Nulta laboratorijska vježba iz Dubokog učenja

Podsjetnik na logističku regresiju, gradijentni spust, Python i Numpy

U ovoj vježbi ćemo ponoviti osnove strojnog učenja, podsjetiti se na optimizaciju gradijentnim spustom te upoznati se s mogućnostima Pythona, Numpyja i Matplotliba. Razvit ćemo pythonske implementacije binarne i višerazredne logističke regresije utemeljene na Numpyjevim primitivima. Ispitat ćemo svojstva naše implementacije analizom numeričkih pokazatelja uspješnosti klasifikacije te iscrtavanjem funkcije odluke primjenom Matplotliba. Cilj vježbe je ponoviti osnovna znanja s prethodnih kolegija te razviti intuiciju koja će nam biti dragocjena pri oblikovanju i debuggiranju dubokih modela strojnog učenja.

U okviru vježbe razvit ćemo tri pythonska modula: `data`, `binlogreg` i `logreg`. Modul `data` će sadržavati operacije vezane uz stvaranje i iscrtavanje skupa slučajnih 2D podataka te operacije vezane uz evaluaciju klasifikacijske performanse. Taj modul ćemo koristiti i u prvoj vježbi gdje ćemo na istim podatcima usporediti duboko učenje sa SVM-om kao prvim izborom u mnogim klasifikacijskim problemima. Preostala dva modula će sadržavati funkcije i testove za binarnu i višerazrednu logističku regresiju.

0a. O parcijalnim derivacijama vektorskih funkcija

Deriviranje vektorskih i skalarnih funkcija više varijabli je ključni koncepti za shvaćanje gradijentne optimizacije parametara modela strojnog učenja i zato ga moramo dobro poznavati. Po definiciji, parcijalna derivacija funkcije više varijabli jest derivacija s obzirom na jednu od tih varijabli pod pretpostavkom da su sve ostale varijable konstantne. Parcijalne derivacije vektorskih funkcija možemo računati odvojeno za svaku komponentu funkcije.

Ilustrirajmo ove pojmove na primjeru sljedeće vektorske funkcije vektorske varijable:

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} x_1^2 + x_2 x_3 \\ x_1 + x_2 + x_3 \end{bmatrix}$$

Parcijalna derivacija prve komponente funkcije $\mathbf{f}$ po drugoj komponenti argumenta $\mathbf{x}$ odgovara izrazu: $\partial f_1/\partial x_2 = x_3$. Parcijalna derivacija druge komponente funkcije po prvoj komponenti argumenta odgovara izrazu: $\partial f_2/\partial x_1 = 1$.

Jakobijeva matrica

Skup svih parcijalnih derivacija vektorske funkcije vektorske varijable možemo izraziti Jakobijevom matricom (ili, kraće, Jakobijanom). Retci Jakobijana odgovaraju komponentama vektorske funkcije, dok stupci odgovaraju komponentama nezavisne varijable. Tako npr. Jakobijan skalarne funkcije D-dimenzionalne vektorske varijable ima jedan redak i D stupaca, dok Jakobijan funkcije sinus ima jedan redak i jedan stupac. Po istoj logici, Jakobijan funkcije $\mathbf{f}$ imao bi dva retka i tri stupca:

$$\mathbf{f}'(\mathbf{x}) = \frac{d\mathbf{f}}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 & x_3 & x_2 \\ 1 & 1 & 1 \end{bmatrix}$$

Često će nam biti interesantne matrice parcijalnih derivacija obzirom na samo neke od nezavisnih varijabli promatrane funkcije. Primjerice, ako kažemo da vektor $\mathbf{q}$ sadrži prve dvije komponente vektora $\mathbf{x}$ (tj. da vrijedi $\mathbf{q} = [x_1 x_2]^\top$), onda možemo pisati:

$$\mathbf{f}'(\mathbf{q}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 & x_3 \\ 1 & 1 \end{bmatrix}$$

Kompozicije funkcija i pravilo ulančavanja

Pretpostavimo sada da naša funkcija $\mathbf{f}$ nema izravan kontakt s nezavisnom varijablom, nego da na nju djeluje posredno, preko funkcije $\mathbf{g}$. Rezultantna funkcija $\mathbf{F}$ odgovara kompoziciji funkcija $\mathbf{f}$ i $\mathbf{g}$:

$$\mathbf{F}(\mathbf{x}) = (\mathbf{f} \circ \mathbf{g})(\mathbf{x}) = \mathbf{f}(\mathbf{g}(\mathbf{x}))$$

Neka je funkcija $\mathbf{g}$ zadana kako slijedi:

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} \sin(x_1) \\ x_2^3 + x_1 x_2 \\ x_3 \end{bmatrix}$$

Pretpostavimo da želimo odrediti derivaciju kompozicije funkcija $\mathbf{F}$. U pomoć nam priskače pravilo ulančavanja vektorskih funkcija koje u općenitom obliku glasi:

$$\mathbf{F}'(\mathbf{x}) = \mathbf{f}'(\mathbf{g}(\mathbf{x})) \cdot \mathbf{g}'(\mathbf{x})$$

Kad pravilo ulančavanja primijenimo na naš konkretni primjer, Jakobijevu matricu funkcije $\mathbf{F}$ dobivamo sljedećim izvodom:

$$\mathbf{F}'(\mathbf{x}) = \mathbf{f}'(\mathbf{g}(\mathbf{x})) \cdot \mathbf{g}'(\mathbf{x}) = \begin{bmatrix} 2g_1(\mathbf{x}) & g_3(\mathbf{x}) & g_2(\mathbf{x}) \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(x_1) & 0 & 0 \\ x_2 & x_1 + 3x_2^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{F}'(\mathbf{x}) = \begin{bmatrix} 2\sin(x_1) & x_3 & x_2^3 + x_1 x_2 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos(x_1) & 0 & 0 \\ x_2 & x_1 + 3x_2^2 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{F}'(\mathbf{x}) = \begin{bmatrix} 2\sin(x_1)\cos(x_1) + x_2 x_3 & x_1 x_3 + 3x_2^2 x_3 & x_2^3 + x_1 x_2 \\ \cos(x_1) + x_2 & x_1 + 3x_2^2 & 1 \end{bmatrix}$$

0b. O optimizaciji gradijentnim spustom

U strojnom učenju posebno često analiziramo svojstva *skalarnih* funkcija više varijabli, a tipičan primjer je optimiranje funkcije gubitka o kojoj ćemo više pričati u odjeljcima 0c i 0d. Ako je funkcija više varijabli skalarna (tj. ima samo jednu komponentu), tada *gradijent* te funkcije sadrži parcijalne derivacije po svim varijablama. Ako gradijent promatramo kao "obični" stupčani vektor (konvencije se razlikuju, ali to je najčešće slučaj), gradijent će odgovarati transponiranoj Jakobijevoj matrici:

$$\nabla f(\mathbf{x}) = \frac{df(\mathbf{x})}{d\mathbf{x}}^\top.$$

Ako funkciju f aproksimiramo Taylorovim razvojem prvog reda, dobivamo aproksimaciju:

$$f(\mathbf{x} + \Delta\mathbf{x}) \approx f(\mathbf{x}) + \frac{df(\mathbf{x})}{d\mathbf{x}}\Delta\mathbf{x} = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \Delta\mathbf{x}.$$

Pogledajmo što se zbiva kad u tu aproksimaciju uvrstimo pomak u smjeru negativnog gradijenta:

$$\Delta\mathbf{x} = -\epsilon \cdot \mathbf{g},\ \mathbf{g} = \nabla f(\mathbf{x}).$$

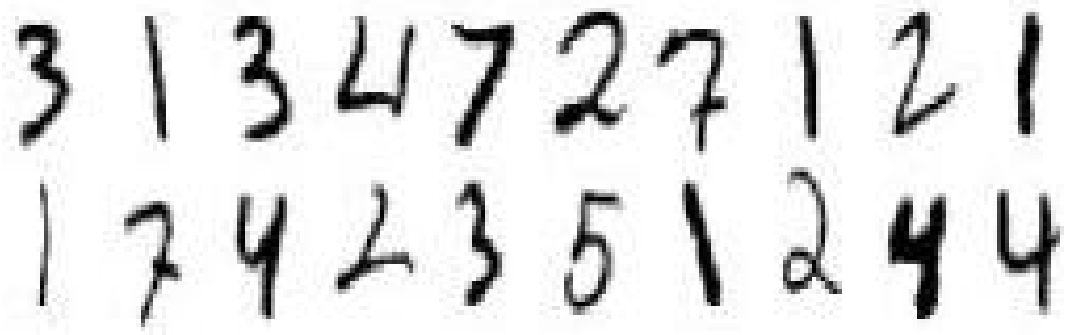
Ako Taylorova aproksimacija vrijedi, vrijednost funkcije f bi trebala opadati:

$$f(\mathbf{x} - \epsilon \cdot \mathbf{g}) \approx f(\mathbf{x}) - \epsilon \cdot \mathbf{g}^\top \mathbf{g} < f(\mathbf{x})\ .$$

Vidimo da iterativno pomicanje u smjeru negativnog gradijenta može dovesti do lokalnog minimuma funkcije f. To je najjednostavnija metoda gradijentne optimizacije, a poznata je pod nazivom *gradijentni spust*. U praksi, aproksimacija će biti tim točnija što je hiper-parametar ϵ manji. Međutim, nije dobro da ϵ bude premali kako učenje ne bi previše trajalo. Zbog toga su smišljene brojne poboljšane metode, a neke od njih ćemo upoznati na predavanjima.

0c. Osnove strojnog učenja

Strojno učenje proučava izradu programskih komponenata koje elemente svoje funkcionalnosti mogu naučiti na podatcima. Pretpostavimo da na ulazu našeg potprograma dobivamo D-dimenzionalni *podatak* $\mathbf{x}$ koji može biti točka ravnine (D=2), siva slika dimenzija 28x28 (D=784) ili nešto treće. Pretpostavimo dalje da naš potprogram ulazni podatak treba *svrstati* u jedan od C razreda (ili klasa). Drugim riječima, naš potprogram na izlazu treba proizvesti *predikciju* y koja odgovara indeksu razreda ulaznog podatka (vrijedi $0 < y < C-1$). U slučaju da je naš potprogram element sustava za raspoznavanje brojeva na skeniranim uplatnicama, razredi bi bili znamenke od 0 do 9. Razredi podataka na sljedećoj slici bili bi: 3, 1, 3, 4, 7, 2, itd:



Kad bi naš zadatak bio raspoznati prometnu signalizaciju u sustavu za sigurnosnu inspekciju prometnica, razredi bi bili standardne oznake prometnih znakova (npr. A03, A04, A05, A33, B31 itd.) kao što to ilustrira sljedeća slika:

Postavlja se pitanje kako organizirati računalni program koji treba podržati učenje na primjerima označenih podataka. Principijelan prvi korak u tom smjeru jest izraziti dio programa koji treba učiti u obliku matematičke funkcije sa slobodnim *parametrima*. Tu funkciju nazivat ćemo *modelom*. Možda najjednostavniji model za naš konkretni primjer je:

$$y = argmax(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}).$$

Slobodni parametri tog modela su $\mathbf{W}$ i $\mathbf{b}$. Pri tome $\mathbf{W}$ predstavlja matricu $\mathrm{C} \times \mathrm{D}$, $\mathbf{b}$ je vektor $\mathrm{C} \times 1$, dok $\cdot$ označava matrično množenje. Intuitivno, takav model ulazni podatak $\mathbf{x}$ uspoređuje sa svakim od C redaka matrice $\mathbf{W}$ (skalarni produkt se pod nekim uvjetima može promatrati kao usporedba), te podatak svrstavamo u razred definiran indeksom retka koji je "najsličniji" podatku. U praksi ćemo koristiti diferencijalne varijante tog modela koje će omogućiti učenje gradijentnim spustom o kojem smo pričali u odjeljku 0b.

