘g‘ri yo‘nalishda ekanini tezda anglash imkoniga ega bo‘lasiz.

Bunga qarama-qarshi holatni tasavvur qiling: Sizda **aniq dev to‘plami va mezon** yo‘q.

Shunday bo‘lsa, har safar yangi mushuk klassifikatorini ishlab chiqganingizda, uni ilovaga qo‘sib, so‘ng **bir necha soat davomida ilova bilan ishlab**, bu yangi model yaxshilanganmi yoki yo‘qmi, degan savolga javob topishingiz kerak bo‘ladi.

Bu usul juda sekin ishlaydi!

Yana bir muammo: agar sizning jamoangiz model aniqligini 95.0% dan 95.1% ga oshirsa, siz bu **0.1% yaxshilanishni oddiy kuzatish orqali deyarli sezal olmaysiz**.

Holbuki, ko‘plab mashina o‘rganish tizimlarida **bir nechta kichik — 0.1% ga teng — yaxshilanishlarni ketma-ket qo‘sish orqali katta taraqqiyotga erishiladi**.

Dev to‘plami va baholash mezoniga ega bo‘lish, sizga:

- qaysi g‘oyalalar kichik (yoki katta) ijobiy natija berayotganini tezda aniqlash,
- foydali bo‘lgan yondashuvlarni rivojlantirishni davom ettirish,
- samara bermayotgan g‘oyalardan voz kechish imkonini beradi.

11. Dev/test to‘plamlari va metrikalarini qachon o‘zgartirish kerak

Yangi loyiha ustida ishlashni boshlaganingizda, **dev/test to‘plamarini tezda tanlab olish** muhim. Bu jamoaga **aniq va o‘lchanadigan maqsad** belgilash imkonini beradi.

Shaxsan men jamoamdan **bir haftadan kam vaqt ichida dastlabki dev/test to‘plami va baholash mezonini aniqlab olishni** so‘rayman — odatda bu jarayon bundan uzoq cho‘zilmaydi.

Mukammallikni ortiqcha o‘ylab ko‘p vaqt sarflashdan ko‘ra, **nomukammal, lekin tezkor boshlanish afzalroq**.

Ammo bu "bir haftalik" muddat **mature — ya’ni yetuk va barqaror ishlayotgan tizimlar uchun mos emas**.

Masalan, spamga qarshi kurash tizimi — chuqur o‘rganishga asoslangan, yetuk bir soha hisoblanadi.

Men bunday tizimlar ustida ishlayotgan jamoalarning **yangi va ancha sifatli dev/test to‘plamlarni yig‘ish uchun oylar sarflaganini ko‘rganman**.

Tarjimon: Sh. Ne’matov

Tahrirchi: A.Dadajonov

Agar vaqt o'tib, sizning tanlagan **dastlabki dev/test to‘plamingiz yoki metrikangiz noto‘g‘ri yo‘lga boshlayotganini sezsangiz**, ularni imkon qadar tez o‘zgartiring.

Masalan, agar dev to‘plamingizdagi baholash natijasi **klassifikator A** ni **klassifikator B** dan ustun deb chiqarsa,

lekin sizning jamoangiz aslida **klassifikator B mahsulotingiz uchun yaxshiroq ishlayapti** deb hisoblasa —

bu holat sizga **dev/test to‘plam yoki metrikani qayta ko‘rib chiqish kerakligini ko‘rsatadi**.

Bunday xatoliklarning uchta asosiy sababi mayjud:

1. Siz muvaffaqiyatga erishmoqchi bo‘lgan real tarqatma (distribution) dev/test to‘plamingizdagи tarqatmadan farq qiladi.

Masalan, sizning dastlabki dev/test to‘plamingiz asosan **kattalar mushuklari** (adult cats) suratlaridan iborat bo‘lgan.

Siz ilovani foydalanuvchilarga yetkazganingizdan so‘ng, ular **kutilganidan ko‘ra ko‘proq mushuk bolalari (kittens)** suratlarini yuklay boshladi.

Demak, dev/test to‘plamingiz **real ma‘lumotlar oqimini yetarlicha aks ettirmayapti**.

Bunday holatda, **dev/test to‘plamlaringizni o‘zgartirib**, ular real foydalanish holatiga mos kelishini ta’minlashingiz kerak.



2. Siz dev to‘plamga haddan tashqari moslashib (overfit) qolgansiz

G‘oyalarni takror-takror dev to‘plamida sinab ko‘rish jarayoni, modelni **shu dev to‘plamga haddan tashqari moslashishga olib kelishi** mumkin.

Siz modelni yakunlaganingizdan so‘ng, uni **test to‘plamida baholaysiz**.

Agar test to‘plamidagi natijalar dev to‘plamiga qaraganda ancha yomon bo‘lsa, bu sizning model dev to‘plamga **overfit** bo‘lganini bildiradi.

Bunday holatda, sizga **yangi, toza dev to‘plami** kerak bo‘ladi.

Agar jamoangizning taraqqiyotini kuzatib borishingiz kerak bo‘lsa, siz test to‘plamida modelni **har hafta yoki har oyda bir marta** baholashingiz mumkin.

Biroq, **test to‘plam natijalariga asoslanib hech qanday qaror qabul qilmang**. Masalan, agar siz test natijasiga qarab, “oldingi haftadagi modelga qaytamiz” deb qaror qilsangiz — bu test to‘plamga moslashishni (overfitting) boshlaganiningizni bildiradi.

Shunda siz test to‘plamga ham ishonib bo‘lmaydigan holatga tushasiz va u tizimning haqiqiy unumdorligini bexato baholay olmaydi.

Bu ayniqsa, siz ilmiy maqola nashr qilayotgan bo‘lsangiz yoki muhim biznes qarorlarini shu metrika asosida qabul qilayotgan bo‘lsangiz, **jiddiy muammo hisoblanadi**.

3. Metrika loyihaning haqiqiy maqsadini aks ettirmayapti

Tasavvur qiling, sizning mushuklarni aniqlovchi ilovangizda **aniqlik (accuracy)** metrikasidan foydalanilmoqda.

Bu metrika hozircha **klassifikator A** ni **klassifikator B** dan ustun deb baholamoqda.

Lekin siz ikkala algoritmni amalda sinab ko‘rgansiz va shuni aniqlagansizki, **klassifikator A ba’zida pornografik tasvirlarning ilovaga o‘tib ketishiga yo‘l qo‘ymoqda**.

Garchi A klassifikator umumiy aniqlik bo‘yicha yaxshiroq ko‘rinsa ham, foydalanuvchida qoldirgan salbiy taassurot sababli u **qabul qilib bo‘lmaydigan** holatga aylanadi.

Bunday holatda metrika, aslida **klassifikator B mahsulotingiz uchun yaxshiroq ekanini ko‘rsata olmayapti**.

Shu sababli, siz endi metrikaga ishonib bo‘lmaydigan holatdasiz.

Endi sizga **yangi metrikani belgilash va jamoaning maqsadini aniq qayta ta’riflash** zarur.

Masalan, siz yangi metrikada **pornografik tasvirlarni o‘tkazib yuborish holatiga qattiq jazo (penalty)** qo‘shishingiz mumkin.

Men sizga qat’iy ravishda **yangi metrika tanlashni** va bu yangi metrikani asos qilib, **jamoa uchun yangi maqsadni aniq belgilashni** tavsiya qilaman.

Aks holda, noto‘g‘ri metrikaga asoslanib ishslash davom etsa, bu **vaqt va resurs isrof bo‘lishi** mumkin.

Loyihaning o‘rtalarida **dev/test to‘plamlarini yoki metrikani o‘zgartirish odatiy holatdir**.

Dastlabki dev/test to‘plami va metrika — sizga tez iteratsiya qilish imkonini beradi. Ammo agar biror vaqtda siz jamoangiz noto‘g‘ri yo‘nalishda harakat qilayotganini sezsangiz — **buni muammo deb qabul qilmang**.

Shunchaki dev/test to‘plam yoki metrikani o‘zgartiring va bu yangi yo‘nalishni jamoaga ochiq e’lon qiling.

12 Xulosa: Dev va test to‘plamlarini qanday tuzish kerak

- **Dev va test to‘plamlarini** siz kelajakda ishlashni xohlagan va yaxshi natija ko‘rsatishni istagan **real ma’lumotlar oqimiga mos bo‘lgan tarqatmadan tanlang**. Bu tarqatma sizning trening (o‘rgatish) to‘plamingiznikidan farq qilishi mumkin.
- Iloji bo‘lsa, **dev va test to‘plamlarini bir xil tarqatmadan** tanlang. Bu sizga ishonchli va izchil baholash imkonini beradi.
- **Bitta raqamli baholash mezonini (single-number evaluation metric)** tanlang — masalan, aniqlik (accuracy), F1-score yoki boshqa bir ko‘rsatkich. Agar sizga bir nechta maqsad muhim bo‘lsa, ularni **birlashtiruvchi formulaga** (masalan, o‘rtacha qiymat olish) aylantiring yoki **asosiy va qo‘sishimcha metrikalar** sifatida belgilang.
- **Mashina o‘rganish iterativ (takroriy) jarayondir.** Siz qoniqarli natijaga erishish uchun o‘nlab g‘oyalarni sinab ko‘rishingiz mumkin.
- **Dev/test to‘plamlari va bitta raqamli metrikaga ega bo‘lish**, sizga algoritmlarni tezda baholash va shuning hisobiga **tezroq iteratsiya qilish** imkonini beradi.
- **Yangi loyiha** boshlayotganingizda, **bir haftadan kamroq vaqt ichida** dev/test to‘plam va metrikani belgilab oling. Ammo yetuk (**mature**) **tizimlar** uchun bu jarayon uzoqroq davom etishi mumkin — bu normal holat.
- **Eski qoida — 70% trening, 30% test to‘plami** — hozirgi katta hajmli ma’lumotlar davrida doim ham to‘g‘ri ishlamaydi. Siz test va dev to‘plamlarni umumiy ma’lumotning **30% dan kamrog‘i** bilan ham tuzishingiz mumkin.
- **Dev to‘plamingiz algoritmnинг aniqligidagi ahamiyatli farqlarni sezishga yetarli hajmda bo‘lishi kerak**, ammo juda katta bo‘lishi shart emas. **Test to‘plami esa** sizga tizimning yakuniy ishlashini **ishonch bilan baholash imkonini beradigan darajada katta bo‘lishi lozim**.
- Agar sizning dev to‘plamingiz yoki metrikangiz **endi jamoangizni to‘g‘ri yo‘nalishiga yetaklamayotgan** bo‘lsa, ularni darhol yangilang:
 - **Agar dev to‘plamga overfit bo‘lgan bo‘lsangiz**, dev to‘plamga yangi ma’lumotlar qo‘sning.
 - **Agar asl ishlashni istagan tarqatmangiz**, dev/test to‘plamnikidan farq qilsa — **yangilangan to‘plamlar to‘plang**.
 - **Agar metrika endi siz uchun muhim bo‘lgan narsani o‘lchamayotgan bo‘lsa** — yangi metrika belgilang.

Asosiy xatoliklarni tahlil qilish

13 Dastlab oddiy tizim qurib oling, so‘ngra iteratsiya qiling

Siz yangi bir email spamga qarshi tizim (**anti-spam system**) yaratmoqchisiz. Jamoangizda bir nechta g‘oyalar mavjud:

- **Spam xatlarning juda katta o‘quv to‘plamini yig‘ish.** Masalan, siz "honeypot" deb ataladigan usulni qo‘llashingiz mumkin: ya’ni, ataylab soxta email manzillarini ma’lum spammerlarga yuborasiz va ular yuborgan spam xatlarni avtomatik tarzda yig‘asiz.
- **Email matn mazmunini tushunishga qaratilgan xususiyatlar (features) ishlab chiqish.**
- **Email sarlavhasi yoki tashqi ko‘rinishiga oid xususiyatlarni ishlab chiqish,** ya’ni xabar qanday internet serverlar orqali o‘tganini aniqlovchi belgilar.
- va yana boshqa g‘oyalar.

Men shaxsan **anti-spam tizimlar ustida ko‘p ishlagan bo‘lsam-da**, yuqoridagi yo‘nalishlardan qaysi birini birinchi bo‘lib tanlashni aniqlashda qiyngagan bo‘lardim.

Agar siz ushbu sohada mutaxassis bo‘lmansangiz, bu qaror **yanada murakkablashadi**.

Shuning uchun, **boshidanoq “mukammal tizim” tuzishga urinmang**.

Buning o‘rniga, **birinchi oddiy (asosiy) tizimni imkon qadar tez qurib oling** — balki atigi **bir necha kun ichida**.

Garchi bu birinchi tizim hali siz qurishingiz mumkin bo‘lgan "eng yaxshi" tizimdan ancha **yiroq bo‘lsa ham**, bu tizim qanday ishlayotganini kuzatish juda foydali.

Siz tez orada qanday yo‘nalishlarga vaqt va kuch sarflash eng foydali bo‘lishini **ko‘rsatadigan muhim belgilarni (clues) topasiz**.

Quyidagi boblarda siz ushbu belgilarni qanday aniqlash va ular asosida qanday qaror qabul qilish haqida o‘rganasiz.



Ushbu tavsiyalar asosan **AI (sun‘iy intellekt) asosida amaliy ilovalar yaratmoqchi bo‘lgan o‘quvchilar uchun mo‘ljallangan**, ya’ni maqsadi **ilmiy maqola chop etish bo‘limgan** kishilar uchun.

Men keyingi boblarda **ilmiy tadqiqot (research)** bilan shug‘ullanish mavzusiga **yana qaytaman**.

14 Xatolarni tahlil qilish: G‘oyalaringizni baholash uchun dev to‘plamdagи misollarni ko‘zdan kechiring



Siz **mushuklarni aniqlovchi ilovangiz** bilan ishlayotganingizda, ba’zi suratlarda modelning itlarni **mushuk deb noto‘g‘ri tanib olayotganini** payqadingiz. Haqiqatan ham, ba’zi itlar mushuklarga o‘xshaydi!

Jamoangizdagi a’zo quyidagicha g‘oyani ilgari suradi:

Tashqi (3rd party) dasturiy ta’minotni integratsiya qilish orqali modelni **it rasmlarida yaxshilash** mumkin.

Bu o‘zgarishlar **bir oy vaqt oladi**, va jamoa a’zosi bu g‘oyaga juda ishtiyoq bilan qaramoqda.

Siz nima qilishingiz kerak? Shu taklifga rozi bo‘lish kerakmi?

Bunday qarorni qabul qilishdan oldin, men quyidagi yondashuvni tavsiya qilaman:

Avvalo, bu o‘zgarish **algoritmnинг umumiy aniqligiga qancha ta’sir ko‘rsatishini taxmin qiling**.

Shundan so‘ng, bir oyni ushbu ishga sarflashga arziydimi, yoki vaqtini boshqa foydaliroq yo‘nalishga yo‘naltirish ma’qulmi — ancha ongli qaror qabul qilasiz.

Buni qanday amalga oshirish mumkin?

1. **Model noto‘g‘ri tasniflagan 100 ta dev to‘plam misolini yig‘ing.**
Ya’ni, model xatoga yo‘l qo‘yan suratlar namunasi.
2. **Bu misollarni qo‘lda ko‘zdan kechiring va ular orasida nechta it rasmi borligini sanang.**

Bu jarayon "**xatolik tahlili**" deb ataladi.

Agar bu tahlil natijasida aniqlansa-ki, **faqat 5% xatoliklar itlarga tegishli bo‘lsa**, unda itlar bilan ishlovchi algoritmnı yaxshilash orqali **butun tizimdagи xatoliklarning atigi 5% qismini yo‘qotish** mumkin bo‘ladi.

Bu degani, bu loyiha foyda beradigan maksimal chegarasi **5% bo‘ladi**.

Masalan, agar hozirgi model **90% aniqlikda ishlayotgan** bo‘lsa (ya’ni 10% xato qilsa), bu loyiha eng yaxshi holatda aniqlikni **90.5% gacha ko‘tarsa** mumkin (ya’ni 9.5% xatolik).

Bu **5% nisbiy yaxshilanish** bo‘ladi (10% dan 9.5% ga tushgan xatolik).

Ammo...

Agar tahlil natijasida bilinsa-ki, **xatoliklarning 50% qismini itlar tashkil qilayotgan bo‘lsa**, unda bu loyiha **jiddiy ijobiy ta’sir ko‘rsatishi** mumkin.

Bunda aniqlik **90% dan 95% gacha ko‘tarilishi** mumkin (ya’ni 10% xatolikdan 5% xatolikka, bu esa **50% nisbiy yaxshilanish** degani).

Nega bu foydali?

Shunchaki xatoliklarni sanash orqali siz biror loyiha qancha foyda keltirishi mumkinligini aniqroq baholashingiz mumkin.

Bu sizga resurslar va vaqtini qaysi yo'nalishga sarflash haqida **aniq va raqamli asoslangan qaror qabul qilish** imkonini beradi.

E'tibor bering:

Men ko'p muhandislarni ko'rganman — ular xatolik tahlilini o'tkazishga unchalik qiziqlaydi. Ular o'zlarini **qiziqarliroq tuyulgan kod yozish yoki yangi g'oyani implementatsiya qilishga** tashlashadi, xolos.

Bu keng tarqalgan xato: Natijada jamoa bir oy vaqt sarflaydi, lekin oxir-oqibat bu deyarli foyda bermagani ayon bo'ladi.

100 ta misolni ko'zdan kechirish **ko'p vaqt olmaydi**. Hatto har bir rasmga **1 daqiqa** vaqt ajratsangiz ham, **ikki soatda ish tugaydi**.

Bu ikki soat sizga bir oy vaqtingizni tejab qolishi mumkin.

15 Xatolik tahlilida bir nechta g'oyalarni parallel baholash

Sizning jamoangizda **mushuklarni aniqlovchi modelni yaxshilash uchun** bir nechta fikrlar mavjud:

- **Modelning itlarni mushuk deb adashtirish** muammosini tuzatish.
- **Katta mushuklarni** (masalan, arslon, yo'lbars, pantera) oddiy uy mushugiga o'xshatib noto'g'ri tasniflash muammosini hal qilish.
- **Sifatlari chiqmagan** (noaniq, xira) suratlarda aniqlikni oshirish.
- va hokazo.

Ushbu barcha g'oyalarni parallel tarzda tahlil qilish mumkin.

Men odatda bir Excel fayli (yoki Google Sheets) ochaman va ~100 ta dev to'plamdagiga noto'g'ri tasniflangan suratlarni ko'zdan kechirganimda shu faylga **katakcha ko'rinishida belgi qo'yib boraman**.

Shuningdek, kerakli joylarda **qisqa izohlar** yozib boraman — bu ma'lum suratlar haqida eslab qolishga yordam beradi.

Quyida ushbu jarayon qanday ishlashini ko'rsatadigan kichik misol bilan tanishing. Faraz qilaylik, bizda **to'rtta noto'g'ri tasniflangan rasm** mavjud:

| Image | Dog | Great cat | Blurry | Comments |
|------------|-----|-----------|--------|---|
| 1 | ✓ | | | Unusual pitbull color |
| 2 | | | ✓ | |
| 3 | | ✓ | ✓ | Lion; picture taken at zoo on rainy day |
| 4 | | ✓ | | Panther behind tree |
| % of total | 25% | 50% | 50% | |

Rasm 3: Bir nechta ustunlar belgilangan bo'lishi mumkin

Yuqoridagi 3-rasmda **yovvoyi mushuk** (Great Cat) va **xira surat** (Blurry) ustunlari **ikkalasi ham belgilangan**.

Bu misol shuni ko`rsatadiki, **bitta surat bir nechta kategoriya** bilan bog`liq bo`lishi mumkin. Shu sababli, jadval ostida chiqarilgan foizlar **yig`indisi 100% bo`lmasligi** mumkin. Chunki bitta surat bir nechta muammo turiga tegishli bo`lishi mumkin.

Amaliy tajriba shuni ko`rsatadiki:

Garchi siz dastlab kategoriya nomlarini (masalan, **It**, **Yovvoyi mushuk**, **Xira surat**) o`ylab topib, so`ngra suratlarni qo`lda shu bo`yicha tasniflamoqchi bo`lsangiz ham, **real holatda bu ro`yxat kengayadi**.

Masalan, siz 10–12 ta rasmni tahlil qilar ekansiz, shuni sezishingiz mumkinki, **ko`plab xatolar Instagram filtrlangan suratlarda ro`y bermoqda**.

Shunda siz jadvalga yangi ustun — "Instagram" degan ustunni qo`shib qo`yishingiz mumkin. Bunday yangi ustunlar yordamida siz tizimda **qaysi muammolar ko`p uchrayotganini** aniqroq tushunasiz va ustuvor yo`nalishlarni belgilash osonlashadi.

Ko`p hollarda siz suratga inson sifatida **to`g`ri baho bera olasizmi-yo`qmi**, deb o`ylash orqali **yangi xatolik kategoriylarini** va ularga yechimlarni ham o`ylab topasiz.

Eng foydali xatolik turlari — ular ustida ishslash rejangiz bo`lganlaridir.

Masalan, agar sizda **Instagram filtrlarini bekor qilish va asl suratni tiklash** g`oyangiz bo`lsa, unda "Instagram" kategoriyasini ayniqsa foydali bo`ladi.

Lekin siz o`zingiz yechim topolmaydigan muammolarni ham kategoriya sifatida belgilashingiz mumkin — bu tahlilning maqsadi **eng istiqbolli yo`nalishlarni tushunib olish**.

Xatolik tahlili — bu iterativ (bosqichma-bosqich) jarayon

Agar siz boshida hech qanday kategoriya bilmasangiz — **xavotir olmang**.

Birinchi bir nechta rasmga qarab chiqqaningizda, sizda allaqachon **bir nechta muammo toifalari** haqida g`oyalar tug`iladi.

Keyin siz yana rasm tahlil qilishda davom etar ekansiz, **yana yangi kategoriylar** xayolingizga keladi.

Bu holatda siz ilgari ko`rilgan suratlarga qaytib, ularni **yangi kategoriylar nuqtai nazaridan qayta ko`zdan kechirishingiz** mumkin.

| Image | Dog | Great cat | Blurry | Comments |
|------------|-----|-----------|--------|---|
| 1 | ✓ | | | Usual pitbull color |
| 2 | | | ✓ | |
| 3 | | ✓ | ✓ | Lion; picture taken at zoo on rainy day |
| 4 | | ✓ | | Panther behind tree |
| ... | ... | ... | ... | ... |
| % of total | 8% | 43% | 61% | |

Endi siz "**It**" (**Dog**) xatolari ustida ishslash orqali eng ko`pi bilan **8% xatolikni kamaytirishingiz** mumkinligini bilasiz.

Lekin **Yovvoyi mushuklar** (**Great Cat**) yoki **xira suratlar** (**Blurry images**) ustida ishslash orqali **ko`proq xatolarni bartaraf etish** mumkin.

Shu sababli, siz e'tiboringizni **ikki oxirgi kategoriya** — yovvoyi mushuklar yoki xira suratlar — dan biriga qaratishni tanlashingiz mumkin.

