

Razpoznavanje Vzorcev

Poročilo izbirne laboratorijske vaje 6B

Avtor: Bor Starčič

Mentorja: izr. prof. dr. Simon Dobrišek, asist. dr. Klemen Grm

Datum: January 14, 2024

Contents

1	Uvod	3
2	Učna množica dveh razredov	3
2.1	osnovna naloga	3
2.2	dodatna naloga	5
3	Učna množica Iris	6
3.1	Drugi načrt za razvrščevalnik	6
3.2	Prvi načrt za razvrščevalnik	8
4	Povzetek	9

1 Uvod

V poročilu bom predstavil razvoj in izvedbo programa, ki izvaja učenje linearne ločilne meje z uporabo postopka perceptrona s stalnim prirastkom.

V osnovnem delu projekta sem se osredotočil na implementacijo perceptrona za učenje linearne ločilne meje med dvema razredoma. Ta postopek sem preizkusil z uporabo učne množice dveh razredov, ki vsebujeta po dva vzorca z dvema značilkama. Program sem zasnoval tako, da obravnava vsak vzorec iz učne množice tudi kot testni vzorec, s čimer sem ocenil učinkovitost naučene ločilne meje.

V dodatnem delu projekta sem program nadgradil, da omogoča določanje ločilnih mej med pari razredov vzorcev. Ta postopek sem testiral na podatkovni zbirki Iris, ki je razvrščena v tri razrede. Kjer je prvi razred ("setosa") linearno ločljiv od preostalih dveh ("versicolor" in "virginica"), medtem ko slednja dva nista linearno ločljiva.

Učni postopek sem izvedel na dveh tretjinah vzorcev vseh razredov iz zbirke Iris, preostalo tretjino pa sem uporabil za preizkus razvrščanja. Preizkusil sem tri različne načrte razvrščevalnikov za trirazredni problem v tem poročilu bom podal analizo uspešnosti prvega in drugega načrta, ker tretjega nisem uspel pravilno adaptirati.

2 Učna množica dveh razredov

Učna množica je sestavljena iz dveh razredov U_1 in U_2 in vsak od njiju vsebuje po 2 vzorca z dvema značilkama.

2.1 osnovna naloga

Priprava podatkov:

Funkcija `prepare_data` je zasnovana za pripravo podatkov pred učnim procesom. Vzorca X in njihove pripadajoče oznake Y se preoblikujejo tako, da vsakemu vzorcu dodamo dodatno dimenzijo z vrednostjo 1. To omogoča perceptronu, da prilagodi odmik ločilne meje od izhodišča. Če je oznaka vzorca enaka 1 (kar označuje razred U_2), in če je parameter `negate_U2` nastavljen na `True`, se vrednosti vzorca negirajo.

To je korak, ki omogoča perceptronu, da učinkovito ločuje dva razreda.

Treniranje perceptrona:

Funkcija `perceptron_train` izvaja učenje perceptrona. Začetne uteži so nastavljene na $[-1, 0, 0]$, kar predstavlja začetno hipotezo ločilne meje. V vsaki iteraciji (epoch) algoritem preverja vsak vzorec učne množice in, če je klasifikacija nepravilna (to je, če je rezultat skalarnega produkta uteži in vzorca manjši ali enak 0), posodobi uteži s seštevanjem trenutnega vzorca. Ta postopek se ponovi za določeno število iteracij, kar v našem primeru znaša 4. Posodabljanje uteži v tej zanki je ključni del perceptronskega algoritma, saj omogoča prilagajanje ločilne meje, da se pravilno ujema z distribucijo podatkov. Koda se nahaja v datoteki `Osnovna_naloga_Šolska_učna_množica.py`.

Testiranje perceptrona:

Za testiranje uspešnosti naučenega perceptrona sem uporabil funkcijo `perceptron_test`. Ta funkcija sprejme uteži, pridobljene z učenjem, in testni vzorec. S skalarnim produktom uteži in testnega vzorca določi, ali vzorec pripada razredu U1 ali U2. Rezultat testiranja je predstavljen kot binarna vrednost, kjer 0 pomeni razred U1 in 1 pomeni razred U2.

