Fazi neuronska mreža

Seminarski rad u okviru kursa
Računarska inteligencija
Matematički fakultet

Katarina Savičić, Bojana Ristanović
mi16261@alas.matf.bg.ac.rs, mi16045@alas.matf.bg.ac.rs

23. septembar 2020

Sažetak

U ovom radu će biti predstavljena fazi neuronska mreža. Konkretnije, ovaj metod je korišćen kao metod klasifikacije podataka koji u trening fazi koristi min-max algoritam, a u test fazi koristi k najbližih suseda. Nakon opisa algoritma, prikazana je njegova praktična primena. Na kraju, rezultati su upoređeni sa autorima rada *Understanding Fuzzy Neural Network using code and animation* kao i sa običnim algoritmom KNN (k najbližih suseda).

Ključne reči: fazi logika, neuronske mreže, min-max, knn, klasifikacija.

Sadržaj

1 Uvod

Fazi neuronska mreža (eng. *Fuzzy neural network*) je sistem za učenje koji pronalazi parametre fazi sistema koristeći tehnike aproksimacije neuronskih mreža. To je hibridni inteligentni sistem koji kombinuje tehnike rasuđivanja fazi logike sa tehnikama učenja neuronskih mreža.[3]

Fazi min-max klasfikator (eng. *The fuzzy min–max (FMM)*) je sistem koji formira hiperboksove za klasifikaciju i predviđanje. U ovom radu je pokušana modifikacija FMM-a korišćenjem algoritma k najbližih suseda (eng. *k-nearest neighbors algorithm (k-NN)*) u procesu predviđanja klasa prosleđenih podataka.

2 Fazi logika

Kod klasične logike, promenljive mogu uzeti jedino vrednosti 1 ili 0 (tačno ili netačno). Kod fazi logike se skup vrednosti proširuje, pri čemu se dozvoljava da promenljive mogu uzeti realne vrednosti unutar nekog intervala. Drugim rečima, pretpostavlja se da ne mora sve biti u potpunosti istinito ili u potpunosti neistinito, već se dozvoljava određen nivo delimične istine. Na primer, za nešto možemo reći da je najverovatnije istina, pa odgovarajuća promenljiva može imati vrednost 0.9, a u nešto drugo možemo imati veliku sumnju, ali ostavljati malu verovatnoću da bude istinito, pa odgovarajuća logička promenljiva tada može imati vrednost 0.1. [2]

Da li nešto pripada skupu možemo odrediti pomoću funkcije pripadnosti. Ona kod klasične logike ima vrednost 0 ili 1, zato što nešto može ili ne da pripada skupu. Kod fazi logike ova funkcija ima vrednost između 0 i 1. [2]

3 Neuronske Mreže

U istraživanju podataka veštačke neuronske mreže su familija statističkih modela učenja inspirisana biološkim neuronskim mrežama. Koriste se u svrhu aproksimacije funkcija koje mogu zavisiti od velike količine ulaznih podataka, a koje su u principu nepoznate. Veštačke neuronske mreže su sistemi međusobno povezanih neurona, koji šalju poruke jedni drugima. Veze između ovih neurona imaju numeričke težine koje mogu biti podložne promenama u zavisnosti od iskustva, što neuronske mreže čini adaptivnim i sposobnim za učenje. Osnovna ideja veštačke neuronske mreže je simulacija velike količine gusto napakovanih, međusobno povezanih nervih ćelija u okviru računara, tako da je omogućeno učenje pojmova, prepoznavanje šablona i donošenje odluka na način koji je sličan čovekovom. Neuronska mreža se ne programira eksplicitno da uči, ona to radi sama, isto kao i mozak.[5]

Tipična neuronska mreža ima od nekolicine do stotinu, hiljadu, ili pak milion veštačkih neurona tj. jedinica, umreženih u serije slojeva, gde je svaki neuron povezan sa oba sloja sa obe strane. Neki od njih, koji se nalaze u početnom sloju, tj. sloju ulaza, dizajnirani su tako da dobijaju različite oblike informacija iz spoljašnjeg sveta. Jedinice koje se nalaze na suprotnoj strani mreže, u krajnjem sloju, tj. sloju izlaza, signaliziraju način na koji mreža reaguje na naučene informacije. Između leži jedan ili više skrivenih slojeva, koji zajedno čine većinu veštačkog mozga.[5]

4 K najbližih suseda

Algoritam k najbližih suseda je algoritam otkrivanja zakonitosti u podacima koji se koristi za probleme predviđanja ishoda (bilo da je numerička ili nenumerička vrednost), a na osnovu sličnosti slučaja za koji treba doneti odluku sa slučajevima iz tabele slučajeva (baze znanja). Kada je potrebno doneti odluku posmatra se k najbližih (najsličnijih) suseda i donosi se ona odluka koja se najčešže pojavljivala kod posmatranih k najbližih suseda.

Izbor parametra k je izuzetno važan i kompleksan korak. Ukoliko izaberemo malu vrednost k, onda možemo doći do situacije da nam šum u podacima (npr. pogrešno doneta odluka) utiče na dalje odluke. Ako izaberemo preveliko k, onda nam slučajevi koji nisu slični slučaju za koji se predviđa ishod utiču na odluku.[1]

5 Fazi min-max klasifikator (FMM)

Osnovna ideja je da se na osnovu funkcije pripadnosti koja se koristi u Fazi logici može odrediti kojoj klasi pripada instanca koju klasifikujemo. U ovom slučaju instanca je uređeni par realnih brojeva, koji se može predstaviti kao tačka u koordinatnom sistemu. Klasa instance će biti ona za koju funkcija pripadnosti daje najveću vrednost. Postavlja se pitanje kako predstaviti fazi skup?[4]

5.1 Hiperboks (eng. *Hyperbox*)

Fazi skup možemo definisati kao pravougaonik čije dužine stranica mogu uzimati vrednost od 0 do 1. Ovaj skup je takav da ako se tačka nalazi unutar pravouganika, vrednost njene funkcije pripadnosti je 1. Pravougaonik je definisan svojom najmanjom (donjom levom) i najvećom (gornjom desnom) tačkom. Te tačke ćemo označiti redom slovima v i w.

Vrednost funkcije pripadnosti za svaku tačku koja je van pravougaonika se smanjuje sa povećanjem udaljenosti tačke od pravougaonika. Formula za računanje funkcije pripadnosti nekom pravougaoniku, tj. fazi skupu, je data u nastavku:

$$b_j(A_h) = \frac{1}{2n}\sum_{i=1}^{n}[max(0, 1-max(0, \gamma min(1, a_{hi}-w_{ji}))) + max(0, 1-max(0, \gamma min(1, v_{ji}-a_{hi})))] \tag{1}$$

A_h obrazac za koji računamo funkciju pripadnosti, b_j fazi skup a n dimenzija (u našem slučaju 2). γ je hiperparametar koji se naziva osetljivost ili stopa smanjenja i može se koristiti za kontrolisanje vrednosti funkcije pripadnosti.

Kod neuronske mreže imamo prvi sloj koji predstavlja ulazne čvorove za podatke. Sledeći sloj su čvorovi koji predstavljaju hiperbokseve, tj. svaki čvor predtavlja jedan hiperboks koji pripada nekoj klasi i definisan je dvema tačkama. Hiperboks je prethodno pomenuti pravougaonik. Poslednji sloj čvorova predstavlja izlaz, odnosno klasu. U poslednjem sloju može biti više čvorova, u zavisnosti od broja postojećih klasa. Kada gledamo vrednost klase, svaki čvor koji predtavlja klasu definisan je njavećom vrednošću funkcije pripadnosti od svih funkcija pripadnosti računate za sve hiperbokseve koje pripadaju datoj klasi. Dakle, svaka klasa može imati jedan ili više hiperbokseva, vrednost u čovoru je maksimalna vrednost funkcije pripadnosti razmatranog podatka tim hiperboksevima. Na kraju se porede vrednosti u svim čvorovima koji predtavljaju klase i bira se klasa koja ima najveću vrdnost. [4]

5.2 Algoritam

Kako bismo sproveli pomenuti način klasifikacije, potrebno je da definišemo hiperbokseve za sve klase. To ćemo uraditi na osnovu skupa za treniranje, odnosno istreniraćemo klasifikator tako što ćemo napraviti hiperbokseve za postojeće klase. Algoritam se sastoji iz dve faze: proširenja i kontrakcije. [4]

5.2.1 Faza proširenja

Imamo instancu X_h iz trening skupa, koja pripada klasi Y. Prvo proverimo sve hiperbokseve koji pripadaju klasi Y i računamo funkciju pripadnosti za svaki. Hiperboks sa najvećom funkcijom pripadnosti je najpogodniji za proširenje. Neka B_j bude jedan takav hiperboks. Pre nego što ga proširimo računamo kriterijum ekspaztije dat sledećom formulom:

$$n\theta >= \sum_{i=1}^{n}(max(w_{ji}, x_{hi}) - min(v_{ji}, x_{hi})) \tag{2}$$

gde je θ hiper parametar koji predstavlja kriterjum proširenja i kontroliše maksimalno proširenje dozvoljeno za hiperboks. Ako zadovoljava kriterijum, hiperboks se širi na sledeći način:

$$v_{ji}^{new} = min(v_{ji}^{old}, x_{hi}) w_{ji}^{new} = min(w_{ji}^{old}, x_{hi}) \tag{3}$$

gde su v i w minimalna i maksimalna tačka hiperboksa. Hiperboks se čiri tako da je X_h uključena u region.

Ukoliko kriterijum širenja nije zadovoljen, pravi se novi hiperboks za klasu Y, takav da su mu minimalna i maksimalna tačka jednake i imaju vrednost X_h. Kada proširimo hiperboks, može doći do preklapanja sa drugim hiperboksom. Ovo nije toliko problem ako je u pitanju hiperboks iz iste klase, ali jeste ukoliko je u pitanju hiperboks koji pripada nekoj drugoj kasi. Zbog toga imamo fazu kontrakcije.[4]

5.2.2 Faza kontrakcije

Pronašli smo hiperboks koji smo proširili i sada treba da proverimo da li postoji preklapanje sa drugim hiperboksom. Dozvoljeno je preklapanje izmedju iperbokseva iste klase, pa treba samo da proverimo za hiperbokseve drugih klasa.

Proveru da li postoji preklapnje vršimo preko tačaka koje odredjuju hiperbokseve. Gledamo po svim dimenzijama da li postoji preklapanje i beležimo rezultate. Onda tražimo dimenziju po kojoj je preklapanje najmanje. Slučajevi koji pokrivaju proveru proveru preklapanja su dati u kodu. Kada nađemo dimenziju po kojoj je preklapanje minimalo, hoćemo da smanjimo hiperboks po toj dimenziji. Slučajevi koji pokrivaju način smanjenja hiperboksa takođe se nalaze u kodu i neće biti predstavljeni ovde.

Poenta celog algoritma je naći odgovarajući hiperboks, proširiti ga ako je to moguće, ako ne naći drugi koji je moguće proširiti, ili dodati novi, proveriti da li postoji preklapanje i izvršiti kontrakciju ukoliko postoji. [4]

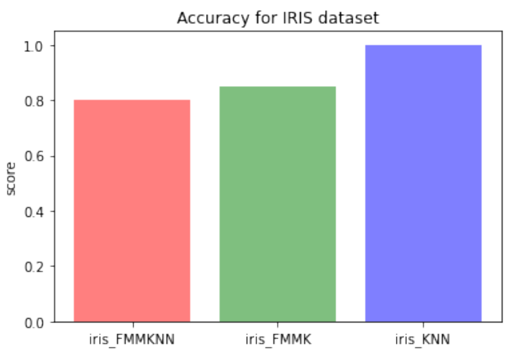
5.3 Modifikacije algoritma

U originalnom algoritmu se za određivanje klase uzima samo najveća vrednost klase, koja predstavlja najveću vrednost funkcije pripadnosti za hiperbokseve te klase. Mi smo algoritam izmenili tako da ne gleda jednu najveću vrednost već uzima u obzir nekoliko vrednosti. Umesto da gledamo funkcije pripadnosti i tražimo najveće vrdnosti, gledali smo rastojanje svake tačke od hiperboksa i tražili najmanje rastojanje. Pretpostavili smo da rastojanje od hiperboksa možemo posmatrati kao rastojanje od prave koja prolazi kroz tačke koje određuju hiperboks. Sortirali smo rastojanja od svih hiperbokseva rastuće i primenili metod k najbližih suseda. Metod smo primenili tako što smo uzeli k najmanjih rastojanja, gledali kojim klasama pripadaju hiperboksevi i kao klasu instance uzeli najbrojniju klasu među hiperboksevima.

6 Rezultati

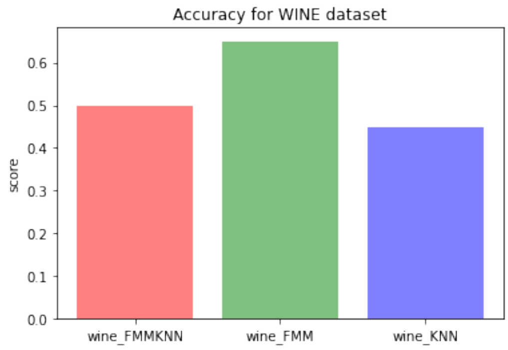
Za testiranje rezultata korišćena su dva skupa podataka. Prvi je iris skup koji se sastoji od 3 vrste cveta Iris - Setosa, Versicolour, Virginica. Sadrži 150 instanci i ima 5 atributa koji predstavljaju dužinu i širinu latica (eng. *petal length, petal width*), dužinu čašičnog listića (eng. *sepal length, sepal width*) i vrstu kojoj cvet pripada (eng. *species*). Pošto je algoritam napisan tako da radi za dvodimenzionalne ulazne podatke, izabrali smo 2 atributa: dužinu latica i dužinu čašičnog lista. Podaci su normalizoavni tako da uzimaju vrednosti od 0 do 1.

Pokrenut je originalni algoritam sa kriterijumom proširenja 0.4, modifikovani algoritam sa istim kriterijumom i 4 najbliža suseda kao i algoritam KNN za 4 najbliža suseda. Na slici 1 vidimo da su najbolji rezltati dobije algoritmom KNN. Rezultati prva dva algoritma se mogu promeniti sa promenom kriterijuma proširenja. Treba biti oprezan sa postavljanjem ovog kriterijuma jer on utiče na veličinu i na broj hiperbokseva, nije poželjno da imamo mnogo malih, kao ni malo velikih hiperbokseva.



Slika 1: Preciznost na podacima skupa Iris

Wine skup podataka je takođe provučen kroz sva tri algoritma radi poređenja rezultata. Na slici 2 možemo videti da algoritam FMM daje najbolje rezultate. Modifikovani KNN algoritam je pogodio klasu za 50% instanci, međutim preciznost ovog algoritma dosta zavisi od prosleđenih parametara tj. od kriterijuma kontrakcije i od broj suseda koji se razmatraju.



Slika 2: Preciznost na podacima skupa Iris

7 Zaključak

Hiper parametar θ se može menjati kako bi se poboljšali rezultati. Ako se koristi visoka vrednost za θ tada se povećava maksimalna dozvoljena veličina hiperboksa, a to povećava šansu da u tom regionu padne pogrešan uzorak. Sa druge strane, ako se uzme premala vrednost za θ, broj hiperbokseva će se povećavati i preprilagodiće nove tačke. Na ovaj način klasifikator gubi sposobnost dobrog generalizovanja i daje loše performanse. Sa ovom vrednošću treba eksperimentisati kao bi se pronašla vrednost koja daje najbolje performanse.[4] Jedno od značajnih poboljšanja algoritma bi bilo njegovo proširenje da radi na više dimenzija.

Literatura

[1] Algoritam K najbližih suseda. on-line at: http://odlucivanje.fon.bg.ac.rs/wp-content/uploads/kNN-v1.1.pdf.

[2] Fazi logika. on-line at: http://poincare.matf.bg.ac.rs/~stefan/ri/3.html.

[3] Fuzzy Neural Network. on-line at: http://www.scholarpedia.org/article/Fuzzy_neural_network.

[4] Min-Max Fuzzy Classifier. on-line at: https://medium.com/@apbetahouse45/understanding-fuzzy-neural-network-with-code-and-graphs-263d1091d773.

[5] Sava Gavran. VEŠTAČKE NEURONSKE MREŽE U ISTRAŽIVANJU PODATAKA: PREGLED I PRIMENA. Master's thesis, Univerzitet u Beogradu, Matematički fakultet, 2016.

A Kod

```python
class FuzzyKNN:
    def __init__(self, sensitivity, exp_bound):
        self.sensitivity = sensitivity
        self.hyperboxes = None
        self.classes = np.array([])
        self.exp_bound = exp_bound

    def membership(self, pattern):
        # racuna pripadnost i vraca niz pripadnosti svakom hiperboksu
        min_pts = self.hyperboxes[:, 0, :]
        max_pts = self.hyperboxes[:, 1, :]

        a = np.maximum(0, (1 - np.maximum(0, (self.sensitivity * np.minimum(1, pattern - max_pts)))))
        b = np.maximum(0, (1 - np.maximum(0, (self.sensitivity * np.minimum(1, min_pts - pattern)))))

        return np.sum(a + b, axis=1) / (2 * len(pattern))

    def overlap_contract(self, index):
        #proveravamo da li se hiperboksevi preklapaju
        contracted = False
        for test_box in range(len(self.hyperboxes)):
            if self.classes[test_box] == self.classes[index]:
                # ignorisemo preklapanje hiperbokseva iste klase
                continue
            expanded_box = self.hyperboxes[index]
            box = self.hyperboxes[test_box]

            vj, wj = expanded_box #onaj za koji gledamo da li se preklapa sa nekim
            vk, wk = box

            # moguci slucajevi preklapanja
            # trazimo najmanje preklapanje
            delta_new = delta_old = 1
            min_overlap_index = -1
            for i in range(len(vj)):
                if vj[i] < vk[i] < wj[i] < wk[i]:
                    delta_new = min(delta_old, wj[i] - vk[i])
                elif vk[i] < vj[i] < wk[i] < wj[i]:
                    delta_new = min(delta_old, wk[i] - vj[i])

                elif vj[i] < vk[i] < wk[i] < wj[i]:
                    delta_new = min(delta_old, min(wj[i] - vk[i], wk[i] - vj[i]))

                elif vk[i] < vj[i] < wj[i] < wk[i]:
                    delta_new = min(delta_old, min(wj[i] - vk[i], wk[i] - vj[i]))

                if delta_old - delta_new > 0:
                    min_overlap_index = i
                    delta_old = delta_new

            # ako ima preklapanja,
            # gledamo po kojoj strani smanjujemo hiperbokseve
            if min_overlap_index >= 0:
                i = min_overlap_index
                if vj[i] < vk[i] < wj[i] < wk[i]:
                    vk[i] = wj[i] = (vk[i] + wj[i])/2

                elif vk[i] < vj[i] < wk[i] < wj[i]:
                    vj[i] = wk[i] = (vj[i] + wk[i])/2

                elif vj[i] < vk[i] < wk[i] < wj[i]:
                    if (wj[i] - vk[i]) > (wk[i] - vj[i]):
                        vj[i] = wk[i]

                    else:
                        wj[i] = vk[i]

                elif vk[i] < vj[i] < wj[i] < wk[i]:
                    if (wk[i] - vj[i]) > (wj[i] - vk[i]):
                        vk[i] = wj[i]

                    else:
                        wk[i] = vj[i]

                self.hyperboxes[test_box] = np.array([vk, wk])
                self.hyperboxes[index] = np.array([vj, wj])
                contracted = True

        return contracted
```

```
    def train_pattern(self, X, Y):
        # funkcija koja trenira klasifikator
        target = Y

        # ako nemamo tu klasu u klasama
        if target not in self.classes:
            # pravimo hiperboks
            if self.hyperboxes is not None:
                self.hyperboxes = np.vstack((self.hyperboxes, np.array([[X, X]])))
                self.classes = np.hstack((self.classes, np.array([target])))

            else:
                self.hyperboxes = np.array([[X, X]])
                self.classes = np.array([target])
        else:

            # sortiramo pripadnosti svim hiperboksevima za trazenu klasu
            memberships = self.membership(X)
            memberships[np.where(self.classes != target)] = 0
            memberships = sorted(list(enumerate(memberships)), key=lambda x: x[1], reverse=True)

            # Sirimo hiperboks
            count = 0
            while True:
                index = memberships[count][0]
                min_new = np.minimum(self.hyperboxes[index, 0, :], X)
                max_new = np.maximum(self.hyperboxes[index, 1, :], X)

                if self.exp_bound * len(np.unique(self.classes)) >= np.sum(max_new - min_new):
                    self.hyperboxes[index, 0] = min_new
                    self.hyperboxes[index, 1] = max_new
                    break
                else:
                    count += 1

                if count == len(memberships):
                    self.hyperboxes = np.vstack((self.hyperboxes, np.array([[X, X]])))
                    self.classes = np.hstack((self.classes, np.array([target])))
                    index = len(self.hyperboxes) - 1
                    break

            contracted = self.overlap_contract(index)

    def fit(self, X, Y):
        for x, y in zip(X, Y):
            self.train_pattern(x, y)


    # predvidjamo klasu
    def predict(self, X, k):

        #uzimamo tacke koje odredjuju hiperbokseve
        min_pts = self.hyperboxes[:, 0, :]
        max_pts = self.hyperboxes[:, 1, :]

        # broj klasa
        # i niz u kome cemo brojati pojavljivanje svake klase
        n_classes = len(np.unique(self.classes))
        cl = np.zeros(n_classes)

        # racunamo udaljenost tacke X od svakog hiperboksa
        # tako sto racunamo udaljenost X od prave koja prolazi kroz tacke koje odredjuju hiperboks
        distance = []
        for i in range(len(min_pts)):
            x1 = min_pts[i][0]
            y1 = min_pts[i][1]
            x2 = max_pts[i][0]
            y2 = max_pts[i][1]

            if(x1 == x2 and y1 == y2):
                d = abs(math.sqrt((x1-X[0])**2 + (y1 - X[1])**2))
                distance.append(d)
            elif(x1 == x2):
                d = min(abs(math.sqrt((x1-X[0])**2 + (y1 - X[1])**2)),
                       abs(math.sqrt((x2-X[0])**2 + (y2 - X[1])**2)))
                distance.append(d)
            elif(y1 == y2):
                d = min(abs(math.sqrt((x1-X[0])**2 + (y1 - X[1])**2)),
                       abs(math.sqrt((x2-X[0])**2 + (y2 - X[1])**2)))
                distance.append(d)
            else:
```

```python
                d = abs((y2-y1)*X[0] - (x2-x1)*X[1] + x2*y1 - y2*x1) / math.sqrt((y2-y1)**2 + (x2-x1)**2)
                distance.append(d)

        # sortiramo udaljenosti od najmanje ka najvecoj
        # i uzimamo prvih k najblizih hiperbokseva
        distance = sorted(list(enumerate(distance)), key=lambda x: x[1])
        distance = distance[:k]
        distance_index = []
        for i in range(len(distance)):
            distance_index.append(distance[i][0])

        # brojimo pojavljivanje svake klase na osnovu hiperboksa koji joj pripada
        for i in range(len(distance_index)):
            index = distance_index[i]
            _class = self.classes[index]
            cl[_class] += 1

        # nalazimo najbrojniju klasu koja je konacna klasa X
        max = 0
        final_class = 0
        for i in range (len(cl)):
            if(cl[i] >= max):
                max = cl[i]
                final_class = i


        return final_class

    # funkcija koja racuna procenat uspesno klasifikovanih instanci
    def score(self, X, Y, k):
        count = 0
        for x, y in zip(X, Y):
            pred = self.predict(x, k)
            if y == pred:
                count += 1
        print(count)
        print(len(Y))

        return count / len(Y)
```

Listing 1: Klasa koja predstavlja Fazi Min Max klasifikator koji koristi KNN