Da bismo model naučili, trebaju nam još dva koncepta: *gubitak* i *optimizacijski postupak*. Gubitak je funkcija koja kaže koliko se dobro predikcije modela poklapaju sa podatcima za učenje. Optimizacijski postupak "štima" parametre tako da gubitak na skupu za učenje bude što manji.

Formalno, algoritam strojnog učenja definiran je modelom, gubitkom i optimizacijskim postupkom. Npr. kod logističke regresije koju ćemo detaljno upoznati u sljedećem odjeljku (0d) imamo:

- model: $\mathrm{softmax}(\mathbf{W} \cdot \mathbf{x} + \mathbf{b})$,
- gubitak: negativna log-izglednost i
- optimizacijski postupak: gradijentni spust.

Izlaz logističke regresije je C-dimenzionalni vektor koji predstavlja vjerojatnosti $P(Y = c_i|x)$ koje su pridružene događajima da podatak $\mathbf{x}$ pripada razredu c_i. Modele s takvim izlazom svrstavamo u diskriminativne probabilističke modele.

0d. Uvod u višerazrednu logističku regresiju

Višerazredna logistička regresija [1] podrazumijeva probabilistički diskriminativni klasifikacijski model pripadnosti vektorskog podatka $\mathbf{x}$ razredima $c_j, j = 0, 1, \ldots, C-1$. Radi se o generaliziranom linearnom modelu koji afinu transformaciju podataka spljošćuje (engl. squash) na vjerojatnosni interval [0,1]. Ishod klasifikacije formalno prikazujemo slučajnom varijablom Y za koju vrijedi $\sum_j P(Y = c_j|\mathbf{x}) = 1 \; \forall \mathbf{x}$.

U binarnom slučaju podatak može pripadati jednom od samo dvaju razreda ($C = 2$). Parametri modela tada su vektor $\mathbf{w}$ te pomak b, a rezultat modela su aposteriorne vjerojatnosti razreda c_0 i c_1:

$$P(Y = c_1|\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b), \text{ gdje je } \sigma(s) = e^s/(1 + e^s)$$
$$P(Y = c_0|\mathbf{x}) = 1 - P(c_1|\mathbf{x})$$

Pri tome podatci $\mathbf{x}$ imaju dimenzije Dx1, parametar $\mathbf{w}$ ima dimenzije Dx1, dok je parametar b skalar.

Kad podatke klasificiramo u C razreda, model možemo sažeto formulirati sljedećim skupom jednadžbi (funkciju softmax ćemo objasniti u nastavku):

$$P(Y = c_j|\mathbf{x}) = \mathrm{softmax}(j, \mathbf{W} \cdot \mathbf{x} + \mathbf{b}), j = 0, 1, \ldots, C-1.$$

Matrice $\mathbf{W}$ i $\mathbf{b}$ sadrže parametre postupka, a njihove dimenzije su CxD odnosno Cx1. Pri tome D predstavlja dimenzionalnost podataka, dok je C broj razreda. Vektor $\mathbf{s} = \mathbf{W} \cdot \mathbf{x} + \mathbf{b}$ ima dimenzije Cx1 te sadrži tzv. klasifikacijske mjere razreda (možemo koristiti i prijevod klasifikacijski rezultat, engl. classification scores). Klasifikacijske mjere pokazuju koliko je odgovarajući razred vjerojatniji (odnosno manje vjerojatan) u odnosu na druge razrede. Konačno, aposteriorne vjerojatnosti razreda dobivamo funkcijom softmax čija j-ta komponenta odgovara izrazu: $\mathrm{softmax}(j, \mathbf{s}) = e^{s_j} / \sum_k e^{s_k}$.

Ako detaljnije razmotrimo definiciju funkcije softmax, primijetit ćemo da su klasifikacijske mjere redundantne: kad svakoj od njih dodamo istu konstantu funkcija softmax na izlazu daje istu razdiobu. Iz toga slijedi da jednu od C klasifikacijskih mjera bez smanjenja općenitosti možemo fiksirati postavljanjem odgovarajućeg retka parametara na nulu ($\mathbf{W}_{j,:} = b_j = 0$). Tako normalizirane klasifikacijske mjere mogu se interpretirati kao logaritmi omjera šansi promatranog razreda u odnosu na fiksirani razred. U praksi se pokazuje da postupak jednako dobro konvergira i ako optimiramo sve retke $\mathbf{W}$ i $\mathbf{b}$. Stoga u implementaciji često optimiramo sve parametre jer se tako dobiva nešto jednostavniji programski kod. Međutim, redundantnost ulaza softmaksa u pravilu koristimo kako bismo izbjegli da eksponenciranje velikih klasifikacijskih mjera rezultira numeričkim preljevom. Primjer takve implementacije donosimo u nastavku:

```
# stabilni softmax
def stable_softmax(x):
```

```
    exp_x_shifted = np.exp(x - np.max(x))
    probs = exp_x_shifted / np.sum(exp_x_shifted)
    return probs
```

Obratite pažnju da prikazani primjer funkcionira samo ako mu se kao ulazni argument pošalje tenzor prvog reda (vektor). U vježbi ćete takvu proceduru trebati primijeniti na matricu $N \times D$ u kojoj će svaki redak sadržavati klasifikacijske mjere odgovarajućeg podatka. Pokušajte to ostvariti bez eksplicitne petlje.

Parametre logističke regresije $\mathbf{W}$ i $\mathbf{b}$ određujemo optimiranjem funkcije gubitka $\mathcal{L}(\mathbf{W}, \mathbf{b})$ na skupu za učenje $\mathcal{D}$. Skup za učenje tipično se sastoji od parova podataka i odgovarajućih oznaka razreda $\mathcal{D} = \{(\mathbf{x_i}, y_i), i = 1, 2, \ldots, N\}$. Ovo je najjednostavnije pojmiti na problemu klasifikacije 2D podataka u C različitih razreda. U tom slučaju $\mathbf{x_i}$ su elementi ravnine, dok su y_i elementi cjelobrojnog intervala između 0 i C-1. Gubitak logističke regresije odgovara negativnoj log-izglednosti parametara:

$$\mathcal{L}_{\text{NLL}}(\mathbf{W}, \mathbf{b}|\mathcal{D}) = -\sum_i \log P(Y = y_i|\mathbf{x}_i) \ .$$

Obratite pažnju na to da $P(Y = y_i|\mathbf{x}_i)$ označava vjerojatnost kojom model podatak $\mathbf{x}_i$ klasificira u točan razred y_i. Ovako izražen gubitak favorizira parametre $\mathbf{W}$ i $\mathbf{b}$ koji točnim oznakama podataka za učenje pridružuju što veće vjerojatnosti. U praksi nas ništa ne sprječava da umjesto negativne log-izglednosti parametara optimiramo neki drugi gubitak. Međutim, za slučaj klasifikacije, tj. kada su predikcije kategoričke, negativna log-izglednost vrlo dobro funkcionira u praksi te je najdosljedniji izbor kad učimo na čvrstim oznakama (za meke oznake treba nam unakrsna entropija). Kada negativnu log-izglednost podijelimo brojem podataka, dobivamo izraz koji odgovara prosječnoj unakrsnoj entropiji između točne i prediktirane distribucije po svim podatcima (provjerite to za vježbu!):

$$\mathcal{L}_{\text{CE}}(\mathbf{W}, \mathbf{b}|\mathcal{D}) = -\frac{1}{N}\sum_i \log P(Y = y_i|\mathbf{x}_i) \ .$$

Ovakva formulacija gubitka pokazuje prosječnu točnost predikcije modela u pojedinim podatcima te tako olakšava praćenje napredovanja učenja i dijagnosticiranje problema s učenjem. Dodatno, apsolutni iznos gradijenata parametara ovakvog gubitka ne ovisi o broju podataka što značajno olakšava ugađanje hiper-parametara postupka. Zbog toga ćemo u praksi uvijek koristiti usrednjeni ukupni gubitak (odnosno unakrsnu entropiju, $\mathcal{L}_{\text{CE}}$) umjesto pozbrojenog ukupnog gubitka (odnosno negativne log-izglednosti, $\mathcal{L}_{\text{NLL}}$).

Nažalost, optimiranje parametara logističke regresije ne možemo provesti u zatvorenom obliku, nego moramo posegnuti za nekim iterativnim postupkom. Na sreću, funkcija gubitka $\mathcal{L}_{\text{CE}}$ je konveksna [2]. To znači da gradijentne metode poput gradijentnog spusta sigurno neće zapeti u nekom lokalnom optimumu (iako mogu podbaciti zbog drugih razloga).

Funkciju gubitka $\mathcal{L}_{\text{CE}}$ možemo optimirati raznim gradijentnim postupcima. Takvi postupci u svakoj iteraciji učenja pomiču parametre u smjeru negativnog gradijenta gubitka. Period tijekom kojeg algoritam dobije na uvid sve podatke za učenje nazivamo epohom. Ovisno o težini problema, za konvergenciju su potrebne desetine, stotine, ili tisuće epoha.

Glavni izazov kod gradijentnih pristupa optimizacije predstavlja određivanje gradijenta gubitka po svim parametrima postupka. S obzirom na to da gubitak i-tog podatka u logističkoj regresiji možemo promatrati kao kompoziciju logaritmiranog softmaksa i afine transformacije podataka, gradijente određujemo primjenom pravila ulančavanja.

Literatura:

1. Jan Šnajder, Bojana Dalbelo Bašić: Strojno učenje. FER, Zagreb.
2. http://qwone.com/~jason/writing/convexLR.pdf

0e. Gradijenti binarne logističke regresije

Ovdje ćemo skicirati optimirani postupak računanja parcijalnih derivacija funkcije gubitka binarne logističke regresije. Za početak, dogovorimo notaciju. Označimo gubitak

i-tog podatka s $\mathcal{L}_i(\mathbf{w}, \mathbf{b}|x_i, y_i) = -\log P(Y = y_i|\mathbf{x}_i)$. Definirajmo parcijalnu derivaciju $\partial\mathcal{L}_i/\partial\mathbf{w}$ i predstavimo je retčanim vektorom u kojem se nalaze gradijenti gubitka $\mathcal{L}_i$ po elementima parametra $\mathbf{w}$. Slično, neka parcijalna derivacija $\partial\mathcal{L}_i/\partial b$ bude retčani vektor gradijenata gubitka $\mathcal{L}_i$ po parametru b.

