SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Izgradnja jakog klasifikatora boostanjem

Mateja Čuljak
Voditelj: *Doc.dr.sc. Siniša Šegvić*

Zagreb, svibanj 2011.

Sadržaj

1. Uvod

Strojno učenje je znanstvena disciplina koja se bavi razvojem algoritama koji računalima omogućavaju promjenu ponašanja i poboljšanje učinkovitosti na temelju iskustva, tj. empirijskih podataka. Cilj strojnog učenja je pronalaženje složenih uzoraka u podacima, te donošenje inteligentnih odluka baziranih na podacima. Pritom je problem u činjenici da je skup svih mogućih ponašanja računala na temelju svih mogućih ulaznih podataka prevelik u odnosu na dostupan skup ulaznih podataka, stoga računalo koje uči treba imati sposobnost generalizacije na osnovi dostupnih podataka. Korisnost strojnog učenja se nalazi u rješavanju problema kod kojih ne postoji dovoljno ljudsko znanje o procesu (npr. raspoznavanje govora), otkrivanju znanja u velikim skupovima podataka (engl. *data mining*) i sl.

Možemo ustanoviti sljedeću podjelu algoritama strojnog učenja:

Nadzirano učenje: Dostupan je skup (x, y) pri čemu su x ulazni podatci te y ciljna vrijednost. Cilj učenja je pronaći vezu h među njima, $y = h(x)$.

Nenadzirano učenje: Dostupni su samo podatci x te je potrebno pronaći pravilnosti u podatcima, npr. grupirati slične podatke.

Ojačano učenje: Cilj je naučiti koje ponašanje primijeniti na temelju poznatih podataka, tako da se maksimizira nagrada.

U ovom radu fokus je na algoritmima nadziranog učenja. Algoritme nadziranog učenja možemo podijeliti ovisno o tome je li razdioba ciljne vrijednosti diskretna ili kontinuirana. U prvom slučaju je to problem klasifikacije, a u drugom problem regresije. Klasifikacija se, dakle, općenito bavi problemom pronalaženja diskretnih klasa u koje možemo svrstati ulazne podatke, a odluka se donosi na temelju otprije klasificiranih podataka, tj. primjera za učenje. Vezu h između podataka i ciljnih vrijednosti klase nazivamo klasifikatorom.

Pronalaženje potpuno "savršenog" algoritma učenja je nemoguće, stoga se prirodno nazire ideja kombiniranja više jednostavnijih osnovnih algoritama učenja

kako bi se postigla veća učinkovitost klasifikacije. Moguće je kombinirati različite algoritme, različite parametre istog algoritma, različite ulazne podatke i sl. Osnovne algoritme pritom nazivamo slabima, jer za njih možemo garantirati samo da je uspješnost nešto veća od slučajnog pogađanja. Krajnji algoritam dobiven kombiniranjem slabih algoritama se naziva jakim, te treba imati visoku moć klasifikacije.

Možemo izdvojiti dvije vrste kombiniranja algoritama:

Nezavisni eksperti: Algoritmi su odvojeni i nezavisni jedan od drugoga. Primjeri takvih algoritama su glasanje, stacking, ECOC (engl. *error correcting output codes*), bagging, MoE (engl. *mixture of experts*) itd.

Višerazinska kombinacija: Algoritmi formiraju niz u kojem svaki algoritam sljedbenik ovisi o rezultatu njegovih prethodnika. Moguće je višerazinsko učenje (sljedbenik uči na podatcima koje je prethodnik krivo klasificirao) ili višerazinska primjena (sljedbenik donosi odluku kad prethodnik nije donio).

Primjer višerazinskog učenja je algoritam boostanja.

2. Opis algoritma boostanja

U algoritmu boostanja su osnovni algoritmi uvijek isti, a varira skup ulaznih podataka. Prvi algoritmi boostanja osmišljeni su u (Schapire, 1990) i (Freund, 1990).

