Instytut Podstawowych Problemów Techniki
Polskiej Akademii Nauk

Efektywne metody selekcji cech i rozwiązywania problemu wieloklasowego w nadzorowanej klasyfikacji danych

mgr Wiesław Chmielnicki

Rozprawa doktorska napisana pod kierunkiem
Prof. dr hab. inż. Katarzyny Stąpor

Kraków 2012

Spis treści

Streszczenie 5

Abstract 6

Wykaz oznaczeń stosowanych w rozprawie 7

Tezy i cele pracy 9

1. Opis problemu i aktualny stan wiedzy 13

1.1. Klasyfikacja nadzorowana 14

1.2. Podstawowe typy klasyfikatorów 16

1.2.1. Klasyfikator RDA 25

1.2.2. Klasyfikator SVM 27

1.2.3. Klasyfikatory generatywne i dyskryminatywne 32

1.2.4. Łączenie klasyfikatorów 33

1.2.5. Metody oceny jakości klasyfikatorów 38

1.2.6. Estymacja bootstrapowa 42

1.3. Selekcja cech 44

1.3.1. Metody rankingowe 46

1.3.2. Metody opakowane 48

1.3.3. Metody SFFS i SFBS 50

1.3.4. Metoda BDS 51

1.3.5. Metody wbudowane 52

1.3.6. Metody oparte na modyfikacji funkcji kryterialnych 52

1.4. Specyfika problemu wieloklasowego 54

1.4.1. Strategia jeden przeciw pozostałym 55

1.4.2. Strategia jeden przeciw jednemu 56

1.4.3. Strategia binarnych drzew decyzyjnych 57

1.4.4. Technika ECOC 59

2. Opis problemów wybranych do testowania zaproponowanych algorytmów 65

2.1. Rozpoznawanie mowy 67

2.2. Identyfikacja autorstwa tekstu 68

2.3. Rozpoznawanie odręcznie pisanych cyfr 69

2.4. Rozpoznawanie gestów 71

2.5. Przewidywanie trzeciorzędowej struktury białek 73

3. Efektywne podejście do problemu wieloklasowego 77
3.1. Nowa strategia budowania drzew decyzyjnych 79
3.2. Metoda przecięć 84
3.3. Tworzenie efektywnych kodów ECOC 89
4. Efektywne metody selekcji cech 95
4.1. Szybkie algorytmy F-SFS i F-SBS 96
4.2. Metoda oparta na podobieństwie klas CS-SFS 99
4.3. Zmodyfikowana metoda oparta na podobieństwie klas CSO-SFS 101
4.4. Wyniki eksperymentów 103
5. Hybrydowy klasyfikator SVM-RDA 109
5.1. Opis klasyfikatora 109
5.2. Opis przeprowadzonej procedury testowej 112
5.3. Wyniki eksperymentów 115
6. Podsumowanie i wnioski 119
6.1. Metody podejścia do problemu wieloklasowego 119
6.2. Nowe metody tworzenia kodów ECOC 120
6.3. Hybrydowy klasyfikator SVM-RDA 121
6.4. Metody selekcji cech 122
6.5. Wnioski 122
Bibliografia 125

Streszczenie

W ostatnich latach metody automatycznej klasyfikacji obiektów rozwijane są bardzo dynamicznie. Zadania stawiane przez systemami komputerowymi są coraz bardziej skomplikowane. Rośnie zarówno liczba klas w zadaniu klasyfikacji jak i liczba cech branych pod uwagę przez klasyfikator. Algorytmy rozwiązujące takie problemy są coraz bardziej kosztowne obliczeniowo. To wszystko stało się motywacją do podjęcia próby opracowania efektywnych metod rozwiązywania problemu wieloklasowego oraz efektywnych metod selekcji cech.

W pierwszej części niniejszej pracy zostały krótko zaprezentowane znane z literatury algorytmy klasyfikacji danych oraz metody selekcji cech. Przedstawione zostały ich wady i zalety. Szczególną uwagę poświęcono złożoności obliczeniowej opisywanych algorytmów w kontekście odpowiednio: dużej ilości klas i dużej ilości cech. Pokazano, że pewne metody stają się w takich wypadkach całkowicie nieefektywne, co oznacza, że nie mogą być w praktyce stosowane.

Jedną z metod rozwiązywania problemu wieloklasowego jest jego podział na problemy binarne. Sposób tego podziału ma zasadniczy wpływ zarówno na złożoność klasyfikatora jak i na jego błąd klasyfikacji. W pracy przedstawiono kilka technik, które pozwalają zmniejszyć ten błąd, nie zwiększając jednocześnie znacząco kosztu obliczeniowego modyfikowanego algorytmu. Techniki te oparte o binarne drzewa decyzyjne BDT i o wyjściowe kody samokorygujące ECOC zostały przetestowane na kilku różnych bazach danych.

W kolejnym rozdziale zostały przedstawione nowe algorytmy selekcji cech. Algorytmy te, będące połączeniem metod rankingowych i obudowanych, pozwoliły na efektywne rozwiązanie problemu poprzez wykorzystanie zalet obu metod. W pracy zostało również omówione generatywne i dyskryminatywne podejście do zadania klasyfikacji danych. Zaproponowano hybrydowy klasyfikator, który łącząc zalety obu metodologii, pozwolił poprawić wyniki osiągane przez każdy z klasyfikatorów osobno. Poprawa ta okazała się statystycznie istotna.

Wszystkie zaproponowane algorytmy zostały zaimplementowane przez autora w języku C++ oraz były testowane na pięciu różnych bazach danych, reprezentujących całkowicie różne dziedziny nauki i techniki.

Abstract

In recent years the automatic classification methods are dynamically developing. The tasks which computer systems should solve are more and more complex. Both, the number of classes and the number of features which are taken into account by the classifier are growing. The algorithms are more and more computationally expensive. It has been an author's motivation to make an attempt to develop efficient methods of solving multi-class problems and to develop efficient feature selection methods.

In the first part of the work the feature selection and classification methods known from the literature have been presented. Their advantages and disadvantages have been shown. The particular attention has been taken to the computational complexity of the described algorithms in the context of the large number of classes and the large number of features. It has been shown that in such cases these algorithms are completely inefficient and that in fact they cannot be used in practice.

One of the methods to solve a multi-class problem is to divide it to several binary problems. The way it is divided impacts both computational complexity of the classifier and its classification error. In the work there are several techniques presented, which allow to decrease this error. However these techniques do not increase the computational cost of the modified algorithms. The techniques based on binary decision trees BDT and error correcting output codes ECOC have been tested on several databases.

In the next chapter there are new feature selection algorithms introduced. These algorithms which are a combination of filter and wrapper methods use the advantages of both techniques. It allows solving the feature selection problem efficiently. The generative and discriminative methodology has been discussed in the work as well. The hybrid classifier has been presented which combines the advantages of both methods. The results of this classifier are better than each of the classifiers alone. It has been shown that this result is statistically relevant.

All proposed algorithms have been implemented by author in C++ language and they have been tested using five different databases. These databases represent several completely different scientific domains.

abbrlist

Wykaz oznaczeń stosowanych w rozprawie

Acc	skuteczność klasyfikatora (ang. accuracy)
C	współczynnik kary (ang. soft margin parameter)
c	liczba klas (ang. number of classes)
d_i	funkcja dyskryminacyjna dla *i-tej* klasy (ang. discriminant function)
d_H	odległość Hamminga (ang. Hamming distance)
f	cecha (ang. feature)
FN	nieprawidłowe wskazanie drugiej klasy (ang. false negative)
FP	nieprawidłowe wskazanie wyróżnionej klasy (ang. false positive)
$J(S)$	funkcja kryterialna (ang. criteria function)
$K(x_i, x)$	funkcja jądrowa (ang. kernel function)
n_i	liczba próbek *i-tej* klasy (ang. number of samples)
N	liczba próbek w zbiorze uczącym (ang. number of samples in learning set)
l_i	etykieta klasy (ang. class label)
L	zbiór etykiet klas (ang. label set)
L_{ij}	funkcja straty (ang. loss function)
O	zbiór obiektów (ang. object set)
$P(c_i\|x)$	prawdopodobieństwo *a posteriori* (ang. *a posteriori* probability)
$p^*(\alpha)$	percentyl rzędu alfa (ang. percentile)
Q	współczynnik poprawnych klasyfikacji (ang. quality coefficient)
R_r	współczynnik odrzucania (ang. rejection rate)
$r_i(x)$	średnia wartość straty (ang. mean loss value)
S	podzbiór cech (ang. feature subset)
S_n	czułość klasyfikatora (ang. sensivity)
S_p	specyficzność klasyfikatora (ang. specificity)
TN	prawidłowe wskazanie drugiej klasy (ang. true negative)
TP	prawidłowe wskazanie wyróżnionej klasy (ang. true positive)
U	zbiór uczący (ang. learning set)
$w_i(x)$	waga głosu klasyfikatora (ang. weight of the classifiers vote)
X	przestrzeń reprezentacji/przestrzeń cech (ang. feature space)
Z	zbiór decyzji klasyfikatora (ang. decision set)

α_i współczynniki Lagranga (ang. Langrange multipliers)

γ parametr jądra dla funkcji RBF (ang. kernel parameter for RBF function)

λ parametr regularyzacji dla klasyfikatora RDA (ang. regularization parameter)

μ_k wektor średni (ang. mean vector)

$\hat{\mu}_k$ estymowany wektor średni (ang. estimated mean vector)

μ_A wartość oczekiwana (ang. expected value)

ψ reguła decyzyjna (ang. decision rule)

Σ macierz kowariancji (ang. covariance matrix)

$\hat{\Sigma}$ estymowana macierz kowariancji (ang. estimated covariance matrix)

σ^2 wariancja (ang. variance)

Tezy i cele pracy

Klasyfikacja danych to część niezwykle dynamicznie rozwijającej się w ostatnich latach dziedziny nauki zwanej rozpoznawaniem obiektów (ang. pattern recognition). Przy czym pojęcie obiekt jest rozumiane bardzo szeroko. Jako obiekt traktować możemy na przykład: pismo, rysunek techniczny, mowę, elektrokardiogram, obraz, zestaw danych z czujników czy nawet sekwencję aminokwasów. Rozpoznawanie obiektów zajmuje się różnymi aspektami projektowania i tworzenia automatycznych systemów klasyfikacji. Ich trzonem jest klasyfikator, czyli algorytm, który na podstawie danych wejściowych opisujących obiekt potrafi przyporządkować go do określonej klasy.

Najdoskonalszym jak dotąd przykładem systemu rozpoznającego obiekty jest ludzki umysł. W naturalnym procesie uczenia się z przykładów, dochodzi w nim do wytworzenia pewnych algorytmów decyzyjnych, które pozwalają później na samodzielne klasyfikowanie i rozpoznawanie bardzo złożonych obiektów. Już nawet kilkuletnie dziecko potrafi bezbłędnie rozpoznawać obiekty, które dla zaawansowanego systemu automatycznego są dużym wyzwaniem. Takim przykładem może być chociażby rozpoznawanie twarzy człowieka czy też identyfikacja ludzkiego głosu.

Automatyczne systemy rozpoznawania próbują naśladować sposób uczenia się mózgu poprzez próbę stworzenia odpowiedniego algorytmu i znalezienia reguł decyzyjnych na podstawie zbioru przykładów (próbek). W rezultacie tego procesu system stara się wypracować pewne reguły, które nie zostały bezpośrednio wprowadzone do niego. Reguły te mogą być reprezentowane na wiele różnych sposobów. Na przykład jako rozkłady prawdopodobieństwa, gramatyki formalne, współczynniki funkcji, czy struktury danych.

W zależności od sposobu uczenia się możemy mówić o klasyfikacji nadzorowanej (ang. supervised classification) lub klasyfikacji nienadzorowanej (ang. unsupervised classification). W pierwszym przypadku do procesu uczenia się wykorzystujemy zbiór obiektów, których przypisanie do klas (ich etykiety) znamy. Zbiór ten nazywamy zbiorem uczącym (ang. learning set) lub zbiorem treningowym (ang. training set).

W zadaniu klasyfikacji nienadzorowanej nie dysponujemy zbiorem uczącym, a jedynie pewną określoną miarą podobieństwa (ang. similarity measure). W tym wypadku zadanie klasyfikacji polega na rozdzieleniu obiektów na podzbiory, tak aby obiekty, w każdym podzbiorze były do siebie maksymalnie podobne (w sensie tej miary podobieństwa). W niniejszej pracy skupimy się wyłącznie na klasyfikacji nadzorowanej.

Najprostszym zadaniem klasyfikacji jest sytuacja, w której musimy przydzielić obiekt do jednej z dwóch klas. Na przykład należy dokonać podziału obiektów na czarne i na białe, oddzielić kwadraty od trójkątów, odróżnić zero od jedynki, czy też zakwalifikować elektrokardiogram jako prawidłowy albo nieprawidłowy. Jednak większość problemów, z jakimi spotykamy się na co dzień, to problemy wieloklasowe, czyli takie, w których mamy do czynienia z więcej niż dwoma klasami obiektów.

Problemy te są zwykle dużo bardziej skomplikowane. Większa ilość klas oznacza zazwyczaj mniejszą liczbę próbek każdej klasy w zbiorze uczącym, większą liczbę cech obiektu, którą musimy uwzględnić, aby poprawnie utworzyć reguły decyzyjne klasyfikatora. Pojawia się również problem braku odpowiedniej wiedzy. Na przykład stosunkowo łatwo opracować reguły pozwalające odróżnić elektrokardiogram prawidłowy od nieprawidłowego. Jednak już przyporządkowanie nieprawidłowego zapisu EKG do konkretnego schorzenia jest niezwykle trudne. Z poprawną klasyfikacją mają problem nawet lekarze specjaliści.

Dodatkowym wyzwaniem jest także to, że część klasyfikatorów, to klasyfikatory binarne. Oczywiście problem wieloklasowy możemy rozłożyć na pewną liczbę problemów binarnych. Na przykład, podział obiektów na cztery klasy: zielone kwadraty, czerwone kwadraty, zielone koła i czerwone koła możemy rozłożyć na dwa etapy. W pierwszym podzielimy obiekty na zielone i czerwone, a w drugim oddzielimy kwadraty od trójkątów.

W literaturze opisano wiele technik podziału problemu wieloklasowego na problemy binarne. Jednak wiele z nich staje się całkowicie nieefektywna, gdy liczba klas znacząco wzrasta. Na przykład jedna z najpopularniejszych technik: metoda *"jeden przeciw jednemu"* OVO (ang. One Versus One) polega na konstruowaniu klasyfikatorów dla wszystkich możliwych par klas. W przypadku n klas tworzone jest zatem $O(n^2)$ podproblemów. Łatwo zauważyć, że w przypadku dużej liczby klas to rozwiązanie będzie bardzo kosztowne obliczeniowo.

Możemy jednak skorzystać z innych metod podziału na problemy binarne, które pozwolą na istotne zmniejszenie liczby podproblemów. Na przykład metoda *"jeden przeciw pozostałym"* OVR (ang. One Versus Rest), polegająca konstruowaniu klasyfikatorów oddzielających każdą z klas od pozostałych, wymaga utworzenia już tylko $O(n)$ podproblemów. Metody wykorzystujące binarne drzewa decyzyjne BDT (ang. Binary Decision Trees) pozwalają zmniejszyć tę liczbę nawet do $O(log_2(n))$.

Problemem jest jednak, że większość z tych metod powoduje znaczny wzrost błędu klasyfikacji. Dlatego też istotne byłoby wskazanie takich technik podziału, które pozwoliłyby na zmniejszenie ilości klasyfikatorów binarnych przy jednoczesnym zachowaniu dobrego wyniku klasyfikacji.

Ciekawą metodą jest pochodząca z telekomunikacji technika kodów samokorygujących ECC (ang. Error Correcting Codes). Po niewielkiej modyfikacji może ona zostać zastosowana

do podziału problemu wieloklasowego. Poprzez odpowiednie kodowanie klas definiuje ona podział całego zbioru na dwa podzbiory. Okazuje się, że taka procedura pozwala na stworzenie efektywnego klasyfikatora wieloklasowego, przy czym błąd klasyfikacji pozostaje na stosunkowo niskim poziomie. Możemy zatem postawić następującą tezę:

— Odpowiednie techniki podziału problemu wieloklasowego na problemy binarne na przykład technika kodowania pozwalają na znaczące obniżenie błędu klasyfikacji.

Celem niniejszej pracy będzie wskazanie nowych metod, które pozwolą na efektywne rozwiązywanie problemów, zwłaszcza takich, w których liczba klas jest bardzo duża. Metody te, dzieląc odpowiednio problem wieloklasowy na podproblemy binarne, pozwolą na poprawę szybkości działania klasyfikatora wieloklasowego jednocześnie zmniejszając jego błąd klasyfikacji.

W szczególności przedstawiona zostanie technika oparta na samokorygujących kodach wyjściowych ECOC (ang. Error Correcting Output Codes). Omówione zostaną jej wady i zalety oraz wskazane zostaną takie strategie budowania kodów, które umożliwiają poprawienie osiąganego przez klasyfikator wieloklasowy wyniku.

W pracy zostaną zaprezentowane również różne podejścia do problemu klasyfikacji: generatywne i dyskryminatywne. Każde z nich ma swoje wady i zalety. Pokazane zostanie, że odpowiednie połączenie obu podejść może znacząco wpłynąć na poprawę wyniku hybrydowego klasyfikatora, dzięki wykorzystaniu silnych stron obu metod.

W zadaniach klasyfikacji, z którymi się spotykamy w życiu, bardzo często nie wiadomo, które cechy obiektu są istotne, a które możemy pominąć. Dlatego też zwykle wybieramy możliwie szeroki wachlarz cech tak, aby maksymalnie zwiększyć efektywność klasyfikatora i nie pominąć żadnej potencjalnie istotnej cechy. O ile jeszcze dwadzieścia lat temu w zadaniach klasyfikacji używane było średnio kilkanaście do kilkudziesięciu cech, to obecnie nawet kilka czy kilkanaście tysięcy cech nie należy do rzadkości.

W praktyce jednak okazuje się, że duża liczba cech nie zawsze zapewnia wzrost odróżnialności klas. Wskutek ich korelacji może dojść do sytuacji, w której połączenie dwóch bardzo dobrych pod względem odróżnialności cech, nie spowoduje poprawy efektywności klasyfikatora, a doprowadzi nawet do jej pogorszenia. Dodatkowo duża liczba cech w klasyfikatorze powoduje wzrost ilości wolnych parametrów koniecznych do oszacowania, zatem i wzrost złożoności klasyfikatora. Zwiększa to niebezpieczeństwo przeuczenia, a w konsekwencji spadku zdolności uogólniających klasyfikatora.

Kolejnym bardzo ważnym problemem jest występowanie zjawiska ”przekleństwa wymiarowości” (ang. curse of dimensionality) [73]. Termin ten oznacza, że chcąc zapewnić taką samą dokładność estymacji w przestrzeni cech, musimy odpowiednio zwiększyć liczbę obiektów w zbiorze uczącym. Okazuje się, że liczba ta musi rosnąć wykładniczo wraz

ze wzrostem wymiarowości przestrzeni cech. Często jednak powiększenie ilości obiektów w zbiorze uczącym jest niemożliwe lub bardzo kosztowne.

Dlatego też w przypadku wielu tych problemów koniecznym staje się zastosowanie odpowiedniego algorytmu wyboru (selekcji) cech. Sposób wyboru tych cech jest niezwykle istotny dla uzyskania dobrego wyniku klasyfikacji. To kolejne zagadnienie, którym zajmuje się niniejsza rozprawa.

Dlatego też w pracy przedstawione zostaną również różne algorytmy selekcji cech oraz ich wpływ na jakość klasyfikacji. Niektóre z tych algorytmów stają się całkowicie nieefektywne jeśli liczba klas jest zbyt duża. Powoduje to, że nie mogą być one stosowane w praktyce. Niniejsza praca zajmuje się także i tym problemem. Zaproponowane zostaną pewne modyfikacje algorytmów, które pozwolą na ich wykorzystanie. Modyfikacje te nie tylko powodują znaczący wzrost szybkości algorytmów, ale nawet nieco poprawiają osiągane przez nie wyniki.

Rozprawa niniejsza składa się z sześciu rozdziałów. Rozdział pierwszy wprowadza w tematykę automatycznego rozpoznawania obiektów. Przedstawia podstawowe definicje pojęć związanych z tematyką pracy oraz opisuje aktualny stan wiedzy na ten temat. Rozdział drugi opisuje problemy wybrane do testowania opracowanych metod. W rozdziale trzecim omówione zostaną zaproponowane metody podejścia do problemu wieloklasowości. Zajmiemy się także metodami łączenia klasyfikatorów binarnych.

Czwarta część to propozycje nowych i zmodyfikowanych algorytmów selekcji cech, które można skutecznie wykorzystać przy rozwiązywaniu problemu wieloklasowego. Rozdział piąty pokazuje w jaki sposób możemy wykorzystać zalety różnych metod podejścia do problemu klasyfikacji: generatywnej i dyskryminatywnej do poprawienia wyniku uzyskanego przez łączony hybrydowy klasyfikator. Na koniec rozdział szósty zawiera podsumowanie opisanych metod oraz wnioski z przeprowadzonych prac badawczych.

1. Opis problemu i aktualny stan wiedzy

Zadaniem niniejszego rozdziału jest wprowadzenie w tematykę automatycznego rozpoznawania obiektów (ang. pattern recognition), formalne przedstawienie podstawowych definicji i pojęć oraz omówienie aktualnego stanu wiedzy w tej dziedzinie. Między innymi zostanie formalnie zdefiniowane pojęcie zadania nadzorowanej klasyfikacji danych oraz pojęcie klasyfikatora.

Następnie przedstawione zostaną podstawowe typy klasyfikatorów: drzewa decyzyjne, sieci neuronowe, klasyfikatory minimalno-odległościowe oraz klasyfikatory bayesowskie. Ze względu na obszerność tematu opis ten będzie dość skrótowy. Bardziej szczegółowo zostaną opisane dwa klasyfikatory wykorzystane w badaniach: maszyna wektorów wspierających SVM (ang. Support Vector Machine) oraz regularyzowana analiza dyskryminacyjna RDA (ang. Regularized Discriminant Analysis).

Tematem kolejnego podrozdziału będzie generatywne i dyskryminatywne podejście do problemu klasyfikacji danych. Przedstawione zostanie zagadnienie tworzenia klasyfikatorów hybrydowych. Omówione zostaną różne techniki i metody łączenia różnych klasyfikatorów znane z literatury przedmiotu. Na koniec przedstawione zostaną metody oceny jakości klasyfikatorów.

W drugiej części rozdziału zdefiniowane zostanie pojęcie selekcji cech. Omówione zostaną trzy główne podejścia do tego problemu: metody rankingowe, opakowane i wbudowane. Przedstawione zostaną przykłady ich zastosowania. W szczególności omówione zostaną algorytmy sekwencyjnego przeszukiwania w przód SFS (ang. Sequential Forward Selection) oraz sekwencyjnego przeszukiwania w tył SBS (ang. Sequential Backward Selection) jak również ich modyfikacje.

W ostatniej części zostanie omówiony problem wieloklasowy. Pokazane zostaną znane z literatury metody rozwiązywania go, jak również ich wady i zalety. Szczególna uwaga zostanie poświęcona efektywności poszczególnych rozwiązań w kontekście dużej ilości klas. Dość dokładnie przedstawiona zostanie zapożyczona z telekomunikacji metoda wyjściowych kodów samokorygujących ECOC (ang. Error Correcting Output Codes).

1.1. Klasyfikacja nadzorowana

Tak jak zostało już wcześniej wspomniane, z klasyfikacją nadzorowaną (ang. supervised classification) mamy do czynienia, gdy oprócz zbioru obiektów, które mamy rozpoznawać, istnieje także podzbiór, w którym znamy przypisanie obiektów do poszczególnych klas. Zatem formalnie:

Oznaczmy przez O zbiór wszystkich obiektów. Zakładamy, że na tym zbiorze istnieje pewien podział na c rozłącznych klas: $O_1, \ldots, O_c$, czyli istnieje funkcja Φ:

$$\Phi : O \to L = \{1, \ldots, c\}, \quad \forall\, o \in O \quad \Phi(o) = i \iff o \in O_i \tag{1.1}$$

Funkcja ta dokonuje odwzorowania zbioru O w zbiór indeksów klas L. Indeks ten będziemy nazywali etykietą klasy. Oczywiście nie znamy reguł, które rządzą przynależnością do danej klasy, co znaczy, że funkcja Φ jest nieznana. W zadaniu klasyfikacji nadzorowanej znamy tylko mały podzbiór zbioru O w postaci zbioru uczącego U.

$$U = \{(x_i = opis(o_i),\ l_i = \Phi(o_i))\}_{i=1}^{N} \tag{1.2}$$

gdzie $opis(o_i)$ przyporządkowuje obiektowi z przestrzeni O jego reprezentację, l_i jest etykietą klasy dostarczoną przez funkcję Φ, a N ilością obiektów w zbiorze U.

W niniejszej pracy będziemy korzystać wyłącznie z reprezentacji wektorowej, w której obiekt jest opisywany przez wektor cech (ang. feature vector). Przez cechy będziemy rozumieli pewne mierzalne wielkości poddające się naszej obserwacji lub pomiarowi. Zatem przestrzeń reprezentacji nazywana będzie przestrzenią cech. W przeprowadzonych badaniach wszystkie cechy były liczbami rzeczywistymi.

Wprowadźmy dodatkowo pojęcie zbioru decyzji $Z = \{0, 1, \ldots, c\} = L \cup \{0\}$, będącego rozszerzeniem zbioru etykiet L o specjalną etykietę 0 oznaczającą brak decyzji. Ta dodatkowa etykieta jest często bardzo użyteczna, ponieważ w pewnych zadaniach klasyfikacji lepiej jest nie podejmować żadnej decyzji. Jest to równoznaczne z odpowiedzią *"nie wiem"*. Mówimy wtedy, że klasyfikator odrzucił dany obiekt.

W wielu wypadkach taka odpowiedź jest lepsza niż podanie niepewnego wyniku. Przykładem takiej sytuacji może być system diagnozujący nowotwory. Jeśli odpowiedź *"tak"* oznaczająca chorobę będzie błędna, to narazimy pacjenta na niepotrzebne i kosztowne leczenie. Odpowiedź *"nie"* będzie jeszcze gorsza w skutkach, ponieważ może narazić pacjenta na utratę życia (w przypadku, gdy będzie błędna). Odpowiedź *"nie wiem"* oznacza, że należy powtórzyć badania lub skorzystać z innych metod diagnostycznych.

Zadaniem w automatycznej nadzorowanej klasyfikacji obiektów będzie zatem znalezienie funkcji ϕ, takiej, że

$$\phi : O \to Z, \quad \forall o \in O \quad \phi(o) = i, \tag{1.3}$$

która będzie jak najlepszym przybliżeniem nieznanej funkcji Φ. To znaczy chcielibyśmy, aby $\phi(o) = i \iff \Phi(o) = i$ dla każdego rozpoznawanego obiektu o. Odwzorowanie ϕ możemy zdefiniować [146] jako złożenie trzech odwzorowań składowych:

$$\phi = \phi_3 \circ \phi_2 \circ \phi_1, \tag{1.4}$$

z których

$$\phi_1 : O \to X \tag{1.5}$$

przyporządkowuje każdemu obiektowi o ze zbioru O jego reprezentację x z przestrzeni reprezentacji X. Drugie odwzorowanie:

$$\phi_2 : X \to R^c \tag{1.6}$$

przyporządkowuje reprezentacji $x \in X$ obiektu $o \in O$ wektor c liczb rzeczywistych:

$$(d_1(x), d_2(x), \ldots, d_c(x)) \tag{1.7}$$

poszczególne składowe są to wartości funkcji dyskryminacyjnych (definicja funkcji dyskryminacyjnej zostanie podana później), a ostatnie odwzorowanie:

$$\phi_3 : R^c \to Z \tag{1.8}$$

przypisuje wektorowi obliczonych wartości funkcji dyskryminacyjnych odpowiednią decyzję z ze zbioru decyzji Z.

Możemy teraz zdefiniować klasyfikator jako funkcję ψ, przyporządkowującą danej reprezentacji obiektu zwanej też próbką (ang. sample) pewną decyzję z ze zbioru decyzji Z. Funkcja ta będzie złożeniem funkcji ϕ_2 i ϕ_3:

$$\psi : X \to Z, \quad \psi = \phi_3 \circ \phi_2 \tag{1.9}$$

Funkcję ψ będziemy nazywać także regułą decyzyjną [87].

Z podanej definicji wynika, że dla działania klasyfikatora potrzebujemy znaleźć c funkcji dyskryminacyjnych d_i, które będą odwzorowywały przestrzeń reprezentacji X w zbiór liczb

rzeczywistych R.

$$d_i : X \to R, \quad i = 1, 2, \ldots, c \tag{1.10}$$

Funkcje te powinny być dobrane tak, aby dla każdej reprezentacji obiektu o z i-tej klasy funkcja ta przyjmowała największą wartość spośród c wartości funkcji tj.

$$\forall_{i=1,\ldots,c} \, \forall_{j=1,\ldots,c, j\neq i} \, \forall_{o\in O_i} \; d_i(x) > d_j(x), \; x = \phi_1(o) \tag{1.11}$$

W praktycznych zastosowaniach warunek ten rzadko kiedy jest spełniony. Wektory w przestrzeni cech mogą nie być liniowo separowalne. Jednak wartość funkcji $d_i(x)$ możemy traktować jako miarę podobieństwa obiektu o reprezentowanego przez x do i-tej klasy.

1.2. Podstawowe typy klasyfikatorów

Metody automatycznej klasyfikacji obiektów są rozwijane już od kilkudziesięciu lat. W tym czasie opracowano niezliczone algorytmy klasyfikacji różniące się podejściem do problemu, szybkością klasyfikacji, złożonością obliczeniową, czy też podatnością na błędy. Nie sposób w niniejszej rozprawie przedstawić wszystkich znanych z literatury klasyfikatorów.

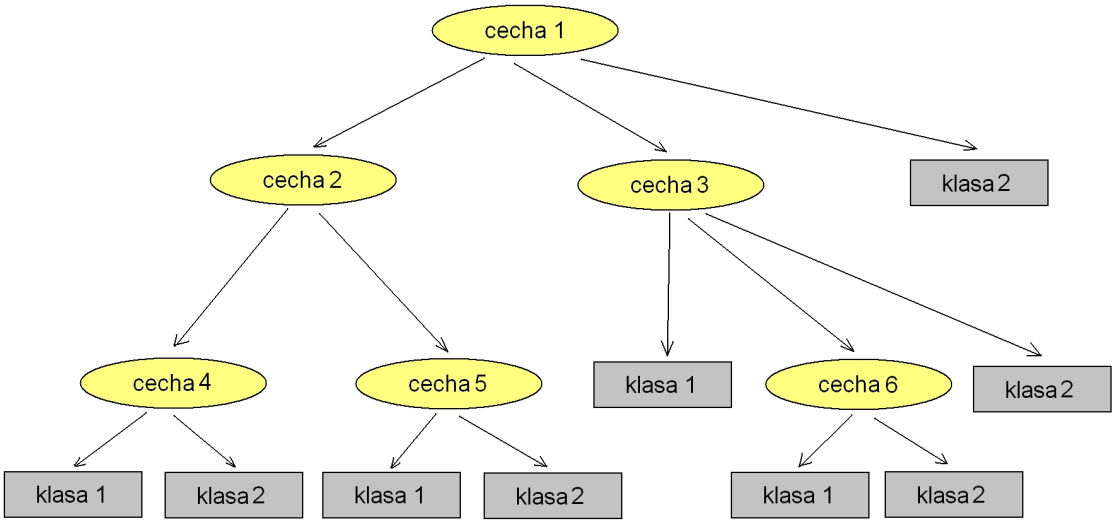
Dlatego też zaprezentowany zostanie tylko bardzo skrótowy opis kilku podstawowych podejść do problemu klasyfikacji. Zostaną wspomniane drzewa decyzyjne, sieci neuronowe, metody k-najbliższych sąsiadów oraz klasyfikatory bayesowskie. W osobnych podrozdziałach bardziej szczegółowo zostaną opisane klasyfikatory, które zostały wykorzystane do testowania zaproponowanych rozwiązań: klasyfikator RDA oraz klasyfikator SVM.

Drzewa decyzyjne

Drzewa decyzyjne (ang. decision trees) to jedno z bardziej intuicyjnych podejść do problemu klasyfikacji. Metody te opierają się na jednym z podstawowych paradygmatów pozwalających skutecznie rozwiązywać złożone zadania obliczeniowe: *"dziel i zwyciężaj"* (ang. divide and conquer). Polega on na podziale złożonego problemu na prostsze, a następnie rekursywnym zastosowaniu tej samej strategii do stworzonych podproblemów.

Drzewo decyzyjne jest skierowanym acyklicznym grafem, w którym każdy wierzchołek jest albo węzłem posiadającym dwóch lub więcej potomków albo liściem. Z każdym liściem skojarzona jest etykieta klasy. Próbka wchodząca na wejście takiego klasyfikatora niejako przechodzi od korzenia w dół drzewa aż do liścia, gdzie otrzymuje etykietę klasy. Śledząc krok po kroku ścieżkę, którą przechodziła dana próbka możemy analizować problem klasyfikacji stopniowo od bardzo ogólnych decyzji do bardzo szczegółowych.

Jeśli mamy do czynienia z problemem wieloklasowym i eksperci są w stanie wskazać pewne grupy klas podobnych do siebie oraz rozgraniczać klasy na różnych poziomach, to informacje takie mogą być również odpowiednio wykorzystane przy budowaniu stosownego drzewa. Przykładowe drzewo decyzyjne jest przedstawione na rysunku 1.1



Rysunek 1.1. Przykładowe drzewo decyzyjne

Jest bardzo wiele algorytmów reprezentujących to podejście. Jedne z najbardziej znanych to: iteracyjny dychotomizer ID3 (ang. Iterative Dychotomiser 3) opracowany przez Quinlana [121], [122], CART (ang. Classification and Regression Trees) [15] oraz algorytm C4.5 [123].

Głównymi zaletami drzew decyzyjnych jest szybka klasyfikacja, zrozumiały (dla człowieka) proces decyzyjny, możliwość stosowania różnego rodzaju cech (a nawet różnych reprezentacji cech) w poszczególnych krokach algorytmu.

Bardzo ważną zaletą drzew decyzyjnych jest także to, że ich tworzenie przebiega rekurencyjnie. Na każdym kolejnym etapie zawężamy analizowany obszar przestrzeni obiektów. Dzięki temu istnieje możliwość wykorzystania cech, które są istotne lokalnie, a które niewiele wnoszą do klasyfikacji całej przestrzeni.

Kolejną zaletą jest możliwość analizowania procesu klasyfikacji przez ekspertów dziedzinowych. Mogą oni na podstawie ścieżek, którymi przechodziły próbki modyfikować poszczególne reguły decyzyjne poprawiając w ten sposób jakość algorytmu.

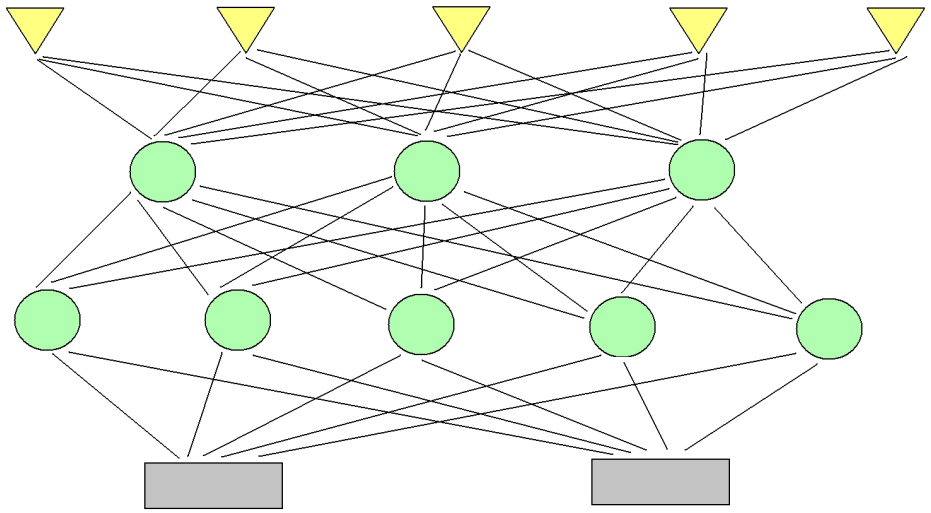
Do największych wad drzew decyzyjnych należy zaliczyć testowanie jednego atrybutu na raz oraz bardzo skomplikowany proces adaptacji drzewa, gdy zbiór uczący ulega powiększeniu. Pierwsza z tych wad jest szczególnie istotna w przypadku problemów o dużej wymiarowości wektora cech.

Sieci neuronowe

Sztuczna sieć neuronowa ANN (ang. Artificial Neural Network) jest modelem matematycznym, który składa się z węzłów obliczeniowych zwanych neuronami oraz połączeń między nimi. Jej działanie stara się symulować ludzki mózg. Każdy neuron jest jednostką, która przetwarza sumę sygnałów wejściowych. Poprzez tak zwaną funkcję aktywacji uwzględnia on wagi, związane z połączeniami międzyneuronowymi. Wartość tej funkcji określa stopień wzbudzenia neuronu.

Istnieje bardzo wiele różnorodnych architektur sieci neuronowych. Różnią się one liczbą neuronów, schematami połączeń pomiędzy nimi, funkcjami aktywacji, czy też metodami uczenia się sieci. Ciekawy przegląd sieci ANN można znaleźć w [45].

Jednym z popularniejszych typów sieci neuronowych jest sieć nazywana wielowarstwowym perceptronem MLP (ang. Multi-Layer Perceptron). Przykład takiej sieci został przedstawiony na rysunku 1.2. Jest to sieć, która składa się z pewnej liczby warstw neuronów. Pierwsza z tych warstw (zaznaczona kolorem żółtym) nazywana jest warstwą wejściową, a ostatnia (zaznaczona kolorem szarym) wyjściową. Pomiędzy nimi umieszcza się warstwy ukryte. Neurony tej samej warstwy nie są ze sobą połączone, ale neurony dwóch sąsiednich warstw są połączone każdy z każdym. Sygnał przechodzi od warstwy wejściowej poprzez warstwy ukryte do warstwy wyjściowej, gdzie jest interpretowany. Podając liczbę warstw sieci zwykle nie liczy się warstwy wejściowej.



Rysunek 1.2. Struktura trójwarstwowej sieci MLP

Z każdym połączeniem pomiędzy neuronami związana jest pewna waga, która wpływa na modyfikację (najczęściej nieliniową) przekazywanych przez neuron wartości sygnału. Proces uczenia się polega na modyfikacji tych wag celem możliwie najlepszej dyskryminacji próbek

przychodzących na wejście. Sam neuron odbiera sygnały od neuronów poprzedniej warstwy, mnoży te sygnały przez wagi związane z połączeniami i sumuje uzyskane wartości. Następnie suma ta przeliczana jest na jakiś skończony przedział i ta wartość zostaje przekazana do następnej warstwy.

Funkcja dyskryminacyjna dwuwarstwowego perceptronu, zawierającego N neuronów w warstwie ukrytej przyjmuje formę:

$$d_i(x) = f_2(\sum_{j=0}^{N} w_{ij}^2 f_1(\sum_{r=0}^{d} w_{jr}^1 x_r^0)), \quad i = 1, \ldots, c \tag{1.12}$$

gdzie x_r^0 są wartościami wejściowymi, w_{ij}^2, w_{jr}^1 wagami odpowiednich warstw sieci d jest wymiarowością wektora wejściowego, c jest ilością klas, a f_1 i f_2 są funkcjami aktywacji. Najczęściej jako funkcji aktywacji używa się sigmoidalnych funkcji unipolarnych danych wzorem:

$$f(x) = \frac{1}{1 + e^{-\beta x}} \tag{1.13}$$

gdzie $\beta \in (0, 1]$ jest pewnym stałym współczynnikiem.

Wagi w węzłach sieci są modyfikowane podczas procesu uczenia się, którego zadaniem jest minimalizowanie odległości pomiędzy pożądaną, a rzeczywistą odpowiedzią sieci. Częstą metodą stosowaną do uczenia sieci neuronowych jest algorytm wstecznej propagacji błędu (ang. backpropagation of errors) [8], w którym każdy neuron lokalnie zmniejsza swój błąd stosując metodę spadku gradientu.

Jedną z większych zalet sieci neuronowych jest ich dość duża odporność na zaszumione dane. Sieć neuronowa potrafi wyciągać poprawne wnioski (przydzielać obiekty do właściwych klas), nawet jeśli dane wejściowe są obarczone dużymi błędami. Sieci neuronowe charakteryzują się dużą szybkością klasyfikacji. Dobrze radzą sobie w sytuacjach, w których trudno jest określić ścisłe reguły klasyfikacji, a dane wejściowe wydają się (dla człowieka) chaotyczne. Przykładem takiego problemu mogą być różnego rodzaju analizy finansowe jak chociażby analiza finansowych szeregów czasowych czy też ocena zdolności kredytowej.

Ciekawą cechą sieci neuronowych, która może być bardzo istotna w przypadku jej sprzętowej implementacji, jest jej duża odporność na uszkodzenia. Nawet w przypadku przerwania niektórych połączeń pomiędzy neuronami sieć działa nadal – podobnie jak mózg człowieka.

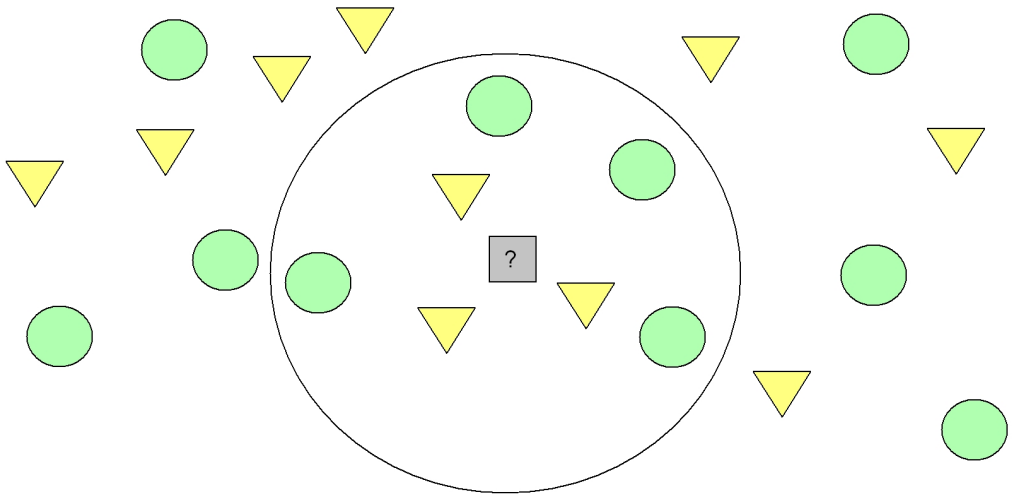
Sieci neuronowe mają również pewne wady. Możemy do nich zaliczyć długi proces uczenia się (dobierania wag). Co więcej pozyskana podczas tego procesu wiedza jest bardzo trudna

(lub nawet niemożliwa) do interpretacji przez człowieka. Dużą trudność stanowi również uwzględnianie tak zwanej wiedzy dziedzinowej (wiedzy ekspertów o danym problemie). Sieci neuronowe są także podatne na proces przeuczenia (ang. overfitting).

Klasyfikatory minimalno-odległościowe

Najbardziej znanym klasyfikatorem tego typu jest metoda k-najbliższych sąsiadów k-NN (ang. k-Nearest Neighbour). Metoda ta polega na przydzieleniu klasyfikowanej próbki do klasy najczęściej występującej wśród jej k najbliższych sąsiadów, w sensie pewnej ustalonej miary odległości. Najczęściej stosowanymi miarami odległości są metryka Euklidesowa (ang. Euclidean distance) lub metryka miejska (ang. Manhattan distance). Rzadziej stosowane, ze względu na wyższy koszt obliczeniowy, są metryki Czebyszewa czy Mahalonobis [**?**].

Zasada działania tego klasyfikatora jest bardzo prosta. Została ona przedstawiona graficznie na rysunku 1.3. Testowana próbka "?" zostanie zakwalifikowana jako koło, ponieważ w jej najbliższym sąsiedztwie jest więcej tego typu obiektów. Warto przy tym zauważyć, że odległość testowanej próbki od obiektów danej klasy nie ma żadnego znaczenia.