Kako bismo omogućili kompaktniji zapis aposteriorne vjerojatnost i-tog razreda, možemo uvesti *proširenu* sigmoidalnu funkciju koja uzima u obzir točan razred podatka čiju klasifikacijsku mjeru transformiramo u vjerojatnost:

$$\sigma_P(y, s) = \begin{cases} \sigma(s), & y = c_1 \\ 1 - \sigma(s), & y = c_0 \end{cases} = \begin{cases} \frac{e^s}{1+e^s}, & y = c_1 \\ \frac{1}{1+e^s}, & y = c_0 \end{cases}$$

Sada gubitak $\mathcal{L}_i$ odgovara kompoziciji logaritmirane proširene sigmoide i afine redukcije i-tog podatka (podsjetimo se, s_i označava skalarnu klasifikacijsku mjeru podatka $\mathbf{x}_i$):

$$\mathcal{L}_i(\mathbf{w}, \mathbf{b}|\mathbf{x}_i) = -\log \sigma_P(y_i, s_i)$$

$$s_i = \mathbf{w}^\top \mathbf{x}_i + b$$

Gradijente parametara stoga možemo izraziti prema pravilu ulančavanja kao umnožak matrica parcijalnih derivacija (tzv. Jacobijevih matrica) funkcija $\mathcal{L}_i$ i s_i:

$$\partial\mathcal{L}_i/\partial\mathbf{w} = \partial\mathcal{L}_i/\partial s_i \cdot \partial s_i/\partial\mathbf{w}$$

$$\partial\mathcal{L}_i/\partial b = \partial\mathcal{L}_i/\partial s_i \cdot \partial s_i/\partial b$$

Parcijalna derivacija gubitka $\mathcal{L}$ po linearnom klasifikacijskom rezultatu s u podatku $\mathbf{x}_i$ jest skalar $\partial\mathcal{L}_i/\partial s_i$ kojeg promatramo kao Jakobijan dimenzija 1x1. Parcijalne derivacije klasifikacijske mjere podatka $\mathbf{x}_i$ po elementima parametara $\mathbf{w}$ odnosno b, su matrica $\partial s_i/\partial\mathbf{w}$ dimenzija 1xD odnosno skalar $\partial s_i/\partial b$ kojeg promatramo kao matricu 1x1. Izrazimo prvo derivaciju gubitka po klasifikacijskoj mjeri tako da ovisnost o točnom razredu i-tog podatka izrazimo Iversonovim uglatim zagradama. Ako se dogovorimo da vrijednost izraza $[\![q]\!]$ iznosi 1 ako je q istinit a nula inače, traženu parcijalnu derivaciju možemo predstaviti sljedećim izrazom (provjerite to za vježbu!):

$$\partial\mathcal{L}_i/\partial s_i = P(c_1|\mathbf{x}_i) - [\![y_i = c_1]\!].$$

Parcijalna derivacija klasifikacijske mjere po parametrima $\mathbf{w}$ i b za podatak $\mathbf{x}_i$ sada je:

$$\partial s_i/\partial\mathbf{w} = \mathbf{x}_i^\top$$

$$\partial s_i/\partial b = 1.$$

Kad uvedemo pokratu $\mathbf{g}_s = [\partial\mathcal{L}_i/\partial s_i]_{i=1}^N$ te uzmemo u obzir da ukupni gubitak zbraja doprinose po svim podatcima, dobivamo konačne izraze:

$$\partial\mathcal{L}/\partial\mathbf{w} = 1/N \cdot \textstyle\sum_i \partial\mathcal{L}_i/\partial\mathbf{w} = 1/N \cdot \sum_i g_{si} \cdot \mathbf{x}_i^\top \ ,$$

$$\partial\mathcal{L}/\partial b = 1/N \cdot \textstyle\sum_i \partial\mathcal{L}_i/\partial b = 1/N \cdot \sum_i g_{si} \ .$$

Najbolja računska performansa danas se postiže izražavanjem petlji optimiranim operacijama nad matricama i vektorima. Zbog toga je jasno da ćemo u implementaciji računanje gradijenta pomaka izraziti pozivom odgovarajuće funkcije numeričke biblioteke. Manje je međutim jasno kako izbjeći pisanje petlje za računanje gradijenata težina, pa ćemo tom važnom detalju pokloniti više pažnje.

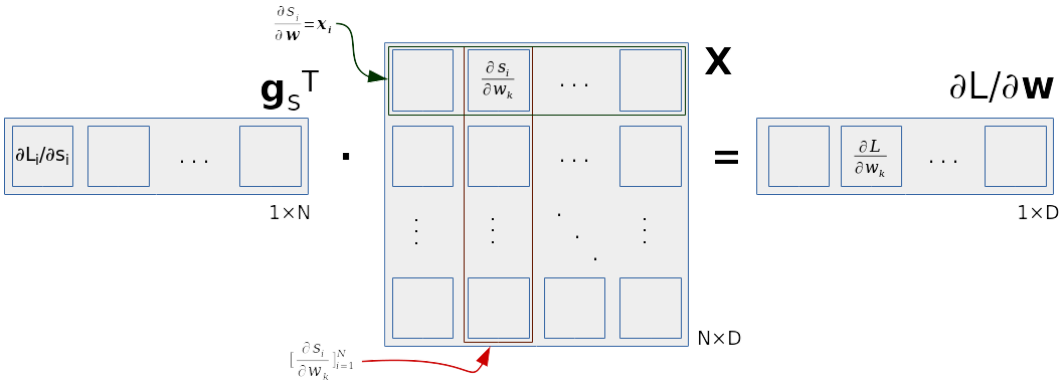
Naime, pažljivim promatranjem možemo uočiti da računanje parcijalne derivacije $\partial\mathcal{L}/\partial\mathbf{w}$ možemo vektorizirati tako da zbroj po podatcima predstavimo matričnim umnoškom. Vektore podataka ćemo organizirati u matricu $\mathbf{X}$ dimenzija NxD. Retci matrice $\mathbf{X}$ odgovaraju podatcima $\mathbf{x}_i$. Parcijalne derivacije gubitaka po klasifikacijskoj mjeri smjestit ćemo u stupčani vektor $\mathbf{g}_s$ dimenzije N. Kao što smo naveli ranije, vrijedi: $\mathbf{g}_s = [\partial\mathcal{L}_i/\partial s_i]_{i=1}^N$. Lako se vidi da gradijent gubitka po k-toj težini odgovara skalarnom produktu vektora $\mathbf{g}_s$ i k-tog stupca matrice $\mathbf{X}$:

$$\partial\mathcal{L}/\partial w_k = 1/N \cdot \textstyle\sum_i g_{si} \cdot x_{ik} = 1/N \cdot \mathbf{g}_s^\top \mathbf{X}_{:k}.$$

Sada je jasno da parcijalne derivacije ukupnog gubitka po svim težinama možemo dobiti produktom vektora $\mathbf{g_s}^\top$ i podatkovne matrice $\mathbf{X}$:

$$\partial\mathcal{L}/\partial\mathbf{w} = 1/N \cdot \mathbf{g_s}^\top \cdot \mathbf{X} \ .$$

Taj umnožak prikazan je na sljedećoj slici. Na slici se vidi da retci podatkovne matrice sadrže pojedinačne podatke te odgovaraju parcijalnoj derivaciji klasifikacijske mjere po vektoru težina za taj podatak. Stupci podatkovne matrice odgovaraju parcijalnim derivacijama po pojedinačnim komponentama vektora težina. Parcijalnu derivaciju ukupnog gubitka po k-toj težini dobivamo skalarnim produktom vektora $\mathbf{g_s}$ i k-tog stupca matrice $\mathbf{X}$.



Napominjemo da postoji dobar razlog da matrica podataka ima strukturu NxD. U tom su slučaju komponente pojedinog podatka pohranjene na uzastopnim memorijskim lokacijama pa miješanje podataka i operacije nad grupama (engl. batch) uzrokuju manje promašaja priručne memorije. U programskoj implementaciji htjet ćemo dobiti gradijente `grad_w` koji po dimenzijama odgovaraju originalnim parametrima pa ćemo gornji izraz naprosto transponirati. U tom slučaju, poboljšane vrijednosti parametara dobivamo oduzimanjem gradijenata `grad_w` moduliranih hiper-parametrom `param_delta`.

0f. Gradijenti višerazredne logističke regresije

Jednadžbe gradijenata parametara višerazredne logističke regresije vrlo su slične odgovarajućim gradijentima binarne logističke regresije. To nas ne treba čuditi jer se lako uviđa da je softmax poopćenje sigmoidalne funkcije. Međutim, izražavanje tih gradijenata otežava struktura parametra $\mathbf{W}$. Da izbjegnemo poteškoće, odvojeno ćemo promatrati gradijente svakog od C redaka matrice $\mathbf{W}$, te usporedno s njima gradijente svakog od C elemenata vektora $\mathbf{b}$. Označimo te retke odnosno elemente s $\mathbf{W}_{j:}$ i b_j, a njihove parcijalne derivacije s $\partial\mathcal{L}/\partial\mathbf{W}_{j:}$ (matrica dimenzija 1xD) i $\partial\mathcal{L}/\partial b_j$ (matrica dimenzija 1x1). Kao i ranije, gradijente je najlakše prikazati tako da ovisnost o točnom razredu i-tog podatka prikažemo Iversonovim uglatim zagradama. Prvo ćemo izraziti gradijent funkcije gubitka za i-ti podatak $\mathcal{L}_i$ po j-toj klasifikacijskoj mjeri (izvedite ovaj izraz za vježbu!): $\partial\mathcal{L}_i/\partial s_{ij} = P(Y = c_j|\mathbf{x}_i) - [\![y_i = c_j]\!]$.

U izvedbi je ove parcijalne derivacije najpraktičnije sakupiti u matricu $\mathbf{G_s} = [[\partial\mathcal{L}_i/\partial s_{ij}]_{j=1}^{C}]_{i=1}^{N}$ dimenzija NxC. Kako bismo tu matricu efikasno izračunali, uvodimo matricu aposteriornih vjerojatnosti podataka $\mathbf{P}_{ij} = P(c_j|\mathbf{x_i})$ te matricu vektorski kodiranih oznaka $\mathbf{Y}$:

$$\mathbf{Y}_{ij} = \begin{cases} 1, & y_i = c_j \\ 0, & y_i \neq c_j \end{cases}.$$

Sada matricu parcijalnih derivacija gubitka po klasifikacijskim mjerama dobivamo jednostavnim matričnim izrazom $\mathbf{G_s} = \mathbf{P} - \mathbf{Y}$.