Agar sizning jamoangizda yetarli odam bo'lsa, **bir nechta yo'nalishda parallel ravishda ishslash** ham mumkin:

- Masalan, ba'zi muhandislar **yovvoyi mushuklar** bilan ishslashsin,
- Boshqalari esa **xira suratlar** ustida optimallashtirish ishlarini olib borishsin.

16 Ishonchsiz (noto'g'ri belgilangan) dev va test to'plam namunalarini tozalash

Xatolik tahlili jarayonida siz dev to'plamingizdagi ayrim namunalar noto'g'ri belgilanganligini payqashingiz mumkin.

Bu yerda “**noto'g'ri belgilangan**” deyayotganimizda, bu:

- Suratni algoritm ko'rib chiqishidan **oldin** inson tomonidan **xato belgi qo'yilganini** bildiradi.
- Ya'ni, misolda berilgan (x, y) juftligida, y — ya'ni **klass yorlig'i noto'g'ri** bo'lgan bo'ladi.

Masalan:

- Ba'zi **mushuk bo'lмаган suratlar** "mushuk" deb belgilangan bo'lishi mumkin,
- Yoki aksincha, mushuk suratlari "mushuk emas" deb belgilangan bo'lishi mumkin.

Agar siz **bunday xatoliklar (noto'g'ri belgilanishlar)** soni e'tiborga molik darajada ko'p deb hisoblasangiz,

shunchaki xatolik tahlili jadvalingizga “**noto'g'ri belgilangan**” (mislabeled) degan alohida **ustun qo'shing**.

| Image | Dog | Great cat | Blurry | Mislabeled | Comments |
|------------|-----|-----------|--------|------------|-----------------------------------|
| ... | | | | | |
| 98 | | | | ✓ | Labeler missed cat in background |
| 99 | | ✓ | | | |
| 100 | | | | ✓ | Drawing of a cat; not a real cat. |
| % of total | 8% | 43% | 61% | 6% | |

Dev to'plamdagи noto'g'ri belgilangan misollarni tuzatish kerakmi?

Eslab qoling: **Dev to'plamning asosiy maqsadi** — turli algoritmlarni tezda baholab, masalan, **A algoritmi B algoritmidan yaxshimi yo'qmi** — shuni aniqlashga yordam berishdir. Agar **dev to'plamdagи noto'g'ri belgilanishlar soni** shunchalik ko'PKI, bu sizga algoritmlarni taqqoslashda xalaqt berayotgan bo'lsa, unda **noto'g'ri belgilangan misollarni tuzatish uchun vaqt ajratish mantiqiy** bo'ladi.

Masalan, sizning klassifikatoringiz quyidagi natijalarni ko‘rsatyapti:

- **Dev to‘plamdagи umumiу aniqlik:** 90%
(demak, 10% xatolik mavjud)
- **Noto‘g‘ri belgilangan misollar tufayli yuzaga kelgan xatolar:** 0.6%
(bu — dev to‘plamdagи barcha xatoliklarning 6% ini tashkil qiladi)
- **Boshqa sabablarga ko‘ra yuzaga kelgan xatolar:** 9.4%
(bu — dev to‘plamdagи xatolarning 94% ini tashkil qiladi)

Noto‘g‘ri belgilangan misollarni tuzatishning foydasi qanday o‘zgaradi?

Tasavvur qiling, sizning klassifikatoringiz quyidagi natijalarni ko‘rsatmoqda:

- **Dev to‘plamdagи umumiу aniqlik:** 98.0%
(ya’ni 2.0% umumiу xatolik)
- **Noto‘g‘ri belgilangan misollar tufayli yuzaga kelgan xatolik:** 0.6%
(ya’ni bu — dev to‘plamdagи barcha xatolarning 30% ini tashkil qiladi)
- **Boshqa sabablarga ko‘ra yuzaga kelgan xatolik:** 1.4%
(ya’ni 70% xatolik boshqa sabablar bilan bog‘liq)

Agar xatolarning **30%** i **noto‘g‘ri belgilangan misollar** tufayli yuzaga kelayotgan bo‘lsa, bu sizning **aniqlik baholaringizga** sezilarli miqdorda xato qo‘shadi.

Shunday ekan, **dev to‘plamdagи etiketkalar sifatini oshirishga** vaqt ajratish o‘zini oqlaydi.

Aynan shu yondashuv sizga quyidagi savolga aniq javob topishga yordam beradi:

Klassifikator xatoligi aslida 2.0% mi yoki 1.4% mi?

Bu esa **nisbatan katta farqdir**.

Ko‘p holatlarda loyiha boshida **ba’zi noto‘g‘ri belgilanishlarga** ko‘z yumuladi.

Ammo tizim yaxshilangan sari, **noto‘g‘ri belgilangan misollar soni umumiу xatolar ichida ko‘proq ulushga ega bo‘la boshlaydi**,

va bu holatda ularni to‘g‘rilash muhimlashadi.

Avvalgi boblarda siz, masalan, “Dog”, “Great Cat” va “Blurry” kabi xatolik toifalarini **algoritmni yaxshilash orqali** kamaytirishni o‘rgandingiz.

Bu bob esa sizga yana bir muhim xatolik toifasini ko‘rsatdi:

Mislabeled — noto‘g‘ri belgilangan misollar.

Bu toifani **ma’lumotlar sifatini oshirish orqali** yaxshilash mumkin.

Agar siz dev to‘plamdagи etiketkalarni tuzatishga qaror qilsangiz, **xuddi shu jarayonni test to‘plamga ham qo‘llang**.

Bu bilan siz **dev va test to‘plamlar bir xil taqsimotdan olinganligiga** ishonch hosil qilasiz.

Bu esa 6-bobda aytib o‘tilgan muammoning oldini oladi:

Ya’ni, **jamoangiz dev to‘plam bo‘yicha optimallashtirishni bajargan bo‘lsa-da**, test to‘plam umuman boshqa taqsimot asosida baho berayotgan bo‘lishi mumkin — bu esa butun ishni chippakka chiqaradi.

Agar siz dev/test to‘plamdagи etiketkalarni yaxshilashga qaror qilsangiz, **faqat noto‘g‘ri tasniflangan misollarni emas**,

to‘g‘ri tasniflanganlarini ham tekshirishni ko‘rib chiqing.

Ba'zi hollarda **na asl etiketka, na sizning model to‘g‘ri bo‘lishi mumkin.**
 Agar siz faqat noto‘g‘ri tasniflangan misollarni to‘g‘rilasangiz,
baholash jarayoniga noxolislik (bias) kiritishingiz mumkin.
 Agar sizda **1,000 ta dev to‘plam misoli** bo‘lsa,
 va klassifikator **98.0% aniqlik** ko‘rsatayotgan bo‘lsa,
 demak siz faqat **20 ta noto‘g‘ri tasniflangan misollarni ko‘rib chiqasiz.**
 Bu juda qulay, ammo **faqat noto‘g‘ri misollarni tekshirish baholashda bias (noaniqlik)**
kiritadi.
 Bu **amaliy loyihalarda unchalik katta muammo emas,**
 lekin ilmiy maqola yozilayotgan bo‘lsa yoki aniq, xolis test aniqligi kerak bo‘lsa, bu kuchli
salbiy ta’sir ko‘rsatadi.

17 Agar sizda katta dev to‘plam bo‘lsa, uni ikkita kichik bo‘limga ajrating — ulardan faqat birini ko‘ring

Tasavvur qiling, sizda **5,000 ta misoldan iborat katta dev to‘plam** bor.

Modelingiz ushbu to‘plamda **20% xatolik** qilmoqda.

Demak, taxminan **1,000 ta rasm noto‘g‘ri tasniflangan.**

Ammo bu 1,000 ta rasmni **qo‘lda ko‘rib chiqish juda ko‘p vaqt oladi.**

Shunday bo‘lsa, **barchasini ko‘rmaslikka qaror qilish** mumkin.

Bu kabi holatlarda, men quyidagicha ish tutishni tavsiya qilaman:

Dev to‘plamni ikkita kichik bo‘limga ajrating:

1. **Tahlil qilinadigan bo‘lim** — siz qo‘lda ko‘rib chiqadigan qismini tashkil qiladi.
2. **Ko‘rilmaydigan bo‘lim** — siz hech qachon ko‘rib chiqmaydigan (ya’ni qo‘lda tahlil qilinmaydigan) qismi bo‘ladi.

Nega bunday qilinadi?

Siz **qo‘lda ko‘rib chiqqan bo‘limga tezroq overfit (haddan tashqari moslashish)** qilasiz.

Ushbu qismdan **xatoliklar tahlili uchun** foydalanasiz.

Lekin **parametrlarni sozlash** yoki **algoritmnini baholash** uchun siz **ko‘rilmagan dev bo‘limi** bilan ishlashingiz kerak bo‘ladi.

Bu sizga **haqiqiy va xolis baho beradi.**

Shunday qilib, **bir bo‘limdan o‘rganasiz,**
ikkinchisi esa sizning o‘rganishingizga xolislik bilan baho beradi.



Keling, oldingi misolimizni davom ettiraylik...

Avval aytiganidek, **algoritm 5,000 ta dev to‘plam misollarining 1,000 tasini noto‘g‘ri tasniflamoqda.**

Endi biz bu 1,000 ta xatolikdan **taxminan 100 tasini qo'lida tekshirib chiqmoqchimiz**, ya'ni **xatoliklarning 10 foizini**.

Buning uchun quyidagicha harakat qiling:

Dev to'plamdan 10% ni tasodifiy tanlab oling — bu **500 ta misol** bo'ladi.

Bu kichik to'plamni "**Ko'z bilan tekshiriladigan dev to'plam**" deb nomlaymiz (inglizchasiga: *Eyeball dev set*) — bu nom, uni biz **ko'z bilan** (ya'ni qo'lida) tekshirayotganimizni eslatib turadi.

Masalan, agar siz **nutqni aniqlash (speech recognition)** ustida ishlayotgan bo'lsangiz, siz bu to'plamni "**Ear dev set**" deb ham atashingiz mumkin edi — chunki siz undagi **audio** yozuvlarni qulqoq bilan eshitasziz.

Demak, **Eyeball dev to'plamida 500 ta misol** bor.

Kutilishicha, shundan **taxminan 100 tasi noto'g'ri tasniflangan** bo'ladi — ya'ni siz aynan shu 100 ta xatolikni qo'lida tahlil qilasiz.

Dev to'plamning qolgan **4,500 ta misolini** esa boshqa joyga joylashtiring.

Bu qismini "**Qora Quti dev to'plami**" deb ataymiz (*Blackbox dev set*).

Bu to'plam sizga quyidagi ishlar uchun xizmat qiladi:

- Algoritmlarning **aniqlik yoki xatolik ko'rsatkichlarini avtomatik o'lchash**;
- **Model tanlash yoki giperparametrлarni sozlash**.

Ammo muhim qoidaga amal qiling:

👉 **Bu to'plamga hech qachon ko'z bilan qarash kerak emas.**

Biz bunga "**Blackbox**" (ya'ni **qora quti**) deb nom beramiz — chunki biz undan **faqat avtomatlashtirilgan baholash** uchun foydalanamiz, va u orqali algoritm qanday ishlayotganini **xolis** baholaymiz.



Nega dev to'plamni aniq qilib Ko'z bilan tekshiriladigan (Eyeball) va Qora quti (Blackbox) qismlarga ajratamiz?

Bu ajratish juda muhim, chunki:

Siz Eyeball dev to'plamdagи misollarni qo'lida ko'rganingiz uchun, ular bo'yicha sezgi va tushuncha hosil qilasiz.

Buning oqibatida:

Siz o'z algoritmingizni tezroq aynan shu ko'rilgan misollar uchun yaxshilay boshlaysiz.

Bu esa, algoritmnинг Eyeball dev to'plamdagи anqligli tez ko'tarilishiga olib keladi.

Lekin **Blackbox dev to'plamida hech qanday real yaxshilanish ko'zga tashlanmaydi**.

Bu holatda demak, siz **Eyeball dev to'plamga haddan tashqari moslashib ketgansiz** — ya'ni siz **Eyeball dev to'plamga overfit bo'lgansiz**.

18 Ko‘z bilan ko‘riladigan (Eyeball) va Qora quti (Blackbox) dev to‘plamlar qanchalik katta bo‘lishi kerak?



Sizning **Eyeball dev to‘plamingiz** — ya’ni qo‘lda ko‘rib chiqiladigan namunalar to‘plami — algoritmingizda **eng muhim xatolik turlarini aniqlash** uchun yetarli miqdorda bo‘lishi kerak. Agar siz insonlar osonlikcha bajara oladigan vazifada ishlayotgan bo‘lsangiz (masalan, rasmda mushukni tanish), quyidagi taxminiy ko‘rsatmalarga amal qilishingiz mumkin:

- **Agar klassifikator atigi 10 ta xatolik qilgan bo‘lsa**, bu Eyeball to‘plam juda **kichik** hisoblanadi.
Faqat 10 ta xatoga asoslanib, siz muhim xatolik turlarini aniqlashda qiynalasiz.
Ammo agar sizda juda ozgina ma’lumot bo‘lsa, shunchasi ham foyda beradi va bu sizga qaysi yo‘nalishlar ustida ishlashni tanlashda yordam beradi.
- **20 ta xatolik bo‘lsa**, siz **asosiy xatolik manbalarini taxminan anglashni** boshlaysiz.
- **50 ta xatolik bo‘lsa**, siz **eng muhim xatolik turlarini yaxshi tushuna** boshlaysiz.
- **100 ta xatolik** — bu juda yaxshi hajm. Bu miqdor sizga eng katta xatolik manbalarini aniqlashda **mustahkam tushuncha** beradi.

Men hatto **qo‘lda 500 tagacha xatolikni tahlil qilgan** muhandislarni ko‘rganman. Agar sizda yetarlicha ma’lumot bo‘lsa, bunday katta miqdorni tahlil qilishda hech qanday zarar yo‘q.

Masalan:

Agar sizning klassifikatingizda **xatolik darajasi 5%** bo‘lsa va siz **100 ta noto‘g‘ri tasniflangan rasmni** Eyeball dev to‘plamda ko‘rmoqchi bo‘lsangiz:

Demak, sizga kamida **2000 ta rasm kerak** bo‘ladi (chunki $5\% \text{ xatolik} \times 2000 = 100 \text{ ta xato}$).

👉 Shunday qilib, **xatolik darajasi qancha past bo‘lsa**,
shuncha **ko‘proq misol** kerak bo‘ladi,
toki yetarli miqdorda xato to‘planib, ularni tahlil qilish mumkin bo‘lsin.

Ammo...

Agar siz shunday muammoni yechayotgan bo‘lsangizki, **insónlar ham uni yaxshi bajara olmasalar**,

bunday Eyeball tahlil unchalik foydali bo‘lmaydi.

Chunki bunday vaziyatlarda, algoritm nimaga xato qilganini aniqlashning o‘zi qiyin.

Bunday murakkab hollarda Eyeball dev to‘plam tuzishni **o‘tkazib yuborish mumkin**.

Biz bu turdagи holatlar uchun qanday yondashuvlar qo‘llash kerakligini keyingi boblarda muhokama qilamiz.



Qora quti (Blackbox) dev to‘plami qanchalik katta bo‘lishi kerak?

Avvalgi boblarda aytganimizdek, **1,000 dan 10,000 gacha misollar** o‘z ichiga olgan dev to‘plamlar odatiy hol sanaladi.

Endi bu fikrni aniqroq qilib olsak:

Blackbox dev to‘plami 1,000–10,000 misoldan iborat bo‘lsa, bu odatda yetarli bo‘ladi — model parametrlarini sozlash va turli modellardan eng yaxshisini tanlash uchun.

Albatta, **yana ko‘proq ma’lumotga ega bo‘lish zarar qilmaydi.**

Agar sizning Blackbox dev to‘plamingizda **faqat 100 ta misol** bo‘lsa, bu juda kichik hisoblanadi,

lekin **shunda ham foydali** bo‘lishi mumkin.

Agar sizda juda kichik dev to‘plam bo‘lsa...

Shunda sizda Eyeball va Blackbox dev to‘plamlarini **ikkiga bo‘lishga imkon bo‘lmaydi**, chunki har ikkalasini yetarli darajada katta qilishning iloji yo‘q.

Bunday holatda, **barcha dev to‘plamni Eyeball dev to‘plam sifatida ishlatalishga to‘g‘ri keladi**

ya’ni siz qo‘lda **hamma misollarni ko‘rib chiqishingiz** kerak bo‘ladi.

Eyeball dev to‘plam muhimroqmi?

Ha. Agar siz shunday vazifada ishlayotgan bo‘lsangizki, **insonlar bu muammoni yaxshi hal qila olsa**,

va **misollarni ko‘rish orqali intuitiv tushuncha hosil qilishingiz mumkin bo‘lsa**, unda:

Eyeball dev to‘plam — ya’ni qo‘lda tahlil qilinadigan dev to‘plam — eng muhim hisoblanadi.

Agar sizda faqat Eyeball dev to‘plam bo‘lsa, quyidagilarni aynan shu to‘plamda bajarsangiz ham bo‘ladi:

- **Xatolar tahlili** (error analysis)
- **Model tanlash** (model selection)
- **Gipermetaparlarni sozlash** (hyperparameter tuning)

Kamchilik shundaki, shunday yondashuvda siz **dev to‘plamga haddan ortiq moslashib** (ya’ni overfit qilib) yuborishingiz mumkin.

Agar sizda ko‘p ma’lumot bo‘lsa...

Bu holatda, Eyeball dev to‘plam hajmi:

Siz qo‘lda necha misolni ko‘rib chiqishga vaqt ajrata olishingizga bog‘liq bo‘ladi.

Men amalda **1,000 tadan ko‘p xatolikni** qo‘lda tahlil qilgan muhandislarni juda kam ko‘rganman.

19 Asosiy xatoliklarni tahlil qilish bo'yicha xulosalar

- Agar siz yangi loyiha boshlayotgan bo'lsangiz, ayniqsa bu soha sizga tanish bo'lmasa,
 - qaysi yo'naliш eng samarali bo'lishini oldindan to'g'ri taxmin qilish juda qiyin.**
 - Shuning uchun, ishni boshlaganda "ideal tizim"ni loyihalash va qurishga urinish shart emas.
- Buning o'rniga, bir necha kun ichida oddiy bir modelni qurib, uni o'qiting. Keyin esa, **xatoliklarni tahlil qilish** orqali eng istiqbolli yo'naliшlarni aniqlang va shu asosda modelni bosqichma-bosqich yaxshilang.
- **Xatolarni tahlil qilish** — bu model noto'g'ri baholagan **taxminan 100 ta dev set misollarini qo'lда ko'rib chiqish**
va asosiy xatolik turlarini hisoblashdan iborat.
Olingan ma'lumotlar asosida **qanday xatolarni tuzatishga ustuvorlik berishni aniqlang**.
 - **Dev setni ikkiga bo'lish** haqida o'ylab ko'ring:
 - **Eyeball dev set** — siz qo'lда ko'rib chiqadigan misollar jamlanmasi.
 - **Blackbox dev set** — siz qo'lда ko'rmaydigan, faqat model bahosi uchun foydalilanladigan jamlanma.
Agar Eyeball dev setdagi natijalar Blackbox dev setdagidan ancha yuqori bo'lsa,
demak siz **Eyeball to'plamga haddan ortiq moslashib (overfit) qolgansiz**
va uni yangilash yoki kengaytirishni o'ylab ko'rishingiz kerak bo'ladi.
 - **Eyeball dev set** shunchalik katta bo'lishi kerakki, unda siz tahlil qiladigan yeterli miqdorda noto'g'ri tasniflangan misollar bo'lsin.
Aksincha, **Blackbox dev set** uchun 1,000 dan 10,000 gacha misol ko'plab ilovalar uchun yetarli bo'ladi.
 - Agar sizning dev setingiz bu kabi ikkiga bo'lish uchun **etarlik darajada katta bo'lmasa**,
dev setning butun qismini Eyeball dev set sifatida ishlating —
va unda xatolar tahlili, model tanlash, hamda gipermetaparlarni sozlash ishlarini bajaring.

Bias va Variance (Og‘ish va Tarqalish)

20 Bias va Variance: Xatolikning ikki asosiy manbasi

Faraz qilaylik, sizning **trening**, **dev**, va **test** to‘plamlaringiz barchasi **bir xil taqsimotdan olingan**. Shunday bo‘lsa, siz har doim yana ko‘proq trening ma’lumotlari to‘plashga harakat qilishingiz kerakmi? Axir, ko‘proq ma’lumot hech qachon zarar qilmaydi, to‘g‘rimi? Aslida, ha — ko‘proq ma’lumot zarar qilmaydi. **Lekin afsuski, bu har doim yordam ham bermaydi.** Ba’zan yangi ma’lumot yig‘ish — vaqt va resursning behuda sarfi bo‘lishi mumkin. Demak, **qachon yangi ma’lumot qo‘sish kerak**, va **qachon bunga arzimaydi** — buni qanday aniqlaysiz?

Bu savolga javob berish uchun, mashina o‘rganishda xatolarning ikki asosiy manbasi borligini tushunib olishimiz kerak: **bias (og‘ish)** va **variance (tarqalish)**. Ularni tushunish sizga modelni yaxshilashning eng samarali yo‘llarini tanlashga yordam beradi — xususan, yangi ma’lumot kerakmi yoki yo‘q.

Masalan ko‘raylik:

Siz **mushuklarni aniqlovchi** model yasayapsiz va uni 5% xatolik bilan ishlashini xohlaysiz.

Ayni paytda sizda quyidagi ko‘rsatkichlar mavjud:

- **Trening to‘plamdagisi xatolik:** 15%
- **Dev to‘plamdagisi xatolik:** 16%

Bu yerda, siz yangi trening ma’lumotlari qo‘sangiz ham, natijani unchalik yaxshilay olmaysiz.

Aslida, trening ma’lumotlarini ko‘paytirish, model uchun treningda yaxshi ishlashni yanada qiyinlashtiradi. (Bu nega shunday bo‘lishini keyingi boblarda ko‘rib chiqamiz.)

Agar sizning model **treningda 85% aniqlik (15% xatolik)** ko‘rsatayotgan bo‘lsa, lekin siz **95% aniqlik (5% xatolik)** ni istasangiz, demak birinchi navbatda **treningdagi ishlashni yaxshilash kerak**.

Shuni esda tuting: **Dev yoki test to‘plamlardagi ishlash odatda treningdagidan yomonroq bo‘ladi.**

Agar siz model ko‘rgan misollarda 85% aniqlik olayotgan bo‘lsangiz, u holda hali ko‘rmagan misollarda 95% aniqlikka erishishingiz deyarli imkonsiz.

Bias va Variance qanday aniqlanadi?

Yuqoridagi misolni olaylik:

- **Dev to‘plamdagisi umumiyligi xatolik:** 16%
 - Buni ikkiga ajratamiz:
 1. **Treningdagi xatolik (bias):** 15%
 2. **Dev/testdagi qo‘sishma xatolik (variance):** 1%
- Bu — modelning **asosiy naqshlarni tushunmayotganligi**, ya’ni **bias (og‘ish)**.
 Ya’ni, model treningdan dev/testga o‘tganda biroz yomonroq ishlayapti — bu **variance (tarqalish)**.

Statistika sohasida **bias** (og‘ish) va **variance** (tarqalish) tushunchalarining **aniq matematik tariflari mavjud**, lekin biz bu erda bunga chuqur kirib bormaymiz.