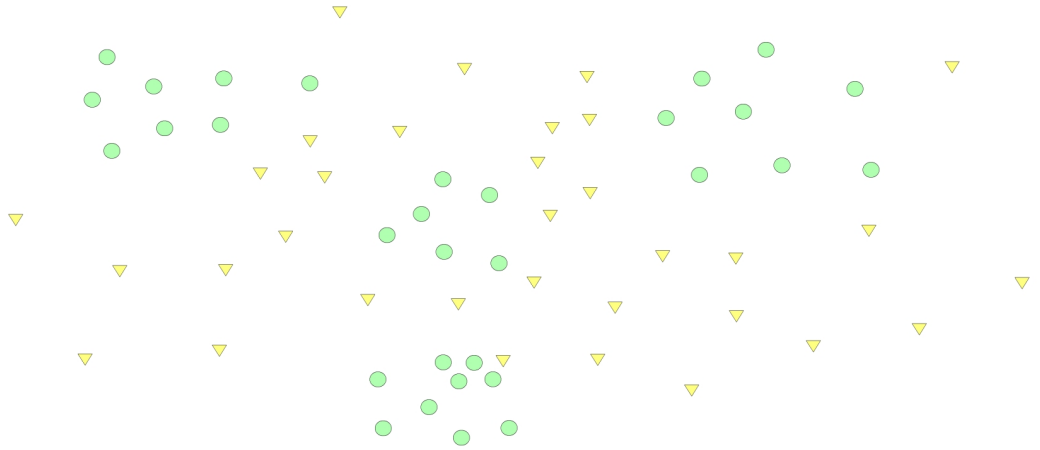
Rysunek 1.3. Zasada działania klasyfikatora 7-NN

Oczywiście, w przypadku parzystej wartości k może się zdarzyć, że liczba sąsiadów z dwóch różnych klas będzie identyczna. Dlatego też najczęściej wybiera się nieparzystą wartość tego parametru. Rozwiązuje to problem tej samej liczby sąsiadów wyłącznie w przypadku problemu binarnego. Jeśli mamy do czynienia z większą ilością klas, to musimy inaczej rozwiązać ten problem. Przy równej liczbie sąsiadów możemy na przykład zakwalifikować obiekt do klasy, do której należy jego najbliższy sąsiad.

Proces uczenia klasyfikatora k-NN polega po prostu na doborze parametru k. W literaturze zaproponowano wiele różnych metod doboru tego parametru. Jednak najprostsze i najczęściej stosowane jest użycie metody kroswalidacji.

Istnieje bardzo wiele modyfikacji metody k-najbliższych sąsiadów. Trudno byłoby przedstawić je tutaj choćby skrótowo. Jedną z ciekawszych, która jednocześnie rozwiązuje problem tej samej liczby sąsiadów jest metoda "k dyplomatycznych najbliższych sąsiadów" k-DNN (ang. k-Diplomatic Nearest Neighbours). Polega ona na tym, że szukamy k najbliższych sąsiadów testowanej próbki z każdej klasy. Dla każdej z klas, obliczamy średnią odległość tak wybranych obiektów od próbki, którą następnie zaliczamy do klasy, dla której ta odległość jest najmniejsza.

Inną metodą, a raczej grupą metod, są rozmyte klasyfikatory k-NN (ang. Fuzzy k-NN classifiers) [79]. Również w tym przypadku odległość testowanej próbki od obiektów danej klasy ma istotne znaczenie.



Rysunek 1.4. Przykład przestrzeni, w której obiekty skupiają się w rozłącznych obszarach

Mimo swojej prostoty, metoda k-najbliższych sąsiadów osiąga bardzo dobre rezultaty w wielu zastosowaniach. Na przykład przy rozpoznawaniu pisanych cyfr, analizie zdjęć satelitarnych, czy też analizie sygnałów EKG [66]. Metoda ta daje szczególnie dobre wyniki w sytuacji, gdy mamy do czynienia z bardzo nieregularnymi granicami decyzyjnymi lub gdy próbki reprezentujące tę samą klasę skupiają się w rozłącznych obszarach przestrzeni cech. Tak jak na przykład na rysunku 1.4.

Do zalet metody k-NN można zaliczyć wyjątkowo prosty proces uczenia się (polega on po prostu na wyborze parametru k), prostotę implementacji oraz asymptotyczną optymalność. Oznacza ona, że gdy liczność zbioru uczącego $n \to \infty$ i $k/n \to 0$, to metoda ta jest zbieżna do błędu Bayesa [53]. Nie bez znaczenia jest także prostota implementacji tej metody jak i jej podatność na modyfikacje.

Metoda ta jednak ma też pewne wady. Jedną z nich jest wolny proces klasyfikacji. W przypadku każdej klasyfikowanej próbki przeglądamy cały zbiór uczący, aby obliczyć wszystkie odległości. Powoduje to, że klasyfikator działa tym wolniej im większy jest zbiór uczący. Jak łatwo zauważyć sam proces uczenia, choć bardzo prosty jest bardzo kosztowny obliczeniowo. Na przykład, aby wybrać optymalną wartość k musimy obliczyć $O(n^2)$ odległości.

Kolejną wadą jest również dość duża, w porównaniu z innymi podejściami, wrażliwość na zbędne cechy. Powoduje to konieczność stosowania algorytmów selekcji cech, które dodatkowo mogą wpływać na koszt uczenia się klasyfikatora. Nie bez znaczenia jest też konieczność pamiętania całego zbioru uczącego podczas procesu klasyfikacji.

Uproszczeniem tej metody jest klasyfikator 1-NN, który przydziela testowaną próbkę do klasy, do której należy jej najbliższy sąsiad. Metoda ta, choć wyjątkowo prosta daje w wielu zastosowaniach dobre rezultaty. W przypadku tej metody proces klasyfikacji nadal wymaga przejrzenia wszystkich próbek ze zbioru uczącego pomijamy jednak czas potrzebny na uczenie się klasyfikatora.

Naiwny klasyfikator Bayesa

Załóżmy, że znany jest model probabilistyczny zadania klasyfikacji oraz prawdopodobieństwa przynależności próbek do klas $P(c_1), P(c_2), \ldots, P(c_c)$. Załóżmy również, że znamy gęstości rozkładów prawdopodobieństwa wystąpienia próbek x w poszczególnych klasach: $f(x|c_1), f(x|c_2), \ldots, f(x|c_c)$ zwane prawdopodobieństwami *a priori*. Korzystając z reguły Bayesa możemy wyliczyć prawdopodobieństwo wystąpienia klasy c_i względem próbki x, czyli inaczej prawdopodobieństwo, że próbka x będzie należała do klasy c_i:

$$P(c_i|x) = \frac{f(x|c_i)P(c_i)}{f(x)}, \tag{1.14}$$

gdzie $f(x|c_i)$ dla $i = 1, 2, \ldots, c$ są gęstościami rozkładów, c jest ilością klas, a $f(x)$ jest sumą gęstości po wszystkich klasach:

$$f(x) = \sum_{i=1}^{c} f(x|c_i)P(c_i) \tag{1.15}$$

Prawdopodobieństwa $P(c_i|x)$ nazywane są prawdopodobieństwami *a posteriori*. Łatwo zauważyć, że średnia wartość straty jaką ponosimy w przypadku przydzielenia próbki x do klasy i-tej wynosi:

$$r_i(x) = \sum_{j=1}^{c} L_{ij} P(c_j|x), \tag{1.16}$$

gdzie $P(c_j|x)$ są prawdopodobieństwami *a posteriori*, a L_{ij} jest tak zwaną funkcją straty, która odzwierciedla ujemne konsekwencje błędnej decyzji, czyli zaliczenia próbki do klasy i podczas gdy w rzeczywistości należy ona do klasy j. Funkcja ta może przyjmować wartości: $0 \leq L_{ij} < \infty,\ i, j = 1, 2, \ldots, c.$

Zdefiniujmy prostą zero-jedynkową funkcję straty, w postaci:

$$L_{ij} = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \qquad i, j = 1, 2, \ldots, c \tag{1.17}$$

Zatem możemy przyporządkować każdemu wektorowi klasę, której odpowiada maksymalna wartość prawdopodobieństwa *a posteriori*.

$$d(x) = argmax_c P(c_i|x) = argmax_c \frac{f(x|c_i)P(c_i)}{f(x)}, \tag{1.18}$$

ponieważ mianownik we wzorze 1.18 nie zależy od klasy, to możemy napisać:

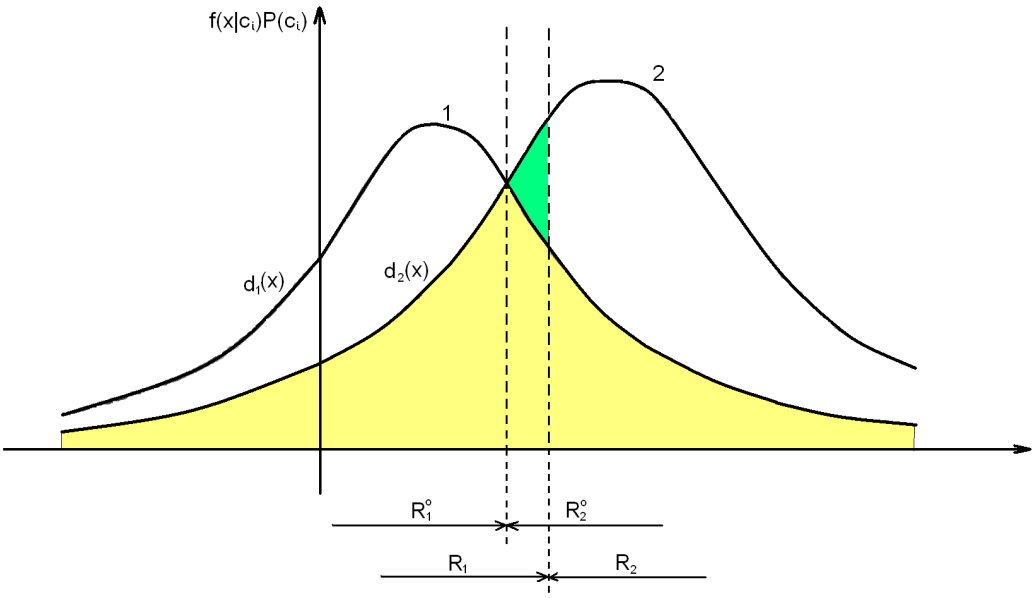
$$d(x) = argmax_c f(x|c_i)P(c_i), \tag{1.19}$$

Model działający według tego wzoru nosi nazwę optymalnego klasyfikatora bayesowskiego BOC (ang. Bayesian Optimal Classifier), zaś powyższa reguła jest nazywana często regułą największego prawdopodobieństwa *a posteriori* MAP (ang. Maximum A posteriori Probability).

Zdefiniowany w ten sposób klasyfikator bayesowski jest klasyfikatorem optymalnym [47] pod względem błędu uogólnienia. Prawdopodobieństwo tego błędu wynosi [87]:

$$P(blad) = \int_{R_1} f(x|c_2)P(c_2)dx + \int_{R_2} f(x|c_1)P(c_1)dx, \tag{1.20}$$

gdzie R_1 i R_2 są obszarami decyzyjnymi (tak jak to zaznaczono na rysunku 1.5). Jest ono równe polu pod krzywymi $d_1(x),\ \ d_2(x)$ (na rysunku zaznaczone kolorem żółtym). Pole to może zostać zmniejszone o tak zwany obszar redukowalny (zaznaczony kolorem zielonym). Można udowodnić, że w przypadku $f(x|c_1)P(c_1) = f(x|c_2)P(c_2)$ pole wyrażające



Rysunek 1.5. Obszary decyzyjne i prawdopodobieństwo błędnej klasyfikacji

prawdopodobieństwo błędnej klasyfikacji jest najmniejsze. Stąd wynika, że reguła Bayesa minimalizuje prawdopodobieństwo błędnej klasyfikacji. Można pokazać [37], że wartość prawdopodobieństwa błędnej klasyfikacji dla reguły Bayesa mieści się w zakresie:

$$0 \leq P(blad) \leq \frac{c-1}{c}. \tag{1.21}$$

Klasyfikator BOC jest optymalny jednak tylko pod warunkiem, że wykorzystane do obliczeń prawdopodobieństwa i gęstości rozkładów prawdopodobieństwa są znane. Zwykle musimy polegać na ich estymatorach, czyli możemy je szacować na podstawie próbek ze zbioru uczącego.

Jednak osiągnięcie dokładnego oszacowania wymagałoby olbrzymiego zbioru danych. Taki klasyfikator byłby bardzo niepraktyczny i całkowicie nieefektywny. Dlatego też, chcąc uprościć zadanie zakładamy, że w ramach jednej klasy, wartości poszczególnych cech są niezależne od siebie. Prowadzi to do oszacowania:

$$f(x|c_i) = f(x_1, x_2, \ldots, x_c|c) \approx \prod f(x_i|c) \tag{1.22}$$

W efekcie otrzymamy regułę klasyfikacji:

$$d(x) = argmax_{c \in C} P(c) \prod_{i=1}^{c} f(x_i|c) \tag{1.23}$$

W większości zastosowań założenie o wzajemnej niezależności cech nie jest spełnione. Jednak mimo to naiwny klasyfikator bayesowski okazuje się bardzo skutecznym narzędziem. Algorytm ten potrafi osiągać lepsze wyniki niż zaawansowane algorytmy oparte o drzewa decyzyjne, czy też sieci neuronowe. Przy czym wymagany nakład obliczeń jest dużo niższy.

1.2.1. Klasyfikator RDA

Aby zdefiniować klasyfikator RDA musimy rozpocząć od zdefiniowania pojęcia kwadratowej analizy dyskryminacyjnej QDA (ang. Quadratic Discriminant Analysis) [57]. Klasyfikator QDA modeluje prawdopodobieństwa klas jako wielowymiarowy rozkład gaussowski o różnych wektorach oczekiwanych i o różnych macierzach kowariancji. Następnie używa prawdopodobieństwa *a posteriori*, aby szacować klasę, do której należy próbka.

Tak jak to zostało wspomniane w poprzednim podrozdziale zwykle nie znamy funkcji gęstości prawdopodobieństwa. Możemy jednak założyć, że jest ona dana wielowymiarowym rozkładem gaussowskim, czyli:

$$f(x|c_k) = \frac{1}{(2\Pi)^{p/2}|\Sigma_k|^{1/2}} e^{-1/2(x-\mu_k)^T \Sigma_k^{-1}(x-\mu_k)} \tag{1.24}$$

korzystając z tego, że logarytm jest funkcją monotoniczną, możemy obłożyć obie strony funkcją ln, otrzymując:

$$ln(f(x|c_k)P(c_k)) = -\frac{1}{2}(x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) - \frac{1}{2}ln|\Sigma_k| + \ln P(c_k), \tag{1.25}$$

Mnożąc obie strony równania przez -2 prowadzi to do funkcji dyskryminacyjnej postaci:

$$d_k(x) = (x-\mu_k)^T\Sigma_k^{-1}(x-\mu_k) + ln|\Sigma_k| - 2lnP(c_k), \tag{1.26}$$

gdzie x oznacza wektor cech, μ_k wektor średni, Σ_k macierz kowariancji, a $P(c_k)$ prawdopodobieństwo *a priori* pojawienia się klasy c_k.

Reguła klasyfikacyjna przyjmuje wtedy postać:

$$d_k(x) = argmin_{1\leq k\leq c} d_k(x) \iff argmax_{1\leq k\leq c} P(c_k|x). \tag{1.27}$$

Oczywiście parametry rozkładu gaussowskiego są estymowane na podstawie zbioru uczącego, zatem wartości Σ_k oraz μ_k zastępujemy we wzorze 1.26 poprzez ich wartości estymowane $\hat{\Sigma}_k$ oraz $\hat{\mu}_k$ czyli:

$$\hat{\mu}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} x_i \tag{1.28}$$

oraz

$$\hat{\Sigma}_k = \frac{1}{n_k - 1} \sum_{i=1}^{n_k} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)^T, \quad k = 1, 2, \ldots, c \tag{1.29}$$

Jednakże gdy liczba próbek w zbiorze uczącym jest zbyt mała w porównaniu do liczby wymiarów wektora cech estymacja macierzy kowariancji jest źle uwarunkowana (ang. ill-posed). Problem ten jest znany jako problem małej ilości próbek SSS (ang. Small Sample Size problem) i jest szeroko opisywany w literaturze. Powoduje on, że estymowane macierze kowariancji stają się osobliwe, a zatem nie jesteśmy w stanie skorzystać ze wzoru 1.26. Możemy znacząco zwiększyć liczbę próbek używanych do estymacji macierzy kowariancji $\hat{\Sigma}_k$ poprzez zastąpienie jej przez średnią $\hat{\Sigma}$ (ang. pooled covariance matrix), w postaci:

$$\hat{\Sigma} = \sum \hat{\Sigma}_k / \sum N_k, \tag{1.30}$$

co prowadzi do linearnej analizy dyskryminacyjnej LDA (ang. Linear Discriminant Analysis). Funkcja dyskryminacyjna przyjmuje wtedy postać:

$$d_i(x) = x^T \hat{\Sigma}^{-1} \hat{\mu}_i - \frac{1}{2} \hat{\mu}_i^T \hat{\Sigma}^{-1} \hat{\mu}_i + ln(\hat{P}(c_i)) \tag{1.31}$$

Jak łatwo zauważyć do estymacji macierzy kowariancji $\hat{\Sigma}$ używamy wszystkich próbek zbioru uczącego reprezentujących wszystkie klasy. Dzięki temu w wielu wypadkach udaje się uniknąć problemu SSS. Takie podejście zakłada, że wszystkie macierze kowariancji są do siebie podobne. Jednak w przypadku większości problemów założenie takie nie jest prawdziwe. To powoduje znaczące pogorszenie wyników uzyskiwanych przez ten klasyfikator.

Inne rozwiązanie zwane regularyzacją macierzy kowariancji zaproponował Friedman [56]. Niech nasza estymowana macierz będzie obliczana za pomocą następującej formuły:

$$\hat{\Sigma}_k(\lambda) = (1 - \lambda)\hat{\Sigma}_k + \lambda\hat{\Sigma}, \tag{1.32}$$

gdzie $0 \leq \lambda \leq 1$, $\hat{\Sigma}_k$ estymowana macierz kowariancji, a $\hat{\Sigma}$ estymowana średnia macierz kowariancji obliczana za pomocą wzoru 1.30.

Jak łatwo zauważyć parametr λ będzie kontrolował stopień podobieństwa indywidualnych macierzy kowariancji $\hat{\Sigma}_k$ do macierzy uśrednionej $\hat{\Sigma}$. W przypadku gdy $\lambda = 1$ wszystkie macierze kowariancji będą takie same, czyli będziemy mieli do czynienia z klasyfikatorem LDA, a w przypadku gdy $\lambda = 0$ regularyzacja nie będzie wykonywana, czyli otrzymamy klasyfikator QDA. Klasyfikator wykorzystujący regularyzację macierzy kowariancji będziemy nazywali klasyfikatorem RDA.

Nie istnieje jedna uniwersalna wartość parametru λ, która będzie sprawdzała się dla wszystkich zadań klasyfikacji. Optymalną dla danego problemu wartość musimy wyznaczyć eksperymentalnie. Zwykle stosuje się do tego celu metodę k-kroswalidacji.

Współcześnie stosuje się również inne metody regularyzacji macierzy kowariancji oparte na dodawaniu do niej w odpowiedniej proporcji macierzy jednostkowych. Na przykład:

$$\hat{\Sigma}_k(\lambda.\gamma) = (1-\gamma)\hat{\Sigma}_k(\lambda) + \gamma\frac{tr[\hat{\Sigma}_k(\lambda)]}{p}I, \tag{1.33}$$

gdzie $0 \leq \gamma \leq 1$, $\hat{\Sigma}_k(\lambda)$ estymowana średnia macierz kowariancji obliczana za pomocą wzoru 1.32, $tr[\hat{\Sigma}_k(\lambda)]$ - ślad macierzy $\hat{\Sigma}_k(\lambda)$, p - liczba wartości własnych, a I macierz jednostkowa.

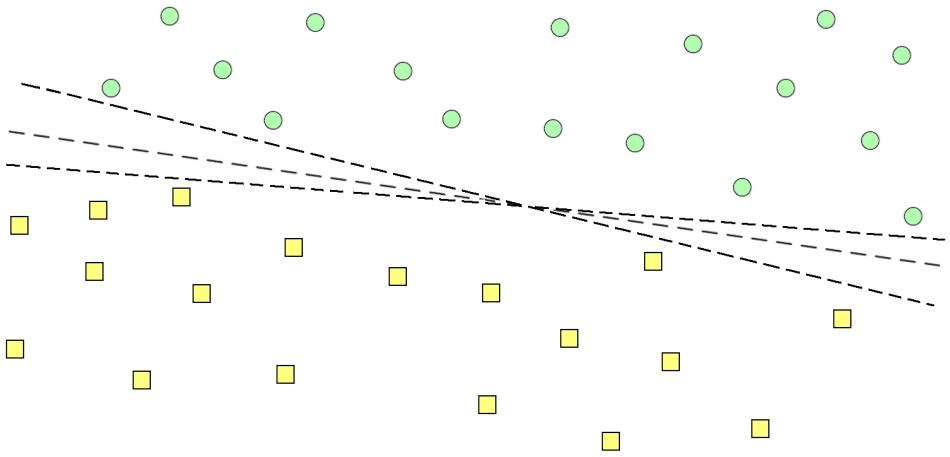
We współczesnych problemach klasyfikacji wymiarowość wektorów cech rośnie (patrz tabela 1.3 w podrozdziale Selekcja cech). Przy czym liczba próbek reprezentujących każdą z klas jest często niezbyt duża. Może to spowodować, że regularyzacja macierzy kowariancji może się okazać niewystarczająca [76].

W takim wypadku nie pozostaje nic innego jak próba zredukowania wymiarowości przestrzeni cech za pomocą algorytmów selekcji. Takie rozwiązanie, klasyfikator bezpośrednio regularyzowanej analizy dyskryminacyjnej RD-QDA (ang. Regularized Direct QDA), zostało zaproponowane w pracy [101]. Jeszcze inne podejście do tego problemu łączące regularyzację macierzy kowariancji z selekcją cech (metoda LASSO) zostało zaprezentowane w pracy Tibshiraniego [151]. Więcej na temat problemu SSS i metodach radzenia sobie z nim można znaleźć w pracy [160].

1.2.2. Klasyfikator SVM

Klasyfikator ten, nazywany maszyną wektorów wspierających SVM (ang. Support Vector Machine), został zaproponowany przez Vapnika [157]. Jest on z sukcesem stosowany w wielu dziedzinach. Dzięki swoim dobrym własnościom generalizacji pokonuje on wiele tradycyjnych podejść. Klasyfikator ten wykazuje lepszą, od innych klasyfikatorów, skuteczność klasyfikacji danych, zwłaszcza w przypadku małej ilości próbek w zbiorze uczącym [1].

W wielu zadaniach klasyfikacji, z którymi spotykamy się we współczesnej nauce i technice pojawia się problem zbyt małej ilości próbek. Zwłaszcza jest on bardzo dotkliwy w problemach wieloklasowych o dużej ilości klas. Dlatego też właśnie ten klasyfikator został wybrany do testowania zaproponowanych metod.



Rysunek 1.6. Możliwe podziały przestrzeni cech

Załóżmy zatem, że mamy dany zbiór próbek wyrażony jako zbiór par: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, gdzie x_i jest wektorem cech opisującym próbkę, a $y_i \in \{-1, 1\}$ są etykietami klas. Zadanie klasyfikacji polega na przydzieleniu próbki do jednej z tych dwóch klas. W przestrzeni cech to zadanie odpowiada zadaniu znalezienia hiperpłaszczyzny, która oddziela te dwie klasy od siebie. Jak łatwo zauważyć na rysunku 1.6 istnieje nieskończenie wiele takich hiperpłaszczyzn. Nasz problem polega na znalezieniu takiej, która będzie optymalna ze względu na zadanie klasyfikacji. Załóżmy zatem, że hiperpłaszczyzna taka będzie opisywana wzorem:

$$f(x) = x^T w + b \tag{1.34}$$

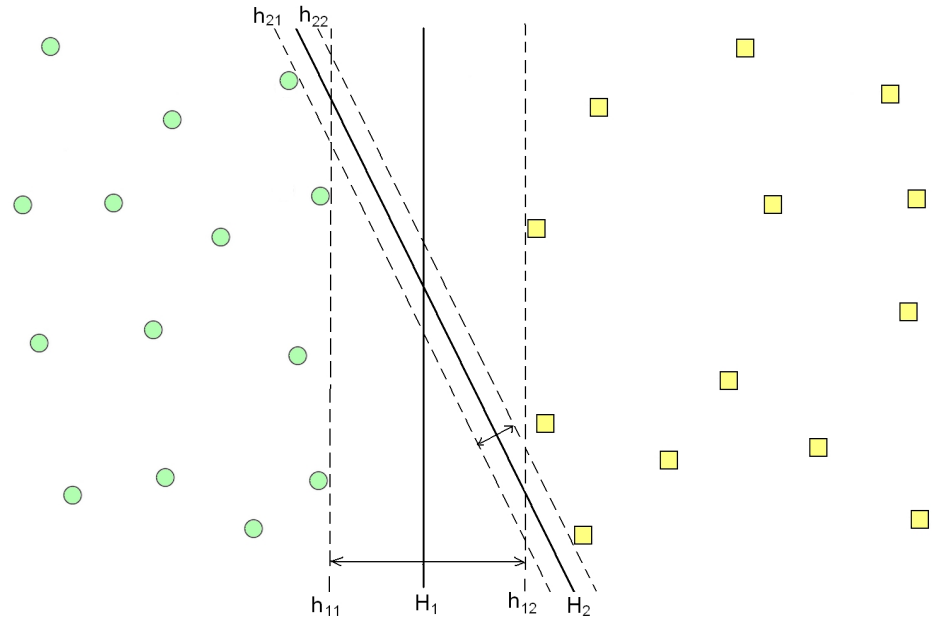
gdzie x oznacza wektor cech, a w jest wektorem wag. Problem jest liniowo separowalny jeśli istnieje taka stała $\varepsilon > 0$, że dla każdej próbki x_i zachodzi:

$$x_i^T w + b \geq \varepsilon, \text{ dla } y_i = 1 \quad \text{oraz} \quad x_i^T w + b \leq -\varepsilon, \text{ dla } y_i = -1 \tag{1.35}$$

Odpowiednie przeskalowanie wektora w oraz wartości b pozwala nadać tej stałej ε wartość 1. Możemy zatem zapisać nierówności 1.35 za pomocą jednej nierówności w następujący

sposób:

$$y_i(x_i^T w + b) \geq 1 \tag{1.36}$$



Rysunek 1.7. Hiperpłaszczyzna H_1 i H_2 dzielące przestrzeń z różnymi marginesami

Na rysunku 1.7 pokazano dwie takie hiperpłaszczyzny: H_1 i H_2. Oczywiście możemy znaleźć nieskończenie wiele innych możliwości. Jednak intuicyjnie, najlepszą z nich będzie taka, która zapewni nam jak najszerszy margines pomiędzy skrajnymi wektorami obu klas. Zauważmy, że odległość pomiędzy hiperpłaszczyznami h_{11} i h_{12} wynosi:

$$\frac{2}{\|w\|} \tag{1.37}$$

Zatem, aby zmaksymalizować margines, musimy znaleźć jej maksimum. Zadanie to możemy sprowadzić do zadania minimalizacji następującej wartości:

$$\frac{1}{2}\|w\| \quad lub \quad \frac{1}{2}\|w\|^2 \tag{1.38}$$

przy spełnionych warunkach:

$$y_i(x_i^T w + b) \geq 1, \quad i = 1, 2, \ldots, n \tag{1.39}$$

Wartość $\|w\|$ możemy zastąpić poprzez $\|w\|^2$ ponieważ obie wielkości są tak samo monotoniczne i osiągają minimum w tym samym miejscu.

Aby rozwiązać ten problem optymalizacyjny znajdujemy odpowiadający mu problem dualny,:

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \tag{1.40}$$

przy ograniczeniach

$$\alpha_i \geq 0 \text{ dla } i = 1, 2, ..., N \text{ oraz } \sum_{i=1}^{N} \alpha_i y_i = 0 \tag{1.41}$$

Rozwiązanie tego problemu prowadzi do funkcji definiującej hiperpłaszczyznę w postaci:

$$f(x) = sign\Big(\sum_{i=1}^{N} \alpha_i y_i x_i x + b \Big) , \tag{1.42}$$

gdzie x_i są wektorami cech nazywanymi wektorami wspierającymi (ang. support vectors), a α_i są niezerowymi współczynnikami Lagranga.

Wektory wspierające to wektory cech, które leżą na marginesach hiperpłaszczyzny. Jest to jedna z największych zalet tego podejścia, ponieważ do wyliczenia funkcji dyskryminacyjnej konieczne są tylko te wektory. Stanowią one zwykle tylko niewielką część ze wszystkich wektorów cech.

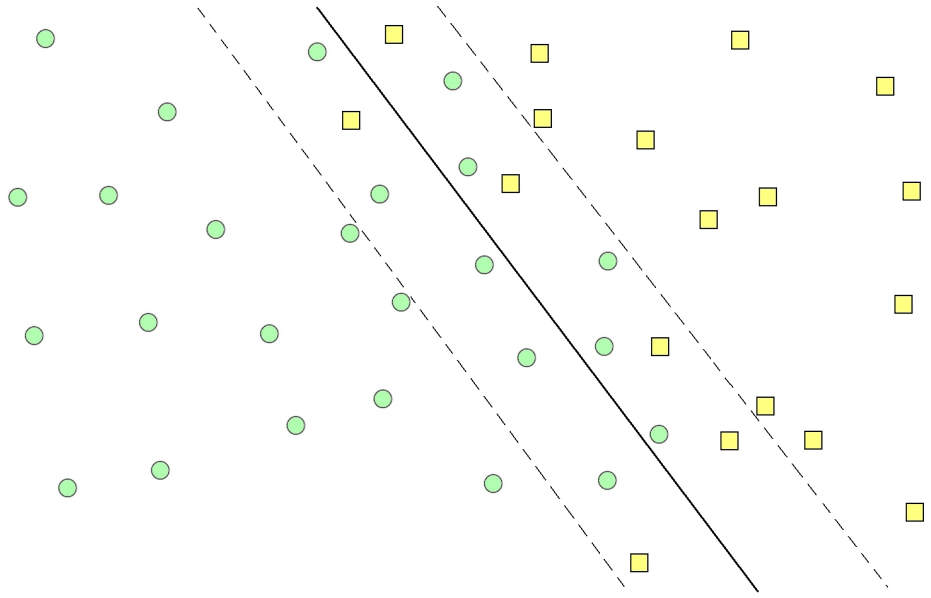
W przypadku wielu problemów jest jednak mało prawdopodobne, aby udało się znaleźć hiperpłaszczyznę, która idealnie będzie oddzielała od siebie próbki dwóch klas. Aby poradzić sobie z takimi problemami wprowadza się hiperpłaszczyznę ze słabymi marginesami (ang. soft margins). Rozwiązanie takie pokazane jest na rysunku 1.8.

Robimy to poprzez wprowadzenie zmiennych ξ_i reprezentujących błędy, to znaczy wektory cech, które będą leżeć wewnątrz marginesu. Dodatkowo wprowadzamy parametr C, który będzie pozwalał nam na ustalenie pewnego kompromisu pomiędzy maksymalizacją marginesu, a minimalizacją ilości błędów. W takim wypadku będziemy minimalizować funkcję:

$$\frac{1}{2}\|w\|^2 + C \sum_{i=1}^{n} \xi_i \tag{1.43}$$

gdzie C jest ustalonym współczynnikiem kary za niespełnienie idealnych ograniczeń. Stałą C dobieramy zwykle z pomocą k-kroswalidacji.

Te wszystkie działania nie przyniosą efektu jeśli nasz problem nie jest liniowo separowalny. Możemy jednak skorzystać z twierdzenia Covera, które mówi, że złożony



Rysunek 1.8. Podział przestrzeni cech z wykorzystaniem hiperpłaszczyzny ze słabymi marginesami.

problem klasyfikacyjny zrzutowany nieliniowo na przestrzeń wielowymiarową ma większe prawdopodobieństwo być liniowo separowalny.

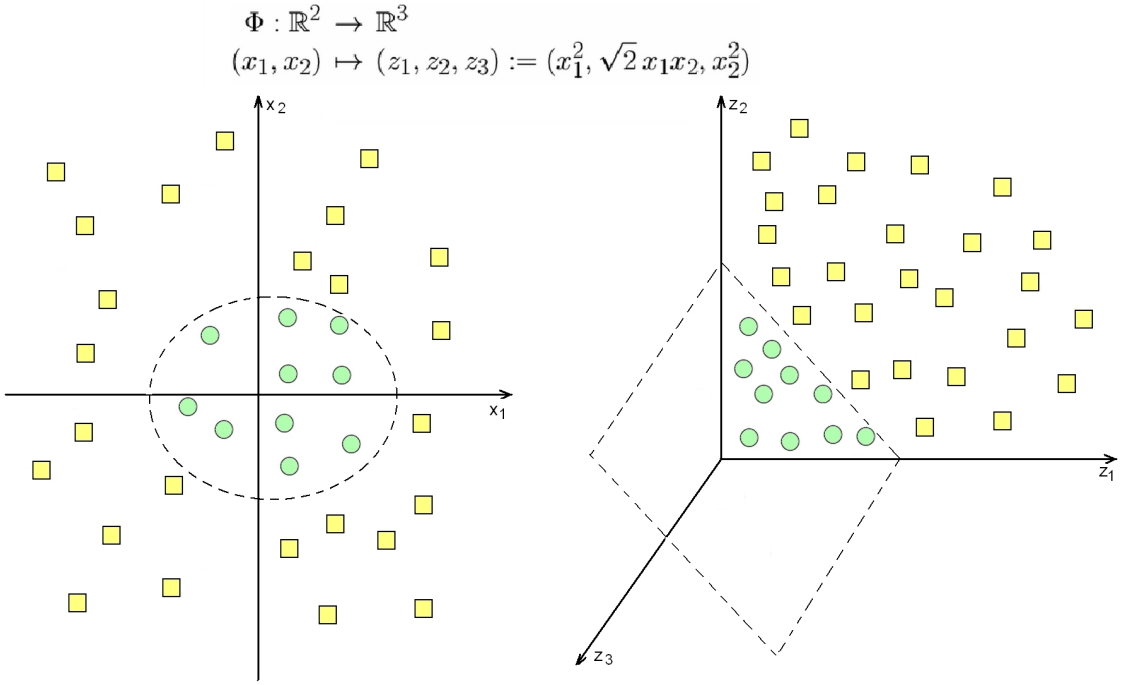
Rozwiązaniem zatem, jest przejście do przestrzeni o dużo większej ilości wymiarów, w której nasz problem będzie już liniowo separowalny. W takim wypadku wektory x_i, x z przestrzeni oryginalnej zostaną zastąpione we wzorze 1.42 przez ich przekształcenia, za pomocą pewnej nieliniowej funkcji, do przestrzeni o dużo większej wymiarowości $\Phi(x_i), \Phi(x)$. Otrzymamy wówczas:

$$f(x) = sign(\sum_{i=1}^{n} \alpha_i y_i \Phi(x_i)\Phi(x) + b) \tag{1.44}$$

W równaniu 1.44 jest używany iloczyn skalarny dwóch wektorów przestrzeni o dużej wymiarowości. Obliczanie tego iloczynu jest zwykle bardzo trudne obliczeniowo. Możemy jednak skorzystać z tak zwanych funkcji jądrowych (ang, kernel functions), aby obliczyć wartość iloczynu skalarnego w rozszerzonej przestrzeni. Po wstawieniu funkcji jądrowej $K(x_i, x)$ do wzoru 1.44 otrzymujemy:

$$f(x) = sign\Big(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x) + b\Big)\,, \tag{1.45}$$

gdzie $K(x_i, x)$ jest funkcją jądrową.



Rysunek 1.9. Przejście do przestrzeni cech o wyższej wymiarowości

W literaturze [34], [134] możemy spotkać wiele różnych funkcji jądrowych. Do najczęściej stosowanych należą:

1. Liniowa: $K(x_i, x) = X^T X_i$
2. Wielomianowa: $K(x_i, x) = (yx^T x + c)^d$
3. RBF (Radial Basis Kernel): $K(x_i, x) = -\gamma||x - x_i||^2, \quad \gamma > 0$
4. Gaussowska RBF: $K(x_i, x) = exp(-||x - x_I||)^2/2\sigma^2$
5. Sigmoidalna: $K(x_i, x) = tanh(yx^T x_i + c)$

W badaniach opisanych w niniejszej pracy została użyta funkcja jądrowa RBF, która dawała najlepsze rezultaty. Wybór tej funkcji pociąga za sobą konieczność wyboru dodatkowego parametru γ. Parametr ten wraz z parametrem C, był dobierany do każdego zadania klasyfikacji z osobna za pomocą procedury k-kroswalidacji metodą grid-search.

Warto też wspomnieć, że klasyfikator SVM może być traktowany jako sieć neuronowa (sieć SVM) stosująca różne funkcje aktywacji i implementująca specjalny sposób uczenia sprowadzający się do programowania kwadratowego [115]. Sieć SVM należy do sieci jednokierunkowych i ma strukturę dwuwarstwową składającą się z warstwy ukrytej i wyjściowej.

1.2.3. Klasyfikatory generatywne i dyskryminatywne

Możemy wyróżnić dwa podejścia do problemu klasyfikacji: podejście generatywne i dyskryminatywne. W pierwszym z nich podchodzimy do zadania klasyfikacji próbując zbudować model obejmujący całość problemu. W drugim interesuje nas jedynie możliwość

separacji klas. Załóżmy, że nasze zadanie polega na rozpoznaniu tekstu napisanego w nieznanym języku. Podejście generatywne będzie polegało na nauczeniu się wszystkich języków i na podstawie tej wiedzy odpowiednie zaklasyfikowanie tekstu. Podejście dyskryminatywne będzie polegało na skupieniu się jedynie na różnicach lingwistycznych pomiędzy językami.

W literaturze spotyka się pogląd, że w wielu dziedzinach klasyfikatory dyskryminatywne są bardziej efektywne niż klasyfikatory generatywne [117]. Niemniej jednak klasyfikatory generatywne mają wiele zalet, które pozwalają im konkurować z podejściem dyskryminatywnym, zwłaszcza gdy zbiór uczący jest mało liczny. W pracy [111] pokazano na przykład, że prosty klasyfikator generatywny (Naiwny Bayes) przewyższa swojego dyskryminatywnego odpowiednika (regresję logistyczną), gdy ilość próbek w zbiorze uczącym jest mała.

Jako przykłady podejścia generatywnego możemy wymienić klasyfikatory takie jak: Naiwny Bayes, Gaussians, analizę dyskryminacyjną [56] (QDA, LDA, RDA), HMM [48], czy też sieci bayesowskie. Dyskryminatywne podejście reprezentują między innymi SVM [157], sieci neuronowe, metody najbliższych sąsiadów, warunkowe pola losowe CRF (ang. Conditional Random Fields) [145] czy ukryte warunkowe pola losowe HCRF (ang. Hidden Conditional Random Fields) [167].

W literaturze możemy znaleźć przykłady łączenia tych klasyfikatorów (na przykład w pracy [119]). Podejście takie umożliwia wykorzystanie zalet obu podejść i osiągnięcie lepszego wyniku klasyfikacji. W niniejszej rozprawie zaprezentowana została nowa metoda tworzenia takiego połączenia wykorzystująca funkcję dyskryminacyjną klasyfikatora RDA. W rozdziale 5 opisany zostanie hybrydowy klasyfikator SVM-RDA, który został wykorzystany w przeprowadzonych badaniach.

1.2.4. Łączenie klasyfikatorów

W poprzednich podrozdziałach opisano wiele różnych podejść do problemu klasyfikacji, wiele różnych klasyfikatorów. Każdy z tych klasyfikatorów ma swoje zalety, ale ma też pewne wady. Idealnym rozwiązaniem byłoby zatem stworzenie takiej hybrydy, która wykorzystałaby zalety wszystkich klasyfikatorów składowych jednocześnie minimalizując ich wady.

Dodatkowym problemem, który pojawia się w przypadku każdego klasyfikatora jest niebezpieczeństwo przeuczenia. Polega ono na tym, że wybrany model klasyfikacji (klasyfikator, jego parametry, wektor cech) mimo, że osiąga bardzo dobre wyniki na zbiorze uczącym, to ma jednak słabe zdolności generalizacji problemu. To znaczy osiąga słabe wyniki na zbiorze nieznanych próbek (zbiorze testowym). Nie da się tego uniknąć ze względu na skończoną wielkość zbiorów danych. Co więcej często dysponujemy bardzo ograniczoną liczbą próbek reprezentujących klasy, a to dodatkowo pogłębia problem.

Rosnące możliwości komputerów również zachęcają do tworzenia coraz bardziej złożonych systemów. W literaturze możemy znaleźć wiele metodologii tworzenia zespołów klasyfikatorów (ang. ensemble classifiers). Możemy podzielić je na trzy grupy:

— Zestawy klasyfikatorów kombinowanych (ang. classifiers fusion), w których klasyfikatory składowe podejmują niezależne decyzje. Decyzje te są na wyższym poziomie scalane [85]. Najczęstszą metodą podejmowania decyzji na wyższym poziomie jest głosowanie większościowe (ang. majority voting).
— Zestawy klasyfikatorów lokalnych (ang. classifier selection). W tym wypadku każda próbka jest klasyfikowana przez jeden wybrany (”kompetentny”) klasyfikator. Przykłady zastosowania tej metodologii możemy znaleźć między innymi w pracach Woźniaka [162] czy Kunchevej [84].
— Zestawy klasyfikatorów kaskadowych (ang. cascade classifiers). Rozpoznawane próbki podawane są na wejście pierwszego klasyfikatora. Jeśli uzna on, że decyzja o przydziale do danej klasy jest obciążona zbyt dużą niepewnością, to próbka wysyłana jest na wejście kolejnego klasyfikatora. Zwykle kolejność klasyfikatorów jest z góry ustalona. Jako pierwszy stosujemy najszybszy z nich, który często odrzuca (przesyłając na wejście kolejnego klasyfikatora) dużo próbek. Przykład takiego klasyfikatora możemy znaleźć w pracy Barama [5].

Wszystkie opisane powyżej metodologie zakładają, że grupa klasyfikatorów osiągnie lepszy wynik niż pojedynczy klasyfikator. Można przytoczyć wiele prac [82], [91], [68], [86], [166], zarówno praktycznych jak i teoretycznych, które potwierdzają, że tak jest w istocie. Więcej, możemy zauważyć, że w wielu wypadkach pojedyncze klasyfikatory dają wyniki co prawda deterministyczne, ale też niestabilne. To znaczy, że przy modyfikacjach zbioru uczącego generują istotnie różniące się modele. Dzięki łączeniu różnych modeli generowanych na podstawie różnych zbiorów uczących możemy uzyskać znaczący przyrost poprawnych klasyfikacji [47].

Dziedzina łączenia klasyfikatorów rozwija się bardzo dynamicznie. Nie sposób w niniejszej pracy omówić choćby skrótowo wszystkich metod opisanych w literaturze. Dość obszerny przegląd można znaleźć w monografii Woźniaka [163]. Poniżej kilka z tych metod zostanie przedstawionych w zarysie.

Bootstraping

Technika ta polega na tworzeniu na podstawie jednego bazowego zbioru uczącego wielu innych zbiorów uczących. Zbiory te są o tej samej mocy co zbiór bazowy, ale mogą zawierać również powtórzone wektory cech. Metodologia ta wywodzi się ze statystyki i jest tam często stosowana do szacowania wartości pewnych parametrów. Tworzenie w ten sposób grup klasyfikatorów stabilizuje ich działanie i znacząco poprawia jakość klasyfikacji [49]. Wadą tej

metody jest jednak jej duża złożoność obliczeniowa, dlatego też większą popularność zyskały opisane poniżej, mniej złożone, metody bagging i boosting.

Bagging

Ta metoda została opracowana przez Breimana w 1996 roku [16]. Bagging (czyli Bootstrap aggregation) polega na wygenerowaniu wielu zbiorów uczących za pomocą losowania ze zwracaniem ze zbioru pierwotnego. W rezultacie niektóre próbki nie występują w zbiorze uczącym w ogóle, natomiast inne mogą pojawiać się w nim wielokrotnie. Wszystkie zbiory uczące mają taką samą liczebność jak zbiór pierwotny.

Następnie generowane są klasyfikatory dla każdego zbioru uczącego, a ostateczna decyzja podejmowana jest poprzez głosowanie większościowe tak utworzonych klasyfikatorów. Quinlan pokazał, że ta technika jest bardzo użyteczna w przypadku zadań klasyfikacji generujących niestabilne klasyfikatory [124]. Schemat klasyfikatora utworzonego metodą bagging został przedstawiony poniżej za pomocą pseudokodu:

1. $i \leftarrow 0$
2. Wygeneruj zbiór uczący U_i ze zbioru U metodą bootstrap
3. Skonstruuj regułę decyzyjną $d(U_i)$
4. Za pomocą $d(U_i)$ sklasyfikuj wszystkie próbki x_i
5. $i \leftarrow i + 1$
6. Jeśli $i < MaxIterations$ przejdź do 2
7. Zlicz liczbę głosów, które padły na każdą klasę $N(x_j)$
8. Przyporządkuj każdą próbkę do klasy, na którą padła największa liczba głosów

Boosting

Boosting to ogólna metoda mająca na celu zwiększenie skuteczności działania dowolnego algorytmu uczącego. Została ona zaproponowana przy okazji prac nad modelem uczenia PAC (ang. Probably Approximately Correct) [78], [156]. Technika ta polega na stosowaniu sekwencji prostych modeli, w taki sposób, aby każdy kolejny model przykładał większą wagę do tych obserwacji, które zostały błędnie zaklasyfikowane przez poprzednie modele.