2.1. Originalni postupak

U originalnom postupku boostanja, skup za učenje se dijeli na 3 podskupa: X_1, X_2 i X_3. Dostupna su 3 slaba algoritma: a_1, a_2 i a_3. Postupak se izvodi na sljedeći način:

1. Naučiti algoritam a_1 nad skupom X_1, te ga evaluirati nad skupom X_2.
2. Naučiti algoritam a_2 nad krivo klasificiranim primjerima iz skupa X_2 (njihov broj je N), te također nad N dobro klasificiranih primjera iz X_2.
3. Evaluirati algoritme a_1 i a_2 nad skupom X_3.
4. Za sve primjere iz X_3 za koje se odluke algoritama a_1 i a_2 ne podudaraju, naučiti algoritam a_3 (a_3 donosi presudu).

Nedostatak osnovnog algoritma je potreba za velikim skupom primjera za učenje.

2.2. AdaBoost

Algoritam AdaBoost predstavlja poboljšanje originalnog postupka. Predstavljen je u (Freund i Schapire, 1995).

AdaBoost algoritam obavlja funkciju *boostanja* kombiniranjem skupa slabih klasifikatora tako da tvore jaki klasifikator, tj. gradi jaki klasifikator kao linearnu kombinaciju $f(x) = \sum_{t=1}^{T} = w_t h_t(x)$ slabih klasifikatora $h_t(x)$. Slijedi algoritam za klasifikaciju u dva razreda, kako je opisano u (Alpaydin, 2004) :

– Uzmemo uzorke za učenje $(x_1, y_1), \ldots, (x_m, y_m)$ pri čemu je $\mathbf{x}_i \in X$ (skup za učenje), $y_i \in Y = \{-1, 1\}$ (razred uzorka).

– Inicijaliziramo vjerojatnosti odabira pojedinog uzorka $p_1(\mathbf{x}_i) = \frac{1}{m}$.

– Za $t = 1, \ldots, T$:

1. Uzorkuj skup X_t iz skupa X koristeći distribuciju p_t
2. Nauči slabi klasifikator $h_t : X_t \to \{-1, 1\}$ na skupu X_t
3. Nađi grešku hipoteze h_t:

$$\epsilon_t = \sum_i p_t(\mathbf{x}_i) y_i,$$

   za sve $y_i \neq h_t(\mathbf{x}_i)$ (loše klasificirani primjeri) iz skupa X_t!
4. Ako je $\epsilon_t > 1/2$ tada izađi iz petlje
5. Izračunaj:

$$\beta_t = \frac{\epsilon_t}{1 - \epsilon_t}$$

6. Prilagodi vjerojatnosti:[1] ako je i-ti primjer dobro klasificiran ($y_i = h_t(\mathbf{x}_i)$), tada $p_{t+1}(\mathbf{x}_i) = \beta_t p_t(\mathbf{x}_i)$. Inače, $p_{t+1}(\mathbf{x}_i) = p_t(\mathbf{x}_i)$.
7. Normaliziraj vjerojatnosti tako da je p_{t+1} distribucija vjerojatnosti:

$$p_{t+1}(i) = \frac{p_{t+1}(\mathbf{x}_i)}{\sum_i p_{t+1}(\mathbf{x}_i)}$$

– Rezultat boostanja je lista parova slabih klasifikatora (h_t) i koeficijenata β_t: $\{h_t, \beta_t\}_T$.

– Završni jaki klasifikator (njime se provodi klasifikacija):

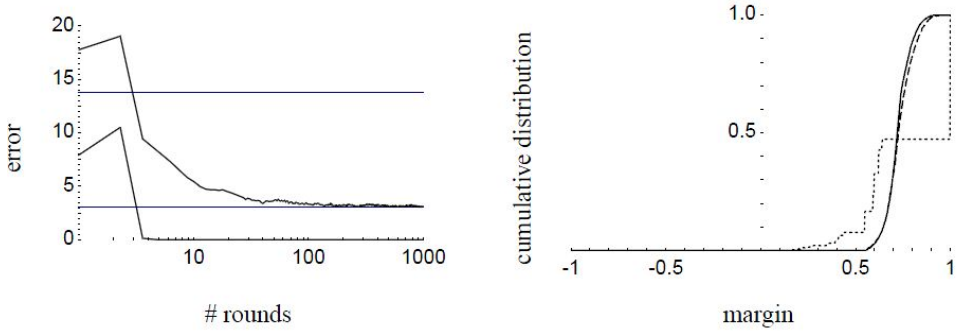
$$y = \sum_t \left(\log \frac{1}{\beta_t} \right) h_t(\mathbf{x})$$

Iz gornjeg algoritma možemo iščitati kako zapravo *boostamo* slabi klasifikator. Svaki put naglašavamo one primjere koji su bili neispravno klasificirani, tako da im održavamo konstantne težine, dok primjerima koji su ispravno klasificirani smanjujemo težine. Svaki klasifikator je forsiran da pokuša bolje klasificirati one primjere koje je njegov prethodnik loše klasificirao. Tako tvorimo jaki klasifikator.

[1] Prilagodba se radi samo nad uzorcima koji su odabrani u postupku uzorkovanja (engl. *boostraping*).

Broj grešaka takvog klasifikatora se eksponencijalno približava nuli s povećanjem broja primjera za učenje koje smo mu zadali, što se može vidjeti na slici 2.1.

Gornji algoritam moguće je modificirati i implementirati na nekoliko načina. Uzorkovanje skupa X_t iz osnovnog skupa primjera za učenje X se najčešće obavlja *bootstrap* algoritmom. Primjer takvog algoritma je dan u poglavlju 3. U četvrtom koraku algoritma moguće je, umjesto izlaska iz petlje i završetka boostanja, vratiti se na početak algoritma te ponovno provesti uzorkovanje i učenje. Naime, greška hipoteze ϵ_t je veća od 0.5 samo kad je skup za učenje nezgodno odabran. To se događa rijetko, no kako uzorkujemo slučajnim odabirom uz vjerojatnosti p_t, takve greške su ponekad moguće. Ponovnim uzorkovanjem se problem najčešće riješi već u idućoj iteraciji algoritma. Kod prilagođavanja vjerojatnosti, moguće je to učiniti tako da vjerojatnosti ispravno klasificiranih uzoraka ne ostaju konstantne, nego se povećavaju, ali pritom treba održati distribuciju jednakom (takav algoritam se nalazi u (Freund et al., 1999)). Razlika je samo u izračunu, a rezultati takvog algoritma nisu promijenjeni. Završni jaki klasifikator se dobija većinskim glasanjem T hipoteza h_t, pri čemu je težina svakog klasifikatora $w_t = \frac{1}{\log \beta_t}$. Ove težine pri glasanju su korištene tako da budu proporcionalne s rezultatima slabih klasifikatora.



Slika 2.1: Greška klasifikatora i margina u ovisnosti o broju slabih algoritama. Korišten je algoritam AdaBoost nad slabim klasifikatorima C4.5. *Lijevo:* Gornja krivulja predstavlja grešku generalizacije, a donja pogrešku učenja boostanog klasifikatora, kao funkcija broja slabih klasifikatora. Horizontalne linije predstavljaju greške učenja slabog klasifikatora i jakog klasifikatora. *Desno:* Distribucija margina primjera za učenje nakon 5 (linija kratkih crtica), 100 (linija dugih crtica) i 1000 (puna linija) iteracija algoritma boostanja.

2.2.1. Slabi klasifikatori

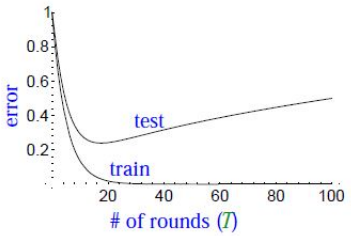
Za dobru uspješnost algoritma, potreban je dovoljno veliki skup ulaznih primjera, te dovoljno slabi i podesivi osnovni algoritmi. Slabi klasifikatori moraju biti nestabilni, tj. male promjene u skupu primjera za učenje moraju dovoditi do velikih promjena u naučenom klasifikatoru. Pristranost klasifikatora tako ostaje dovoljno mala, a varijancu smanjujemo boostanjem. Često korišteni osnovni algoritmi su stabla odluke dubine 1 (engl. *decision stumps*), C4.5, perceptroni, naivni Bayesov klasifikator, ostala stabla odluke itd. Stabilni algoritmi, kao npr. linearna diskriminantna analiza (LDA) i k-NN imaju malu varijancu i veliku pristranost te ne postižu dobre rezultate boostanjem. Kad bi kao slabi klasifikator koristili neki stabilni klasifikator, u svakom sljedećem koraku algoritma boostanja ne bi ostajalo dovoljno krivo klasificiranih primjera, već samo primjeri koji jako odskaču od ostalih, čime algoritam postaje osjetljiv na šumove.