Parcijalne derivacije klasifikacijskih mjera po parametrima iste su kao i kod binarne logističke regresije:

$\partial s_{ij}/\partial\mathbf{W}_{j:} = \mathbf{x_i}^\top$

$\partial s_{ij}/\partial b_j = 1.$

Kad ulančimo gradijente komponenata gubitka te uzmemo u obzir da ukupni gubitak zbraja doprinose po svim podatcima, dobivamo izraz kojeg možemo koristiti u optimizacijskom postupku:

$\partial \mathcal{L}/\partial \mathbf{W}_{j:} = 1/N \cdot \sum_i \partial \mathcal{L}_i/\partial \mathbf{W}_{j:} = 1/N \cdot \sum_i \partial \mathcal{L}_i/\partial s_{ij} \cdot \partial s_{ij}/\partial \mathbf{W}_{j:}$

$\partial \mathcal{L}/\partial \mathbf{W}_{j:} = 1/N \cdot \sum_i G_{sij} \cdot \mathbf{x}_i^\top$

$\partial \mathcal{L}/\partial b_j = 1/N \cdot \sum_i \partial \mathcal{L}_i/\partial b_j = 1/N \cdot \sum_i G_{sij}.$

Pažljivim promatranjem ovog izraza uočit ćemo ogromnu sličnost s odgovarajućim izrazom gradijenata parametara binarne logističke regresije. To nas ne treba čuditi, jer retci matrice težina $\mathbf{W}$ u višerazrednom gubitku sudjeluju na potpuno isti način kao i težine $\mathbf{w}$ u gubitku binarne logističke regresije. Komponente gradijenta $\partial \mathcal{L}/\partial \mathbf{W}_{j:}$ odgovaraju skalarnom produktu j-tog stupca matrice $\mathbf{G}_s$ sa stupcima podatkovne matrice $\mathbf{X}$. Stoga bismo, kao i kod binarne logističke regresije, sve gradijente j-tog retka matrice težina mogli izračunati umnoškom vektora $\mathbf{G}_{\mathbf{s}:j}$ i matrice $\mathbf{X}$:

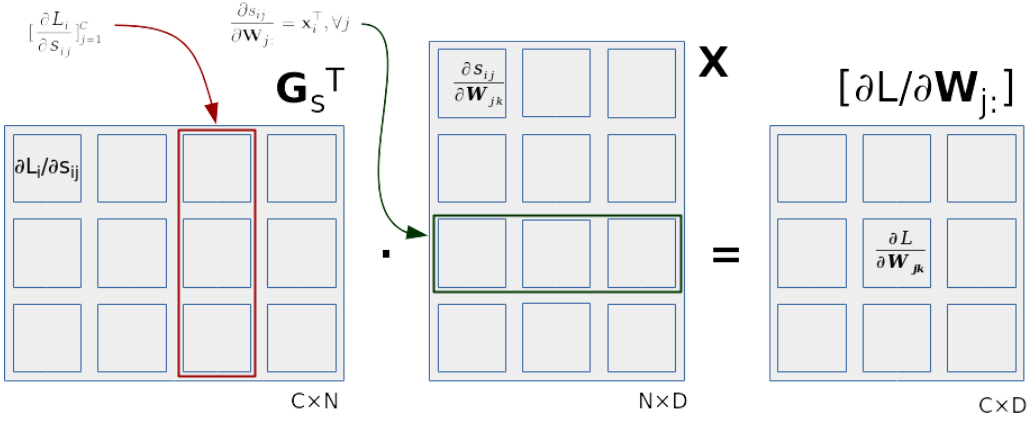
$$\partial \mathcal{L}/\partial \mathbf{W}_{j:} = 1/N \cdot \mathbf{G}_{\mathbf{s}:j} \cdot \mathbf{X} \ .$$

Međutim, pokazuje se da u praksi možemo i bolje od toga. Važno svojstvo našeg problema jest da su parcijalne derivacije klasifikacijskih mjera po odgovarajućim redcima matrice težina međusobno jednake: $\partial s_{ij}/\partial \mathbf{W}_{j:} = \mathbf{x_i}^\top \forall j$. Odatle slijedi da gradijente svih redaka matrice težina vrlo praktično možemo izračunati samo jednim matričnim umnoškom:

$$[\partial \mathcal{L}/\partial \mathbf{W}_{j:}]_{j=1}^{C} = 1/N \cdot \mathbf{G_s}^\top \cdot \mathbf{X}$$

Prikazani izraz relativno je lako zapamtiti, ali iznimno teško objasniti reverse engineeringom. Često se može čuti objašnjenje kako do tog izraza dolazimo izravnom primjenom pravila ulančavanja. Međutim, takvo objašnjenje je netočno jer pravilom ulančavanja ne možemo računati parcijalne derivacije po matričnoj varijabli. Ako ustrajemo na takvom pristupu, morat ćemo ili izravnati matricu u vektor ili prihvatiti da Jakobijan u tom slučaju postaje tenzor trećeg reda. U oba slučaja složenost našeg algoritma bi se povećala jer ne bismo mogli iskoristiti inherentnu algebarsku strukturu problema, koju možemo jednostavno izraziti ovako: $\partial s_{ij}/\partial \mathbf{W}_{k:} = 0, j \neq k$. Stoga, izraz $\mathbf{G_s}^\top \cdot \mathbf{X}$ treba promatrati kao *kompliciranu optimizaciju*, a ne kao početnu točku za razumijevanje načina za određivanje gradijenata matrice težina višerazredne logističke regresije. Molimo studente koji prakticiraju nelinearno učenje da ovu netrivijalnu ali vrlo važnu činjenicu prihvate i zapamte što je moguće prije.

Optimirani izraz za računanje gradijenata višerazredne logističke regresije prikazan je na sljedećoj slici. Na slici vidimo da doprinos svakog podatka gradijentima težina odgovara vanjskom umnošku stupca matrice $\mathbf{G_s}^\top$ (odnosno, retka matrice $\mathbf{G_s}$) i odgovarajućeg retka podatkovne matrice. Kad bismo imali jedan podatak (indeks i) i kad bismo računali gradijente samo jednog retka matrice težina (indeks j), matrica $\mathbf{G_s}$ svela bi se na element $\mathbf{G}_{\mathbf{s}ij}$, a matrica $\mathbf{X}$ na redak $\mathbf{x}_i$.



U praksi, gradijente težina $[\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{j:}}]_{j=1}^{C}$ računat ćemo jednim pozivom funkcije matrične biblioteke, npr. `grad_W = 1/N * np.dot(GsT, X)`. Na taj način jednim pozivom rješavamo sljedeća dva zadatka:

- istovremeno računanje gradijenata svih redaka matrice $\mathbf{W}$
- akumuliranje doprinosa svih podataka $\mathbf{x}_i$;

Vrlo sličan postupak imat ćemo i za gradijent pomaka $\mathbf{b}$, samo što ćemo umjesto matričnog množenja pozivati funkciju za istovremeno zbrajanje svih stupaca matrice, npr. `np.sum(GsT, axis=1)`.

Ovakve manevre vrlo rado ćemo koristiti u praksi kako programske petlje ne bi bile u programskom jeziku visoke razine nego u optimiranom bibliotečnom kodu koji može biti i do 100 puta brži od naivnog koda u C-u. Naime, ako je numpy preveden s OpenBLAS-om, izvedba matričnog množenja će koristiti optimirani kod koji jako pazi na promašaje cachea i temelji se na ručno optimiranom strojnom kodu koji je prilagođen za konkretnu mikroarhitekturu. Nadalje, ako je OpenBLAS konfiguriran da koristi OpenMP, u naš algoritam biti će upregnute sve procesorske jezgre - a to bi rezultiralo dodatnim ubrzanjem. Neke konkretne brojeve možete pogledati ovdje (u tim eksperimentima OpenMP je bio isključen).

0g. Uvod u Python

Python je moderni dinamički jezik opće namjene koji je zbog svoje univerzalne primjenljivosti postao iznimno popularan u područjima vezanim uz umjetnu inteligenciju. Ako niste položili Skriptne jezike, predlažemo da samostalno proučite skriptu ili knjigu. Koristit ćemo Python 3. Preporučamo vam da isprobate naprednu interaktivnu ljusku ipython.

Izvorni kod u Pythonu smještamo u datoteke koje nazivamo modulima. Svaki modul u sebi treba sadržavati kratki ispitni program koji testira funkcionalnost modula i ilustrira njegovo pravilno korištenje. Ispitne programe smještamo na kraj modula u tijelu uvjetne naredbe koja testira je li modul pokrenut kao glavni program. Na taj način omogućavamo provođenje testiranja jednostavnim izvršavanjem modula.

```
if __name__=="__main__":
  # test, test, test!
```

Postoji više načina za debuggiranje u Pythonu. Jedan od najzgodnijih oslanja se na ugrađeni Pythonov debugger `pdb` kojeg možemo pozvati izravno iz koda. Da bismo to proveli, potrebno je prvo uključiti odgovarajući modul:

```
import pdb
```

Zatim, na mjestu gdje želimo prekinuti izvođenje (breakpoint!) navedemo poziv:

```
pdb.set_trace()
```

Nakon tog poziva Python će otvoriti interaktivnu ljusku u kojoj će biti dostupna sva imena programa koja su bila vidljiva u trenutku poziva funkcije `set_trace`. Iz ljuske možemo ispisivati objekte, pozivati proizvoljne funkcije (!) i izvršavati proizvoljne naredbe Pythona (!). Pored toga, na raspolaganju nam je i čitav niz specijalnih naredbi za debuggiranje. Popis tih naredbi dobivamo naredbom `help` (neke od važnijih naredbi su `up`, `down` i `continue`). Sličan učinak možemo postići i pozivom funkcije `IPython.embed()` (međutim, tu naredbe za debuggiranje nisu dostupne).

Na kraju ćemo pokazati kako debuggirati kod koji baca neočekivane iznimke na sljedećem minimalnom primjeru:

```
def proba():
  a=5
  raise ValueError("Iznimka!")

if __name__=="__main__":
  proba()
```

Testiranjem ovog programa vrlo lako se možemo uvjeriti da njegovo izvršavanje završava iznimkom `ValueError`. Malo teže je međutim doći do podrobnijih informacija o stanju programa na mjestu gdje se iznimka dogodila, poput npr. ispisivanja vrijednosti imena `a`. Problem možemo riješiti pokretanjem debuggera `pdb proba.py` te zadavanjem naredbe `continue` (skraćeno "`c`"). Dubugger će početi izvoditi program te kad dođe do neuhvaćene iznimke omogućit će nam uvid u stanje programa preko tekstovnog

korisničkog sučelja. Primjerice, naredba `print a` će ispisati vrijednost imena `a` neposredno prije bacanja iznimke. Do debuggera možemo doći i iz alternativne ljuske `ipython` na način da nakon interaktivnog poziva čije izvršavanje završava neuhvaćenom iznimkom zadamo naredbu `debug`.

Ovaj tekst pretpostavlja rad u čistom Pythonu, a ne korištenje posrednika poput Jupytera i sl. Ako vam se sviđa, slobodno koristite Jupyter, ali ne garantiramo da će sve raditi kako piše u ovim uputama.

0h. Uvod u Numpy

Numpy je popularna numerička biblioteka za Python. U primjerima ćemo koristiti standardnu pokratu `np` do koje dolazimo naredbom:

```
import numpy as np
```

Matplotlib je biblioteka koja se često kombinira s numpyjem, a služi za izradu raznih vrsta grafova u Pythonu. U primjerima ćemo koristiti standardnu pokratu `plt` do koje dolazimo naredbom:

```
import matplotlib.pyplot as plt
```

Važna prednosti numpyja jest mogućnost svođenja iterativnih postupaka na vektorske i matrične izraze. Za primjer, pogledajmo kako bismo u numpyju izrazili afinu transformaciju podataka koja predstavlja važan korak pri učenju logističke regresije. Neka su podatci zadani numpyjevom matricom `X` dimenzija NxD (svaki redak odgovara jednom D-dimenzijskom podatku), neka je transponirana matrica težina $\mathbf{W}^\top$ zadana numpyjevom matricom `W_t` dimenzija DxC te neka je transponirani vektor pomaka $\mathbf{b}^\top$ zadan numpyjevim vektorom `b_t` dimenzija 1xC. Tada afinu transformaciju *svih* podataka dobivamo jednostavnim pozivom numpyjevih funkcija za matrično množenje i zbrajanje:

```
H = np.dot(X, W_t) + b_t
```

Obratimo međutim pozornost na jedan detalj koji bi nakon površnog pogleda mogao i promaknuti. Pažljivom analizom prikazane naredbe Pythona uočavamo da dimenzije pribrojnika ne odgovaraju: matrični umnožak `np.dot(X, W_t)` ima dimenzije NxC dok vektor `b_t` ima dimenziju 1xC (N != 1). Biblioteka numpy takva nesuglasja rješava pod pretpostavkom da vektor `b_t` treba dodati *svakom* retku matričnog umnoška. Stoga će nakon izvršavanja naredbe dodjeljivanja ime `H` uistinu referencirati matricu čiji retci odgovaraju afino transformiranim retcima podatkovne matrice. Kad se podsjetimo da se i-ti podatak nalazi u i-tom retku matrice podataka ($\mathbf{x}_i = \mathbf{X}_{i:}^\top$), konačnu vrijednost i-tog retka matrice `H` moći ćemo zapisati i sljedećim izrazom: $\mathbf{H}_{i:}^\top = \mathbf{W} \cdot \mathbf{x}_i + \mathbf{b}$.