Pierwszy taki algorytm został zaprezentowany w 1990 roku przez Schapire'a [133]. Algorytm ten miał jednak sporo praktycznych wad. W roku 1995 Freund i Schapire przedstawili algorytm AdaBoost (ang. Adaptive Boost) [55], który był pozbawiony wad swojego poprzednika.

Działanie tego algorytmu rozpoczyna się od nadania każdej próbce ze zbioru uczącego jednakowej wagi, tak aby suma wszystkich wag wynosiła 1. Każda waga reprezentuje stopień trudności klasyfikacji danej próbki przez klasyfikator, który jest tworzony w danej rundzie. Algorytm tworzy pewną określoną liczbę klasyfikatorów składowych (po jednym w każdej

rundzie) i dla każdego z tych klasyfikatorów określa jego wagę w końcowym głosowaniu. Każda runda rozpoczyna się od określenia zbioru uczącego na podstawie wag poszczególnych próbek. Następnie przy użyciu tego zbioru tworzony jest klasyfikator.

Kolejnym krokiem jest wyznaczenie błędu klasyfikatora na jego zbiorze uczącym. Określamy w ten sposób pewien współczynnik "ważności" klasyfikatora. Współczynnik ten jest tym większy im większa część próbek została poprawnie sklasyfikowana. Następnie modyfikujemy wagi próbek. Wagi próbek, które zostały błędnie sklasyfikowane zostają zwiększone, zaś wagi próbek, poprawnie zaklasyfikowanych zostają zmniejszone. Dzięki takiemu postępowaniu każdy następny klasyfikator koncentruje się na "trudniejszych" próbkach. Ostateczna decyzja klasyfikatora AdaBoost jest podejmowana za pomocą głosowania z wykorzystaniem wag utworzonych w ostatnim kroku algorytmu.. Pseudokod tego algorytmu jest zamieszczony poniżej:

1. $i \leftarrow 0$
2. $w_{0j} \leftarrow 1/n; \quad j = 1, 2, \ldots, n$
3. Wygeneruj zbiór uczący U_i ze zbioru U z uwzględnieniem wag
4. Skonstruuj regułę decyzyjną $d(U_i)$
5. Za pomocą $d(U_i)$ sklasyfikuj wszystkie próbki x_j
6. Na podstawie błędów popełnionych przez klasyfikator zmodyfikuj wagi w_{ij}
7. $i \leftarrow i + 1$
8. Jeśli $i < K$ przejdź do 3
9. Zlicz liczbę głosów, które padły na każdą klasę $N(x_j)$ uwzględniając wagi w_{Kj}
10. Przyporządkuj każdą próbkę do klasy, na którą padła największa liczba głosów

Zespoły klasyfikatorów

Zespoły to po prostu pakiety klasyfikatorów podejmujące decyzje za pomocą głosowania. W zasadzie w metodach Bagging i Boosting mamy również do czynienia z zespołami. Jednak w obu tych przypadkach są to zespoły takich samych klasyfikatorów. W ogólnym przypadku zespoły mogą łączyć wiele różnych klasyfikatorów wykorzystując mocne cechy każdego z nich.

W najprostszym przypadku ostateczna decyzja klasyfikatora jest podejmowana w głosowaniu. Zwycięża ta klasa, która otrzyma największą liczbę głosów. Czasami stosuje się także inne strategie. Na przykład większości głosów 50% + 1 lub jednomyślności. Tego typu schematy głosowania są stosowane głównie w medycynie, w zastosowaniach, w których podjęcie poprawnej decyzji jest niezwykle istotne. Taki klasyfikator, w przypadku nieuzyskania większości głosów, lub braku jednomyślności daje odpowiedź *"nie wiem"*, czyli odrzuca próbkę.

Innym podejściem stosowanym w przypadku zespołów jest podawanie prawdopodobieństwa przynależności do klasy na podstawie rozkładu głosów klasyfikatorów

składowych. Często też takim klasyfikatorom przydziela się wagi, na przykład na podstawie ich wiarygodności. Przez wiarygodność klasyfikatora składowego rozumiemy wynik jaki uzyskał on na zbiorze uczącym.

Jeszcze innym podejściem do schematu głosowania jest przypisywanie kompetencji poszczególnym klasyfikatorom w zakresie pewnych klas lub części przestrzeni cech. To podejście może być szczególnym przypadkiem przydziału wag. Z tą tylko różnicą, że wagi nie są w tym wypadku związane z klasyfikatorem, ale z konkretnymi klasami. Możemy też pójść jeszcze dalej i stosować różne wagi przydzielając je do pewnych wektorów cech.

Oczywiście w przypadku tworzenia klasyfikatora złożonego opartego o metodologię zespołów bardzo ważną kwestią jest w jaki sposób dobierać klasyfikatory składowe (członków zespołu). Często kierujemy się po prostu intuicją jednak należy zwrócić uwagę na trzy podstawowe założenia:

— *Niezależność statystyczna.* Jeśli w zespole będą występować klasyfikatory dokonujące identycznych klasyfikacji dla podzbiorów danych, to nie jest możliwe, aby wynik zespołu mógł być lepszy od wyniku pojedynczego modelu. A zatem ważne jest, aby klasyfikatory popełniały możliwie różne błędy. Z podobnym wymaganiem spotkamy się w jednym z kolejnych podrozdziałów 1.4.4 dotyczącym kodów ECOC.
— *Efektywność.* Działanie członków zespołu musi być stosunkowo szybkie, ponieważ tylko to pozwoli na uruchomienie w sensownym czasie odpowiedniej liczby klasyfikatorów składowych.
— *Jakość.* Dokładność klasyfikatorów składowych musi być znacząco lepsza od klasyfikatorów losowych w przeciwnym przypadku nastąpi pogorszenie wyniku osiągniętego przez zespół.

Oczywiście wszystkie te założenia trudno jest spełnić. W praktyce największą uwagę należy zwrócić na niezależność statystyczną, a w drugiej kolejności na jakość [47].

Lasy losowe

Lasy losowe (ang. random forest) to jedna z dość popularnych technik łączenia klasyfikatorów. Została zaproponowana przez Breimana w 1996 roku. To metoda stosowana do drzew decyzyjnych. Polega ona na tym, że najpierw losujemy K prób bootstrapowych, a następnie dla każdej z nich budujemy drzewo decyzyjne, w taki sposób, że w każdym węźle bierzemy pod uwagę m cech, które będą uczestniczyły w wyborze najlepszego podziału.

Zakładamy oczywiście, że $m < c$, gdzie c oznacza liczbę cech. Ostateczna obserwacja wyłaniana jest na podstawie głosowania. Zwykle zakłada się, że m powinno być znacznie mniejsze od wymiaru wektora cech. Często przyjmuje się $m = \sqrt{c}$. Ciekawy przykład zastosowania tej techniki znajdziemy w [67]. Poniżej zamieszczony jest pseudokod algorytmu wykorzystującego technikę lasów losowych.

1. $i \leftarrow 0$
2. Wygeneruj n elementowy zbiór uczący U_i ze zbioru U (losowanie ze zwracaniem)
3. Na podstawie U_i utwórz węzeł z wykorzystaniem m wylosowanych cech.
4. Stosując do nich przyjętą regułę dokonaj podziału węzła optymalizując wybrane kryterium
5. Jeśli po podziale zostały jakieś węzły, które nie są liśćmi, to przejdź do 3
6. Dokonaj klasyfikacji za pomocą utworzonego drzewa
7. $i \leftarrow i + 1$
8. Jeśli $i < K$ przejdź do 2
9. Zlicz liczbę głosów, które padły na każdą klasę $N(x_j)$
10. Przyporządkuj każdą próbkę do klasy, na którą padła największa liczba głosów

1.2.5. Metody oceny jakości klasyfikatorów

Budując jakikolwiek system podejmowania decyzji musimy się liczyć z tym, że będzie on w określonych wypadkach popełniał błędy. Są one nieuniknione, ponieważ nie mamy żadnych gwarancji, że rozpoznawane klasy są separowalne. Często również brakuje nam pełnej wiedzy na temat klasyfikowanych obiektów lub ilość próbek w zbiorze uczącym jest niewystarczająca do wytworzenia się prawidłowych reguł decyzyjnych klasyfikatora. W takich przypadkach pojawiają się błędy klasyfikacji.

Na początek przyjmijmy dla uproszczenia, że mamy do czynienia z problemem binarnym. Zadaniem naszego klasyfikatora jest zatem zakwalifikowanie próbki, która trafia na jego wejście do pewnej wyróżnionej klasy (niech to będzie klasa o etykiecie 1) lub jej odrzucenie, jako należącej do drugiej klasy (o etykiecie 0).

	obiekt należy do klasy 1	obiekt należy do klasy 0
klasyfikator przydzielił obiekt do klasy 1	TP	FP
klasyfikator przydzielił obiekt do klasy 0	FN	TN

Tablica 1.1. Rodzaje błędów klasyfikacji

W takim przypadku możemy wyróżnić cztery decyzje klasyfikatora, to jest: *prawidłowe wskazanie wyróżnionej klasy* TP (ang. true positive), *prawidłowe wskazanie drugiej klasy* TN (ang. true negative), *nieprawidłowe wskazanie drugiej klasy* FN (ang. false negative), *nieprawidłowe wskazanie wyróżnionej klasy* FP (ang. false positive). Schematycznie przedstawiono to w tabeli 1.1.

Dobry klasyfikator powinien zapewnić nam jak najlepsze wyniki, czyli najmniejszą liczbę błędów. W statystyce definiuje się dwie główne miary pozwalające na ocenę wyniku klasyfikatora: czułość Sn (ang. sensitivity)

$$Sn = \frac{TP}{TP + FN} \tag{1.46}$$

oraz specyficzność Sp (ang. specificity)

$$Sp = \frac{TN}{TN + FP} \tag{1.47}$$

Możemy również zdefiniować skuteczność klasyfikatora Acc (ang. accuracy) jako:

$$Acc = \frac{TP + TN}{TP + TN + FN + FP} \tag{1.48}$$

W przypadku wielu klas taka klasyfikacja błędów jest jednak niemożliwa. Dlatego też najczęściej błędy klasyfikatora przedstawia się w postaci macierzy pomyłek CF (ang. confusion matrix), w której wiersze odpowiadają poprawnym klasom decyzyjnym, a kolumny decyzjom przewidzianym przez klasyfikator. W takiej macierzy na przecięciu wiersza i oraz kolumny j mamy liczbę próbek n_{ij} należących oryginalnie do klasy i-tej, a zaliczonych do klasy j-tej. Oznacza to, że na przekątnej takiej macierzy mamy liczbę elementów, które zostały poprawnie sklasyfikowane.

W niniejszej pracy do oceny jakości klasyfikatora będziemy stosowali współczynnik poprawnych klasyfikacji Q (w skrócie: współczynnik klasyfikacji) [3], który będzie zdefiniowany następująco:

$$Q = \frac{c_1 + c_2 + ... + c_p}{n_1 + n_2 + ... + n_p} * 100\% = \frac{C}{N} * 100\% \tag{1.49}$$

gdzie $N = n_1 + n_2 + ... + n_p$ będzie liczbą próbek w zbiorze testowym, n_i liczbą próbek należących do klasy i, zaś $C = c_1 + c_2 + ... + c_p$ liczbą poprawnie rozpoznanych próbek (sumą wartości na przekątnej macierzy pomyłek), gdzie c_i liczba poprawnie rozpoznanych próbek klasy i.

Niestety jak widać z powyższego wzoru zdefiniowany współczynnik poprawności rozpoznawania zależy od wyboru zbioru testowego (jak również od ilości próbek, które użyjemy do testowania). Jeśli zatem zbiór testowy będzie źle dobrany lub zbyt mało liczny, to nasz wynik będzie obciążony. W literaturze opisanych jest wiele metod radzenia sobie z tym problemem. Możemy je podzielić na cztery główne podejścia [87], [73].

— metoda resubstytucji (ang. resubstitution method) – cały zbiór danych jest używany w procesie uczenia, a następnie wykorzystujemy go do testowania. Metoda ta jest

całkowicie niepoprawna, ponieważ powoduje zawyżanie wyniku poprzez dostosowanie się klasyfikatora do testowanego zbioru.

— metoda wydzielania (ang. holdout method) – Losowo lub arbitralnie ustalona część zbioru (zwykle około połowy) służy do uczenia się klasyfikatora jako zbiór uczący, a druga część do testowania. Wadą tej metody jest zwykle pesymistyczne obciążenie klasyfikatora.

— metoda minus jednego elementu (ang. leave-one-out method). W tej metodzie generowane jest N zbiorów uczących, gdzie N oznacza liczbę próbek, którymi dysponujemy. Następnie każdy z tych zbiorów staje się zbiorem uczącym. Klasyfikator testujemy zaś na jednoelementowym zbiorze testowym, którego elementem jest pozostała próbka. Metoda ta daje nieobciążony wynik jednak jest bardzo kosztowna obliczeniowo. Zwłaszcza w przypadku, gdy dysponujemy bardzo dużą liczbą próbek.

— metoda k-krotnej walidacji skrośnej (ang. k-fold crossvalidation), często nazywana metodą k-kroswalidacji – Ta metoda stanowi pewien kompromis pomiędzy metodą wydzielania, a metodą minus jednego elementu. Polega ona na podziale zbioru próbek na k losowych podzbiorów (ang.folds). Każdy podzbiór zawiera tyle samo (lub prawie tyle samo) elementów. Następnie każda z k części staje się po kolei zbiorem testowym, zaś pozostałe $k-1$ części zbiorem uczącym.

— metoda k-krotnej warstwowej walidacji skrośnej (ang. k-fold layer crossvalidation) – Jest to odmiana metody k-krotnej walidacji. W przypadku, gdy część klas reprezentowana jest przez stosunkowo niewielką liczbę próbek może się zdarzyć, że w przypadku losowego podziału na k podzbiorów, pojawią się takie podzbiory, w których nie będzie ani jednej próbki danej klasy. Aby tego uniknąć w metodzie k-krotnej kroswalidacji warstwowej zakładamy, że w każdym k podzbiorów znajdzie się taka sama (lub prawie taka sama) liczba próbek każdej klasy.

W niniejszej pracy stosowane były dwie z powyżej opisanych metod: metoda wydzielania oraz metoda k-krotnej warstwowej walidacji skrośnej. Metoda wydzielania była stosowana jednak wyłącznie w celu porównania wyników osiąganych przez algorytmy przedstawione w tej pracy ze znanymi z literatury rozwiązaniami zaproponowanymi przez innych autorów.

Rzetelne porównanie różnych klasyfikatorów nie jest sprawą prostą. Sam współczynnik poprawnych klasyfikacji jest przecież wartością punktową uzyskaną na pewnym konkretnym zbiorze testowym. Nie wiemy czy porównanie tych samych klasyfikatorów na innym zbiorze nie da zupełnie innych wyników.

Jedynym ze sposobów oceny jakości klasyfikatora jest krzywa ROC (ang. Receiver Operating Characteristic). Jest to krzywa, która charakteryzuje związek pomiędzy specyficznością S_p, a czułością S_n. Zwykle na osi rzędnych odkładamy wartości $1 - Sp$, a na osi odciętych wartości S_n. Idealny klasyfikator będzie osiągał wartości $1 - S_p = 0$ i $S_n = 1$ dla

pewnego punktu odcięcia reguły decyzyjnej. Klasyfikator losowy będzie zaś charakteryzował się wykresem pokrywającym się z przekątną $y = x$.

Krzywe ROC możemy wykorzystywać do porównywania ze sobą różnych klasyfikatorów. Bardzo częstym sposobem jest obliczanie pola pod wykresem krzywej ROC. Pole to oznaczane jest jako AUC (ang. Area Under Curve) i traktowane jest jako miara trafności danego klasyfikatora [14]. Wskaźnik ten przyjmuje wartości z przedziału $[0, 1]$. Im bardziej jego wartość zbliżona jest do jedynki, tym lepszy klasyfikator. Możemy zatem porównując klasyfikatory porównywać wartości pól AUC.

Stosując metodę k-kroswalidacji możemy podawać wynik średni oraz wyniki minimalny i maksymalny, uzyskane na kolejnych podzbiorach testowych. Taki sposób prezentacji daje nam pewien pogląd na to jakich wyników możemy spodziewać się po danych klasyfikatorach. Jednak trudno na tej podstawie jednoznacznie stwierdzić, że jeden klasyfikator jest istotnie lepszy od drugiego.

problem	wynik klasyfikatora		
	RDA	SVM	SVM-RDA
białka	56,1%	57,2%	62,1%
	52,1% – 58.3%	54,2% – 59,3%	60,1% – 64,2%
gesty (A)	88,0%	81,1%	89,3%
	83,2% – 93,6%	76,3% – 86,4%	85,1% – 93,6%
gesty (B)	91,3%	89,8%	93,1%
	86,2% – 95,4%	86,2% – 92,4%	89,1% – 96,4%
cyfry	98,82%	99,11%	99,13%
	98,12% – 99,21%	98,73% – 99,44%	98,81% – 99,55%

Tablica 1.2. Współczynniki klasyfikacji osiągnięte dla kilku baz danych

W tabeli 1.2 zestawiono wyniki osiągnięte przez klasyfikatory RDA, SVM i SVM-RDA na kilku różnych bazach danych. W pierwszym wierszu zostały podane wyniki średnie, zaś w drugim wynik minimalny i maksymalny. Jak łatwo zauważyć wyniki maksymalny i minimalny różnią się znacząco, nierzadko nawet o dziesięć punktów procentowych.

Analizując tabelę 1.2, na podstawie wyników średnich, moglibyśmy stwierdzić, że klasyfikator SVM-RDA osiąga najlepsze wyniki. Jednak po głębszej analizie rodzą się dwa pytania. Pierwsze, to czy te różnice, niekiedy bardzo małe (na przykład 0,02%, 0,08%), nie są dziełem przypadku. Drugie to z jakim prawdopodobieństwem możemy się spodziewać konkretnego wyniku. Na oba te pytania postaramy się odpowiedzieć w dalszej części tego tekstu.

1.2.6. Estymacja bootstrapowa

Zacznijmy od próby odpowiedzi na drugie pytanie. Przyjrzyjmy się konstrukcji pewnego znanego ze statystyki narzędzia tj. przedziału ufności. Poniżej przedstawiona zostanie definicja i sposób wyznaczania tego pojęcia, zgodnie z pracą [142].

Przedział ufności dla dowolnego parametru α rozkładu zmiennej losowej X możemy zbudować na podstawie jakiegoś estymatora punktowego $T_n = h(X_1, X_2, \ldots, X_n)$ tego parametru. Rozkład tego parametru musi być znany. Jednak w przypadku klasyfikatora tego rozkładu nie znamy, a zatem musimy go aproksymować na podstawie N realizacji próby bootstrapowej $(X_1^{*i}, X_2^{*i}, \ldots, X_n^{*i})$. W literaturze przyjmuje się, że wartość N powinna wynosić co najmniej 1000.

Realizację $(x_1^{*i}, x_2^{*i}, \ldots, x_n^{*i})$ próby bootstrapowej generuje się przez losowanie ze zwracaniem spośród posiadanych elementów próby $(x_1, x_2, \ldots, x_n)$ (w naszym przypadku wyników otrzymanych za pomocą k-kroswalidacji), które traktujemy jako populację. Przez N-krotne generowanie takich realizacji uzyskujemy ciąg N wartości statystyki T_n:

$$(t_n^{*1}, t_n^{*2}, \ldots, t_n^{*N}) \tag{1.50}$$

czyli:

$$(h(x_1^{*1}, x_2^{*1}, \ldots, x_n^{*1}), h(x_1^{*2}, x_2^{*2}, \ldots, x_n^{*2}) \ldots, h(x_1^{*N}, x_2^{*N}, \ldots, x_n^{*N})) \tag{1.51}$$

Na jego podstawie możemy zbudować rozkład empiryczny estymatora T_n, który będziemy nazywać rozkładem bootstrapowym.

Dla wyznaczenia bootstrapowego przedziału ufności dla parametru α można wykorzystać tak zwane percentyle rozkładu bootstrapowego statystyki T_n.

Percentylem rzędu α $(0 < \alpha < 1)$ nazywamy wartość:

$$p^*(\alpha) = T_n^{*\alpha N} = h(x_1^{*\alpha N}, x_2^{*\alpha N}, \ldots, x_1^{*\alpha N}) \tag{1.52}$$

zajmującą w uporządkowanym ciągu 1.51 pozycję o numerze $\alpha * N$. Bootstrapowy przedział ufności na poziomie ufności $1 - \alpha$ dla parametru α ma zatem postać:

$$(p^*(\alpha/2), p^*(1 - \alpha/2)) \tag{1.53}$$

Jeśli zatem oszacujemy przedziały ufności dla wyników poszczególnych klasyfikatorów, to będziemy mogli określić z jakim prawdopodobieństwem α wynik klasyfikatora mieści

się w danym przedziale. Im mniejszą przyjmiemy wartość parametru α tym większe prawdopodobieństwo, że wynik będzie mieścił się w danym przedziale. Trzeba jednak pamiętać, że w takim wypadku zakres wyników obejmowanych przez przedział ufności będzie szerszy. Zwykle przyjmuje się przedziały na poziomie ufności 95%, czyli dla $\alpha = 0,05$.

Istność statystyczna wyniku klasyfikacji

W poprzednim podrozdziale wskazane zostało narzędzie, które pozwala nam stwierdzić, że z pewnym prawdopodobieństwem wynik klasyfikatora będzie zawierał się w jakimś przedziale wartości. Jeśli zatem chcielibyśmy porównać dwa klasyfikatory, możemy wyznaczyć przedziały ufności dla osiąganych przez nie wyników. Jeśli będą to rozłączne przedziały, to z określonym prawdopodobieństwem możemy stwierdzić, że jeden z nich jest lepszy od drugiego. Niestety z taką sytuacją mamy do czynienia dość rzadko. Zwykle wyznaczone przez nas przedziały ufności będą miały niepuste przecięcie. Wracamy zatem do pierwszego pytania, czy różnice w wynikach osiąganych przez klasyfikatory nie są dziełem przypadku.

Rzetelne porównanie jakości dwóch klasyfikatorów powinno zawierać ocenę prawdopodobieństwa, że jeden z nich osiągnie lepszy wynik niż drugi. Spróbujmy zatem wykorzystać tutaj metody weryfikacji hipotez statystycznych. Jeżeli założymy, że wyniki klasyfikacji mają rozkład normalny, to możemy uwzględnić ich wartości oczekiwane oraz odchylenia standardowe i obliczyć odpowiednie prawdopodobieństwo.

Niestety nie znamy ani wartości oczekiwanej ani odchylenia standardowego. Co więcej zazwyczaj dysponujemy zbyt małą ilością wyników klasyfikacji, a to oznacza, że nie możemy dobrze estymować tych wartości. Dlatego lepszą metodą jest skorzystanie z testu t-studenta. Używając go możemy sprawdzić czy hipoteza o statystycznej istotności różnicy w wynikach dwóch klasyfikatorów jest prawdziwa (z określonym prawdopodobieństwem p), czy też należy ją odrzucić.

A zatem oznaczmy przez $A_1, A_2, \ldots, A_n$ wynki osiągane przez klasyfikator A oraz przez $B_1, B_2, \ldots, B_m$ wyniki osiągane przez klasyfikator B. Wtedy możemy oszacować wartość oczekiwaną jako:

$$\mu_A = \frac{1}{n}\sum_{i=1}^{n} A_i, \quad \mu_B = \frac{1}{n}\sum_{i=1}^{m} B_i \tag{1.54}$$

oraz wariancję tych rozkładów jako:

$$\sigma_A^2 = \frac{1}{n-1}(\sum_{i=1}^{n} A_i^2 - \frac{1}{n}(\sum_{i=1}^{n} A_i)^2) \tag{1.55}$$

$$\sigma_B^2 = \frac{1}{m-1}(\sum_{i=1}^{m} B_i^2 - \frac{1}{m}(\sum_{i=1}^{m} B_i)^2) \tag{1.56}$$

A zatem wariancję dla całej populacji będziemy szacować za pomocą średniej ważonej jako:

$$\sigma^2 = \frac{(n-1)\sigma_A^2 + (m-1)\sigma_B^2}{(n-1)+(m-1)} \tag{1.57}$$

zaś wariancja dla $\mu_A - \mu_b$

$$\sigma_\mu = \frac{\sigma^2}{n} + \frac{\sigma^2}{m} \tag{1.58}$$

zatem ostatecznie wartość testu t-studenta wyniesie:

$$t = \frac{\mu_A - \mu_b}{\sigma^2} \tag{1.59}$$

Teraz jeśli $t \geq t_{p,n+m-2}$, gdzie $t_{p,n+m-2}$ jest kwantylem dla prawdopodobieństwa p rozkładu t-studenta o $n+m-2$ stopniach swobody, to możemy powiedzieć, że z prawdopodobieństwem p hipoteza o tym, że klasyfikator A daje lepsze wyniki od klasyfikatora B jest prawdziwa. W przeciwnym wypadku odrzucamy tę hipotezę.

Ten test także nie jest idealny [36], ponieważ należy pamiętać, że można go stosować jedynie przy założeniu, że wyniki klasyfikatorów mają rozkład normalny oraz że każdy wynik został wygenerowany niezależnie z odpowiedniego rozkładu. O ile to pierwsze założenie jest zwykle prawdziwe, o tyle w drugim przypadku musimy pamiętać, że w rzeczywistości jest ono zwykle niemożliwe do spełnienia.

Dysponujemy przecież skończonym (i zwykle niezbyt licznym) zbiorem próbek. Jeśli więc stosujemy k-krotną kroswalidację, to $k-1/k$ każdego ze zbiorów uczących jest taka sama, a to oznacza, że wyniki nie są niezależne. Jednak mimo tych zastrzeżeń powyższa metoda dobrze rozwiązuje problem porównania wyników dwóch klasyfikatorów.

1.3. Selekcja cech

Jak już zostało wspomniane wcześniej, w zadaniu klasyfikacji rozpoznawane obiekty są reprezentowane przez wektory cech. To znaczy pewne mierzalne wielkości poddające się naszej obserwacji lub pomiarowi. Niestety bardzo często nie wiadomo, które z tych wartości są istotne z punktu widzenia algorytmu klasyfikacji. To powoduje, że zwykle dostarcza się możliwie jak największą ich liczbę tak, aby nie pominąć żadnej, która mogłaby być potencjalnie istotna i która mogłaby poprawić wynik klasyfikacji.

W roku 1997, w specjalnym przeglądowym numerze Artificial Intelligence [144] poświęconemu istotności wnioskowania (ang. relevance reasoning) znalazło się kilka prac dotyczących selekcji cech. Większość z nich zajmowała się zbiorami cech liczącymi mniej niż 40 elementów. Jednak już w 2003 roku Guyon w swojej pracy An Introduction to Variable

and Feature Selection [63] opisuje wektory liczące sobie nawet do stu tysięcy cech. Z kolei w 2009 roku do bazy danych UCI Machine Learning Repository [154] został dodany zbiór ”URL reputation”, w którym pojawiają się wektory cech liczące 3,2 miliona elementów.

lata	ilość baz danych	średnia ilość klas	średnia ilość cech
1988–1992	41	5,6	26
1993–1997	25	8,2	46
1998–2002	16	14,3	147
2003–2007	3	—	—
2008–2012	55	35,3	5776 (61500)

Tablica 1.3. Średnia ilość klas oraz średnia wymiarowość wektora cech w bazach danych dodawanych do UCI Machine Learning Repository w poszczególnych latach

Dość dobry pogląd na rosnącą wymiarowość wektora cech w problemach klasyfikacji daje analiza baz danych dodawanych w poszczególnych latach do zbioru UCI Machine Learning Repository. W tabeli 1.3 przedstawiono średnią wymiarowość wektora cech w latach 1988–2012. Wyniki zostały uśrednione w kolejnych pięcioletnich okresach.

Okres 2003–2007 został pominięty, ponieważ w tym okresie zostały dodane tylko trzy bazy danych. Wynik za lata 2008–2012 został podany z pominięciem jednej z baz danych: ”URL reputation”, w której wektor cech liczy sobie ponad 3,2 miliona cech (wynik uwzględniający tę bazę danych jest podany w nawiasach). Przeglądając tę tabelę widać jak szybko rośnie wymiarowość wektora cech. Od zaledwie 26 cech w latach 1988–1992, do 5776 cech w latach 2008–2012.

Gwałtownie rosnąca liczba cech stanowi bardzo poważny problem. Zwiększa ona złożoność obliczeniową algorytmu, jego wymagania pamięciowe, wydłuża proces uczenia oraz powoduje wzrost złożoności samego klasyfikatora. Jednak co najważniejsze często powoduje także spadek liczby poprawnie sklasyfikowanych obiektów. Jest to związane z tak zwanym *”przekleństwem wymiarowości”* (ang. curse of dimensionality) [141]. Ze zjawiskiem tym mamy do czynienia, gdy liczebność zbioru uczącego jest mała w stosunku do ilości cech. Na przykład we wspomnianym już wcześniej zbiorze ”URL reputation” mamy 3,2 miliona cech podczas gdy sam zbiór liczy sobie tylko 2,4 miliona próbek.

Rozwiązaniem powyższych problemów jest stosowanie algorytmów selekcji cech. Zadaniem tych algorytmów jest wybranie ze zbioru wszystkich cech, reprezentujących obiekty, pewnego podzbioru. Ten podzbiór jest następnie używany w procesie uczenia się klasyfikatora oraz w procesie klasyfikacji. Zadaniem algorytmu selekcji cech jest takie dobranie tego podzbioru, aby wynik klasyfikatora był jak najlepszy. Wśród wielu zalet stosowania selekcji cech wymienić możemy:

— Redukcję wymiarowości przestrzeni cech, która zmniejsza wymagania pamięciowe stosowanych algorytmów, jak również zmniejsza ich złożoność obliczeniową.

— Zmniejszenie ilości wolnych parametrów w klasyfikatorze koniecznych do oszacowania.
— Lepsze zrozumienie danego problemu. Na przykład, w systemach diagnostyki medycznej, łatwiej uda się zrozumieć, które symptomy (cechy) są istotne dla rozpoznania danej choroby, jeśli ich ilość ograniczymy z kilkudziesięciu do kilkunastu.
— Zmniejszenie kosztów akwizycji danych. Zbierając dane kolejny raz możemy skupić się wyłącznie na cechach istotnych dla algorytmu klasyfikacji.

Istnieje bardzo wiele metod selekcji cech opisanych w literaturze prezentujących bardzo różnorodne podejście do problemu. Pełen ich przegląd, nawet pobieżny, wymagałby napisania sporej monografii. Dlatego też, w tym rozdziale skupimy się na opisaniu trzech głównych metodologii tworzenia algorytmów selekcji cech. Opisane zostaną metody rankingowe często nazywane też filtrami (ang. filter methods), metody opakowane (ang. wrapper methods) oraz metody wbudowane (ang. embedded methods). Często metody te są łączone tworząc hybrydy. W niektórych pracach wyróżnia się połączenia metod rankingowych i obudowanych (ang. frappers).

W poniższych podrozdziałach zostanie przedstawiona idea każdego z tych podejść do problemu selekcji cech. Zostaną przedstawione przykładowe algorytmy reprezentujące daną metodologię. Wskazane również zostaną zalety i wady każdej z nich.

1.3.1. Metody rankingowe

Do zadania selekcji cech możemy podejść w następujący sposób [89]. Wśród cech opisujących obiekty wyróżniamy trzy grupy: cechy istotne (ang. relevant), nieistotne (ang. irrelevant) oraz redundantne (ang. redundant). Pierwsze z nich to te, które dobrze ”odróżniają” klasy od siebie, wpływając na poprawę efektywności algorytmu klasyfikacji. Cechy nieistotne to takie, których wartości są przypadkowe, w każdej z klas. Zwykle nie tylko nie powodują poprawy efektywności klasyfikacji, ale nawet ją pogarszają. Trzecia grupa cech, to te których role mogą przejąć inne cechy.

Istotą metod rankingowych jest próba znalezienia jakiejś miary, która pozwoli stworzyć ranking cech (biorąc pod uwagę ich istotność) i w ten sposób pomoże wyeliminować cechy nieistotne. Do stworzenia takiej metody potrzebny jest pewien współczynnik określający, dla każdej z osobna cechy, jej jakość wedle przyjętego kryterium. Wyznaczanie tego kryterium jest niezależne od używanego klasyfikatora, co stanowi jedną z głównych zalet, ale i jednocześnie wad tego podejścia.

Istnieje wiele sposobów budowania kryteriów. Główne grupy algorytmów to metody oparte na korelacji wartości danej cechy z numerem (etykietą) klasy [135], odległościach pomiędzy ich rozkładami, miarach pochodzących z teorii informacji [153] i kryteriach używanych w drzewach decyzji. Ciekawy przegląd tych metod można znaleźć w pracach [46], [54].

Wszystkie te metody opierają się na podobnej zasadzie. Cechy są oceniane przy pomocy kryterium, a następnie są sortowane na podstawie jego wartości. Ta uporządkowana lista cech to ranking. Selekcja polega na wyborze najlepszych cech z rankingu. Oznacza to konieczność wyboru dodatkowego parametru, którym zwykle jest pewna liczba pierwszych cech, które będą użyte w zadaniu klasyfikacji. Pozostałe cechy z rankingu są odrzucane.

Bardzo często stosowaną miarą wykorzystywaną w metodach rankingowych jest współczynnik korelacji Pearsona [74]. Jest on zdefiniowany następująco:

$$\mathcal{R}(i) = \frac{cov(X_i, Y)}{\sqrt{var(X_i)var(Y)}} \tag{1.60}$$

gdzie X_i jest i-tym współczynnikiem wektora cech, $var(X_i)$ - wariancją, a $cov(X_i, Y)$ – kowariancją.

Oczywiście zwykle nie znamy tych wartości, więc musimy posługiwać się ich estymatorami obliczonymi na podstawie zbioru uczącego. Wtedy współczynnik ten przyjmuje następującą postać:

$$\mathcal{R}(i) = \frac{\sum_{k=1}^{m}(x_{ki} - \bar{x_i})(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{ki} - \bar{x_i})^2 \sum_{k=1}^{m}(y_k - \bar{y})^2}} \tag{1.61}$$

gdzie x_{ki} - i-ty współczynnik k-tego wektora cech, y_k - klasa do której należy k-ty wektor cech, $\bar{x_i}, \bar{y}$ - to odpowiednio wartości średnie po indeksie k.

Największą zaletą selekcji rankingowej jest jej uniwersalność. Uzyskany podzbiór cech jest niezależny od używanego klasyfikatora, a także od problemu klasyfikacji. Drugą ważną cechą jest jej niska złożoność obliczeniowa. Oczywiście zależy ona od użytego algorytmu wyznaczania kryterium, jednak zwykle algorytm ten jest wielokrotnie mniej kosztowny od algorytmu uczenia się klasyfikatora czy też algorytmu klasyfikacji.

Łatwo zauważyć jednak, że algorytmy te nie odrzucają cech redundantnych. Można próbować skorygować tę niepożądaną własność tych metod poprzez dwuetapową selekcję rankingową. Na przykład w pracy [165] został zaproponowany algorytm, który w pierwszym kroku porządkuje cechy ze względu na ich istotność, a następnie w kolejnym kroku, ze względu na ich redundancję. Przy czym za cechę redundantną uważa się cechę, która jest bardziej skorelowana z cechą istotniejszą niż klasą.

Wadą metod rankingowych jest to, że z definicji, nie uwzględniają one zależności pomiędzy cechami. Cechy potencjalnie nieistotne, a zatem odrzucane przez te algorytmy jako zajmujące niskie miejsca w rankingu, mogą w połączeniu z innymi, znacznie poprawiać efektywność klasyfikacji. Co więcej metody te nie uwzględniają również specyfiki danego algorytmu klasyfikacji. Może to być zaletą, gdy najważniejszym kryterium jest dobór uniwersalnego

wektora cech. Jednak zwykle celem, który jest w badaniach najważniejszy, jest maksymalizacja wyników dla konkretnego klasyfikatora. Opisane w następnym podrozdziale metody nie mają wymienionych wyżej wad.

1.3.2. Metody opakowane

W odróżnieniu od metod rankingowych, gdzie proces selekcji jest niezależny od klasyfikatora, ocena podzbiorów cech w metodach opakowanych jest dokonywana przy użyciu konkretnego modelu klasyfikacyjnego. Miarą jakości podzbioru jest efektywność klasyfikatora, oszacowana na przykład przy użyciu kroswalidacji lub zbioru weryfikacyjnego. Metoda selekcji cech jest opakowana wokół tego modelu – stąd jej nazwa. Ten sam model używany jest w trakcie selekcji, jak i później, do klasyfikacji. Oczywiście, znalezienie optymalnego podzbioru jest w praktyce najczęściej niemożliwe ze względu na ilość możliwych kombinacji podzbiorów cech, która rośnie wykładniczo wraz z ilością elementów tego zbioru.

Jeśli liczba cech nie jest zbyt duża, to oczywiście możemy przeszukać ich wszystkie możliwe kombinacje. Jednak już przy dwudziestu cechach liczba wszystkich kombinacji zaczyna zbliżać się do miliona, a przy trzydziestu do miliarda. Istnieje wiele strategii zaprojektowanych w celu osiągnięcia rozsądnego kompromisu pomiędzy jakością znalezionego (nieoptymalnego) podzbioru cech, a czasem obliczeń.

Wiele z tych strategii można znaleźć w pracy [127]. Rozwijają one dwa podejścia: stopniowo dodają zmienne startując ze zbioru pustego (ang. forward selection), albo je odrzucają zaczynając ze wszystkimi dostępnymi cechami (ang. backward selection). Uwzględnienie współdziałania cech odbywa się za cenę złożoności obliczeniowej, znacznie zwiększonej poprzez proces przeszukiwania. Stąd metody te dobrze sprawdzają się z raczej szybkimi klasyfikatorami.

Zalety tego podejścia, to przede wszystkim jego większa efektywność. Dzięki temu, że do oceny wybieranego podzbioru cech używamy klasyfikatora, to rozwiązanie to daje dużo lepsze efekty. Powoduje to niestety, że wybrany podzbiór cech nie jest uniwersalny i gdy zostanie użyty inny algorytm proces selekcji cech musi zostać powtórzony.

Kolejną zaletą jest uwzględnianie korelacji cech. W przypadku metod rankingowych, jeśli dwie cechy zostaną umieszczone nisko w rankingu, to nie zostaną wybrane do szukanego podzbioru cech, nawet jeśli po połączeniu ich własność generalizacji będzie bardzo wysoka. W metodach opakowanych miarą stosowaną do oceny jakości podzbioru cech jest wynik klasyfikacji, a zatem takie cechy zostaną uwzględnione przez algorytm.

Widoczną wadą tych metod jest ich dużo większa złożoność obliczeniowa. Konieczność korzystania z klasyfikatora na każdym etapie oceny podzbioru cech powoduje, że metody te są dość kosztowne. Dodatkowo sprawę pogarsza fakt, że aby uniknąć dostosowania się

do specyfiki konkretnego problemu, i tym samym uzyskania obciążonego wyniku, musimy korzystać z techniki kroswalidacji.

Metody SFS i SBS

Algorytmy sekwencyjnego przeszukiwania w przód SFS (ang. sequential forward selection) i sekwencyjnego przeszukiwania w tył SBS (ang. Sequential Backward Selection) to przykłady prostych metod opakowanych. W przypadku pierwszej metody algorytm startuje od pustego podzbioru cech, dodając w każdym kolejnym kroku nową cechę. Dodawana cecha maksymalizuje pewną funkcję kryterialną. Najczęściej wartością funkcji kryterialnej jest wynik klasyfikatora na znalezionym dotychczas podzbiorze cech, do którego dodano testowaną cechę. W odwrotny sposób działa algorytm SBS. Zaczynając od pełnego podzbioru cech, w każdym kolejnym kroku odejmuje od niego jedną cechę maksymalizując funkcję kryterialną. Schematycznie algorytm SFS przedstawiony został poniżej:

1. $S \leftarrow \emptyset$
2. $J_{max} \leftarrow 0$, $m \leftarrow$ liczba wszystkich cech
3. Dla wszystkich cech $f_i \notin S$, policz $J(S \cup \{f_i\})$
4. $J_{max} \leftarrow max_{i=1,\ldots,m}\{J(S \cup \{f_i\})\}$, $f_{max} = argmax_{i=1,\ldots,m}\{J(S \cup \{f_i\})\}$
5. Jeśli $J \geq J_{max}$ zakończ działanie algorytmu
6. W przeciwnym przypadku $S \leftarrow S \cup \{f_{max}\}$, $m \leftarrow m - 1$
7. Jeśli $m = 0$ to zakończ działanie algorytmu
8. Przejdź do 3.

Obie te metody są często stosowane w praktyce ze względu na swoje zalety:

— Są bardzo proste w implementacji,

— Po każdym przebiegu algorytmu podzbiór cech jest powiększany/pomniejszany o jedną cechę.

Jak można łatwo zauważyć maksymalizacja funkcji kryterialnej jest najbardziej kosztowną operacją tego algorytmu, ponieważ wymaga uruchomienia m zadań klasyfikacji (gdzie m oznacza liczbę cech) w pierwszym kroku. W każdym następnym kroku uruchamiamy klasyfikator o jeden raz mniej. Oznacza to, że w pesymistycznym przypadku będziemy musieli uruchamiać klasyfikator $m * (m + 1)/2$ razy. Oznacza to, że całkowita złożoność takiego algorytmu cech wynosi $O(m^2) *$ złożoność klasyfikatora.

Trudno jest porównać wyniki uzyskiwane metodami SFS i SBS ze sobą. Zależą one od specyfiki optymalnego podzbioru cech. Generalnie jeśli optymalny podzbiór cech jest mały, to lepsze wyniki daje metoda SFS. Jeśli optymalny podzbiór cech zbliżony jest do zbioru pełnego wtedy lepsze wyniki uzyskuje metoda SBS. Ciekawym kompromisem pomiędzy tymi metodami jest metoda BDS (ang. BiDirectional Search) opisana w jednej z kolejnych sekcji niniejszej pracy.

Warto tutaj zauważyć, że algorytm SBS w pewnych sytuacjach może być całkowicie nieużyteczny. Załóżmy, że korzystamy z klasyfikatora RDA. Jeżeli wymiarowość wektora cech jest większa od ilości próbek (problem SSS), to estymowane macierze kowariancji będą osobliwe. To oznacza, że klasyfikator ten nie będzie mógł zostać użyty. Z drugiej strony algorytm SFS będzie działał poprawnie, ponieważ startuje on od jednoelementowych wektorów cech.

Obie te metody mają podobne wady. Pierwsza z nich cierpi na tak zwany efekt gniazda. Jeśli jakaś cecha zostanie już dodana do wybranego podzbioru, to nie zostanie z niego nigdy usunięta, nawet jeśli po dodaniu innej cechy stanie się zbędna lub co gorsze jej obecność będzie powodowała zmniejszenie wartości funkcji kryterialnej. Tego efektu pozbawiona jest metoda SBS. Jednak w tym wypadku cecha uznana za zbędną nigdy nie będzie ponownie sprawdzona, mimo, że po usunięciu jednej z kolejnych cech, może stać się już pożądana.

Mimo, że złożoność obliczeniowa obu metod jest taka sama, to jednak w praktyce metoda SFS jest dużo szybsza. Dotyczy to zwłaszcza problemów o dużej wymiarowości wektora cech. Wynika to z faktu, że największym kosztem obliczeniowym jest uruchomienie klasyfikatora. Koszt ten zaś często zależy od ilości cech. Tymczasem o ile w metodzie SFS zaczynamy od jednoelementowego zbioru cech, to metoda SBS zaczyna od pełnego zbioru cech.

1.3.3. Metody SFFS i SFBS

Opisanych powyżej wad pozbawione są metody pływającego przeszukiwania w przód SFFS (ang. sequential floating forward selection) i pływającego przeszukiwania w tył SFBS (ang. sequential floating backward selection). W przypadku obu tych metod po etapie przeszukiwania w przód/w tył następuje etap przeszukiwania w odwrotnym kierunku. Pozwala to w przypadku algorytmu SFS usunąć cechę, która stanie się po dodaniu innych zbędna, zaś w przypadku algorytmu SBS, pozwala na rozpatrywanie danej cechy ponownie, mimo, że we wcześniejszym kroku algorytmu została ona usunięta. Poniżej przedstawiony jest zarys algorytmu SFFS:

1. $S \leftarrow \emptyset$
2. $J_{max} \leftarrow 0$, $m \leftarrow$ liczba wszystkich cech, $n \leftarrow 0$
3. // przeszukiwanie w przód
4. Dla wszystkich cech $f_i \notin S$, policz $J(S \cup \{f_i\})$
5. $J_{max} \leftarrow max_{i=1,\ldots,m}\{J(S \cup \{f_i\})\}$, $f_{max} = argmax_{i=1,\ldots,m}\{J(S \cup \{f_i\})\}$
6. Jeśli $J \geq J_{max}$ zakończ działanie algorytmu
7. W przeciwnym przypadku $S \leftarrow S \cup \{f_{max}\}$, $m \leftarrow m - 1$, $n \leftarrow n + 1$
8. // przeszukiwanie w tył
9. Dla wszystkich cech $f_i \in S$, policz $J(S \cup \{f_i\})$
10. $J_{max} \leftarrow max_{i=1,\ldots,n}\{J(S \cup \{f_i\})\}$, $f_{max} = argmax_{i=1,\ldots,n}\{J(S \cup \{f_i\})\}$
11. Jeśli ($J \geq J_{max}$) i (m > 0) przejdź do 3.