V praktičnem primeru sem kot učno množico uporabil preprost nabor vzorcev X in njihove oznake Y . Za oceno uspešnosti modela sem uporabil metodo Leave-One-Out validacije. Ta metoda vključuje odstranitev enega vzorca iz učne množice, treniranje perceptrona na preostalih vzorcih in nato testiranje uspešnosti na odstranjenem vzorcu. Ta postopek se ponovi za vsak vzorec v množici. Tako sem dobil natančno oceno, kako dobro model generalizira na novih, nevidenih podatkih.

V vsakem testu sem izpisal dejanski razred vzorca (Y_{test}), napovedani razred (Y_{pred}) in trenutne uteži perceptrona ($weights$). Ta izpis mi je omogočil natančen vpogled v delovanje in odločitve, ki jih perceptron sprejema v procesu učenja in testiranja.

```
Test 1: Pravi razred: 0, Napovedani razred: 1, Uteži: [-2 1 0]
Test 2: Pravi razred: 0, Napovedani razred: 0, Uteži: [-2 0 1]
Test 3: Pravi razred: 1, Napovedani razred: 1, Uteži: [-3 0 2]
Test 4: Pravi razred: 1, Napovedani razred: 1, Uteži: [-2 0 1]
```

Figure 1: Rezultati razvrščevalnika

Iz teh rezultatov je razvidno, da je perceptronski model v treh od štirih testnih primerov uspešno ločil razrede. Samo v prvem testu je prišlo do napačne klasifikacije. Ta izid lahko kaže na to, da je model na splošno učinkovit, vendar morda potrebuje prilagoditev za boljše rezultate, kljub temu da je natančnost 75%.

2.2 dodatna naloga

V nadaljevanju poročila opisujem nadgradnjo kode, ki sem jo izvedel za reševanje dodatne naloge. Ta nadgradnja vključuje implementacijo funkcionalnosti za treniranje in testiranje perceptrona na način, ki omogoča ločevanje med več kot dvema razredoma. Ta pristop je zlasti uporaben pri obravnavanju podatkovne zbirke Iris, kjer imamo opravka s tremi razredi. Koda se nahaja v datoteki `Dodatna_naloga_Šolska_učna_množica.py`.

Usposabljanje perceptronov za pare razredov:

Ključna novost v tej nadgradnji je funkcija `train_perceptrons`, ki omogoča usposabljanje več perceptronov, vsakega za par razredov. Vsak perceptron se nauči ločiti med dvema specifičnima razredoma. V našem primeru, ker imamo tri razrede, to pomeni usposabljanje treh različnih perceptronov: enega za ločevanje med prvim in drugim razredom, drugega za ločevanje med prvim in tretjim razredom ter tretjega za ločevanje med drugim in tretjim razredom.

Vsak perceptron se trenira na podmnožici učne množice, ki vsebuje samo primere dveh razredov. Po usposabljanju se uteži vsakega perceptrona shranijo skupaj z oznakami razredov, ki jih ločuje.

Testiranje perceptronov:

Funkcija `test_perceptrons` implementira testiranje za scenarij večrazredne klasifikacije. Za vsak testni vzorec se uporabi vsak perceptron, da se napove, kateremu od dveh razredov, ki jih perceptron ločuje, vzorec pripada. Vsak razred prejme glas

od perceptrona, ki ločuje ta razred od drugega. Na koncu testiranja se izbere razred z največ glasovi kot končna napoved za dani vzorec.

Izvajanje testiranja:

Nadgradnja vključuje tudi izvedbo testiranja na učni množici, kjer vsak vzorec iz množice služi kot testni vzorec. Izhajajoči so rezultati vsakega testa, vključno s pravim razredom vzorca, napovedanim razredom in utežmi, ki so bile uporabljene za to napoved. Ta pristop omogoča oceno, kako dobro so perceptroni uspeli naučiti ločilne meje med razredi in kako uspešno lahko klasificirajo nove vzorce.

```
Test 1: Pravi razred: 0, Napovedani razred: 0, Uteži: [-2  0  1]
Test 2: Pravi razred: 0, Napovedani razred: 0, Uteži: [-2  0  1]
Test 3: Pravi razred: 1, Napovedani razred: 1, Uteži: [-2  0  1]
Test 4: Pravi razred: 1, Napovedani razred: 1, Uteži: [-2  0  1]
```

Figure 2: Rezultati razvrščevalnika

V vseh štirih testnih primerih je bila napoved perceptrona skladna s pravimi razredi, kar pomeni, da je model dosegel 100% natančnost na tej testni množici. Enakost uteži pri vseh testih kaže na stabilnost modela in njegovo sposobnost doslednega ločevanja med razredoma.

3 Učna množica Iris

Učna množica Iris je sestavljena iz treh razredov ("setosa", "versicolor" in "virginica") in 150 vzorcev, kjer ima vsak vzorec štiri botaničnih meritev rož perunika.

3.1 Drugi načrt za razvrščevalnik

V tej sekciji poročila opisujem nadaljnjo nadgradnjo kode, ki je namenjena obdelavi in klasifikaciji podatkovne zbirke Iris z uporabo drugega načrta razvrščevalnika. Ta del kode predstavlja praktično implementacijo algoritma perceptrona, prilagojenega za delo z resničnimi podatki in večrazredno klasifikacijo. Koda je prikazana v datoteki `Dodatna_Iris_2.py`.

Nalaganje in priprava podatkov:

Začel sem z nalaganjem podatkov iz zbirke Iris, ki so shranjeni v datoteki `iris.data`. Funkcija `load_iris_data` bere podatke, razčleni vrstice in jih pretvori v format, primeren za nadaljnjo obdelavo. Značilke (X) so shranjene kot plavajoča števila, medtem ko so oznake razredov (Y) kodirane z uporabo `LabelEncoder` za pretvorbo v numerične vrednosti.

Standardizacija podatkov:

Podatki so nato standardizirani s pomočjo `StandardScaler` iz knjižnice `scikit-learn`. Standardizacija normalizira obseg vrednosti značilk in izboljša uspešnost učenja.

Treniranje perceptronov za pare razredov:


Funkcija `train_perceptrons_for_pairs` implementira logiko usposabljanja več perceptronov, pri čemer vsak perceptron ločuje med parom razredov. Vsak perceptron se nauči ločiti med dvema specifičnima razredoma, pri čemer se uporabljajo samo podatki, ki pripadajo tem dvema razredoma.

Klasifikacija in evalvacija:

Za klasifikacijo neznanih vzorcev uporabljam funkcijo `classify_with_pairs`, ki za vsak testni vzorec uporabi vse naučene perceptrone in zbira glasove za vsak razred. Razred, ki prejme največ glasov, je izbran kot napovedan razred za dani vzorec.

Podatki so razdeljeni na učno in testno množico, pri čemer tretjina podatkov služi kot testni nabor. Po usposabljanju perceptronov na učni množici so ti uporabljeni za klasifikacijo testne množice. Uspešnost modela je ocenjena z izračunom natančnosti, ki meri delež pravilno klasificiranih vzorcev v testni množici.

Koda implementira večrazredno klasifikacijo s perceptroni. Vsak perceptron se nauči ločevati med dvema razredoma, nato pa se uporabi glasovanje za končno odločitev o razredu novega vzorca.



Natančnost: 92.0%

Figure 3: Rezultati razvrščevalnika

Natančnost 92.0% pomeni, da je model pravilno klasificiral 92% vzorcev v testni

množici. To kaže na to, da je model zmožen učinkovito ločiti med različnimi razredi Iris na podlagi njihovih botaničnih značilnosti, kot so dolžina in širina cvetnih listov ter čašnih listov.

3.2 Prvi načrt za razvrščevalnik

V tej sekciji poročila opisujem implementacijo in uporabo prvega načina razvrščevalnika z uporabo perceptrona za klasifikacijo podatkovne zbirke Iris. Ta pristop se osredotoča na treniranje ločenega perceptrona za vsak razred, kar predstavlja drugačen pristop od metode opisane v prejšnji sekciji. Koda je prikazana v datoteki `Dodatna_Iris_1.py`.

Nalaganje in priprava podatkov:

Podobno kot prej, se začne s procesom nalaganja in obdelave podatkov iz datoteke `iris.data`. Uporabljeni so standardni koraki za branje, razčlenjevanje in kodiranje podatkov z uporabo `LabelEncoder`. Poleg tega so podatki standardizirani z `StandardScaler`, kar je ključnega pomena za učinkovito učenje in klasifikacijo.

Treniranje perceptrona za posamezen razred:

Za vsak razred v podatkovni zbirki Iris se posebej trenira perceptron. To se izvede z uporabo funkcije `train_perceptron_for_class`, ki ustvari binarno oznako za ciljni razred (1 za ciljni razred, 0 za vse ostale) in nato trenira perceptron, da loči ta ciljni razred od ostalih. Vsak perceptron se nauči prepoznati, ali dani vzorec pripada njegovemu ciljnemu razredu ali ne.


Treniranje perceptronov za vsak razred:

V funkciji `train_perceptrons_for_each_class` se izvede zgoraj opisani postopek za vsak razred posebej, s čimer se ustvari zbirka perceptronov - vsak specializiran za prepoznavanje enega razreda.

Klasifikacija in evalvacija:

Pri klasifikaciji se za vsak testni vzorec izračuna rezultat vsakega perceptrona. Perceptron, ki vrne najvišji rezultat, določi napovedani razred vzorca.

Končno, uspešnost modela se oceni z izračunom natančnosti klasifikacije na testni množici. Ta metrika ponuja vpogled v to, kako dobro model lahko generalizira in pravilno klasificira nove, nevidene vzorce.



Natančnost: 40.0%

Figure 4: Rezultati razvrščevalnika

Nizka natančnost 40.0% kaže na potrebo po nadaljnjem razvoju in izboljšanju modela. To bi lahko vključevalo uporabo bolj sofisticiranih algoritmov strojnega učenja, prilagajanje parametrov modela ali uporabo več značilnosti za boljše razlikovanje med razredi. Prav tako bi bilo koristno podrobneje analizirati, kje in zakaj model napoveduje napačno in ga popraviti.

4 Povzetek

V prvem delu sem predstavil perceptronski model s preprosto podatkovno zbirko, ki vključuje dva razreda, vsak z dvema vzorcema. Najprej sem uporabil perceptron s konstantnim prirastkom med razredoma, kasneje pa sem program nadgradil z ločilno mejo za vsak par dveh razredov vzorcev in izvedel klasifikacijo vzorcev na osnovi predznaka funkcije ločilne meje.

V drugem delu sem predstavil in analiziral učinkovitost perceptronskega modela pri klasifikaciji podatkovne zbirke Iris, ki vključuje tri različne razrede cvetov: Setosa, Versicolor in Virginica. Preizkušena sta bila dva različna načrta za razvrščevalnik z odločitvenimi funkcijami. Prvi načrt je pokazal omejeno uspešnost pri ločevanju razredov, zlasti pri težavah z razlikovanjem med podobnimi razredi, kot sta Versicolor in Virginica. Drugi načrt je dosegel najvišjo natančnost klasifikacije pri ločevanju med posameznimi pari razredov. Pri tretjem načrtu mi pa ni uspelo pravilno napisati kode in ga zato nisem vključil v poročilo.

Iz rezultatov je razvidno, da je bil drugi način najučinkovitejši pri klasifikaciji podatkovne zbirke Iris. Ta metoda je omogočila boljšo prilagoditev ločilnih mej za posamezne pare razredov, kar je privedlo do visoke natančnosti pri klasifikaciji. Nasprotno pa je prvi način pokazal manjšo učinkovitost, kar nakazuje na njegovo omejitev pri obvladovanju kompleksnosti in podobnosti med različnimi razredi v zbirki Iris.