Upravo smo vidjeli da numpy po potrebi automatski *umnožava* operand manje dimenzionalnosti (engleski termin je broadcasting) kako bi došao do valjanog izraza. Ovakvo izražavanje je ispočetka vrlo teško prihvatiti, ali nakon nekog vremena postaje jasno da prednosti (sažetost i efikasnost) uvjerljivo nadjačavaju nedostatke.

1. Stvaranje umjetnog skupa 2D podataka

Rješenje ove vježbe slobodno preuzmite ovdje.

Ponovite Python, numpy, i matplotlib prema odjeljcima 0g i 0h.

U ovoj vježbi ćemo izvesti razred `Random2DGaussian` koji će nam omogućiti uzorkovanje 2D podataka generiranih slučajnom Gaussovom distribucijom. Konstruktor razreda treba stvarati parametre slučajne bivarijatne Gaussove razdiobe (vektor μ i matricu Σ). Metoda `get_sample(n)` treba vratiti `n` slučajno uzorkovanih 2D podataka u numpyjevom polju dimenzija Nx2. Položaj i varijanca razdiobe trebaju biti ograničeni fiksnim parametrima.

Pomoć:

- Definirajte raspon prihvatljivih vrijednosti za obje koordinate sredine razdiobe μ (npr. `minx=0`, `maxx=10`, `miny=0`, `maxy=10`); neka to budu podatkovni članovi razreda.
- Slučajno izaberite sredinu razdiobe μ prema uniformnoj razdiobi (`np.random.random_sample`).
- Slučajno odaberite svojstvene vrijednosti kovarijacijske matrice Σ prema uniformnoj razdiobi i organizirajte ih u dijagonalnu matricu D. Neka gornja granica svojstvenih vrijednosti ovisi o rasponu prihvatljivih vrijednosti parametra μ, npr.

```
eigvalx = (np.random.random_sample()*(maxx - minx)/5)**2
```

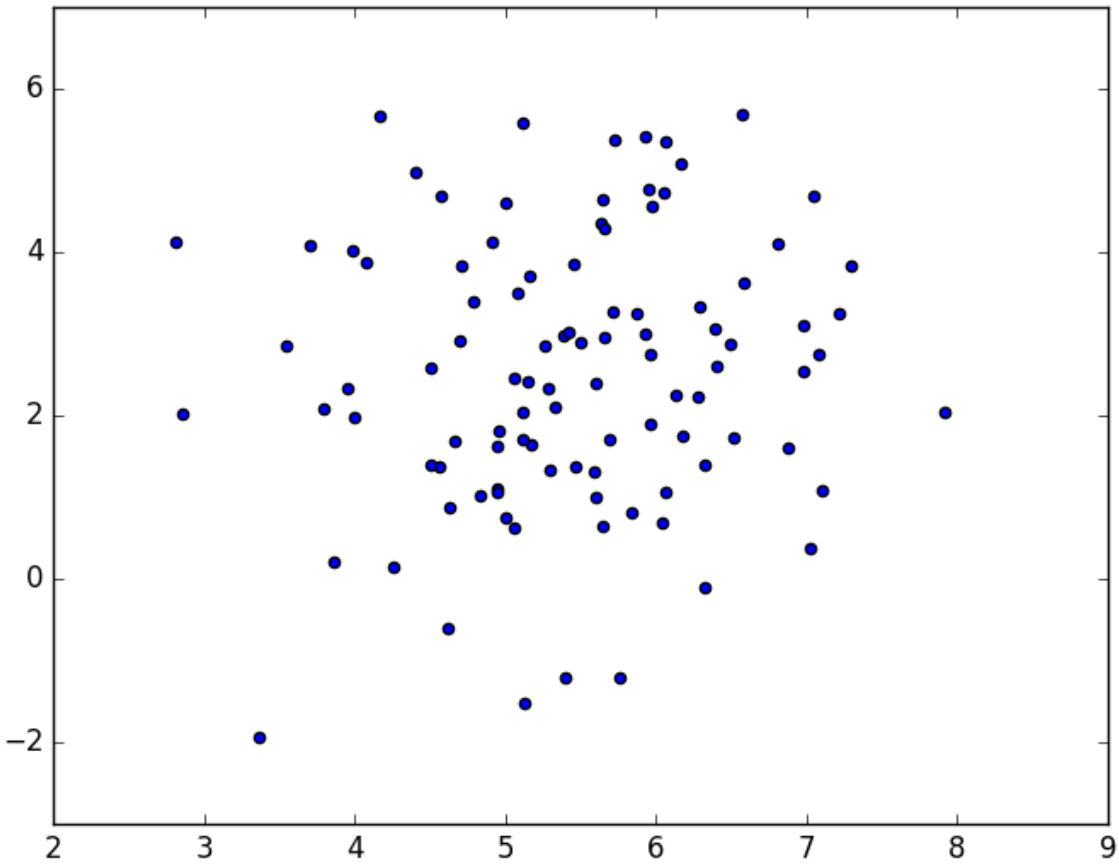
- Slučajno odaberite kut rotacije kovarijacijske matrice i na temelju njega formirajte rotacijsku matricu $R = \begin{bmatrix} \cos(\varphi) & -\sin(\varphi) \\ \sin(\varphi) & \cos(\varphi) \end{bmatrix}$.
- Odredite matricu Σ kao umnožak transponirane matrice R, matrice D i matrice R.
- Za uzorkovanje koristite funkciju `np.random.multivariate_normal`.
- Kako biste osigurali ponovljivost eksperimenata sa slučajno generiranim podatcima, numpyjev generator slučajnih brojeva slobodno inicijalizirajte na konstantu:

```
np.random.seed(100)
```

Ispravnost koda provjerite sljedećim ispitnim programom:

```
if __name__=="__main__":
  G=Random2DGaussian()
  X=G.get_sample(100)
  plt.scatter(X[:,0], X[:,1])
  plt.show()
```

Ovisno o stanju generatora slučajnih brojeva, vaš rezultat mogao bi izgledati kao na sljedećoj slici:



Ako je rezultat prihvatljiv, pohranite kod u datoteku `data.py`.

2. Učenje binarne logističke regresije gradijentnim spustom

Ponovite binarnu logističku regresiju prema odjeljcima 0d i 0e.

Napišite funkciju `binlogreg_train` koja optimizira parametre logističke regresije $\mathbf{w}$ i b na zadanom skupu podataka za učenje. U vašem izvornom kodu koristite sljedeća imena:

- `X`: matrica podataka dimenzija NxD;
- `Y_`: vektor *točnih* razreda podataka dimenzija Nx1 (koristimo ga tijekom učenja);
- `Y`: vektor *predviđenih* razreda podataka dimenzija Nx1 (koristimo ga tijekom ispitivanja performanse).

Neka funkcija ima sljedeće sučelje:

```
def binlogreg_train(X,Y_):
  '''
    Argumenti
      X:  podatci, np.array NxD
      Y_: indeksi razreda, np.array Nx1

    Povratne vrijednosti
      w, b: parametri logističke regresije
  '''
```

Upute:

- Inicijalizirajte w prema normalnoj distribuciji N(0,1) (np.random.randn) te b na nulu.
- Otvorite petlju gradijentnog spusta. Broj iteracija postupka zadajte hiper-parametrom `param_niter`.

- Izračunajte klasifikacijske mjere svih podataka na način da petlje prepustite numpyju (`scores = np.dot(X, w) + b`).
- Izračunajte aposteriorne vjerojatnosti razreda u svim podatcima $P(c_1|\mathbf{X})$ na način da petlje prepustite numpyju
- Izračunajte gubitak $\mathcal{L}(\mathbf{w}, \mathbf{b})$ na način da petlje prepustite numpyju.
- Izračunajte gradijente `grad_w` i `grad_b` na način da petlje prepustite numpyju.
- Izmijenite parametre u smjeru negativnog gradijenta (neka faktor pomaka `param_delta` bude hiper-parametar postupka).

Struktura vaše izvedbe gradijentnog spusta trebala bi izgledati ovako:

```
 # gradijentni spust (param_niter iteracija)
for i in range(param_niter):
  # klasifikacijske mjere
  scores = ...    # N x 1

  # vjerojatnosti razreda c_1
  probs = ...     # N x 1

  # gubitak
  loss  = ...     # scalar

  # dijagnostički ispis
  if i % 10 == 0:
    print("iteration {}: loss {}".format(i, loss))

  # derivacije gubitka po klasifikacijskim mjerama
  dL_dscores = ...     # N x 1

  # gradijenti parametara
  grad_w = ...     # D x 1
  grad_b = ...     # 1 x 1

  # poboljšani parametri
  w += -param_delta * grad_w
  b += -param_delta * grad_b
```

Napišite funkciju `binlogreg_classify` koja klasificira zadani skup podataka u skladu sa zadanim parametrima logističke regresije. Neka funkcija ima sljedeće sučelje:

```
'''
  Argumenti
      X:    podatci, np.array NxD
      w, b: parametri logističke regresije

  Povratne vrijednosti
      probs: vjerojatnosti razreda c1
'''
```

Napišite funkciju `sample_gauss_2d(C, N)` koja stvara `C` slučajnih bivarijatnih Gaussovih razdioba (prisjetimo se, njih smo u zadatku 1 implementirali razredom `Random2DGaussian`), te iz svake od njih uzorkuje `N` podataka. Funkcija treba vratiti matricu `X` dimenzija (N·C)x2 čiji retci odgovaraju uzorkovanim podatcima te matricu točnih razreda `Y` dimenzija (N·C)x1 čiji jedini stupac sadrži indeks razdiobe iz koje je uzorkovan odgovarajući podatak. Ako je i-ti redak matrice `X` uzorkovan iz razdiobe j, onda mora biti `Y[i,0]==j`.

Napišite funkciju `eval_perf_binary(Y,Y_)` koja na temelju predviđenih i točnih indeksa razreda određuje pokazatelje performanse binarne klasifikacije: točnost (engl. accuracy), preciznost (engl. precision) te odziv (engl. recall). Implementaciju te funkcije temeljite na brojnostima istinitih pozitiva (TP), lažnih pozitiva (FP), istinith negativa (TN) i lažnih negativa (FN).

Napišite funkciju `eval_AP` koja računa prosječnu preciznost binarne klasifikacije. Neka funkcija na ulazu prima rangiranu listu točnih razreda $\mathbf{Y_r}$ koju dobivamo kad matricu točnih razreda $\mathbf{Y}$ sortiramo prema aposteriornim vjerojatnostima odgovarajućih podataka $P(c_1|\mathbf{x})$. Rangiranu listu točnih razreda možete dobiti pozivom metode

`argsort` numpyjevog polja. Prosječnu preciznost možete izračunati primjenom sljedećeg izraza: $\mathrm{AP} = \frac{\sum_{i=0}^{N-1} \mathrm{Preciznost}(i) \cdot \mathbf{Y}_{ri}}{\sum_{i=0}^{N-1} \mathbf{Y}_{ri}}$. Pri tome $\mathrm{Preciznost}(i)$ odgovara preciznosti u slučaju kad podatke s indeksom većim ili jednakim i pridružimo razredu c_1, a podatke s indeksom manjim od i - razredu c_0. Evo kako bi se trebala ponašati vaša funkcija:

```
>>> import data
>>> data.eval_AP([0,0,0,1,1,1])
1.0
>>> data.eval_AP([0,0,1,0,1,1])
0.9166666666666666
>>> data.eval_AP([0,1,0,1,0,1])
0.7555555555555555
>>> data.eval_AP([1,0,1,0,1,0])
0.5
```

Napišite ispitni kod za modul `binlogreg.py`. Formirajte skup za učenje pozivom funkcije `sample_gauss_2d`. Pozovite funkciju za učenje te nakon toga provedite klasifikaciju primjera za učenje. Predviđene vjerojatnosti podataka pretvorite u indekse razreda Y pod pretpostavkom odabira razreda s najvećom vjerojatnošću (izbjegnite petlju u Pythonu!). Ispišite pokazatelje performanse dobivene pozivima funkcija `eval_perf_binary` te `eval_AP`. Neka vaš ispitni kod izgleda kako slijedi.

```
if __name__=="__main__":
  np.random.seed(100)

  # get the training dataset
  X,Y_ = data.sample_gauss_2d(2, 100)

  # train the model
  w,b = binlogreg_train(X, Y_)

  # evaluate the model on the training dataset
  probs = binlogreg_classify(X, w,b)
  Y = # TODO

  # report performance
  accuracy, recall, precision = data.eval_perf_binary(Y, Y_)
  AP = data.eval_AP(Y_[probs.argsort()])
  print (accuracy, recall, precision, AP)
```

Ako vaš postupak ne postiže točnost od 100%, pokušajte naći objašnjenje.

Ako je rezultat prihvatljiv, pohranite kod u datoteku `binlogreg.py`. Funkcije `sample_gauss_2d`, `eval_AP` i `eval_perf_binary` pohranite u datoteku `data.py`.

Dodatni zadatci za radoznale.

- Provjerite poklapaju li se vrijednosti analitičkih gradijenata s njihovim numeričkim aproksimacijama (engl. gradient checking).
- Dodajte regularizacijski gubitak u obliku L2 norme težina w pomnožene hiper-parametrom `param_lambda` (parametre b najčešće nema smisla regularizirati). Ne zaboravite izraziti utjecaj te promjene na gradijente.
- Eksperimentirajte s različitim vrijednostima hiper-parametara `param_niter`, `param_delta` i `param_lambda`. Pronađite kombinacije hiper-parametara za koje vaš program ne uspijeva pronaći zadovoljavajuće rješenje i objasnite što se događa.
- Smislite način za vizualiziranje plohe funkcije gubitka i napredovanje postupka optimizacije.
- Procijenite pristranost (eng. bias) i varijancu vašeg postupka većim brojem eksperimenata na podatcima za testiranje. Podatke za testiranje dobijte uzorkovanjem iste podatkovne distribucije koja je korištena za dobivanje skupa za učenje. Možete li zadati distribuciju podataka za koju će pristranost klasifikatora biti vrlo velika?

3. Grafički prikaz rezultata klasifikacije

Rješenje ovog zadatka slobodno preuzmite ovdje.

Napišite funkciju `graph_data` za vizualiziranje rezultata binarne klasifikacije 2D podataka. Neka funkcija ima sljedeće parametre.

```
'''
  X  ... podatci (np.array dimenzija Nx2)
  Y_ ... točni indeksi razreda podataka (Nx1)
  Y  ... predviđeni indeksi razreda podataka (Nx1)
'''
```

Upute:

- Podatke iscrtajte funkcijom `plt.scatter`; koristite imenovane argumente `c` i `marker` za zadavanje boje, veličine i oblika simbola koji predstavljaju podatke.
- Točan razred podatka prikažite bojom simbola: razred 0 iscrtajte sivom, a razred 1 bijelom bojom.
- Točno klasificirane podatke (`Y_==Y`) prikažite kružićima, a netočno klasificirane podatke kvadratima. Kako biste izbjegli petlju u Pythonu, funkciju `plt.scatter` možete pozvati dva puta: jednom za točno klasificirane, a jednom za netočno klasificirane podatke.

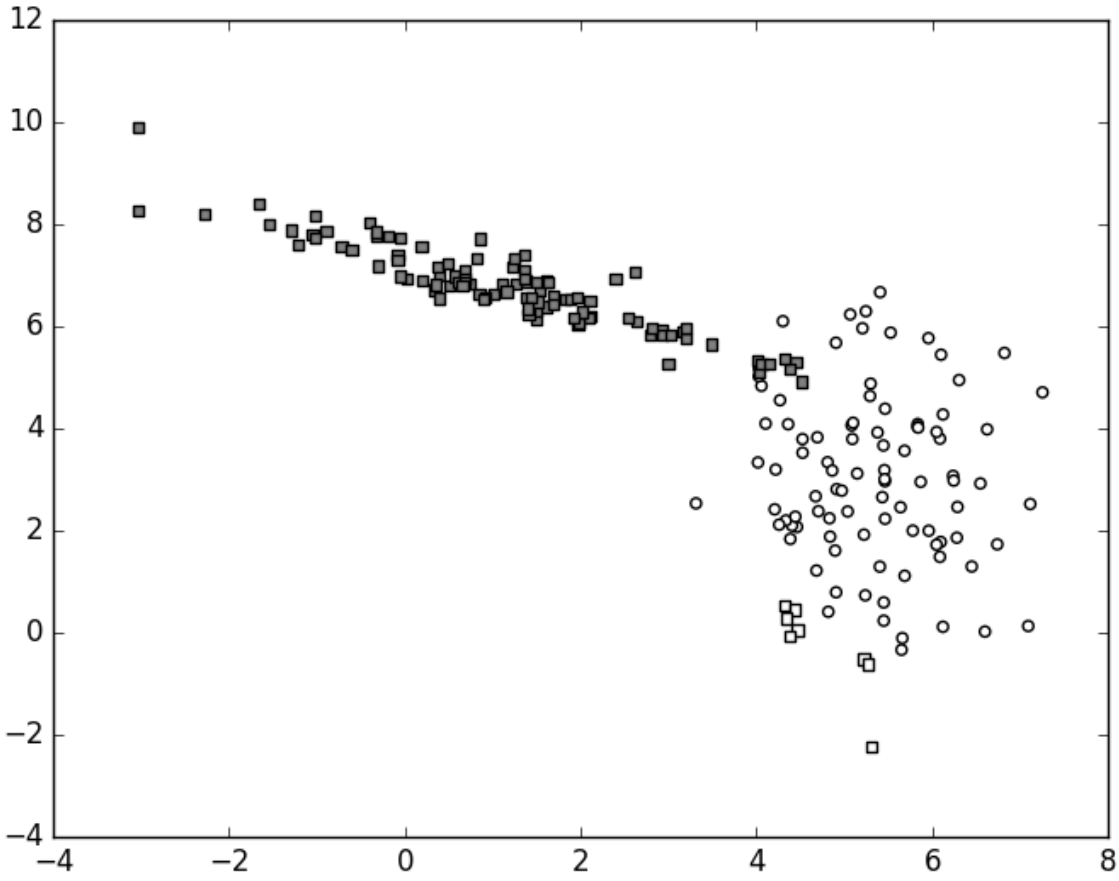
Ispitajte vaš dosadašnji kod na podatcima generiranim dvjema slučajnim razdiobama. Za potrebe prvog testiranja, predviđene indekse razrede podataka (`Y`) odredite proizvoljnom funkcijom odluke, npr:

```
def myDummyDecision(X):
  scores = X[:,0] + X[:,1] - 5
  return scores
```

Vaš novi ispitni program za modul `data` mogao bi izgledati ovako:

```
if __name__=="__main__":
  np.random.seed(100)

  # get the training dataset
  X,Y_ = data.sample_gauss_2d(2, 100)

  # get the class predictions
  Y = myDummyDecision(X)>0.5

  # graph the data points
  graph_data(X, Y_, Y)

  # show the results
  plt.show()
```

Ako koristimo istu verziju generatora slučajnih brojeva, vaš rezultat sada bi trebao izgledati kao na sljedećoj slici:



Funkcija odluke `myDummyDecision` mijenja vrijednost po dijagonali sjeverozapad-jugoistok, a njena granica prolazi blizu dna nakupine "bijelih". Stoga su "bijeli" podatci uglavnom ispravno klasificirani (kružići), dok su svi "sivi" podatci neispravno klasificirani (kvadrati).

Ako je rezultat ispitivanja prihvatljiv, pohranite kod u datoteku `data.py`.

4. Iscrtavanje funkcije odluke

Rješenje ovog zadatka slobodno preuzmite ovdje.

U praksi, funkcije odluke klasifikatora najčešće ne vraćaju predviđeni razred podatka kao što je to bio slučaj kod naše funkcije `myDummyDecision`. Primjerice, funkcija odluke stroja s potpornim vektorima vraća klasifikacijske mjere. Predznak rezultata određuje predviđeni razred, a granicu između dvaju razreda čine podatci u kojima je klasifikacijska mjera jednaka nuli. Binarna logistička regresija vraća aposteriornu vjerojatnost $P(Y = c_1|\mathbf{x})$. Ako je ta vjerojatnost veća od 0.5, klasifikator predviđa razred c_1, a granicu čine podatci u kojima je ta vjerojatnost jednaka 0.5. Jednaku interpretaciju najčešće imaju i funkcije odluke dubokih klasifikatora koji, za razliku od logističke regresije, mogu modelirati i vrlo nelinearne granice među razredima. Kod svih navedenih klasifikatora oblik funkcije odluke pruža kvalitativan uvid u ono što je klasifikator naučio.