Taxminan aytganda:

- **Bias** — juda katta o‘quv to‘plamga ega bo‘lganda, sizning algoritmingizning **trening to‘plamdagisi xatolik darajasi**.
- **Variance** esa — shu holatda, **test to‘plamda algoritm qanchalik yomonroq ishlashi**, ya’ni treningdan qanchalik uzoqlashgani.

Tarjimon: Sh. Ne’matov

Tahrirchi: A.Dadajonov

Agar sizning **xatolik** o‘lchov mezoningiz "mean squared error" (o‘rtacha kvadratik xatolik) bo‘lsa, bu ikki komponentni matematik formulalar bilan aniqlash va quyidagi ifodani isbotlash mumkin:

Umumiy xatolik = Bias + Variance

Lekin bu kitobda bizga bunday formal ta’riflar shart emas. Biz uchun, **ML muammosida qanday qilib tezroq natijaga erishishni hal qilishda bu yerda keltirilgan norasmiy ta’riflar yetarli** bo‘ladi.

Ba’zi metodlar mavjudki, ular **bir vaqtning o‘zida ham bias, ham variance ni kamaytiradi**. Bunday metodlar odatda **algoritm arxitekturasida tub o‘zgarishlar** qilishni talab qiladi. Biroq, **bu yechimlarni aniqlash va amalda qo‘llash ancha murakkab** bo‘ladi.

21 Bias va Variance misollari

Keling, mushuklarni aniqlovchi model (klassifikator) vazifasini ko‘rib chiqamiz. Ushbu

topshiriqda “ideal” **klassifikator** (masalan, inson) deyarli mukammal natijaga erisha oladi.

Faraz qilaylik, sizning algoritmingiz quyidagicha ishlaydi:

- **Treningdagi xatolik (error)** = 1%
- **Ishlab chiqish (dev) to‘plamidagi xatolik** = 11%

Bu holatda qanday muammo bor? Oldingi bobdag'i ta'riflarga asoslanib:

- Bias (ya’ni, modelning treningdagi xatoligi) $\approx 1\%$
- Variance (ya’ni, dev to‘plamda yomonlashish darajasi) = $11\% - 1\% = 10\%$

Demak, bu modelda **yuqori variance (tarqalish)** mavjud. Klassifikator treningda juda yaxshi ishlayapti, lekin dev to‘plamga nisbatan **umumlashtira olmayapti**. Bunga **ortiqcha moslashish (overfitting)** deyiladi.

Endi quyidagini ko‘raylik:

- **Treningdagi xatolik** = 15%
- **Dev xatoligi** = 16%

Bu yerda:

- Bias $\approx 15\%$
- Variance = 1%

Bu model trening ma'lumotlariga ham yaxshi moslasha olmagan — **yuqori bias** mavjud. Biroq, dev to‘plamidagi xatolik unga juda yaqin, shuning uchun variance **past**. Bunday algoritm **kam moslashgan (underfitting)** deyiladi.

Yana bir misol:

- **Treningdagi xatolik** = 15%
- **Dev xatoligi** = 30%

Bu yerda:

- Bias = 15%
- Variance = 15%

Bu holatda model **ham treningda yomon ishlayapti (yuqori bias), ham devda yanada yomonroq ishlayapti (yuqori variance)**. Ya’ni, **bir vaqtning o‘zida ham underfitting, ham overfitting holati mavjud**. Bunday holatda "qaysi biri" degan savolga aniq javob berish qiyin.

Oxirgi misol:

- **Trening xatoligi** = 0.5%
- **Dev xatoligi** = 1%

Bu — a'lo natija! Model **treningda ham, devda ham** juda yaxshi ishlayapti:

- **Past bias**
- **Past variance**

22 Optimal xatolik darajasi bilan solishtirish

Mushuklarni aniqlovchi dasturimiz misolida, “ideal” xatolik darajasi deyarli **0%** deb hisoblanadi. Inson suratga qarab, unda mushuk bor yoki yo‘qligini deyarli har doim aniqlay oladi. Demak, **kompyuter ham shunga yaqin ishlashi mumkin** degan umid bor.

Biroq ba'zi muammolar bundan ancha murakkabroq. Masalan, **nutqni tanish (speech recognition)** tizimini ishlab chiqayotgan bo‘lsangiz, 14% audio yozuvlarda shunchalik ko‘p fon shovqini yoki tushunarsiz nutq bo‘lishi mumkin — hattoki inson ham nima deyilganini aniqlay olmaydi. Bu holatda, **eng mukammal** nutqni aniqlash tizimi ham **kamida 14% xatolik qiladi**. Endi tasavvur qiling, siz ishlab chiqayotgan tizim quyidagi natijalarini ko‘rsatdi:

- **Trening xatoligi** = 15%
- **Dev xatoligi** = 30%

Bu yerda, trening to‘plamdagи natijalar deyarli optimal xatolik darajasi (14%) bilan teng. Ya’ni:

- Treningda ko‘p yaxshilanish imkoniyati yo‘q → **bias past**
- Biroq dev to‘plamda juda yomon natija → **variance yuqori**

Bu holat, avvalgi bobdagи uchinchi misolga o‘xshaydi (15% training, 30% dev). Ammo agar **optimal xatolik 0%** deb hisoblasak, 15% training xatoligi juda yomon ko‘rinadi — bu **biasni kamaytirish kerak** degan ma’noni beradi.

Agar optimal xatolik 14% bo‘lsa, 15% training xatoligi deyarli optimalga yaqin bo‘lib qoladi — **bias kamaytiriladigan holat emas, balki variancega e’tibor qaratish kerak**.

Shunday hollarda xatolikni quyidagicha tahlil qilish mumkin:

- **Optimal xatolik** (ya’ni, "qochib bo‘lmaydigan bias"): 14%
- **Qochish mumkin bo‘lgan bias (avoidable bias)**: 1% → bu: 15% (training) – 14% (optimal)
- **Variance**: 15% → bu: 30% (dev) – 15% (training)

Bias = Optimal xatolik + Qochish mumkin bo'lgan bias

Bu formulada:

- **Optimal xatolik** — eng yaxshi algoritm ham yo'q qila olmaydigan xatolik
- **Qochish mumkin bo'lgan bias** — sizning algoritmingizda, trening to'plamda qolayotgan "yo'qotish"
- **Variance** esa — modelingiz dev/test to'plamga qanday umumlashtira olmasligini ko'rsatadi

Variance har doim kamaytirilishi mumkin (nazariy jihatdan), chunki **katta trening to'plam** bilan variance deyarli **0%** ga tushiriladi. Shuning uchun variance **doim "qochish mumkin"** hisoblanadi, "**qochib bo'lmaydigan variance**" degan narsa yo'q.

Yana bitta misol:

- **Optimal xatolik:** 14%
- **Training xatolik:** 15%
- **Dev xatolik:** 16%

Bu yerda:

- **Qochish mumkin bo'lgan bias:** 1%
- **Variance:** 1%
- Umumiyl xatolik optimaldan atigi 2% yomon → bu juda yaxshi ko'rsatkich!

23 Bias va Variance muammolarini qanday hal qilish mumkin

Mana eng sodda qoida:

- **Agar sizda katta (yuqori) bias bo'lsa**, ya'ni modelingiz treningda ham yaxshi ishlamasma:
 - **Model hajmini oshiring.**
Masalan: neyron tarmoqni kattalashtiring — qatlamlar yoki neyronlar sonini ko'paytiring.
 - **Agar sizda katta (yuksak) variance bo'lsa**, ya'ni model treningni yaxshi o'rgansa-yu, testda yomon ishlasa:
 - **Trening ma'lumotlarini ko'paytiring.**
Ko'proq ma'lumot modelni umumlashtirishga (generalizatsiya) o'rgatadi.

Bu ikkalasi juda kuchli vosita:

Agar siz:

- Model hajmini cheksiz oshira olsangiz (masalan, ko‘proq qatlam qo‘shsangiz),
 - Trening ma’lumotlarini ko‘proq to‘play olsangiz,
- ...ko‘plab ML muammolarida **juda yaxshi natijalarga erishish mumkin.**

Ammo amaliyotda cheklovlar bor:

- **Juda katta model** ishlatalish hisoblashni **sekinlashtiradi** va ko‘p resurs talab qiladi.
- Ko‘p ma’lumot **har doim ham mavjud emas**. Masalan: internetda ham cheklangan mushuk rasmlari bor!

Boshqa arxitekturalar haqida nima deyish mumkin?

Har xil model arxitekturasi (ayniqsa neyron tarmoqlarda) har xil bias/variance balansi beradi. Bugungi **deep learning tadqiqotlari** bu borada ko‘plab innovatsion yondashuvlarni taqdim qilmoqda.

Agar siz **neyron tarmoqlar bilan ishlayotgan** bo‘lsangiz:

- **Akademik maqolalar** sizga ilhom beradi.
- **GitHubda** juda ko‘p ochiq manba kodlar mavjud.

Ammo bu arxitekturalar natijasi **oldindan aniq bo‘lmaydi**.

Katta model + regularizatsiya = xavfsiz yo‘l

Model hajmini oshirsangiz, bu odatda **biasni kamaytiradi**, ya’ni treningda yaxshilanadi. Ammo bu **variance-ni oshirib**, **ortiqcha moslashuv (overfitting)**ga olib kelishi mumkin. Buni oldini olish uchun:

- **Yaxshi regularizatsiya** ishlating:
 - Masalan, **L2 regularizatsiya**
 - Yoki **dropout**
 - Yoki boshqa usullar

Agar siz **regularizatsiyani to‘g‘ri tanlasangiz**, model hajmini oshirish **xavfsiz** va foydali bo‘ladi.

24 Bias va Variance o'rtasidagi muvozanat (tradeoff)

Siz ehtimol "Bias va Variance tradeoff" degan iborani eshitgansiz. Ko‘pgina mashinaviy o‘rganish algoritmlarida mavjud o‘zgarishlarning ba’zilari **bias xatoliklarni kamaytiradi**, lekin **variance xatoliklarni oshiradi** va aksincha. Bu holat **bias va variance o'rtasida muvozanat**, ya’ni "tradeoff" deb ataladi.

Masalan:

- **Model hajmini oshirish** — masalan, nevron tarmog‘iga yangi qatlamlar yoki nevronlar qo‘shish, yoki yangi xususiyatlar (features) qo‘shish:
 - odatda **biasni kamaytiradi** (ya’ni model treningda yaxshiroq ishlaydi),
 - ammo **variance-ni oshirishi** mumkin (ya’ni test ma’lumotlarida yomonroq ishlaydi).
- **Regularizatsiya qo‘shish** — masalan, L2 (Ridge) yoki dropout kabi metodlardan foydalanish:
 - odatda **variance-ni kamaytiradi** (ya’ni model ortiqcha o‘rganishdan saqlanadi),
 - lekin **biasni oshiradi** (ya’ni model trening ma’lumotini yetarlicha o‘rganmasligi mumkin).

Zamonaviy tendensiya: kamroq muvozanat, ko‘proq imkoniyat

Bugungi kunda:

- bizda ko‘plab katta ma’lumotlar to‘plami mavjud,
- va katta hajmli nevron tarmoqlar (**deep learning**) dan foydalanish imkonи bor. Shu sababli, ko‘plab hollarda bu "tradeoff" (muvozanat) **katta muammo emas**. Endi:
 - **biasni kamaytirish**: model hajmini oshirish va regularizatsiyani moslashtirish orqali,
 - **variance-ni kamaytirish**: ko‘proq ma’lumot qo‘shish orqali amalga oshiriladi.

Masalan, siz nevron tarmoq hajmini oshirsangiz, va regularizatsiyani ham yaxshi sozlasangiz, bias kamayadi, va variance esa sezilarli darajada oshmaydi.

Yoki ko‘proq ma’lumot qo‘shsangiz, variance kamayadi, bias esa odatda o‘zgarmaydi.

Model arxitekturasi tanlash

Agar siz o‘z vazifangiz uchun **mos model arxitekturasini** tanlasangiz, bu ham bias, ham variance-ni **bir vaqtning o‘zida kamaytirishi mumkin**. Ammo, **to‘g‘ri arxitekturani tanlash oson emas**, bu juda tajriba va bilim talab qiladi.

25 Keraksiz (avoidable) biasni kamaytirish usullari

Agar sizning o‘rganish algoritmingiz **yuqori darajadagi keraksiz bias** (ya’ni, training setda past aniqlik) muammosiga duch kelsa, quyidagi texnikalarni sinab ko‘rishingiz mumkin:

Biasni kamaytiruvchi foydali usullar:

1. Model hajmini oshiring

(Masalan: neyronlar sonini yoki qatlamlar sonini ko‘paytiring):

- Bu usul **biasni kamaytiradi**, chunki model training setdagi ma’lumotlarni yaxshiroq o‘rganadi.
- Agar natijada variance ortib ketsa, unda **regularizatsiyadan foydalaning**, bu variance-ni odatda pasaytiradi.

2. Error tahlilidan ilhomlanib yangi xususiyatlar (featurelar) qo‘shing

- Agar siz error analysis (xatolar tahlili) orqali yangi, foydali featurelar (kirish belgilarini) aniqlasangiz, ularni modellarga qo‘shing.
- Bu featurelar **biasni ham, variance-ni ham kamaytirishi mumkin**.
- Teoriyada ko‘p featurelar variance-ni oshirishi mumkin, lekin agar bu ro‘y bersa — **regularizatsiya** buni bartaraf etadi.

3. Regularizatsiyani kamaytiring yoki olib tashlang

(L2, L1, Dropout kabi):

- Bu **biasni kamaytiradi**, ya’ni model trening ma’lumotlarga nisbatan ko‘proq “moslashadi”.
- Ammo bu **variance-ni oshirishi** mumkin, shuning uchun ehtiyojkorlik bilan foydalaning.

4. Model arxitekturasini moslashtiring

(Masalan: neyron tarmoq arxitekturasini o‘zgartiring):

- Siz tanlagan arxitektura vazifaga **mos bo‘lsa**, bu **bias va variance-ni bir vaqtning o‘zida kamaytirishi** mumkin.
- Bu usul **yuqori samarali**, lekin to‘g‘ri arxitekturani tanlash **tajriba va testlarni talab qiladi**.

Foydasiz bo‘lgan usul (biasga qarshi):

- Ko‘proq trening ma’lumot qo‘shish
 - Bu usul odatda variance **muammolarida foydali**,
 - Lekin **biasni kamaytirishda deyarli hech qanday yordam bermaydi**.

26 Trening to‘plamida xatolarni tahlil qilish (Error analysis)

Algoritmingiz avvalo trening to‘plamida (training set) yaxshi ishlashi kerakki, undan keyin **dev/test to‘plamlarda** ham yaxshi ishlashini kutishingiz mumkin.

Yuqorida **biasni kamaytirish bo‘yicha** berilgan texnikalarga qo‘sishimcha ravishda, ba’zida **trening ma’lumotlarida ham xatolarni tahlil qilaman**, bu **Eyeball dev** setda ishlataladigan xatolik tahliliga o‘xshaydi.

Bu yondashuv ayniqsa **yuqori bias mavjud bo‘lsa** — ya’ni, algoritm trening ma’lumotlarini yetarlicha o‘rgana olmayotgan bo‘lsa — foydali bo‘ladi.

Misol:

Aytaylik, siz **nutqni aniqlash tizimi** (speech recognition system) yaratmoqchisiz va trening ma’lumotlar to‘plami sifatida ko‘ngillilardan audio yozuvlar to‘plagansiz.

Agar sizning tizimingiz trening to‘plamida yaxshi ishlamayotgan bo‘lsa, siz taxminan **100 ta noto‘g‘ri aniqlangan audio klipni** tinglab chiqib, **asosiy xatolik kategoriylarini** tushunishga harakat qilishingiz mumkin.

Bu holatda ham, **dev set error analysisdagi** kabi, siz xatolarni turli **kategoriyalarga bo‘lib sanashingiz** mumkin.

| Audio clip | Loud background noise | User spoke quickly | Far from microphone | Comments |
|------------|-----------------------|--------------------|---------------------|-----------------------------------|
| 1 | ✓ | | | Car noise |
| 2 | ✓ | | ✓ | Restaurant noise |
| 3 | | ✓ | ✓ | User shouting across living room? |
| 4 | ✓ | | | Coffeeshop |
| % of total | 75% | 25% | 50% | |

Bu misolda siz algoritmingiz ayniqsa **ko‘p fon shovqini (background noise)** mavjud bo‘lgan trening misollarida qiynalayotganini payqashingiz mumkin.

Shunday ekan, siz **fon shovqiniga ega audio misollarni yaxshiroq o‘rgana oladigan texnikalarga** e’tibor qaratishingiz mumkin.

Shuningdek, quyidagi savolni ham o‘zingizga bera olasiz:

Aynan shu audio yozuvni eshitgan inson uni tushunib transkripatsiya qila olarmidi?

Agar fon shovqini shu darajada kuchli bo‘lsa-ki, **hech kim bu audiolarda nima deyilganini aniqlay olmasa**, unda bunday holatlarda **algoritmdan to‘g‘ri ishlashni kutish mantiqsiz bo‘lib qoladi**.

Bunday holatlarni tahlil qilishda **algoritmingizni inson darajasidagi ishlash ko‘rsatkichi bilan taqqoslash** juda foydali bo‘ladi.

Bu mavzuni "**Inson darajasidagi ishlash bilan taqqoslash**" bo‘limida yanada chuqurroq ko‘rib chiqamiz.

27 Variansni kamaytirish usullari

Agar sizning o'rganish algoritmingiz **yuqori varians** (ya'ni overfitting) bilan bog'liq muammoga duch kelayotgan bo'lsa, quyidagi usullarni sinab ko'rishingiz mumkin:

Variansni kamaytirish uchun foydali usullar:

1. Ko'proq trening ma'lumot qo'shing

Bu — variansni kamaytirishning **eng oddiy va ishonchli usuli**, agar sizda katta hajmdagi ma'lumot va uni qayta ishlash uchun yetarli kompyuter resurslari bo'lsa.

2. Regulyarizatsiya qo'shing

(L2 regulyarizatsiya, L1 regulyarizatsiya, *dropout*)

Bu texnika **variansni kamaytiradi**, ammo **biasni biroz oshirishi mumkin**.

3. Early stopping (erta to'xtatish)

Gradient descentni **dev setdag'i xatolik oshishni boshlaganida to'xtating**.

Bu ham **variansni kamaytiradi**, lekin **biasni oshiradi**.

Early stopping ko'p jihatdan regulyarizatsiyaga o'xshaydi, hattoki ayrim tadqiqotchilar uni regulyarizatsiya deb atashadi.

4. Feature selection (ya'ni, kiruvchi xususiyatlarni tanlash yoki kamaytirish)

Bu texnika **variansni kamaytirishi** mumkin, lekin **biasni oshirishi** ehtimoli ham bor.

- Masalan, 1000 ta xususiyatdan 900 tasini qoldirish juda katta ta'sir qilmasligi mumkin.
- Ammo 1000 dan 100 gacha kamaytirish (10 baravar kamaytirish) jiddiy farq qiladi va ba'zi foydali xususiyatlarni yo'qotishingiz mumkin.
- Zamonaviy deep learningda, agar ma'lumot yetarli bo'lsa, ko'pincha **hamma xususiyatlar beriladi**, va algoritm o'zi qaysi birini ishlatalishni aniqlaydi.
- Agar trening ma'lumoti kam bo'lsa, feature selection foydali bo'lishi mumkin.**

5. Model o'chamini kamaytiring

(ya'ni, neyronlar yoki qatlamlar sonini kamaytirish)

- Bu usul variansni kamaytirishi mumkin, ammo **biasni oshirish ehtimoli bor**.
- Odatda **regulyarizatsiya variansni kamaytirish uchun** yaxshiroq natija beradi.
- Modelni kichikroq qilishning asosiy foydasi — **hisoblash xarajatlarini kamaytirish va modelni tezroq o'qitish**.
- Agar sizga aynan bu tezlik kerak bo'lsa, bu usulni ko'rib chiqishingiz mumkin.

Oldingi bobdan eslatma sifatida yana ikki foydali taktika:**6. Xatolik tahliliga asoslangan xususiyatlar yaratish**

Agar siz error analysis (xatolik tahlili) orqali yangi foydali xususiyatlar kiritishga ilhom olgan bo‘lsangiz, bu **bias va variansni ham kamaytirishga yordam** berishi mumkin.

Agar bu yangi xususiyatlar variansi oshirsa, siz **regulyarizatsiya yordamida** buni kamaytirishingiz mumkin.

7. Model arxitekturasini o‘zgartirish

Masalan, **neyraviy tarmoq arxitekturasini o‘zgartirish** orqali muammoga ko‘proq moslashtirish.

Bu usul ham **bias**, ham **variансга** ta’sir qiladi.

Learning Curves (O‘rganish grafigi)

28 Bias va Variance muammolarini Learning Curve orqali aniqlash

Oldingi boblarda modelning trening va dev to‘plamlaridagi xatoliklari orqali bias va variance darajasini aniqlash usullarini ko‘rdik. Endi esa bundan ham ko‘proq ma’lumot beruvchi usul — learning curve (o‘rganish grafigi) haqida so‘z yuritamiz.

Learning curve bu — trening to‘plamidagi misollar soni oshgani sari modelning dev (test) to‘plamidagi xatoligi qanday o‘zgarishini ko‘rsatadigan grafikdir.

Bunday grafikni tuzish uchun quyidagilar bajariladi:

Aytaylik sizda 1000 ta trening misolingiz bor. Shunda quyidagicha bosqichlar amalga oshiriladi:

1. Modelni 100 ta misolda o‘qitasiz, dev to‘plamdagagi xatoni o‘lchaysiz.
2. So‘ngra 200 ta misolda o‘qitasiz, yana dev xatosini o‘lchaysiz.
3. Bu jarayon 1000 ta misolgacha davom ettiriladi.

Har bir bosqichda:

- Trening xatosi
- Dev xatosi

o‘lchanadi va yozib boriladi.

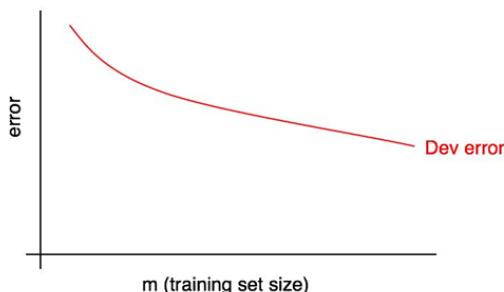
Shundan so‘ng learning curve (grafik) chiziladi. Uning gorizontal o‘qida trening misollar soni, vertikal o‘qida esa xatolik foizi bo‘ladi.

Bu grafik orqali quyidagi holatlarni farqlash mumkin:

1. Agar modelda yuqori bias bo‘lsa (underfitting):
 - a. Trening xatosi yuqori bo‘ladi.
 - b. Dev xatosi ham yuqori va trening xatosiga yaqin bo‘ladi.
 - c. Trening ma’lumotlari soni oshgani bilan dev xatosi kamaymaydi.
2. Agar modelda yuqori variance bo‘lsa (overfitting):
 - a. Trening xatosi past bo‘ladi.
 - b. Dev xatosi esa ancha yuqori bo‘ladi.
 - c. Trening namunalar soni oshgan sari dev xatosi kamayadi.
3. Agar model yaxshi ishlayotgan bo‘lsa (low bias, low variance):
 - a. Trening va dev xatoliklari ham past, ham o‘zaro yaqin bo‘ladi.