12. Jeśli $m > 0$ zakończ działanie algorytmu
13. W przeciwnym przypadku $S \leftarrow S \setminus \{f_{max}\}$, $m \leftarrow m + 1$, $n \leftarrow n - 1$
14. Przejdź do 3.

W przypadku tego algorytmu po każdorazowym dodaniu cechy do szukanego podzbioru, sprawdzamy czy któraś z już wybranych do tej pory cech nie pogarsza nam wyniku klasyfikacji. Takie postępowanie prowadzi zwykle do lepszego podzbioru cech. Jednak tracimy w tym wypadku jedną w ważnych zalet algorytmu SFS. Jak łatwo zauważyć nie mamy gwarancji, że algorytm SFFS w każdym kroku doda co najmniej jedną cechę do szukanego podzbioru.

Oznacza to niestety również, że tracimy pesymistyczne oszacowanie algorytmu selekcji jako $O(m^2)$. Teoretycznie algorytm SFFS może wymagać nawet wykładniczej ilości uruchomień klasyfikatora. W praktyce jednak często okazuje się, że liczba koniecznych operacji nie jest tak duża, co wynika z faktu, że usunięcie cechy z już wybranego podzbioru jest wykonywane tylko wtedy, gdy oznacza ono poprawienie wyniku klasyfikatora.

W przypadku algorytmu SFFS ważną kwestią jest zapewnienie warunku stopu algorytmu. Jest to zapewnione przez słabą nierówność w 6 i 11 kroku algorytmu. Powoduje ona, że po każdorazowym dodaniu lub usunięciu cechy wynik klasyfikacji zwiększa się. Gwarantuje to, że nigdy nie będziemy rozpatrywali takiego samego zbioru cech dwa razy.

1.3.4. Metoda BDS

Dość ciekawą metodą, która pozwala uniknąć wad metod SFS i SBS, a jednocześnie gwarantuje nam oszacowanie złożoności na poziomie $O(m^2)$ jest przeszukiwanie dwustronne BDS (ang. BiDirectional Search). Metoda ta polega na równoległym zastosowaniu algorytmów SFS i SBS. Szukane rozwiązanie jest wspólnym rozwiązaniem dla obu algorytmów. Aby zapewnić, że rozwiązanie będzie zbieżne algorytm SFS sprawdza dodatkowy warunek. Czy cecha, którą ma zamiar dodać nie została wyeliminowana już przez algorytm SBS. Algorytm SBS z kolei nigdy nie usuwa cechy, która została dodana w dotychczasowych krokach przez algorytm SFS

W przypadku równoległego działania obu algorytmów mogłoby się zdarzyć, że algorytm SFS i SBS próbowałyby odpowiednio dodać/usunąć tę samą cechę. W takim przypadku zakładamy, że któryś z tych dwóch algorytmów ma pierwszeństwo w działaniu. Oczywiście wszystkie cechy dodane czy usunięte ze zbioru cech nie są już brane pod uwagę w kolejnych krokach obu algorytmów.

1. $S_1 \leftarrow \emptyset$, $T_1 \leftarrow$ wszystkie cechy
2. $S_2 \leftarrow$ wszystkie cechy, $T_2 \leftarrow \emptyset$
3. $f \leftarrow SFS(T_1)$
4. $S_1 \leftarrow S_1 \cup \{f\}$, $T_2 \setminus \{f\}$

5. $f \leftarrow SBS(T_2)$
6. $S_2 \leftarrow S_2 \setminus \{f\}, T_1 \setminus \{f\}$
7. Jeśli $S_1 = S_2$ zakończ działanie algorytmu
8. Przejdź do 3.

Metoda ta zapewnia nam górne oszacowanie ilości kroków algorytmu, ponieważ aby dojść do wspólnego podzbioru cech musimy wykonać $O(m^2)$ klasyfikacji. Każdy z algorytmów wykonuje ich $m + (m-1) + \ldots (m-n)$, gdzie m jest równe ilości cech, a n równe jest ilości cech w znalezionym podzbiorze.

1.3.5. Metody wbudowane

Metody wbudowane (ang. embedded methods) korzystają z wewnętrznych reprezentacji wybranych klasyfikatorów, które w procesie uczenia dokonują pośrednio oceny przydatności cech, ich ważenia bądź wręcz selekcji. W literaturze znaleźć można przykłady algorytmów opartych między innymi na dyskryminacji liniowej, sieciach neuronowych, SVM, oparte na drzewach decyzji [20], [62], [103], które podczas procesu uczenia również dokonują wyboru przydatnych cech.

Dużą zaletą tych metod jest ich szybkość wynikająca z faktu, że metoda selekcji jest wbudowana w proces uczenia się klasyfikatora. Metody te zwykle dają również dużo lepsze wyniki niż metody filtrów, co jest związane z faktem, że są one projektowane pod kątem konkretnego algorytmu klasyfikacji. Z kolei dzięki temu, że proces selekcji nie wymaga wielokrotnego wywoływania klasyfikatora dla każdego sprawdzanego podzbioru cech, metody te są wielokrotnie szybsze od metod obudowanych.

Metody wbudowane, jak sama nazwa wskazuje, są związane z konkretnym klasyfikatorem. Są wbudowane w proces uczenia się klasyfikatora, co powoduje, że wybrany podzbiór cech nie nadaje się do zastosowania z innym typem klasyfikatora.

1.3.6. Metody oparte na modyfikacji funkcji kryterialnych

Grupą bardzo ciekawych i współczesnie intensywnie rozwijających się metod są metody oparte na modyfikacji funkcji kryterialnych poprzez dodanie do nich członów z kosztami poszczególnych cech. Jedną z takich metod jest metoda LASSO (ang. Least Absolute Shrinkage and Selection Operator) zaproponowana przez Tibshiraniego w pracy [151]. Inną tego typu metodą jest metoda RLS (ang. Relaxed Linear Separability) zaproponowana w [10]. Metody te sprawdzają się szczególnie w sytuacji, gdy liczba cech jest dużo większa od liczby próbek w zbiorze uczącym.

W przypadku metody LASSO minimalizujemy funkcję straty L daną następującym wzorem:

$$L = \sum_i (y_i - \sum_p \beta_p x_{ip})^2 + \lambda \sum_p |\beta_p| \tag{1.62}$$

gdzie x_{ip} oznacza p-ty element i-tego wektora cech, y_i wartość odpowiedzi na ten wektor, β_p oznacza współczynnik regresji p-tej cechy, a λ jest współczynnikiem kary.

Dzięki zastosowaniu członu z kosztami cech $\lambda \sum_p |\beta_p|$ metoda ta prowadzi do rozwiązania, w którym większość współczynników β_p przyjmuje wartości zerowe. Oznacza to, że współczynniki najmniej istotnych (ang. irrelevant) lub redundantnych cech stają się zerowe. Analiza przeprowadzona w pracy [112] wskazuje, że metoda LASSO jest szczególnie efektywna gdy mamy bardzo dużo nieistotnych cech i niewiele próbek w zbiorze uczącym.

Jednym z ograniczeń tej metody jest założenie liniowości przestrzeni cech. Stąd jej wyniki mogą być gorsze w przypadku, gdy mamy do czynienia z nieliniowymi zależnościami. Próbę rozwiązania tego problemu podjęto w pracy [131]. Zaproponowana metoda GLR (ang. generalized Lasso regressions) wprowadza w miejsce x_{ip} funkcję jądrową $K(x_i, x_p)$.

Metoda RLS oparta jest na minimalizacji funkcji kar, które spełniają dodatkowe założenia. Wymagamy od nich aby były wypukłe i odcinkowo liniowe (ang. convex and piecewise linear) CPL. Pozwala to zastosować bardzo efektywny algorytm wymiany rozwiązań bazowych w celu minimalizacji funkcji kary. Rozważmy dwa rozłączne podzbiory uczące G^+ i G^- zawierające odpowiednio wektory cech dla dwóch klas. Zdefiniujmy funkcję kryterialną w postaci:

$$\Phi(v) = \sum_{j \in J^+} \alpha_j \varphi_j^+(v) + \sum_{j \in J^-} \alpha_j \varphi_j^-(v) \tag{1.63}$$

gdzie $\alpha_j > 0$ są kosztami poszczególnych wektorów wag v, J^- i J^+ sa rozłącznymi zbiorami indeksów wektorów cech należących odpowiednio do zbiorów uczących G^+ i G^-, a $\varphi_j^+(v)$ i $\varphi_j^-(v)$ są wypukłymi i odcinkowo liniowo funkcjami kary. Definicje wektorów wag v jak i funkcji kar $\varphi_j^+(v)$, $\varphi_j^-(v)$ można znaleźć w [10].

Oznaczmy przez $F[k]$ przestrzeń cech opartą o k-wymiarowe wektory cech, a przez $F[k']$ przestrzeń cech opartą o k'-wymiarowe wektory cech. Można dowieść, że fukcja kryterialna $\Phi(v)$ ma własność monotoniczności [9]:

$$F_{i'}[k'] \subset F_i[k] \Rightarrow (\Phi_{i'}^* \geq \Phi_i^*) \tag{1.64}$$

gdzie Φ_i^* jest minimalną wartością funckji kryterialnej zdefiniowanej w 1.63 na przestrzeni cech $F_i[k]$.

Nazwijmy zatem tę minimalną wartość Φ_i^* miarą liniowej nieseparowalności zbiorów uczących $G^+[k]$ i $G^-[k]$ w przestrzeni cech $F[k]$. Zgodnie ze wzorem 1.22 jeśli odrzucimy pewne cechy z przestrzeni cech $F[n]$ miara nieseparowalności nie może się zmniejszyć. Zatem odrzucając odpowiednio dużą ilość cech spowodujemy zwiększenie wartości Φ_i^*. To oznacza, że algorytm selekcji cech RLS możemy sformułować następująco [9]:

Odrzuć maksymalną liczbę cech z przestrzeni $F(n)$ pod warunkiem, że nie spowoduje to wzrostu miary liniowej niesparowalności Φ_i^* większej niż pewna z góry określona wartość $\gamma_0 > 0$.

Dodatkową poprawę jakości selekcji cech można uzyskać dodając do funkcji $\Phi(v)$ człon $\Phi_0(v)$. Jest to człon uwzględniający koszty poszczególnych cech. Funkcja zerowa $\Phi_0(v)$ przyjmuje postać:

$$\Phi_0(v) = \lambda \sum_i \gamma_i |v_i| \tag{1.65}$$

gdzie $\lambda > 0$ jest ogólnym współczynnikiem kosztów zaś $\gamma_i > 0$ są współczynnikami kosztów poszczególnych cech.

1.4. Specyfika problemu wieloklasowego

Najprostszym problemem klasyfikacji jest problem binarny. W takim wypadku zadaniem klasyfikatora jest podjęcie decyzji typu: tak – nie, pacjent chory – pacjent zdrowy, kolor ciemny – kolor jasny, trend wzrostowy – trend spadkowy. W konkretnych problemach, przed którymi staje nauka i technika, mamy jednak często do czynienia z sytuacją, gdy musimy przydzielić klasyfikowaną próbkę do jednej z wielu klas.

Jest oczywiste, że ilość klas wpływa na złożoność problemu klasyfikacji. Konieczna jest większa liczba próbek w zbiorze uczącym, większa liczba cech. Zwykle pojawia się też więcej parametrów do estymacji. Rośnie również złożoność obliczeniowa algorytmu, która jest zwykle zależna od ilości klas, a często również od ilości cech.. Kolejnym problemem, z którym musimy się zmierzyć jest także to, że niektóre klasyfikatory zostały zdefiniowane jako klasyfikatory binarne – na przykład maszyna wektorów wspierających SVM.

Klasyfikator ten możemy zmodyfikować, tak aby rozpatrywać wszystkie klasy jako jeden problem optymalizacyjny. Takie podejście zostało opisane przez Crammera i Singera [33] oraz Lee [94]. Jednak w takim przypadku gwałtownie rośnie nam złożoność obliczeniowa algorytmu. O ile w przypadku binarnego klasyfikatora SVM mamy do czynienia z optymalizacją kwadratową k zmiennych, to w przypadku wieloklasowym mamy już optymalizację kwadratową $(n-1)k$ zmiennych, gdzie n jest liczbą klas. Bardziej efektywną

wersję takiego podejścia zaproponowali Wang i Shen [159]. Opracowana przez nich metoda L_1–norm MSVM (Multi category SVM) jest dużo bardziej efektywna. Jednak przy dużej ilości klas to podejście jest nadal bardzo kosztowne obliczeniowo.

Możemy jednak podejść do problemu wieloklasowego nieco inaczej próbując ”rozłożyć” go na problemy binarne. W literaturze opisana jest cała gama tego typu metod: metoda *jeden przeciw pozostałym* OVR (ang. One Versus Rest), *jeden przeciw jednemu* OVO (ang. One Versus One), skierowanych grafów acyklicznych DAG (ang. Directed Acyclic Graph), adaptacyjnych skierowanych grafów acyklicznych ADAG (ang. Adaptive DAG) [118], [80], binarnych drzew decyzyjnych BDT (ang. Binary Decision Tree) [52], metody DB2 (ang. Divide By 2) [158], metoda zestawiania par (ang. pairwise coupling) [65] czy użycie wyjściowych kodów samokorygujących ECOC (ang. Error Correcting Output Codes) [38].

1.4.1. Strategia jeden przeciw pozostałym

Strategia ta nazywana jest czasami w literaturze *”jeden przeciw wszystkim”* OVA (ang. One Versus All). Jednak wydaje się, że właściwszą nazwą, która lepiej oddaje ideę działania tej metody, jest nazwa *”jeden przeciw pozostałym”* OVR. Dlatego ta ostatnia nazwa będzie konsekwentnie używana w tej pracy. Metoda ta, choć bardzo prosta, jest bardzo efektywna. Była stosowana z sukcesem w wielu eksperymentach [44], [17].

Załóżmy, że mamy do czynienia z problemem, w którym mamy $N > 2$ klas. Możemy zatem podzielić ten problem na N podproblemów binarnych, w których rozpatrujemy zadania separacji każdej z N klas od $N - 1$ pozostałych. Otrzymujemy w ten sposób N klasyfikatorów binarnych. Każdy z nich będzie przydzielał próbki do ”swojej” (one) klasy albo do ”pozostałych” (rest).

Następnie każda próbka jest testowana za pomocą każdego z klasyfikatorów. W idealnym przypadku tylko jeden z nich przydzieli próbkę ”swojej” klasy. Niestety w praktyce rzadko spotykamy się z takim rezultatem. Zwykle próbki przydzielane są do więcej niż jednej klasy, a zdarzają się również takie próbki, których żaden z klasyfikatorów nie przydzieli do ”swojej” klasy.

W opisanych powyżej przypadkach możemy zastosować kilka strategii, aby rozstrzygnąć problem przynależności do klasy. Najprostszym sposobem jest niepodejmowanie żadnej decyzji (odrzucanie). Takie podejście może pozwolić na uzyskanie bardzo dobrych wyników. Na przykład w przypadku przewidywania trzeciorzędowej struktury białek pozwoliło to na uzyskanie współczynnika klasyfikacji na poziomie 84,1% [28] (przy wynikach znanych z literatury na poziomie 56% – 62%).

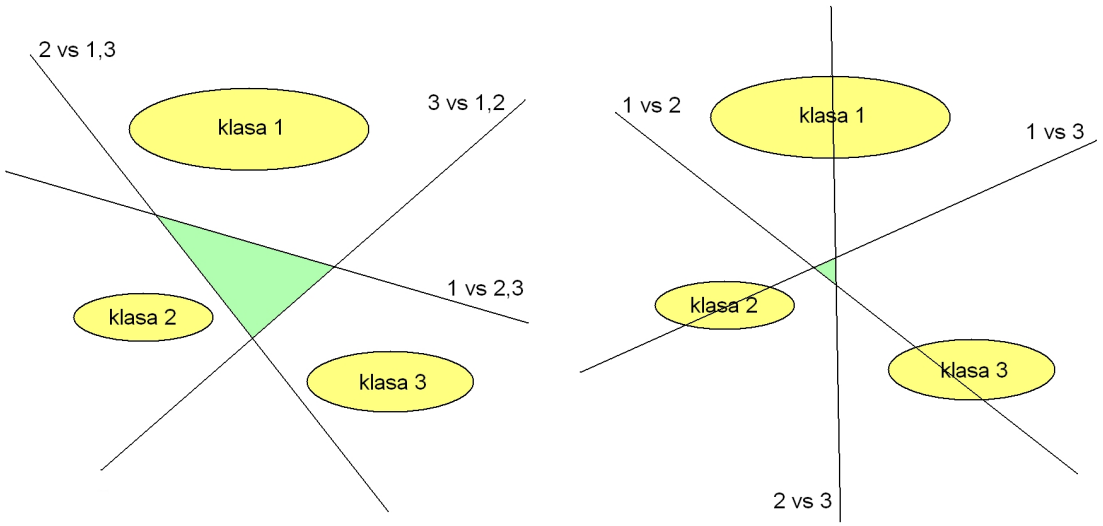
Jednak ceną za tak dobry wynik był bardzo wysoki współczynnik odrzucania (ang. rejection rate) $R_r = 59,2\%$. Oznacza to, że w przypadku ponad połowy próbek klasyfikator nie potrafił

podjąć decyzji. Współczynnik klasyfikacji liczony dla całego zbioru wyniósł w takim wypadku tylko $Q = 34,3\%$.

1.4.2. Strategia jeden przeciw jednemu

Nieco inne podejście prezentuje strategia *"jeden przeciw jednemu"* OVO. Jest ona znana także pod nazwą *"wszyscy przeciw wszystkim"* AVA (ang. All Versus All). W tym przypadku bierzemy wszystkie możliwe kombinacje par klas (l_i, l_j) dla $i, j = 1, 2, \ldots N$. W ten sposób konstruujemy $N * (N - 1)$ binarnych klasyfikatorów. Każdy z nich uczony jest wyłącznie na zbiorze próbek reprezentujących klasy etykietowane przez l_i i l_j. Następnie, tak jak w poprzednim przypadku, każdy taki klasyfikator decyduje o przydziale każdej próbki do pewnej klasy, głosując niejako w ten sposób na klasę l_i albo l_j.

Oznacza to, że każda próbka otrzyma $N * (N - 1)$ głosów. W idealnym przypadku $N - 1$ głosów będzie wskazywać na poprawną klasę, zaś pozostałe głosy powinny rozłożyć się przypadkowo. Zwykle jednak rozkład jest bardziej równomierny, dlatego też za poprawną uznajemy tę klasę, która dostała maksymalną liczbę głosów. Moglibyśmy wprawdzie pozostać przy założeniu, że poprawna klasa musi otrzymać $N - 1$ głosów (jeśli takiej nie ma odrzucamy próbkę). Jednak takie podejście pogarsza nam współczynnik klasyfikacji, choć ilość próbek zaklasyfikowanych niepoprawnie maleje.



Rysunek 1.10. Różnica pomiędzy metodami *jeden przeciw pozostałym* i *jeden przeciw jednemu*

W przypadku zastosowania strategii OVO musimy skonstruować $N * (N - 1)/2$ klasyfikatorów binarnych. Wpływa to znacząco na koszt obliczeniowy algorytmu. O ile w przypadku 10 klas (rozpoznawanie cyfr) daje nam to 45 klasyfikatorów, to już w przypadku 50 klas (rozpoznawanie autorów) jest to 1225. Jednak na przykład szacuje się, że w przypadku białek występuje około 1000 różnych struktur trzeciorzędowych (klas) [29], a to już oznacza

499 500 klasyfikatorów binarnych. To zaś oznacza, że użycie tej strategii byłoby całkowicie nieefektywne, o ile w ogóle możliwe.

Na rysunku 1.10 przedstawiona została różnica w podejściach *jeden przeciw pozostałym* (po lewej stronie) i *"jeden przeciw jednemu"* (po prawej). Jak łatwo zauważyć trójkątny obszar na środku każdego z rysunków (zaznaczony kolorem zielonym) wskazuje tak zwany "obszar niepewności". Próbki, które trafiają do tego obszaru nie zostaną w ogóle sklasyfikowane (w metodzie *"jeden przeciw pozostałym"* lub zostaną błędnie sklasyfikowane (w metodzie *"jeden przeciw jednemu"*).

Obszar ten jest zwykle dużo mniejszy w przypadku metody *"jeden przeciw jednemu"*. Wyniki eksperymentów autora [27] jak i wyniki opisane w innych publikacjach [2], [71], [58] wskazywałyby, że metoda ta pozwala osiągać lepsze wyniki klasyfikacji. Jednak w przeglądowej pracy [128] Rifkin i Klautau przekonują, że w ogólnym przypadku nie można mówić o wyższości jednej metodologii nad drugą.

Na korzyść metody OVR przemawia jej mniejsza złożoność obliczeniowa $O(n)$, która w przypadku bardzo dużej ilości klas stanowi argument przemawiający za jej stosowaniem. Jednak strategia OVO zapewnia dużo lepsze zbalansowanie reprezentacji przeciwstawnych klas. W przypadku klas reprezentowanych przez tę samą liczbę próbek mamy równą reprezentację klas na etapie uczenia klasyfikatora. Zastosowanie strategii OVR prowadzi zawsze do nadreprezentacji jednej z klas. Na przykład w przypadku 100 klas i równej reprezentacji wszystkich klas pojawi się sytuacja, w której przeciwstawiamy reprezentację jednej klasy reprezentacji 99 razy większej. To powoduje spolaryzowanie klasyfikatora i wpłynie na osiągane przez niego wyniki. Jak łatwo zauważyć im większa liczba klas tym problem będzie się pogłębiał.

1.4.3. Strategia binarnych drzew decyzyjnych

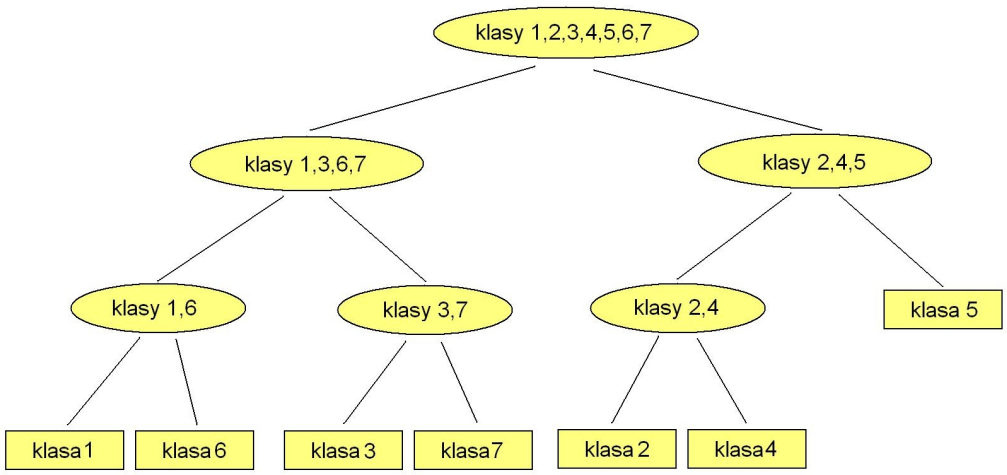
Strategia binarnych drzew decyzyjnych BDT (ang. Binary Decision Trees) rozwiązuje problem dużej ilości klasyfikatorów binarnych. Ogranicza ona ich liczbę poprzez stworzenie pewnej struktury drzewiastej. W strukturze tej, w każdym węźle drzewa klasyfikator binarny przydziela próbkę do jednego z dwóch podzbiorów klas, zaś klasyfikatory znajdujące się w liściach drzewa przydzielają próbki do podzbiorów jednoelementowych czyli konkretnych klas.

Tworzenie takiego drzewa rozpoczynamy od znalezienia podziału zbioru wszystkich klas na dwa podzbiory zawierające taką samą lub prawie taką samą liczbę klas. To znaczy podzbiory liczące sobie $\lfloor N/2 \rfloor$ i $\lceil N/2 \rceil$ elementów.

Następnie wszystkie próbki reprezentujące klasy należące do pierwszego podzbioru otrzymują etykietę "1", a pozostałe próbki etykietę "2". W ten sposób otrzymujemy problem binarny. Teraz uczymy klasyfikator za pomocą tak zmodyfikowanego zbioru uczącego

i w kolejnym kroku dzielimy testowane próbki na dwa zbiory reprezentujące nowe metaklasy ”1” i ”2”.

Pozostaje teraz zastosować rekurencyjnie powyższą strategię, aby otrzymać podział na pojedyncze klasy, w klasyfikatorach, które znajdą się w liściach tak stworzonego drzewa. Przykładowa struktura binarnego drzewa klasyfikatorów została przedstawiona na rysunku 1.11.



Rysunek 1.11. Przykładowe binarne drzewo decyzyjne klasyfikatorów

W przypadku użycia tej strategii konieczne jest skonstruowanie i wytrenowanie $N - 1$ klasyfikatorów. Jednak do sklasyfikowania konkretnej próbki wystarczy tylko $\lceil log_2 N \rceil$ z nich. Porównując tę strategię ze strategią OVO otrzymujemy (przy założeniu, że rozpoznajemy 1000 klas) odpowiednio: 999 i 499 500 klasyfikatorów binarnych niezbędnych na etapie uczenia oraz 10 i 499 500 klasyfikatorów binarnych na etapie rozpoznawania próbki. Jak łatwo można przewidzieć przyrost wydajności jest ogromny.

Główny problem, który pojawia się w przypadku binarnych drzew decyzyjnych to sposób podziału klas na podzbiory. W rozdziale 3 zostanie pokazane, że wybór takiego podziału istotnie wpływa na wynik klasyfikatora. W literaturze znajdziemy opisanych wiele różnych metod dzielenia klas na podzbiory. Na przykład Madzarow [102] proponuje pewien rodzaj miary podobieństwa opartej na środkach ciężkości klas. W innej pracy [77] proponuje się podział losowy, ale z użyciem strategii korygowania otrzymanych podzbiorów próbek po każdej klasyfikacji cząstkowej.

Podobną strategię opisuje metoda *”dziel na dwa”* DB2 (ang. Divide-By-2) [158]. Polega ona na podziale zbioru klas na dwa podzbiory, które następnie znów dzielimy na dwa aż otrzymamy podzbiory zawierające jeden element. Każdy podział na te dwa podzbiory traktowany jest jako problem klasteryzacji. Autorzy proponują zastosowanie jednej z trzech

metod klasteryzacji. Pierwszą z nich jest k-klasteryzacja, drugą użycie średnich odległości od środka układu współrzędnych w przestrzeni cech, a trzecią użycie zrównoważonych podzbiorów (tzn. zbiorów, w których różnice pomiędzy próbkami są minimalne).

1.4.4. Technika ECOC

Bardzo ciekawą techniką rozwiązywania problemu wieloklasowego jest użycie kodów samokorygujących ECC (ang. Error Correcting Codes). Kody te są powszechnie stosowanie w telekomunikacji. Za ich pomocą możliwe jest wykrywanie i korekcja zakłóceń, zniekształceń i przekłamań sygnału podczas transmisji. Ta technika, po odpowiedniej modyfikacji, może zostać zastosowana do rozwiązywania problemu wieloklasowego. Pierwszy raz wyjściowe kody samokorygujące ECOC (ang. Error Correcting Output Codes) zostały zastosowane przez Diettriecha i Bakiriego [38]. W kolejnych pracach pokazano, że technika ta poprawia zdolności generalizacyjne klasyfikatora [2], [161].

W dalszej części tego podrozdziału opisana zostanie idea kodów ECC i ECOC. Przedstawiona zostanie metoda pozwalająca na zastosowanie ich do problemu klasyfikacji wieloklasowej. Następnie opisane zostaną metody tworzenia optymalnych i suboptymalnych kodów ECOC. Na koniec wyjaśnione zostanie, dlaczego tworzone w ten sposób kody są nieodpowiednie z punktu widzenia zadania klasyfikacji.

Kody ECC

Podczas transmisji sygnału pomiędzy nadajnikiem, a odbiornikiem może dojść do jego zniekształcenia lub zakłócenia. Powoduje to, że w przesyłanym ciągu zer i jedynek (zakładamy, że przesyłamy sygnał binarny) dojdzie do przekłamań. Ważne jest zatem, aby odbiornik potrafił takie przekłamania wykrywać, a jeszcze lepiej byłoby gdyby potrafił je korygować.

Pierwszy postulat jest łatwiejszy do spełnienia. Jeśli podzielimy transmitowany ciąg bitów na słowa kodowe o ściśle określonej długości, to możemy do każdego z nich dodać tak zwany bit parzystości. Bit ten będzie przyjmował wartość 1 jeśli liczba jedynek w przesyłanej wiadomości jest nieparzysta i 0, gdy liczba jedynek w przesyłanej wiadomości jest parzysta. Taki kod nazywamy kodem detekcyjnym EDC (ang. Error Detecting Code).

Pozwala on odbiornikowi wykrywać błędy poprzez przeprowadzenie kontroli parzystości PC (ang. Parity Checking). Jak łatwo zauważyć, jeśli stosujemy powyższą technikę, to każde słowo kodowe powinno zawierać parzystą liczbę jedynek. Jeśli w odebranym słowie odbiornik wykryje ich nieprawidłową liczbę, to sygnalizuje błąd i może na przykład zażądać ponownego przesłania słowa.

Takie rozwiązanie ma jednak trzy zasadnicze wady. Po pierwsze nie zawsze możliwe jest ponowne przesłanie słowa kodowego. Po drugie powtórne przesłanie słowa może nie

skorygować błędu (zarysowana płyta CD, DVD). Po trzecie możemy w ten sposób wykryć błąd jedynie wtedy, gdy dojdzie do przekłamania tylko jednego bitu w słowie.

Najprostszym podejściem, które pozwala na stworzenie kodu, który odbiornik mógłby sam korygować, to ustalenie długości słowa na 3 i transmitowanie każdego bitu trzy razy. W ten sposób odbiornik odbierając kolejne słowa może nie tylko wykrywać błędy, ale również je naprawiać. Wystarczy sprawdzać, czy wszystkie odebrane bity słowa są takie same.

Oczywiście wciąż, gdy dojdzie do podwójnego przekłamania w słowie powoduje, że nie uda się poprawnie odtworzyć wiadomości. Dodatkowo taka metoda jest mało praktyczna, ponieważ powoduje znaczne ograniczenie pasma transmisyjnego. W praktyce stosuje się tak zwane kody Hamminga [105]. Jednak nadal kody te pozwalają naprawiać słowa pod warunkiem, że dojdzie tylko do jednego przekłamania bitu w słowie. Poniżej opiszemy inną metodę tworzenia kodów, która pozwoli na skorygowanie większej ilości przekłamań.

Załóżmy, że mamy pewną skończoną liczbę wiadomości N, które mogą być transmitowane. Każdą z tych wiadomości kodujemy za pomocą słowa kodowego o długości L składającego się z zer i jedynek. Zdefiniujmy również odległość między słowami kodowymi $w, z \in \{0,1\}^L$ jako:

$$d_H(w,z) = \sum_{i=1}^{L} (w_i \neq z_i) \tag{1.66}$$

gdzie L oznacza długość słowa kodowego, a w_i, z_i $i-$tą pozycję w słowie kodowym. Odległość tę będziemy nazywać odległością Hamminga (ang. Hamming distance). Zdefiniujmy również minimalną odległość Hamminga dla macierzy słów kodowych o wymiarach NxL w następujący sposób:

$$d_{HM}(M_{NL}) = min_{i \neq j} d_H(w_i, w_j) \tag{1.67}$$

gdzie M_{NL} macierz słów kodowych o wymiarach N wierszy i L kolumn, d_H odległość Hamminga pomiędzy wierszami $i-$tym i j-tym.

Załóżmy, że minimalna odległość Hamminga dla macierzy słów kodowych $d_{HM}(M_{NL}) \geq 3$, co oznacza, że dwa dowolne słowa kodowe reprezentujące różne wiadomości różnią się co najmniej na 3 bitach. Teraz, jeśli otrzymamy jakieś słowo kodowe, to przeszukujemy macierz NL w poszukiwaniu najbardziej podobnego wiersza (w sensie odległości Hamminga). Uznajemy, że słowo kodowe reprezentowane przez ten wiersz jest tym, które zostało wysłane przez nadajnik. Jeśli podczas transmisji doszło tylko do jednego błędu, to mamy pewność, że jest to poprawna decyzja.

Łatwo przy tym zauważyć, że im większa minimalna odległość Hamminga pomiędzy słowami, tym bardziej transmisja będzie odporna na zakłócenia. Dokładniej jeśli

$d_{HM}(M_{NL}) >= T$, to słowo kodowe zostanie poprawnie rozpoznane o ile popełnimy mniej niż $\lfloor (T-1)/2 \rfloor$ błędów.

Kody ECOC

Załóżmy teraz, że mamy N klas i że z każdą z nich łączymy słowo kodowe o długości L. Powstaje w ten sposób macierz kodów o wymiarach NxL. Jak pamiętamy z poprzedniego podrozdziału każde słowo kodowe składa się z zer i jedynek. Zatem każda kolumna macierzy definiuje nam pewien podział N klas na dwa rozłączne podzbiory etykietowane przez zera i jedynki. W sumie otrzymujemy L podziałów klas na dwa podzbiory.

Z każdym z takich podziałów możemy związać klasyfikator binarny, który jest następnie trenowany za pomocą próbek ze zbioru uczącego U, a dokładniej przez jego odpowiednik U', zdefiniowany przez podział związany z kolumną macierzy kodów. W ten sposób konstruujemy L klasyfikatorów binarnych C_i.

Kolejnym krokiem jest sklasyfikowanie każdej próbki ze zbioru testowego przez każdy z klasyfikatorów C_i. Każdy z nich przydziela próbce do klasy etykietowanej zero lub jedynkę, tworząc w ten sposób kod wyjściowy próbki. Następnie próbka jest klasyfikowana do odpowiedniej klasy na podstawie tego kodu. Próbka zostaje zaliczona do klasy, do której słowa kodowego jest najbliższa w sensie odległości Hamminga.

To podejście sugeruje, że podchodzimy do automatycznego rozpoznawania obiektów jak do pewnego problemu telekomunikacyjnego, w którym próbka jest kodowana za pomocą pewnego słowa kodowego. Następnie słowo to jest transmitowane poprzez niedoskonały system telekomunikacyjny. To kodowanie, a następnie "przesyłanie" bit po bicie (czyli w naszym przypadku poprzez kolejne klasyfikatory binarne) może być traktowane jako przesyłanie z błędami (przekłamaniami) transmisji. Cały system zaś jest w stanie prawidłowo rozpoznać przesyłaną próbkę pomimo błędów popełnianych przez poszczególne klasyfikatory binarne [38].

Kluczowym problemem przy zastosowaniu techniki ECOC jest to, w jaki sposób konstruować optymalne kody wyjściowe dla klas. To znaczy takie, które zapewnią jak najlepszą korekcję błędów. Należy zauważyć, że przydzielając klasom etykietę zero lub jeden, tworzymy w ten sposób kolumny macierzy kodów, definiujemy pewien podział klas na dwa podzbiory. To oznacza, że tworzymy w ten sposób pewien problem, który będzie rozwiązywany przez związany z nim klasyfikator binarny.

Oczywistym jest więc, że konstruując macierz kodów, musimy unikać kolumn podobnych lub identycznych, ponieważ związane z nimi klasyfikatory będą popełniały podobne lub takie same błędy. To oznacza, że takie kolumny nie wnoszą żadnej nowej informacji do rozwiązania naszego zadania klasyfikacji. Zatem, odległość Hamminga pomiędzy kolumnami powinna być maksymalizowana.

klasa	kolumny														
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
3	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1
4	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
5	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0

Tablica 1.4. Wyczerpujące kody ECOC dla pięciu klas

Łatwo również zauważyć, że kolumny będące swoimi dopełnieniami, to znaczy posiadające w tych samych wierszach zamienione zera i jedynki, definiują taki sam klasyfikator. Klasa etykietowana przez zero staje się klasą etykietowaną przez jeden i na odwrót. Czyli maksymalizując odległość pomiędzy kolumnami powinniśmy brać także pod uwagę ich dopełnienia. A zatem zdefiniujmy odległość Hamminga pomiędzy kolumnami jako:

$$d_{H'}(k,l) = min(d_H(k,l), d_H(k',l)) \tag{1.68}$$

gdzie $k, l \in \{0,1\}^N$ są kolumnami macierzy kodów M_{NL}, $l' \in \{0,1\}^N$ jest dopełnieniem kolumny l, zaś $d_H(k,l)$ jest odległością Hamminga.

A zatem będziemy starali się minimalizować następującą wielkość:

$$d_{HM'}(M_{NL}) = min_{i \neq j} d_{H'}(k_i, k_j) \tag{1.69}$$

gdzie M_{NL} macierz słów kodowych o wymiarach N wierszy i L kolumn, $d_{H'}$ odległość Hamminga pomiędzy kolumnami i-tą i j-tą.

Jednak nadal musimy pamiętać o maksymalizowaniu odległości pomiędzy wierszami macierzy kodów. Im jest ona większa tym więcej błędów jesteśmy w stanie korygować, a to oznacza lepszy stopień separowalności klas. W literaturze możemy spotkać cztery podstawowe techniki tworzenia kodów ECOC, w zależności od ilości klas w zadaniu klasyfikacji. Poniżej zostaną one pokrótce omówione:

— Kody wyczerpujące (ang. exhaustive codes) – Te kody możemy zastosować jeśli liczba klas nie jest zbyt duża, zwykle $3 \leq N \leq 7$. Konstruowanie kodów zaczynamy od wiersza zawierającego same jedynki, drugi wiersz będzie zawierał 2^{N-2} zer oraz $2^{N-2} - 1$ następujących po nich jedynek. Kolejny wiersz tworzymy z 2^{N-3} zer oraz $2^{N-3} - 1$ jedynek, i tak kolejno dla pozostałych wierszy. Budując kody w ten sposób otrzymamy maksymalne odległości pomiędzy wierszami i kolumnami. Przykładowy kod wyczerpujący dla pięciu klas jest przedstawiony w tabeli 1.4.

— Wybrane kolumny z kodów wyczerpujących (ang. column selection from exhaustive codes) – Jeśli liczba klas zawiera się w przedziale $8 \leq N \leq 11$. Konstruujemy kody wyczerpujące, a następnie wybieramy z nich odpowiedni podzbiór kolumn, tak aby w każdym momencie zachować maksymalną odległość pomiędzy wierszami.

— Technika losowego wpinania się RHC ang. Randomized Hill Climbing – W przypadku liczby klas $N > 11$ wygenerowanie wszystkich kodów pełnych jest niemożliwe ze względu na dużą liczbę możliwych permutacji. Musimy zatem zastosować pewną technikę heurystyczną opisaną poniżej.

  Zaczynamy od wylosowania N kodów o długości L. Następnie algorytm wyszukuje pary wierszy, które są sobie najbliższe (w sensie odległości Hamminga d_H) oraz parę kolumn, która jest najbardziej odległa w sensie tej odległości $d_{H'}$. Następnie algorytm znajduje cztery punkty przecięcia się tych wierszy i kolumn i zamienia je tak, aby poprawić separowalność wierszy i kolumn. Ten krok powtarzamy tak długo aż osiągniemy stan, w którym nie mamy już możliwości poprawy żadnej takiej pary kolumn i wierszy. Wtedy algorytm zaczyna wyszukiwać dowolne pary wierszy i kolumn i stara się poprawić ich separowalność.

  Szczegółowy opis algorytmu można znaleźć w pracy Diettricha [38]. Warto tutaj zauważyć, że w przypadku dużych N wykorzystanie tego algorytmu byłoby bardzo czasochłonne albo nawet niemożliwe ze względu na zbyt dużą ilość kombinacji do sprawdzenia.

— BCH codes – Dla problemów o ilości klas $N > 11$ możemy zastosować też pewne techniki algebraiczne (teorię algebry abstrakcyjnej Galois), aby zbudować niemal optymalne kody ECOC [13].

Wszystkie opisane tutaj metody mają jednak jedną zasadniczą wadę. Pozwalają na konstruowanie kodów optymalnych lub niemal optymalnych przy założeniu, że wszystkie klasyfikatory popełniają błędy, które rozkładają się losowo pomiędzy poszczególnymi słowami kodowymi. Takie podejście jest wskazane w zastosowaniach telekomunikacyjnych, gdzie rzeczywiście błędy podczas transmisji pojawiają się przypadkowo.

Z nieco inną sytuacją mamy do czynienia w przypadku zadań klasyfikacji. Konkretne klasyfikatory popełniają określoną liczbę błędów. Błędy te zależą od sposobu w jaki klasy zostały podzielone na dwa podzbiory. Ilość błędów zmienia się też w zależności od jakości klasyfikatora czy tez doboru jego parametrów. Pewne podziały klas są dla klasyfikatorów ”bardzo trudne” (osiągane współczynniki klasyfikacji są na przykład bliskie 50%), a inne stosunkowo łatwe (wynik klasyfikatora jest bardzo wysoki).

W podrozdziale 1.4.4 przedstawiona została analiza tego problemu. Omówiony został wpływ błędów popełnianych przez klasyfikatory binarne na wynik klasyfikatora końcowego. Zaproponowane zostały metody takiego tworzenia kodów ECOC, które powodują, że wykorzystywane są w nich klasyfikatory, które popełniają stosunkowo mało błędów (osiągające

wysokie współczynniki klasyfikacji). Dzięki temu, mimo, że generowane w ten sposób kody ECOC nie są optymalne, to wynik osiągany przez klasyfikator końcowy jest znacząco lepszy.

2. Opis problemów wybranych do testowania zaproponowanych algorytmów

Wybór baz danych, które mogłyby zostać użyte do przetestowania zaproponowanych w niniejszej pracy rozwiązań, nie był sprawą prostą. Z jednej strony ważne było, aby reprezentowały one konkretne problemy, z którymi boryka się współczesna nauka, technika czy medycyna. Z drugiej, ze względu na temat rozprawy, musiały to być zadania wieloklasowe, których próbki byłyby opisywane przez wektory cech o dużej wymiarowości. Jako minimum zostało przyjęte, że odpowiedni problem powinien dotyczyć klasyfikacji na co najmniej 10 klas, a wektory cech powinny liczyć minimum 100 elementów.

Pierwszym pomysłem było skorzystanie z bazy danych UCI Machine Learning Repository [154]. Baza ta zawiera 211 różnorodnych problemów (stan na styczeń 2012 r.) wraz z odpowiednimi zbiorami próbek. Jednak założenie, że ma to być zadanie klasyfikacji ograniczyło tę liczbę do 142, zaś założenie, że próbki powinny być opisywane przez co najmniej stuwymiarowe wektory cech, dodatkowo zmniejszyło wybór do zaledwie 19 baz danych. Z tej liczby 16 to problemy binarne. Z pozostałych trzech problemów zostały wybrane dwa, które zostały wykorzystane do badań: ISOLET oraz Amazon Commerce reviews set.

Pierwszy z nich to problem rozpoznawania mowy. Rozpoznawane obiekty, to litery angielskiego alfabetu wypowiadane przez 150 różnych osób. Każda z nich wypowiadała każdą z liter dwukrotnie. Dzięki temu otrzymano 52 próbki od każdej z osób. Dało to 300 próbek na każdą literę czyli w sumie 7800 próbek. W rzeczywistości w bazie danych brakuje trzech z nich (prawdopodobnie z powodu problemów z jakością nagrania). Baza danych nie zawiera oryginalnych nagrań, ale gotowe 617-wymiarowe wektory cech.

Amazon Commerce reviews set to problem rozpoznawania tożsamości człowieka na podstawie jego tekstów pisanych (recenzji). Wykorzystano teksty ze strony internetowej Amazon. Najpierw zidentyfikowano 50 najbardziej aktywnych użytkowników (każdy z nich napisał co najmniej 30 recenzji). Ich tożsamość ustalono na podstawie unikalnego identyfikatora ID oraz nazwy użytkownika (ang. username) w serwisie. Każdy z użytkowników był autorem co najmniej 30 tekstów, z których każdy liczył co najmniej 1000 słów. Następnie każdy z tych tekstów został opisany za pomocą wektora liczącego sobie 10000 cech. Również ta baza danych nie zawiera oryginalnych recenzji użytkowników, a jedynie identyfikator każdego z nich oraz wektory cech opisujące poszczególne próbki.

Kolejną bazę danych udało się znaleźć przeglądając publikacje dotyczące metod rozpoznawania obiektów. Założone kryteria spełniała baza danych odręcznie pisanych liter i cyfr opracowana przez amerykański instytut NIST (ang. National Institute of Standards and Technology). Baza ta zawiera próbki liter i cyfr od ponad 3600 różnych osób.

Niestety ten zbiór danych nie jest bezpłatnie dostępny, dlatego też do badań wykorzystana została baza danych MNIST (ang. Modified NIST) [149] zawierająca tylko odręcznie pisane cyfry. W bazie tej mamy do dyspozycji 70000 próbek reprezentujących 10 klas. Próbki te dostarczane są w postaci obrazków o 256 odcieniach szarości. Konieczne było zatem stworzenie opisujących je wektorów cech. Wektory takie (102-wymiarowe) zostały opracowane w oparciu o cechy gradientowe [22].

Czwarty zbiór, który został wykorzystany w eksperymentach, to efekt prac prowadzonych przez Instytut Informatyki Teoretycznej i Stosowanej PAN nad systemami opartymi na interakcji człowiek-komputer, a w szczególności nad rozpoznawaniem ludzkich gestów. Opracowana tam baza danych [60] doskonale nadaje się do testowania zaproponowanych w niniejszej pracy algorytmów. Zawiera ona 22 różne klasy. Każda próbka opisana jest przez serie pomiarów pewnych wartości dostarczanych przez urządzenia wychwytujące ruch (ang. motion capture units). Pomiary te musiały zostać przekształcone na odpowiednie wektory cech. Opis zastosowanej procedury znajduje się w dalszej części tego rozdziału. Jej efektem są 256-wymiarowe wektory cech.

Ostatnim problemem, wybranym do badań, było przewidywanie trzeciorzędowej struktury białek (ang. protein fold recognition). To jeden z ważniejszych problemów współczesnej bioinformatyki. Przewidywanie tej struktury jest bardzo istotne, ponieważ jest ona powiązana z biologicznymi funkcjami, które pełni dane białko [18]. Jest to bardzo ważne na przykład przy opracowywaniu nowych leków. Do testowania została wykorzystana baza danych opracowana przez Dinga i Dubchaka [40]. Zawiera ona prawie 700 próbek, które reprezentują 27 różnych klas. Każda z próbek jest opisana za pomocą 126-wymiarowego wektora cech.

Baza danych	Ilość klas	Ilość próbek	Wymiar wektora cech
baza MNIST	10	70 000	102
baza danych gestów	22	1 320	256
baza ISOLET	26	7 797	617
baza danych białek	27	698	126
baza ACRS	50	1 500	10 000

Tablica 2.1. Zestawienie wykorzystanych w pracy baz danych

W sumie do testowania zaproponowanych w niniejszej rozprawie algorytmów i strategii zostało wykorzystane pięć baz danych. Każda z nich reprezentuje pewne konkretne zadanie klasyfikacji wieloklasowej. Są to problemy całkowicie różne od siebie, reprezentujące odmienne dziedziny nauki i techniki. Zestawienie podstawowych parametrów opisujących te

bazy danych (liczbę klas, ilość próbek i wymiarowość wektora cech) prezentuje tabela 2.1. W dalszej części niniejszego rozdziału problemy te zostaną dokładniej omówione. Zostanie przedstawiona istota każdego z nich, jego znaczenie, opisane zostaną znane z literatury dotychczasowe metody jego rozwiązywania. Przedstawione zostaną uzyskane wyniki.

2.1. Rozpoznawanie mowy

Zastosowanie komputera do rozpoznawania mowy jeszcze kilkanaście lat temu było traktowane jako fantastyka naukowa. Dzisiaj sterowane głosem aplikacje w telefonach komórkowych nie budzą niczyjego zdziwienia. We współczesnych systemach możemy wyróżnić dwie główne funkcje rozpoznawania mowy. Pierwsza to zastosowanie komputera do transkrypcji, druga to zastosowanie komend głosowych do sterowania komputerem. W obu tych przypadkach szybkość działania algorytmu jest bardzo istotna, ponieważ bardzo często zależy nam, aby system taki działał w trybie online.

W przypadku rozpoznawania mowy możemy skupić się na rozpoznawaniu całych fraz czy komend, poszczególnych słów lub na rozpoznawaniu pojedynczych fonemów czy liter. Baza danych ISOLET (ang. ISOated LETters) reprezentuje to ostatnie podejście. Powstała ona poprzez nagranie 26 liter angielskiego alfabetu wypowiadanych przez 150 osób - anglojęzycznych Amerykanów (w tym 75 kobiet i 75 mężczyzn). Każda z liter została nagrana dwukrotnie. Otrzymano w ten sposób 7797 próbek - trzy próbki zostały odrzucone (zapewne z powodów słabej jakości nagrania).

Każda próbka została następnie zakodowana za pomocą 617-wymiarowego wektora cech. Wektor ten został opisany w pracach [31] i [51]. Został on oparty między innymi na współczynnikach spektralnych (ang. spectral coefficients), cechach konturowych (ang. contour features), cechach opartych o spółgłoski półotwarte (ang. sonorant features). Zostały one dobrane tak, aby umożliwić dobrą dyskryminację pomiędzy poszczególnymi literami. Dokładne uzasadnienie wyboru cech można znaleźć w pracy [51].

Metoda	wynik klasyfikacji
HMM [100]	97,4%
ANN [51]	96,0%
M^2SVM [75]	97,0%
MKSM [33]	97,3%

Tablica 2.2. Zestawienie wyników klasyfikacji uzyskanych na bazie danych ISOLET

W literaturze możemy znaleźć dość dużą liczbę publikacji wykorzystujących bazę danych ISOLET. Ich autorzy stosowali różne klasyfikatory do rozwiązywania tego problemu, uzyskując wyniki od 96,0% do 97,3% (pełne zestawienie wyników znajduje się w tabeli 2.2). Należy

oczywiście pamiętać, że podane wyniki nie są do końca porównywalne, ponieważ użyte w tych badaniach klasyfikatory były w różny sposób przez autorów testowane.

2.2. Identyfikacja autorstwa tekstu

Problem identyfikowania autorów anonimowych tekstów na podstawie statystycznej analizy charakterystyki stylu autora jest wykorzystywany przez historyków (głównie historyków literatury) czy biegłych sądowych. Początki tej dziedziny zwanej stylometrią możemy odnaleźć już w XV wieku, kiedy to włoski historyk i filozof Lorenzo Valla zakwestionował autentyczność dokumentu *Donacja Konstantyna* na podstawie analizy łaciny użytej w tym tekście w porównaniu do innych dokumentów z tej epoki.

Współcześnie metody stylometrii są wykorzystywane przez biegłych sądowych na przykład do potwierdzania autentyczności listów samobójczych czy anonimów. Na przykład w 1998 roku na podstawie takiej analizy został skazany Unabomber (Ted Kaczyński), który w latach 1978–1995 wysyłał do różnych osób listy-bomby.

Każdy tekst pisany przez człowieka nosi cechy charakterystyczne jego twórcy. W stylometrii analizuje się wiele takich cech, między innymi:

— częstotliwość użycia poszczególnych liter
— częstotliwość występowania cyfr
— częstotliwość występowania znaków interpunkcyjnych
— częstotliwość występowania słów funkcyjnych (np. do, z, na, po)
— częstotliwość występowania słów o określonej długości
— obecność słów zawierających różne kombinacje dużych i małych liter
— występowanie pewnych par słów
— częstotliwość występowania błędów (np. ortograficznych, literówek)

Ilość cech, które należy wziąć pod uwagę jest bardzo duża, dlatego też wykorzystanie technik automatycznego rozpoznawania obiektów i wykorzystanie komputerów pozwoliło na znaczny postęp w tej dziedzinie. Postęp tak znaczny, że niosący za sobą pewne zagrożenia dla prywatności i swobody wypowiedzi w internecie. Wykazali to autorzy w pracy [110] demonstrując klasyfikator pozwalający na identyfikację anonimowych blogerów. Pokazali oni, że nawet analizując problemy o olbrzymiej ilości klas (analizowali 100 000 potencjalnych autorów) można uzyskać stosunkowo duży współczynnik poprawnych odpowiedzi. Udało im się zidentyfikować około 20% autorów.

Problem, który został wykorzystany w niniejszych badaniach jest dużo prostszy. Wykorzystano teksty jedynie 50 autorów publikujących recenzje w serwisie Amazon. Każdy z autorów był reprezentowany przez 30 próbek tekstu. Próbki te zostały scharakteryzowane

przez 10 000 cech. Problem ten jest najtrudniejszym z wykorzystanych w eksperymentach ze względu na bardzo dużą ilość klas oraz olbrzymią liczbę cech.

Metoda	wynik klasyfikacji
SSN [98]	80,5%
SVM [98]	78,6%

Tablica 2.3. Zestawienie wyników klasyfikacji uzyskanych na bazie danych ACRS

Baza danych ACRS została udostępniona w UCI Machine Learning Repository dopiero pod koniec 2011 roku. W związku z tym trudno znaleźć wyniki różnych badań. Autorzy bazy danych zastosowali do tego problemu klasyfikator SVM oraz synergetyczną sieć neuronową SSN (ang. Synergetic Neural Network). Wyniki ich eksperymentów zostały umieszczone w tabeli 2.3.

2.3. Rozpoznawanie odręcznie pisanych cyfr

O praktycznym znaczeniu rozpoznawania pisma nie trzeba nikogo przekonywać. To bardzo szeroka i dynamicznie rozwijająca się gałąź dziedziny automatycznego rozpoznawania obiektów zwana optycznym rozpoznawaniem znaków OCR (ang. Optical Character Recognition). Można w niej wyróżnić dwa główne kierunki badań. Jeden z nich skupia się na rozpoznawaniu pisma drukowanego, drugi zaś zajmuje się problemem odręcznie pisanych znaków (ang. handwriting). W niniejszej pracy wykorzystana została baza danych odręcznie pisanych cyfr.

Z rozpoznawaniem pisma wiąże się cała gama metod związanych z akwizycją danych, ich wstępnym przetwarzaniem oraz doborem odpowiednich cech. Nie jest to sprawa prosta. Co więcej wybór odpowiedniej metody wstępnego przetwarzana, jak i dobór wektora cech ma duży wpływ na wynik klasyfikacji. Na przykład wpływ jednej z metod wstępnego przetwarzania (metody normalizacji) został pokazany w pracy [22]. W efekcie przeprowadzonych badań został opracowany wektor cech oparty o cechy gradientowe, który został wykorzystany w późniejszych eksperymentach.

Jedną z największych baz danych pisma odręcznego jest baza danych opracowana przez amerykański instytut NIST (National Institute of Standards and Technology). Baza ta, pod nazwą NIST Special Database 19 [113], zawiera ponad 800 tysięcy obrazów liter i cyfr pochodzących od 3600 różnych osób. Wszystkie te próbki pisma zostały przeskanowane w rozdzielczości 300 dpi do czarno-białych map bitowych.

Baza danych NIST Special Database 19 jest płatną bazą danych, dlatego też ostatecznie w badaniach została użyta baza danych MNIST (Modified NIST) [149]. Jest to baza danych zawierająca same cyfry. Została ona stworzona na podstawie baz NIST Special Database

3 i NIST Special Database 1 (zostały one później zastąpione jedną bazą danych NIST Special Database 19). Przykładowy zestaw próbek pochodzących z tej bazy danych jest przedstawiony na rysunku 2.1.

Rysunek 2.1. Przykładowe próbki z bazy danych MNIST

Oryginalne obrazy cyfr z bazy danych NIST, przed zaimportowaniem do nowej zmodyfikowanej bazy danych MNIST, zostały poddane wstępnemu przetwarzaniu. Czarno-białe mapy bitowe zostały znormalizowane, tak aby mieściły się w kwadracie o wielkości 20x20 pikseli. Następnie zostały umieszczone na obrazku o wielkości 28x28 pikseli, w taki sposób, aby ich środek ciężkości mieścił się na środku obrazka. Zmodyfikowane obrazy cyfr zawierają odcienie szarości, ponieważ algorytm normalizacji, który został użyty korzystał z techniki antyaliasingu.

Tak jak zostało wspomniane wcześniej, wadą tej bazy danych jest to, że zawiera wyłącznie próbki obiektów (cyfr), a nie gotowe już wektory cech. Konieczne było zatem ich wyodrębnienie z dostępnych obrazów cyfr. Zostało opracowane kilka różnych wektorów cech. Oparte o niezmienniki geometryczne [126], [164], momenty Zernika [147], [107], [152] oraz cechy gradientowe [97], [138]. Dokładny opis i sposób wyliczenia cech można znaleźć w [22]. W późniejszych testach został użyty wyłącznie 102-wymiarowy wektor cech oparty o cechy gradientowe, dla którego osiągane wyniki były najlepsze.

Metoda	wynik klasyfikacji
Dong et al. [41]	99,01%
Mayraz et al. [104]	99,30%
Belongie et al. [6]	99,37%
Teow et al. [148]	99,41%
Liu et al. [95]	99,41%

Tablica 2.4. Zestawienie wyników klasyfikacji uzyskanych na zbiorze cyfr MNIST

MNIST jest to jedna z najczęściej używanych baz danych do testowania algorytmów rozpoznawania znaków. Dzięki temu można łatwo porównać uzyskane rezultaty. Współczynnik klasyfikacji osiągany przez algorytmy opisane w literaturze jest bardzo wysoki i wynosi od 99,01% w [41] aż do 99,41% w [148] i [95]. Zestawienie kilku dodatkowych wyników prezentuje tabela 2.4.

2.4. Rozpoznawanie gestów

Jednym z elementów systemu opartego na interakcji człowiek-komputer HCI (ang. Human Computer Interaction) jest automatyczne rozpoznawanie gestów człowieka. Budowa systemu, który będzie to potrafił jest niewątpliwie dużym naukowym wyzwaniem. Jest to też jedno z zastosowań nauki, które może zostać łatwo i szybko zaadoptowane do przemysłu. Świadczy o tym najlepiej wysyp komercyjnych urządzeń rozpoznających ludzkie gesty. Wystarczy wspomnieć tylko akcelerometry montowane w nowoczesnych telefonach komórkowych, czy też bardziej zaawansowane urządzenia takie jak na przykład: Nintendo Wii Remote, Cyberglove Systems Cyberglove czy też Microsoft Kinect.

Wiele z tych urządzeń potrafi zapewnić doskonałej jakości pomiary wykonywanych przez człowieka ruchów. Jednak nadal interpretacja tych pomiarów i rozpoznanie na tej podstawie gestu człowieka nie jest sprawą prostą. Obecność szumu w danych, a także olbrzymia wymiarowość wektorów cech czyni problem bardzo trudnym do analizy. Co więcej gesty wykonywane przez różne osoby mogą się bardzo istotnie różnić od siebie, co czyni zadanie jeszcze trudniejszym. Niemniej jednak potencjalnie duża ilość klas i bardzo duża liczba cech, z którymi trzeba się zmierzyć czyni ten temat atrakcyjnym dla testowania algorytmów zaproponowanych w niniejszej pracy.

Jest kilka dostępnych baz danych gestów opisywanych w literaturze (na przykład w pracach [72], [139], [81]). Jednak wszystkie one zawierają zaledwie kilka, kilkanaście klas. Dodatkowo zawierają stosunkowo niedużą liczbę próbek. Szukając baz danych odpowiednich do naszych testów udało się natrafić na bazę danych opracowaną w Instytucie Informatyki Teoretycznej i Stosowanej PAN [129], która zawiera dużą liczbę różnych gestów (22 klasy). Ich opis można

znaleźć w tabeli 2.5. Pozostawiono w niej oryginalne nazwy gestów (i ich typów) w języku angielskim, aby uniknąć ewentualnych nieporozumień związanych tłumaczeniem.

etykieta	Nazwa gestu	typ gestu
1	A-OK	symbolic
2	Walking	iconic
3	Cutting	iconic
4	Shove away	iconic
5	Point at self	deictic
6	Thumbs up	symbolic
7	Crazy	symbolic
8	Knocking	iconic
9	Cutthroat	symbolic
10	Money	symbolic
11	Thumbs down	symbolic
12	Doubting	symbolic
13	Continue	iconic
14	Speaking	iconic
15	Hello	symbolic
16	Grasping	manipulaitve
17	Scaling	manipulaitve
18	Rotating	manipulative
19	Come here	symbolic
20	Telephone	symbolic
21	Go away	symbolic
22	Relocate	symbolic

Tablica 2.5. Zestawienie gestów w wykorzystanej bazie danych

Baza ta zawiera pomiary wykonane za pomocą trzech różnych systemów (DG5VHand glove, ShapeWrapIII suit and CyberGlove / CyberForce system). Każda sekwencja ruchów została pomierzona 10-krotnie (sześć pomiarów przy normalnej szybkości wykonywania gestu, dwa przy wolnej i dwa przy szybkiej). W eksperymentach uczestniczyło 7 różnych osób. Niestety jednak nie dla wszystkich pomiary wykonano na każdym z urządzeń. Po konsultacji z autorami bazy danych do eksperymentów zostały wybrane pomiary wykonane za pomocą urządzenia DG5VHand glove.

Same pomiary to kilka do kilkunastu parametrów mierzonych w pewnych odstępach czasu. Niestety to oznacza, że każda sekwencja ruchów opisywana jest przez różną liczbę pomiarów. Na przykład sekwencja wykonana szybciej będzie opisywana przez mniejszą liczbę pomiarów niż sekwencja wykonywana wolniej. Również gesty wykonywane przez różne osoby, nawet te wykonywane z tą samą szybkością, różnią się ilością pomiarów. Dlatego też wszystkie pomiary zostały najpierw znormalizowane, w ten sposób, aby ich liczba była taka sama dla każdego z gestów. Odpowiednie wartości pomiarów były interpolowane.

W ten sposób każdy z gestów był opisywany przez taką samą liczbę wierszy zawierających pewne wartości. Następnie wybrano pierwszy wiersz (odpowiadający pozycji początkowej ręki) i ostatni (odpowiadający pozycji ręki po zakończeniu wykonywania ruchu). W kolejnym kroku wybrano pozostałe wiersze tak, aby liczba cech wynosiła 256. Wiersze były wybierane w taki sposób, aby były możliwie równo odległe od siebie.

Metoda	wynik klasyfikacji
Random Forest	67,8%
kNN	74,9%
SVM	79,9%

Tablica 2.6. Zestawienie wyników klasyfikacji uzyskanych na zbiorze gestów (A)

Zadanie klasyfikacji gestów można rozpatrywać na dwa sposoby. W pierwszym z nich klasyfikator uczony jest na próbkach pochodzących od pięciu osób, a do testowania używane są próbki pochodzące od szóstej osoby. Odpowiada to sytuacji, w której urządzenie uczone jest rozpoznawania odpowiednich sekwencji ruchów na etapie produkcji, a następnie jest wypuszczane na rynek i jego zadaniem jest rozpoznawanie gestów osób, które je zakupiły. Ten typ zadania klasyfikacji będzie oznaczany literą (A).

Z drugą sytuacją mamy do czynienia, gdy stosujemy do tej bazy danych metodę warstwowej k-kroswalidacji. Wtedy klasyfikator uczony jest za pomocą próbek należących do wszystkich sześciu osób. Odpowiada to sytuacji, w której urządzenie po zakupie najpierw uczy się rozpoznawania gestów wszystkich osób, które będą go używały. Ten typ zadania klasyfikacji będzie oznaczany literą (B).

W tabeli 2.6 przedstawiono wyniki osiągnięte za pomocą kilku różnych klasyfikatorów na bazie danych gestów. Wszystkie te wyniki zostały osiagnięte przy zastosowaniu wariantu klasyfikacji oznaczonego literą (A).

2.5. Przewidywanie trzeciorzędowej struktury białek

Przewidywanie trzeciorzędowej struktury białek (ang. protein fold recognition) jest jednym z kluczowych problemów biologii molekularnej. Trzeciorzędowa struktura białka, to sposób w jaki łańcuch aminokwasów, które go tworzą, zwija się, tworząc pewną trójwymiarową strukturę. Dlatego też często trzeciorzędowa struktura białka jest nazywana strukturą trójwymiarową. Co jednak bardzo ważne, ta struktura rzutuje na wiele własności funkcjonalnych takiego białka [18]. Zatem jej znajomość ma kluczowe znaczenie, na przykład w badaniach nad opracowywaniem nowych leków.

Dlaczego jednak nie badamy struktury białka metodami eksperymentalnymi? Otóż, sekwencjonowanie łańcuchów aminokwasowych jest metodą stosunkowo szybką, tanią

i niezawodną. Za to eksperymentalne metody określania trzeciorzędowej struktury białka są bardzo drogie, żmudne i w praktyce nieraz dość zawodne. Są one oparte na krystalografii rentgenowskiej (ang. X-ray crystallography) i magnetycznym rezonansie jądrowym NMR (ang. Nuclear Magnetic Resonance).

Wystarczy wspomnieć, że w strukturalnej bazie danych białek UniProt (Universal Protein Resource) [155], zawierającej znane łańcuchy aminokwasowe białek, mamy około 6 milionów pozycji (stan na styczeń 2010 r.). Za to w bazie danych PDB (Protein Data Bank) [7] jest zarejestrowanych tylko około 58 500 białek, których trójwymiarową strukturę znamy (stan na lipiec 2009 r.) [120].

Skoro metody eksperymentalne w tym wypadku zawodzą, to nasuwa się pytanie czy nie możemy przewidzieć w jaki sposób zwinie się łańcuch aminokwasów na podstawie jego budowy oraz znajomości reguł fizycznych i biochemicznych. Jest to możliwe i w literaturze jest opisanych wiele metod bazujących na tej wiedzy. Jednak i one również są zawodne, a dodatkowo bardzo kosztowne obliczeniowo. Dobry przegląd tych metod można znaleźć w pracach [132], [59]. Okazuje się więc, że stosowanie metod statystycznego rozpoznawania obiektów jest w tej sytuacji uzasadnione i potrzebne.

Istnieje jeszcze jedna grupa metod pozwalająca na przewidywanie trzeciorzędowej struktury białek. Są to metody bazujące na podobieństwie łańcuchów aminokwasowych. Intuicyjnie, białka posiadające podobne struktury w swoim łańcuchu aminokwasowym będą tworzyły podobne struktury trójwymiarowe. W przypadku kiedy udaje się zidentyfikować podobieństwa w sekwencjach aminokwasowych stosuje się metody modelowania homologicznego (ang. homology modeling) i nawlekania (ang. threading). Dobry przegląd tych metod można znaleźć w pracy Baldiego i Brunaka [4].

Zakres zastosowania tych metod jest jednak bardzo ograniczony, ponieważ bardzo wiele białek o całkowicie różnych sekwencjach aminokwasowych tworzy takie same struktury trzeciorzędowe. W celu ułatwienia wyboru odpowiednich białek definiuje się procentową miarę podobieństwa łańcuchów aminokwasowych [140] oraz tworzy listy podobieństwa. Na przykład lista PDBselect [120] zawiera białka, których podobieństwo łańcuchów aminokwasowych jest mniejsze niż określona wartość procentowa.

Nawet z wykorzystaniem takiej listy stworzenie bazy danych do testów byłoby trudne. Dlatego też w badaniach została użyta gotowa baza danych opisana w [39], [40]. Są to dwa zbiory opracowane na podstawie bazy białek SCOP (Structural Classiffcation Of Proteins). Zbiór uczący składa się z 313 próbek, a zbiór testowy z 383 próbek. Oba te zbiory zawierają próbki reprezentujące 27 różnych struktur białkowych. Zbiór uczący został oparty na zbiorze PDBselect [69], [70], zaś zbiór testowy na zbiorze PDB-40D [99], z którego wybrano te same 27 klas reprezentowanych w zbiorze uczącym. Białka oraz klasy, które reprezentują zostały przedstawione w tabeli 2.7.

etykieta klasy	Nazwa struktury (fold)	ilość próbek w zbiorze uczącym	ilość próbek w zbiorze testowym
1	Globin-like	13	6
2	Cytochrome c	7	9
3	DNA-binding 3-helical bundle	12	20
4	4-helical up-and-down bundle	7	8
5	4-helical cytokines	9	9
6	Alpha; EF-hand	7	9
7	Immunoglobulin-line β-sandwich	30	44
8	Cupredoxins	9	12
9	Viral coat and capsid proteins	16	13
10	ConA-like lectins/glucanases	7	6
11	SH3-like barrel	8	8
12	OB-fold	13	19
13	Trefoil	8	4
14	Trypsin-like serine proteases	9	4
15	Lipocalins	9	7
16	(TIM)-barrel	29	48
17	FAD (also NAD)-binding motif	11	12
18	Flavodoxin-like	11	13
19	NAD(P)-binding Rossmann-fold	13	27
20	P-loop containing nucleotide	10	12
21	Thioredoxin-like	9	8
22	Ribonuclease H-like motif	10	14
23	Hydrolases	11	4
24	Periplasmic binding protein-like	11	4
25	β-grasp	7	8
26	Ferrodoxin-like	13	27
27	Small inhibitors, toxin, lectins	12	27

Tablica 2.7. Lista struktur białkowych w bazie danych

Powyższa baza danych zawiera gotowe wektory cech, opracowane przez Dinga i Dubchaka [43], [40]. Zostały one oparte na sześciu parametrach: sekwencji aminokwasowej C (amino acid Composition), przewidywanej strukturze drugorzędowej S (predicted Secondary structure) [130], hydrofobowości H (Hydrophobity), znormalizowanej objętości van der Waalsa V (normalized van der waals Volume), biegunowości P (Polarity) i polaryzowalności Z (polariZability).

Każdy z tych parametrów odpowiada za 21 wyliczanych cech oprócz parametru C, który opowiada za 20 cech. Zbiory danych zawierające wyliczone cechy można znaleźć w [39]. W badaniach ten wektor został minimalnie zmieniony poprzez dodanie do wektora C długości sekwencji aminokwasowej. Zatem pełen wektor cech (C, S, H, V, P, Z) liczył będzie 6 x 21 = 126 cech.

Metoda	wynik klasyfikacji
MLP [30]	48,8%
RBFN [116]	51,2%
SVM [40]	56,0%
HKNN [114]	57,4%
RS1_HKNN_K25 [108]	60,3%
DIMLP-B [12]	61,2%
Random forest [35]	62,7%

Tablica 2.8. Zestawienie wyników klasyfikacji uzyskanych na zbiorze białek

W literaturze możemy znaleźć wiele publikacji zajmujących się przewidywaniem trzeciorzędowej struktury białek za pomocą metod automatycznej klasyfikacji obiektów. Na przykład Ding i Dubchak [40] eksperymentowali z klasyfikatorem SVM oraz sieciami NN, Shen i Chou zaproponowali pewien model oparty o metodę najbliższego sąsiada [136], zmodyfikowany algorytm k-najbliższych sąsiadów K-local Hyperplane (HKNN) został przedstawiony w pracy Okuna [114]. Nanni [108] testował połączenie liniowego klasyfikatora Fishera z klasyfikatorem HKNN. Równie ciekawe podejście Hum-mPLoc zaproponował Shen [137].

Jeszcze inną grupą algorytmów stosowanych w tej dziedzinie są metody oparte o ukryte łańcuchy markowa HMM (ang. Hidden Markov Models) [92], [93]. Kilka kolejnych podejść opisanych jest w pracach [42], [50], [109]. Szeroki przegląd metod zajmujących się rozpoznawaniem trzeciorzędowej struktury białek można znaleźć w [143].

Współczynnik poprawnych klasyfikacji, który został osiągnięty w tych pracach wynosi od 48,8% – 62,7%. Pełne zestawienie tych wyników można znaleźć w tabeli 2.8.

3. Efektywne podejście do problemu wieloklasowego

W podrozdziale 1.4 zostały omówione metody podejścia do problemu wieloklasowego. Wiele z nich staje się bardzo kosztowna obliczeniowo i nieefektywna, gdy liczba klas znacząco wzrasta. Tymczasem, podczas gdy jeszcze dwadzieścia, trzydzieści lat temu większość z rozpatrywanych problemów klasyfikacji stanowiły problemy binarne, to współcześnie, zadania klasyfikacji, w których pojawia się kilkadziesiąt klas są normą, a zadania klasyfikacji kilkuset, czy nawet kilku tysięcy klas nie należą do rzadkości.

Wzrost ilości klas w zadaniach klasyfikacji możemy prześledzić przeglądając bazę danych UCI Machine Learning Repository. Na przykład w latach 1998–1992 średnia ilość klas w zadaniu klasyfikacji wynosiła $5,6$, podczas gdy w latach 2008–2012 było to już $35,3$. Tabela 2.1 obrazująca wzrastającą ilość klas w zadaniach klasyfikacji została zamieszczona w podrozdziale 1.3 dotyczącym selekcji cech.

Najlepiej wpływ liczby klas na koszt obliczeniowy algorytmu widać w tabeli 3.1. Jeśli zastosujemy strategię *"jeden przeciw jednemu"* OVO, to w przypadku problemu 10-klasowego potrzebujemy tylko 45 klasyfikatorów binarnych. Liczba ta jednak szybko rośnie wraz z rosnącą ilością klas. W zadaniu rozpoznawania mówionych liter ISOLET potrzebne jest już 325 klasyfikatorów, a problem identyfikacji autorów ACRS wymaga ich ponad tysiąca.

Jeśli zaś rozważymy zadanie opisane w pracy [110] (10000 klas), to ilość klasyfikatorów binarnych wzrasta do niebotycznej liczby 5 miliardów. Nie trudno się domyślić, że zastosowanie strategii OVO w tym przypadku staje się nie tylko nieefektywne, ale wręcz niewykonalne. W tabeli 3.1 zestawiono liczbę klas oraz klasyfikatorów binarnych niezbędnych do sklasyfikowania próbki dla kilku wybranych strategii.

problem	ilość klas	ilość klasyfikatorów binarnych dla strategii		
		OVO	OVA	BDT
rozpoznawanie cyfr	10	45	10	4
rozpoznawanie gestów	22	231	22	5
rozpoznawanie mówionych liter	26	325	26	5
identyfikacja autorów	50	1225	50	11
identyfikacja autorów [110]	100 000	ok. $5 * 10^9$	100 000	17

Tablica 3.1. Zestawienie ilości klasyfikatorów binarnych niezbędnych do sklasyfikowania próbki dla różnych strategii

Jak łatwo zauważyć, w przypadku zastosowania strategii OVO, wpływ ilości klas na złożoność obliczeniową algorytmu klasyfikacji wynosi $O(n^2)$ (gdzie n oznacza liczbę klas). Jako, że za każdym razem musimy skorzystać z pomocy $n*(n-1)/2$ klasyfikatorów binarnych. W przypadku strategii OVR mamy już tylko zależność $O(n)$. Zdecydowanie najlepszą, ze względu złożoność, jest metoda BDT. Należy jednak pamiętać, że w tabeli umieszczono tylko liczbę klasyfikatorów niezbędną na etapie klasyfikowania próbki. Na etapie uczenia może być wymagana większa ich liczba.

Zatem można zapytać czy nie lepiej skorzystać od razu z klasyfikatora wieloklasowego. Jednak również w takim przypadku ilość klas wpływa znacząco na wydajność algorytmu. Na przykład w klasyfikatorze RDA, od ilości klas zależy rozmiar macierzy kowariancji, którą należy odwrócić w procesie klasyfikacji. Odwracanie macierzy o rozmiarze n x n implikuje złożoność $O(n^3)$, przy zastosowaniu standardowego algorytmu. Nawet użycie bardziej wyrafinowanych metod, na przykład metody zaproponowanej przez Coppersmitha i Winograda [32] daje nam złożoność rzędu $O(n^{2,376})$.

Można także skorzystać z technik modyfikacji klasyfikatora SVM [33], [94], które pozwalają klasyfikować od razu na wiele klas. Jednak, tak jak to zostało wspomniane w podrozdziale 1.4, zwiększa to znacząco jego koszt obliczeniowy. Dodatkowo zadanie optymalizacji kwadratowej, które należy wtedy rozwiązać sprawia, że wyniki osiągane przez ten zmodyfikowany klasyfikator nie są zadowalające.

W niniejsza praca zajmuje się technikami podziału problemu wieloklasowego na problemy binarne. Zaproponowano w niej metody pozwalające zmniejszyć zarówno ilość klasyfikatorów binarnych potrzebnych do sklasyfikowania próbki jak i błąd klasyfikacji.

Jak wspomniano w podrozdziale 1.4.3 dobre wyniki daje zastosowanie binarnych drzew decyzyjnych BDT. W przypadku zastosowania tej strategii możemy liczyć na ograniczenie ilości klasyfikatorów niezbędnych do sklasyfikowania próbki do poziomu $O(log_2 n)$.

Oczywiście sposób konstruowania tych drzew ma bardzo duży wpływ na skuteczność algorytmu (błąd klasyfikacji). W podrozdziale 3.1 zostały zaproponowane takie strategie budowania drzew klasyfikatorów binarnych, które pozwalają ją zwiększyć, zachowując jednocześnie "sensowną" liczbę klasyfikatorów.

W kolejnym podrozdziale została zaproponowana metoda przecięć. Ta technika podziału problemu wieloklasowego jest dość podobna do metody OVR. Między innymi wymaga skonstruowania podobnej ilości $O(n)$ klasyfikatorów binarnych. Jednak pozwala ona na zmniejszanie błędu klasyfikacji poprzez stosowanie wielokrotnych podziałów klas na różne podzbiory.

Bardzo ciekawym podejściem do problemu wieloklasowego jest metoda wyjściowych kodów samokorygujących ECOC. W dalszej części tego rozdziału zostanie pokazane, że

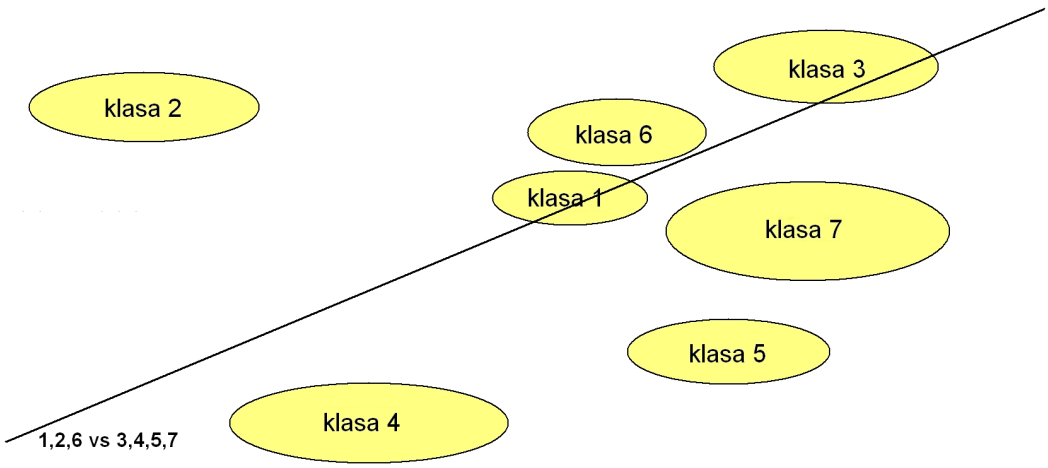
metody konstruowania takich kodów, proponowane w literaturze, są oderwane od związanego z nimi zadania klasyfikacji.

Tworząc macierz słów kodowych definiujemy (kodujemy) podziały zbioru klas na dwa podzbiory. Jednak nie każdy z takich podziałów jest jednakowo ”dobry” (niektóre z nich prowadzą do stosunkowo dużych błędów klasyfikacji). W niniejszej pracy zaproponowana została prosta metoda, która minimalizując błędy generowane przez te podziały (a raczej przez związane z nimi klasyfikatory binarne), pozwala poprawić końcowy wynik algorytmu klasyfikacji.

3.1. Nowa strategia budowania drzew decyzyjnych

Tak jak wspomniano w podrozdziale 1.4.4 głównym problemem jaki pojawia się przy konstruowaniu binarnych drzew decyzyjnych BDT jest to, w jaki sposób podzielić zbiór klas na dwa rozłączne podzbiory, tak aby podział ten był optymalny ze względu na późniejszą klasyfikację próbek w danym węźle drzewa.

Zbiór n klas możemy podzielić na dwa równe lub prawie równe podzbiory (zawierające $\lfloor n/2 \rfloor$ i $\lceil n/2 \rceil$ elementów) na $n!/(\lfloor n/2 \rfloor! * (n - \lfloor n/2 \rfloor)!)$ sposobów. Jednak nie każdy z tych podziałów jest dobry. Niektóre z nich mogą nawet być bardzo szkodliwe przydzielając próbki tej samej klasy do dwóch różnych podzbiorów. Przykład takiego podziału przedstawiono na rysunku 3.1. Z kolei na rysunku 3.2 mamy przykład podziału, który byłby bardzo pożądany. Zatem przy podziale zbioru klas powinniśmy użyć kryterium, które minimalizowałoby ilość błędów na tym etapie klasyfikacji. Takim kryterium może być na przykład wynik klasyfikatora binarnego uzyskany na zbiorze uczącym.



Rysunek 3.1. Przykład złego podziału zbioru klas

Już w przypadku kilkunastu klas ilość możliwych podziałów jest zbyt duża, aby je wszystkie sprawdzić. To oznacza konieczność skorzystania z jakiejś heurystyki. W literaturze zaproponowano kilka metod rozwiązywania tego problemu. Jednak zwiększają one znacząco złożoność algorytmu. Dlatego też w opisanym poniżej podejściu zdecydowano się na użycie losowych podziałów, żądając jednocześnie, aby podziały te uzyskiwały pewien z góry założony poziom $AssumedRecognitionRatio$ współczynnika klasyfikacji Q (mierzony na zbiorze uczącym).

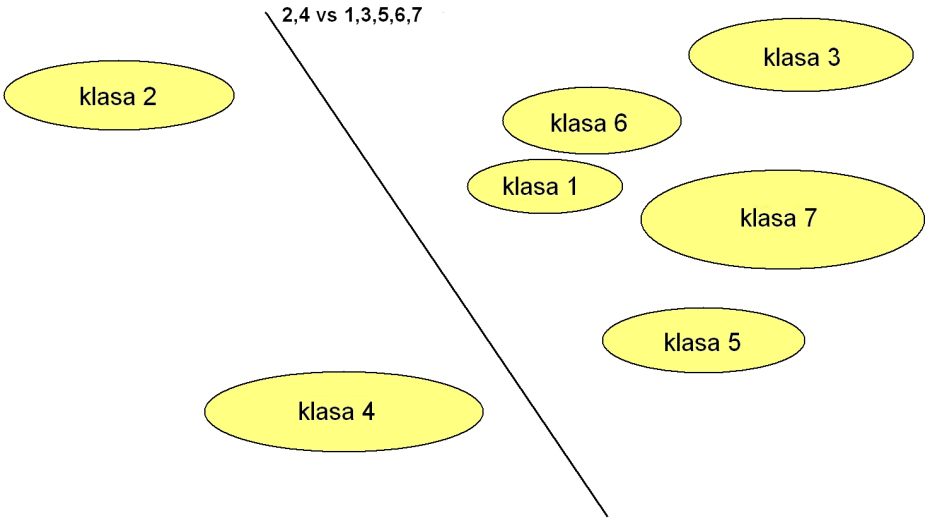
Poniżej przedstawiony został pseudokod algorytmu prezentującego wyżej opisaną strategię. Jednak osiągnięcie tej ustalonej wartości współczynnika klasyfikacji może się okazać trudne, a nawet niemożliwe. Zwłaszcza jeśli zostanie on założony na zbyt wysokim poziomie. Dlatego też w poniższym algorytmie maksymalna liczba iteracji wyszukujących podział została ograniczona do:

$$MaxPartition(k,n) = min\left(\frac{k!}{\lfloor k/2 \rfloor ! * (k - \lfloor k/2 \rfloor)!}, n\right) \tag{3.1}$$

gdzie k jest liczbą klas w węźle, którego poddrzewo jest aktualnie konstruowane, a n jest całkowitą liczbą klas.

W poniższym kodzie funkcje *"Checked"* i *"Store"* są używane, aby uniknąć dwukrotnego sprawdzania tej samej kombinacji. Pierwsza z nich sprawdza czy wylosowana kombinacja nie była już wcześniej testowana, druga zapamiętuje przetestowany podział na liście.

```
1.  i = 0;
2.  while (i < MaxPartitions(k, n)) {
3.        Partition = GenerateRandomPartition();
4.        if (not Checked(Partition, List)) {
5.              Store(Partition, List);
6.              Result = CrossValidate(Partition);
7.              if (Result > AssumedRecognitionRatio)
8.                 return Partition;
9.              else if (Result > MaxResult)
10.                MaxResult = Result;
11.                SavePartition = Partition;
12.             endif
13.             i = i + 1;
14.       }
15. }
16. return SavePartition;
```



Rysunek 3.2. Przykład dobrego podziału klas

Strategia opisana powyżej ma pewną wadę. Zakładamy równy (lub prawie równy) podział klas na dwa podzbiory. Na rysunku 3.2 pokazano siedem różnych klas. Łatwo można zauważyć, że istnieje bardzo dobry podział na dwa podzbiory: $\{2, 4\}$ i $\{1, 3, 5, 6, 7\}$, ale bardzo trudno jest znaleźć dobry podział na dwa podzbiory o prawie równej ilości klas.

Jednak sama struktura drzewa binarnego nie narzuca podziału na równe podzbiory. Jeśli pozwolimy na tworzenie drzewa z użyciem podzbiorów o dowolnej wielkości (w tym jednoelementowych), to drzewo takie stanie się niezrównoważone. Wzrośnie również ilość klasyfikatorów binarnych koniecznych na etapie rozpoznawania próbki.

W przypadku zrównoważonego drzewa decyzyjnego, aby rozpoznać próbkę potrzebujemy $\lceil log_2(n) \rceil$ klasyfikatorów. Jeśli drzewo nie będzie zrównoważone, to liczba ta będzie rosła. Jednak w pesymistycznym przypadku nie przekroczy $n - 1$. Zostanie ona osiągnięta, gdy za każdym razem będziemy mieli do czynienia z podziałem na zbiór zawierający pojedynczą klasę i drugi, który będzie zawierał klasy pozostałe.

Stworzone w taki sposób drzewo, to w rzeczywistości metoda *"jeden przeciw pozostałym"* OVR. Wynika z tego, że jeśli skorzystamy z techniki BDT i będziemy konstruować niezrównoważone drzewa decyzyjne, to w najgorszym przypadku będziemy potrzebowali tylu klasyfikatorów co w metodzie OVR. Przy okazji warto zauważyć, że w każdym z tych przypadków, liczba klasyfikatorów, które musimy skonstruować na etapie uczenia nie zmienia się.

Analizując sposób konstrukcji drzewa decyzyjnego oraz jego późniejsze działanie możemy zauważyć pewną wadę tej metody. Otóż podczas procesu klasyfikacji, przechodzimy od korzenia drzewa w dół, w każdym węźle dzieląc klasyfikowane próbki na dwa rozłączne podzbiory. Łatwo zauważyć, że błędy klasyfikacji popełnione w korzeniu drzewa oraz na

każdym kolejnym poziomie kumulują się. Raz źle przydzielona próbka już nigdy nie zostanie poprawnie sklasyfikowana.

W literaturze są opisane próby radzenia sobie z tym problemem. Jedna z nich [77] polega na korygowaniu otrzymanych podzbiorów próbek po każdej klasyfikacji cząstkowej. Możemy jednak zmienić nieco sposób konstruowania drzewa i w ten sposób spróbować ograniczyć błąd klasyfikacji.

metoda	70%	75%	80%
z użyciem zrównoważonego drzewa BDT	52,5% (51,2% – 53,7%)	53,3% (52,5% – 54,1%)	55,8% (54,4% – 57,1%)
z użyciem niezrównoważonego drzewa BDT	53,5% (52,8% – 54,3%)	53,0% (52,1% – 54,0%)	56,1% (54,8% – 57,4%)
z użyciem zrównoważonego drzewa BDT down-up	52,0% (50,8% – 53,1%)	52,5% (51,4% – 53,5%)	54,0% (53,2% – 54,9%)

Tablica 3.2. Zestawienie wyników przy użyciu drzew BDT na zbiorze białek

metoda	80%	85%	90%
z użyciem zrównoważonego drzewa BDT	70,6% (69,5% – 71,8%)	74,2% (72,9% – 75,4%)	78,5% (77,4% – 79,5%)
z użyciem niezrównoważonego drzewa BDT	70,8% (69,6% – 72,1%)	74,5% (73,1% – 75,9%)	78,9% (77,2% – 80,5%)
z użyciem zrównoważonego drzewa BDT down-up	70,5% (69,4% – 71,7%)	74,1% (72,6% – 75,6%)	78,5% (77,2% – 79,8%)

Tablica 3.3. Zestawienie wyników przy użyciu drzew BDT na zbiorze gestów (A)

metoda	85%	90%	95%
z użyciem zrównoważonego drzewa BDT	90,2% (88,2% – 92,3%)	92,5% (90,8% – 94,2%)	93,1% (91,8% – 94,5%)
z użyciem niezrównoważonego drzewa BDT	90,5% (88,6% – 92,5%)	92,7% (91,0% – 94,3%)	93,5% (93,1% – 94,9%)
z użyciem zrównoważonego drzewa BDT down-up	90,0% (87,9% – 92,0%)	92,1% (89,3% – 93,8%)	92,9% (91,2% – 94,5%)

Tablica 3.4. Zestawienie wyników przy użyciu drzew BDT na zbiorze ISOLET

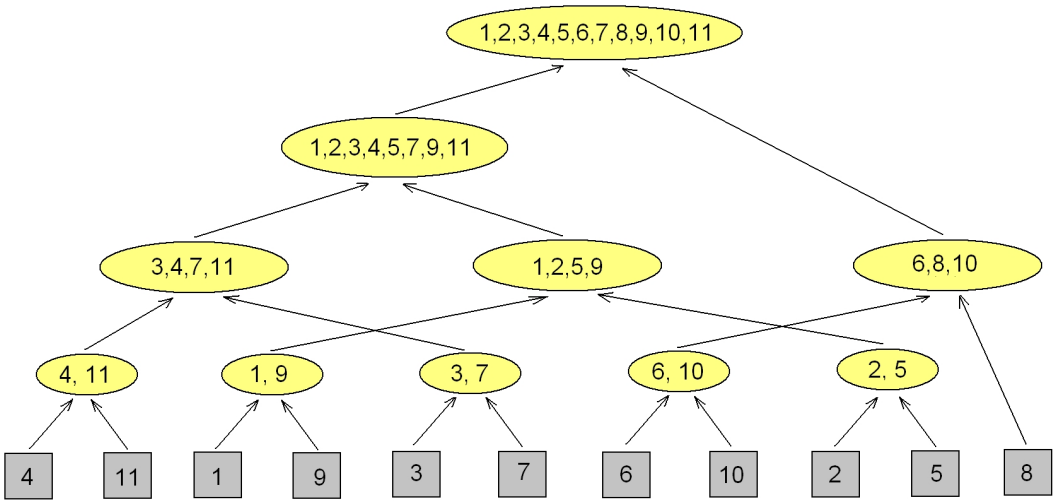
Kumulowanie się błędów oznacza, że nawet jeśli klasyfikatory na najniższym poziomie osiągają bardzo dobre wyniki (nawet 100% poprawnych odpowiedzi), to nie uda się otrzymać lepszego rezultatu jeśli klasyfikatory na wyższych poziomach popełniły zbyt dużo błędów. Skoro tak, to może odwrócić proces konstrukcji drzewa? Spróbujmy rozpocząć od utworzenia zbiorów dwuelementowych z klas, które najlepiej poddają się procesowi klasyfikacji.

metoda	80%	85%	90%
z użyciem zrównoważonego drzewa BDT	71,2% (68,9% – 73,4%)	71,7% (69,6% – 73,8%)	72,4% (70,3% – 74,4%)
z użyciem niezrównoważonego drzewa BDT	71,5% (69,6% – 73,5%)	72,2% (70,5% – 74,0%)	72,4% (70,9% – 73,5%)
z użyciem zrównoważonego drzewa BDT down-up	71,2% (69,0% – 73,3%)	71,8% (69,9% – 73,9%)	72,2% (70,5% – 73,9%)

Tablica 3.5. Zestawienie wyników przy użyciu drzew BDT na zbiorze ACRS

Następnie pary te możemy łączyć w czteroelementowe zbiory, stosując to samo kryterium do stworzonych w poprzednim kroku par. W ten sposób drzewo decyzyjne zostanie zbudowane od liści w kierunku korzenia. Łatwo zauważyć, że ten sposób budowania drzewa będzie prowadził do zrównoważonego drzewa decyzyjnego.

Opisany powyżej proces będzie działał przy założeniu, że liczba klas jest całkowitą potęgą dwójki. W pozostałych przypadkach będziemy zmuszeni do tworzenia węzłów drzewa zawierających podzbiory o różnej ilości klas. Przykładową konstrukcję drzewa z 11 klas przedstawia schematycznie rysunek 3.3.



Rysunek 3.3. Przykład konstrukcji drzewa metodą z dołu do góry

Rezultaty trzech zaproponowanych strategii zestawiono w tabelach 3.2 – 3.5. W wierszach tabeli mamy wymienione poszczególne metody, zaś w kolumnach założone minimalne współczynniki klasyfikacji. Każdy wynik został podany w postaci pojedynczej wartości będącej średnim wynikiem k-kroswalidacji (pierwszy wiersz) oraz w postaci przedziału ufności na poziomie 95% (drugi wiersz).

Poszczególne tabele prezentują wyniki dla kolejnych zbiorów danych za wyjątkiem zbioru cyfr MNIST. Zbiór ten został wykluczony ze względu na małą liczbę klas. W przypadku tego

zbioru stosowanie opisanej heurystyki nie na sensu, ponieważ jesteśmy w stanie sprawdzić wszystkie 252 możliwe podziały zbioru 10 cech na dwa równe podzbiory. Drugim powodem, dla którego ten zbiór został pominięty, jest fakt, że w jego przypadku do zastosowania strategii OVO wymagane jest użycie jedynie 45 klasyfikatorów.

Analizując tabele możemy zauważyć, że im większy założony współczynnik Q, tym lepsze wyniki uzyskiwane przez klasyfikator końcowy. Jeśli przypomnimy sobie dyskusję na temat kumulacji błędów, to wynik ten staje się oczywisty. Im mniej błędów popełniają klasyfikatory na poszczególnych poziomach drzewa, tym ogólny wynik będzie lepszy.

Jednak, aby sprawdzić czy ta prawidłowość jest statystycznie znacząca założono (jako hipotezę zerową), że liczba błędów popełnianych przez te klasyfikatory jest taka sama. Po przeprowadzaniu testów permutacyjnych (założono poziom 1000 iteracji) hipoteza ta została odrzucona z prawdopodobieństwem popełnienia błędu na poziomie 1%.

Powstaje zatem pytanie czy nie można przyjąć wyższych współczynników $AssumedRecognitionRatio$. Taka strategia rodzi jednak niebezpieczeństwo, że założony współczynnik może być nieosiągalny. W takim wypadku sprawdzone zostanie $MaxPartition$ podziałów – zgodnie ze wzorem 3.1. Z ograniczenia tego nie możemy zrezygnować, ponieważ w przeciwnym przypadku algorytm mógłby się nigdy nie zakończyć.

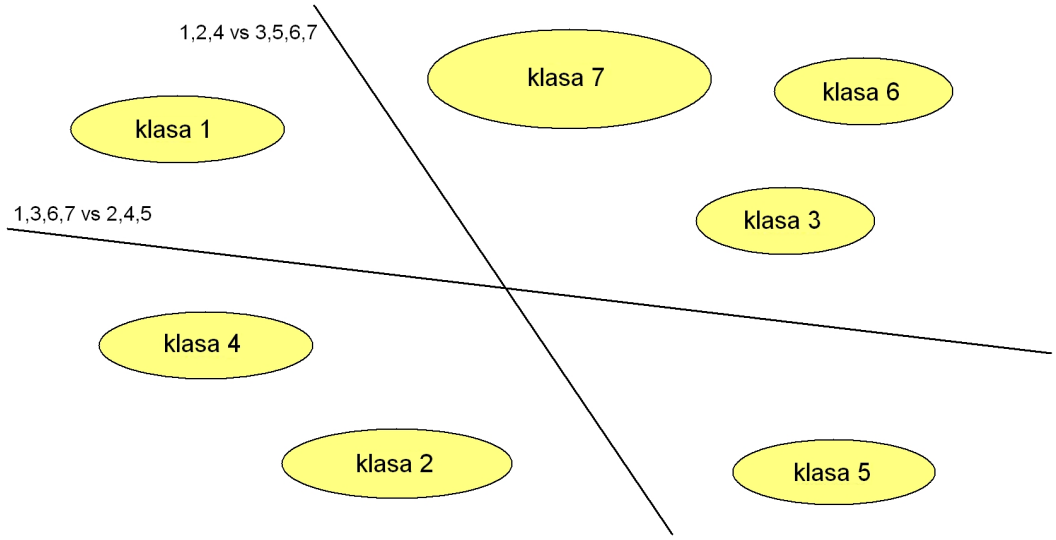
Nim zajmiemy się porównaniem zaproponowanych metod ze strategiami OVO i OVR, przedstawimy jeszcze jedną próbę podejścia do rozważanego tematu.

3.2. Metoda przecięć

Na rysunku 3.4 przedstawiono schematycznie dwa klasyfikatory binarne. Jeden z nich dzieli klasy na podzbiory $\{1, 2, 4\}$ i $\{3, 5, 6, 7\}$, a drugi na podzbiory $\{2, 4, 5\}$ i $\{1, 3, 6, 7\}$. Bardzo łatwo można zauważyć, że przecięcie tych dwóch klasyfikatorów może zostać wykorzystane do wydzielenia czterech podzbiorów: $\{2, 4\}$ – $\{3, 6, 7\}$ – $\{1\}$ – $\{5\}$.

Taka procedura może zostać rekursywnie powtórzona aż do uzyskania zbiorów jednoelementowych. Podzbiór $\{3, 6, 7\}$ podzielony zostanie za pomocą klasyfikatorów 7 vs 3,6 oraz 3 vs 6,7, a podzbiór $\{2, 4\}$ za pomocą klasyfikatora 2 vs 4. Metodę tę możemy obrazowo nazwać metodą nożyczek (lub metodą przecięć) [23], gdyż za każdym podziałem *"wycina"* z pewnego podzbioru klas, co najmniej dwie pojedyncze klasy. Rysunek 3.5 obrazuje działanie tego algorytmu.

Możliwe jest jeszcze inne podejście. Zauważmy, że po wykonaniu pierwszego kroku i *"wycięciu"* klas 1 i 5, pozostanie nam podzbiór $\{2, 3, 4, 6, 7\}$. Stosując jeszcze raz metodę nożyczek, tym razem klasyfikatory 2,7 vs 3,4,6 oraz 2,4 vs 3,6,7, otrzymamy podział na podzbiory: $\{3, 6\}$ – $\{2\}$ – $\{4\}$ – $\{7\}$. Teraz wystarczy użyć jeszcze klasyfikatora 3 vs 6, aby wyodrębnić pozostałe dwie klasy.



Rysunek 3.4. Przecięcie się dwóch klasyfikatorów

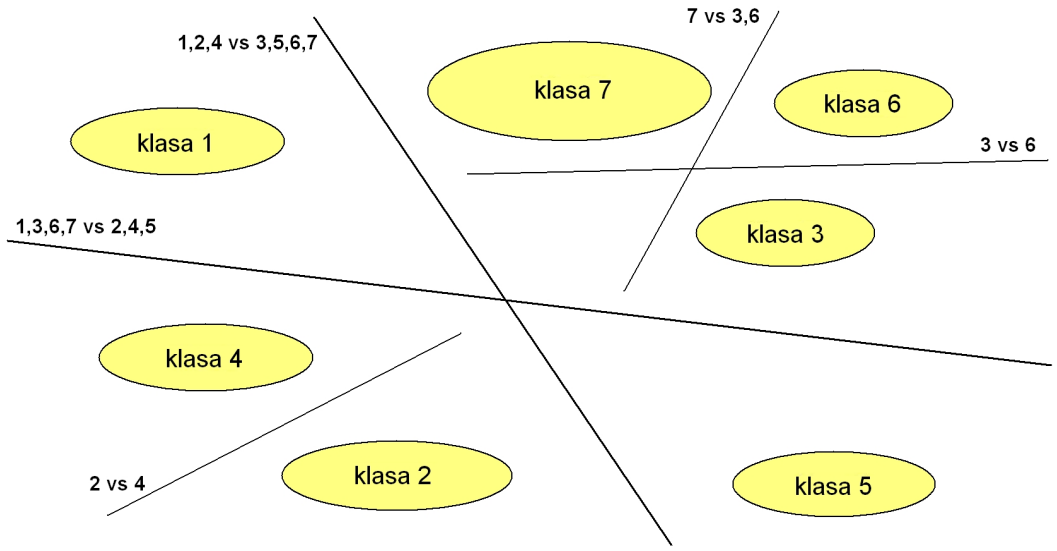
Spróbujmy oszacować ilość klasyfikatorów niezbędnych do sklasyfikowania wszystkich próbek za pomocą opisanych wyżej metod. W przypadku metody nożyczek (strategii rekursywnej) najlepszy możliwy podział, to taki, który wycinając dwie klasy, pozostałe dzieli na podzbiory, z których jeden jest jednoelementowy. Taki podział wydziela trzy klasy za pomocą dwóch klasyfikatorów, pozostawiając zbiór $N-3$ klas do kolejnego etapu. Oznacza to, że w tym przypadku będziemy potrzebowali maksymalnie $\lceil N/3 \rceil$ etapów czyli $2 * \lceil N/3 \rceil$ klasyfikatorów.

W pesymistycznym przypadku każdy podział *'wycina"* dwie klasy, a tylko ostatnie dwa *'wycinają"* po trzy. Zatem maksymalnie będziemy potrzebować $N-2$ klasyfikatorów składowych. Powyższe szacunki dotyczą liczby klasyfikatorów, które muszą zostać skonstruowane.

Jednak na etapie klasyfikacji poszczególnych próbek potrzeba będzie ich znacznie mniej. Do sklasyfikowania niektórych z nich wystarczą już 2 klasyfikatory (jeśli próbki te należą do jednej z dwóch klas, które *"wycinane"* są jako pierwsze. W przypadku próbek *"wycinanych"* jako ostatnie potrzebne będzie od $\lceil log_2(N) \rceil$ do $2 * \lceil N/3 \rceil$ klasyfikatorów.

Nieco prościej będzie w przypadku zastosowania drugiej wersji metody nożyczek, ponieważ za każdym razem *"wycinane"* są dwie klasy (tylko za ostatnim razem cztery). Oznacza to, że skonstruować musimy $N-2$ klasyfikatorów. Do sklasyfikowania próbki potrzebujemy ich od 2 do $N-2$.

Porównując powyższe metody z metodą BDT możemy zauważyć, że zwłaszcza pierwsza z nich pozwala nie tylko ograniczyć ilość konstruowanych klasyfikatorów binarnych, ale także dla wielu próbek zmniejsza ich liczbę na etapie klasyfikacji.



Rysunek 3.5. Podział zbioru na pojedyncze klasy

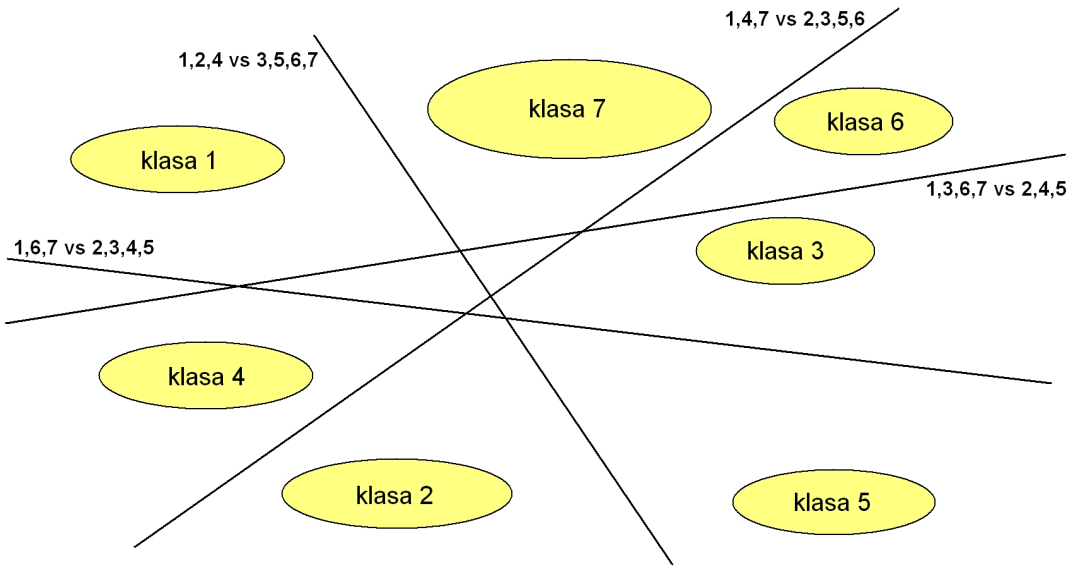
Jeszcze inna metodologia przedstawiona jest na rysunku 3.6. W tym przypadku zawsze dzielimy zbiór klas na dwa podzbiory o równej lub prawie równej wielkości. Jak łatwo można zauważyć po dokonaniu czterech takich podziałów wszystkie klasy zostaną rozpoznane. Strategię tę będziemy nazywali *"pół na pół"*.

W przypadku metody *"pół na pół"* każde dwa podziały musi różnić co najmniej jedna klasa. W przeciwnym wypadku są one takie same. To oznacza, że w najgorszym razie potrzebować będziemy $2 * (N - 1)$ takich podziałów. W najlepszym każdy podział "pomoże" w wydzieleniu dwóch klas, a zatem będzie potrzebne ich tylko $\lceil N/2 \rceil$.

Podobnie jak i w poprzednio opisywanych metodach liczba klasyfikatorów niezbędnych do sklasyfikowania próbki nie będzie stała. Tym razem będzie się zmieniała w zakresie od 2 dla próbek należących do klas wydzielanych jako pierwsze i od $\lceil N/2 \rceil$ do $2 * (N - 1)$ dla próbek wydzielanych jako ostatnie.

Przeglądając tabele 3.6 i 3.7 (kolumna "bez głosowania") łatwo zauważyć, że wyniki osiągane przez tak skonstruowane klasyfikatory są bardzo słabe. Możemy jednak wykorzystać fakt, że przy ich konstrukcji używamy losowych podziałów klas. Oznacza to, każde kolejne uruchomienie opisanej procedury doprowadzi do utworzenia innego klasyfikatora. Różnić będą się także błędy popełniane przez takie klasyfikatory.

W podrozdziale 1.2.4 opisując metodę zepołów klasyfikatorów wspomniane zostało, że mając do dyspozycji wiele słabych klasyfikatorów możemy, poprzez połączenie ich w jeden złożony klasyfikator, znacznie zmniejszyć błąd klasyfikacji. Ważne przy tym jest, aby wyniki uzyskiwane przez członków takiego zespołu były niezależne statystycznie oraz aby były to klasyfikatory stosunkowo szybkie. Oba te warunki są w tym wypadku spełnione.



Rysunek 3.6. Podział zbioru na pojedyncze klasy metodą pół-na-pół

metoda	sposób głosowania			
	bez głosowania	20 głosowań	40 głosowań	60 głosowań
baza danych białek	38,1% (37,2% – 38,9%)	53,5% (52,4% – 54,5%)	54,3% (53,5% – 55,0%)	54,7% (53,9% – 55,6%)
baza danych gestów (A)	54,2% (50,1% – 59,6%)	69,9% (67,2% – 72,6%)	74,7% (72,3% – 77,1%)	75,6% (73,2% – 77,2%)
baza danych ISOLET	71,1% (64,2% – 77,4%)	83,4% (79,8% – 87,2%)	88,8% (86,1% – 90,4%)	90,2% 88,9% – 91,5%)
baza danych ACRS	53,5% (49,9% – 57,6%)	66,3% (63,6% – 68,5%)	68,3% (66,5% – 70,0%)	69,7% (67,2% – 71,2%)

Tablica 3.6. Zestawienie wyników przy użyciu metody przecięć, podejście rekursywne

W przeprowadzonych eksperymentach opisana wyżej procedura była uruchamiana 20, 40 i 60 razy. Wyniki uzyskiwane przez poszczególne klasyfikatory były zapamiętywane, a następnie wynik końcowy został ustalony metodą głosowania. Otrzymane wyniki przedstawione są w tabelach 3.6 i 3.7. Podobnie jak w przypadku drzew decyzyjnych BDT podany został średni wynik k-kroswalidacji (przyjęto $k = 6$) oraz przedział ufności na poziomie 95%.

Kolejna tabela 3.8 to zestawienie wyników uzyskanych za pomocą wszystkich zaproponowanych metod. Przy czym poszczególnymi cyframi zostały oznaczone: 1 – metoda z użyciem zrównoważonego drzewa BDT, 2 – metoda z użyciem niezrównoważonego drzewa BDT, 3 – metoda z użyciem zrównoważonego drzewa BDT down-up, 4 – metoda przecięć, strategia rekursywna, 5 – metoda *"pół na pół"*. W przypadku każdej z metod wybrano najlepszy osiągnięty wynik.

metoda	sposób głosowania			
	bez głosowania	20 głosowań	40 głosowań	60 głosowań
baza danych białek	40,1% (39,4% – 41,7%)	53,2% (52,4% – 54,3%)	54,7% (53,7% – 55,8%)	55,4% (54,3% – 56,5%)
baza danych gestów	54,4% (50,9% – 59,2%)	70,7% (67,9% – 73,4%)	75,8% (72,9% – 78,5%)	77,3% (74,7% – 79,8%)
baza danych ISOLET	71,7% (67,2% – 77,1%)	84,5% (81,3% – 88,4%)	89,7% (87,5% – 92,8%)	91,3% (89,4% – 92,6%)
baza danych ACRS	49,4% (44,2% – 54,2%)	63,2% (59,9% – 67,3%)	66,3% (63,4% – 68,2%)	67,1% (65,3% – 68,4%)

Tablica 3.7. Zestawienie wyników przy użyciu metody przecięć, podejście pół na pół

baza	zastosowana metoda						
	SVM-OVR	SVM – OVO	1	2	3	4	5
białek	34,3%	57,2%	55,8%	56,1%	54,0%	54,7%	55,4%
gestów (A)	59,8%	81,1%	78,5%	78,9%	78,5%	75,6%	77,3%
ISOLET	69,4%	96,5%	93,1%	93,5%	92,9%	90,2%	91,3%
ACRS	60,7%	73,2%	72,4%	72,4%	72,2%	69,7%	67,1%

Tablica 3.8. Zestawienie wyników osiągniętych przy pomocy różnych strategii (klasyfikator SVM)

Wyniki zastosowania zaproponowanych strategii są obiecujące. We wszystkich przypadkach osiągnięto lepszy współczynnik klasyfikacji niż w przypadku metody OVR. Wprawdzie żadnej z zaproponowanych strategii nie udało się poprawić wyniku osiągniętego za pomocą metody OVO, ale należy pamiętać, że wszystkie klasyfikatory 1 – 5 mają dużo mniejszą złożoność obliczeniową, co przekłada się na czas działania algorytmów.

Oczywiście w przypadku jeśli ilość klas pozwala na zastosowanie strategii OVO, to głównym celem do którego powinniśmy dążyć jest osiągnięcie jak najlepszego współczynnika klasyfikacji. Dyskusja złożoności obliczeniowej zaproponowanych rozwiązań wskazuje, że metody te mogą być stosowane w przypadku, gdy ilość klas powoduje, że skorzystanie ze strategii OVO jest zbyt czasochłonne. Z tabeli 3.8 wynika jednak, że w przypadku proponowanych metod musimy liczyć się wtedy z gorszym wynikiem klasyfikatora.

W przeprowadzonych eksperymentach zaproponowane algorytmy wykonywały się od kilku do kilkunastu razy szybciej w stosunku do metody OVR i od kilkunastu do kilkudziesięciu razy szybciej w stosunku do metody OVO. Zgodnie z oczekiwaniami zysk był tym większy z im większą liczbą klas musiał się zmierzyć klasyfikator.

3.3. Tworzenie efektywnych kodów ECOC

W podrozdziale 1.4.4 została opisana metoda tworzenia wyjściowych kodów samokorygujących. Przedstawiono w jaki sposób możemy tworzyć kody optymalne. To znaczy takie, które zapewniają maksymalną separowalność kodów wyjściowych reprezentujących klasy. W przypadku, gdy stworzenie takiego kodu jest niemożliwe ze względu na ilość kombinacji, przedstawione zostały strategie poszukiwania kodu suboptymalnego.

Niemniej jednak łatwo można zauważyć, że definicja kodu optymalnego zakłada, że błędy popełniane podczas "transmisji" słowa kodowego (co w naszym wypadku oznacza klasyfikację za pomocą klasyfikatora binarnego zdefiniowanego przez kolumnę macierzy kodów) są losowe. Takie założenie jest jednak nieprawdziwe. Jedne klasyfikatory popełniają więcej błędów niż inne. Zastanówmy się najpierw jak ilość błędów popełnianych przez nie wpływa na wynik końcowy.

Dokładna ocena tego wpływu nie jest możliwa, ponieważ musielibyśmy wiedzieć jak wygląda rozkład błędów każdego z użytych klasyfikatorów binarnych. Jednak możemy wpływ ten szacować zakładając, że każdy klasyfikator popełnia średnio taką samą liczbę błędów i że są one rozłożone równomiernie. Wtedy ilość błędów popełnionych przez wszystkie klasyfikatory będzie zależała wyłącznie od długości słowa kodowego.

Im dłuższe słowo kodowe i im mniejsza sprawność klasyfikatora binarnego, tym suma błędów będzie większa. Aby wynik końcowej klasyfikacji był poprawny, oczekiwana liczba błędów powinna być mniejsza od połowy odległości Hamminga pomiędzy wierszami. Optymalnie byłoby, gdyby zawsze liczba popełnianych błędów była mniejsza od połowy minimalnej odległości Hamminga pomiędzy wierszami. W takim przypadku i przy założeniu idealnie równomiernego rozkładu błędów mielibyśmy efektywność klasyfikacji na poziomie 100%. Niestety w rzeczywistości błędy te nie rozkładają się równomiernie.

Zatem, z jednej strony powinniśmy dążyć do jak najdłuższego słowa kodowego, ponieważ jego wydłużanie będzie powodować wzrost odległości Hamminga pomiędzy wierszami. Jednak z drugiej strony wydłużanie tego słowa spowoduje, że suma błędów popełnianych przez klasyfikatory binarne będzie rosła. Oznacza to, że bardzo trudno znaleźć jest jego optymalną długość. Z eksperymentów przeprowadzanych na potrzeby niniejszej rozprawy wynika, że dobre wyniki uzyskujemy dla słów kodowych o długości $2n - 3n$, gdzie n oznacza liczbę klas.

Przypomnijmy, że każda kolumna macierzy kodów odpowiada klasyfikatorowi binarnemu dzielącemu zbiór klas na dwa podzbiory. Jednakże niektóre z tych podzbiorów mogą być łatwiej separowalne od innych. Możemy to zobaczyć na rysunkach 3.1 i 3.2. Zatem jeśli podzbiory są trudne do podziału, to odpowiedni klasyfikator popełnia dużo błędów. Takich klasyfikatorów (i odpowiadających im kolumn) powinniśmy unikać. Możemy więc podzielić nasz algorytm

na dwa etapy. W pierwszym utworzony zostanie ranking pewnej liczby losowo wybranych kolumn, a następnie korzystając z tego rankingu budowane będą kody ECOC.

baza	ilość podziałów w rankingu	współczynnik klasyfikacji w rankingu na pozycji	
		pierwszej	ostatniej
białek	8192	50,5%	90,4%
gestów (A)	8192	58,3%	93,5%
ISOLET	8192	67,3%	98,7%
ACRS	16384	54,5%	94,3%

Tablica 3.9. Współczynniki klasyfikacji podziałów: pierwszego i ostatniego w rankingu

Przeglądając tabelę 3.9 możemy zauważyć, że współczynniki klasyfikacji osiągane przez klasyfikatory definiowane przez pierwszą i ostatnią w rankingu kolumnę różnią się drastycznie. W przypadku bazy danych białek klasyfikator powiązany z pierwszą kolumną osiągał 90,4% podczas, gdy klasyfikator powiązany z ostatnią z kolumn osiągnął wynik zaledwie 50,5%. Najmniejsze różnice, dla bazy danych ISOLET, są równie znaczące. Od 67,3% dla ostatniej kolumny w rankingu do 98,7% dla pierwszej.

Wybranie L pierwszych kolumn z takiego rankingu i skonstruowanie z nich słów kodowych będzie minimalizowało sumę błędów popełnianych przez klasyfikatory składowe. Niestety taka strategia powoduje, że wykorzystujemy kolumny, które leżą bardzo blisko siebie. Dodatkowo przeglądając tabelę 3.12 możemy zauważyć, że minimalna odległość Hamminga pomiędzy, generowanymi w ten sposób słowami kodowymi, również będzie bardzo mała. Lepiej zatem byłoby wybierać te kolumny z rankingu, które znajdują się w pewnej odległości od siebie. W przeprowadzonych eksperymentach zastosowano zatem następujące strategie:

— Znaną z literatury strategię RHC – szerszy opis znajduje się podrozdziale 1.4.4.
— Pierwsze kolumny z rankingu FCRL (ang. First Columns from the Ranking List). Do stworzenia macierzy słów kodowych algorytm bierze pierwsze L kolumn z listy rankingowej (gdzie przez L rozumiemy długość słowa kodowego).
— Odległe kolumny z rankingu DCRL (ang. Distant Columns from the Ranking List). Algorytm bierze pierwszą kolumnę z listy. Następnie szuka kolejnej kolumny, która jest odległa od pierwszej o co najmniej D. Następnie szukamy kolumny, która jest odległa od dwu pierwszych kolumn o co najmniej D. Wartość D zależy od ilości klas $D < C$ i musi zostać wyznaczona eksperymentalnie.
— Zmodyfikowana strategia odległych kolumn z rankingu MDCRL (ang. Modified Distant Columns from the Ranking List). Macierz słów kodowych jest budowana tak jak poprzednio. Następnie przeprowadzana jest specjalna procedura, która pozwala zwiększyć minimalną odległość Hamminga pomiędzy wierszami. Zaczynając od ostatniej dodanej kolumny, algorytm próbuje zamienić ją na kolumnę znajdującą się wyżej w rankingu.

Zamiana jest dokonywana tylko wtedy gdy spowoduje wzrost minimalnej odległości Hamminga pomiędzy słowami kodowymi (wierszami macierzy).

Poniżej przedstawiono pseudokod algorytmu DCRL:

1. Zbuduj listę kodów ECOC_List o długości Len.
2. Posortuj listę kodów malejąco pod względem uzyskanego współczynnika klasyfikacji
3. $CodeMatrix \leftarrow CodeMatrix \cup ECOC\_List[0]$
4. $Len \leftarrow 1, i = 1$
5. Jeżeli minimalna odległość Hamminga pomiędzy ECOC_List[i] i $CodeMatrix \geq C$, to $CodeMatrix \leftarrow CodeMatrix \cup ECOC\_List[i]$, $Len \leftarrow Len + 1$, $i \leftarrow i + 1$
6. w przeciwnym przypadku $i \leftarrow i + 1$
7. Jeśli $Len \geq CodeLen$, to zakończ działanie algorytmu
8. Przejdź do 5.

W algorytmie MDCRL po znalezieniu macierzy kodów wykonywana jest dodatkowa procedura, która próbuje ”ulepszyć” odległości pomiędzy wierszami macierzy. Pseudokod tej procedury został zamieszczony poniżej:

1. $MinHamDist \leftarrow$ obliczona minimalna odległość Hamminga dla macierzy $CodeMatrix$
2. $Ind \leftarrow CodeLen - 1$
3. $Pos \leftarrow$ znajdź pozycję kodu $CodeMatrix[Ind]$ na liście $ECOC\_List$
4. $i \leftarrow 0$
5. Zamień kolumny $ECOC\_List[i] \leftrightarrow CodeMatrix[Ind]$
6. $NewHamDist \leftarrow$ obliczona minimalna odległość Hamminga dla macierzy $CodeMatrix$
7. Jeśli $NewHamDist > MinHamDist$, to $MinHamDist \leftarrow NewHamDist$
8. W przeciwnym przypadku zamień kolumny $ECOC\_List[i] \leftrightarrow CodeMatrix[Ind]$
9. $i \leftarrow i + 1$
10. Jeśli $i < Pos$, to Przejdź do 5.
11. $Ind \leftarrow Ind - 1$
12. Jeśli $Ind = 0$, to zakończ działanie algorytmu
13. Przejdź do 3.

W przeprowadzonych eksperymentach użyte zostały słowa kodowe o długościach $32, 48, 64, 80, 96, 128$ i 256. Uzyskane wyniki zostały przedstawione w tabelach 3.10 i 3.11. Eksperymenty przeprowadzono jedynie dla baz danych białek oraz bazy ACRS, ponieważ w przypadku małej ilości klas istnieje możliwość skonstruowania kodów pełnych i wybrania z nich optymalnych kodów ECOC. Poszerzona dyskusja przeprowadzona została w oparciu o wyniki uzyskane w eksperymencie z białkami.

Jak łatwo można zauważyć w tej metodzie najważniejsze jest uzyskanie jak największych wartości minimalnej odległości Hamminga. Z tabeli 3.10 widać, że w przypadku metody FCRL wartości te są jednak bardzo małe. Co więcej dla słowa kodowego o długości 32, ta wartość wynosi nawet 0. Co oznacza, że pewne klasy są nieseparowalne.

Jednak jak wynika z danych zawartych w tabeli 3.12, wynik osiągnięty przez dla słowa kodowego o tej długości jest lepszy dla metody FCRL niż dla metody RHC. Mimo iż ta ostatnia generuje słowa kodowe o minimalnej odległości Hamminga równej 8. Taki wynik eksperymentu nie zaskakuje, gdy spojrzymy na średnią odległość Hamminga pomiędzy wierszami. W przypadku metody FCRL wynosi ona 13, a zatem większość klas jest separowalna, a wybór kolumn będących na szczycie rankingu powoduje, że ilość błędów klasyfikacji popełniana przez klasyfikatory składowe jest bardzo mała.

długość kodu	RHC	FCRL	DCRL	MDCRL
32	52,0%	52,5%	56,1%	57,7%
64	54,8%	57,1%	61,0%	60,8%
80	53,0%	56,1%	61,3%	62,6%
96	54,0%	56,4%	60,8%	62,3%
128	53.5%	56,9%	59,7%	61,6%
256	53.8%	57,1%	59,7%	56,4%

Tablica 3.10. Wyniki uzyskane z użyciem utworzonych kodów ECOC dla białek

długość kodu	RHC	FCRL	DCRL	MDCRL
32	16,1%	18,5%	19,1%	19,3%
64	43,2%	42,7%	48,3%	49,2%
80	64,8%	52,3%	68,3%	69,9%
96	65,7%	65,2%	69,3%	72,1%
128	68,2%	68,7%	72,2%	74,5%
256	68,1%	69,2%	72,0%	73,7%

Tablica 3.11. Wyniki uzyskane z użyciem utworzonych kodów ECOC dla ACRS

Przyjrzyjmy się uważnie tabeli 3.12. Przedstawia ona minimalne i średnie odległości Hamminga pomiędzy wierszami macierzy kodów uzyskane za pomocą różnych strategii. Metoda RHC uzyskuje najlepsze wyniki w przypadku obu wartości. Jednak mimo, że słowa kodowe są zbliżone do optymalnych, to wyniki osiągane za ich pomocą są dużo gorsze (patrz 3.10). Nawet metoda FCRL, która daje bardzo słabe słowa kodowe uzyskuje lepsze rezultaty. Podobne wyniki możemy zauważyć w kolejnej tabeli opisującej zachowanie się słów kodowych dla bazy danych ACRS. Oczywiście, w tym przypadku odpowiednie odległości są mniejsze, co wynika z większej ilości klas i tym samym większej ilości wierszy w macierzy kodów.

Jak łatwo się domyślić poprawę wyniku uzyskujemy korzystając z lepszych klasyfikatorów binarnych. To znaczy takich, które popełniają mniejszą liczbę błędów. Na przykład

klasyfikatory powiązane z kolumnami wykorzystywanymi w metodzie FCRL uzyskiwały efektywność na poziome 90% (czyli popełniały średnio 10% błędów), podczas gdy średnia dla klasyfikatorów używanych w metodzie RHC wynosiła około 70% (czyli 30% błędów).

Niemniej jednak możemy zauważyć, że korzystając z metody FCRL minimalna odległość Hamminga, nawet dla stosunkowo długich kodów, jest dość mała. Na przykład dla kodu o długości 128 wynosi zaledwie 6. Oznacza to, że możemy odróżnić od siebie dwie najbliższe klasy pod warunkiem, że nie popełnimy więcej niż $(6-1)/2 = 2$ błędy. To znacząco rzutuje na efektywność klasyfikatora zbudowanego tą metodą.

Z tym problemem stara się poradzić sobie metoda DCRL. Sposób konstrukcji słów kodowych powoduje, że uzyskujemy słowa o dużo większej minimalnej odległości Hamminga. Odbywa się to kosztem użycia kolumn (i związanych z nimi klasyfikatorów binarnych) gorszej jakości. Możemy zauważyć, że zwiększa się również średnia odległość między wierszami. Wprawdzie klasyfikatory binarne popełniają więcej błędów jednak tolerancja klasyfikatora finalnego zwiększa się dzięki dużo większej minimalnej odległości Hamminga. Na przykład w przypadku kodu o długości 256 i użyciu metody DCRL możemy popełnić nawet $(63-1)/2 = 31$ błędów i nadal będziemy w stanie poprawnie odróżnić dowolne dwie klasy.

długość kodu	RHC	FCRL	DCRL	MDCRL
32	16,2 – 8	13,0 – 0	14,4 – 0	14,2 – 4
64	32,5 – 27	26,7 – 1	29,7 – 7	29,4 – 13
80	40,6 – 33	33,7 – 1	37,6 – 15	37,3 – 20
96	48,9 – 43	40,8 – 2	45,7 – 18	45,1 – 28
128	65,4 – 59	55,7 – 6	61,7 – 29	60,9 – 42
256	130,1 – 122	115,3 – 19	88,8 – 63	88,0 – 71

Tablica 3.12. Zestawienie minimalnych i średnich odległości Hamminga pomiędzy wierszami dla zbioru białek

długość kodu	RHC	FCRL	DCRL	MDCRL
32	12,1 – 4	8,3 – 0	10,2 – 0	9,3 – 1
64	21,7 – 15	14,7 – 0	17,4 – 3	16,5 – 7
80	33,7 – 21	19,4 – 1	23,3 – 8	22,9 – 11
96	40,5 – 34	35,3 – 1	37,8 – 11	36,3 – 17
128	47,3 – 42	42,1 – 2	44,6 – 19	43,2 – 27
256	93,2 – 74	82,6 – 7	62,3 – 35	61,9 – 46

Tablica 3.13. Zestawienie minimalnych i średnich odległości Hamminga pomiędzy wierszami dla zbioru ACRS

Kolejną poprawę jakości możemy uzyskać korzystając ze zmodyfikowanej strategii MDCRL. Jako, że jej jedynym zadaniem jest zamiana kolumny na lepszą (tzn. będącą wyżej w rankingu) i co ważne nie powodującą pogorszenia minimalnej odległości pomiędzy

wierszami, to wynik klasyfikatora znów się poprawia. Widać to również w tabeli 3.12. Minimalna odległość Hamminga rośnie, choć możemy zauważyć, że jej średnia wartość nieznacznie spada.

baza danych	długość kodu	RHC	FCRL	DCRL	MDCRL	SVM-OVO
białka	80	54,8%	56,1%	61,0%	62,6%	57,2%
ISOLET	80	89,2%	90,3%	93,1%	96,9%	96,5%
gesty (A)	64	72,7%	74,3%	79,8%	83,0%	81,1%
ACRS	128	68,2%	68,7%	72,2%	74,5%	73,2%

Tablica 3.14. Zestawienie wyników uzyskanych na różnych bazach danych

W tabeli 3.14 zestawiono wyniki uzyskane za pomocą strategii ECOC z wynikami uzyskanymi za pomocą strategii OVO. W drugiej kolumnie umieszczona została długość kodu, dla którego osiągnięto najlepsze rezultaty, zaś w kolejnych kolumnach wyniki otrzymane przy użyciu poszczególnych metod tworzenia kodów ECOC.

Jak łatwo zauważyć wyniki uzyskane z wykorzystaniem kodów ECOC są lepsze niż rezultaty stosowania strategii OVO. Poprawa została osiągnięta na wszystkich testowanych bazach danych. Przy czym należy pamiętać, że długość kodu ECOC odpowiada ilości klasyfikatorów binarnych, które są niezbędne, aby sklasyfikować próbkę. Daje to pewien pogląd na to jak efektywnym rozwiązaniem są kody ECOC.

Dla porównania w przypadku bazy danych ISOLET najlepsze wyniki uzyskujemy stosując kod o długości 80 bitów, podczas gdy w przypadku stosowania strategii OVO potrzebowalibyśmy 321 klasyfikatorów. W przypadku bazy danych ACRS jest to odpowiednio słowo kodowe o długości 128 bitów i 1225 klasyfikatorów potrzebnych w strategii OVO.

Przeglądając tabelę 3.14 możemy również zauważyć, że długość słowa kodowego, dla którego osiągamy najlepsze wyniki, rośnie wraz z ilością klas w zadaniu klasyfikacji. W przypadku bazy danych gestów wynosi ona 64 bity, zaś w przypadku bazy ACRS jest to już 128 bitów. Jednak wzrost ten jest dużo mniejszy niż w przypadku strategii OVO. Analizując dane eksperymentalne możemy zauważyć, że dobre wyniki uzyskujemy już stosując kody o długości $2n$ – $3n$ (gdzie n oznacza liczbę klas).

Podobnie jak w przypadku pozostałych eksperymentów z użyciem klasyfikatora SVM wykorzystano funkcję jądrową RBF, której zastosowanie dawało najlepsze rezultaty. Parametry klasyfikatora dobierano metodą grid-search za pomocą k-kroswalidacji. Zestawienie dobranych parametrów dla różnych baz danych znajduje się tabeli 5.4

4. Efektywne metody selekcji cech

W podrozdziale 1.3 przedstawione zostało zagadnienie selekcji cech. Opisano trzy różne metodologie rozwiązywania tego problemu wraz z przykładowymi algorytmami. Pokazano, że selekcja cech jest bardzo użytecznym narzędziem w procesie klasyfikacji danych. Pozwala ona w szczególności na redukcję wymiarowości przestrzeni cech, co zarówno zwiększa efektywność klasyfikatora jak i pozwala poradzić sobie z problemem "przekleństwa wymiarowości".

W podrozdziale 1.2.1 pokazane zostało, że selekcja cech może być także niezbędna do poprawnego działania klasyfikatora RDA. Dzieje się tak w przypadku problemów, w których wymiarowość wektorów cech jest bardzo duża, a liczba próbek w zbiorze uczącym mała (problem SSS).

Najlepszym sposobem wyboru optymalnego podzbioru cech byłoby sprawdzenie ich wszystkich możliwych kombinacji. Jednak liczba ta rośnie wykładniczo wraz z wymiarowością wektora cech i rzadko kiedy takie sprawdzenie jest wykonalne.

W pewnych wypadkach można zastosować grupowanie cech w większe konglomeraty, które są następnie traktowane jako jedna złożona metacecha. Jeśli uda się odpowiednio zmniejszyć ich liczbę, to możliwe jest sprawdzenie wszystkich kombinacji takich metacech. Podejście takie zostało wykorzystane w pracy [21].

Pozwoliło to w efekcie na zastosowanie klasyfikatora RDA do problemu przewidywania trzeciorzędowej struktury białek. Bez zastosowania selekcji cech wykorzystanie tego klasyfikatora nie byłoby możliwe ze względu na problem SSS (wymiarowość wektorów cech była dużo większa od liczby próbek). Oczywiście podejście takie jest sensowne jedynie wtedy, gdy grupowane cechy są w jakiś sposób powiązane ze sobą. W przypadku białek były to cechy oparte o te same własności biochemiczne.

Grupowanie pozwoliło rozwiązać problem, w tym konkretnym zadaniu klasyfikacji. Jednak oczywiste jest, że w ogólnym przypadku nie będzie ono efektywne. Dlatego też w toku dalszych prac wybór padł na powszechnie znaną metodę selekcji cech SFS czyli przeszukiwanie w przód. Algorytm ten jest bardzo prosty w implementacji i stosunkowo dobrze radzi sobie z redukcją wymiarowości przestrzeni cech.

W przypadku zastosowania tego algorytmu musimy wybrać funkcję J_k oceny jakości wybranego podzbioru cech S_k. W opisanych w pracy eksperymentach jakość podzbioru

cech była oceniana za pomocą wyniku klasyfikacji klasyfikatora RDA (lub SVM) na zbiorze uczącym.

Niestety, jak zostało wspomniane w podrozdziale 1.3, algorytm SFS wymaga wielokrotnego obliczania wartości funkcji kryterialnej. W naszym wypadku będzie to oznaczać konieczność wielokrotnego uruchamiania klasyfikatora. W pesymistycznym przypadku będziemy musieli wykonanać $O(m^2)$ klasyfikacji (gdzie m oznacza wymiarowość pełnego wektora cech). O ile przy małej ilości cech nie stanowi to problemu, o tyle w przypadku wektorów o dużej wymiarowości w praktyce uniemożliwia to jego stosowanie.

W dalszej części tego rozdziału wskazane zostaną metody modyfikacji tego algorytmu, które ograniczają liczbę wymaganych przebiegów klasyfikatora. Pozwoliło to znacznie przyspieszyć jego działanie i umożliwiło praktyczne stosowanie go do problemów wieloklasowych opisanych w niniejszej pracy.

4.1. Szybkie algorytmy F-SFS i F-SBS

Przypomnijmy pokrótce algorytm selekcji cech poprzez przeszukiwanie w przód SFS opisany w podrozdziale 1.3.2. Zaczynamy od pustego podzbioru cech S. Następnie szukamy cechy f_i dla której klasyfikator osiągnie najlepszy rezultat. Ta cecha dodawana jest do zbioru S, a osiągnięty wynik jest zapamiętywany.

Następnie algorytm zaczyna przeglądanie pozostałych cech f_j dla $j \neq i$. Szuka takiej dla której wynik klasyfikacji, z wykorzystaniem zbioru $S \cup \{f_j\}$, jest najlepszy. Jeżeli wynik ten jest lepszy od uzyskanego w poprzednim kroku, to znaleziona cecha jest dodawana do zbioru S. Krok ten jest powtarzany tak długo, jak długo udaje się zmniejszyć błąd klasyfikacji. Oczywiście algorytm zakończy się również, gdy do podzbioru S zostaną dodane wszystkie cechy.

Jak łatwo zauważyć w pierwszym kroku tego algorytmu musimy uruchomić klasyfikator m razy, w kolejnym $m-1$ i w każdym kolejnym o jeden raz mniej. Trzeba jednak pamiętać, że każdy kolejny krok jest bardziej kosztowny. Wynika to z faktu, że klasyfikacja danych odbywa się z użyciem coraz większej liczby cech.

Powoduje to, że w wielu przypadkach efektywne wykorzystanie tego algorytmu nie jest możliwe. Na przykład algorytm SFS (dla klasyfikatora RDA) uruchomiony na testowanych bazach danych pozwolił na uzyskanie wyniku tylko dla bazy MNIST. Zastosowany na pozostałych bazach danych po ponad 24 godzinach obliczeń nie zakończył się.

Z tym problemem możemy sobie poradzić na dwa sposoby. Pierwszy, to ograniczenie maksymalnej liczby kroków algorytmu do pewnej, z góry ustalonej, stałej wielkości. To jednak nadal nie gwarantuje nam, że obliczenia zakończą się w sensownym czasie.

Drugi, to ograniczenie maksymalnego czasu obliczeń. Po każdym kroku algorytm sprawdza czy ten czas nie został przekroczony. Jeśli tak, to przerywa pracę, a jego wynikiem jest znaleziony do tego czasu podzbiór cech S. Oczywiście w obu przypadkach otrzymany wynik będzie gorszy od tego, który uzyskalibyśmy, gdyby proces selekcji cech zakończył się zgodnie z założeniami oryginalnego algorytmu.

Analizując opisaną wyżej procedurę, pod kątem możliwości jej przyspieszenia, możemy zauważyć, że najbardziej kosztowną operacją jest proces klasyfikacji. Spróbujemy zatem skupić się na ograniczeniu liczby wywołań klasyfikatora. Przyjrzyjmy się algorytmowi zaprezentowanemu poniżej:

1. $S \leftarrow \emptyset$
2. $J_{max} \leftarrow 0$
3. $T \leftarrow$ wszystkie cechy, które nie należą do S
4. Weź dowolny $f_i \in T$; policz $J(S \cup \{f_i\})$
5. Jeśli $J(S \cup f_i) > J_{max}$, to $J_{max} = J(S \cup f_i)$; $S \leftarrow S \cup \{f_i\}$; przejdź do 3.
6. $T \leftarrow T \setminus \{f_i\}$
7. Jeśli $T = \emptyset$, to zakończ działanie algorytmu
8. Przejdź do 4.

Wartość J_{max} na początku wynosi 0, a zatem w pierwszym kroku algorytmu klasyfikator zostanie uruchomiony tylko jeden raz. Pomijamy tu teoretycznie możliwy przypadek, gdy ani jedna próbka nie zostanie poprawnie sklasyfikowana. W każdym kolejnym kroku ilość wywołań klasyfikatora będzie też na pewno nie większa niż w oryginalnym algorytmie SFS, ponieważ algorytm ten sprawdza wszystkie możliwe cechy nienależące do S.

Zaproponowana modyfikacja nie zmienia pesymistycznej złożoności algorytmu, ponieważ możliwe jest, że w każdym kroku procedury, najlepszy wynik otrzymamy testując ostatnią z cech należących do zbioru T. Jeżeli jeszcze za każdym razem warunek $J(S \cup f_i) > J_{max}$ będzie spełniony, to wykona się w sumie $O(m^2)$ klasyfikacji.

Rozważmy jednak optymistyczny przypadek. To znaczy taki, w którym każde sprawdzenie pierwszej w kolejności cechy powoduje poprawę wyniku. Wtedy klasyfikator uruchamiany jest w każdym kroku tylko jeden raz. Po znalezieniu optymalnego podzbioru cech wykonamy jeszcze tylko maksimum $m - s$ klasyfikacji, gdzie s będzie ilością cech w znalezionym podzbiorze cech S. Widać zatem, że klasyfikator zostanie uruchomiony $O(m)$ razy.

Oczywiście taki przypadek jest mało prawdopodobny. Jednak wyniki eksperymentów pokazują, że algorytm ten jest wielokrotnie szybszy od swojego odpowiednika (patrz tabela 4.1). W dalszej części tego tekstu będziemy zatem nazywać ten klasyfikator szybkim algorytmem SFS, w skrócie F-SFS (ang. Fast SFS). Analogiczne modyfikacje można

baza danych	ilość cech	algorytm SFS		algorytm F-SFS	
		wynik	czas	wynik	czas
baza MNIST	102	98.9%	11,5h	98,7%	2,5h
baza białka	126	54,2%	> 24h	53,1%	4,5h
baza gesty (A)	256	81,2%	> 24h	83,4%	6,5h
baza ISOLET	617	86,7%	> 24h	92,3%	15,0h
baza ACRS	10000	31,2%	> 24h	47,6%	> 24h

Tablica 4.1. Wyniki osiągnięte za pomocą algorytmów selekcji cech SFS i F-SFS z użyciem klasyfikatora RDA

zastosować do pozostałych algorytmów SBS, SFFS, SFBS znacząco przyśpieszając ich działanie. Pseudokod algorytmu F-SFFS został umieszczony poniżej:

1. $S \leftarrow \emptyset$
2. $J_{max} \leftarrow 0$; $m \leftarrow$ liczba wszystkich cech; $n \leftarrow 0$
3. // przeszukiwanie w przód
4. $T \leftarrow$ wszystkie cechy, które nie należą do S
5. Weź dowolny $f_i \in T$; policz $J(S \cup \{f_i\})$
6. Jeśli $J(S \cup f_i) > J_{max}$, to $J_{max} = J(S \cup f_i)$; $S \leftarrow S \cup \{f_i\}$; przejdź do 11
7. $T \leftarrow T \setminus \{f_i\}$
8. Jeśli $T = \emptyset$, to zakończ działanie algorytmu
9. przejdź do 5
10. // przeszukiwanie w tył
11. $T \leftarrow S$
12. Weź dowolny $f_i \in T$; policz $J(S \setminus \{f_i\})$
13. Jeśli $J(S \setminus f_i) > J_{max}$, to $J_{max} = J(S \setminus f_i)$; $S \leftarrow S \setminus \{f_i\}$; przejdź do 4
14. $T \leftarrow T \setminus \{f_i\}$
15. Jeśli $T = \emptyset$, to przejdź do 4

Jak można się spodziewać przyśpieszenie działania tych algorytmów okupione zostało kosztem pogorszenia wyniku osiąganego przez klasyfikator końcowy, pracujący na znalezionym podzbiorze cech S. Jednak wynik ten będzie gorszy jedynie wtedy, gdy jesteśmy w stanie zakończyć obliczenia za pomocą oryginalnego algorytmu. Jeśli obliczenia zostaną przerwane ze względu na maksymalny limit czasu, to wyniki uzyskane za pomocą zmodyfikowanego algorytmu będą zwykle lepsze.

Trzeba jednak zauważyć, że szybki algorytm F-SFS tak jak i jego odpowiednik SFS cierpi na efekt gniazda. Dodatkowo wynik jego działania zmienia się w zależności od kolejności w jakiej testowane są poszczególne cechy. Na przykład łatwo można zauważyć, że algorytm

ten zawsze wybierze do szukanego podzbioru pierwszą cechę, którą będzie testował. Wynika to z faktu, że na starcie algorytmu przyjmujemy $J_{max} = 0$ jako punkt odniesienia.

To, że wynik działania tego algorytmu zależy od kolejności wyboru cech możemy wykorzystać, w celu poprawienia jego skuteczności. Zamiast wybierać cechy do testowania losowo możemy je uporządkować w pewnej, z góry ustalonej, kolejności. Jeśli uda nam się ją ustalić tak, aby zawsze rozpoczynać sprawdzanie od cech najbardziej wartościowych, to potencjalnie powinniśmy uzyskać poprawę wyniku. Powinniśmy także uzyskać przyspieszenie działania algorytmu, ponieważ w każdym kroku będziemy potrzebować mniej klasyfikacji, aby poprawić znalezioną do tej pory wartość J_{max}.

Problem jednak polega na tym, że bardzo trudno jest znaleźć dobre kryterium wartościowania poszczególnych cech. Pierwsze co się narzuca, to wykorzystanie jednego kryteriów opisywanych przy okazji omawiania metod rankingowych.

Za pomocą znanych ze statystyki testów permutacyjnych możemy stwierdzić jak bardzo dwie cechy są ze sobą skorelowane [106]. Możemy zatem obliczyć te współczynniki korelacji i stworzyć listę rankingową cech. Tego typu algorytm, reprezentujący metody rankingowe, został zaproponowany w pracy [125]. Podobna metoda oparta na wzajemnej korelacji cech została opisana w [64]. Jednak stworzoną w ten sposób listę rankingową możemy wykorzystać w nieco inny sposób. Sprawdzając kolejne cechy algorytm F-SFS może brać je w kolejności ustalonej przez ten ranking.

W ten sposób jego słabość, czyli tendencja do umieszczania w szukanym podzbiorze cech, które są testowane jako pierwsze, zostanie zneutralizowana. Więcej, może stać się nawet zaletą, ponieważ po tej modyfikacji algorytm ten będzie miał tendencję do wybierania cech, które są stosunkowo słabo ze sobą skorelowane. Takie podejście sprawdza się w praktyce. Zostało ono zaimplementowane i wykorzystane w eksperymentach na bazie danych białek. Wyniki tych eksperymentów zostały opublikowane w pracy [27].

4.2. Metoda oparta na podobieństwie klas CS-SFS

Jak już zostało wcześniej wspomniane, aby uniknąć problemu SSS przy korzystaniu z klasyfikatora RDA, konieczna była redukcja wymiarowości wektora cech. W literaturze jest opisanych wiele metod selekcji cech. Próby zastosowania niektórych z tych metod, w tym metod opisanych w [63], czy też w [90] skończyły się niepowodzeniem. Wszystkie te metody okazały się zbyt kosztowne obliczeniowo. Pierwszym udanym podejściem było skorzystanie z opisanych w poprzednim podrozdziale algorytmów F-SFS i F-SFFS

W poprzednim podrozdziale, analizując algorytm F-SFS stwierdziliśmy, że najlepiej byłoby, aby zaczynał on testowanie cech od tych najbardziej wartościowych. Niestety niełatwo jest odpowiedzieć na pytanie, które cechy są wartościowe, a które mało istotne. Próbowaliśmy

wykorzystać do tego listę rankingową utworzoną na podstawie współczynnika korelacji cech. Jednak zasadniczą wadą takiego podejścia jest to, że nie bierze ono pod uwagę współzależności cech.

Już w przeglądowej pracy Kohaviego i Johna [83] pokazano, że cechy istotne (ang. relevant) nie muszą się znaleźć w optymalnym podzbiorze cech. Pokazano również, że w tym podzbiorze mogą się z kolei znaleźć cechy nieistotne (ang. irrelevant). Dwie cechy nieistotne mogą, w połączeniu, wpływać na polepszenie wyniku, podczas gdy połączenie dwóch cech istotnych może go nie poprawić, ale nawet czasami spowodować jego pogorszenie.

Zastanówmy się zatem nad nieco innym tworzeniem list rankingowych. Zacznijmy od tego jak powinien wyglądać optymalny wektor cech? Dobrze byłoby, aby wszystkie próbki należące do tych samych klas, były reprezentowane przez wektory cech, które są do siebie maksymalnie podobne.

Spróbujmy zatem znaleźć miarę, która oceniałaby to podobieństwo. Jako, że z miary tej korzystalibyśmy w każdym kroku algorytmu, powinna być ona prosta i łatwa do obliczania. Dobrym wyborem wydaje się średnia odległość euklidesowa pomiędzy wszystkimi wektorami cech reprezentującymi daną klasę, to jest:

$$Dist(k,S) = \frac{\sum_{i=1}^{N_k}\sum_{j=i+1}^{N_k}\|x_i - x_j\|^2}{N_k * (N_k - 1)/2} \tag{4.1}$$

gdzie k jest indeksem klasy, N_k ilością próbek klasy k, a x_i, x_j są wektorami cech reprezentującymi i-tą i j-tą próbkę należącą do klasy k, zaś S jest podzbiorem cech.

Następnie możemy obliczyć średnią wartość tej odległości $Dist(k,S)$ dla wszystkich klas, tak jak to zdefiniowano poniżej:

$$AvgDist(S) = \frac{\sum_{k=1}^{N} Dist(k,S)}{N}, \tag{4.2}$$

gdzie S jest podzbiorem cech, N ilością klas w zbiorze, a $Dist(k,S)$ odległością zdefiniowaną we wzorze 4.1.

Łatwo można zauważyć, że jeśli wartość $AvgDist(S)$ jest mniejsza, to wektory cech reprezentujące tę samą klasę leżą bliżej siebie w przestrzeni cech (w sensie odległości euklidesowej). Oczywiście nie jesteśmy w stanie obliczyć tej wartości dla wszystkich możliwych podzbiorów cech. Możemy jednak obliczać ją w kolejnych krokach algorytmu i wykorzystywać do wyboru kolejności testowania cech. W ten sposób otrzymamy algorytm selekcji cech oparty o podobieństwo klas CS-SFS (ang. Class Similarity based SFS).

Zaczynamy od pustego podzbioru cech S. Następnie wyliczamy średnią odległość dla każdego jednoelementowego podzbioru $AvgDist(\{f_i\})$ tworząc listę tych wartości oraz

odpowiadającą im listę cech. Lista ta jest następnie sortowana rosnąco. W kolejnym kroku sprawdzamy wartość współczynnika klasyfikacji dla cechy powiązanej z pierwszym elementem listy. Wynik zapamiętujemy, a cechę dodajemy do S.

W następnym kroku algorytmu obliczamy średnią odległość dla dwuelementowych podzbiorów cech $AvgDist(S \cup \{f_i\})$. To znaczy podzbiorów składających się ze zbioru S, do którego została dodana jedna cecha. Znów tworzymy listę tych wartości i odpowiadającą im listę cech. Sortujemy ją, a następnie wykorzystujemy przy szukaniu cechy poprawiającej wynik uzyskany w poprzednim kroku algorytmu. Po znalezieniu jej i dodaniu do podzbioru S przechodzimy do obliczania średnich odległości dla zbiorów trójelementowych. Algorytm kończy się, gdy w kolejnym kroku nie uda nam się poprawić wyniku. Pseudokod tego algorytmu przedstawiony został poniżej:

1. $S \leftarrow \emptyset$; $J_{max} \leftarrow 0$
2. $T \leftarrow$ wszystkie cechy, które nie należą do S
3. Oblicz $AvgDist(S \cup \{f_i\})$ dla każdego $f_i \in T$
4. Stwórz listę rankingową cech f_i względem rosnącej odległości $AvgDist(S \cup \{f_i\})$
5. Weź kolejną cechę f_i z listy rankingowej i policz $J(S \cup \{f_i\})$
6. Jeśli $J(S \cup f_i) > J_{max}$, to $J_{max} = J(S \cup f_i)$; $S \leftarrow S \cup \{f_i\}$; przejdź do 2.
7. $T \leftarrow T \setminus \{f_i\}$
8. Jeśli $T = \emptyset$, to zakończ działanie algorytmu
9. Przejdź do 5.

Powyższe podejście ma dwie podstawowe zalety. Wybierane cechy są potencjalnie bardziej wartościowe dzięki czemu jest szansa, że szybciej znajdziemy cechę, która poprawi rezultat osiągnięty w poprzednim kroku algorytmu. Tym samym ograniczamy liczbę klasyfikacji, które są najbardziej kosztowne obliczeniowo. Jest także duża szansa, że wybierając te cechy uda nam się znaleźć bardziej optymalny podzbiór cech. Wyniki eksperymentów przedstawione w podrozdziale 4.4 potwierdzają trafność tego podejścia.

4.3. Zmodyfikowana metoda oparta na podobieństwie klas CSO-SFS

Analizując dokładniej metodę CS-SFS możemy zauważyć, że samo rozpatrywanie wyłącznie podobieństwa klas względem pewnych wektorów cech powoduje, że algorytm ten ma pewną wadę. Otóż miara zdefiniowana wzorem 4.1 nic nie mówi nam o tym, w jaki sposób dany podzbiór cech pozwala odróżnić poszczególne klasy od siebie. Załóżmy przykładowo, że pewna cecha f będzie przyjmowała takie same wartości dla wszystkich próbek. Według zaproponowanej miary, taka cecha zostanie umieszczona na pierwszym miejscu w rankingu, ponieważ $AvgDist(\{f\}) = 0$.

Oczywiście taka cecha zostanie prawdopodobnie odrzucona na etapie jej testowania za pomocą klasyfikatora, jednak przykład ten pokazuje, że opracowany ranking może preferować cechy, które nie są do końca przez nas pożądane.

Aby uniknąć takiej sytuacji możemy zdefiniować miarę podobieństwa do pozostałych klas, która sprawdzałaby na ile próbki danej klasy są podobne do próbek reprezentujących klasy pozostałe. Możemy to zrobić, tak jak to przedstawiono poniżej:

$$Dist_{others}(k, S) = \frac{\sum_{i=1}^{N_k} \sum_{j \neq i}^{N} \|x_i - x_j\|^2}{N_k * (N - N_k)}, \tag{4.3}$$

i podobnie jak poprzednio:

$$AvgDist_{others}(S) = \frac{\sum_{k=1}^{N} Dist_{others}(k, S)}{N}, \tag{4.4}$$

Tym razem za najlepszy podzbiór uznamy taki, który osiąga maksymalną wartość $AvgDist_{others}(S)$. Czyli taki wektor cech, względem którego próbki każdej klasy są jak najmniej podobne do próbek z pozostałych klas.

Nie chcielibyśmy jednak porzucać całkowicie poprzedniego kryterium, które również może być użyteczne. Najlepiej byłoby wybierać podzbiory S takie, które minimalizowałby odległość $AvgDist(S)$ i jednocześnie maksymalizowały odległość $AvgDist_{others}(S)$. To znaczy chcielibyśmy, aby próbki tej samej klasy były do siebie maksymalnie podobne, ale jednocześnie, próbki pochodzące z różnych klas były jak najmniej podobne. Niestety zwykle nie jest możliwe znalezienie podzbioru S spełniającego oba te kryteria równocześnie. Musimy zatem znaleźć jakiś kompromis.

W algorytmie bazującym na podobieństwie i niepodobieństwie klas CSO-SFS (ang. Class Similarity Others based SFS) [26] tworzymy dwie posortowane listy rankingowe cech. Pierwsza lista sortowana jest rosnąco względem $AvgDist(S)$, a druga lista malejąco względem $AvgDist_{others}(S)$. Wysoka pozycja na pierwszej liście świadczy o tym, że względem danego podzbioru cech S, próbki reprezentujące tę samą klasę, są do siebie bardzo podobne. Wysoka pozycja na drugiej liście oznacza, że względem danego podzbioru cech S, próbki reprezentujące różne klasy, są do siebie mało podobne. A zatem powinniśmy preferować te cechy, które znajdują się na wysokich pozycjach na obu tych listach jednocześnie.

Aby to uzyskać, te dwie listy łączone są w trzecią. O miejscu cechy na tej liście decyduje suma pozycji na liście numer jeden i dwa. Dzięki temu cecha przyjmująca takie same wartości dla wszystkich klas zajmie wprawdzie pierwsze miejsce na liście numer jeden, ale też i ostatnie

miejsce na liście numer dwa. To oznacza, że jej pozycja na liście numer trzy nie będzie zbyt wysoka. Pseudokod tego algorytmu jest przedstawiony poniżej:

1. $S \leftarrow \emptyset$, $J_{max} \leftarrow 0$
2. $T \leftarrow$ wszystkie cechy, które nie należą do S
3. Oblicz $AvgDist(S \cup \{f_i\})$ dla każdego $f_i \in T$
4. Oblicz $AvgDist_{others}(S \cup \{f_i\})$ dla każdego $f_i \in T$
5. Stwórz listę rankingową L_1 cech f_i względem rosnącej odległości $AvgDist(S \cup \{f_i\})$
6. Stwórz listę rankingową L_2 cech f_i względem malejącej odległości $AvgDist_{others}(S \cup \{f_i\})$
7. Stwórz listę rankingową L_3 cech f_i sumując ich miejsca na listach L_1 i L_2
8. Weź kolejną cechę f_i z listy rankingowej L_3 i policz $J(S \cup \{f_i\})$
9. Jeśli $J(S \cup f_i) > J_{max}$, to $J_{max} = J(S \cup f_i)$; $S \leftarrow S \cup \{f_i\}$; przejdź do 2.
10. $T \leftarrow T \setminus \{f_i\}$
11. Jeśli $T = \emptyset$, to zakończ działanie algorytmu
12. Przejdź do 8.

W ten sposób eliminujemy istotną wadę przedstawionego wcześniej algorytmu CS-SFS. Pozwala to na dalsze poprawienie uzyskanych wyników. Niestety możemy również zauważyć, że odbywa się poprzez zwiększenie czasu działania algorytmu. Konstrukcja odległości $Dist_{others}(f, S)$ powoduje, że jest ona dość kosztowna obliczeniowo, zwłaszcza gdy mamy dużą liczbę próbek reprezentujących klasy.

W kolejnym podrozdziale przedstawione zostaną wyniki eksperymentów z zastosowaniem omówionych metod selekcji cech. Metody te zostaną porównane na kilku bazach danych zarówno pod kątem szybkości działania jak i uzyskiwanych wyników.

4.4. Wyniki eksperymentów

W eksperymentach opisanych w poniższym podrozdziale zostały wykorzystane wszystkie bazy danych opisane w rozdziale 2. Przyjęto następujące założenia:

— Testowane będą algorytmy SFS i SFFS oraz ich modyfikacje zaproponowane w niniejszym rozdziale.
— Aby uniknąć obciążenia wyniku (dobrania zbioru cech, do konkretnego podzbioru testowego) zastosowano procedurę warstwowej k-kroswalidacji. We wszystkich testach przyjęto wartość $k = 6$.
— Został ograniczony czas działania algorytmów. Po każdorazowym dodaniu nowej cechy do szukanego podzbioru, sprawdzany był czas, który upłynął od uruchomienia programu. Jeśli przekraczał on 24 godziny, to jego działanie było przerywane. Jako wynik przyjmowano otrzymany tej pory podzbiór.

Czasy podane w tabelach 4.1 i 4.2 mierzone były na były na komputerze klasy PC z dwurdzeniowym procesorem Athlon X2 250 wyposażonym w 8GB pamięci RAM. Algorytmy zostały zaimplementowane w języku C++.

W tabeli 4.1 (zamieszczonej w podrozdziale 4.1) oraz w tabeli 4.2 zestawione są czasy działania algorytmów SFS i SFFS z czasami działania ich uproszczonych szybkich wersji. Możemy zauważyć, że korzystając ze zmodyfikowanych algorytmów jesteśmy w stanie otrzymać w założonym czasie (poniżej 24 godzin) wyniki dla prawie wszystkich baz danych.

Ze względu na przyjęte ograniczenie czasowe nie można dokładnie oszacować o ile zaproponowana modyfikacja przyspiesza działanie algorytmów. Przeglądając wyniki zamieszczone w tabelach możemy jednak zauważyć, że dla testowanych baz danych, są one wielokrotnie szybsze od swoich odpowiedników.

Moglibyśmy zrezygnować z narzuconego ograniczenia czasowego, aby zmierzyć czasy działania algorytmów SFS i SFFS. Jednak niewiele wniosłoby to do wyników badań. Jak zostało pokazane w podrozdziale 4.1 pesymistyczna złożoność algorytmu F-SFS jest taka sama jak algorytmu SFS. Co więcej czas działania algorytmu F-SFS zależy istotnie od kolejności w jakiej testowane są kolejne cechy.

baza danych	ilość cech	algorytm SFFS		algorytm F-SFFS	
		wynik	czas	wynik	czas
baza MNIST	102	99,0%	15,5h	99,0%	3,5h
baza białka	126	53,3%	> 24h	54,6%	5,5h
baza gesty (A)	256	82,3%	> 24h	86,2%	9,5h
baza ISOLET	617	90,6%	> 24h	93,2%	22,0h
baza ACRS	10000	32,2%	> 24h	48,1%	> 24h

Tablica 4.2. Wyniki osiągnięte za pomocą algorytmów selekcji cech SFFS i F-SFFS z użyciem klasyfikatora RDA

Załóżmy przez chwilę, że jesteśmy w stanie uporządkować cechy od najbardziej istotnych do najmniej istotnych. Przy czym są one tak uporządkowane, że biorąc kolejne podzbiory pierwszych s cech, uzyskujemy coraz lepsze współczynniki klasyfikacji. Łatwo zauważyć, że w takiej sytuacji algorytm F-SFS będzie działał w czasie liniowym $O(m)$ (gdzie m jest wymiarowością wektora cech), ponieważ w każdym kroku algorytmu klasyfikator będzie uruchamiany tylko jeden raz.

Każde zaburzenie tego porządku spowoduje, że liczba klasyfikacji w poszczególnych krokach algorytmu będzie rosła, wydłużając tym (zwykle) czas działania programu. Warto zauważyć, że może też dojść do skrócenia czasu obliczeń w sytuacji, gdy w wyniku działania algorytmu, zostanie znaleziony podzbiór o mniejszej ilości cech.

Próbą znalezienia takiego porządku jest tworzenie rankingów cech. Przeglądając tabele 4.1 i 4.2 możemy wnioskować, że jest to próba udana. Algorytmy PF-SFS i CS-SFS wykonują się

w wielu wypadkach szybciej od algorytmu F-SFS. Oznacza to, że w ich przypadku wykonuje się średnio mniej klasyfikacji.

W przypadku algorytmu CSO-SFS nie udaje się zauważyć takiej zależoności. Jednak łatwo to wytłumaczyć. Obliczanie odległości $Dist_{others}(f, S)$ jest bardzo kosztowne obliczeniowo, zwłaszcza gdy rośnie liczba cech w zbiorze S. Wypływa stąd wniosek, że szukając odpowiedniej miary, za pomocą której będziemy tworzyć ranking, musimy zwracać uwagę, aby jej obliczanie nie było zbyt kosztowne.

Analizując sposób działania algorytmów F-SFS i F-SFFS możemy się spodziewać, że im większa będzie wymiarowość wektora cech, tym większy będzie wzrost szybkości algorytmu w stosunku do jego niezmodyfikowanej wersji.

Wynika to z faktu, że zmodyfikowane algorytmy *"zadowalają się"* pierwszą cechą, która poprawia wynik klasyfikacji, podczas gdy algorytmy SFS i SFFS sprawdzają także wszystkie pozostałe cechy. To zaś oznacza, że w każdym kroku *"oszczędzamy"*, tym większą liczbę uruchomień klasyfikatora, z im większą liczbą cech mamy do czynienia.

Nie możemy ocenić jak dużo uruchomień w ten sposób *"zaoszczędzimy"*, ponieważ jak zostało wspomniane wcześniej, zależy to od kolejności w jakiej testowane są cechy. Jednak możemy zauważyć, najwięcej zyskujemy w pierwszych krokach algorytmu, gdy stosunkowo łatwo jest poprawić wynik klasyfikacji. Z kolei w ostatnim kroku algorytmu nie zyskujemy nic, ponieważ musimy sprawdzić wszystkie cechy, aby przekonać się, że dodanie żadnej z nich nie powoduje już poprawy wyniku (jest to warunek stopu algorytmu).

metoda selekcji cech	współczynnik klasyfikacji osiągnięty na zbiorze			
	białek (126)	gestów A (256)	cyfr (102)	ISOLET (617)
SFS	54,2% (27)	81,2% (65)	98,9% (78)	86,7% (112)
F-SFS	53,1% (22)	83,4% (62)	98,7% (72)	92,3% (132)
PF-SFS	53,7% (23)	84,1% (61)	98,7% (82)	93,1% (125)
CS-SFS	55,0% (28)	86,2% (65)	98,7% (81)	93,7% (123)
CSO-SFS	55,8% (32)	87,6% (68)	98,9% (77)	94,6% (127)

Tablica 4.3. Współczynniki uzyskane za pomocą różnych metod selekcji cech opartych o algorytm SFS przez klasyfikator RDA

Kolejna tabela 4.3 przedstawia współczynniki klasyfikacji uzyskane na czterech bazach danych. Wyniki na bazie danych ACRS nie zostały zamieszczone, ponieważ żadnen z porównywanych algorytmów nie zakończył się (wszystkie zostały przerwane ze względu na ograniczenie czasowe). W tabeli, obok wyniku procentowego, została podana w nawiasach wymiarowość znalezionego wektora cech.

Analizując tabele 4.3 i 4.4 możemy zaobserwować, że zaproponowane modyfikacje pozwalają nie tylko na skrócenie czasu obliczeń, ale także na poprawienie osiąganego współczynnika klasyfikacji.

metoda	współczynnik klasyfikacji osiągnięty na zbiorze			
selekcji cech	białek (126)	gestów A (256)	cyfr (102)	ISOLET (617)
SFFS	54.6% (29)	82,3% (68)	98,9% (79)	90,6% (121)
F-SFFS	52.7% (25)	86,2% (61)	98,7% (75)	93,2% (127)
PF-SFFS	53.3% (32)	86,4% (69)	98,7% (84)	93,5% (125)
CS-SFFS	55.2% (40)	87,2% (72)	98,8% (88)	94,1% (129)
CSO-SFFS	56.1% (44)	88,0% (70)	98,9% (84)	95,2% (133)

Tablica 4.4. Współczynniki uzyskane za pomocą różnych metod selekcji cech opartych o algorytm SFFS przez klasyfikator RDA

Oczywiście na pewno algorytmy SFS i SFFS uzyskałyby nieco lepsze wyniki, gdyby ich działanie nie zostało przerwane. Jednak porównując otrzymane rezultaty z uzyskanymi przez innych autorów (zamieszczonymi w rozdziale 2), możemy przypuszczać, że poprawa ta nie byłaby znacząca. Jednocześnie wykorzystanie znalezionych podzbiorów cech w klasyfikatorze złożonym (opisanym w następnym rozdziale) pozwoliło poprawić rezultaty znane z literatury.

metoda	Czas działania algorytmu na zbiorze			
selekcji cech	białek (126)	gestów A (256)	cyfr (102)	ISOLET (617)
SFS	> 24h	> 24h	11,5h	> 24h
F-SFS	4,5h	6,5h	2,5h	15,0h
PF-SFS	4,0h	6,5h	2,0h	13,5h
CS-SFS	4,5h	7,0h	2,5h	14,5h
CSO-SFS	5,5h	8,5h	3,5h	21,0h

Tablica 4.5. Czasy działania algorytmów selekcji cech, dla klasyfikatora RDA

W tabeli 4.6 zaprezentowane zostały wyniki działania zaproponowanych algorytmów z klasyfikatorem SVM. Również w tym przypadku udało się poprawić osiągane wyniki, choć poprawa ta była nieco mniejsza. Można też zauważyć, że znalezione podzbiory cech są bardziej liczne (zwłaszcza w przypadku zbioru ISOLET). Zapewne wynika to z faktu, że klasyfikator ten dobrze radzi sobie z dużą ilością cech.

metoda	współczynnik klasyfikacji osiągnięty na zbiorze			
selekcji cech	białek (126)	gestów A (256)	cyfr (102)	ISOLET (617)
SFS	57,5% (43)	78,6% (73)	98,7% (91)	95,2% (193)
F-SFS	56,1% (38)	73,2% (68)	98,5% (86)	93,0% (211)
PF-SFS	56,4% (36)	75,2% (69)	98,8% (85)	93,4% (163)
CS-SFS	58,2% (41)	78,9% (72)	98,9% (89)	95,3% (154)
CSO-SFS	58,7% (49)	81,3% (72)	99,2% (87)	96,9% (147)

Tablica 4.6. Współczynniki uzyskane za pomocą różnych metod selekcji cech przez klasyfikator SVM

Podsumowując, zastosowanie zaproponowanych metod selekcji cech pozwoliło na bardzo znaczące skrócenie czasu obliczeń. W przypadku baz danych białek, gestów i ISOLET

pozwoliło zaś na praktyczne zastosowanie klasyfikatora RDA. Wyniki osiągane przez te algorytmy są porównywalne, a nawet w wielu wypadkach nieco lepsze od uzyskanych za pomocą oryginalnych metod, co przy dużo niższym koszcie obliczeniowym stanowi ich niezaprzeczalną zaletę.

5. Hybrydowy klasyfikator SVM-RDA

W podrozdziale 1.2.3 przedstawione zostały podstawowe różnice pomiędzy klasyfikatorami generatywnymi i dyskryminatywnymi. Pokazano, że reprezentują one odmienne podejścia do procesu uczenia się. Powoduje to, że w pewnych warunkach, klasyfikatory dyskryminatywne osiągają lepsze wyniki. Na przykład w sytuacji, gdy mamy wystarczająco dużo próbek reprezentujących klasy.

Z kolei wyniki klasyfikatorów generatywnych są bardziej stabilne, gdy liczba próbek jest mała. Są one także bardziej odporne na próbki odstające (ang. outliers), podczas gdy klasyfikatory dyskryminatywne są na nie bardziej podatne ze względu na to, że ich regiony decyzyjne są zwykle otwarte [61]. Szerszą dyskusję na temat porównania metod dyskryminatywnych i generatywnych można znaleźć w pracy Liu i Fujisawy [96].

W przypadku problemów wieloklasowych, zwłaszcza o dużej ilości klas, połączenie tych dwóch typów klasyfikatorów wydaje się być szczególnie atrakcyjne. W takich problemach często dysponujemy stosunkowo małą liczbą próbek reprezentujących poszczególne klasy. Przykładowo w zbiorze ACRS znajduje się tylko po trzydzieści próbek na klasę, a w zbiorze białek pojawiają się nawet klasy reprezentowane przez zaledwie 11 próbek.

W niniejszym rozdziale zostanie opisane połączenie dwóch klasyfikatorów: generatywnego RDA i dyskryminatywnego SVM. Pokazane zostanie, że stworzony w ten sposób klasyfikator hybrydowy osiąga lepsze wyniki. Z pomocą testu t-studenta wykazane zostanie, że poprawa ta jest statystycznie znacząca.

5.1. Opis klasyfikatora

Przypomnijmy, że aby skorzystać ze strategii *"jeden przeciw jednemu"* OVO musimy skonstruować $N * (N-1)/2$ klasyfikatorów binarnych dla wszystkich możliwych par klas. Następnie wszystkie próbki są przez te klasyfikatory oceniane i każdy z nich przydziela daną próbkę do pewnej klasy, oddając na nią swój głos. W ten sposób powstaje tablica głosów wszystkich klasyfikatorów, a próbkę przydzielamy do klasy, która uzyskała najwięcej głosów. Przykładowy fragment takiej tablicy został przedstawiony w tabeli 5.1. W poszczególnych kolumnach umieszczone są ilości głosów (w kolejności malejącej), które padły na klasę, której etykieta znajduje się w nawiasach.

próbka	klasa	ilość głosów przypadająca na klasę					
		1	2	3	4	5	6
1	1	21(1)	20(15)	19(8)	18(9)	16(6)	15(16)
2	1	21(1)	20(6)	19(8)	18(7)	17(13)	16(20)
3	1	21(1)	20(9)	19(6)	18(15)	17(8)	17(11)
4	2	21(2)	20(7)	19(12)	18(15)	17(5)	16(17)
5	2	21(7)	20(12)	19(2)	18(13)	17(14)	16(13)
6	2	21(2)	20(5)	19(14)	18(22)	17(12)	15(13)
7	3	21(3)	20(17)	19(11)	18(19)	17(21)	16(12)
8	3	21(3)	19(4)	18(6)	18(11)	17(18)	16(17)
9	3	21(11)	20(17)	19(4)	18(3)	17(6)	16(8)

Tablica 5.1. Fragment tablicy głosów dla problemu gestów (wariant A)

Jak łatwo zauważyć pierwsze trzy próbki uzyskały największą liczbę głosów (21) dla klasy o etykiecie 1, do której w rzeczywistości należą. Jednak druga w rankingu klasa uzyskała bardzo podobny rezultat. Analizując tabelę 5.1 można zauważyć, że różnice w ilości głosów są nieznaczne. Zwykle kolejne w rankingu klasy różni zaledwie jeden głos. Warto zastanowić się dlaczego ta różnica jest taka mała.

Rozważmy przykładowo klasyfikator, który został wytrenowany na próbkach reprezentujących klasy 1 i 2 (nazwijmy go klasyfikatorem "1 vs 2"). Jest on kompetentny, gdy na jego wejście trafia próbka należąca do jednej z tych dwóch klas. Jednak ten klasyfikator musi podjąć decyzję dla pozostałych próbek.

Oznacza to, że nawet jeśli próbka nie należy do klasy 1 ani do klasy 2, to musi on zagłosować na jedną z nich (ten schemat nie przewiduje możliwości odrzucenia próbki). Te głosy klasyfikatora "1 vs 2" są niewiarygodne. Gdybyśmy wiedzieli, że aktualnie testowana próbka nie należy do żadnej z tych dwóch klas moglibyśmy ten głos po prostu zignorować.

Problem jednak tkwi w tym, że klasyfikator "1 vs 2" nie wie z jaką próbką ma do czynienia w danej chwili. Nawet więcej, nie wie on o istnieniu jakichkolwiek innych klas poza klasami 1 i 2, ponieważ został wytrenowany wyłącznie na próbkach należących do tych dwóch klas.

Załóżmy jednak, że na testowanym właśnie zbiorze uruchomiliśmy wcześniej klasyfikator RDA. Możemy wtedy skorzystać z wartości jego funkcji dyskryminacyjnej. Warto tutaj zauważyć, że klasyfikator RDA wie o istnieniu wszystkich klas.

Jak pamiętamy, z podrozdziału 1.2.1, im wyższa jest wartość funkcji dyskryminacyjnej klasyfikatora RDA $d_i(x)$ dla danej próbki x, tym mniejsze prawdopodobieństwo, że należy ona do i-tej klasy. Zatem możemy skorzystać z tej wartości i przydzielić każdemu z głosów klasyfikatora binarnego SVM wagę opartą o tę wartość.

Załóżmy, że binarny klasyfikator SVM, wytrenowany dla klas i-tej i j-tej, klasyfikuje próbkę opisywaną przez wektor cech x. Załóżmy, że klasyfikator ten głosuje na klasę i (czyli przydziela próbce etykietę i). Oznacza to, że im mniejsza różnica pomiędzy wartością

funkcji dyskryminacyjnej $d_i(x)$, a minimalną wartością tej funkcji $d_{min}(x)$, tym większa jest wiarygodność tego głosu. Formalnie możemy zdefiniować tę wagę jako:

$$w_i(x) = 1 - \frac{d_i(x) - d_{min}(x)}{d_{min}(x)}, \tag{5.1}$$

gdzie $d_i(x)$ jest wartością funkcji dyskryminacyjnej klasyfikatora RDA dla wektora x oraz i-tej klasy, $d_{min}(x) = \min\{d_k(x)\}$, $k = 1, 2, \ldots, N$, a N jest liczbą klas.

W ten sposób każdy głos klasyfikatora binarnego otrzymuje swoją wagę. Maksymalna wartość tej wagi wynosi 1 dla głosu, który przydziela próbkę reprezentowaną przez wektor cech x do klasy o minimalnej wartości funkcji dyskryminacyjnej $d_i(x) = d_{min}(x)$. Możemy również zauważyć, że wartość tej wagi będzie równa 0, gdy wartość funkcji dyskryminacyjnej będzie dwa razy większa od wartości minimalnej $d_i(x) = 2 * d_{min}(x)$. Jeśli wartość funkcji dyskryminacyjnej będzie jeszcze większa, to waga stanie się ujemna.

Można uniknąć ujemnych wartości wag zakładając, że minimalna wartość wagi nie może być mniejsza od zera. Jednak wyniki eksperymentów pokazały, że nie wpływa to w żadnym stopniu na wynik klasyfikacji. Łatwo to wyjaśnić. Waga, a wraz z nią głos klasyfikatora staje się ujemna, gdy wartość $d_i(x)$ będzie ponad dwa razy większa od wartości minimalnej, a to oznacza, że prawdopodobieństwo, aby w rzeczywistości należała ona do klasy i jest bardzo małe.

Analizując zachowanie się funkcji dyskryminacyjnej dla klasyfikatora RDA możemy zauważyć, że zależność pomiędzy wzrostem jej wartości, a prawdopodobieństwem przynależności próbki do danej klasy nie jest liniowa. Dlatego też w pracy [27] zdecydowano się użyć nieco zmodyfikowanych wag. We wzorze 5.1 została dodana trzecia potęga, dzięki czemu wpływ wartości $d_i(x)$ na wagę głosu jest większy. Użycie trzeciej potęgi jest wygodne, ponieważ nie musimy uwzględniać sytuacji, w której waga staje się ujemna (w przypadku drugiej potęgi taka waga zmieniłaby się na dodatnią).

$$w_i(x) = \left(1 - \frac{d_i(x) - d_{min}(x)}{d_{min}(x)}\right)^3, \tag{5.2}$$

Użyteczność tak zdefiniowanych wag możemy prześledzić na następującym przykładzie. Weźmy binarny klasyfikator SVM, który został wytrenowany na próbkach należących do klas i-tej i j-tej. Załóżmy teraz, że ma on sklasyfikować pewną próbkę reprezentowaną przez wektor x. Jeśli w rzeczywistości należy ona do klasy o etykiecie i lub j, to zapewne jedna z wartości $d_i(x)$ lub $d_j(x)$ będzie minimalna lub zbliżona do minimalnej. Zatem głos oddany na jedną z tych klas będzie miał wagę 1 lub zbliżoną do 1.

Jeśli jednak klasyfikowana próbka należy do jednej z pozostałych klas, to prawdopodobnie wartości $d_i(x)$ i $d_j(x)$ będą dużo wyższe. Oczywiście może się zdarzyć, że któraś z wartości będzie zbliżona do minimalnej lub nawet będzie minimalna (bo klasyfikator RDA też nie jest nieomylny). Jednak system wag pozwoli nam *"pozbyć się"* większości niechcianych głosów. Przykładowe wektory wag dla kilku próbek zostały przedstawione w tabeli 5.2.

klasa	wagi dla poszczególnych klas							
	1	2	3	4	5	6	7	8
1	1,000	0,632	0,817	0,211	-2,321	0,821	0,943	0,275
2	0,812	1,000	0,638	0,210	0,012	-1,341	0,863	-1,532
3	0,134	0,911	1,000	0,354	0,739	0,741	-2,581	0,953
4	1,000	0,035	0,341	0,942	0,553	-0,532	-3,183	-1,234
5	-1,196	0,932	0,892	0,113	0,964	1,000	-1,384	0,426

Tablica 5.2. Wagi dla kilku przykładowych próbek

Klasyfikacja próbki za pomocą klasyfikatora SVM-RDA odbywa się zatem w następujących krokach:

— Próbka opisana wektorem cech x trafia na wejście klasyfikatora RDA, który na wyjściu daje nam wektor wartości funkcji dyskryminacyjnych $d_i(x)$ dla $i = 1, 2, \ldots, N$, gdzie N oznacza ilość klas.

— Na podstawie wektora wartości funkcji dyskryminacyjnych tworzony jest wektor wag $w_i(x)$ zgodnie ze wzorem 5.1 lub 5.2.

— Następnie próbka trafia na wejście $N*(N-1)/2$ binarnych klasyfikatorów SVM, z których każdy oddaje na nią swój głos. W ten sposób otrzymujemy wektor głosów $g_i(x)$ dla $i = 1, 2, \ldots, N$.

— Ostatecznie mnożymy ilość głosów $g_i(x)$ przez odpowiednią wagę $w_i(x)$. W ten sposób otrzymujemy wektor ważonych głosów, z którego wybieramy klasę, dla której ilość głosów jest największa.

Analizując tablicę ważonych głosów przedstawioną w tabeli 5.3 możemy zauważyć, że tym razem różnice głosów pomiędzy klasami są dużo większe. Możemy również zauważyć, że próbka numer 5 została teraz poprawnie sklasyfikowana.

5.2. Opis przeprowadzonej procedury testowej

Opisany w tym rozdziale hybrydowy klasyfikator został przetestowany na czterech różnych bazach danych. Nie udało się przetestować go na bazie danych ACRS ze względu na zbyt dużą wymiarowość wektora cech. Nawet zaproponowane szybkie algorytmy selekcji nie były w stanie znaleźć odpowiedniego podzbioru cech w sensownym czasie (algorytmy były przerywane po 24 godzinach działania).

próbka	klasa	ilość głosów przypadająca na klasę					
		1	2	3	4	5	6
1	1	21,0(1)	18,4(15)	16,2(8)	15,2(6)	14,1(9)	12,2(16)
2	1	21,0(1)	17,1(6)	14,3(8)	14,1(7)	12,9(13)	11,6(20)
3	1	21,0(1)	19,2(6)	12,1(9)	11,3(15)	10,8(8)	9,4(11)
4	2	21,0(2)	16,3(7)	13,1(12)	10,3(15)	9,3(5)	7,5(17)
5	2	19,0(2)	18,7(7)	16,2(13)	14,2(12)	12,5(14)	10,3(13)
6	2	21,0(2)	17,2(14)	13,4(5)	12,3(22)	11,2(12)	9,9(13)
7	3	21,0(3)	14,2(17)	10,3(11)	9,3(19)	9,0(21)	8,2(12)
8	3	21,0(3)	12,1(4)	11,2(6)	10,7(18)	9,8(11)	7,7(17)
9	3	21,0(11)	18,7(17)	12,9(6)	11,3(4)	10,5(3)	9,6(8)

Tablica 5.3. Fragment tablicy głosów ważonych dla problemu białek

Klasyfikator RDA został przez autora zaimplementowany w całości. W przypadku klasyfikatora SVM została wykorzystana biblioteka LIBSVM [19] w wersji 2.91. Dostarcza ona również dodatkowych narzędzi np. skalowania zbiorów danych, do k-kroswalidacji, doboru parametrów C i γ, czy do klasyfikacji wieloklasowej.

W eksperymentach opisanych w niniejszej rozprawie wykorzystano wyłącznie funkcje ”svm_train” oraz ”svm_predict” z biblioteki LIBSVM dla problemu binarnego. Narzędzia do kroswalidacji, procedura doboru parametrów, klasyfikacja wieloklasowa z użyciem strategii OVO, OVR czy użyciem metody ECOC zostały zaimplementowane przez autora. Całość została zaimplementowana w języku C++.

Eksperymenty na bazie danych gestów zostały przeprowadzone w dwóch wariantach A i B, opisanych w rozdziale 2. W skrócie, w przypadku wariantu A klasyfikator był testowany na próbkach pochodzących od jednej osoby, a uczony na próbkach pochodzących od pozostałych. W wariancie B próbki były losowo dzielone na 6 podzbiorów, a następnie testowane na jednym z nich, a uczone na 5 pozostałych.

Pierwszym krokiem procedury testowej było uruchomienie procedury selekcji cech dla każdego problemu. Za pomocą tej procedury dobierany był także parametr regularyzacji λ dla klasyfikatora RDA. Testowane były wartości z przedziału $[0, 1]$ z krokiem $0, 1$. Wartości parametru, dla których uzyskano najlepsze wyniki, zostały zestawione w tabeli 5.4.

Kolejnym krokiem było dobranie współczynników C i γ dla klasyfikatora SVM. Zastosowana została metoda przeszukiwania siatki (ang. grid search) z krokami odpowiednio dla $C = 2^{-1}, 2^0, \ldots, 2^{10}$ i $\gamma = 2^{-10}, 2^{-9}, \ldots 2^0$. Parametry były dobierane metodą k-kroswalidacji (k = 6) z pomocą wektorów cech o pełnej wymiarowości. Wektory te zostały przeskalowane liniowo, tak aby zawierały wartości z przedziału $[-1, 1]$. Skalowanie takie jest niezbędne, aby cechy o większych wartościach nie zdominowały tych o mniejszym zakresie. Wartości parametrów C i γ uzyskane dla poszczególnych baz danych zostały umieszczone w tabeli 5.4

baza	metoda	kroswalidacja	RDA		SVM	
danych	selekcji cech	k	cechy	λ	C	γ
białka	CSO-SFFS	6	44	0,7	128	0,0156250
gesty (A)	CSO-SFFS	6	70	0,4	4	0,0312500
gesty (B)	CSO-SFFS	6	68	0,3	4	0,0312500
cyfry	CSO-SFFS	6	84	0,1	1	0,0625000
ISOLET	CSO-SFFS	6	133	0,2	2	0,0156250
ACRS	CSO-SFFS	6	214	0,1	64	0,0009765

Tablica 5.4. Parametry użyte w eksperymentach

Trzecim krokiem było uruchomienie hybrydowego klasyfikatora SVM-RDA na poszczególnych zbiorach z wykorzystaniem znalezionych w poprzednich krokach wektorów cech (dla klasyfikatora RDA) oraz odpowiednich parametrów dla obu klasyfikatorów. Wyniki otrzymane za pomocą procedury k-kroswalidacji zostały przedstawione w tabeli 5.6 (wyniki średnie) oraz w tabeli 5.7 (wyniki podane za pomocą przedziałów ufności dla 95% prawdopodobieństwa.

Ostatnim krokiem procedury testowej było sprawdzenie czy różnice pomiędzy wynikami poszczególnych par klasyfikatorów (SVM – SVM-RDA, RDA – SVM-RDA) są statystycznie znaczące. Wyniki prezentuje tabela 5.8.

Analizując tabelę 5.4 możemy zauważyć, że wartość parametru λ zależy bardzo istotnie od ilości cech i od ilości próbek w danym zbiorze. Jeśli mamy dużą ilość cech i małą liczbę próbek, wtedy wartość tego parametru zbliża się do jedynki. Oznacza to, że idziemy w kierunku średniej macierzy kowariancji (ang. pooled covariance matrix) czyli nasz algorytm zbliża się do podejścia LDA. W sytuacji, gdy liczba próbek jest duża wartość ta z kolei zmniejsza się, co oznacza, że opieramy się na macierzach indywidualnych, a zatem podejście nasze jest zbliżone do metody QDA.

baza danych	SVM-RDA był poprawny, ale		SVM-RDA był poprawny, ale
selekcji	RDA był błędny	SVM był błędny	RDA i SVM były błędne
cyfry	206	62	8
białka	45	39	15
gesty (A)	23	127	11
gesty (B)	27	52	12
ISOLET	272	89	24

Tablica 5.5. Porównanie błędów klasyfikatorów RDA, SVM i SVM-RDA

Ciekawa jest analiza tabeli 5.5, w której zamieszczono liczbę błędów popełnianych przez poszczególne klasyfikatory SVM i RDA, w porównaniu do klasyfikatora złożonego SVM-RDA. Informacje zawarte w pierwszych dwóch kolumnach pokazują, że klasyfikator SVM-RDA potrafi korygować błędy popełniane przez klasyfikatory składowe. Trzecia kolumna daje zaś

nadzieję jeszcze lepszą skuteczność. Wynika z niej, że nawet jeśli oba klasyfikatory się mylą, to i tak istnieje szansa, że SVM-RDA da poprawną odpowiedź.

Z prostego zsumowania tych wyników wypływałby wniosek, że klasyfikator SVM-RDA powinien dużo bardziej poprawić wynik klasyfikacji. Jednak analizując otrzymane rezultaty możemy zauważyć, że pojawiają się również sytuacje, w których klasyfikatory SVM, RDA *"mają rację"*, a myli się klasyfikator złożony.

Może to świadczyć o tym, że istnieje lepszy sposób konstruowania wektora wag lub też, że inne połączenie klasyfikatorów może być bardziej skuteczne. Najlepiej byłoby, aby SVM-RDA nie popełniał błędów, w sytuacji, gdy któryś z jego klasyfikatorów składowych daje poprawną odpowiedź. Innymi słowy chcielibyśmy, aby zachowywał się jak wyrocznia (ang. oracle). Wyrocznia, to abstrakcyjny model zespołu klasyfikatorów, który zawsze wie, który z jego członków daje poprawną odpowiedź i przydziela testowaną próbkę do tej właśnie klasy.

W pracy [168] pokazano, że żaden zespół klasyfikatorów oparty o wagi głosów nie może osiągnąć wyniku lepszego od wyroczni. Dalej autorzy pokazują, że modele oparte o wartości cech, etykiety klas, czy też o funkcje dyskryminacyjne mogą osiągnąć lepsze wyniki od wyroczni. To sugeruje, że podejście zaprezentowane w niniejszej pracy może być rozwijane i potencjalnie może pozwolić na osiągnięcie jeszcze lepszych wyników.

5.3. Wyniki eksperymentów

Wyniki przeprowadzonych eksperymentów zostały przedstawione w tabeli 5.6. Możemy zauważyć, że klasyfikator SVM zwykle osiągał lepsze wyniki niż klasyfikator RDA (klasyfikator RDA był lepszy tylko dla zbioru gestów). Jednak klasyfikator hybrydowy zawsze poprawiał wyniki uzyskane za pomocą klasyfikatorów SVM czy RDA stosowanych osobno.

problem	wynik klasyfikatora		
	RDA	SVM	SVM-RDA
białka	56,1%	57,2%	62,1%
gesty (A)	88,0%	81,1%	89,3%
gesty (B)	91,3%	89,8%	93,1%
cyfry	98,8%	99,1%	99,1%
ISOLET	94,2%	96,5%	97,5%

Tablica 5.6. Wyniki średnie klasyfikatorów SVM, RDA i SVM-RDA

Należy podkreślić, że wyniki zamieszczone w tabeli 5.6 są to wyniki średnie uzyskane metodą k-kroswalidacji. Nie można ich zatem bezpośrednio porównywać z wynikami innych autorów zamieszczonych w tabeli 2.8, które są wynikami punktowymi uzyskanymi za zbiorze testowym. Wynik punktowy klasyfikatora SVM-RDA na zbiorze testowym był wyższy i wyniósł 64,2% [26].

W tabeli nie zamieszczono wyników uzyskanych na zbiorze ACRS. Tak jak zostało wspomniane wcześniej, nie udało się uzyskać odpowiedniej jakości wektora cech. Osiągnięty współczynnik klasyfikacji (za pomocą algorytmu CSO-SFFS) wyniósł zaledwie 45,3%. Klasyfikator SVM uruchomiony na pełnym zbiorze cech osiągnął wynik 73,2%. Klasyfikator SVM-RDA uruchomiony na tym zbiorze poprawił wynik klasyfikatora RDA (do 61,2%, w przypadku wag danych wzorem 5.1 i 55,7%, w przypadku wag obliczanych za pomocą wzoru 5.2).

Tabela 5.7 prezentuje wyniki klasyfikatorów RDA, SVM oraz hybrydowego SVM-RDA na poszczególnych bazach danych podane za pomocą przedziałów ufności (dla 95% prawdopodobieństwa). Analizując je możemy spodziewać się, że klasyfikator SVM-RDA w większości przypadków osiągnie lepszy rezultat niż każdy z klasyfikatorów zastosowany osobno. Jednak to za mało, aby mieć pewność, że uzyskana poprawa jest znacząca statystycznie.

problem	wynik klasyfikatora		
	RDA	SVM	SVM-RDA
białka	54,1% – 58,2%	55,3% – 59,2%	60,9% – 63,4%
gesty (A)	86,3% – 89,6%	79,4% – 82,6%	86,4% – 92,1%
gesty (B)	89,5% – 93,1%	87,9% – 91,6%	91,0% – 95,1%
cyfry	98,5% – 99,0%	98,9% – 99,3%	99,0% – 99,3%
ISOLET	92,6% – 95,7%	95,2% – 97,7%	96,1% – 98,8%

Tablica 5.7. Wyniki klasyfikatorów SVM, RDA, SVM-RDA podane za pomocą przedziałów ufności

Dlatego też w kolejnej tabeli zamieszczono t-statystyki obliczone tak jak to przedstawiono w podrozdziale 1.2.6. Pozwalają one stwierdzić na poziomie ufności 0,01 lub 0,05, że klasyfikator SVM-RDA osiąga lepsze wyniki dla prawie wszystkich baz danych (oprócz bazy danych MNIST).

klasyfikatory	t-statystyka	p
RDA vs SVM–RDA (cyfry)	t = 0,12	< 0,90
SVM cs SVM–RDA (cyfry)	t = 0,21	< 0,90
RDA vs SVM–RDA (białka)	t = 5,94	< 0,01
SVM cs SVM–RDA (białka)	t = 2,62	< 0,05
RDA vs SVM–RDA (gesty A)	t = 2,84	< 0,05
SVM cs SVM–RDA (gesty A)	t = 7,12	< 0,01
RDA vs SVM–RDA (gesty B)	t = 2,59	< 0,05
SVM cs SVM–RDA (gesty B)	t = 6,37	< 0,01
RDA vs SVM–RDA (ISOLET)	t = 4,69	< 0,01
SVM cs SVM–RDA (ISOLET)	t = 3,12	< 0,05

Tablica 5.8. Zestawienie t-statystyk dla par klasyfikatorów

W przypadku tej bazy danych mamy do czynienia z bardzo dużą ilością próbek (70000) jak i małą liczbą klas (10). W tym zbiorze jest także stosunkowo niewiele próbek odstających. To powoduje, że niewiele wnoszą tutaj zalety klasyfikatora generatywnego (większa odporność na próbki odstające, czy też większa stabilność przy małej liczbie próbek), stąd tylko nieznaczna poprawa wyniku.

6. Podsumowanie i wnioski

W niniejszej pracy przedstawione zostały różne metody rozwiązywania problemu wieloklasowego. Pokazano, że wiele z nich staje się całkowicie nieefektywna, gdy liczba klas w zadaniu klasyfikacji znacząco wzrasta. W rozdziale 3 zaproponowano metody, które dzięki niższej złożoności obliczeniowej mogą być w takich przypadkach stosowane.

W podrozdziale 1.4.4 została opisana metoda wyjściowych kodów samokorygujących ECOC. Jest to bardzo ciekawe podejście zaczerpnięte z telekomunikacji. Jednak sposób tworzenia tych kodów, optymalny z punktu widzenia zastosowań telekomunikacyjnych, jest nieodpowiedni, jeżeli rozważamy je w kontekście zadania klasyfikacji. W podrozdziale 3.3 została zaproponowana metoda takiego konstruowania kodów, która pozwala na znaczące zmniejszenie błędu klasyfikacji.

Wiele nadziei wiąże się ostatnio z łączeniem klasyfikatorów reprezentujących różne metodologie. W podrozdziale 1.2.3 zostały opisane dwa odmienne podejścia do problemu klasyfikacji: generatywne i dyskryminatywne, a w podrozdziale 1.2.4 przedstawiono metody łączenia klasyfikatorów. Następnie w rozdziale 5 zaproponowany został hybrydowy klasyfikator SVM-RDA. Połączenie to sprawdza się w praktyce, poprawiając wyniki osiągane przez każdy z klasyfikatorów składowych osobno.

W opisywanych w tej pracy problemach mieliśmy do czynienia z wektorami cech o dużej wymiarowości. W takich sytuacjach stosuje się algorytmy selekcji cech, na przykład proste i łatwe w implementacji metody SFS, czy też SFFS. Jednak złożoność obliczeniowa obu tych algorytmów powoduje, że przy dużej ilości cech ich wykorzystanie jest praktycznie niemożliwe. W rozdziale 4 zaproponowano pewne modyfikacje tych metod, które pozwalają na znaczące przyspieszenie algorytmów, poprawiając jednocześnie uzyskiwane wyniki.

Poniżej, w krótkich podrozdziałach podsumowano zaproponowane rozwiązania oraz przedstawiono uzyskane wyniki.

6.1. Metody podejścia do problemu wieloklasowego

Jedną z bardziej efektywnych metod podejścia do problemu wieloklasowego jest metoda binarnych drzew decyzyjnych BDT. Pozwala ona zmniejszyć ilość klasyfikatorów binarnych niezbędnych do sklasyfikowania próbki do poziomu $O(log_2(n))$, w porównaniu do $O(n^2)$ dla metody OVO, czy $O(n)$ dla metody OVR.

Jednak w przypadku techniki BDT kluczowym problemem jest sposób budowania drzewa klasyfikatorów. W niniejszej pracy zaproponowano trzy różne podejścia do tego problemu. Uzyskane wyniki są obiecujące. Wprawdzie są one nieco gorsze od otrzymanych za pomocą metody OVO. Jednak należy pamiętać, że oferują znaczącą redukcję kosztów obliczeniowych, które powodują, że przy bardzo dużej ilości klas, stosowanie metody OVO jest w praktyce niemożliwe.

Wyniki osiągane przez klasyfikatory zbudowane przy pomocy drzew BDT, zależą istotnie od sposobu podziału problemu wieloklasowego na podproblemy binarne. Podziały te są oderwane od struktury zbiorów uczących. Od tej struktury, jak również od różnic w liczebności próbek, w stworzonych podzbiorach istotnie zależy jakość tych podziałów. Zastosowane w zaproponowanych algorytmach kryterium wymagające, aby wykorzystany podział osiągał pewien z góry założony współczynnik klasyfikacji $AssumedRecognitionRatio$ częściowo pozwala na eliminację podziałów o słabej jakości. Jednak należy zauważyć, że metody tworzenia podziałów uwzględniające strukturę tych zbiorów, na przykład metoda cięć rangowych [11], mogłyby zapewne znacząco podnieść osiągane współczynniki klasyfikacji.

Zaproponowane techniki pozwalają optymalizować te podziały na etapie konstrukcji drzewa. Powoduje to zwiększenie kosztów obliczeniowych algorytmu. Jednak koszt ten zwiększa się wyłącznie na etapie budowy klasyfikatora złożonego. Etap klasyfikacji próbek pozostaje równie szybki jak dla oryginalnej metody BDT.

Dwie inne zaproponowane metody: metoda przecięć i metoda *pół na pół* również pozwalają znacząco zmniejszyć liczbę klasyfikatorów binarnych niezbędnych do sklasyfikowania próbki. W przypadku tych technik mamy do czynienia z sytuacją odwrotną niż opisana powyżej. Sama konstrukcja klasyfikatora złożonego jest bardzo szybka. Jednak na etapie klasyfikacji mamy do czynienia z zespołem klasyfikatorów. Od ilości członków tego zespołu będzie zależał ostateczny koszt obliczeniowy jak i wynik końcowej klasyfikacji.

Otwarte pozostaje pytanie czy zwiększanie ilości członków zespołu będzie stale prowadziło do poprawy wyniku oraz w jaki sposób ustalić ”sensowną” ich ilość. Jak łatwo zauważyć maksymalna ich liczba jest ograniczona przez liczbę możliwych podziałów zbioru klas na równe lub prawie równe podzbiory. Już dla sześciu klas jest ona większa od ilości możliwych par wykorzystywanych w strategii OVO.

6.2. Nowe metody tworzenia kodów ECOC

W metodzie wyjściowych kodów samokorygujących ilość klasyfikatorów binarnych, potrzebnych do rozwiązania problemu wieloklasowego, zależy od długości słowa kodowego. Wprawdzie trudno jest wyznaczyć optymalną długość tego słowa, jednak w praktyce już kody

o długości $2n$ – $3n$ (gdzie n oznacza ilość klas), dają wyniki lepsze niż stosowanie techniki OVO.

Znane z literatury sposoby konstruowania kodów ECOC nie są optymalne z punktu widzenia zadania klasyfikacji. Zakładają one, że błędy podczas ”transmisji” kodu (klasyfikacji próbki) pojawiają się losowo. Rzeczywiście, w przypadku transmisji danych od nadajnika do odbiornika, nie mamy wpływu na ilość i częstotliwość pojawiania się błędów. Jeśli jednak mamy do czynienia z zadaniem klasyfikacji, to odpowiedni dobór słów kodowych może spowodować zmniejszenie ich liczby.

W niniejszej pracy zaproponowano metodę, która minimalizuje liczbę błędów popełnianych przez pojedyncze klasyfikatory binarne. Umożliwia to zmniejszenie błędu klasyfikacji finalnego klasyfikatora. Stosowanie jej pozwala również na skrócenie długości słów kodowych. To z kolei oznacza zmniejszenie wymaganej ilości klasyfikatorów składowych, a co za tym idzie dalsze zwiększenie szybkości klasyfikacji.

Również i w tym wypadku pewne pytania pozostają otwarte. W jaki sposób dobierać optymalną długość słowa kodowego? W jaki jeszcze sposób można zmniejszać ilość błędów popełnianych przez poszczególne klasyfikatory binarne? Jak konstruować bardziej optymalne słowa kodowe? Odpowiedzi na te pytania pozwoliłyby na dalsze poprawienie wyników jak i zwiększenie szybkości klasyfikacji.

6.3. Hybrydowy klasyfikator SVM-RDA

Klasyfikatory generatywne i dyskryminatywne reprezentują dwa odmienne podejścia do problemu klasyfikacji. W niniejszej pracy zaproponowane zostało połączenie dwóch klasyfikatorów SVM i RDA, reprezentujących oba te podejścia. Eksperymentalnie pokazane zostało, że hybrydowy klasyfikator SVM-RDA dobrze radzi sobie z różnymi zadaniami klasyfikacji.

Klasyfikator ten został przetestowany na czterech bazach danych (pięciu zadaniach klasyfikacji). W każdym przypadku zaobserwowano poprawę wyników, z tym, że w przypadku zbioru danych MNIST zmiana ta była minimalna. Za pomocą testów statystycznych wykazano, że ta poprawa wyników jest statystycznie znacząca na poziomie istotności 0,05 (w niektórych wypadkach nawet na poziomie 0,01).

Jedynie w przypadku zbioru cyfr MNIST poprawa wyniku okazała się nieznacząca. Jednak w przypadku tej bazy danych mamy do dyspozycji bardzo dużą liczbę próbek, a współczynniki klasyfikacji osiągane przez klasyfikatory składowe są zbliżone do 99%. Dodatkowo liczba klas jest stosunkowo mała. Zaproponowana metoda wykazuje zaś najlepsze własności przy dużej ilości klas i małej ilości próbek.

Uzyskane wyniki są bardzo obiecujące, choć należy poświęcić jeszcze wiele uwagi sposobowi konstruowania wag (tworzonych na podstawie funkcji dyskryminacyjnej klasyfikatora RDA). Nierozwiązaną kwestią pozostaje również stosowanie tego klasyfikatora w przypadku bardzo wysokiej wymiarowości wektora cech. O ile klasyfikator SVM doskonale sobie radzi z tym problemem, o tyle w przypadku klasyfikatora RDA konieczne jest znalezienie odpowiednich algorytmów selekcji cech.

Oczywiście zastosowana w klasyfikatorze metoda OVO z regułą większościową ma spore wady. Wynikają one z faktu, że zastosowane podziały nie uwzględniają struktury zbiorów uczących. Tworzone binarne podziały mogą być niskiej jakości, zwykle też występuje duża dysproporcja w liczności zbiorów reprezentujących klasy podziału binarnego. Otwartym zatem problemem pozostaje jak tego typu klasyfikator sprawdziłby się w przypadku wyboru innego rodzaju strategii podziału na problemy binarne np. metody cięć rangowych [11].

6.4. Metody selekcji cech

Problem zbyt dużej wymiarowości wektora cech pojawił się w przypadku klasyfikatora RDA. Jeśli w zadaniu klasyfikacji mamy zbyt mało próbek reprezentujących poszczególne klasy, a próbki te są opisane przez wektory o dużej wymiarowości, to estymacja macierzy kowariancji prowadzi do macierzy osobliwych. To z kolei uniemożliwia wykorzystanie tego klasyfikatora.

Konieczne staje się zastosowanie jakiegoś algorytmu selekcji cech. Najprostsze, a przy tym bardzo łatwe w implementacji algorytmy SFS i SFFS mają jednak złożoność $O(n^2)$, a to oznacza, że przy dużej ilości cech są całkowicie nieefektywne. Zaproponowane modyfikacje tych algorytmów zmniejszają ich optymistyczną złożoność pozwalając na ich praktyczne zastosowanie.

Zmodyfikowane w ten sposób algorytmy reprezentują połączenie dwóch technik selekcji cech: rankingowej i obudowanej. Wykorzystanie rankingów cech w kolejnych krokach metod F-SFS i F-SFFS pozwala na uzyskanie lepszych wyników. Jednocześnie koszt obliczeniowy tych metod jest znacząco mniejszy niż oryginalnych algorytmów SFS i SFFS.

6.5. Wnioski

Dziedzina, która jest tematem niniejszej rozprawy rozwija się bardzo dynamicznie. Zadania stawiane przed algorytmami klasyfikacji danych są coraz bardziej skomplikowane. Znacząco rośnie przy tym ilość rozpoznawanych klas. Opierając się na bazie danych UCI Machine Learning Repository możemy szacować, że średnia liczba klas w zadaniu klasyfikacji wzrosła

w ciągu dwudziestu lat prawie siedmiokrotnie, zaś wymiarowość wektorów cech wzrosła ponad 200-krotnie. Niniejsza rozprawa starała się zmierzyć z tym problemem, a zatem:

— W podrozdziale 1.4 omówiono kilka metod rozwiązywania problemu wieloklasowego. Pokazano, że powszechnie stosowana strategia OVO ma złożoność $O(n^2)$, co przy dużej liczbie klas uniemożliwia jej praktyczne stosowanie. Strategia OVR o złożoności $O(n)$ daje zaś w wielu przypadkach bardzo słabe wyniki.
— W rozdziale 3 omówione zostały techniki podziału problemu wieloklasowego na podproblemy binarne. Pozwalają one, ze względu na niską złożoność obliczeniową $O(log_2(n))$, rozwiązywać w praktyce zadania o bardzo dużej ilości klas. Metody, które zostały opisane (binarne drzewa decyzyjne BDT i kody ECOC) nie są nowością. Jednak zaproponowane techniki podziału problemu wieloklasowego, oparte na tych metodach, pozwalają znacząco zmniejszyć błąd klasyfikacji.
— Podsumowując, w pracy zaprezentowano różne techniki podziału problemu wielkoklasowego na podproblemy binarne. Pokazano, że pozwalają one, poprzez uwzględnienie specyfiki tych podproblemów, na zmniejszenie błędu klasyfikacji klasyfikatora finalnego.

Zaproponowane w pracy metody zostały przetestowane na czterech różnych zadaniach klasyfikacji reprezentujących całkowicie różne dziedziny wiedzy. Wyniki eksperymentów są zachęcające. Opisane metody są efektywne, a uzyskane współczynniki klasyfikacji porównywalne lub lepsze od tych, które są osiągane za pomocą znanych do tej pory metod.

Analiza ich złożoności obliczeniowej wskazuje iż mogą być one stosowane również w przypadku problemów o dużo większej ilości klas. Pewne pytania pozostają jednak nadal otwarte. Jak na przykład problem doboru optymalnej długości słowa kodowego w metodzie ECOC, czy też doboru odpowiedniej metody budowania drzew BDT. Wskazują one jednocześnie możliwe kierunki dalszych prac badawczych.

W przypadku zaproponowanych metod selekcji cech, osiągnięto bardzo znaczący przyrost wydajności algorytmów. Dzięki nim udało się zastosować klasyfikator RDA do baz danych opisywanych w tej pracy (bez nich było to możliwe jedynie dla zbioru cyfr). Proponowane algorytmy będące połączeniem metod rankingowych i obudowanych, opublikowane w pracach [26], [27], pozwoliły także na znaczną poprawę współczynnika klasyfikacji hybrydowego klasyfikatora SVM-RDA.

Bardzo dobrym testem dla zaproponowanych metod selekcji okazał się zbiór ACRS. Bardzo duża ilość cech (10000) pokazała, że mimo wielokrotnego przyspieszenia algorytmów, wciąż mogą pojawić się problemy, dla których to rozwiązanie okaże się niewystarczające. Stanowi to kolejne wyzwanie i daje motywację do dalszej pracy w tym kierunku. Obiecującym wyborem wydaje się być sięgnięcie po metody wbudowane.

Rosnące możliwości komputerów, to zachęta do tworzenia klasyfikatorów coraz bardziej złożonych, które mogłyby uzyskiwać coraz lepsze wyniki, poprzez wykorzystywanie mocnych stron różnych podejść do problemu klasyfikacji. Próbą zmierzenia się z tym problemem był zaproponowany hybrydowy klasyfikator SVM-RDA [21], [24], [27]. Połączenie dwóch klasyfikatorów, reprezentujących podejście generatywne i dyskryminatywne, przyniosło znaczną poprawę współczynnika klasyfikacji. Zwłaszcza w przypadku zbiorów o małej ilości próbek.

Również i w tym przypadku łatwo zauważyć możliwe tematy dalszych badań. Na przykład bardzo istotnie na wynik klasyfikacji wpływają wagi obliczane na podstawie funkcji dyskryminacyjnej klasyfikatora RDA. Modyfikacja tych wag, a nawet ich dobór zależny od rozwiązywanego problemu, czy rozpatrywanej przestrzeni cech, mógłby pozwolić na dalszą poprawę wyników osiąganych przez ten klasyfikator.

Temat, z którym starała się zmierzyć niniejsza rozprawa, jest bardzo szeroki. W trakcie pisania pracy pojawiały się wciąż nowe pomysły, otwierały się wciąż nowe kierunki badań. Na przykład baza danych ACRS, która pojawiła się w zbiorze UCI Machine Learning Repository pod koniec 2011 roku, pokazała jak duże jeszcze wyzwania stoją przed algorytmami selekcji cech.

Metody rozwiązywania problemu wieloklasowego, które zostały zaproponowane, choć efektywne, nadal pozostawiają duże pole do popisu, zwłaszcza jeśli chodzi o osiągane przez nie współczynniki klasyfikacji. Świadczy o tym choćby problem identyfikacji anonimowych blogerów [110] (100 000 klas), w którym osiągnięto zaledwie 20% poprawnych klasyfikacji. Takie zadania stanowią jednak doskonałą zachętę do prowadzenia dalszych badań w tym kierunku oraz podejmowania kolejnych wyzwań.

Bibliografia

[1] S. Abe, Support Vector Machines for Pattern Classification, Springer Verlag, New York, 2010.

[2] E. Allwein, R. Schapire, Y. Singer, Reducing multiclass to binary: A unifying approach for margin classifiers. Journal of Machine Learning Research 1, pp. 113–141, 2000.

[3] P. Baldi, S. Brunak, Y. Chauvin, C. Andersen, H. Nielsen, Assessing the accuracy of prediction algorithms for classification: an overview, Bioinformatics, 16, pp. 412–424, 2000.

[4] P. Baldi, S. Brunak, Bioinformatics: the machine learning approach, MIT Press, Cambridge, MA, Second edition, 2001.

[5] Y. Baram, Partial Classification: The benefit of deferred decision, , Vol. 20, No. 8, pp. 769–776, 1998.

[6] S. Belongie, J. Malik, J. Puzicha, Shape matching and object recognition using shape contexts, IEEE Transations on Pattern Analysis and Machine Intelligence, Vol. 24, No. 4, pp. 509–522, 2002.

[7] H.M. Berman, The Protein Data Bank: a historical perspective, Acta Crystallographica Section A: Foundations of Crystallography A64 (1), pp. 88–95, 2008.

[8] C.M. Bishop, J. Lasserre, Generative or Discriminative? Getting the Best of Both Worlds, Bayesian Statistics 8, pp. 3–24, 2007.

[9] L. Bobrowski, Data mining based on convex and piecewise linear (CPL) criterion functions, Technical University Białystok 2005.

[10] L. Bobrowski, T. Łukaszuk, Feature selection based on relaxed linear separability, Biocybernetics and Biomedical Engineering, Vol. 29(2), pp. 43–58, 2009.

[11] L. Bobrowski, Induction of linear separability through the ranked layers of binary classifiers, Engineering applications of neural networks, in IFIP Advances in Information and Communication Technology, Vol.363, pp. 69–77, Heidelberg, Springer, 2011

[12] G. Bologna, R.D. Appel, A comparison study on protein fold recognition, Proceedings of the 9th ICONIP, Singapore, Vol. 5, pp. 2492–2496, 2002.

[13] R.C. Bose, D.K. Ray-Chaudhuri, On a class of error-correcting binary group codes, Information and Control, Vol. 3, No. 1, pp. 68–79, 1960.

[14] A.P. Bradley, The use of the area under the ROC curve in the evaluation of machine learning algorithms, Pattern Recognition, Vol. 30(7), pp. 1154–1159, 1997.

[15] L. Breiman, J.H. Friedman, R.A. Olshen, C.J. Stone, Classification and Regression Trees, Wadsworth, Belmont, 1984.

[16] L. Breiman, Bagging Predictors, Machine Learning, Vol. 24, No. 2, pp. 123–140, 1996.

[17] M.R.S. Brown, W.N. Grundy, D. Lin, N. Cristianini, C. Sugnet, M. Ares, D. Haussler, Knowledge-based Analysis of Microarray Gene Expression Data By Using Support Vector Machines, Proceedings of the National Academy of Sciences, Vol. 97, No. 1, pp. 262–267, 2000.

[18] H.S. Chan, K. Dill, The protein folding problem. Physics Today (Feb.), pp. 24–32, 1993.

[19] C.C. Chang, C.J. Lin: LIBSVM: a library for support vector machines. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm, 2001.

[20] O. Chapelle, S. Keerthi, Multi-Class Feature Selection with Support Vector Machines, In Proceedings of the American Statistical Association, 2008.

[21] W. Chmielnicki, K. Stąpor, Protein Fold Recognition with Combined SVM-RDA Classifier, HAIS 2010 Part I, Lecture Notes in Artificial Intelligence 6076, pp.162–169, 2010.

[22] W. Chmielnicki, K. Stąpor, Investigation of Normalization Techniques and Their Impact on a Recognition Rate in Handwritten Numeral Recognition, Schedae Informaticae, Vol. 19, pp. 53–77, 2010.

[23] W. Chmielnicki, K. Stąpor, I. Roterman-Konieczna, An Efficient Multi-class Support Vector Machine Classifier for Protein Fold Recognition, Advances in Intelligent and Soft Computing, Vol. 74, pp. 77–84, 2010.

[24] W. Chmielnicki, K. Stąpor, A combined SVM-RDA classifier for protein fold recognition, Bio-Algorithms and Med-Systems, Vol. 7, No. 13, pp. 67–70, 2011.

[25] W. Chmielnicki, K. Stąpor, A New Approach to Multi-class SVM-Based Classification Using Error Correcting Output Codes, Advances in Intelligent and Soft Computing, Vol. 95, pp. 499–506, 2011.

[26] W. Chmielnicki, K. Stąpor, An Effective Feature Selection Algorithm Based on the Class Similarity Used with a SVM-RDA Classifier to Protein Fold Recognition, Lecture Notes in Computer Science Vol. 6679, 2011, pp. 205–212, 2011.

[27] W. Chmielnicki, K. Stąpor, A hybrid discriminative/generative approach to protein fold recognition, Neurocomputing, Vol. 75, No. 1, pp. 194–198, 2012.

[28] W. Chmielnicki, I. Roterman-Konieczna, K. Stąpor, An improved protein fold recognition with support vector machines, Expert Systems, Vol. 20, No. 2, pp. 200–211, 2012.

[29] C. Chothia, One thousand families for the molecular biologist, Nature 357, pp. 543–544, 1992.

[30] I.F. Chung, C.D. Huang, Y.H. Shen, C.T. Lin, Recognition of structure classification of protein folding by NN and SVM hierarchical learning architecture, Artificial Neural Networks and Neural Information Processing – ICANN/ICONIP 26–29 June, Istanbul, Turkey, pp. 1159–1167, 2003.

[31] R. Cole, Y. Muthusamy, M. Fanty, The ISOLET Spoken Letter Database, Technical Report 90-004, Computer Science Department, Oregon Graduate Institute, 1990.

[32] D. Coppersmith, S. Winograd, On the asymptotic complexity of matrix multiplication, Proceedings of the 22nd Annual Symposium on Foundations of Computer Science, pp. 82–90, 1981.

[33] K. Crammer, Y. Singer, On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines, Journal of Machine Learning Research 2, pp. 265–292, 2001.

[34] N. Cristianini, B. Scholkopf, Support vector machines and Kernel methods: the new generation of learning machines, AI Magazine, Vol. 13, No. 3, pp. 31–41, 2002.

[35] A. Dehzangi, S. Phon-Amnuaisuk, O. Dehzangi, Using Random Forest for Protein Fold Predition Problem: An Empirical Study, Journal of Information Science and Engineering 26, pp. 1941–1956, 2010.

[36] J. Demsar, Statistical Comparisons of Classifiers over Multiple Data Sets, Journal of Machine Learning Research 7, pp. 1–30, 2006.

[37] P.A. Devijver, J. Kittler, Pattern Recognition: A Statistical Approach, Prentice-Hall, 1982.

[38] T.G. Dietterich, G. Bakiri, Solving multiclass problems via error-correcting output codes, Journal of Artificial Intelligence Research 2, pp. 263–286, 1995.

[39] C.H. Ding, I. Dubchak, Protein Fold Prediction Data sets used in the paper "Multi-class Protein Fold Recognition Using Support Vector Machines and Neural Networks", http://ranger.uta.edu/ chqding/protein/, 2001.

[40] C.H. Ding, I. Dubchak, Multi-class protein fold recognition using support vector machines and neural networks, Bioinformatics 17, pp. 349–358, 2001.

[41] J.X. Dong, A. Krzyzak, C.Y. Suen, A multi-net learning framework for pattern recognition, Proceedings of the Sixth International Conference on Document Analysis and Recognition, Seattle, pp. 328–332, 2001.

[42] I. Dubchak, I. Muchnik, S.R. Holbrook, S.H. Kim, Prediction of protein folding class using global description of amino acid sequence, Proc. Natl. Acad. Sci. USA, 92, pp. 8700–8704, 1995.

[43] I. Dubchak, I, Muchnik, S.H. Kim, Protein folding class predictor for SCOP: approach based on global descriptors, Proceedings of ISMB, 1997.

[44] I. Dubchak, I. Muchnik, C. Mayor, I. Dralyuk, S.H. Kim, Recognition of protein fold in the context of the Structural Classification of Proteins (SCOP) classification, Proteins 35, pp. 401–407, 1999.

[45] W. Duch, N. Jankowski, Survey of neural transfer functions, Neural Computing Surveys 2, pp. 163–212, 1999.

[46] W. Duch, Filter methods, In: Feature extraction, foundations and applications, Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, pp. 89-118, 2006.

[47] R.O. Duda, P.E. Hart, G.D. Stork, Pattern classification, John Wiley and Sons, New York, 2001.

[48] S.R. Eddy, Hidden Markov models. Curr. Opin. Struct. Biol., 6, pp. 361–365, 1995.

[49] B. Efron, R. Tibshirani, An Introduction to the Bootstrap, New York, Chapman & Hall, 1993.

[50] T.P. Exarchos, C. Papaloukas, Ch. Lampros, D.I. Fotiadis, Mining sequential patterns for protein fold recognition, Journal of Biomedical Informatics 41, pp. 165–179, 2008.

[51] M. Fanty, R. Cole, Spoken letter recognition, In Advances in Neural Information Processing Systems 3, San Mateo, Morgan Kaufman, 1991

[52] B. Fei, J. Liu, Binary Tree of SVM: A New Fast Multiclass Training and Classification Algorithm, IEEE Transaction on Neural Networks, Vol. 17, No.3, pp. 696–704, 2006.

[53] E. Fix, J.L. Hodges, Discriminatory analysis. Nonparametric discrimination: consistency properties, Report Number 4, Project Number 21-49-004, 1951, Reprinted in International Statistical Review, 57, pp. 238–247, 1989.

[54] G. Forman, An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research 3, pp. 1289–1305, 2003.

[55] Y. Freund, Boosting weak learning algorithm by majority, Information and Computation, No. 121(2), pp. 256–285, 1995.

[56] J.H. Friedman, Regularized Discriminant Analysis, Journal of the American Statistical Association, 84(405), pp. 165–175, 1989.

[57] K. Fukunaga, Introduction to Statistical Pattern Recognition, 2nd Ed. Academic Press, New York, 1990.

[58] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, F. Herrera, An overview of ensemble methods for binary classifiers in multi-class problems: Experimental study on one-vs-one and one-vs-all schemes, Pattern Recognition 44, pp. 1761–1776, 2011.

[59] K. Ginalski, N.V. Grishin, A. Godzik, L. Rychlewski, Practical lessons from protein structure prediction, Nucleic Acids Research, Vol. 33, No. 6, pp. 1874–1891, 2005.

[60] P. Glomb, M. Romaszewski, S. Opozda, A. Sochan, Choosing and modeling hand gesture database for natural user interface, In GW2011: The 9th International Gesture Workshop Gesture in Embodied Communication and Human-Computer Interaction, 2011.

[61] M. Gori, F. Scarselli, Are multilayer perceptrons adequate for pattern recognition and verification? IEEE Transations on Pattern Analysis and Machine Intelligence, Vol. 20, No. 11, pp. 1121–1132, 1998.

[62] I. Guyon, J. Weston, S. Barnhill, V. Vapnik, Gene selection for cancer classification using support vector machines, Machine Learning, Vol. 46(1/3), pp. 389–422, 2002.

[63] I. Guyon, A. Elisseeff, An Introduction to Variable and Feature Selection. Journal of Machine Learning Research 3 pp. 1157–1182, 2003.

[64] M. Haindl, P. Somol, D. Ververidis, C. Kotropoulos, Feature Selection Based on Mutual Correlation, Proceedings of Progress in Pattern Recognition, Image Analysis and Application, 4225, pp 569–577, 2006.

[65] T. Hastie, R. Tibshirani, Classification by pairwise coupling, Annals of Statistics, Vol. 26, No. 2, pp. 451–471, 1998.

[66] T. Hasti, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition, Springer, 2009.

[67] T.K. Ho, The Random Subspace Method for Constructing Decision Forests, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 8, pp. 832–844, 1998.

[68] T.K. Ho, Multiple classifier combination: Lessons and the next steps. In Hybrid Methods in Pattern Recognition. World Scientific Publishing, pp. 171–198, 2002.

[69] U. Hobohm, M. Scharf, R. Schneider, C. Sander, Selection of a representative set of structures from the Brookhaven Protein Bank. Protein Sci. 1, pp. 409–417, 1992.

[70] U. Hobohm, C. Sander, Enlarged representative set of proteins structures, Protein Sciemce, Vol. 3, pp. 522–524, 1994.

[71] C. Hsu, C. Lin, A Comparison of methods for multi-class support vector machines, IEEE Transactions on Neural Networks, 13, pp. 414–425, 2002.

[72] B.W. Hwang, S. Kim, S.W. Lee, 2D and 3D full-body gesture database for analyzing daily human gestures. In Advances in Intelligent Computing, Lecture Notes in Computer Science, Springer, vol. 3644, pp. 611–620, 2005.

[73] A.K. Jain, R.P.W. Duin, J. Mao, Statistical pattern recognition, a review. IEEE Transations on Pattern Analysis and Machine Intelligence, Vol. 22, No. 1, pp. 4–37, 2000.

[74] M. James, Pattern Recognition, John Wiley and Sons, New York, pp. 36–40, 1988.

[75] Y. Ji, S. Sun, Multitask Multiclass Support Vector Machines, IEEE 11th International Conference on Data Mining Workshops (ICDMW), pp. 512–518, 2011.

[76] Y. Jiang, P. Guo, Regularization versus dimension reduction, which is better? in: Lecture Notes in Computer Science, Vol. 4492, part II, pp. 474–482, 2007.

[77] L. Jinbai, F. Ben, X. Lihong, Binary tree of Support Vector Machine in multi-class classification problem 3rd ICECE, 2004.

[78] M.J. Kearns, R.E. Schapire, L.M. Sellie, Towards efficient agnostic learning, Machine Learning, Vol. 17, pp. 115–141, 1994.

[79] J.M. Keller, M.R. Gray, J.A. Givens, A Fuzzy K-Nearest Neighbor Algorithm, IEEE Transactions on Systems, Man and Cybernetics, Vol. 15(4) pp. 580–585, 1985.

[80] B. Kijsirikul, N. Ussivakul, Multiclass support vector machines using adaptive directed acyclic graph, Proceedings of IJCNN, pp. 980–985, 2002.

[81] T.K. Kim, S.F. Wong, R. Cipolla, Tensor canonical correlation analysis for action classification. In: Proceedings of IEEE Conference on Computer Vision and Pattern Recognition CVPR, 2007.

[82] J. Kittler, M. Hatef, R. P. W. Duin, J. Matas, On combining classifiers. IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20, No. 3, pp. 226–239, 1998.

[83] R. Kohavi, G.H. John, Wrappers for feature selection, Artificial Intelligence 97, pp. 273–324, 1997.

[84] L.I. Kuncheva, Clustering-and-selection model for classifier combination, Proceedings of 4th International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies, Brighton, UK, pp. 185–188, 2000.

[85] L.I. Kuncheva, J.C. Bezdek, R.P.W. Duin, Decision templates for multiple classifiers fusion, Pattern Recognition, Vol. 34, No. 2, pp. 299–214, 2001.

[86] L.I. Kuncheva, Combining pattern classifiers: Methods and algorithms, Wiley Interscience, New Jersey, 2004.

[87] M. Kurzyński, Rozpoznawanie obiektów. Metody statystyczne, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1997.

[88] M. Kurzyński, M. Woźniak, Combining classifiers under probabilistic models: experimental comparative analysis of methods, Expert Systems, Vol. 29, No. 4, pp. 374-393, 2012.

[89] L. Ladha, T. Deepa, Feature selection methods and algorithms, International Journal on Computer Science and Engineering (IJCSE), Vol. 3, No. 5, pp. 1787–1797, 2011.

[90] C. Lai, M.J. Reinders, L. