Uniwersytet Warszawski
Wydział Matematyki, Informatyki i Mechaniki

Damian Łoziński
Nr albumu: 234352

Zespoły klasyfikatorów

Praca licencjacka
na kierunku MATEMATYKA

Praca wykonana pod kierunkiem
prof. dra hab. Andrzeja Skowrona
Instytut Matematyki

Czerwiec 2008

Oświadczenie kierującego pracą

Potwierdzam, że niniejsza praca została przygotowana pod moim kierunkiem i kwalifikuje się do przedstawienia jej w postępowaniu o nadanie tytułu zawodowego.

Data Podpis kierującego pracą

Oświadczenie autora (autorów) pracy

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa została napisana przeze mnie samodzielnie i nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam również, że przedstawiona praca nie była wcześniej przedmiotem procedur związanych z uzyskaniem tytułu zawodowego w wyższej uczelni.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Data Podpis autora (autorów) pracy

Streszczenie

Praca zawiera przegląd wybranych metod tworzenia zespołu z nieskomplikowanych klasyfikatorów bazowych, przegląd metod zbierania wyników oraz najnowszych osiągnięć.

Słowa kluczowe

zespoły klasyfikatorów, Bagging, Boosting, AdaBoost

Dziedzina pracy (kody wg programu Socrates-Erasmus)

11.0 Matematyka, Informatyka
11.4 Sztuczna inteligencja

Klasyfikacja tematyczna

68T05 Learning and adaptive systems
68T37 Reasoning under uncertainty

Tytuł pracy w języku angielskim

Classifier Ensembles

Spis treści

Wprowadzenie

W miarę rozwoju dziedziny *uczenia maszynowego*, potrafimy nauczyć komputer rozwiązywać coraz to bardziej złożone zadania. Jednak wiele podstawowych, z punktu widzenia człowieka, problemów jest wciąż poza zasięgiem współczesnych technik komputerowych.

Znane jest np. zadanie rozpoznawania twarzy. Mimo intensywnych prac w ciągu kilkudziesięciu ostatnich lat, nadal nie możemy uzyskać satysfakcjonującego nas rozwiązania. Wypracowano wiele metod i algorytmów radzących sobie zupełnie nieźle z przeróżnymi problemami. Jednak coraz wyżej stawiana poprzeczka nieustannie pobudza dalszy rozwój.

Jedną z metod ulepszania "tego co już mamy" jest łączenie klasyfikatorów w zespoły. Jak się później okaże, umiejętne dobieranie zespołu może przynieść wiele korzyści.

Niniejsza praca składa się z sześciu rozdziałów. W **rozdziale 1** wyjaśniłem pojęcia, którymi będziemy się posługiwać. **Rozdział 2** zawiera krótkie opisy podstawowych klasyfikatorów, z których będziemy budować zespoły. W **rozdziale 3** przedstawione są najważniejsze metody konstrukcji zespołów klasyfikatorów. Znaczna jego część poświęcona jest algorytmowi AdaBoost, który moim zdaniem jest jedną z ważniejszych metod budowy zespołu. Nie zabrakło w nim też omówienia innych, równie przydatnych, narzędzi.
Rozdział 4 pokazuje różne sposoby wyłuskiwania ostatecznej odpowiedzi multi–klasyfikatora.
Dodatek A przedstawia jedno z najnowszych osiągnięć w dziedzinie rozpoznawania wzorców.

Rozdział 1

Pojęcia podstawowe[1]

Definicja 1.0.1 Klasyfikatorem bazowym f *nazywamy system podejmujący decyzje* d *na podstawie danych, paramatrów wejściowych* $a_1, \ldots, a_n$

$$f(a_1, \ldots, a_n) = d \ .$$

Czasami mówimy o kilku – m decyzjach. Wtedy $f(a_1, \ldots, a_n) = (d_1, \ldots, d_m)$. Często też przyjmujemy, że $a_i \in \mathbb{R}$ lub $a_i \in \{1, \ldots, l\}$ oraz $d_i \in \{0, 1\}$ lub $d_i \in \{1, \ldots, k\}$.

Definicja 1.0.2 Zespół klasyfikatorów Z *jest to kasyfikator, w którego skład wchodzą na ogół mniej skomplikowane klasyfikatory bazowe* $f_1, \ldots, f_k$ *(*$Z = \{f_1, \ldots, f_k\}$*), na podstawie których podejmowana jest ostateczna decyzja.*

1.1. Problem decyzyjny

Dany mamy zestaw atrybutów (parametrów wejściowych) ozn. $A = \{a_1, \ldots, a_n\}$ oraz zestaw decyzji $\{d_1, \ldots, d_m\}$. Naszym zadaniem jest sklasyfikować przychodzący obiekt x opisany za pomocą A ($a_i(x) = b_i$, gdzie b_i są dane), tzn. określić wartości $d_j(x)$ dla $j \in \{1, \ldots, m\}$ na podstawie otrzymanych wartości b_i.

Często bywa tak, że zbiór decyzji, które trzeba podjąć jest jednoelementowy: $\{d\}$. Jeśli decyzja d może przyjmować k wartości, tzn. $d \in \{1, \ldots, k\}$ (albo równoważnie: $d = dec_1, \ldots, dec_k$) mówimy o k klasach decyzyjnych.

Definicja 1.1.1 Klasy decyzyjne *dla decyzji* d *odpowiadają różnym wartościom (decyzjom), jakie może przyjąć* d*.*

Mówimy, że obiekt x należy do klasy decyzyjnej dec_l jeśli $d(x) = dec_l$.

1.2. Uczenie

Definicja 1.2.1 Zbiór przykładów *tworzą pary* $\langle x_i, c(x_i)\rangle$*, gdzie* x_i *to obiekty przykładowe opisane za pomocą* A*, a* $c(x_i)$ *to odpowiadające im znane decyzje (*$d(x_i) = c(x_i)$*)*

Definicja 1.2.2 *Funkcję* $c(x)$*, której próbujemy się nauczyć nazywamy* funkcją celu*. Przeszukiwana przy tym przestrzeń* $H = \{h(x) \mid h((b_1, \ldots, b_n)) = d\}$ *to* przestrzeń hipotez *(*x *jest reprezentowany jako wektor* $(b_1, \ldots, b_n)$*).*

[1] por. [Mitch97]

Uczenie polega na przeszukiwaniu przestrzeni H w celu znalezienia jak najlepszego przybliżenia $h(x) \sim c(x)$. Zbiór przykładów D, używany podczas uczenia naszego klasyfikatora, nazywamy *zbiorem treningowym*. A zbiór T, służący do testowania ucznia, to *zbiór testowy*.

Definicja 1.2.3 Błąd na zbiorze treningowym (błąd treningowy) *definiujemy następującym wzorem*

$$err_D = \frac{1}{|D|} \sum_{x \in D} \delta[d(x_i) \neq c(x_i)], \tag{1.1}$$

a na zbiorze testowym (błąd testowy)

$$err_T = \frac{1}{|T|} \sum_{x \in T} \delta[d(x_i) \neq c(x_i)], \tag{1.2}$$

gdzie $\delta[warunek]$ *przyjmuje* 1 *jeśli warunek jest spełniony i* 0 *w p.p.*

Ważne jest, aby zbiór testowy nie był widziany podczas treningu. Mówimy, że klasyfikator ma skuteczność p jeśli $p = 1 - err_T$.

Rozdział 2

Klasyfikatory bazowe

Przedstawimy teraz, pokrótce, w zasadzie jedynie wymienimy, klasyfikatory bazowe, które w dalszej części tego tekstu, posłużą nam do budowy zespołów. Opiszemy jedynie ich główne cechy. Są one szczegółowo omówione np. w [Mitch97, Kun04].

2.1. Klasyfikator liniowy

Definicja 2.1.1 *Załóżmy, że mamy k klas decyzyjnych* $dec_1, \ldots, dec_k$. *Wtedy* funkcją rozróżnialności *nazywamy funkcję* $F : D \to \mathbb{R}^k$, *która dla danego obiektu przypisuje k wartości* — współczynniki wsparcia *dla poszczególnych klas. Im większy współczynnik wsparcia tym bardziej prawdopodobne, że obiekt należy do danej klasy.*

Niech $F = (f_1, \ldots, f_k)^T$. Obiekt x jest przypisywany do klasy o najwyższym współczynniku wsparcia, tzn. jeśli

$$M = \max_i \{f_i(x)\},$$

wtedy

$$d(x) = dec_i \text{ dla } i \text{ takiego, że } M = f_i(x). \tag{2.1}$$

Najprostszą funkcją rozróżnialności jest funkcja afiniczna

$$F(x) = W^T x + v, \tag{2.2}$$

gdzie $W = [w_1, \ldots, w_k] \in \mathbb{R}^{k \times n}$ jest macierzą, której kolumny to wektory obliczonych współczynników dla f_i $(i = 1, \ldots, k)$, a $v = (v_1, \ldots, v_k) \in \mathbb{R}^k$ to wektor współczynników wolnych

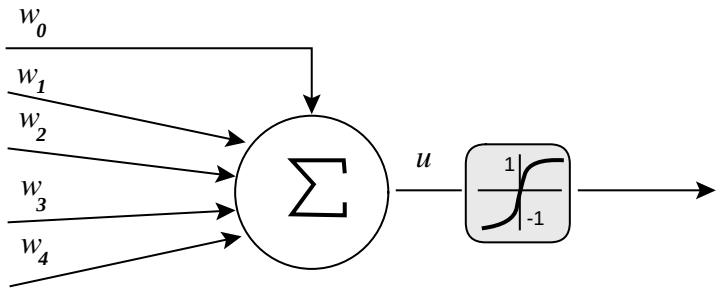
$$f_i(x) = w_i^T x + v_i.$$

Macierz W i wektor v są liczone tak, by minimalizować błąd na danych treningowych[1].

2.2. Sztuczne sieci neuronowe

Definicja 2.2.1 Sztuczne sieci neuronowe *są to struktury składające się z sieci połączonych ze sobą* receptorów. *Każdy* receptor *ma n wejść i jedno wyjście. Sygnały z wejścia* $x = (x_1, \ldots, x_n)$ *są sumowane z odpowiednimi wyuczonymi wagami* $w = (w_1, \ldots, w_n), w_0$ *i podawane na wyjście u*

[1] por. [Kun04]



Rysunek 2.1: Perceptron

$$u = w^T x + w_0. \tag{2.3}$$

Wyjście u może być jeszcze modyfikowane przez tzw. *funkcję progową*, która ma za zadanie przeskalowanie wyniku do przedziału $[-1, 1]$, jednocześnie "wyostrzając" różnice w zerze. Sieci mogą składać się z warstw. Wyjścia receptorów z warstwy i połączone są z wejściami warstwy $i + 1$. Rozważa się też inne topologie sieci (np. rekurencyjne).

2.3. K najbliższych sąsiadów (k-NN [2])

Inną prostą metodą klasyfikacji przychodzącego obiektu jest przydzielenie go do klasy decyzyjniej na podstawie kilku najbardziej "podobnych" do niego znanych przykładów. Zakładamy tutaj, że obiekty x, które mamy sklasyfikować są elementami przestrzeni $\mathbb{R}^n$. Wtedy podobieństwo między dwoma obiektami x, y określa miara euklidesowa

$$dist(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2$$

Im mniejsza jest wartość tej miary, tym uważamy je za bardziej podobne.
Klasyfikacja nowego obiektu x polega na znalezieniu k najbliższych przykładów w *danych treningowych* D i przypisaniu mu klasy reprezentowanej przez większość spośród wyznaczonych k reprezentantów.

2.4. Drzewa decyzyjne

Bardzo często stosowanym rozwiązaniem jest kombinacja kilku "małych" drzew. Czasem są to wręcz pojedyncze reguły tzw. pieńki decyzyjne (ang. decision stumps). Takie struktury okazują się bardzo wydajne w połączeniu z AdaBoost - znanym algorytmem budowy klasyfikatora zespołowego, któremu poświęcona jest duża część następnego rozdziału.

Definicja 2.4.1 Drzewo decyzyjne *jest to struktura składająca się z wierzchołków połączonych gałęziami (krawędziami). Wierzchołkiem wyróżnionym jest* korzeń*, który znajduję się na szczycie struktury, ale pełni w zasadzie tę samą rolę, co reszta zwykłych wierzchołków. Innymi wyróżnionymi wierzchołkami są* liście*, które zaś cechuje odmienna budowa. Poza liściami, każdy wierzchołek, zawiera* test*, tym samym dzieli klasyfikowany zbiór w tym wierzchołku na pewną liczbę parami rozłącznych podzbiorów.*

[2] ang. k - nearest neighbours

Drzewo jest konstruowane w ten sposób, aby w liściach zostawały podzbiory elementów jednej klasy decyzyjnej (a przynajmniej w przeważającej większości). Wtedy liściowi (czyli wszystkim wpadającym do niego elementom) przypisujemy jedną klasę decyzyjną, reprezentowaną przez większość znajdujących się w nim przykładów ze *zbioru treningowego* D. Wierzchołki nie będące liśćmi zawierają *testy na atrybutach*. Testy te mogą być binarne:

$$\text{jeśli } a_i(x) < r_i \ (a_i(x) = r_i) \text{ to } \textit{idź w lewo}, \text{ w p.p. } \textit{idź w prawo}$$

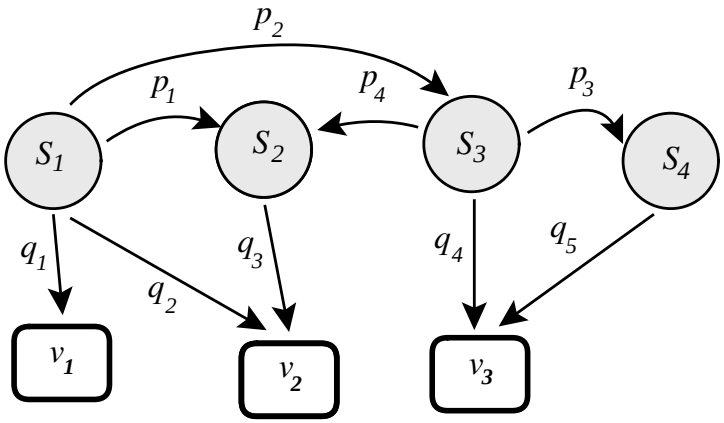
lub nieco bardziej skomplikowane, jeśli np. $a_i(x) \in \{1, \ldots, l\}$

$$\text{dla } j = 1, \ldots, l: \quad \text{jeśli } a_i(x) = j \text{ to } \textit{idź gałęzią nr } j.$$

Budowa optymalnych drzew nie jest rzeczą łatwą z punktu widzenia złożoności obliczeniowej. Kluczowa jest tutaj kolejność wybierania atrybutów do testów. Jest to dość szeroko opisane w [Mitch97].

2.5. Ukryty model Markowa (HMM[3])

Jest to model statystyczny, który zakłada, że obserwowany proces jest procesem Markowa, a jego parametry nie są znane. Jego zadaniem jest znalezienie tych parametrów na podstawie obserwowanych zmiennych, o których zakładamy, że są ściśle powiązane z poszczególnymi stanami. Znalezione parametry można później wykorzystać do klasyfikowania przychodzących nowych obiektów. HMM stanowi dobry trzon do budowy klasyfikatorów np. do rozpoznawania wzorców. Możemy też patrzeć na HMM, jak na najprostszą dynamiczną sieć Bayesowską[4]. W zwykłym modelu Markowa poszczególne stany procesu dają się obserwować, a jedynymi parametrami są prawdopodobieństwa przejścia z jednego stanu do drugiego. Ukryty model Markowa udostępnia do wglądu jedynie zmienne, modyfikowane przez ukryte stany z zadanym rozkładem prawdopodobieństwa.



Rysunek 2.2: Parametry ukrytego modelu Markowa – p, q; ukryte stany – S; widoczne zmienne – v

[3] ang. Hidden Markov Model
[4] zob. [Mitch97]

Rozdział 3

Budowa zespołu[1]

Po nieco przydługim wstępie, ale wydaje się, że jednak potrzebnym, zajmiemy się wreszcie tworzeniem zespołu klasyfikatorów. Dokładniej, przedstawimy różne metody pozwalające zbudować "dobry" zespół z "mniej – zdolnych" członków. Trzeba umieć wykorzystać zalety prostych klasyfikatorów takie, jak nieskomplikowana złożoność obliczeniowa ich konstruowania czy prostota budowy, do skonstruowania z nich, bardziej złożonego systemu decyzyjnego. Takie właśnie podejście, czyli łączenie ze sobą kilku *różnych* klasyfikatorów wydaje się mieć sporo zalet. Oczekujemy, że będziemy mogli dysponować większą siłą wyrażania, a co za tym idzie powinniśmy dać radę nauczyć się trudniejszych pojęć. Można się też spodziewać, że taki multi–klasyfikator będzie mniej podatny na osobliwości danych i bardziej odporny na tzw. *przeuczenie się*. Mimo, iż często są to tylko pobożne życzenia, w następnym paragrafie przekonamy się, że przy odpowiednich założeniach się spełniają.

3.1. Boosting – Algorytm AdaBoost

Metoda ta została opracowana przez Yoava Freunda i Roberta Schapire'a [Schep03]. Nazwa wzięła się od *Adaptative Boosting* (z ang. wzmocnienie adaptacyjne). AdaBoost w kolejnych t iteracjach trenuje t "słabych" klasyfikatorów na zbiorze przykładów D ze zmienianymi wagami. Zacznijmy od definicji "słabego" *ucznia*

Definicja 3.1.1 "Słabym" klasyfikatorem *nazywamy klasyfikator stosunkowo prosty, o niezbyt dużej sile wyrażania, potrafiący klasyfikować dane testowe ze skutecznością większą niż 50%.*

3.1.1. Implementacja

Dany mamy zbiór treningowy $D = \{\langle x_1, c(x_1)\rangle, \ldots, \langle x_n, c(x_n)\rangle\}$ składający się z n przykładów. Niech $\mathcal{P} = \{p_1, \ldots, p_n\}$ będzie zbiorem wag obiektów $x_1, \ldots, x_n$ odpowiednio. Niech ponadto $f_1, \ldots, f_t$ będą "słabymi" klasyfikatorami. Poniższy fragment kodu przedstawia schemat działania algorytmu.

```
1:   P := { p[1] = 1/n, ..., p[n] = 1/n }
2:   Z := {}
3:   dla k = 1...t :
4:      trenuj f[k] na zbiorze D z wagami P
```

[1] por. [Kun04]

```
5:      err[k] := oblicz błąd ważony f[k] na D
6:      w[k]   := oblicz wagę klasyfikatora f[k]
7:      P      := oblicz nowy zbiór wag dla D
8:      dodaj f[k] z wagą w[k] do Z
```

Przypuśćmy, że chcemy teraz sklasyfikować przychodzący obiekt x

```
 9:   dla każdego f[i] ze zbioru Z :
10:      dec[i] := policz f[i](x)
11:   przyporządkuj x do klasy decyzyjnej o największej sumie w[i]
```

Widzimy, że na początku wagi wszystkich obiektów treningowych są jednakowe (krok 1), a zespół, nie zawiera jeszcze, żadnego klasyfikatora (krok 2). Jak wygląda trenowanie f_k z uwzględnieniem wag, w kroku 4? Otóż wagi $p_1, \ldots, p_n$ możemy interpretować, jako rozkład prawdopodobieństwa atomów $\langle x_1, c(x_1)\rangle, \ldots, \langle x_n, c(x_n)\rangle$. Podczas treningu losujemy przykłady ze zbioru D zgodnie z rozkładem P. Zatem proces uczenia musi być wrażliwy na wielokrotne otrzymanie danej pary $\langle x_j, c(x_j)\rangle$. Musimy też zadbać o to, by przy każdej iteracji pętli

$$\sum_{j=1}^{n} p_j^{(k)} = 1. \tag{3.1}$$

Co kryje się pod pojęciem *błąd ważony* w kroku 5?

$$err_k = \sum_{j=1}^{n} p_j^{(k)}\ \delta[f_k(x_j) \neq c(x_j)] \quad (\delta \text{ jak w definicji 1.2.3}). \tag{3.2}$$

Wagę w_k liczymy wg następującej formuły

$$w_k = 1/2 \cdot \log\left(\frac{1 - err_k}{err_k}\right). \tag{3.3}$$

Aby wyznaczyć *nowy* rozkład $\mathcal{P}^{(k+1)} = \{p_1^{(k+1)}, \ldots, p_n^{(k+1)}\}$, dla $j = 1, \ldots, n$ obliczamy

$$\tilde{p}_j^{(k)} = \begin{cases} p_j^{(k)}\ e^{-w_k} & \text{jeśli } f_k(x_j) = c(x_j) \\ p_j^{(k)}\ e^{w_k} & \text{jeśli } f_k(x_j) \neq c(x_j) \end{cases}, \tag{3.4}$$

a następnie, pamiętając o warunku 3.1, normalizujemy

$$\begin{aligned} s_k &= \textstyle\sum_{i=1}^{n} \tilde{p}_i^{(k)}\ , \\ p_j^{(k+1)} &= \tfrac{\tilde{p}_j^{(k)}}{s_k} \quad (j = 1, \ldots, n). \end{aligned} \tag{3.5}$$

3.1.2. Komentarz

Podczas działania algorytmu, w każdym przebiegu pętli koncentrujemy się na obiektach źle sklasyfikowanych przez poprzednich członków. Z podstawienia 3.4 widać, że jeśli obiekt x_j został *źle* sklasyfikowany przez f_k, to jego waga (a tym samym prawdopodobieństwo wylosowania następnym razem) *rośnie*, natomiast *maleje*, gdy f_k sobie z nim poradził. Takie postępowanie pozwala "słabym" *uczniom* skupić się na odpowiednim fragmencie przestrzeni

i zawężać go stopniowo w kolejnych iteracjach. Co więcej, każdy uczeń jest oceniany i zostaje mu przypisana waga w_k. Ze wzorów 3.2 i 3.3 wynika, że jeśli f_k popełnia błędy na źle klasyfikowanych przykładach przez poprzedników z Z, to otrzymuje gorszą ocenę.

Warto podkreślić, że algorym AdaBoost może znacznie poprawić jakoś klasyfikacji, jednak owa poprawa obserwowana jest tylko wtedy, gdy jako składowych używamy "słabych" klasyfikatorów. Aby uzyskać rozsądną efektywność wystarczy, by skuteczność "słabych" *uczniów* była nieco wyższa niż 50%. Stosowanie algorytmu do bardziej skomplikowanych klasyfikatorów nie prowadzi na ogół do znaczącego zwiększenia skuteczności.

3.1.3. Oszacowanie błędu

Skupmy się na chwilę na problemie o dwóch klasach decyzyjnych $\{-1, 1\}$. Robert E. Schapire w [Schep03] podaje następujące oszacowanie *błędu treningowego* dla $Z = \{\langle f_1, w_1\rangle, \ldots, \langle f_t, w_t\rangle\}$ (por. def. 1.2.3)

$$err_D(Z) = \frac{1}{n}\sum_{i=1}^{n} \delta[z(x_i) \neq c(x_i)] \leqslant \frac{1}{n}\sum_{i=1}^{n} \exp(-c(x_i)\tilde{z}(x_i)) = \prod_{k=1}^{t} s_k \tag{3.6}$$

gdzie $\tilde{z}(x) = \sum_{k=1}^{t} w_k f_k(x)$, czyli $z(x) = \text{sign}(\tilde{z})$, a s_k jest zdefiniowane wzorem 3.5.

Nierówność w formule 3.6 bierze się stąd, że

$$\sum_{i=1}^{n} \exp(-c(x_i)\tilde{z}(x_i)) = \sum_{i,c(x_i)=f(x_i)} e^{-\sum_k w_k} + \sum_{i,c(x_i)\neq f(x_i)} e^{\sum_k w_k} \geqslant \sum_{i,c(x_i)\neq f(x_i)} 1,$$

gdzie, $\sum_k$ oznacza $\sum_{k=1}^{t}$. Natomiast ostatnią równość w 3.6 można uzasadnić w sposób następujący. Po pierwsze zauważmy, że zgonie z podstawieniami 3.4 i 3.1 mamy $p_j^{(k+1)} = \frac{p_j^{(k)} \exp(-w_k c(x_j) f(x_j))}{s_k}$ (bo $f(x), c(x) \in \{-1, 1\}$), z czego wynika, że $\frac{p_j^{(k+1)} s_k}{p_j^{(k)}} = \exp(-w_k c(x_j) f(x_j))$. Dalej

$$\sum_{i=1}^{n} \exp(-c(x_i)\tilde{z}(x_i)) = \sum_{i=1}^{n} e^{-\sum_k w_k c(x_i) f(x_i)} = \sum_{i=1}^{n} \prod_{k=1}^{t} e^{-w_k c(x_i) f(x_i)}$$

i korzystając z poczynionej przed chwilą uwagi otrzymujemy

$$\sum_{i=1}^{n} \exp(-c(x_i)\tilde{z}(x_i)) = \sum_{i=1}^{n} \prod_{k=1}^{t} \frac{p_i^{(k+1)} s_k}{p_i^{(k)}} = \sum_{i=1}^{n} \left(\frac{p_i^{(2)} s_1}{p_i^{(1)}} \frac{p_i^{(3)} s_2}{p_i^{(2)}} \cdots \frac{p_i^{(t+1)} s_t}{p_i^{(t)}} \right) = \sum_{i=1}^{n} \frac{p_i^{(t+1)}}{p_i^{(1)}} \prod_{k=1}^{t} s_k.$$

Zatem pamiętając, że $p_j^{(1)} = \frac{1}{n}$ dla $j = 1, \ldots, n$ oraz $\sum_i p_i^{(k)} = 1$ upraszczamy ostatnią sumę i dostajemy pożądane oszacowanie

$$\frac{1}{n}\sum_{i=1}^{n} \exp(-c(x_i)\tilde{z}(x_i)) = \frac{1}{n}\sum_{i=1}^{n} n p_i^{(t+1)} \prod_{k=1}^{t} s_k = \prod_{k=1}^{t} s_k.$$

□

Dla dwóch klas decyzyjnych klasyfikator losowy ma skuteczność 50%. Niech err_D^k będzie błędem f_k na danych treningowych, a γ_k niech będzie takie, że

$$err_D^k = 1/2 - \gamma_k,$$

wtedy γ_k mówi, o ile f_k jest lepszy od klasyfikatora losowego. Korzystając ze wzoru 3.3 mamy

$$\prod_{k=1}^{t} s_k = \prod_{k=1}^{t} 2\sqrt{err_D^k(1 - err_D^k)} = \prod_{k=1}^{t} \sqrt{1 - 4\gamma_k^2} \leqslant \exp\left(-2\sum_{k=1}^{t} \gamma_k^2\right). \tag{3.7}$$

Widzimy, że nawet niewielka wartość $0 < \gamma \leqslant \gamma_k$ daje wykładniczy spadek $err_D(Z)$. Obecnie algorytm posiada wiele ulepszonych wariantów – AdaBoost.MH, MR, M2. (por. [Fre99]) Liczne eksperymenty wykazały, że AdaBoost jest odporny na *przeuczenie się*. A czasem nawet po osiągnięciu zerowego błędu na danych treningowych, nadal obserwowano dalszy spadek błędu na danych testowych (por. [Fre99]).

3.2. Bagging

Nieco inne – prostrze – podejście prezentuje metoda *Bagging*[2]. Klasyczny algorytm w każdej iteracji $k = 1, \ldots, t$ losuje ze zwracaniem reprezentatywną[3] próbkę przykładów ze zbioru D i trenuje na nich f_k. Decyzja podejmowana jest większością głosów. Spójrzmy na fragment kodu

```
1:   Z = {}
2:   dla k = 1...t
3:      S[k] := wylosuj ze zwracaniem próbkę ze zbioru D
4:      trenuj f[k] na S[k]
5:      dodaj f[k] do Z
```

Teraz klasyfikujemy przychodzący obiekt x

```
5:   dla każdego f[i] w Z:
6:      oblicz f[i](x)
7:   przyporządkuj x do klasy decyzyjnej o jakwiększej liczbie głosów
```

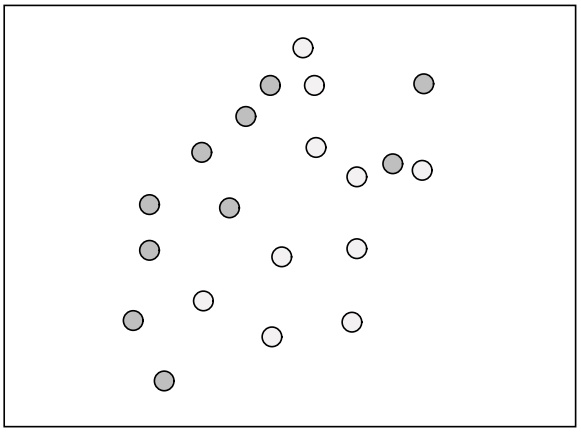
Ta metoda, podobnie jak poprzednia, wymaga, aby proces uczenia był wrażliwy na powtarzanie przykładów. Dzięki wielokrotnemu losowaniu próbek wg stałego rozkładu jest dość odporna na szumy – w przeciwieństwie do *boosting-u*, gdzie początkowo, być może, lekceważony błąd w danych, w kolejnych iteracjach zyskuje coraz większą wagę (a tym samym większy wpływ na cały proces uczenia). Bagging dostarcza rozwiązanie będące uśrednieniem otrzymanych wyników indywidualnych. Ewentualne sporadyczne nieprawidłowości w danych treningowych nie będą miały dużego wpływu na cały proces uczenia i klasyfikacji, gdyż dzięki metodzie *bootstrap* będą stanowiły bardzo niewielką część ogółu wylosowanych przykładów.

[2] akronim z angielskich słów *Bootstrap AGGregateING*

[3] (ang. bootstrap – cholewa) losujemy m przykładów zgodnie z rozkładem jednostajnym na D, jeśli $m = |D|$ to spodziewamy się, że w każdej próbce będzie ok. 2/3 unikalnych obiektów i 1/3 będzie się powtarzać

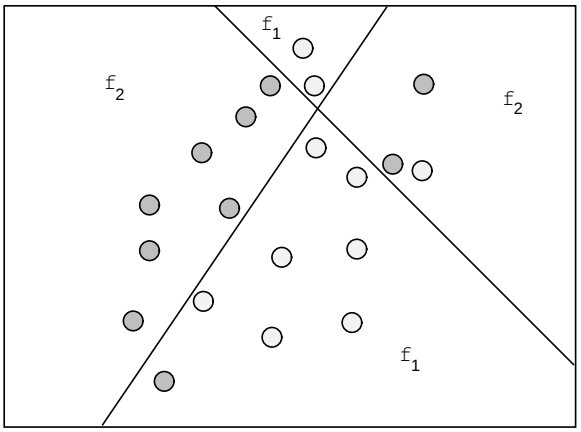
3.3. Selekcja

Mając zespół klasyfikatorów, można jeszcze spróbować podejść do problemu klasyfikacji inaczej, dzieląc przestrzeń hipotez na podzbiory (dla wygody załóżmy, że rozłączne) i przypisując tym podzbiorom poszczególnych członków zespołu. Wtedy wpadający obiekt zostaje sklasyfikowany za pomocą jednego, odpowiedniego ucznia, który jest odpowiedzialny za dany obszar. Spójrzmy na poniższy przykład.



Rysunek 3.1: Dwie klasy decyzyjne - skuteczność zespołu wynosi 50%

Załóżmy, że mamy do dyspozycji dwa klasyfikatory bazowe, z których jeden zawsze wskazuje klasę "szarą", a drugi – zawsze "białą". Skuteczność obu na całej przestrzeni wynosi 50% – nie możemy zatem liczyć na *Bagging* ani *Boosting*. Podzielmy za to przestrzeń na cztery podzbiory, jak na rysunku 3.3 i przypiszmy każdemu fragmentowi odpowiedni klasyfikator.



Rysunek 3.2: Skuteczność zespołu na podzielonej przestrzeni wynosi 90%

Tym razem tylko dwa spośród dwudziestu obiektów zostały źle sklasyfikowane. Otrzymujemy skuteczność 90%. To bardzo duża poprawa. Tym bardziej, że użyliśmy do tego z pozoru bezużytecznych klasyfikatorów. Tylko skąd wzięliśmy linie podziału? W tym przypadku nie trudno było je wymyślić, ale jest to przypadek bardzo szczególny (tylko dwie klasy decyzyjne i raptem 20 obiektów). Łatwo można sobie wyobrazić, bardziej skomplikowane zadanie. Aby skutecznie rozwiązać problem tą metodą musimy umieć: znaleźć *obszary kompetencji* poszczególnych klasyfikatorów, oszacować ich skuteczność oraz wybrać metodę pozyskiwania

końcowej odpowiedzi (np. głosowanie, głosowanie ważone wg kompetencji, selekcja).

Niech $Z = \{f_1, \ldots, f_t\}$ będzie zespołem t klasyfikatorów. Niech $\mathbb{R}^n$ będzie podzielone na m obszarów $R_1, \ldots, R_m$. Podział ten jest ustalony arbitralnie. Obszary nie są przypisane do żadnych klas, nie muszą mieć też żadnego szczególnego kształtu ani rozmiarów. Niech ponadto $\hat{f} \in Z$ będzie klasyfikatorem o największej skuteczności na całej przestrzeni $\mathbb{R}^n$. Przez $P(f_k|R_j)$ oznaczamy prawdopodobieństwo poprawnego sklasyfikowania obiektu x przez f_k na przestrzeni R_j. Niech $f_{idx(j)} \in Z$ oznacza klasyfikator odpowiedzialny za obszar R_j ($idx : \{1, \ldots, m\} \to \{1, \ldots, t\}$). Zatem prawdopodobieństwo prawidłowej klasyfikacji przez nasz zespół wynosi

$$P(Z) = \sum_{j=1}^{m} r_j P(f_{idx(j)}|R_j), \tag{3.8}$$

gdzie przez r_j oznaczamy prawdopodobieństwo tego, że losowo wybrany obiekt z $\mathbb{R}^n$ trafi do podzbioru R_j. Maksymalizując $P(Z)$ wybieramy taką funkcję idx, by

$$\forall_{k \in \{1,\ldots,t\}} P(f_{idx(j)}|R_j) \geqslant P(f_k|R_j). \tag{3.9}$$

Konflikty rozwiązywane są w sposób losowy. Z równań 3.8 oraz 3.9 mamy

$$P(Z) \geqslant \sum_{j=1}^{m} r_j P(\hat{f}|R_j) = P(\hat{f}). \tag{3.10}$$

Otrzymujemy zatem skuteczność co najmniej równą skuteczności najlepszego klasyfikatora bazowego, niezależnie od tego, jak podzieliliśmy przeszukiwaną przestrzeń. Jedynym warunkiem jaki musi być spełniony jest to by $f_{idx(j)}$ był rzeczywiście najlepszym klasyfikatorem dla obszaru R_j.

Rozdział 4

Zbieranie wyników

Gdy system decyzyjny składa się z więcej niż jednego klasyfikatora, pojawia się wtedy pytanie, jak wybrać ostateczną odpowiedź. Czy wybrać jednego najlepszego reprezentanta, czy przeprowadzić głosowanie, czy może tez uśrednić wszystkie wyniki. Ten rozdział omawia rożne sposoby. Ciekawą opcją wydaje się zbudowanie klasyfikatora końcowego, który by potrafił nauczyć się interpretować[1] odpowiedzi uczniów wchodzących w skład zespołu.

4.1. Jednomyślność

W niektórych zastosowaniech, gdy wymagamy dużej pewności, a podjęcie złej decyzji jest dużo gorsze niż niepodjęcie jej w ogóle, możemy wymagać zgodności wszystkich klasyfikatorów bazowych.[2] Jednak w praktyce takie podejście nie zawsze się sprawdza, a to za sprawą tego, że w większości przypadków mamy do czynienia z klasyfikatorami obarczonymi pewnym błędem. Rzadko kiedy dysponujemy możliwością przeanalizowania całej przestrzeni hipotez, dlatego musimy stosować różne heurystyki, które z definicji jedynie przybliżają rozwiązanie.

Możemy za to dopuścić pewien mały margines niepewności i zgodzić się by 1 z 10 uczniów dawał inną odpowiedź[3].

4.2. Głosowanie równoprawne

Jednym z podstawowych sposobów stosowanym np. w *bagging-u* jest głosowanie. Jest to rodzaj uśredniania wyniku w przypadku decyzji nieciągłej. Ta prosta metoda pozwala ominąć osobliwości i uodpornić się na zaburzenia – dlatego jest dobrym rozwiązaniem, gdy pracujemy na "niepewnych" danych.
Niech $\{d_k^j(x)\}_{j=1}^m \in \{0,1\}^m$ będzie decyzją klasyfikatora f_k tzn.

$$d_k^j(x) = \delta[f_k(x) = dec_j] \quad (\delta \text{ jak w def. 1.2.3}), \tag{4.1}$$

wtedy decyzja $Z = \{f_1, \ldots, f_t\}$ wygląda tak:

$$d_Z(x) = dec_p \iff \sum_{k=1}^{t} d_k^p(x) = \max_j \left(\sum_{k=1}^{t} d_k^j(x) \right). \tag{4.2}$$

[1] dokładniej, nauczyć się jak reagują poszczególne klasyfikatory składowe na dane przypadki testowe – czyli wygenerować ostateczną odpowiedź na podstawie *wyjść* klasyfikatorów bazowych

[2] np. w medycynie – diagnozowanie chorych; lepiej nie podać, żadnych leków niż podać złe

[3] otrzymujemy wtedy *prawie-jednomyślność*

Głosowanie proste nie wymaga dodatkowych obliczeń – jedynie zliczenie głosów, dzięki czemu jest efektywnym i często stosowanym sposobem zbierania wyników. Przyjrzyjmy się temu nieco dokładniej. W tym celu załóżmy, że dla każdego k prawdopodobieństwo dobrej odpowiedzi f_k wynosi p oraz odpowiedzi wszystkich f_k są niezależne, tzn. $\forall A \subseteq Z, A = \{f_{i_1}, \ldots, f_{i_k}\}$:

$$P(f_{i_1}(x) = d_1, \ldots, f_{i_k}(x) = d_k) \;=\; P(f_{i_1}(x) = d_1) \cdot \ldots \cdot P(f_{i_k}(x) = d_k). \tag{4.3}$$

Ze wzoru 4.2 widzimy, że zespół Z da poprawną odpowiedź, jeśli *co najmniej* $\lfloor t/2 \rfloor + 1$ spośród t klasyfikatorów się nie pomyli. Zatem skuteczność Z możemy policzyć ze wzoru

$$P_t = \sum_{k=\lfloor t/2 \rfloor + 1}^{t} \binom{t}{k} p^k (1-p)^{t-k}. \tag{4.4}$$

Twierdzenie 4.2.1 (Marquis de Condorcet 1785). *Załóżmy, że liczba t niezależnych klasytikatorów składowych zespołu Z jest nieparzysta oraz P_t jest zdefiniowane formułą 4.4, wtedy:*

- *jeśli $p > 1/2$, to $\lim_{t\to\infty} P_t = 1$*
- *jeśli $p < 1/2$, to $\lim_{t\to\infty} P_t = 0$*
- *jeśli $p = 1/2$, to $P_t = 1/2$ dla każdego t*

przy czym w dwóch pierwszych przypadkach zbieżność jest monotoniczna (rosnąca do 1, malejąca do 0).

Dowód. Załóżmy, że mamy k klasyfikatorów z czego m głosuje poprawnie. Rozważmy, co się stanie, gdy dodamy kolejne dwa (lub więcej – nie zmieniając parzystości). Decyzja Z zmienia się jedynie w następujących dwóch przypadkach:

- m było za małe by decyzja Z była poprawna (o 1 mniejsze), ale po dodaniu dwóch poprawnych głosów całkowita liczba jest już wystarczająca,
- m było o 1 większe niż $\lfloor k/2 \rfloor$, ale doszły dwa niepoprawne głosy i Z daje złą odpowiedź.

W pozostałych przypadkach nowe głosy się równoważą, bądź też różnica głosów się powiększa lub pomniejsza, ale nadal jest za duża by zmienić wynik. Zatem interesuje nas jedynie sytuacja, gdy pojedynczy głos (spośród pierwszych k) decyduje o wyniku całego głosowania. Ograniczając uwagę do takiej sytuacji możemy sobie wyobrazić, że pierwsze $k-1$ głosów się równoważy, zatem k-ty jest decydujący. W tym przypadku prawdopodobieństwo uzyskania poprawnego rezultatu wynosi p (a nieprawidłowego $1-p$). Rozważmy teraz dwa dodatkowe klasyfikatory. Prawdopodobieństwo, że zmienią one decyzję ze złej na dobrą wynosi $(1-p)p^2$, a z dobrej na złą — $p(1-p)^2$. Pierwsza z tych dwóch liczb jest większa od drugiej wtedy i tylko wtedy, gdy $p > 1/2$, co dowodzi powyższego twierdzenia.

□

Założenie nieparzystości w powyższym twierdzeniu nie jest istotne. Eliminuje jedynie przypadek nierozstrzygnięty (gdy obie decyzje mają po tyle samo głosów). Jednak ten sam argument działa również dla parzystych k, jeśli takie sytuacje rozstrzygamy rzucając monetą.
Twierdzenie to daje nam teoretyczne uzasadnienie efektywności głosowania prostego. Widzimy, że mając do dyspozycji zespół klasyfikatorów o rozsądnej skuteczności (przynajmniej przewyższającej 50%) możemy znacznie poprawić wyniki.

Pewnym uogólnieniem tej i poprzedniej (par. 4.1) metody jest następująca metoda: Wybieramy $\alpha \in (0,1]$. Niech $\rho_Z(j);\ j = 1,\ldots,m$ będzie współczynnikiem wsparcia Z dla decyzji dec_j, tzn.

$$\rho_Z(j) = \frac{1}{t}\sum_{k=1}^{t} d_k^j,$$

wtedy decyzja podejmowana jest "α-większością" głosów, czyli gdy

$$\rho_Z(d_Z(x)) > \alpha \quad (d_Z \text{ jak we wzorze 4.2}).$$

W przeciwnym wypadku zespół odmawia podania odpowiedzi i obiekt pozostaje niesklasyfikowany. $\alpha = 1/2$ odpowiada głosowaniu równoprawnemu, a $\alpha = 1$ to jednomyślność. Wybierając α z przedziału $[\frac{1}{2}, 1]$ możemy zdecydować, jak bardzo chcemy być pewni poprawnego sklasyfikowania przychodzącego obiektu. Wybierając większe α uzyskujemy większą pewność decyzji kosztem nierozpoznania, być może, niektórych przypadków.

4.3. Głosowanie ważone

Można też postąpić nieco inaczej – tak jak w AdaBoost. Decyzja jest również podejmowana na zasadzie głosowania, ale tym razem nie każdy głos liczy się jednakowo. W takim przypadku każdemu uczniowi przypisujemy wagę. Niech $Z = \{\langle f_1, w_1\rangle, \ldots, \langle f_t, w_t\rangle\}$, a d_j^k zdefiniowane wzorem 4.1. Wtedy

$$d_Z(x) = dec_p \iff \sum_{k=1}^{t} w_k d_k^p(x) = \max_j \left(\sum_{k=1}^{t} w_k d_k^j(x)\right).$$

Takie podejście pozwala nadać priorytety "lepszym"[4] *uczniom*. Z drugiej strony wymaga ono umiejętności ocenienia (albo przynajmniej porównania) klasyfikatorów bazowych.

Twierdzenie 4.3.1 *Niech $Z = \{\langle f_1, w_1\rangle, \ldots, \langle f_t, w_t\rangle\}$ będzie zespołem t* niezależnych[5] *klasyfikatorów o skuteczności ach $p_1, \ldots, p_t$ odpowiednio. Odpowiedź jest udzielana przez głosowanie ważone. Wtedy skuteczność zespołu jest maksymalizowana przez nadanie wagom w_k wartości:*

$$w_k = \log\left(\frac{p_k}{1-p_k}\right) \tag{4.5}$$

Dowód. Niech $d_1, \ldots, d_t$ będą odpowiedziami poszczególnych klasyfikatorów składowych $f_1, \ldots, f_t$, tzn. $d_k = dec_j \iff d_k^j = 1$. Zbiorem optymalnych Bayesowskich funkcji rozróżnialności[6] jest np.

$$\rho_j(x) = \log(P(dec_j)P(d|dec_j)) \quad j = 1,\ldots,m \tag{4.6}$$

gdzie $d = [d_1, \ldots, d_t]^T$. Z niezależności f_k mamy

[4] w jakimś sensie – np. odpowiedzialnym za większy kawałek przestrzeni

[5] w myśl warunku 4.3

[6] Bayesowska optymalna funkcja rozróżnialności jest obliczana tak, by minimalizować błąd treningowy, gdy stosujemy model probabilistyczny, który polega na założeniach, że klasy przynależności obiektów są niezależnymi zmiennymi losowymi o zadanym rozkładzie. Więcej można się dowiedzieć w [Kun04].

$$
\begin{aligned}
\rho_j(x) &= \log\left(P(dec_j)\prod_{k=1}^{t}P(d_k|dec_j)\right)\\
&= \log P(dec_j) + \log\left(\prod_{k,d_k=dec_j}P(d_k|dec_j)\prod_{k,d_k\neq dec_j}P(d_k|dec_j)\right)\\
&= \log P(dec_j) + \log\left(\prod_{k,d_k=dec_j}p_k\prod_{k,d_k\neq dec_j}(1-p_k)\right)\\
&= \log P(dec_j) + \log\left(\prod_{k,d_k=dec_j}\frac{p_k(1-p_k)}{1-p_k}\prod_{k,d_k\neq dec_j}(1-p_k)\right)\\
&= \log P(dec_j) + \log\left(\prod_{k,d_k=dec_j}\frac{p_k}{1-p_k}\prod_{k=1}^{t}(1-p_k)\right)\\
&= \log P(dec_j) + \sum_{k,d_k=dec_j}\log\frac{p_k}{1-p_k} + \sum_{k=1}^{t}\log(1-p_k).
\end{aligned}
$$

Ostatni składnik powyższego wyrażenia nie zależy od j, dlatego możemy uprościć funkcję rozróżnialności do postaci

$$\rho_j(x) = \log P(dec_j) + \sum_{k=1}^{t} d_k^j \log\frac{p_k}{1-p_k}. \tag{4.7}$$

□

Widzimy teraz, że sposób dobierania wag w algorytmie AdaBoost (por. warunek 3.3) jest uzasadniony. Nie daje to jednak 100% pewności minimalizacji błędu klasyfikacji. Trzeba też brać pod uwagę prawdopodobieństwa wystąpienia obiektu danej klasy.

4.4. Meta-klasyfikator

Można spróbować inaczej. Najpierw trenujmy kilka klasyfikatorów na danych D, a następnie spróbujmy "nauczyć" się interpretować ich wyniki za pomocą np. sieci neuronowej. Być może zespół o pozornie małej skuteczności, tak na prawdę odkrywa pewne zależności w danych, lecz nie jesteśmy w stanie zinterpretować tego za pomocą głosowania? A może potrzeba do tego kilku prostych reguł? Tworzenie klasyfikatora opierającego się na wyjściach innych klasyfikatorów może być ryzykownym przedsięwzięciem. Ryzykownym w sensie takim, że może się źle sprawdzać w rzeczywistości mimo, iż na danych treningowych otrzymamy dużą dokładność. Intuicyjnie, im bardziej skomplikowany układ, tym bardziej podatny na przeuczenie się. Takie postępowanie można porównać do próby rozwiązania problemu bez jego dokładnego zrozumienia. Uczymy się po prostu dawać poprawne odpowiedzi, nie wiedząc tak na prawdę "co się dzieje wewnątrz". Czasem jednak nie innego mamy wyjścia, bo nie umiemy znaleźć prostrzej metody.

A czasem jest zupełnie inaczej. Spójrzmy na taki oto przykład. Mamy system do badania przesiewowego pacjentów w określonym wieku. W razie złych wskaźników pacjent jest uznawany za *chorego* i wysyłany na dokładniejsze badania. Jest kilka modułów (klasyfikatorów

bazowych), dostarczających na wyjściach wielu wskaźników, np. badanie krwi, temperatury ciała, rytmu serca[7]. Za każde badanie odpowiada jakiś klasyfikator (być może zespół). Następnie wyniki przekazywane są do klasyfikatora końcowego, który za pomocą reguł decyzyjnych bądź drzewa[8] wystawia diagnozę (*zdrowy/być może chory*). Tutaj, w przeciwieństwie do tego, o czym mówiliśmy wcześniej, bardzo dobrze znamy problem i takie rozłożenie zadań na poszczególne klasyfikatory wydaję się być całkowicie uzasadnione.

[7] i innych o bardziej specjalistycznych nazwach

[8] czy temperatura ciała jest w przedziale [36,37], a ciśnienie tętnicze krwi wynosi...

Dodatek A

Zespoły klasyfikatorów w rozpoznawaniu wzorców

W tym rozdziale chciałbym przedstawić jeden z nowszych wyników w tej dziedzinie. Simon Günter i Horst Bunke w [Gün04] zaproponowali model oparty na HMM (por. par. 2.5), którego zadaniem było rozpoznawanie pisma odręcznego. Wyzwaniem, któremu próbowali sprostać, było poprawne rozpoznanie dowolnego słowa znajującego się w słowniku. Problem analizy pisma odręcznego podejmowano od wielu lat, jednak nigdy w takiej ogólności. Jest to zadanie, z którym nawet człowiek ma problemy[1], nie mówiąc już o komputerze. Rozważane wcześniej przypadki ograniczały się na ogół jedynie do rozpoznawania pojedynczych znaków[2], z czym nieźle radzą sobie sieci neuronowe.

Do zbudowania zespołu z HMM posłużono się trzema metodami: *boosting-iem, bagging-iem* oraz *wyboru losowej podprzestrzeni*[3]. Parametrami dającymi kryterium porównawcze były: liczba wyrazów w słowniku, rozmiar zbioru treningowego, a także liczność zespołu klasyfikatorów.

A.1. Budowa

Zadanie rozpoznawania pojedynczego wyrazu podzielono na kilka bloków. Najpierw wykonywane są operacje mające na celu "unormowanie" otrzymanego słowa – reprezentowanego jako obraz o rozdzielczości 300dpi. Następnie obiekt ten jest przekształcany w wektor cech, na podstawie którego dokonywana jest jego późniejsza klasyfikacja (zobacz Rysunek A.1). System zawiera trzy podstawowe moduły, które są odpowiedzialne za: *przetwarzanie wstępne* (ang. preprocessing), *ekstrakcję cech* oraz *klasyfikację*.

A.1.1. Preprocesor

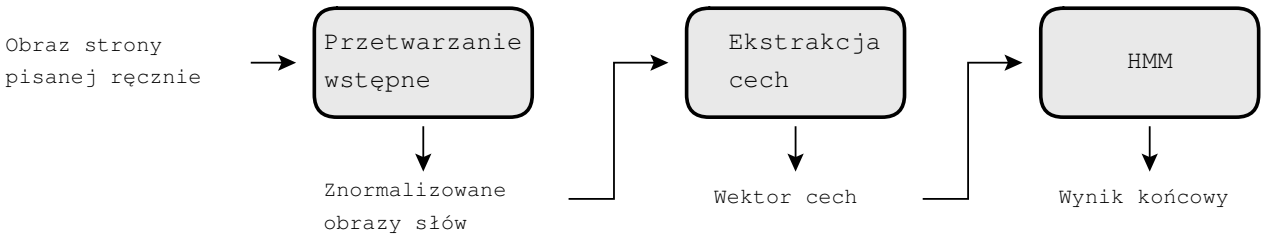
Przed analizą obrazu preprocesor wykonuje szereg czynności:

- *korekta przekrzywienia* – słowo jest obracane tak, by było równoległe do osi x
- *korekta pochylenia* – pochylone pismo (kursywa) zostaje "wyprostowane"

[1] spróbuj przeczytać receptę od lekarza

[2] np. czytanie kodów z powodzeniem stosowane dziś na pocztach, czy wartości wpisywanych na czekach, bądź też różnego rodzaju formularzach – gdzie znaki znajdują się w odpowiednich kratkach

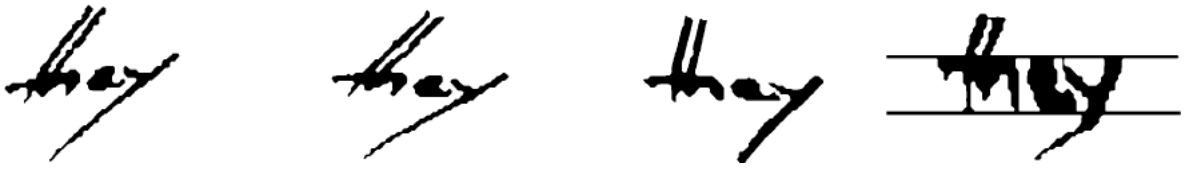
[3] ang. *random subspace method* – każdy klasyfikator składowy podczas trenowania i testowania, posługuje się jedynie podzbiorem atrybutów; liczba atrybutów jest ustalona i są one za każdym razem wybierane losowo spośród wszystkich dostępnych; wybory atrybutów dla poszczególnych klasyfikatorów są niezależne



Rysunek A.1: Schemat działania systemu

- *normowanie wysokości liter* – duże i małe litery zostają przeskalowane tak, by dotykały linii górnej i środkowej odpowiednio

Działanie preprocesora przedstawia rysunek A.1.1



Rysunek A.2: Przetwarzanie wstępne: *obrót, korekta kursywy, korekta wysokości liter*

A.1.2. Wektory cech

Aby wyekstrahować wektor cech przeskanowano dane słowo od lewej do prawej ramką szerokości jednego piksela i wysokości całego słowa. Dla każdego położenia ramki obliczono szereg wielkości, takich jak: liczba czarnych pikseli, środek ciężkości, górna i dolna granica tekstu, wektory przesunięcia górnej i dolnej granicy[4]. W efekcie, dla każdego słowa, dostajemy szereg dziewięcio - elementowych wektorów – po jednym w każdym ustawieniu ramki.

A.1.3. HMM

Rozwiązując problem klasyfikacji konstruujemy HMM dla każdej klasy decyzyjnej. W tym przypadku, takie bezpośrednie podejście jest z praktycznego punktu widzenia niewygodne – musielibyśmy zbudować tyle modeli, ile wyrazów mamy w słowniku. Dlatego postanowiono skonstruować jeden model dla każdego znaku (litery). Każdy HMM składa się z czternastu liniowo ułożonych stanów (w każdym mamy dwie możliwości: możemy w nim pozostać lub przejść do sąsiedniego). Prawdopodobieństwo wystąpienia danej sekwencji stanów $S_1, \ldots, S_n$ jest dane jako iloczyn prawdopodobieństw możliwych obserwacji o_i w stanie S_i oraz prawdopodobieństw przejść pomiędzy stanami S_i i S_{i+1}. Prawdopodobieństwo wystąpienia określonej sekwencji obserwacji $o_1, \ldots, o_n$ dla danego modelu uzyskujemy biorąc największe prawdopodobieństwo spośród wszystkich możliwych sekwencji stanów[5]. Liczba stanów, macierze przejść oraz prawdopodobieństwa obserwacji zostały dobrane doświadczalnie.

[4] por. [Gün04]
[5] Viterbi recognition

A.1.4. Eksperymenty

Przeprowadzono szereg prób, porównując różne metody budowy zespołów dla różnych rozmiarów słownika, danych testowych oraz liczby klasyfikatorów składowych. Do wyłuskania wyniku użyto dwóch metod: głosowania prostego[6] i głosowania ważonego[7]. Testy z różnymi wielkościami zbioru treningowego wykazały, że Bagging i AdaBoost na mniejszych zbiorach wykazują gorszą skuteczność niż klasyfikator bazowy. Skuteczność rośnie wraz ze wzrostem wielkości zbioru treningowego – dla 1000 i więcej przykładów obie metody dają lepsze rezultaty niż wspomniany pojedynczy klasyfikator. Dla mniejszych zbiorów metoda Bagging daje nieco lepsze rezultaty niż AdaBoost, która to z kolei wygrywa na dużych zestawach danych. Sposoby głosowania zaczynają odgrywać znaczącą rolę w przypadku metody wybierania losowej podprzestrzeni. Okazuje się ona być bardziej skuteczna niż Bagging i Boosting dla mniejszych zbiorów treningowych.

Wyniki poszczególnych metod (skuteczność) dla różnych zbiorów treningowych:

		Głosowanie proste			Głosowanie ważone		
Rozm. zbioru	Bazowy	Bagging	AdaBoost	WLP	Bagging	AdaBoost	WLP
300	54,10	53,38	52,63	56,69	52,44	51,59	57,1
1000	61,33	63,16	62,23	63,26	62,88	61,6	63,54
3000	65,9	66,73	66,7	66,42	66,54	66,6	67,26
9861	66,23	67,35	68,29	67,54	67,92	68,86	68,67

Tabela A.1.1
Bazowy – klasyfikator bazowy, *WLP* – metoda wyboru losowej podprzestrzeni

W kolejnej serii eksperymentów zmieniano wielkość słownika. Podobnie jak poprzednio dla małego słownika i głosowania ważonego metoda losowego wyboru podprzestrzeni okazała się najlepsza, jednak znacząco przegrywa z AdaBoost i Bagging, gdy słownik liczy ponad 1500 wyrazów, a głosowanie odbywa się w sposób prosty.

Wyniki poszczególnych metod dla różnych słowników:

		Głosowanie proste			Głosowanie ważone		
Rozm. słown.	Bazowy	Bagging	AdaBoost	WLP	Bagging	AdaBoost	WLP
651	79,51	81,63	82,28	80,56	81,14	81,46	81,95
998	77,01	79,84	81,3	79,19	79,84	80,81	81,14
1518	74,15	77,89	79,51	75,28	77,89	79,19	78,21
2296	68,78	73,17	75,61	69,433	73,5	75,61	73,82

Tabela A.1.2

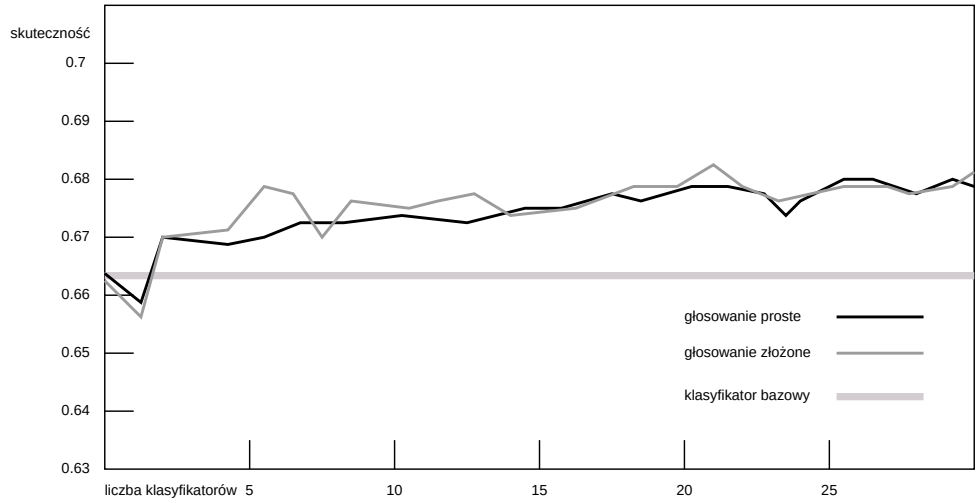
Trzecia faza testów pokazała różnice wynikające z liczby klasyfikatorów w zespole. Na

[6] bierzemy pod uwagę jedynie te klasy, które otrzymały najwyższy współczynnik wsparcia dla poszczególnych klasyfikatorów; wybieramy z nich tę która otrzymała najwyższą liczbę głosów; w przypadku konfliktu posługujemy się zasadą maksimum, stosowaną dla konkurujących ze sobą klas tzn. wybieramy tę, która otrzymała najwyższą liczbę punktów dla wszystkich klasyfikatorów

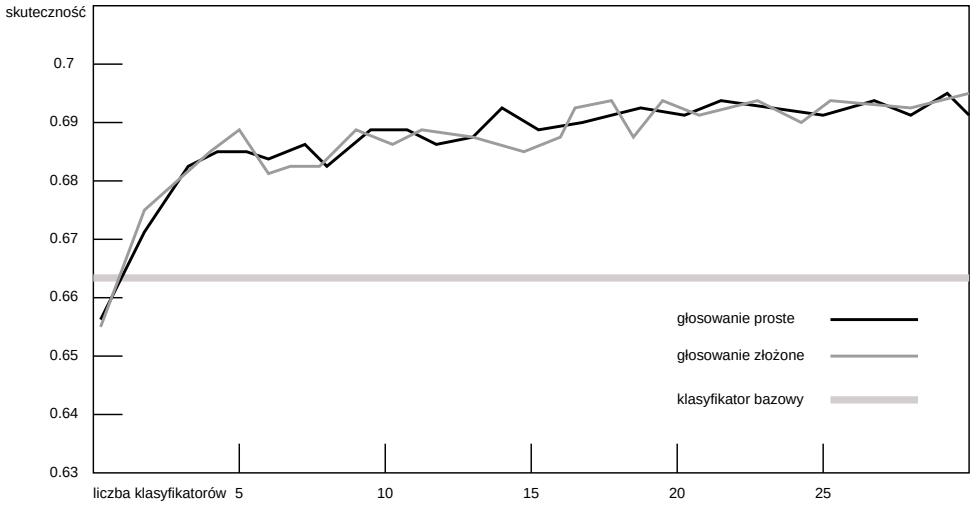
[7] podobnie interesują nas jedynie klasy o najwyższych współczynnikach wsparcia, ale tym razem każdy klasyfikator ma przypisaną wagę; wygrywa klasa o najwyższej sumie wag; wagi obliczane są za pomocą algorytmu genetycznego na podstawie skuteczności zespołu na danych treningowych

początku możemy zauważyć, że głosowanie ważone zachowuje się niestabilnie w porównaniu z głoswaniem prostym, zarówno w metodzie Bagging jak i AdaBoost (Rysunki A.3, A.4). W obu tych przypadkach zespół dość szybko zaczął dawać lepsze wyniki niż klasyfikator bazowy. Bagging dawał stopniowy, jednostajny wzrost skuteczności dla zespołów wielkości od 0 do 20 członków, natomiast skuteczność metody AdaBoost bardzo szybko rosła w przedziale $0 - 5$, stabilizując się w 10. Inaczej sytuacja wygląda dla metody wyboru losowej podprzestrzeni. Tutaj dopiero przy 6 klasyfikatorach otrzymano skuteczność wyższą niż klasyfikatora bazowego oraz sposób głosowania miał znaczący wpływ na wyniki (Rysunek A.5). Głosowanie ważone prawie przez cały czas dawało lepszą skuteczność, która sięgnęła nawet 69.35% – najwyższa otrzymana podczas tych eksperymentów.

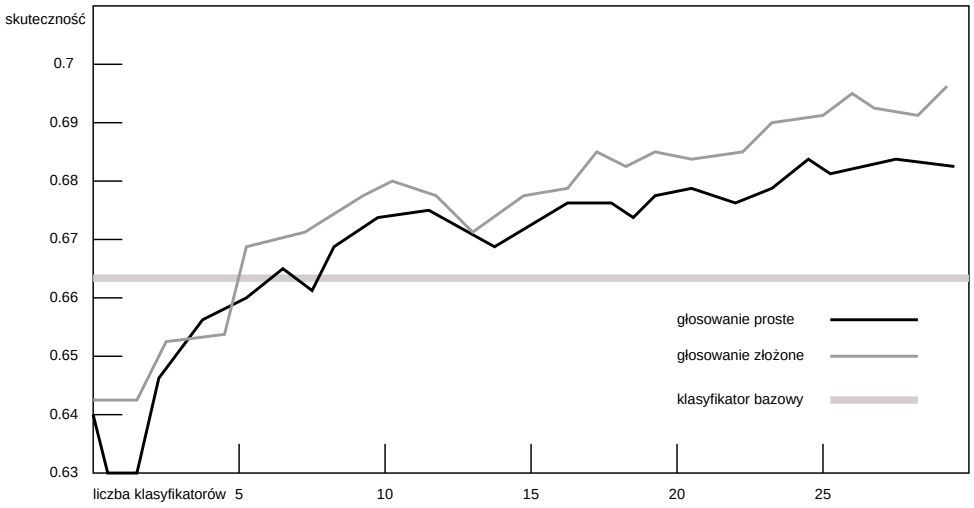
Osiągnięty wynik jest bardzo satysfakcjonujący. Widać duże możliwości i potencjał w zastosowaniu zespołu klasyfikatorów do rozpoznawania pisma. Autorzy wierzą, że odpowiednio dobierając parametry oraz ulepszając nieco metody łączenia klasyfikatorów można dojść nawet do 80%.



Rysunek A.3: Bagging



Rysunek A.4: AdaBoost



Rysunek A.5: Wybór losowej podprzestrzeni

Bibliografia

[Kun04] Lidmila I. Kunchewa, *Combining Pattern Classifiers, Methods and Algorithms*, John Wiley & Sons, Inc., Hoboken, New Jersey 2004.

[Mitch97] Tom Mitchel, *Machine Learning*, McGraw Hill, USA 1997.

[Schep03] Robert E. Schapire, *The Boosting Approach to Machine Learning, An Overview*, Nonlinear Estimation and Classification, Springer, 2003

[Fre99] Yoav Freund, Robert E. Schapire, *A Short Introduction to Boosting* Journal of Japanese Society of Artificial Inteligence 14(5) 771-780, September 1999

[Gün04] Simon Günter, Horst Bunke, *Off-line cursive handwritng recognition, ussing multiple classifier systems - on the influance of vocabulary, ensemble and trainnig set size*, University of Bern, Switzerland 2004.