Learning curve yordamida quyidagilarni aniqlash mumkin:

- Ko‘proq ma’lumot qo‘shish foyda beradimi yoki yo‘qmi.
- Modelni murakkablashtirish yoki soddalashtirish kerakmi.
- Modelda bias muammosi bormi yoki variance.



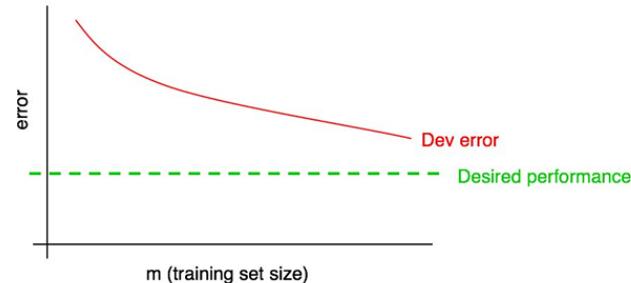
Trening ma'lumotlar soni ortib borgani sari, odatda dev (tasdiqlovchi) to'plamdag'i xato kamayishi kerak. Bu modelning umumlashma (generalization) qobiliyatining yaxshilanishini bildiradi.

Ko'pincha bizda biror "orzu qilingan xatolik darajasi" bo'ladi, ya'ni modeldan kutayotgan minimal xato darajasi. Masalan:

- Agar maqsad inson darajasida ishlaydigan model bo'lsa, unda **inson xatoligi darajasi** biz uchun "desired error rate" bo'lishi mumkin.
- Agar model biror mahsulotda (masalan, mushuk rasmlarini aniqlovchi tizimda) ishlatsa, foydalanuvchi uchun **qoniqarli tajriba** ta'minlaydigan minimal aniqlik darajasi bizning "desired error rate" bo'ladi.
- Agar siz ushbu model ustida uzoq vaqt ishlagan bo'lsangiz, **kelgusi chorakda yoki yilda** qanday yaxshilanishlarni amalga oshirish mumkinligi haqida tasavvurga ega bo'lasiz — bu ham sizning orzu qilingan darajangiz bo'lishi mumkin.

Shuning uchun **learning curve (o'rGANISH grafigi)** ga ushbu orzu qilingan xatolik darajasini ham chizing. Bu sizga quyidagilarda yordam beradi:

- Model hozirgi holatda bu darajaga yaqinlashyaptimi yoki yo'qmi, ko'rish.
- Modelni yaxshilash uchun ko'proq trening ma'lumot kerakmi yoki boshqa usullarni qo'llash kerakmi, aniqlash.
- Yangi yondashuvlar foyda bermayotganini bilib, resurslarni boshqa joyga yo'naltirish.



Agar siz **dev error** (tasdiqlovchi to'plamdag'i xato) chizig'ini vizual tarzda davom ettirsangiz, shunchaki grafikga qarab taxmin qilishingiz mumkinki, yana qancha ma'lumot qo'shilsa, kerakli aniqlikka yaqinlashish mumkinmi yoki yo'q.

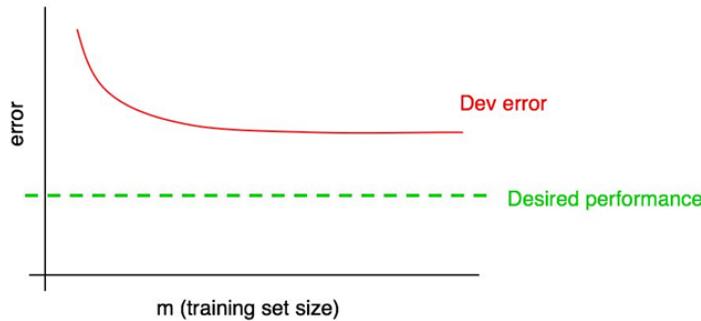
Masalan:

Agar grafikda **dev xato chizig'i hali pastlayotgan** bo'lsa (ya'ni, u hali **tekislanmagan**, pasayishda davom etyapti), unda quyidagilar haqiqatga yaqin bo'lishi mumkin:

- Trening to'plami hajmini ikki baravarga oshirsangiz, siz **orzu qilingan xatolik darajasiga** yaqinlasha olasiz.
- Ma'lumot yig'ishni davom ettirish **samarali yo'l** bo'lishi mumkin.

Biroq, agar dev xato chizig`i plateau qilgan bo`lsa (ya`ni, tekislanib qolgan, deyarli o`zgarmayapti), unda:

- Qo`sishma ma'lumot qo`shish **foyda bermaydi**.
- Chunki model dev to`plamda allaqachon maksimal darajada yaxshi ishlayapti, yoki u o`rganishda boshqa omillar bilan cheklanmoqda (masalan, model o`zi oddiy bo`lishi mumkin, yoki kerakli xususiyatlar (features) mavjud emas).]



Nima uchun trening xatolik chizig`ini chizish kerak?

Agar siz faqat **dev set xatolik** (dev error) ni ko`rsangiz, bu sizga **umid qilinayotgan aniqlikga** yetishish mumkinmi yoki yo`qmi degan savolga javob bera olmaydi. Chunki:

- Dev error pastlashda davom etsa ham, **training error** juda yuqori bo`lishi mumkin — bu **yuqori bias** (model oddiy, o`rganishga qodir emas) bo`lishi mumkinligini ko`rsatadi.
- Yoki training error juda past bo`lib, dev error yuqori bo`lsa — bu **variance** (model o`zlashtirgan, lekin umumlashtira olmayapti) muammosidir.

Shuning uchun ikkita chizmani birga ko`rib chiqamiz:

1. **Training error** — ko`proq ma'lumot bilan o`zgarmaydi yoki ozgina o`zgaradi.
2. **Dev error** — ko`proq ma'lumot bilan qanday o`zgarayotganiga qarab xulosa chiqariladi.

29 Trening xatolikni chizish (Plotting training error)

Agar siz modelingizni turli **trening to‘plam o‘lchamlari** bilan o‘qitsangiz (masalan, 100, 500, 1000, 5000 ta misol), siz quyidagi muhim holatni ko‘rasiz:

- **Dev set xatolik** odatda kamayadi — chunki ko‘proq ma’lumot modelga yaxshi umumlashtirish imkonini beradi.
- **Training set xatolik esa ko‘tariladi** — bu g‘ayritabiyy tuyulishi mumkin, lekin aslida bu normal holat.

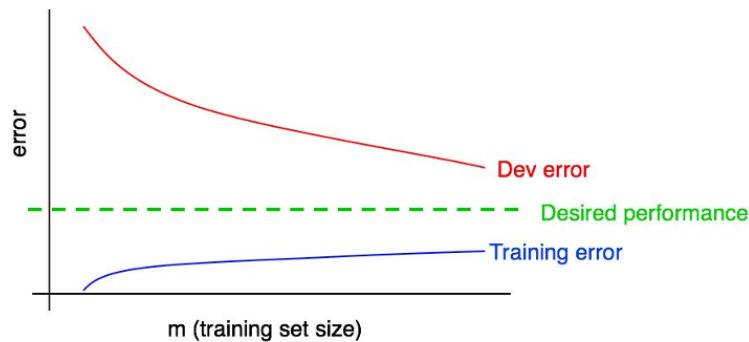
Nima uchun trening xatolik ortadi?

Kichik training to‘plamda model **eslab qolishi** (memorize) oson:

- Agar sizda atigi **2 ta rasm** bo‘lsa (biri mushuk, biri mushuksiz), model ularni eslab qoladi → **0% xatolik**.
- Ammo sizda **100 ta rasm** bo‘lsa, ulardan ba’zilari xatolik bilan markazlashgan, ba’zilari noaniq yoki chalkash bo‘lishi mumkin. Model hammasini eslab qololmaydi.
- **10000 ta rasm** bo‘lsa, ba’zilari juda chalkash yoki buzilgan bo‘ladi, shuning uchun hatto trening to‘plamda ham xatolik oshadi.
-

Learning Curve chizmalarida qanday ko‘rinadi?

1. **Training error curve:**
 - Odatda yuqoriga ko‘tariladi.
 - Boshlanishda (kam data bilan) xatolik past (hatto 0%), keyin ko‘proq data qo‘shilgani sayin oshadi va **barqarorlashadi**.
2. **Dev error curve:**
 - Odatda pastga tushadi.
 - Ko‘proq data bilan model umumlashtirishni yaxshiroq o‘rganadi.

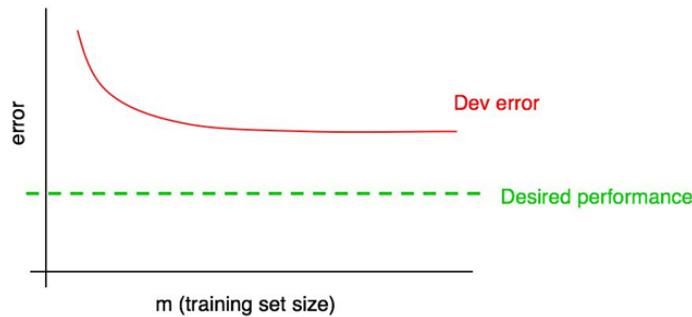


Trening va Dev xatolik chizmalarini qanday talqin qilish kerak?

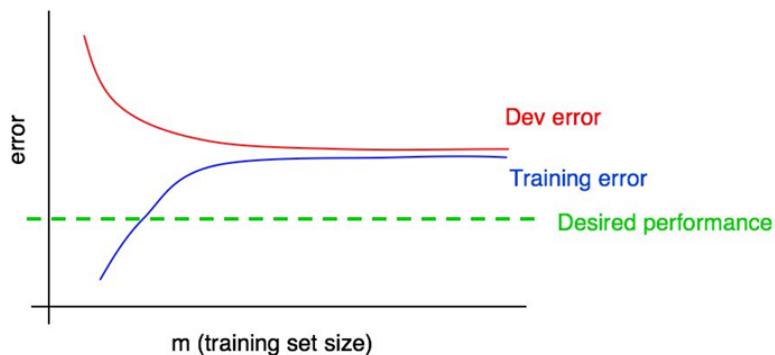
Siz learning curve grafigida odatda quyidagilarni ko‘rasiz:

- **Ko‘k chiziq (Training Error)** – bu odatda **oshadi** trening to‘plam hajmi ortganda.
- **Qizil chiziq (Dev Error)** – bu odatda **kamayadi**, chunki ko‘proq ma’lumot modelni umumlashtirishga o‘rgatadi.
- Dev xatolik odatda trening xatolikdan **yuqorida bo‘ladi**, chunki model ko‘rmagan datalarga nisbatan yomonroq ishlaydi.

30 O‘rganish chiziqlarini tahlil qilish: Yuqori og‘ish (high bias)



- Trening xatoligi yuqori va deyarli o‘zgarmayapti
- Dev (tasdiqlash) xatoligi ham treningnikiga juda yaqin
- Har ikki chiziq birga “plateau” qilgan (ya’ni, tekis va pasaymayapti)
- Bu grafik sizda **yuqori bias** (underfitting) borligini ko‘rsatadi. Ya’ni:
- Modelingiz **yeterlicha kuchli emas**
- Ko‘proq ma’lumot qo‘sish foyda bermaydi
- Yechim: **model arxitekturasini yaxshilash**, ya’ni **kattaroq model, yaxshiroq xususiyatlar (features)** yoki **kamroq cheklov (regularization)** bilan urinib ko‘rish kerak

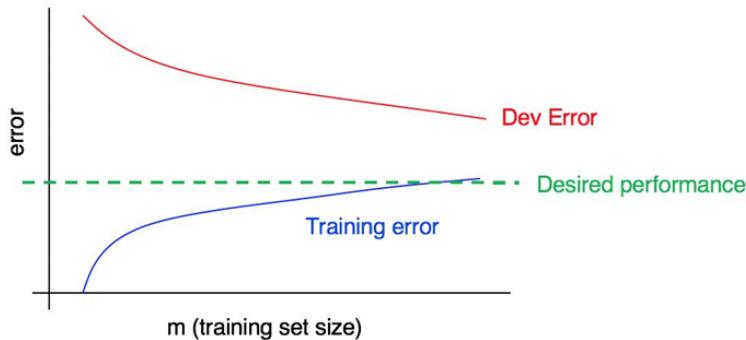


Endi siz **aniq** bilasizki, **yangi ma’lumot qo‘sish** o‘zi bilan **yeterli emas** — ya’ni bu holatda ko‘proq ma’lumot yig‘ish foyda bermaydi. Nega?

Eslab qoling, ikki muhim kuzatuv mavjud:

1. **Trening ma’lumotlari soni oshganda, trening xatoligi faqat oshishi mumkin**
 - Algoritm ko‘proq ma’lumotni “yodlab olish” o‘rniga umumlashtirishga harakat qiladi. Shuning uchun ko‘proq ma’lumot qo’shsangiz, **trening xatoligi kamaymaydi, balki ko‘payadi yoki bir xil qoladi**.
2. **Dev xatoligi har doim deyarli trening xatoligidan yuqoriroq bo‘ladi**
 - Shuning uchun **trening xatoligi** allaqachon siz istagan darajadan yuqorida bo‘lsa, **dev xatoligini** istalgan darajagacha tushirish deyarli **imkonsiz** bo‘ladi.

31 O'rganish grafiklarini (Learning curves) tahlil qilish: Boshqa holatlar



Agar sizda mashinani o'rgatish (machine learning) modelining **learning curve** grafigida **trening xatosi** (training error) past va **test yoki dev xatosi** (dev error) yuqori bo'lsa, bu juda muhim diagnostika signalidir. Bunday grafikdan quyidagilarni anglash mumkin.

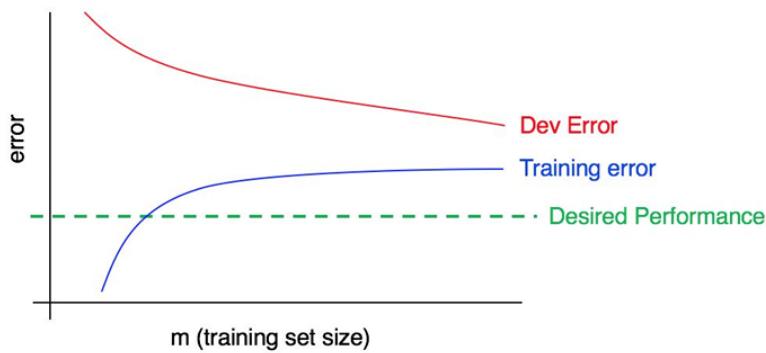
Avvalo, trening xatosining pastligi modelning trening to'plamdagи ma'lumotlarni yaxshi o'zlashtirganini bildiradi. Bu shuni anglatadiki, model trening ma'lumotlarida juda yaxshi natija beradi — demak, bu holatda **bias**, ya'ni **yo'l qo'yiladigan xatolik (yuzaga keladigan sistemmatik kamchilik)** past bo'ladi. Model yaxshi o'rganmoqda va bu yaxshi alomatdir.

Biroq, grafikda dev xatosi juda yuqori bo'lsa, bu model **umumlashtira olmayotgani**, ya'ni trening to'plamdan tashqaridagi ma'lumotlarga nisbatan noto'g'ri qarorlar qabul qilayotganini bildiradi. Bu esa **variance**, ya'ni **o'zgaruvchanlik** muammosining yuqoriligi alomati hisoblanadi. Model trening ma'lumotlariga haddan tashqari "yopishib" qolgan — bu holat ko'pincha **overfitting**, ya'ni ortiqcha moslashuv deb ataladi.

Shunday qilib, bu grafikdan quyidagicha xulosa qilish mumkin: modelda **kam bias** va **yuqori variance** mavjud. Bu holatda, modelni yaxshilash uchun bir necha strategiyalar qo'llanilishi mumkin. Avvalo, **trening ma'lumotlarining hajmini oshirish** orqali variance kamaytiriladi. Katta ma'lumot to'plami modelni umumiyoq naqshlarni aniqlashga majbur qiladi, bu esa umumlashtirish qobiliyatini oshiradi.

Shuningdek, **regulyarizatsiya** usullaridan foydalanish mumkin. Masalan, **L2 regulyarizatsiya** (Ridge regression), **dropout** yoki **early stopping** — bular modelni haddan tashqari murakkab bo'lishdan saqlaydi va variance ni kamaytirishga yordam beradi. Agar kerak bo'lsa, model arxitekturasi soddalashtirilishi mumkin, lekin bu biasni oshirishi mumkin, shuning uchun ehtiyyotkorlik bilan bajarilishi kerak.

Xulosa qilib aytganda, bu turdagи learning curve bizga modelda **kam bias, ammo yuqori variance** mavjudligini ko'rsatadi. Shuning uchun asosiy e'tibor modelning umumlashtirish qibiliyatini oshirishga qaratilishi kerak. Bunga erishish uchun, qo'shimcha ma'lumot yig'ish, regulyarizatsiya qo'llash va ehtimol modelning tuzilmasini optimallashtirish foydali bo'ladi.



Agar sizda trening (o'rnatish) xatosi juda yuqori bo'lsa va bu siz xohlagan darajadagi ideal natijadan ancha yomon bo'lsa, bu holat sizning modelda **yuqori bias**, ya'ni **sistemmatik xatolik** borligini bildiradi. Model trening ma'lumotlarini ham yaxshi o'rganolmayapti, ya'ni u etarlicha kuchli emas yoki arxitekturasi mos emas. Bu turdagи xatoliklar odatda model sodda bo'lganda yoki noto'g'ri tanlanganda yuzaga keladi.

Bundan tashqari, agar dev (tasdiqllovchi) to'plamdagи xatolik yana ham yuqori bo'lsa va trening xatosidan sezilarli farq qilsa, bu **yuqori variance** borligini ko'rsatadi. Bu shuni anglatadiki, model nafaqat trening ma'lumotlariga yaxshi moslasha olmayapti, balki ularni umumlashtirishda ham qiynalayapti. Boshqacha qilib aytganda, model yangi, ko'rmagan ma'lumotlarga nisbatan ham noto'g'ri natijalar bermoqda.

Bunday holatda sizda **ikkita asosiy muammo** bor: **bias ham yuqori, variance ham yuqori**. Bu esa murakkab holat bo'lib, modelda ham kuch yetishmasligi, ham umumlashtirish qobiliyati zaifligini bildiradi. Bunday vaziyatda quyidagi strategiyalarni qo'llash foydali bo'ladi:

1. **Model arxitekturasini kuchaytirish**, ya'ni qatlamlar yoki neyronlar sonini oshirish orqali modelni kuchliroq qilish.
2. **Xatolik tahlilini amalga oshirish** va yangi, foydali input xususiyatlar (features) kiritish. Bu modelga yanada chuqurroq o'rganishga yordam beradi.
3. **Regulyarizatsiya metodlarini** (masalan, L2 regulyarizatsiya yoki dropout) to'g'ri sozlash orqali variance ni kamaytirish.
4. **Yangi trening ma'lumotlarini yig'ish** yoki mavjudlarini tozalash, bu esa variance ni pasaytiradi va modelga yaxshiroq umumlashtirish imkonini beradi.

31 Kichik ma'lumotlar to'plamida learning curve (o'rGANISH EGRI CHIZIG'I) CHIZISH

Tasavvur qiling, sizda atigi 100 ta trening namunadan iborat kichik o'quv to'plamingiz bor. Siz ushbu ma'lumotlar ichidan tasodifiy tanlab 10 ta, keyin 20 ta, 30 ta va shunday qilib 100 tagacha trening namunalarida modelni o'rgatasiz. Har bir bosqichda modelni alohida o'rgatib, olingan xatoliklarni belgilab borasiz. Shu tarzda siz **learning curve** – ya'ni trening to'plami kattaligiga qarab model xatoligini ko'rsatadigan grafik chizasiz.

Kichik miqdordagi ma'lumotlarda bu grafiklar odatda **noaniq va "shovqinli"** ko'rindi. Chunki faqat 10 ta misol asosida modelni o'rgatsangiz, sizga "omad kulib boqishi" va yaxshi namunalar tanlanishi mumkin, yoki aksincha – "yomon" va chalkash namunalar tanlanishi mumkin. Bu esa trening va test xatoliklarining **kutilgan qiymatlardan keskin farq qilishiga** olib keladi.

Agar sizning masalangizda **klasslar taqsimoti noteng bo'lsa** — masalan, mushukni aniqlash masalasi bo'lib, unda 80% rasmda mushuk yo'q, faqat 20% rasmda mushuk bor bo'lsa — bu holatda tasodifiy 10 ta misoldan iborat kichik trening to'plamda faqat mushuk bo'lmagan rasmlar tanlanib qolishi ehtimoli yuqori. Bu esa modelga kerakli bilimni o'rgatishga to'sqinlik qiladi.

Shovqinli learning curve ga qarshi ikki yechim:

- Bir emas, bir nechta kichik o'quv to'plamlarda modelni o'rgatish:** Masalan, 10 ta namunadan iborat to'plamni bir necha marta (3–10 marta) turlicha tanlab oling (bunda replacement bilan tanlash mumkin). Har bir tanlov uchun modelni alohida o'rgating va ularning o'rtacha trening/dev xatolarini hisoblang. Shunda siz real tendensiyalarni yanada aniqroq ko'ra olasiz.
- Muvozanatlangan kichik o'quv to'plamlar tanlash:** Masalan, agar klasslar taqsimoti 20% (pozitiv) va 80% (negativ) bo'lsa, siz 10 ta namunada shunday balansni saqlang – 2 ta ijobiy, 8 ta salbiy misollar bo'lsin. Bu yondashuv modelga har ikkala klassni ko'rsatishga imkon beradi.

Agar sizning trening to'plamingiz **yeterlicha katta (masalan, 10,000 ta namunadan ortiq)** bo'lsa, va klasslar taqsimoti ham muvozanatli bo'lsa, yuqoridagi usullarga murojaat qilmasangiz ham bo'ladi.

Hisoblash xarajati haqida

Learning curve chizish uchun siz ko'plab alohida modellarning treninglarini bajarishingiz kerak bo'ladi. Bu hisoblash nuqtai nazaridan arzon emas. Ayniqsa, katta trening to'plamlarda bu ko'p vaqt talab qiladi. Shuning uchun ma'lumot sonini 1000, 2000, 4000, 6000, 10000 kabi **eksponent shaklda bosqichma-bosqich oshirib**, har bosqichda modelni o'rGANISH orqali grafik tuzish samaraliroq bo'ladi. Bu orqali siz egri chiziqning asosiy tendensiyasini aniqlay olasiz va kerakli qarorlarni tezroq qabul qilishingiz mumkin.

Inson darajasidagi ishlash bilan solishtirish

Comparing to human-level performance

33 Nega inson darajasidagi ishlash bilan solishtiramiz

Ko‘plab mashinaviy o‘rganish (ML) tizimlari insonlar yaxshi bajara oladigan vazifalarni avtomatlashtirishga qaratilgan. Bunga tasvirlarni tanish, nutqni aniqlash va spam email’larni ajratish kiradi. So‘nggi vaqtarda ML algoritmlari shu darajada kuchli bo‘lib qoldiki, ular tobora ko‘proq vazifalarda inson darajasidan oshib ketmoqda.

Insonlar yaxshi bajara oladigan vazifalarda ML tizimi yaratish bir necha sababga ko‘ra osonroq:

1. **Insonlar tomonidan belgilangan ma’lumotlarni olish oson.** Masalan, odamlar mushuk rasmlarini osongina tanib oladi, shuning uchun sizning algoritmingiz uchun aniq belgilangan (label qilingan) ma’lumotlarni olish oson bo‘ladi.
2. **Xatolarni tahlil qilishda inson intuitsiyasidan foydalanish mumkin.** Masalan, agar nutqni aniqlovchi tizim “This recipe calls for a *pear* of apples” deb noto‘g‘ri yozsa (“pair” o‘rniga “pear”), inson intuitsiyasiga tayanib, to‘g‘ri transkripsiyanı aniqlashda qanday kontekst ishlatilganini tushunib, shunga mos algoritmi yaxshilash mumkin.
3. **Inson darajasidagi natijalarni “optimal xato darajasi” va “maqsadli xato darajasi” sifatida ishlatalish.** Masalan, sizning algoritmingiz 10% xato qilsa, lekin inson bu vazifani 2% xato bilan bajarsa, demak optimal xato darajasi 2% yoki undan ham past, va sizda kamida 8% “oldini olish mumkin bo‘lgan xatolik” (avoidable bias) mavjud. Shunda biasni kamaytiruvchi usullarga e’tibor qaratish kerak bo‘ladi.

Garchi 3-band juda muhim bo‘lmasdek tuyulsa-da, men tajribamda shuni ko‘rganmanki, aniq, erishilishi mumkin bo‘lgan xato darajasini belgilab olish jamoaning ishlashini tezlashtiradi. Agar sizning algoritmingizda bias yuqori ekanini aniqlasangiz, bu juda foydali bo‘ladi va siz sinab ko‘rishingiz mumkin bo‘lgan aniq choralar ro‘yxatini ochadi.

Biroq ba’zi vazifalar bor, ular insonlar uchun ham murakkab. Masalan: foydalanuvchiga qaysi kitobni tavsiya qilish, veb-saytda qaysi reklamani ko‘rsatish yoki aksiyalar bozorini bashorat qilish. Kompyuterlar allaqachon ko‘pchilik insonlarga nisbatan bu vazifalarni yaxshi bajarmoqda. Bunday vaziyatlarda quyidagi muammolarga duch kelamiz:

- **Label’larni olish qiyinlashadi.** Masalan, insonlar kitob tavsiyasi bazasini “eng yaxshi” kitob deb belgilab bera olmaydi. Agar sizda kitob sotiladigan ilova yoki sayt bo‘lsa, foydalanuvchilarga kitob ko‘rsatib, ular nima sotib olishini kuzatib, ma’lumot yig‘ishingiz mumkin. Aks holda, ma’lumot yig‘ish uchun ijodiy usullar topishingizga to‘g‘ri keladi.
- **Inson intuitsiyasi bilan ishlash qiyinlashadi.** Masalan, deyarli hech kim fond bozorini aniq bashorat qila olmaydi. Shuning uchun, agar sizning algoritmingiz tasodifiy taxmindan yaxshi ishlama, uni qanday yaxshilash mumkinligini tushunish ham mushkul bo‘ladi.
- **Optimal va maqsadli xato darajasini aniqlash qiyin.** Masalan, sizda allaqachon yaxshi ishlayotgan kitob tavsiyasi tizimi bo‘lsa, uni yana qancha yaxshilash mumkinligini inson mezoniga ega bo‘lmasdan aniqlash oson emas.

Shu sababli, inson darajasidagi ishlash bilan taqqoslash ML jarayonida juda foydali bo‘ladi.

34 Inson darajasidagi ishlashni qanday aniqlash mumkin

Tasavvur qiling, siz rentgen tasvirlaridan avtomatik tashxis qo'yadigan tibbiy tasvirlash ilovasida ishlayapsiz. Oddiy inson, ya'ni bu sohaga oid maxsus bilimga ega bo'limgan, faqat asosiy tayyorgarlikka ega shaxs bu vazifada 15% xatolik bilan ishlaydi. Yosh shifokorlar 10% xatolik, tajribali shifokorlar 5% xatolik qiladi. Birgalikda har bir rasmni muhokama qilib baholaydigan shifokorlar jamoasi esa atigi 2% xatolik bilan ishlaydi. Shunday holatda qaysi ko'rsatkichni "inson darajasidagi ishlash" (ya'ni optimal xato darajası) deb hisoblash kerak?

Bu holatda men 2% xatolikni inson darajasidagi ishlash sifatida olgan bo'lardim. Ya'ni, bu bizning tizimimiz uchun **optimal xato darajasini** anglatadi. Shu bilan birga, 2% ni **maqsadli natija** sifatida ham belgilash mumkin, chunki bu daraja uchta muhim mezonga mos keladi:

1. **Label qilingan (yorliqlangan) ma'lumotlarni olish oson** — Siz bir guruh shifokorlardan 2% xatolik bilan aniq label'lar olishingiz mumkin.
2. **Xatolarni tahlil qilishda inson intuitsiyasidan foydalanish mumkin** — Rasmni ko'rib, shifokorlar bilan muhokama qilish orqali nima sababdan tizim xato qilayotganini tushunish mumkin bo'ladi.
3. **Inson darajasidagi ishlash orqali optimal va maqsadli xato darajasini belgilash mumkin** — 2% ni optimal deb olish mantiqan to'g'ri, chunki shifokorlar jamoasi bu ko'rsatkichga erishishi mumkin. Bu ko'rsatkichdan pastroq xato bo'lishi mumkin (nazariy jihatdan), lekin undan yuqoriq bo'lishi mumkin emas. Shu bois 5% yoki 10% ni optimal deb hisoblash mantiqsiz bo'ladi, chunki ular allaqachon yomonroq natijalar.

Shuni ham yodda tutish kerakki, **barcha rasmlarni shifokorlar jamoasi bilan muhokama qilish amaliy jihatdan qimmatga tushadi**. Amalda, siz arzonroq usulni tanlashingiz mumkin: ko'pgina oddiy holatlarni yosh shifokorlar label qilsin, murakkab holatlar esa tajribali shifokorlar yoki jamoaviy muhokamaga olib chiqilsin.

Agar sizning hozirgi ML tizimingiz 40% xato qilayotgan bo'lsa, sizga kim label berayotgani unchalik muhim emas — yosh shifokormi, tajribalimi. Lekin agar tizimingiz allaqachon 10% xato bilan ishlayotgan bo'lsa, unda **inson darajasini 2% deb belgilash sizga tizimni yanada yaxshilash uchun kuchliroq mezon beradi**.

35 Inson darajasidan yuqori natijaga erishish

Tasavvur qiling, siz nutqni tanish tizimi ustida ishlayapsiz va sizda ko'plab audio yozuvlardan iborat dataset mavjud. Ushbu datasetdagi yozuvlar juda shovqinli bo'lib, hatto odamlar ham ularda **10% xato bilan transkripatsiya** qilishadi. Ammo sizning tizimingiz allaqachon **8% xato** bilan ishlamoqda — bu inson darajasidan yaxshiroq. Endi savol tug'iladi: **33-bobda ko'rib chiqilgan uchta yondashuv yordamida bu tizimni yanada yaxshilash mumkinmi?**

Javob: Agar siz insonlar sizning tizimingizdan ancha yaxshi natija berayotgan **ma'lum bir holatlar to'plamini (subset)** aniqlay olsangiz, unda bu yondashuvlar hali ham samarali bo'ladi.

Masalan: sizning tizimingiz shovqinli audio yozuvlarda nutqni tanishda insonlardan yaxshiroq ishlayapti, lekin **juda tez gapirilgan** audio yozuvlarda odamlar hali ham tizimdan ustun.

Ana shunday tez gapirilgan yozuvlar uchun quyidagi amaliy qadamlarni qo'llashingiz mumkin:

1. **Inson transkriptorlaridan yuqori sifatli transkriptlar olish** mumkin — tizim xato qilgan bo'lsa-da, insonlar to'g'ri yozadi.
2. **Inson intuitsiyasidan foydalanish** — nima uchun odamlar bu gapni to'g'ri eshitdi, tizim esa xato qildi? Shu savolga javob izlash orqali tizimni takomillashtirish mumkin.
3. **Tez gapirilgan nutqlar uchun inson darajasidagi aniqlikni** maqsad sifatida belgilash mumkin.

Umumiy holatda

Agar dev/test to‘plamingizda hali ham **insonlar to‘g‘ri, tizim noto‘g‘ri javob berayotgan namunalar mavjud bo‘lsa**, unda ilgari ko‘rib chiqilgan ko‘plab texnikalar hamon qo‘llanilishi mumkin. Hatto sizning tizimingiz o‘rtacha ko‘rsatkichlar bo‘yicha insonlardan yaxshiroq ishlayotgan bo‘lsa ham!

E'tiborga loyiq holat

Bugungi kunda **mashinalar ko‘p sohalarda insonlardan ustun**: masalan, film reytinglarini bashorat qilish, yetkazib berish mashinasining qayergacha borishini aniqlash, yoki kredit arizasini tasdiqlash-yo‘qligini baholash.

Biroq **mashinalar allaqachon insonlardan yaxshiroq ishlayotgan vazifalarda**:

- **O‘quv ma’lumotlarini label qilish** qiyinlashadi.
- **Inson intuitsiyasiga tayanish** qiyin bo‘ladi — chunki odamlar ham xatoga yo‘l qo‘yadi.
- **Qayerda tizim xato qilayotganini aniqlash** murakkablashadi.

Shu sababli, **mashina hali inson darajasiga yetmagan masalalarda taraqqiyot tez bo‘ladi**, lekin **mashina allaqachon inson darajasidan o‘tgan joylarda esa o‘sish sekinlashadi**.

Turli taqsimotlarda (distributions) o‘qitish va test qilish

36 Qachon o‘qitish va test ma’lumotlari turli taqsimotdan bo‘lishi kerak?

Tasavvur qiling, siz mushuk rasmi ilovasida foydalanuvchilar tomonidan yuklangan 10,000 ta rasmga egasiz va bu rasmlarning har biriga mushuk bormi yoki yo‘qmi deb **qo‘lda belgi qo‘yilgansiz**. Shuningdek, siz internetdan yuklab olingen **yana 200,000 ta rasmga** egasiz. Endi savol tug‘iladi: siz **train/dev/test** to‘plamlarini qanday aniqlashingiz kerak?

Agar siz aynan foydalanuvchilarning yuklagan rasmlari asosida yaxshi ishlaydigan model tuzmoqchi bo‘lsangiz, **10,000 ta user image** asosida **dev** va **test** to‘plamlarini tanlagan ma’qul. Ayni paytda, agar siz **katta hajmli model** (chuqur o‘rganish)ni o‘rgatyapsiz va sizga ko‘p ma’lumot kerak bo‘lsa, qolgan **200,000 internet rasmlarni** faqat **train** uchun ishlatishtingiz mumkin. Bu holatda, sizning **train** to‘plamingiz va **dev/test** to‘plamlaringiz turli taqsimotdan bo‘lgan bo‘ladi.

Nimaga bu yondashuv ma’qul?

Ba’zilar 210,000 rasmni aralashtirib, random tarzda train/dev/test’ga bo‘lishni taklif qilishi mumkin. Ammo bunda **dev** va **test** to‘plamlarining **97.6% internetdan** bo‘ladi, ya’ni bu **asl foydalanuvchi muhitini aks ettirmaydi**. Shu sababli:

Dev va test to‘plamlarini siz kelajakda oladigan va yaxshi natijaga erishmoqchi bo‘lgan haqiqiy ma’lumotlarga yaqin bo‘lishi kerak.

Tarixiy kontekst

Ilk mashina o‘rganish davrida ma’lumot kam edi, shuning uchun mavjud yagona datasetni shunchaki bo‘lib olardik: train, dev, test. Bu vaqtida barcha ma’lumotlar bitta taqsimotdan edi — bu usul to‘g‘ri ishlardi.

Ammo hozirda **katta ma’lumotlar davrida**, sizda internetdan olgan **millionlab ma’lumotlar** bo‘lishi mumkin. Bu ma’lumotlar train uchun juda foydali, hatto agar ular dev/test’dan farqli bo‘lsa ham.

Yaxshi amaliy yondashuv

Mushuk rasm misolida:

- 10,000 user-rasmlaridan 5,000 tasini **dev/test** uchun ajrating,
- qolgan 5,000 user-rasm va 200,000 internet rasmni **training** uchun foydalaning.

Bu usul train to‘plamiga **haqiqiy foydalanuvchi ma’lumotlaridan** ham bir qismi qo‘silganini ta’minlaydi, bu esa modelga haqiqiy holatlarga yaxshiroq moslashishga yordam beradi.

37 Barcha ma'lumotlaringizni ishlatingiz kerakmi – qanday qaror qilish kerak?

Tasavvur qiling, siz mushuklarni aniqlovchi (cat detector) model qurmoqdasiz. Sizda:

- **10,000 ta foydalanuvchi yuklagan rasm** bor — bu sizning dev/test to‘plamingizga o‘xhash bo‘lgan, real va sizga kerakli taqsimotdan kelgan ma'lumotlar.
- Yana **20,000 ta internetdan olingan rasm** mavjud.

Endi savol: siz **hamma 30,000 rasmni** modelga berishingiz kerakmi yoki **20,000 internet rasmlarni** chiqarib tashlashingiz kerakmi (modelni noto‘g‘ri yo‘lga olib ketmasligi uchun)?

Avvalgi algoritmlar davrida

Ilgari, klassik (oddiy) mashina o‘rganish usullarida — masalan, qo‘lda ishlab chiqilgan tasvir xususiyatlari + oddiy chiziqli klassifikatorlar ishlatalganida — bu **xavf real edi**. Ya’ni, bir xil bo‘lмаган taqsimotdagi ma'lumotlarni aralashtirish natijada modelning yomonroq ishlashiga olib kelardi. Shu sababli ba’zi muhandislar sizni 20,000 ta internet rasmlarni qo‘shmaslikka chaqirishi mumkin.

Ammo zamonaviy chuqur o‘rganish davrida...

Bugungi **neural network** (asosan chuqur o‘rganish) tizimlari **kuchli va moslashuvchan** bo‘lib, bu xavf sezilarli darajada kamaygan. Agar siz **etarlik darajada katta tarmoq** qurishga resursingiz bo‘lsa (ko‘p qatlamli, ko‘p neyronli), unda siz **20,000 ta rasmni qo‘shishingiz mumkin**, va bu katta ehtimol bilan model sifatini yaxshilaydi.

Bu quyidagilarga asoslanadi:

Ikkala rasm turida (internet + mobil ilova) **bir xil $x \rightarrow y$ (kirishdan chiqishga) xarita** mavjud. Ya’ni, manba turini bilmasdan ham, biror tizim bu rasmida mushuk bor yoki yo‘qligini to‘g‘ri aniqlay oladi.

20,000 ta qo‘shimcha rasmning ikki asosiy ta’siri bor:

1. **Ijobiy ta’sir:** Modelga ko‘proq mushuklar qanday ko‘rinishga ega ekanini o‘rgatadi. Internetdagи rasm va foydalanuvchi yuklagan rasm o‘rtasida o‘xhashliklar bo‘lishi mumkin — bu o‘xhashlikdan model foydalansa bo‘ladi.
2. **Salbiy ta’sir (nazariy):** Tarmoq **internet rasmlarning o‘ziga xos xususiyatlarini ham** (masalan: yuqori aniqlik, kadrga joylashish tarzi) o‘rganadi. Bu, o‘z navbatida, uning "xotirasining" bir qismini foydalanuvchi rasm taqsimotiga emas, boshqa narsalarga sarflashiga olib keladi. Bu nazariy jihatdan model ishlashini pasaytirishi mumkin.

Sherlock Holmes analogiyasi

Sherlock Holmes shunday degan: **miyangiz buxgalteriya xonasiga o‘xshaydi** — cheklangan joy bor. Har bir yangi fakt, eski bir faktni siqib chiqaradi. Ya’ni: **"foydasiz faktlar foydali bilimlarni siqib chiqaradi."**

Ammo, **agar sizda yetarli darajada kuchli neyron tarmog‘i bo‘lsa**, unda bu muammo emas — sizda katta "attic" bor, shuning uchun internet va foydalanuvchi rasm ma'lumotlarini birga berishingiz mumkin.

Qachon ehtiyyot bo'lish kerak?

Agar sizda **kichik model** bo'lsa (kompyuter kuchi yetarli emas, model oddiy), unda siz trenirovka ma'lumotlarini ehtiyyotkorlik bilan tanlashingiz kerak — ayniqsa ular **dev/test to'plam taqsimotiga** mos kelmas ekan.

Noto'g'ri yoki yaroqsiz ma'lumotlar haqida

Agar sizda **maqsadli vazifaga hech qanday aloqasi bo'lmagan ma'lumotlar** bo'lsa — masalan, siz odam, joy, mushuk rasmilariga asoslangan ilova uchun model qurayapsiz, lekin sizda **ko'p miqdordagi tarixiy hujjatlar (scanned PDFs)** bor — bu ma'lumotlarni qo'shish kerak emas.

Sababi oddiy:

- Ular **relevant emas**, ya'ni o'rghanish uchun foydasi yo'q.
- Ularni ishslash **kompyuter kuchini behuda sarflaydi**.

Agar sizda mavjud bo'lgan ba'zi hujjatlar umuman mushuklarga o'xshash narsalarni o'z ichiga olmasa va hatto sizning dev/test to'plamingizdagi rasmlarga ham umuman o'xshamasa, u holda bu ma'lumotlarni modelga berish foydasiz bo'ladi. Chunki bunday ma'lumotlar sizning modelga mushukni qanday tanish haqida deyarli hech qanday foydali bilim bera olmaydi.

Neyron tarmoqlar o'z "xotirasi"da, ya'ni representatsiya imkoniyatlarida, ma'lum darajadagi ma'lumotni saqlay oladi. Agar siz foydasiz hujjatlarni modelga bersangiz, bu neyron tarmoqning imkoniyatlarini "bekorchi" ma'lumotlarga sarflashga olib keladi. Bundan tashqari, bu jarayon kompyuter resurslarini ham isrof qiladi — xotira, hisoblash quvvati, o'qitish vaqtini ortadi.

Shuning uchun, agar sizda model uchun mutlaqo befoyda, kontekstdan yiroq, real muammo bilan aloqasi bo'lmagan ma'lumotlar bo'lsa — masalan, mushuklarni aniqlovchi model uchun skan qilingan tarixiy matnlar — bunday ma'lumotlarni trening to'plamiga qo'shmaslik eng to'g'ri qaror bo'ladi. Asosiy tamoyil shuki: modelni faqat o'rgatilmoqchi bo'lgan vazifaga mos, foydali bilim bera oladigan ma'lumotlar bilan o'rgating.

38 Izchil bo'lmagan ma'lumotlarni qo'shish kerakmi yoki yo'qmi – qanday qaror qilish kerak

Uy narxlarini bashorat qilish modelini ishlab chiqayotganingizni tasavvur qiling, masalan, Nyu-York shahridagi uy narxlarini maydon (x) ga qarab oldindan bilmoxchisiz. Nyu-Yorkda uy narxlari ancha yuqori bo'lishi mumkin. Sizda yana Detroyt shahridan olingan uy narxlarini ma'lumotlari mavjud bo'lsa, savol tug'iladi: bu ikki turdag'i ma'lumotni birlashtirish kerakmi? Muammo shundaki, Nyu-York va Detroyt shaharlari orasida juda katta farq mavjud — ayniqsa uy narxlari bo'yicha. Bir xil xususiyatga ega, masalan, 100 kvadrat metrli uy Nyu-Yorkda 1 million dollarga baholanishi mumkin, lekin Detroytda atigi 100 ming dollar bo'lishi mumkin. Agar siz faqat Nyu-Yorkdagi uy narxlarini oldindan aytmoqchi bo'lsangiz, bu holda Detroytdagi narxlar bilan bog'liq ma'lumotlarni trening to'plamiga qo'shish modelni chalg'itadi va umumiy anqlikni pasaytiradi. Shuning uchun, bunday holatda Detroytdagi ma'lumotlarni tashlab yuborganingiz ma'qul.

Bu holatni mushuklarni aniqlash loyihasidagi mobil ilova va internetdan olingan rasmlar misoli bilan solishtirsak, farq sezilarli. Mushuklarni aniqlashda, rasm internetdanmi yoki mobil ilovadanmi kelganligini bilmasdan ham, x (rasm) dan y (mushuk bor yoki yo'qligi) ni aniq

aniqlash mumkin. Ya'ni, bu yerda $f(x) \rightarrow y$ bog'liqligi saqlanadi, manba muhim emas. Bu holatda turli manbalar ma'lumotlarini birlashtirish foydali bo'lishi mumkin.

Ammo uy narxlari misolida esa bu bog'liqlik izchil emas. Ya'ni, bir xil x bo'lsa ham, y yaqqol farq qiladi — uy qayerda joylashganligi hal qiluvchi ahamiyatga ega. Shuning uchun, siz faqat Nyu-Yorkdagi uylar uchun model yaratmoqchi bo'lsangiz, Detroytdagi ma'lumotlar sizning model faoliyatizingizga salbiy ta'sir qiladi. Bunday ma'lumotlar "mos kelmaydigan" yoki "izchil bo'lman" ma'lumotlar deyiladi va ularni ishlatmaslik kerak.

39 Ma'lumotlarni og'irlilik (Weighting data)

Tasavvur qiling, sizda internetdan olingan 200,000 ta rasm va mobil ilova foydalanuvchilaridan olingan 5,000 ta rasm mavjud. Bu yerda ma'lumotlar hajmi o'rtasida 40:1 nisbati bor. Nazariy jihatdan, agar siz juda katta neyron tarmoq qursangiz va uni 205,000 ta rasmda yetarlicha uzoq vaqt davomida o'rgatsangiz, modelning internetdag'i hamda mobil ilovadagi rasmiga nisbatan yaxshi ishlashida hech qanday muammo bo'lmasligi kerak.

Ammo amaliyotda, 40 baravar ko'p internet rasmlariga ega bo'lish, bu ikkala ma'lumot turini teng darajada modelga o'rgatish uchun sizga 40 baravar yoki undan ham ko'proq hisoblash quvvatini talab qiladi. Ya'ni, model ko'proq resursni internetdag'i rasm ma'lumotlariga sarflaydi, bu esa siz uchun muhim bo'lgan — mobil ilova foydalanuvchilari rasmlarida — pastroq natijalarga olib kelishi mumkin.

Agar sizda yetarlicha katta hisoblash resurslari bo'lmasa, bir murosa sifatida internet rasmlariga pastroq og'irlilik (ya'ni, ta'sir darajasi) berish mumkin.

Masalan, agar sizning optimizatsiya maqsadingiz kvadrat xatolikni kamaytirish bo'lsa (bu klassifikatsiya vazifasi uchun eng yaxshi tanlov emas, lekin tushuntirishni soddalashtiradi), u holda o'quv algoritmingiz quyidagi funktsiyani optimallashtirishga harakat qiladi:

Yuqoridagi formuladagi birinchi yig'indi — bu 5,000 ta mobil ilova rasmlari ustidan olinadi, ikkinchi yig'indi esa 200,000 ta internet rasmlariga tegishli. Buning o'rniغا siz optimallashtirish funksiyasiga qo'shimcha parametr — β (beta) kiritishingiz mumkin.

Bu parametr yordamida siz mobil ilova va internet ma'lumotlariga o'zaro **turlicha og'irlilik** (ta'sir darajasi) berasiz. Misol uchun, yo'qotish funksiyasini quyidagicha yozish mumkin:

Agar siz $\beta = 1/40$ deb belgilasangiz, algoritm **5,000 ta mobil ilova rasmlari va 200,000 ta internet rasmlari ga bir xil og'irlilik** beradi. Boshqacha aytganda, har bir mobil ilova rasmi internet rasmlariga qaraganda **40 barobar ko'proq ahamiyatga ega** bo'ladi. Siz, shuningdek, β ni boshqa qiymatlarga ham o'rnatishingiz mumkin — masalan, **dev to'plamida aniqlikni oshirishga yordam beradigan qiymatni tanlab**.

Qo'shimcha internet rasmlariga **kamroq og'irlilik berish orqali**, sizga juda katta neyron tarmoq qurishning hojati qolmaydi. Bu usul modelni **har ikki turdag'i ma'lumotlar ustida yaxshi ishlashini ta'minlaydi**, lekin mobil ilova foydalanuvchilari ma'lumotlariga ko'proq e'tibor qaratadi.

Bunday qayta og'irlilik berish (re-weighting) faqat quyidagi hollarda zarur bo'ladi:

1. Agar qo'shimcha ma'lumotlar (**masalan, internet rasmlari**) sizning **dev/test to'plamingizdan ancha farq qilsa**.
2. Yoki agar qo'shimcha ma'lumotlar **hajmi** sizga kerakli ma'lumotlar hajmidan **judu katta bo'lsa**.

40 Trening to‘plamdan dev to‘plamga umumlashtirish (Generalizing from the training set to the dev set)

Tasavvur qiling, siz mashinaviy o‘rganish tizimini yaratmoqdasiz va trening to‘plamingiz Internet tasvirlari hamda mobil ilova foydalanuvchilarining tasvirlaridan iborat. Biroq sizning dev va test to‘plamlaringiz faqatgina mobil ilova tasvirlaridan tashkil topgan. Shu holatda, agar sizning algoritmingiz dev/test to‘plamlarida yuqori xatoga ega bo‘lsa, bu quyidagi muammolardan birini bildiradi:

1. **Yuqori bias** – ya’ni, algoritm trening to‘plamida ham yaxshi ishlamayapti.
2. **Yuqori variance** – ya’ni, algoritm trening to‘plamida yaxshi ishlaydi, ammo shu taqsimotdagi yangi ma’lumotlarga yomon umumlashtiradi.
3. **Ma’lumotlar mos kelmasligi (data mismatch)** – ya’ni, trening ma’lumotlarida yaxshi ishlaydi, ammo dev/test to‘plamiga o‘xshash bo‘lmagan taqsimotda ishlay olmaydi.

Agar sizning modelingiz:

- Trening to‘plamda 1% xatoga ega bo‘lsa,
- Trening taqsimotdagi yangi ma’lumotlarda 1.5% xatoga ega bo‘lsa,
- Dev to‘plamda esa 10% xatoga ega bo‘lsa,

bu holat **data mismatch**, ya’ni ma’lumotlar mos kelmasligi muammosini ko‘rsatadi.

Bunday holatda nima qilish kerak?

Muammoning asl sababini aniqlash uchun siz mavjud trening ma’lumotlarini quyidagi **to‘rtta to‘plamga ajratishingiz** mumkin:

1. **Trening to‘plami** – model aynan o‘qitiladigan ma’lumotlar (masalan, internet + mobil tasvirlar).
2. **Training dev to‘plami** – trening bilan bir xil taqsimotdan olingan, lekin o‘qitishda qatnashmaydigan kichik to‘plam. Modelning umumlashtirish qobiliyatini baholash uchun kerak.
3. **Dev to‘plami** – testga o‘xshash, haqiqiy foydalanuvchi muhiti taqsimotini ifodalaydi (masalan, faqat mobil tasvirlar).
4. **Test to‘plami** – yakuniy baholash uchun, dev to‘plam bilan bir xil taqsimotdan.

Shundan so‘ng siz quyidagilarni tahlil qilishingiz mumkin:

- Trening xatosi (training error) – modelga berilgan ma’lumotlardagi ishlash holati.
- Training dev xatosi – shu taqsimotdagi yangi ma’lumotlarga umumlashtirish darajasi.
- Dev/Test xatosi – siz aslida qiziqayotgan muhitdagi (foydalanuvchining real ma’lumotlarida) model ishlashi.

41 Xatolik turlarini aniqlash: Bias, Variance va Data Mismatch

Mashina o'rganishda modelning ishlash sifatini tahlil qilishda **bias** (og'ish), **variance** (farqlilik), va **data mismatch** (ma'lumotlar mos kelmasligi) muammolarini to'g'ri aniqlash hal qiluvchi ahamiyatga ega. Bu tahlillar nafaqat modelni diagnostika qilishga yordam beradi, balki uni qanday rivojlantirish kerakligini ham ko'rsatadi. Quyidagi fikrlar sizni bu murakkab tushunchalarni mukammal anglash sari yetaklaydi.

Tasavvur qiling, siz mushuklarni aniqlovchi algoritm ustida ishlayapsiz. Insonlar ushbu vazifada deyarli 0% xatolik bilan ishlay olishadi — bu shuni anglatadiki, **ideal modelning optimal xatolik darajasi $\approx 0\%$** bo'lishi kerak.

Holat 1: Yuqori Variance

Agar sizning algoritmingiz quyidagi xatolik darajalariga ega bo'lsa:

- Trening to'plamda: **1%**
- Training dev to'plamda: **5%**
- Dev to'plamda: **5%**

Bu natijalar shuni ko'rsatadiki, sizning modelingiz **trening ma'lumotlarini yaxshi o'zlashtirgan**, lekin u umumlashtirishda qiyalmoqda. Ya'ni, bu holatda muammo **variance** bilan bog'liq. Bunday vaziyatda **ensembling**, **regularization**, yoki **ko'proq ma'lumotlar qo'shish** kabi variance kamaytiruvchi texnikalar yordam beradi.

Holat 2: Yuqori Bias

Endi quyidagicha bo'lsin:

- Trening to'plamda: **10%**
- Training dev to'plamda: **11%**
- Dev to'plamda: **12%**

Bu yerda xatolik darajalari deyarli barcha to'plamlarda yuqori, bu esa modelning asosiy o'rganish bosqichida ham xatolikka yo'l qo'yayotganini bildiradi. Bu **yuqori bias** degani. Sizning model oddiy, o'rganish quvvati past yoki noto'g'ri funksiyaga asoslangan. **Modelni murakkablashtirish, daha to'g'ri funksional shakl tanlash** yoki **yaxshiroq xususiyatlar chiqarish** bu muammoni hal qilishi mumkin.

Holat 3: Bias + Data Mismatch

Endi yana murakkabroq vaziyat:

- Trening to'plamda: **10%**
- Training dev to'plamda: **11%**
- Dev to'plamda: **20%**

Bu yerda siz ikki muammo bilan duch kelasiz. Birinchisi, **bias hali ham yuqori** – bu modelning o'zi trening to'plamida ham yomon ishlayotganini bildiradi. Ikkinchisi, dev to'plamdagagi xatolik juda keskin oshib ketgan – bu esa **data mismatch**, ya'ni trening va dev/test to'plamlarining taqsimoti bir-biriga o'xshamasligini bildiradi. Bunday holatda sizga **trening ma'lumotini dev/test ma'lumotlariga yaqinlashtirish**, yoki **dev ma'lumotlaridan ko'proq misollarni treningga qo'shish** foyda beradi.

Mushuk tasvirlarini aniqlovchi algoritm misolida davom etar ekanmiz, siz x-o'qida ikkita turli ma'lumotlar taqsimoti mavjudligini ko'rishingiz mumkin. Y-o'qida esa bizda uch xil xato turi mavjud: inson darajasidagi xato, algoritm o'rgangan (trening) misollardagi xato, va algoritm hali ko'rmagan (yangi) misollardagi xato.

Avvalgi bobda tahlil qilgan turli xatolar asosida ushbu jadvaldagи kataklarni to‘ldirish mumkin. Bunda, har bir taqsimot bo‘yicha (masalan, A — internet tasvirlari, B — mobil ilova tasvirlari) mos ravishda inson darajasidagi xato, o‘rgangan misollar bo‘yicha xato, va ko‘rmagan misollar bo‘yicha xato belgilanadi.

Agar xohlasangiz, ushbu jadvaldagи yana ikkita katakchani ham to‘ldirishingiz mumkin:

1. **Mobil tasvirlardagi inson darajasidagi xato (yuqori o‘ng katak):** Buni aniqlash uchun bir nechta odamga mobil ilova mushuk tasvirlarini belgilash vazifasini bering va ularning xato foizini o‘lchang. Bu orqali siz modelning ideal ishlashi kerak bo‘lgan darajani aniqlaysiz.
2. **Treningga kiritilgan mobil tasvirlar bo‘yicha model xatosi (pastki o‘ng katak):** Buning uchun mobil tasvirlar to‘plamidan kichik bir qismini ajratib, uni modelga trening uchun bering. So‘ngra, aynan shu o‘rgatilgan mobil tasvirlar bo‘yicha model xatosini o‘lchang. Bu, model bu taqsimotdagi ma’lumotlarga qanchalik yaxshi moslasha olishini ko‘rsatadi.

Ushbu ikki qo‘sishimcha katakni to‘ldirish orqali siz algoritm ikki xil ma’lumot taqsimotida — ya’ni **Internet tasvirlari (A)** va **Mobil ilova tasvirlari (B)** — qanday ishlayotganligi haqida qo‘sishimcha tushuncha olishingiz mumkin.

Shunday qilib, **algoritm qaysi turdagи xatolardan eng ko‘p aziyat chekayotganini aniqlasangiz**, sizga quyidagi yo‘nalishlardan qaysi biriga e’tibor qaratish lozimligini aniqlash osonlashadi:

- Biasni kamaytirish (agar trening xatolari yuqori bo‘lsa),
- Variansni kamaytirish (agar trening va dev/test o‘rtasida katta tafovut bo‘lsa),
- Ma’lumotlar mos kelmasligini kamaytirish (agar taqsimotlar o‘rtasida farq bo‘lsa).

Bu yondashuv yordamida siz model ishlashini chuqr tahlil qila olasiz va uni yaxshilash uchun aniq yo‘nalish tanlaysiz.

42 Ma’lumotlar mos kelmasligini (data mismatch) hal qilish

Tasavvur qiling, siz nutqni tanish (speech recognition) tizimi yaratgansiz. Ushbu tizim trening to‘plamida va trening dev (baholash) to‘plamida juda yaxshi natijalarga erishadi. Biroq, dev (tekshiruv) to‘plamida sust ishlaydi. Bunday holatda sizda **ma’lumotlar mos kelmasligi** muammosi mavjud. Unda nima qilish kerak?

Men quyidagi ikki bosqichli yondashuvni tavsiya qilaman:

1. **Trening to‘plami va dev to‘plami o‘rtasidagi farqlarni aniqlang.** Yani, qanday xususiyatlar bo‘yicha bu ikki to‘plam bir-biridan farq qiladi, shuni tushunishga harakat qiling.
2. **Algoritmingiz qiynalayotgan dev misollariga ko‘proq o‘xhash trening ma’lumotlarini topishga harakat qiling.**

Masalan, siz dev to‘plamdagи xatolarni tahlil qilasiz — 100 ta misolni qo‘lda ko‘rib chiqasiz va qayerda tizim xato qilayotganini tahlil qilasiz. Siz aniqlaysizki, dev to‘plamdagи ko‘pchilik audio yozuvlar avtomobil ichida yozilgan, ammo treningdagi yozuvlar asosan tinch, shovqinsiz muhitda yozilgan. Motor tovushi va yo‘l shovqini nutqni tanish tizimingiz natijalarini jiddiy yomonlashtirgan.

Bunday holda, siz **avtomobilda yozilgan** ko‘proq audio yozuvlardan iborat trening ma’lumotlarini qo‘lga kiritishga harakat qilishingiz kerak bo‘ladi. Xatolar tahlilining asosiy maqsadi — trening va dev to‘plamlar o‘rtasidagi **asosiy farqlarni aniqlash**, aynan mana shu farqlar “data mismatch” muammosiga sabab bo‘ladi.

Agar sizning trening va trening dev to‘plamingizda ham avtomobil ichida yozilgan ba’zi yozuvlar mavjud bo‘lsa, siz shu bo‘lim bo‘yicha algoritmingiz ishlashini alohida tekshirishingiz mumkin. Agar tizim avtomobil audio yozuvlariga trening to‘plamida yaxshi moslashgan bo‘lsa, ammo trening dev to‘plamida yomon natija ko‘rsatsa, demak sizning gipotezangiz — “ko‘proq avtomobil muhitidagi ma’lumotlar kerak” — yanada mustahkamlanadi.

Shu sababli biz oldingi bobda **dev/test to‘plamiga o‘xshash** ba’zi ma’lumotlarni treningga qo‘sishni tavsiya qilgan edik. Bu yondashuv, algoritmi aynan siz muhim deb bilgan ma’lumotlarga qanday moslashayotganini tekshirishga imkon beradi.

Afsuski, bu jarayonda aniq kafolatlar yo‘q. Agar sizda dev to‘plamga yaqinroq trening ma’lumotlarini topishning imkonи bo‘lmasa, tizimni yaxshilash bo‘yicha aniq yo‘lni topa olmasligingiz mumkin. Shunday bo‘lsa-da, **data mismatch muammosini aniqlash va unga mos yechim qidirish** har qanday ML loyihasida juda muhim bosqich hisoblanadi.

43 Sun’iy ma’lumotlar yaratish (Artificial data synthesis)

Aytaylik, sizning nutqni tanish tizimingiz avtomobil ichida yozib olingan tovushlarga o‘xshash ko‘proq ma’lumotlarga muhtoj. Avtomobilda yurib, yangi audio yozuvlar yig‘ishning o‘rniga, buni **sun’iy ma’lumotlar yaratish** orqali osonroq amalga oshirish mumkin.

Misol uchun, siz internet orqali **ko‘plab avtomobil yoki yo‘l shovqini** audio kliplarini yuklab olishingiz mumkin. Shuningdek, sizda **jim xona sharoitida yozilgan odam nutqi** bo‘yicha katta hajmda trening ma’lumotlari mavjud. Agar siz odamning nutqi yozilgan audio klipga avtomobil shovqinini qo‘sangiz, bu yangi audio xuddi odam mashinada gapirayotgandek eshitiladi. Ana shu jarayon orqali siz **sun’iy ravishda avtomobil ichida yozilgandek eshitiladigan katta hajmda ma’lumotlar** yaratishingiz mumkin.

Bu yondashuv ko‘p hollarda ishlaydi. Masalan, **mushuklarni aniqlovchi rasm tanish tizimi** ustida ishlayotgan bo‘lsangiz va dev to‘plamdagи rasmlarda ko‘proq harakat (motion blur) mavjudligini sezсангиз — bu odatda foydalanuvchilar telefonni ozgina siljитib suratga olayotgani sababli yuz beradi — siz internetdan olingan tiniq rasmlarga sun’iy harakat xiraligini (motion blur) qo‘sishningiz mumkin. Bu ularni dev to‘plamga o‘xshash qiladi.

Ammo sun’iy ma’lumotlar yaratishda ayrim **qiyinchiliklar** mavjud. Ba’zida, **inson ko‘ziga real ko‘ringan ma’lumotlar, kompyuter uchun real bo‘lmasligi** mumkin. Misol uchun:

- Tasavvur qiling, sizda **1,000 soatlik odam nutqi** mavjud, lekin faqat **1 soatlik avtomobil shovqini** bor. Agar siz shu 1 soatlik avtomobil shovqinini qayta-qayta ishlatsangiz, sun’iy yaratgan ma’lumotlar to‘plamida **doim bir xil shovqin** bo‘ladi. Inson uchun bu sezilmasligi mumkin (hamma mashina shovqini o‘xshash tuyuladi), lekin **algoritm aynan shu 1 soatlik shovqinga haddan tashqari moslashib**, haqiqiy, yangi turdagи shovqinda yomon ishlashi mumkin.
- Yoki sizda **1,000 soatlik avtomobil shovqini** bo‘lishi mumkin, lekin bu yozuvlar faqat **10 xil mashinadan** olingan bo‘lsa, algoritm aynan shu 10 ta mashinaga moslashib qolishi mumkin. Keyinchalik yangi mashina shovqinlari bilan ishlaganda u yomon natijalar berishi mumkin.

Afsuski, bunday muammolarni **aniqlash juda qiyin** bo‘lishi mumkin.

Shunday qilib, **sun’iy ma’lumotlar yaratish** — modelni dev/test to‘plamdagи holatlarga moslashtirishda samarali usul, lekin u ehtiyyotkorlik bilan bajarilishi va turli xil real sharoitlarni inobatga olgan holda amalga oshirilishi kerak.

Keling, yana bir misolni ko‘rib chiqamiz: siz **avtomobilarni aniqlovchi kompyuter ko‘rish tizimi** (computer vision system) yaratmoqdasiz. Siz video o‘yin kompaniyasi bilan hamkorlik qilasiz va ular sizga bir nechta avtomobilning **3D grafik modellarini** beradi. Shundan so‘ng siz bu modellar asosida **sun’iy avtomobil rasmlari** yaratib, algoritmingizni o‘rgatmoqchisiz.

Bu rasmiylashtirilgan rasmlar **inson ko‘ziga juda realistik** ko‘rinishi mumkin. Ammo bu yondashuv, ko‘pchilik tomonidan taklif etilgan bo‘lishiga qaramay, **amalda yaxshi ishlamasligi mumkin**.

Nega?

- Video o‘yinda, masalan, **atigi 20 ta avtomobil dizayni** bo‘lishi mumkin.
- Yangi 3D avtomobil modelini yaratish juda qimmat. Shuning uchun siz o‘yinni o‘ynayotganda, ehtimol, bir xil avtomobilarni qayta-qayta ko‘rayotganingizni sezmay qolishingiz mumkin — ular faqat boshqa rangda bo‘lishi mumkin.
- Bu tasvirlar sizga **haqiqiy ko‘rinadi**, lekin **haqiqiy dunyodagi** avtomobilarning xilma-xilligini hisobga olsak, o‘sha 20 ta sun’iy avtomobil dizayni bu xilma-xillikning **juda kichik qismini** tashkil qiladi.

Shu sababli, agar siz 100,000 ta trening misolini faqat shu 20 xil avtombildan olgan bo‘lsangiz, algoritmingiz **shu 20 ta dizaynga haddan tashqari moslashib qoladi (overfit qiladi)** va yangi **avtomobillar** bilan ishlaganda (dev/test to‘plamlarida) yomon ishlaydi.

Shaxsan men ishlagan jamoalarda ham, ba’zida **haqiqiy dunyoga juda yaqin bo‘lgan sun’iy ma’lumotlarni tayyorlash** uchun **haftalar vaqt ketgan**. Ammo agar siz **realistik detallarni to‘g‘ri ko‘rsata olsangiz**, unda siz katta hajmda, foydali trening ma’lumotlariga ega bo‘lasiz.

Debugging inference algorithms — Chiqim (natija) algoritmlarini nosozlikdan tozalash (xatoliklarni aniqlash)

44 The Optimization Verification Test (Tarjimasi: Optimallashtirishni tekshirish testi)

Tasavvur qiling, siz nutqni tanish (speech recognition) tizimi yaratmoqdasiz. Ushbu tizim A deb belgilangan audio klipni qabul qiladi va har bir mumkin bo'lgan natija jumlesi S uchun biror ScoreA(S) baho hisoblab chiqadi.

Muammo:

Ingliz tilida 50,000 ta so'z bo'lsa, uzunligi N bo'lgan barcha jumlalar soni (**50,000**)^N bo'ladi. Buni to'liq hisoblash amalda imkonsiz. Shu sababli, **beam search** kabi yaqinlashtiruvchi (**approximate**) qidiruv algoritmlari ishlataladi. Beam search K ta eng yaxshi nomzodni qidiruv davomida saqlab qoladi.

Ammo bu algoritmlar har doim optimal (eng yaxshi) S ni topishiga kafolat yo'q.

Misol:

Aytaylik, audio yozuvda kishi "I love machine learning" deb aytgan.

Ammo tizim "**I love robots**" degan noto'g'ri natija beradi.

Bu xatolikning ikki sababi bo'lishi mumkin:

1. Qidiruv (search) algoritmida xatolik

Beam search **to'g'ri** S* ni topa olmadı, garchi u $ScoreA(S) > ScoreA(S_{out})^*$ bo'lsa ham.

2. Baholash funksiyasida (scoring function) xatolik

Tizim $ScoreA(S) < ScoreA(S_{out})^*$ deb noto'g'ri baho berdi. To'g'ri matnga past, noto'g'ri matnga yuqori baho berdi.

Qanday aniqlash mumkin: Optimization Verification Test

1. **Sout** = tizim chiqargan noto'g'ri matn ("I love robots")
2. **S*** = to'g'ri matn ("I love machine learning")
3. **ScoreA(S*)** va **ScoreA(Sout)** ni hisoblang.

Ikkita holat yuzaga chiqadi:

1-holat: ScoreA(S*) > ScoreA(Sout)

Demak, **learning (organish)** algoritmingiz yaxshi ishlagan.

Lekin **qidiruv algoritmi** (masalan, beam search) eng yuqori ballni topishda xatoga yo'q qo'yan.

Qidiruv algoritmini yaxshilang. Masalan, **beam width** (n tadan nechta variantni saqlash) ni oshiring.

2-holat: ScoreA(S*) ≤ ScoreA(Sout)

Bu holda muammo **baholash funksiyasida**.

To'g'ri javobga past ball, noto'g'riga esa yuqori ball bergan.

Learning modelni yaxshilang — ya'ni **ScoreA(S)** ni qanday o'rganayotganingizni o'zgartiring (masalan, model arxitekturasi, loss function, data sifati va h.k.).

Amaliyotda qo'llash:

- Dev setdagi har bir xatolik uchun S^* va S_{out} ni olaylik.
- Har biri uchun $Score_A(S^*) > Score_A(S_{out})$ ni tekshiring.
- Statistikani tuzing:
Misol uchun, agar **xatoliklarning 95%** baholash funksiyasi sababli bo'lsa, siz **learning modelga e'tibor qaratishingiz** kerak.
Faqat 5% optimallashtirish (search) sababli bo'lsa, **beam search ni yaxshilash ko'p foyda bermaydi.**

45 Optimization Verification Test – umumiy shakli (Optimallashtirishni tekshirish testining umumlashtirilgan shakli)

Mashina o'r ganish (ML) sohasida biz ko'pincha kirish qiymatlari (x) va chiqish natijalari (y) o'rta sidagi aloqani o'r ganishimiz kerak bo'ladi. Har bir kirish uchun turli chiqishlar mumkin va ularning har biri baholanishi mumkin. Buning uchun $Score_x(y)$ deb nomlanuvchi baholash funksiyasidan foydalaniladi, u y chiqishining x kirishga mosligini bildiradi.

Ko'p hollarda biz $Score_x(y)$ funksiyasini maksimal qiladigan y qiymatini topishimiz kerak bo'ladi. Boshqacha aytganda, biz:

$y = \arg \max Score_x(y)$ qiymatini topishga harakat qilamiz.

Ammo bu amaliyotda juda murakkab bo'lishi mumkin, chunki chiqishlar soni juda ko'p (masalan, 50 000 ta so'zdan iborat jumlalar kombinatsiyasi). Shuning uchun **aniq (exact)** natija topilmaydi, balki **yaqinlashtiruvchi (approximate)** qidiruv algoritmlari (masalan, *beam search*) ishlataladi.

Bu yerda ikki xil muammo yuzaga kelishi mumkin:

1. **Qidiruv algoritmi muammosi** – yaqinlashtiruvchi algoritm eng yaxshi chiqishni topa olmayapti.
2. **Baholash funksiyasi muammosi** – $Score_x(y)$ funksiyasi to'g'ri javobga yetarlicha yuqori baho bermayapti.

Qaysi biri muammo ekanligini aniqlash uchun biz **Optimization Verification test** deb nomlangan testdan foydalanamiz.

Sun'iy intellekt (AI) tizimlarida tez-tez uchraydigan holatlardan biri – **chiqishlar (y)** soni juda katta bo'lganligi sababli **to'liq qidiruv (exact search)** qilishning imkoniy yo'qligidir. Masalan, Xitoycha jumlalarni Inglizchaga tarjima qiluvchi tizim yaratmoqdasiz. Bu yerda:

- **C** – Xitoycha jumla (kirish, ya'ni x)
- **E** – inglizcha tarjima (chiqish, ya'ni y)
- Siz har bir E uchun $Score_C(E)$ deb ataluvchi baho hisoblab chiqasiz. Masalan, bu $P(E|C)$ bo'lishi mumkin – ya'ni, Xitoycha C jumlesi berilganda E tarjimanining ehtimoli.

Ammo, ingliz tilida mumkin bo'lgan barcha jumlalar soni juda katta bo'lgani uchun, siz **to'liq qidiruvni bajara olmaysiz**. Shuning uchun **heuristik qidiruv algoritmi**, masalan *beam search*, ishlataladi.

46 Optimization Verification Test – Matn Ko‘rinishida Tushuntirish

Siz helikopterni murakkab harakatlar, masalan, **dvigatelsiz qo‘nish** (autorotation) kabi manevrarni bajara oladigan tarzda boshqarishni o‘rgatmoqchisiz. Maqsadingiz – helikopterni xavfsiz qo‘nish bilan yakunlanuvchi **traektoriya T** bo‘yicha uchirish.

Buning uchun siz **reward function** (mukofot funksiyasi) $R(T)$ aniqlaysiz. Bu funksiya har bir traektoriyaga baho beradi:

- Agar helikopter halokatga uchrasa $\rightarrow R(T) = -1000$ (katta salbiy baho)
- Agar yumshoq va xavfsiz qo‘nsa $\rightarrow R(T)$ musbat baho bo‘ladi, qo‘nishdagi silliqlik, aniq joyga qo‘nish va yo‘lovchilar uchun qulayliklarga qarab baholanadi.

Bu reward funksiyasi **qo‘lda tuziladi** va qo‘nishdagi turli jihatlarni muvozanatlashtiradi.

Shundan so‘ng, sizning reinforcement learning algoritmingiz $R(T)$ ni **maksimizatsiya qilishga harakat qiladi**:

$$T_{\text{opt}} = \operatorname{argmax}_T R(T)$$

Muammo: Model natijalari insonnikidan yomon

Siz reward funksiyasini aniqladingiz, modelni o‘rgatdingiz. Lekin natijalar inson uchuvchisinikiga qaraganda yomonroq:

- Qo‘nish xavfiroq yoki qattiqroq.
- Traektoriya istalganidek chiqmagan.

Endi savol: **Muammo qayerda?**

1. **RL algoritmi** noto‘g‘ri traektoriya tanlayaptimi?
2. Yoki **reward funksiyasi** noto‘g‘ri baho berayaptimi (yaxshi natijani past, yomon natijani baland baholayaptimi)?

Optimization Verification Test – qanday aniqlash mumkin?

Faraz qilaylik:

- T_{human} = inson uchuvchisi bajargan, yaxshi qo‘nishga olib keluvchi traektoriya
- T_{out} = sizning algoritmingiz bajargan, yomonroq natijaga olib kelgan traektoriya

Endi **quyidagini tekshiramiz**:

$$R(T_{\text{human}}) > R(T_{\text{out}}) ?$$

1-holat: $R(T_{\text{human}}) > R(T_{\text{out}})$

Bu shuni bildiradiki, sizning reward funksiyangiz to‘g‘ri ishlayapti. U yaxshi traektoriyani yomonroq traektoriyaga qaraganda yuqoriroq baholayapti.

Muammo **algoritmda** – u optimal natijani topolmayapti. Demak, **reinforcement learning algoritmini yaxshilash** kerak.

2-holat: $R(T_{\text{human}}) \leq R(T_{\text{out}})$

Bu esa reward funksiyasining noto‘g‘ri ishlayotganini bildiradi. U aslida yaxshi bo‘lgan traektoriyani pastroq baholayapti.

Muammo **reward funksiyasida** – uni qayta ko‘rib chiqib, **real qo‘nish sifatiga mos** bo‘ladigan tarzda tuzish lozim.

End-to-end deep learning

47 End-to-end o‘rganishning rivojlanishi

Aytaylik, siz onlayn mahsulot sharhlarini tahlil qilish va muallif ushbu mahsulotni yoqtirganmi yoki yo‘qmi, avtomatik tarzda aniqlaydigan tizim yaratmoqchisiz. Masalan, siz quyidagi sharhnijobiy deb tan olishga umid qilasiz:

Bu juda ajoyib pol artgich!

Va quyidagisini esa salbiy deb:

Bu pol artgich sifatsiz — uni sotib olganimdan afsusdaman.

Ijobiy yoki salbiy fikrlarni aniqlash muammosi “hissiyotlar tasnifi” (sentiment classification) deb ataladi.

Ushbu tizimni qurish uchun siz ikki komponentdan iborat “pipeline” yaratishingiz mumkin:

1. **Parser:** matnga eng muhim so‘zlarni aniqlovchi ma’lumotlar bilan belgi qo‘yadigan tizim. Masalan, siz parserdan barcha sifatlar va otlarni belgilash uchun foydalanishingiz mumkin. Shunda quyidagi belgilangan matn hosil bo‘ladi:
Bu ajoyib<Adjective> pol artgich<Noun>!
2. **Hissiyotlar tasniflagichi** (Sentiment classifier): belgilangan matnni kiritma sifatida qabul qiluvchi va umumiyl hissiyotni bashorat qiluvchi o‘rganish algoritmi. Parserning belgilari bu algoritmga juda katta yordam berishi mumkin: sifatlarga yuqori og‘irlilik berish orqali, algoritm muhim so‘zlar — masalan, “ajoyib” kabi so‘zlarni tezda aniqlay oladi va “bu” kabi kamroq muhim so‘zlarni e’tibordan chetda goldiradi.

Biz ushbu ikki komponentli “pipeline”ni quyidagicha tasvirlashimiz mumkin:

Yaqinda pipeline tizimlarini bitta o‘rganish algoritmi bilan almashtirishga moyillik paydo bo‘ldi. Ushbu vazifa uchun end-to-end o‘rganish algoritmi shunchaki kirish sifatida asl, xom matnni — masalan, “Bu juda ajoyib pol artgich!” — qabul qiladi va bevosita hissiyotni aniqlashga harakat qiladi:

Neyron tarmoqlar end-to-end (boshidan oxirigacha) o‘rganish tizimlarida keng qo‘llaniladi. “End-to-end” atamasi shuni anglatadiki, biz o‘rganish algoritmiga kirishdan bevosita kerakli chiqishga o‘tishni so‘rayapmiz. Ya’ni, o‘rganish algoritmi tizimning “kirish uchini” to‘g‘ridan-to‘g‘ri “chiqish uchi” bilan bog‘laydi.

Ma’lumotlar mo‘l bo‘lgan muammolarda end-to-end tizimlar nihoyatda muvaffaqiyatli bo‘lgan. Biroq, ular har doim ham eng yaxshi tanlov emas. Keyingi bir nechta boblarda end-to-end tizimlarning boshqa misollari keltiriladi, shuningdek, ularni qachon ishlatish va qachon ishlatmaslik kerakligi haqida tavsiyalar beriladi.

48 End-to-end o'rganishning yana boshqa misollari

Tasavvur qiling, siz nutqni tanish (speech recognition) tizimi qurmoqchisiz. Siz bu tizimni quyidagi uchta komponentdan iborat qilib qurishingiz mumkin:

Komponentlar quyidagicha ishlaydi:

1. **Xususiyatlarni hisoblash:** Oldindan ishlab chiqilgan (qo'lda yaratilgan) xususiyatlar, masalan MFCC (Mel-frequency cepstrum coefficients) xususiyatlari ajratib olinadi. Bu xususiyatlar nutq mazmunini aks ettiradi, lekin nutqchining ovoz balandligi kabi unchalik muhim bo'lмаган xususiyatlarni e'tiborsiz qoldiradi.
2. **Fonema aniqlovchi (Phoneme recognizer):** Ba'zi tilshunoslar tilning asosiy tovush birliklari — *fonemalar* mavjudligiga ishonadi. Masalan, "keep" so'zidagi boshidagi "k" tovushi va "cake" so'zidagi "c" tovushi aslida bir xil fonema hisoblanadi. Ushbu modul audio yozuvdagi fonemalarni aniqlashga harakat qiladi.
3. **Yakuniy aniqlovchi:** Aniqlangan fonemalar ketma-ketligidan foydalanib, yakuniy matnli transkripsiyanı tuzadi.

Bunga qarama-qarshi tarzda, end-to-end tizimi esa audio yozuvni kirish sifatida olib, **bevosita yakuniy transkripsiyanı** chiqarishga harakat qiladi:

Hozirgacha biz faqat to'liq **chiziqli** (ya'ni ketma-ket) mashinani o'rganish "pipeline"larini tasvirladik: har bir bosqichda natija keyingisiga ketma-ket tarzda uzatiladi. Biroq, **pipelinelar** bundan murakkabroq bo'lishi mumkin.

Masalan, quyida **avtonom avtomobil** (o'z-o'zini boshqaruvchi mashina) uchun oddiy arxitektura keltirilgan:

(izoh: *bu yerda rasm yoki blok diagramma keltirilishi kerak, chunki matn "architecture" haqida gapiryapti.*)

Bu kabi tizimlar bir nechta modullardan iborat bo'lib, ular bir vaqtning o'zida ishlashi, yoki turli modullar o'zaro parallel ravishda ma'lumot almashishi mumkin. Shu sababli pipeline faqat ketma-ket emas, **filiallangan yoki tarmoqlangan shaklga** ega bo'lishi mumkin.

Hozirgacha biz faqat to'liq **chiziqli** (ya'ni ketma-ket) mashinani o'rganish "pipeline"larini tasvirladik: har bir bosqichda natija keyingisiga ketma-ket tarzda uzatiladi. Biroq, **pipelinelar** bundan murakkabroq bo'lishi mumkin.

Masalan, quyida **avtonom avtomobil** (o'z-o'zini boshqaruvchi mashina) uchun oddiy arxitektura keltirilgan:

(izoh: *bu yerda rasm yoki blok diagramma keltirilishi kerak, chunki matn "architecture" haqida gapiryapti.*)

Bu kabi tizimlar bir nechta modullardan iborat bo'lib, ular bir vaqtning o'zida ishlashi, yoki turli modullar o'zaro parallel ravishda ma'lumot almashishi mumkin. Shu sababli pipeline faqat ketma-ket emas, **filiallangan yoki tarmoqlangan shaklga** ega bo'lishi mumkin.

Garchi end-to-end o'rganish (ya'ni, kirishdan chiqishgacha bo'lgan to'liq avtomatlashtirilgan o'rganish) ko'plab muvaffaqiyatlarga erishgan bo'lsa-da, bu har doim ham eng yaxshi yondashuv emas. Masalan, **end-to-end nutqni tanish** tizimlari juda yaxshi natija bermoqda.

Ammo, **avtonom boshqaruv (o'z-o'zini boshqaruvchi mashinalar)** uchun end-to-end o'rganishdan foydalangan holda yakuniy tizim yaratishga men biroz shubha bilan qarayman. Keyingi boblarda bu shubhaning sabablari tushuntiriladi.

49 End-to-end o'rganishning afzalliklari va kamchiliklari

Keling, oldingi misoldagi **nutqni tanish pipeline (bosqichma-bosqich)** tizimini yana ko'rib chiqamiz:

Bu an'anaviy tizim quyidagicha bo'lishi mumkin:

Ko'p hollarda bu **pipeline (bosqichli tizim)** quyidagi kabi "qo'lda ishlab chiqilgan" komponentlardan iborat bo'ladi:

- **MFCC (Mel-frequency cepstral coefficients)** — bu **qo'lda ishlab chiqilgan audio xususiyatlari to'plami**. Ular audioni yaxshi tarzda ifodalarydi, lekin **ba'zi axborotlarni yo'qotish orqali** signalni soddalashtiradi.
- **Fonemalar** — bu **tilshunoslar tomonidan ixtiro qilingan** tovush birliklari. Fonemalar nutq tovushlarining **mukammal bo'limgan** modelidir. Agar fonema haqiqiy tovushlar uchun yetarlicha mos bo'lmasa, unda tizim faqat fonemaga asoslanib ishlasa, **uning ishlashi cheklanishi mumkin**.

Bu **qo'lda yaratilgan komponentlar** tizimning potensialini chegaralaydi. Shunga qaramay, ular **ba'zi afzalliklarni ham beradi**:

MFCC xususiyatlari:

- Nutqdagi **kontentga ta'sir qilmaydigan jihatlarga**, masalan, **gapiruvchining tovush balandligiga** nisbatan barqaror (robust).
- Shu sababli, ular **model uchun masalani soddalashtirishga yordam beradi**.

Fonemalar:

- Agar ular nutq uchun **yaxshi model** bo'lsa, algoritm **asosiy tovush birliklarini tushunishiga yordam beradi**.
- Bu esa **model samaradorligini oshiradi**.

Kam ma'lumot bo'lsa, ko'proq qo'lda yaratilgan komponentlarga ega tizimlar afzal bo'lishi mumkin.

MFCC va fonemalar kabi komponentlar yordamida **algoritmga oldindan berilgan bilim** qo'shiladi, bu esa **ma'lumot yetishmasligida foydali bo'ladi**.

Ushbu tizimda inson tomonidan maxsus ishlab chiqilgan bilimlar mavjud emas. Shuning uchun, agar o'quv (trening) to'plami kichik bo'lsa, u inson tomonidan ishlab chiqilgan pipeline tizimidan yomonroq ishlashi mumkin.

Biroq, agar o'quv to'plami katta bo'lsa, u MFCC yoki fonema asosidagi tasvirlarning cheklovlar bilan to'siqlanmaydi. Agar o'rganish algoritmi yetarlicha katta neyron tarmoq bo'lsa va yetarlicha o'quv ma'lumotlari bilan o'rgatilsa, u juda yaxshi natijalarga erishishi mumkin, hatto optimal xatolik darajasiga yaqinlashishi mumkin.

End-to-end (boshidan oxirigacha) o'rganish tizimlari odatda o'zini yaxshi ko'rsatadi, agar "ikki uchi" — ya'ni, kirish uchi va chiqish uchi uchun ko'p miqdorda belgilangan ma'lumot mavjud bo'lsa. Bu misolda, bizga (audio, transkript) juftliklaridan iborat katta ma'lumotlar to'plami kerak bo'ladi. Agar bu turdag'i ma'lumotlar mavjud bo'lmasa, end-to-end o'rganishga juda ehtiyyotkorlik bilan yondoshuv lozim.

Agar siz mashinani o'rganish muammosi ustida ishlayotgan bo'lsangiz va o'quv to'plamingiz juda kichik bo'lsa, unda algoritmingizning asosiy bilimlari sizning insoniy tushunchalariningizdan kelib chiqadi. Ya'ni, inson tomonidan ishlab chiqilgan komponentlardan.

Agar siz end-to-end tizimdan foydalanmaslikni tanlasangiz, unda siz pipeline'ning qanday bosqichlardan iborat bo'lishini va ularning qanday bog'lanishini hal qilishingiz kerak bo'ladi. Keyingi boblarda shunday pipeline tizimlarini loyihalash bo'yicha ba'zi takliflarni ko'rib chiqamiz.

50 Pipeline (bosqichli) komponentlarini tanlash: Ma'lumotlar mavjudligi

Agar siz end-to-end bo'lmanan pipeline tizimini qurayotgan bo'lsangiz, pipeline'ning komponentlari uchun qanday yaxshi nomzodlar bor? Pipeline'ni qanday loyihalashingiz tizimning umumiyligi ishlashiga katta ta'sir ko'rsatadi. Muhim omillardan biri — har bir komponentni o'rgatish uchun ma'lumotlarni yig'ish osonmi yoki yo'qmi, shuni aniqlashdir. Masalan, quyidagi avtonom haydash arxitekturasini ko'rib chiqing:

Siz mashinani o'rganish (machine learning) yordamida mashinalar va piyodalarni aniqlashingiz mumkin. Bundan tashqari, ular uchun ma'lumotlarni topish ham qiyin emas: kompyuter ko'rish (computer vision) sohasida mashinalar va piyodalar belgilangan juda ko'p sonli ma'lumotlar to'plamlari mavjud. Siz, shuningdek, kraudsorsing (masalan, Amazon Mechanical Turk) orqali yanada katta ma'lumotlar to'plamini olishingiz mumkin. Shunday qilib, mashina aniqlagich va piyoda aniqlagich yaratish uchun o'quv ma'lumotlarini olish nisbatan oson.

Bunga qarama-qarshi ravishda, sof end-to-end yondashuvni ko'rib chiqing:

Bunday tizimni o'rgatish uchun sizga katta hajmdagi (Rasm, Boshqaruv yo'nalishi) juftliklari kerak bo'ladi. Bunday ma'lumotlarni yig'ish juda ko'p vaqt va mablag' talab qiladi, chunki odamlarni avtomobilarni boshqarishga jalb qilib, ularning boshqaruv harakatlarini yozib borish kerak bo'ladi. Buning uchun maxsus uskunalangan avtomobillar parki va juda ko'p har xil holatlarni qamrab oladigan haydash jarayonlari talab etiladi. Bu esa end-to-end tizimni o'rgatishni murakkablashtiradi. Boshqa tomonдан, belgilangan mashina yoki piyoda tasvirlari uchun katta ma'lumotlar to'plamini olish ancha oson.

Umuman olganda, agar pipeline'ning "oraliq modullari"ni (masalan, mashina aniqlagich yoki piyoda aniqlagich) o'rgatish uchun ko'p ma'lumotlar mavjud bo'lsa, unda siz ko'p bosqichli pipeline'dan foydalanishni o'ylab ko'rishingiz mumkin. Bu struktura afzal bo'lishi mumkin, chunki mavjud barcha ma'lumotlardan oraliq modullarni o'rgatishda foydalanish imkonini beradi.

To'liq end-to-end ma'lumotlar mavjud bo'lguncha, men avtomatlashtirilgan haydash uchun non-end-to-end yondashuvni ancha istiqbolli deb hisoblayman: uning arxitekturasi mavjud ma'lumotlar hajmiga ko'proq mos keladi.

51 Pipeline komponentlarini tanlash: Vazifa soddaligi

Faqatgina ma'lumotlarning mavjudligi emas, balki pipeline komponentlarini tanlashda yana bir omilni ham hisobga olish kerak: har bir komponent qanday darajada sodda vazifani bajaradi? Siz har bir bosqichda alohida o'rganish yoki qurish oson bo'lgan komponentlardan foydalanishga harakat qilishingiz kerak.

Lekin biror komponentni "o'rganish oson" degani nima?

Quyidagi mashinaviy o'rganish (machine learning) vazifalariga qarang, ular osondan murakkabroqqa qarab tartiblangan:

1. Rasm haddan tashqari yorug' (overexposed) ekanligini aniqlash
2. Rasm yopiq joyda (indoor) yoki ochiq joyda (outdoor) olinganini aniqlash
3. Rasmda mushuk bor-yo'qligini aniqlash
4. Rasmda qora va oq tukli mushuk bor-yo'qligini aniqlash
5. Rasmda Siam (Siamese) zotli mushuk bor-yo'qligini aniqlash

Ularning har biri bu — ikkilik tasvir klassifikatsiyasi vazifasi bo'lib, sizga biror rasm beriladi va siz javob sifatida 0 yoki 1 qaytarishingiz kerak. Ammo ro'yxat boshidagi vazifalar neyron tarmoq uchun "osonroq" tuyuladi. Bu osonroq vazifalarni kamroq o'quv misollar bilan o'rganish mumkin bo'ladi.

Hozirgi vaqtida mashinaviy o'rganishda biror vazifa qanchalik oson yoki qiyin ekanligini aniqlovchi rasmiy ta'rif mavjud emas. Deep learning va ko'p qatlamlili neyron tarmoqlar ommalashgani sababli, ba'zida vazifa "oson" deb ataladi agar u kamroq hisoblash bosqichlari bilan (ya'ni sayoz, "shallow" neyron tarmoq orqali) hal etilsa, va "qiyin" deb ataladi agar u ko'proq hisoblash bosqichlarini talab etsa (ya'ni chuqur, "deep" neyron tarmoq kerak bo'lsa). Ammo bu ta'riflar hozircha norasmiy hisoblanadi.

Agar siz murakkab bir vazifani olib, uni sodda kichik vazifalarga bo'la olsangiz, va ushbu kichik vazifalarni bosqichma-bosqich aniq kodlasangiz, bu orqali siz algoritmgaga oldindan ma'lumot (prior knowledge) berasiz. Bu esa algoritmgaga vazifani samaraliroq o'rganishga yordam beradi.

Aytaylik, siz Siam mushuklarini aniqlovchi (detektor) tizim yaratmoqchisiz. Bu esa to'liq end-to-end (boshidan oxirigacha) arxitekturasi bo'ladi:

Bunga zid ravishda, siz quyidagi ikki bosqichli pipeline (quvurlar tizimi) arxitekturasidan foydalanishingiz mumkin:

Birinchi bosqich (mushuk aniqlovchi) tasvirdagi barcha mushuklarni aniqlaydi.

Ikkinci bosqichda esa aniqlangan har bir mushukning kesib olingen (cropped) tasviri (birma-bir) mushuk zoti klassifikatoriga uzatiladi va agar aniqlangan mushuklardan biri Siam zoti bo'lsa, tizim yakuniy chiqishda 1 (ha) natijasini beradi.

Faqat 0/1 belgilaridan foydalangan holda to'liq end-to-end klassifikatorni o'qitish bilan solishtirganda, pipeline'dagi har ikkala komponent — mushuk aniqlovchi va mushuk zoti klassifikatori — ancha osonroq o'rganiladi va sezilarli darajada kamroq ma'lumot talab qiladi. Yakuniy misol sifatida, keling, avtonom haydovchi tizimi (autonomous driving pipeline) ni yana bir bor ko'rib chiqamiz.

Tarjimon: Sh. Ne'matov

Tahrirchi: A.Dadajonov

Ushbu pipeline (quvurlar zanjiri) dan foydalangan holda siz algoritmga quyidagi uchta asosiy haydash bosqichi mavjudligini bildirayotgan bo‘lasiz:

1. Boshqa mashinalarni aniqlash
2. Piyodalarni aniqlash
3. Mashinangiz uchun yo‘l rejasini tuzish

Bundan tashqari, ushbu har bir bosqich nisbatan oddiyroq funksiyalar bo‘lib, ularni sof end-to-end (boshdan oxirgacha) yondashuvga qaraganda kamroq ma’lumotlar bilan o‘rganish mumkin.

52 To‘g‘ridan-to‘g‘ri boy chiqishlarni o‘rganish

Tasvirni klassifikatsiya qilish algoritmi kiritma sifatida biror tasvirni \hat{x} oladi va natijada ob'ekt turini bildiruvchi butun sonni chiqaradi.

Savol: **Algoritm butun son emas, balki tasvirni to‘liq ifodalovchi butun bir gapni (jumlanı) chiqarishi mumkinmi?**

y = “Yashil daraxtlar va yashil maysazor fonida yo‘ldan harakatlanayotgan sariq avtobus.” An’anaviy nazorat ostida o‘rgatiladigan (supervised learning) ilovalarda odatda \hat{y} funksiyasi o‘rganiladi, bu yerda chiqish \hat{y} odatda butun son yoki haqiqiy son bo‘ladi. Misol uchun: End-to-end chuqur o‘rganish (deep learning) sohasidagi eng hayajonli yutuqlardan biri shuki, u bizga oddiy raqamdan ancha murakkabroq bo‘lgan chiqishlar \hat{y} ni bevosita o‘rganishga imkon bermoqda. Yuqorida rasmga izoh yozish (image-captioning) misolida, neyron tarmoq rasmni kirish sifatida (\hat{y}) olib, bevosita izohni (\hat{y}) chiqarishi mumkin. Bu chuqur o‘rganishda kuchayib borayotgan tendensiyadir: agar sizda to‘g‘ri (kirish, chiqish) juftliklar mavjud bo‘lsa, ba’zida siz butun bir gap, rasm, audio yoki oddiy raqamdan ko‘ra boyroq bo‘lgan boshqa natijalarni ham end-to-end tarzda o‘rgatishingiz mumkin.

Xatolarni qismlarga ajratib tahlil qilish (Error analysis by parts)

53 Xatolarni qismlar bo'yicha tahlil qilish (Error analysis by parts)

Aytaylik, sizning tizimingiz murakkab mashinaviy o'rganish (machine learning) pipeline asosida qurilgan va siz uning samaradorligini oshirmoqchisiz. Pipeline'ning qaysi qismini yaxshilash kerak? Har bir xatoni pipeline'ning aniq bir qismiga bog'lash orqali siz qaysi qism ustida ishlashga ustuvorlik berishni hal qilishingiz mumkin.

Keling, bunga misol sifatida **Siam mushugini aniqlovchi klassifikator** tizimini olaylik:

Birinchi qism — **mushuk aniqlovchi modul**, tasvirdagi mushuklarni aniqlaydi va ularni kesib oladi (crop qiladi). Ikkinci qism — **mushuk zoti (breed) klassifikatori**, aniqlangan mushuk Siam mushugimi yoki yo'qmi, shuni aniqlaydi. Ushbu pipeline'ning har bir qismini yaxshilashga yillar sarflash mumkin. Ammo qaysi komponentaga e'tibor qaratish kerakligini qanday hal qilasiz?

Xatolarni qismlar bo'yicha tahlil qilish orqali siz algoritm tomonidan qilingan har bir xatoni pipeline'dagi ikki qismdan biriga (yoki ba'zan ikkalasiga) bog'lashga harakat qilishingiz mumkin.

Masalan, algoritm quyidagi rasmida **Siam mushugi yo'q** deb noto'g'ri natija beradi ($y = 0$), holbuki to'g'ri natija $y = 1$ bo'lishi kerak edi.

Keling, algoritmning ikki bosqichi qanday ishlaganini qo'lda tahlil qilamiz. Faraz qilaylik, **Siam mushugi aniqlovchi modul** quyidagicha mushukni aniqladi:

Bu shuni anglatadiki, mushuk zoti aniqlovchi klassifikatorga quyidagi rasm beriladi:

Mushuk zotini aniqlovchi klassifikator bu rasmni to'g'ri tarzda Siam mushuki yo'q deb baholaydi. Demak, mushuk zotini aniqlovchi modelda ayb yo'q: unga faqat toshlar uyumi berilgan va u juda mantiqiy tarzda $y=0$ (Siam mushuki yo'q) deb javob bergen. Darhaqiqat, agar inson ham shu kesilgan rasmga qaraganida, u ham xuddi shunday xulosa qilgan bo'lardi. Shunday ekan, bu xatoni aniq ravishda mushukni aniqlovchi detektor zimmasiga yuklash mumkin.

Agar, boshqa tomonidan, mushuk detektori quyidagi bounding boxni (chegaralovchi to'rtburchakni) chiqargan bo'lsa:

Shunda siz xulosa qilasizki, bu safar mushuk detektori o'z ishini to'g'ri bajargan, va ayb mushuk zotini aniqlovchi klassifikatorda.

Aytaylik, siz dev to'plamdagagi noto'g'ri klassifikatsiya qilingan 100 ta rasmni ko'rib chiqqansiz va ularning 90 tasida xato mushuk detektoriga tegishli, faqat 10 tasida mushuk zotini aniqlovchi klassifikatorga. Bu holatda, siz asosli ravishda e'tiboringizni mushuk detektorini yaxshilashga qaratishingiz kerakligini xulosa qilishingiz mumkin.

Bundan tashqari, sizda endi 90 ta misol bor — bu misollarda mushuk detektori noto'g'ri bounding box (chegaralovchi to'rtburchak) chiqargan. Ushbu 90 ta misoldan mushuk detektorida chuqurroq xatolik tahlilini o'tkazish uchun foydalanishingiz mumkin.

Biz hozirgacha xatolikni pipeline (quvur liniyasi) qismlaridan qaysi biriga yuklashni norasmiy tarzda tavsiflab keldik: siz har bir qismning chiqishini ko'rib chiqasiz va xatoni kim qilganini

aniqlashga harakat qilasiz. Ushbu norasmiy usul ko‘pincha yetarli bo‘lishi mumkin. Ammo keyingi bobda siz xatoni yanada aniqroq, rasmiyoq tarzda ajratish usuli bilan tanishasiz.

54 Xatoni bitta qismga yuklash

Keling, shu misoldan davom etamiz:

Aytaylik, mushukni aniqlovchi model quyidagi bounding box (chegaralovchi to‘rburchak) ni chiqardi:

Shunday qilib, mushuk zoti aniqlovchi modelga aynan shu qirqilgan rasm uzatiladi va u noto‘g‘ri ravishda $y=0$, ya’ni rasmida mushuk yo‘q degan natijani beradi.

Mushuk aniqlovchi (cat detector) o‘z ishini yaxshi bajarmagan. Shunga qaramay, yuqori malakali inson bu noto‘g‘ri qirqilgan rasmida ham Siam mushugini tanib olishi mumkin. Unda biz bu xatolikni mushuk aniqlovchiga, mushuk zoti aniqlovchiga, yoki ikkala modelga birdek yuklashimiz kerakmi? Bu holat noaniq.

Agar shunday noaniq holatlar soni kam bo‘lsa, siz istalgan qarorni qabul qilishingiz mumkin va natija sezilarli darajada o‘zgarmaydi. Ammo quyida keltirilgan yanada aniqroq test sizga xatolikni aynan bitta qismga yuklash imkonini beradi:

1. Mushuk aniqlovchi model chiqishini qo‘lda belgilangan (hand-labeled) bounding box bilan almashtiring.

2. Qo‘lda belgilangan “to‘g‘ri” bounding box (chegaralovchi ramka) yordamida olingan kesilgan rasmni mushuk zoti aniqlovchi modeldan o‘tkazing. Agar mushuk zoti aniqlovchi hali ham noto‘g‘ri natija chiqarsa, xatolikni aynan mushuk zoti aniqlovchiga yuklang. Aks holda, xatolik mushuk aniqlovchiga tegishli deb hisoblang. Boshqacha aytganda, mushuk zoti aniqlovchi modelga “mukammal” kirish (input) bergen holda tajriba o‘tkazing. Bu yerda ikkita holat bo‘lishi mumkin:

- **1-holat:** “Mukammal” bounding box berilganiga qaramay, mushuk zoti aniqlovchi noto‘g‘ri $y = 0$ natijani qaytaradi. Bu holatda xatolik aynan mushuk zoti aniqlovchiga tegishli.
- **2-holat:** “Mukammal” bounding box berilganda, mushuk zoti aniqlovchi to‘g‘ri $y = 1$ javobini beradi. Bu shuni ko‘rsatadiki, agar mushuk aniqlovchi yanada aniqroq bounding box bergenida edi, butun tizim to‘g‘ri ishlagan bo‘lardi. Demak, xatolikni mushuk aniqlovchiga yuklash kerak. Misclassified (noto‘g‘ri klassifikatsiya qilingan) dev set tasvirlari ustida shunday tahlilni o‘tkazish orqali siz har bir xatolikni aniq bitta komponentga bog‘lashingiz mumkin. Bu esa sizga pipeline’ning har bir qismiga to‘g‘ri e’tibor qaratish va ustuvor yo‘nalishni aniqlash imkonini beradi.

55 Xatolikni aniqlashning umumiy holati (General case of error attribution)

Quyidagilar xatolikni komponentlarga ajratishning umumiy bosqichlaridir. Faraz qilaylik, sizning pipeline (ma'lumotlar oqimi) tizimingizda ketma-ket uchta bosqich mavjud: **A**, **B** va **C**, bu yerda **A** chiqishi **B** ga, **B** chiqishi esa **C** ga uzatiladi.

Har bir xatolik uchun, tizim dev to‘plamida qilganida quyidagilarni bajaring:

1. A bosqichining chiqishini qo‘lda “mukammal” natijaga o‘zgartirishga harakat qiling (masalan, mushuk uchun “mukammal” bounding box), so‘ng qolgan pipeline qismlarini (B va C) shu natija bilan ishga tushiring. Agar algoritm endi to‘g‘ri natija bersa, demak xatolik A komponentida, ya‘ni faqat A yaxshilansa, tizimning umumiy natijasi to‘g‘ri bo‘lardi. Aks holda, 2-qadamga o‘ting.
2. B bosqichining chiqishini qo‘lda “mukammal” natijaga o‘zgartirishga harakat qiling. Agar algoritm endi to‘g‘ri natija bersa, demak xatolik B komponentidadir. Aks holda, 3-qadamga o‘ting.
3. Xatolik C komponentiga taalluqlidir.

Keling, bundan murakkabroq misolga qaraylik:

O‘zingizning o‘zini boshqaruvchi avtomobilingiz quyidagi pipeline’dan foydalanadi. Qaysi komponent(lar)ga e’tibor qaratishni aniqlash uchun qismma-qism xato tahlilini qanday o‘tkazasiz?

Siz uchta komponentni A, B, C ga quyidagicha taqsimlashingiz mumkin:

- A: Mashinalarni aniqlash
- B: Yo‘lovchilarni aniqlash
- C: Avtomobil uchun yo‘lni rejalahtirish

Yuqorida tasvirlangan tartibni bajarib, faraz qilaylik, siz o‘zingizning avtomobilingizni yopiq poyga yo‘lagida sinab ko‘rdingiz va avtomobil malakali haydovchiga qaraganda boshqaruvni keskinroq yo‘naltirgan holatni topdingiz. Avtomatik boshqaruv dunyosida bunday holat “ssenariy” deb ataladi. Bunday vaziyatda quyidagilarni qilasiz:

1. A komponenti (mashinalarni aniqlash) natijasini “mukammal” qilib qo‘lda o‘zgartirib ko‘ring (masalan, boshqa mashinalarning joylashuvini aniq ko‘rsatib bering). So‘ngra pipeline’ning qolgan qismlari B va C ni shu o‘zgartirilgan A natijasi bilan ishga tushiring. Agar algoritm endi avtomobil uchun ancha yaxshiroq yo‘l rejalahtirsса, demak xato A komponentidan. Shunda bu xatonи A ga bog‘lashingiz mumkin. Aks holda, 2-qadamga o‘ting.
2. B komponenti (yo‘lovchilarni aniqlash) natijasini “mukammal” qilib qo‘lda o‘zgartirib ko‘ring. Agar algoritm endi to‘g‘ri natija bersa, xato B komponentida. Aks holda, 3-qadamga o‘ting.
3. Xatonи C komponentiga (yo‘lni rejalahtirish) bog‘lang.

Mashina o‘rganish pipeline’ining komponentlari yo‘nalgan siklsiz graf (Directed Acyclic Graph, DAG) ko‘rinishida tartiblangan bo‘lishi kerak. Bu shuni anglatadiki, ularni ma’lum bir chapdan o‘ngga tartibda hisoblash mumkin va keyingi komponentlar faqat oldingi komponentlarning chiqishlariga bog‘liq bo‘ladi. Agar komponentlarning A->B->C tartibi DAG tartibiga mos kelsa, xato tahlili to‘g‘ri ishlaydi

- A: Yo‘lovchilarni aniqlash (oldin mashinalarni aniqlash edi)
 - B: Mashinalarni aniqlash (oldin yo‘lovchilarni aniqlash edi)
 - C: Yo‘lni rejalahtirish),
- bu tahlil natijalari hali ham haqiqiy bo‘lib, qaysi komponentga ko‘proq e’tibor

qaratish kerakligini aniqlash uchun foydali bo‘ladi. 56 Qismlarga bo‘lingan xato tahlili va inson darajasidagi natijalar bilan solishtirish

Mashina o‘rganish algoritmidagi xatolarni tahlil qilish – bu ma’lumotshunoslik yordamida ML tizimining xatolarini o‘rganib, keyingi qadamda nima qilish kerakligi haqida tushuncha olishdir. Eng oddiy ko‘rinishda, qismlarga bo‘lingan xato tahlili qaysi komponent(lar)ning samaradorligini yaxshilashga eng ko‘p e’tibor qaratish kerakligini ko‘rsatadi.

Faraz qilaylik, sizda veb-sayt orqali xarid qilayotgan mijozlar haqidagi ma’lumotlar to‘plami bor. Ma’lumotshunos ko‘plab turli yo‘llar bilan bu ma’lumotlarni tahlil qilishi mumkin. U veb-sayt narxlarini oshirish kerakmi, turli marketing kampaniyalari orqali olingan mijozlarning umr bo‘yi qiymatini qanday baholash kerak kabi ko‘plab turli xulosalarni chiqarishi mumkin.

Ma’lumot to‘plamini tahlil qilish uchun yagona “to‘g‘ri” yo‘l yo‘q, foydali ko‘plab tushunchalar hosil qilish mumkin. Shuningdek, xatolarni tahlil qilish uchun ham yagona “to‘g‘ri” usul yo‘q. Ushbu boblarda siz ML tizimingiz haqida foydali tushunchalar olish uchun keng tarqalgan ko‘plab dizayn usullarini o‘rgandingiz, lekin xatolarni tahlil qilishda boshqa usullarni ham sinab ko‘rishga erkin bo‘lishingiz kerak.

Endi o‘z-o‘zini boshqaruvchi avtomobil dasturiga qaytamiz, bu yerda mashina aniqlash algoritmi yaqin atrofdagi avtomobillar joylashuvi (va ehtimol tezligini) chiqaradi, piyoda aniqlash algoritmi esa yaqin atrofdagi piyodalar joylashuvini ko‘rsatadi, va bu ikkala chiqish oxir-oqibat avtomobil uchun yo‘l rejalashtirishda ishlatiladi.

Ushbu pipeline’ni xatolarini aniqlash uchun, avvalgi bobda ko‘rgan qat’iy protsedurani amal qilish o‘rniga, siz quyidagilarni biroz noformal tarzda so‘rashingiz mumkin:

1. Avtomobillarni aniqlash komponenti inson darajasidagi aniqlikdan qancha orqada?
2. Piyodalarni aniqlash komponenti inson darajasidagi aniqlikdan qancha orqada?
3. Umumiyligi tizimning ishlashi inson darajasidagi natijadan qanchalik orqada? Bu yerda inson darajasidagi natija shundan iboratki, inson faqat oldingi ikkita pipeline komponentining chiqishlariga asoslanib mashina uchun yo‘l rejasini tuzadi (kameradan olingan tasvirlarga kira olmaydi). Boshqacha aytganda, “Yo‘l rejalashtirish” komponentining ishlashi insonni bilan solishtirilganda qanday?

Agar siz komponentlardan biri inson darajasidan ancha orqada ekanini aniqlasangiz, endi shu komponentning ishlashini yaxshilashga e’tibor qaratish yaxshi qaror bo‘ladi.

Ko‘plab xato tahlili jarayonlari insonlar yaxshi bajara oladigan va shuning uchun inson darajasidagi natijalarga solishtirish mumkin bo‘lgan vazifalarni avtomatlashtirishga uringanida eng yaxshi natijani beradi. Oldingi misollarimizning aksariyati shu shartga asoslangan.

Agar siz yaratgan ML tizimingizda yakuniy natija yoki oraliq komponentlardan ba’zilari inson ham yaxshi bajara olmaydigan ishlarni bajarayotgan bo‘lsa, unda ba’zi protseduralar ishlamasligi mumkin.

Bu esa insonlar hal qila oladigan muammolar ustida ishlashning yana bir afzalligi — sizda kuchliroq xato tahlili vositalari bo‘ladi va shuning uchun jamoangiz ishlarni samaraliroq ustuvorlashtira olasiz.

57 ML pipeline dagi kamchiliklarni aniqlash

Agar ML pipeline'ingizdagi har bir alohida komponent inson darajasida yoki unga yaqin darajada ishlayotgan bo'lsa, lekin umumiy pipeline inson darajasidan ancha past natija ko'rsatayotgan bo'lsa, odatda bu pipeline'da muammo borligini va uni qayta loyihalash kerakligini anglatadi.

Xato tahlili shuningdek, pipeline'ni qayta loyihalash zarurligini tushunishga yordam beradi.

Oldingi bobda biz uchta komponentning har biri inson darajasida ishlayaptimi, degan savolni ko'rib chiqdik. Faraz qilaylik, uchala savolga ham "ha" javobi berildi. Ya'ni:

1. "Detect cars" (mashinalarni aniqlash) komponenti kameradan olingan tasvirlar asosida mashinalarni aniqlashda taxminan inson darajasida ishlaydi.
2. "Detect pedestrians" (piyodalarni aniqlash) komponenti kameradan olingan tasvirlar asosida piyodalarni aniqlashda taxminan inson darajasida ishlaydi.
3. "Plan path" (yo'l rejalashtirish) komponenti esa, oldingi ikkita komponentdan olingan chiqishlarga asoslanib, yo'lni rejalashtirishda inson darajasiga yaqin ishlaydi (ya'ni, bu komponent kamera tasvirlariga bevosita kira olmaydi).

Misol uchun, siz inson haydovchisi uchun yo'l belgilarining joylashuvi ham muhim ekanligini tushundingiz. Bu esa pipeline'ni quyidagicha qayta loyihalash kerakligini ko'rsatadi:

1. [Yangi pipeline tarkibini kriting — bu yerda pipeline'ni qayta loyihalash haqida gap ketmoqda]

Ammo, sizning butunlay avtonom mashinangiz inson darajasidan sezilarli past natija ko'rsatmoqda. Ya'ni, kamera tasvirlariga to'liq kira oladigan inson haydovchi mashina uchun ancha yaxshi yo'llarni rejalashtira olmoqda. Bu qanday xulosaga olib keladi?

Eng mantiqiy xulosa shuki, ML pipeline'ida muammo bor. Bu holatda, "Plan path" komponenti mavjud kirishlar asosida eng yaxshi natijani bermoqda, ammo bu kirishlar yetarli ma'lumotga ega emas. Siz o'zingizga savol berishingiz kerak: yo'l rejalashtirish uchun oldingi ikki komponent chiqishlaridan tashqari qanday qo'shimcha ma'lumotlar kerak? Boshqacha qilib aytganda, malakali inson haydovchiga yo'l rejalashtirish uchun yana qanday ma'lumotlar kerak.

Agar sizning pipeline'ingizning har bir alohida komponenti inson darajasida ishlasa ham (ya'ni har bir komponentga berilgan kirish ma'lumotlari insonga berilgandek bo'lsa), lekin umuman olganda pipeline inson darajasiga yetolmasa, demak pipeline'da muammo bor va uni qayta loyihalash kerak.

Xulosa

58 Superqahramon jamoasini yaratish

Tabriklaymiz, ushbu kitobni tamomlaganingiz uchun!

2-bobda, bu kitob sizga jamoangizning superqahramoni bo‘lishingizda qanday yordam berishi haqida gapirdik.

Superqahramon bo‘lishdan ham zo‘rrog‘i — bu superqahramonlar jamoasining bir qismi bo‘lishdir. Umid qilamanki, siz ushbu kitobdan do‘srlaringiz va jamoa a‘zolaringizga ham nusxa berasiz va boshqa superqahramonlar paydo bo‘lishiga hissa qo‘shasiz!

Ushbu kitob mashinani o‘rganish (machine learning) tizimlarini yaratish va takomillashtirish jarayonida duch keladigan muhim muammolarni tushunishga va ularni samarali hal qilishga qaratilgan. Kitobda murakkab tizimlarni modul va bosqichlarga ajratib, har bir qismdagi xatolarni aniqlash va ularni qanday tuzatish bo‘yicha strategiyalar bayon etilgan.

Asosiy e’tibor quyidagilarga qaratilgan:

- **Optimizatsiya va skoring funksiyalari:** Algoritm natijasini yaxshilash uchun qayerda muammo borligini aniqlash (qidiruv algoritmda yoki baholash funksiyasida).
- **End-to-end yondashuv:** Butun tizimni bitta chuqur o‘rganish modeliga topshirish va uning afzallik hamda kamchiliklari.
- **Pipeline tizimlari:** Murakkab vazifalarni sodda, oson o‘rganiladigan bosqichlarga bo‘lish orqali samaradorlikni oshirish.
- **Xato tahlili:** Tizimdagi xatolarni har bir modulga taqsimlab, qaysi qismga ko‘proq e’tibor qaratish kerakligini aniqlash usullari.
- **Inson darajasidagi natijalar bilan solishtirish:** Modullar va umumiy tizim qanchalik inson darajasida ishlashini baholash, tizim dizaynini yaxshilashga yordam beradi.

Kitob sizga mashinani o‘rganish loyihalarida tizimli fikrlash va tahlil qilish ko‘nikmalarini rivojlantirishga, samarali modellar yaratish uchun to‘g‘ri strategiyalarni tanlashga yordam beradi.