2.2.2. Analiza pogreške učenja

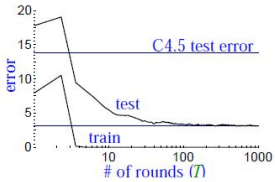
Osnovno svojstvo algoritma boostanja je reduciranje pogreške učenja. Ako je svaki slabi klasifikator tek malo bolji od slučajnog pogađanja, ukupna greška učenja jakog klasifikatora dobivenog boostanjem pada eksponencijalnom brzinom. Naime, pogrešku svakog slabog klasifikatora ϵ_t možemo zapisati kao $\frac{1}{2} - \gamma_t$, te γ_t tada mjeri za koliko je slabi klasifikator bolji od slučajnog pogađanja, koje ima pogrešku vrijednosti $\frac{1}{2}$. Tada je ukupna pogreška učenja boostanog jakog klasifikatora jednaka

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq exp\left(-2\sum_t \gamma_t^2\right),$$

što je pokazano u (Freund i Schapire, 1995). Ako je tada svaki slabi klasifikator t malo bolji od slučajnog pogađanja tako da je $\gamma_t \geq \gamma$ za neki $\gamma > 0$, pogreška učenja pada eksponencijalno. Prethodnici AdaBoost algoritma su zahtijevali poznavanje donje granice γ unaprijed. Taj zahtjev je bio težak za ispuniti budući da granicu γ nije lako ustanoviti unaprijed. Za razliku od prethodnika, AdaBoost je adaptivan algoritam – prilagođava se vrijednostima pogreški učenja svakog slabog klasifikatora. Zbog smanjenja greške učenja i greške generalizacije (objašnjeno u idućem pododjeljku) AdaBoost zaista gradi jake klasifikatore od slabih, tj. vrši algoritam boostanja.



(a) Očekivana greška generalizacije. Pri prevelikom broju koraka algoritma, događa se prenaučenost i greška generalizacije raste.



(boosting C4.5 on "letter" dataset)

(b) Greška generalizacije dobivena testiranjem boostanog C4.5 algoritma. Ne dolazi do prenaučenosti.

Slika 2.2: Očekivana greška generalizacije i stvarna greška generalizacije.

2.2.3. Analiza pogreške generalizacije

Autori AdaBoosta su u (Freund i Schapire, 1995) ustanovili da pogreška generalizacije ovisi o parametru $\sqrt{\frac{T \cdot d}{m}}$, pri čemu je T broj koraka boostanja, tj. broj slabih klasifikatora, m je veličina skupa za učenje, a d je VC-dimenzija prostora hipoteza (Vapnik i Chervonenkis, 1971). Dakle, može doći do prenaučenosti ako je broj slabih klasifikatora prevelik. Međutim, u ranijim eksperimentima je pokazano da često ipak ne dolazi do prenaučenosti. Naprotiv, boostanje često smanjuje pogrešku generalizacije čak i kad je pogreška učenja dosegla minimum, što je suprotno od očekivanog. Ta kontradikcija je prikazana na slici 2.2.

U (Schapire et al., 1998) je uveden novi pristup analizi pogreške generalizacije, definiranjem margine primjera za učenje. Margina primjera (x, y) se definira kao

$$\frac{y \sum_t \alpha_t h_t(x)}{\sum_t \alpha_t},$$

pri čemu je $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$. Vrijednost margine je pozitivna akko krajnja hipoteza jakog klasifikatora H dobro klasificira primjer x, te se može interpretirati kao mjera pouzdanosti hipoteze H.