Stoga ćemo se u ovom zadatku posvetiti vizualizaciji funkcije odluke. Naš cilj je razviti potprogram `graph_surface` koji iscrtava plohu zadane skalarne funkcije 2D varijable kako bismo njime mogli iscrtavati plohe funkcije odluke klasifikatora dvodimenzionalnih podataka. Potprogram `graph_surface` treba imati sljedeće argumente:

```
'''
  fun    ... decizijska funkcija (Nx2)->(Nx1)
  rect   ... željena domena prikaza zadana kao:
             ([x_min,y_min], [x_max,y_max])
  offset ... "nulta" vrijednost decizijske funkcije na koju
```

```
            je potrebno poravnati središte palete boja;
            tipično imamo:
            offset = 0.5 za probabilističke modele
               (npr. logistička regresija)
            offset = 0 za modele koji ne spljošćuju
               klasifikacijske mjere (npr. SVM)
  width,height ... rezolucija koordinatne mreže
'''
```

Kao i kod potprograma `graph_data` naš kod će biti tanak omotač oko biblioteke matplotlib. Kako bismo pozivatelja zaštitili od izvedbenih detalja iscrtavanja, argument `fun` potprograma `graph_surface` predstavlja funkciju odluke (radi se o oblikovnom obrascu Strategija). Funkcija odluke `fun` preslikava podatke u numeričke vrijednosti koje određuju predviđeni razred odgovarajućeg podatka pa će to u praksi biti tanak omotač oko klasifikatora koejg želimo vizualizairati. Kako bismo izbjegli potrebu za višestrukim pozivanjem iz pythonskih petlji, funkcija odluke treba primiti podatkovnu matricu NxD, te vratiti vektor klasifikacijskih mjera dimenzija Nx1. Argument `rect` potprograma `graph_surface` zadaje dio podatkovnog vektorskog prostora u kojem treba vizualizirati funkciju odluke. U praksi, taj argument će ovisiti o rasponu podataka: nema smisla prikazivati funkciju odluke u dijelovima prostora gdje nema podataka za učenje. Potprogram `graph_surface` treba primiti i argument `offset` koji zadaje vrijednost funkcije odluke koja predstavlja granicu između razreda. Za probabilističke modele ta granica će biti 0.5, dok ćemo kod modela koji maksimiziraju marginu tu granicu postaviti na 0. Konačno, potprogram `graph_surface` treba primiti i dimenzije koordinatne mreže koje će definirati zrnatost vizualizacije.

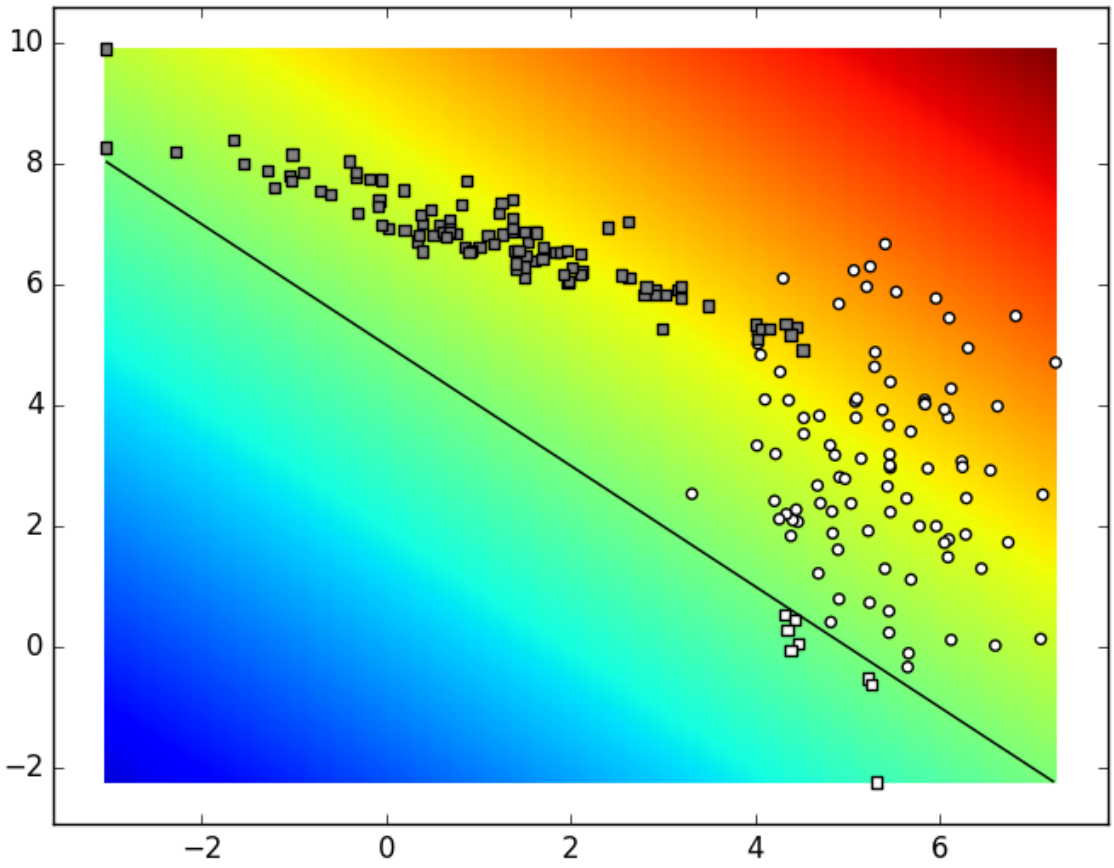
Plohe u matplotlibu iscrtavamo primjenom funkcija `np.meshgrid` i `plt.pcolormesh`, kao što je to pokazano u ovom primjeru i službenoj dokumentaciji biblioteke Matplotlib. Kako ta procedura nije baš najintuitivnija, za ovaj zadatak ćemo dati nešto detaljnije instrukcije.

- Izradite 1D koordinatne raspone rezolucije `width`×`height` u okvirima zadanog pravokutnika `rect` korištenjem funkcije `np.linspace` (ili `np.arange`).
- Izradite 2D koordinatnu mrežu pozivom `np.meshgrid`.
- Izrazite koordinatnu mrežu 2D matricom dimenzija Nx2. Ovaj korak nam treba kako bismo koordinatnu mrežu mogli koristiti kao parametar decizijske funkcije. Koristite metodu `flatten` i funkciju `np.stack`
- Pozovite decizijsku funkciju kako biste dobili njene vrijednosti u točkama koordinatne mreže. Preoblikujte dobivene vrijednosti u oblik kojeg zahtijeva funkcija `plt.pcolormesh` metodom `reshape`.
- Iscrtajte plohu pozivom funkcije `plt.pcolormesh`. Koristite argumente `vmin` i `vmax` tako da paletu centrirate u vrijednosti `offset` te tako da sve vrijednosti funkcije budu u dometu palete.
- Iscrtajte konturu uzduž koje decizijska funkcija poprima vrijednost `offset`. Koristite crnu boju.

Evo kako bismo predloženim potprogramom iscrtali plohu decizijske funkcije `myDummyDecision` u pravokutniku određenom koordinatama skupa točaka `X`.

```
    bbox=(np.min(X, axis=0), np.max(X, axis=0))
    graph_surface(myDummyDecision, bbox, offset=0)
```

Ako ovaj odsječak uvrstimo u ispitni program modula `data.py` neposredno prije poziva funkcije `graph_data`, ispod naših podataka osvanut će decizijska ploha u duginim bojama. Točke u kojima decizijska funkcija poprima vrijednost praga označene su crnom linijom. Bijeli podatci iznad crne linije označeni su kružićima (`myDummyClassifier` ih je dobro klasificirao). Bijeli podatci ispod linije i crni podatci iznad linije označeni su kvadratićima (`myDummyClassifier` ih je loše klasificirao).



Ako je rezultat ispitivanja prihvatljiv, pohranite kod u datoteku `data.py`.

5. Grafički prikaz binarne logističke regresije

Sada imamo gotovo sve sastojke za konačnu provjeru naše implementacije binarne logističke regresije. Nedostaje nam samo funkcija koja bi bila prikladna za slanje potprogramu `graph_surface`. U slučaju binarne logističke regresije, željena funkcija trebala bi primiti podatke `X`, a vratiti vektor aposteriorne vjerojatnosti razreda. Funkcija `binlogreg_classify` izgleda kao obećavajući kandidat, ali ne može se izravno primijeniti jer prima dva argumenta viška (`w` i `b`). U programskim jezicima sa statičkim funkcijama ovaj izazov bismo teško riješili bez globalnih varijabli. Međutim, u Pythonu ovo možemo lakoćom izvesti primjenom kontekstne funkcije (engl. closure) koju možemo konstruirati pozivom sljedeće funkcije:

```
def binlogreg_decfun(w,b):
  def classify(X):
    return binlogreg_classify(X, w,b)
  return classify
```

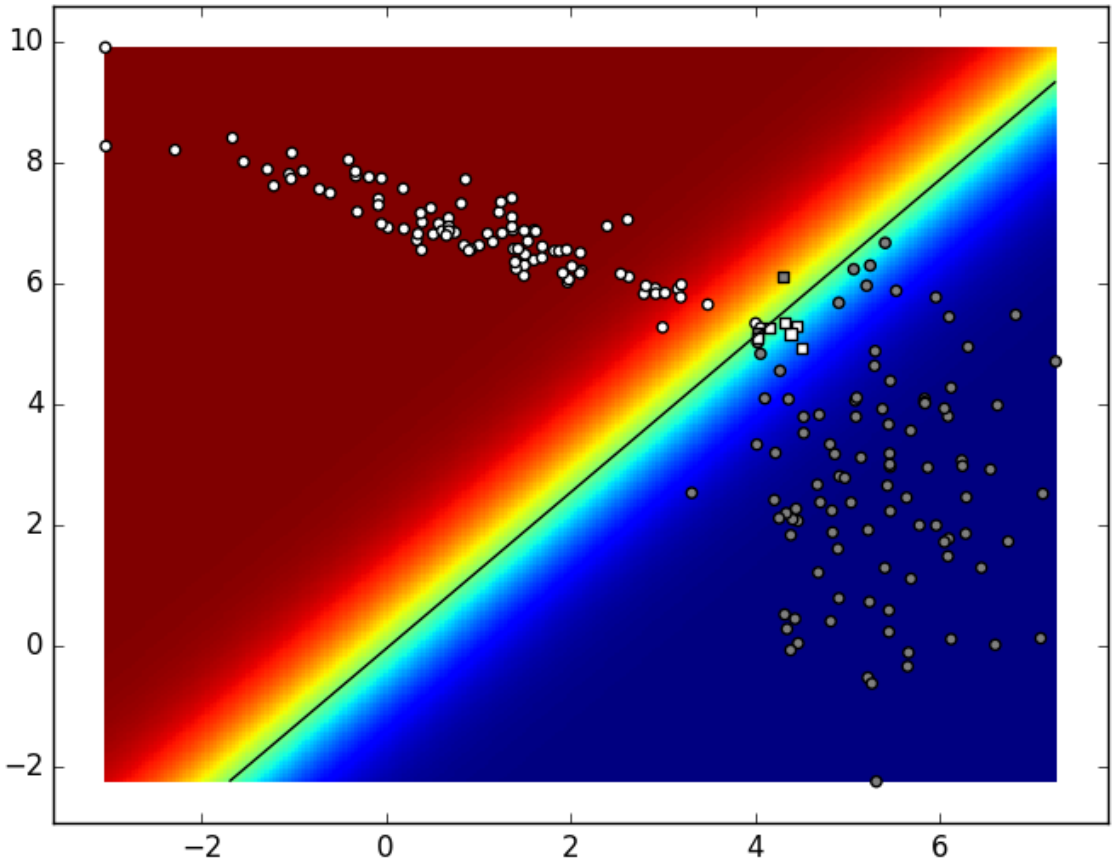
Funkcija `binlogreg_decfun` vraća lokalnu funkciju koja pamti kontekst (w,b). Stoga tu lokalnu funkciju možemo koristiti kao argument funkcije `graph_surface` (radi se o oblikovnom obrascu Dekorator). Prikazana fukcionalnost može se sažetije ostvariti lambda izrazom (napravite to za vježbu!). Početnicima savjetujemo da napreduju malim koracima te da lambda izraze koji pamte kontekst počnu koristiti tek nakon što u potpunosti usvoje kontekstne funkcije.

Sada konačno možemo iscrtati rezultate naše izvedbe binarne logističke regresije sljedećim kodom:

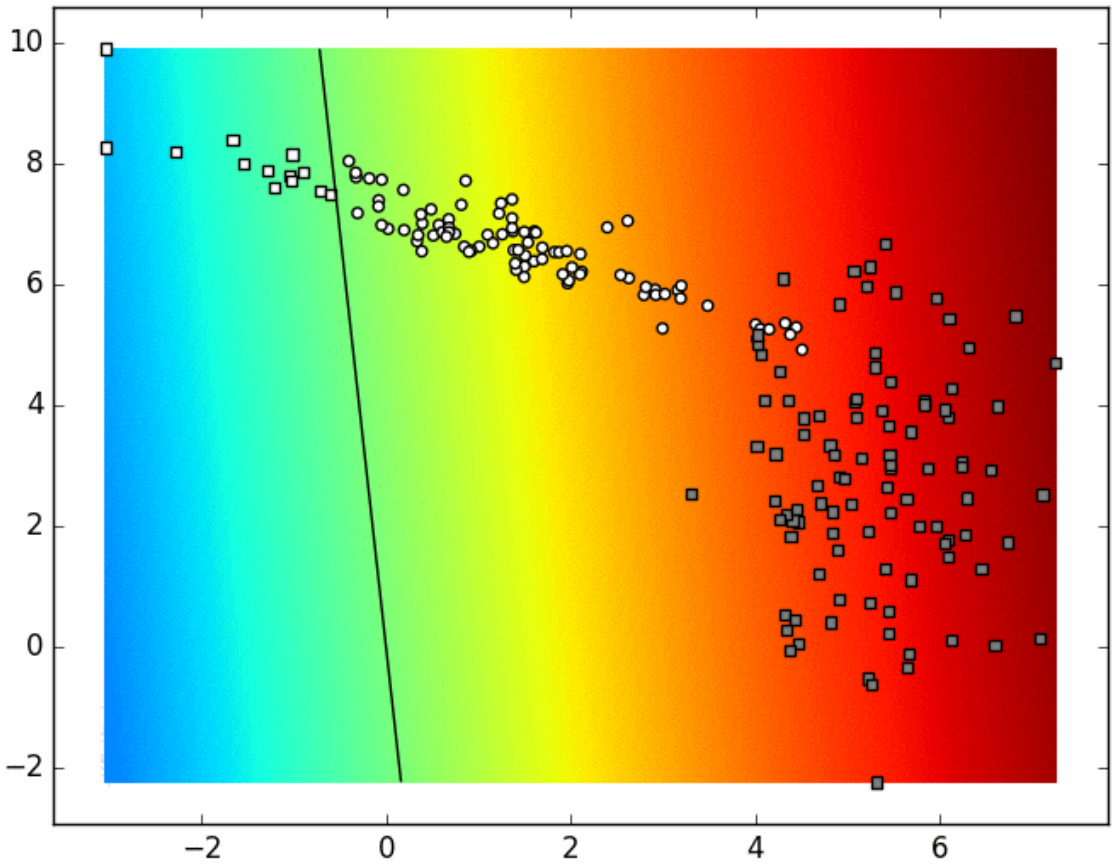
```
# instantiate the dataset
# ...
```

```
# train the logistic regression model
# ...

# evaluate the model on the train set
# ...

# recover the predicted classes Y
# ...

# evaluate and print performance measures
# ...

# graph the decision surface
decfun = binlogreg_decfun(w,b)
bbox=(np.min(X, axis=0), np.max(X, axis=0))
data.graph_surface(decfun, bbox, offset=0.5)

# graph the data points
data.graph_data(X, Y_, Y, special=[])

# show the plot
plt.show()
```

Ovisno o stanju generatora slučajnih brojeva te vrijednostima hiper-parametara vaš rezultat mogao bi izgledati kao na sljedećoj slici:



Ako sliku izradimo u svakoj iteraciji postupka, dobivamo sljedeću animaciju. Pokušajte objasniti razlike između konačnog stanja ove animacije i funkcije odluke koja je prikazana na prethodnoj slici.



6. Višerazredna logistička regresija

Ponovite višerazrednu logističku regresiju prema odjeljcima 0d i 0f.

Naš konačni zadatak je napisati kod za učenje i primjenu logističke regresije nad podatcima proizvoljnog broja razreda. Potrebno je napisati funkciju `logreg_train` koja kao argumente prima podatkovnu matricu `X` i matricu indeksa točnih razreda `Y_`. Na izlazu je potrebno vratiti parametre logističke regresije `W` i `b`. Dimenzije tih parametara trebaju ovisiti o broju razreda `C` kojeg možete odrediti izrazom `max(Y_) + 1`.

Struktura tijela petlje gradijentnog spusta trebala bi izgledati ovako:

```
# eksponencirane klasifikacijske mjere
  # pri računanju softmaksa obratite pažnju
  # na odjeljak 4.1 udžbenika
  # (Deep Learning, Goodfellow et al)!
  scores = ...    # N x C
  expscores = ... # N x C

  # nazivnik sofmaksa
  sumexp = ...    # N x 1

  # logaritmirane vjerojatnosti razreda
  probs = ...     # N x C
  logprobs = ...  # N x C

  # gubitak
  loss  = ...     # scalar

  # dijagnostički ispis
  if i % 10 == 0:
    print("iteration {}: loss {}".format(i, loss))

  # derivacije komponenata gubitka po mjerama
  dL_ds = ...     # N x C
```

```
    # gradijenti parametara
    grad_W = ...    # C x D (ili D x C)
    grad_b = ...    # C x 1 (ili 1 x C)

    # poboljšani parametri
    W += -param_delta * grad_W
    b += -param_delta * grad_b
```

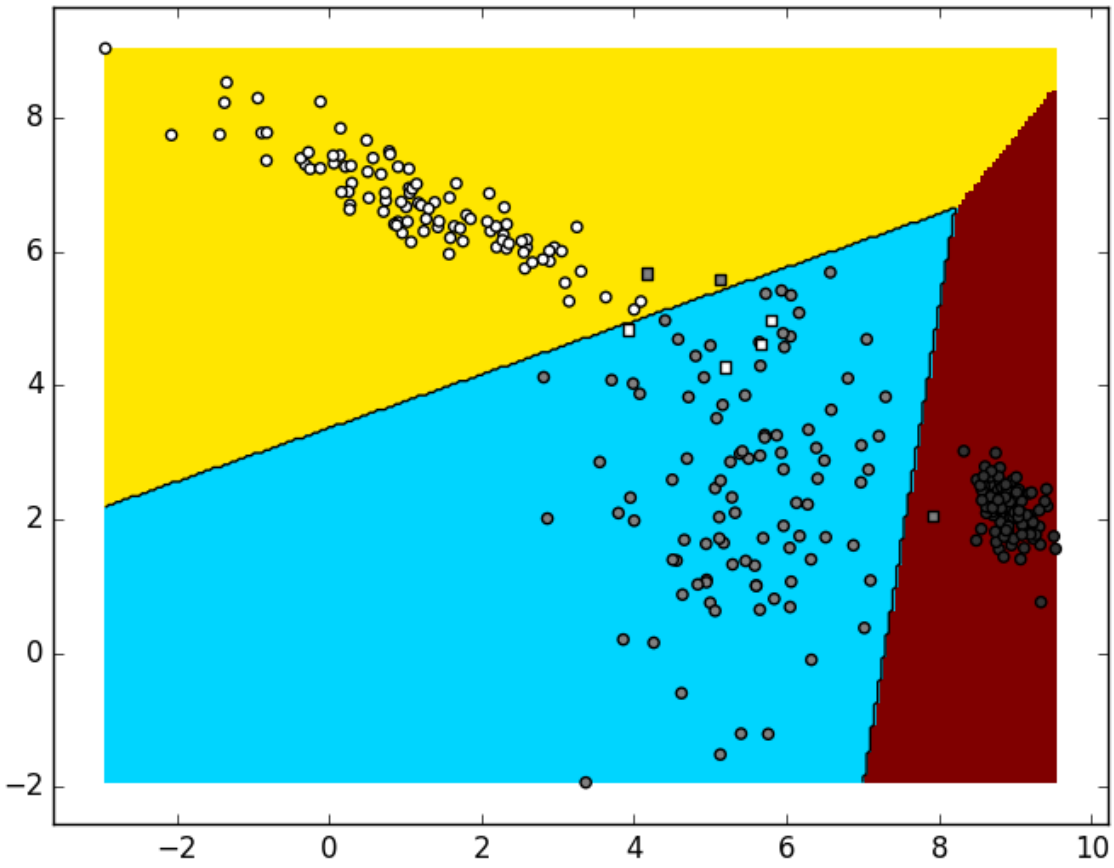
Napišite funkciju `logreg_classify` koja kao rezultat vraća matricu dimenzija NxC gdje svaki redak `i` sadrži vjerojatnosti klasificiranja podatka $\mathbf{x_i}$ u razrede c_j, $j \in \{0, 1, \ldots, C-1\}$.

Napišite funkciju `eval_perf_multi(Y,Y_)` koja na temelju predviđenih i točnih indeksa razreda određuje pokazatelje klasifikacijske performanse: i) ukupnu točnost klasifikacije, ii) matricu zabune (engl. confusion matrix) u kojoj retci odgovaraju točnim razredima a stupci predikcijama te iii) vektore preciznosti i odziva pojedinih razreda. U implementaciji prvo izračunajte matricu zabune, a onda sve ostale pokazatelje na temelju nje.

Konstruirajte dekorator oko `logreg_classify` koji omogućava evaluiranje modela iz funkcije `graph_surface`. Decizijsku plohu možete prikazati na više načina:

- kao vjerojatnost nekog određenog razreda,
- kao maksimalnu vjerojatnost klasificiranja u bilo koji razred,
- kao razred koji ima najveću aposteriornu vjerojatnost.

Isprobajte vaš kod najprije za slučaj C=2. Trebali biste dobiti vrlo slične rezultate kao i u binarnom slučaju. Nakon toga, provjerite što se zbiva kad vašem kodu pošaljete podatke uzorkovane iz tri slučajne Gaussove distribucije. Obratite pažnju da, kao i ranije, podatke iz svake pojedine distribucije valja označiti zasebnim razredom. Ovisno o stanju generatora slučajnih brojeva te vrijednostima hiper-parametara vaš rezultat mogao bi izgledati kao na sljedećoj slici:



Ako je rezultat ispitivanja prihvatljiv, pohranite kod u datoteku `logreg.py`.

Ove stranice sastavljaju Petra Bevandić, Marin Oršić, Ivan Grubišić, Josip Šarić i Siniša Šegvić.
Prva verzija ovih stranica bila je rezultat istraživačkog projekta MULTICLOD (I-2433-2014) Hrvatske zaklade za znanost.
Stranice su izrađene vi-jem i geditom.
Posljednja promjena: Monday, 01-Jun-2020 19:53:01 CEST
Svi komentari su dobrodošli: sinisa.segvic@fer.hr Povratak