Sada se gornja granica pogreške generalizacije može izraziti kao

$$p[margin(x, y) < \theta] + O\left(\frac{d}{m\theta^2}\right),$$

za neki $\theta > 0$, gdje je $p[\cdot]$ empirijska vjerojatnost. Greška generalizacije više ne ovisi o broju slabih klasifikatora T. Također je dokazano da veće margine dovode do bolje grešku generalizacije (tj. jaki klasifikator može biti puno manjih dimenzija), te da se algoritam boostanja koncentrira na primjere s najmanjim

marginama. Čak i nakon što greška učenja dosegne minimum, algoritam povećava marginu primjera što rezultira smanjenjem greške generalizacije.

2.2.4. Veza s SVM-om

Budući da algoritam AdaBoost optimira marginu između razreda, može se ustanoviti veza AdaBoosta sa strojevima potpornih vektora (engl. *support vector machines,* SVM). Strojevi potpornih vektora pokušavaju maksimizirati najmanje margine odvajanja među razredima. Međutim, postoje određene razlike. Boostanje koristi l_1 i l_∞ normu:

$$||\boldsymbol{\alpha}||_1 = \sum_t |\alpha_t|, \quad ||\mathbf{h}(\mathbf{x})||_\infty = \max_t |h_t(x)|,$$

dok SVM koristi l_2 normu:

$$||\boldsymbol{\alpha}||_2 = \sqrt{\sum_t \alpha_t^2}, \quad ||\mathbf{h}(x)||_2 = \sqrt{\sum_t h_t^2(x)}.$$

Osim toga, SVM koristi kvadratno programiranje, dok se AdaBoost može interpretirati kroz linearno programiranje (Freund i Schapire, 1996), pa su zahtjevi izračuna različiti.

2.3. Ostale varijante boostanja

Još neke novije varijante algoritma boostanja su Gentle Adaboost predložen u (Friedman et al., 2000) koji nešto manje stavlja naglasak na primjere koji jako odskaču od ostalih (engl. *outliers*) (ako ih ima puno, klasičan AdaBoost ne može dobro klasificirati, dok se s Gentle AdaBoostom taj slučaj dobro rješava); BrownBoost predložen u (Freund, 2001) koji je adaptivna verzija *boost-by-majority* algoritma te otvara veze između boostanja, Brownovog gibanja, i igranja ponovljenih igara (teorija igara). Također, varijante algoritma AdaBoost postoje i za višeklasnu klasifikaciju – AdaBoost.MH, AdaBoost.M2 i sl.

3. Opis implementacije

Implementacija je izvedena koristeći programski jezik *Python* te algoritam opisan u (Alpaydin, 2004). Implementirana je AdaBoost varijanta boostanja za dva razreda. Osnova implementacije je skripta `adaboost.py` (vidi: dodatak A). Implementacija se može upogoniti s proizvoljnim slabim klasifikatorom i skupom podataka. Za pokretanje potreban je Python interpreter verzije 2.7 (druge verzije nisu testirane, no pretpostavlja se da implementacija radi sa verzijama $[2.4, 3.0)$). Implementacija ne ovisi o vanjskim bibliotekama.

3.1. Programsko sučelje

3.1.1. Boostanje (učenje)

Za postupak boostanja (učenje) odgovorna je funkcija `adaboost()`:

```
def adaboost(dataset, steps, weakTrain, weakClassify).
```

Opis argumenata:

dataset: skup podataka za učenje – lista formata: $[(\mathbf{x}, y)^*]$ (lista uređenih parova $\mathbf{x}$ i y) pri čemu je $\mathbf{x}$ vektor značajki jednog primjera za učenje, a y oznaka pripadnosti razredu koja može poprimiti vrijednost 1 ili -1 (ako je $y = -1$ tada $\mathbf{x}$ pripada razredu s oznakom -1).

steps: broj slabih klasifikatora koje boostamo (broj koraka T).

weakTrain: funkcija koja stvara slabi klasifikator, tj. mora vrijediti:

```
classifier = weakTrain(dataset),
```

pri čemu je `classifier` instanca slabog klasifikatora izgrađena nad skupom `dataset`.

weakClassify: funkcija za klasifikaciju uporabom slabog klasifikatora, primjer:

```
classifier = weakTrain(dataset)
cls = weakClassify(x, classifier),
```

pri čemu je `x` uzorak kojeg je potrebno klasificirati, a `cls` razred kojem `x` pripada (sudeći po slabom klasifikatoru `classifier`).

Izlaz postupka boostanja je lista uređenih parova klasifikatora i pripadnih koeficijenata (β): `[(classifier, beta), ...]`.

3.1.2. Klasifikacija

Postupak klasifikacije se provodi funkcijom:

```
def classify(x, classifiers, weakClassify),
```

pri čemu je `x` uzorak kojeg je potrebno klasificirati, `classifiers` izlaz koji je dao postupak boostanja, a `weakClassify` funkcija za klasifikaciju slabim klasifikatorom. Primjer povezivanja postupka boostanja i klasifikacije:

```
classifiers = adaboost(dataset, 20, weakTrain, weakClassify)
cls = classify(x, classifiers, weakClassify)
```

Izlaz klasifikacije je oznaka razreda kojem pripada uzorak `x`, tj. vrijednost 1 ili -1.

3.2. Implementacija slabog klasifikatora

Slabi klasifikator kompatibilan sa navedenom implementacijom mora implementirati funkcije:

```
classifier = weakTrain(dataset)
cls = weakClassify(x, classifier)
```

Za potrebe testiranja i evaluacije implementiran je slabi klasifikator *decision stump*. Radi se o *stablu odluke* (engl. *decision tree*) dubine 1. Implementacija se nalazi u skripti `treestump.py`.

3.3. Uzorkovanje skupa podataka

AdaBoost algoritam se oslanja na dobro variranje skupa podataka za učenje koje se dobiva uzorkovanjem (engl. *bootstraping*) osnovnog skupa za učenje na temelju

izgrađene distribucije. Implementirani postupak kreće sa uniformnom distribucijom uzorkovanja te generira skupove podataka koji su jednake veličine kao i početni skup (ako je osnovni skup izrazito velik preporuka je uzorkovanjem graditi skupove manje veličine).

Kod uzorkovanja:

```
def sampling(dataset, dist, sampleSize):
    sampled = []
    idxMapping = {}
    for i in xrange(0, sampleSize):
        r = random()
        dsum = 0
        for d in xrange(len(dist)):
            dsum += dist[d]
            if dsum > r:
                idxMapping[d] = i
                sampled.append(dataset[d])
                break
    return (sampled, idxMapping)
```

Parametar `dist` je distribucija po kojoj provodimo uzorkovanje, inicijalno uniformna, a `sampleSize` veličina generiranog (novog) skupa podataka. Funkcija uz sam generirani skup (`sampled`) vraća i informacije o mapiranju indeksa podataka među uzorkovanim i orginalnim skupom.

4. Evaluacija implementacije

Cilj evaluacije je dati usporedbu samostalnog slabog klasifikatora i klasifikatora dobivenog boostanjem te evaluirati postupak boostanja u ovisnosti o broju slabih klasifikatora. Uspješnost klasifikatora dobivenog boostanjem ovisi o ishodu uzorkovanja (koje se izvodi slučajnim odabirom nad određenom distribucijom). Za boostani klasifikator evaluacija je provedena 20 puta te su navedene tri vrijednosti uspješnosti: najbolja, srednja i najlošija (korišteno je 50 slabih klasifikatora).

Rezultat evaluacije prikazan je tablicom 4.1. Evaluacija boostanog klasifikatora u ovisnosti o broju slabih klasifikatora prikazana je tablicom 4.2 (tablica prikazuje srednje rezultate 20 evaluacija).

4.1. Mjera kvaliteta klasifikacije

Kako ukupna točnost ne objašnjava ponašanje klasifikatora na svakom razredu pojedinačno, za svaki razred računaju se preciznost i odziv te pomoću njih ukupna težinska F mjera. Za dani razred c preciznost je omjer broja točno klasificiranih primjera u c s brojem svih primjera koji su klasificirani u razred c. Odziv je omjer broja točno klasificiranih primjera u c s brojem svih primjera koji se stvarno nalaze u razredu c. F mjera računa se za svaki razred c_i prema sljedećoj formuli:

$$F_i = \frac{2 \cdot preciznost_i \cdot odziv_i}{preciznost_i + odziv_i},$$

gdje su $preciznost_i$ i $odziv_i$ mjere preciznosti i odziva za razred c_i

Ukupna težinska mjera za dva razreda računa se tada prema formuli:

$$F_u = \frac{|c_1| \cdot F_1 + |c_2| \cdot F_2}{|c_1| + |c_2|},$$

gdje je n ukupan broj razreda, $|c_i|$ broj primjera u razredu c_i, a F_i je F mjera za razred c_i.

Tablica 4.1: Rezultat evaluacije.

Klasifikator	Točno	Netočno	F_u [%]
Decision stump (DS)	1324	276	82.32
Boostani DS (max)	1540	60	96.25
Boostani DS (avg)	1508	92	94.23
Boostani DS (min)	1474	126	92.12

Tablica 4.2: Evaluacija boostanja u ovisnosti o broju slabih klasifikatora.

Broj klasifikatora	Točno	Netočno	F_u [%]
1	1324	276	82.32
5	1504	96	94.00
10	1516	84	94.75
100	1484	116	92.75
1000	1472	128	92.00

4.2. Skup podataka

Evaluacija je provedena nad skupom podataka *Splice* dostupnom na: `http://www.cs.toronto.edu/~delve/data/splice/desc.html`. Skup sadrži označene podatke o dvama tipovima *spliceova* u DNK nizovima. Sama domena te svrha *spliceova* za ovaj rad nije bitna. Navedeni skup je odabran jer se radi o javno dostupnom skupu koji sadrži podatke podjeljene u dva razreda (originalni skup sadrži i razred koji označava da *splice* ne pripada niti jednom od navedenih tipova, no ti uzorci su izbačeni).

Skup sadrži oko 1500 uzoraka koji su nasumično izmješani te je 800 uzoraka uzeto za skup za učenje, a ostatak je ostavljen za skup za testiranje. Primjer uzorka skupa:

```
EI, GAGCAGCCAGTGCTCCAAGCCCGGTGTCATGTAAGTGCCAGTCTTCCTGCTCACCTCTAT
```

Svaki uzorak sadrži informaciju o razredu ("`EI`" ili "`IE`") te DNK niz koji uz standardne baze (ACTG) može sadržavati vrijednosti: "NDRS."

5. Primjena boostanja

Algoritam AdaBoost je brz i jednostavan. Nema parametara koje je potrebno podešavati, izuzev broja algoritama T. Ne zahtijeva znanje o slabim algoritmima koje kombinira te se stoga može kombinirati s bilo kojom metodom nalaženja slabih klasifikatora. Također je njegova učinkovitost pouzdana uz postojanje dovoljno podataka i slabih algoritama koji rade tek nešto bolje od slučajnog odabira. Zanimljivo svojstvo AdaBoost algoritma je lako pronalaženje ulaznih podataka koji odskaču od ostalih (engl. *outliers*), tj. teški su za klasifikaciju, krivo su labelirani i sl. Podatci koji u postupku boostanja dobiju maksimalnu težinu se često pokažu kao *outlieri*. Zbog ovih svojstava boostanje se pokazuje kao koristan i fleksibilan algoritam primjenjiv u raznolikim situacijama (npr. kategorizacija teksta, optičko prepoznavanje znakova, detekcija lica itd.).

AdaBoost algoritam se koristi u često spominjanom algoritmu detekcije objekata u realnom vremenu – algoritam Viole i Jonesa (Viola i Jones, 2001).

6. Zaključak

U ovom radu je dan prikaz algoritma boostanja za kombiniranje skupa slabih klasifikatora, njegova implementacija te evaluacija postignutih rezultata. Postignuti rezultati su zadovoljavajući i pokazuju svojstvo algoritma da izgrađuje klasifikator jačanjem slabih klasifikatora. Algoritam se pokazuje kao iznimno koristan jer se dizajner sustava strojnog učenja više ne mora brinuti o pronalaženju hipoteza koje vrlo dobro odvajaju razrede, već je dovoljno modelirati slabi klasifikator s uspješnošću tek nešto većom od 50%, te boostanjem ostvariti jaki klasifikator. Vrlo bitna stavka je i brzina boostanja u usporedbi s drugim algoritmima strojnog učenja, stoga je algoritam efikasan za korištenje u stvarnom vremenu. Uspješnost algoritma ovisi o odabiru slabog klasifikatora, o rezultatima uzorkovanja skupa za učenje te o broju slabih klasifikatora, pa je moguće modificirati predloženi algoritam po ovim parametrima.

7. Literatura

E. Alpaydin. *Introduction to machine learning*. The MIT Press, 2004. ISBN 0262012111.

Y. Freund. Boosting a weak learning algorithm by majority. U *Proceedings of the Third Annual Workshop on Computational Learning Theory: University of Rochester, Rochester, New York, August 6-8, 1990*, stranica 202. Morgan Kaufmann, 1990. ISBN 1558601465.

Y. Freund. An adaptive version of the boost by majority algorithm. *Machine learning*, 43(3):293–318, 2001.

Y. Freund i R. Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. U *Computational learning theory*, stranice 23–37. Springer, 1995.

Y. Freund i R.E. Schapire. Game theory, on-line prediction and boosting. U *Proceedings of the ninth annual conference on Computational learning theory*, stranice 325–332. ACM, 1996.

Y. Freund, R. Schapire, i N. Abe. A short introduction to boosting. *JOURNAL-JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE*, 14:771–780, 1999. ISSN 0912-8085.

J. Friedman, T. Hastie, i R. Tibshirani. Additive logistic regression: A statistical view of boosting. *The annals of statistics*, 28(2):337–374, 2000.

R.E. Schapire. The strength of weak learnability. *Machine learning*, 5(2):197–227, 1990. ISSN 0885-6125.

R.E. Schapire, Y. Freund, P. Bartlett, i W.S. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

V.N. Vapnik i A.Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16:264, 1971.

P. Viola i M. Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001. ISSN 1063-6919.

8. Sažetak

Rad predstavlja izgradnju učinkovitog i jakog klasifikatora korištenjem algoritma boostanja. Srž boostanja je kombiniranje skupa slabih osnovnih klasifikatora uz naglasak na klasifikaciju neispravno klasificiranih uzoraka. Algoritam je temeljen na glasanju i variranju skupa uzoraka. Ova metoda nam je posebice zanimljiva ukoliko je bitna brzina obrade, s obzirom na to da su uobičajene metode klasifikacije (SVM i sl.) puno sporije. Česta primjena se nalazi u području računalnog vida pri detekciji objekata u stvarnom vremenu. U radu će biti opisan algoritam te će biti dana implementacija metode.

Dodatak A
Programski kod implementacije

```
def adaboost(dataset, steps, weakTrain, weakClassify):
    N = len(dataset)
    sampleSize = N
    dist = []
    classifiers = []
    for i in xrange(N):
        # initial distribution for 1st algorithm: p = 1/N
        dist.append(1.0/N)

    for i in xrange(steps):
        dbg("Step " + str(i))
        errTotal = 0
        err = []
        for j in xrange(N): err.append(0)

        (sampled, mapping) = sampling(dataset, dist, sampleSize)
        classifier = weakTrain(sampled)

        for ci in mapping.keys():
            j = mapping[ci]
            x = sampled[j][0]
            if weakClassify(x, classifier) != sampled[j][1]:
                err[ci] = 1
                errTotal += err[ci]*dist[ci]

        dbg("ErrTotal: " + str(errTotal))
```

```
            if errTotal == 0:
                dbg("Stop boosting! Total error is 0.")
                break

            if errTotal > 0.5:
                break

            beta = errTotal/(1-errTotal)

            sumadist = 0
            for j in mapping.keys():
                dist[j]=beta**(1-err[j])*dist[j]
                sumadist += dist[j]

            if sumadist != 0:
                sumadist = float(sumadist)
                for j in mapping.keys():
                    dist[j] = dist[j]/sumadist

            classifiers.append((classifier, beta))

        return classifiers


def classify(x, classifiers, weakClassify):
    res = 0
    for h, beta in classifiers:
        res += -log(beta, 10)*weakClassify(x, h)

    if res < 0: return -1
    else: return 1
```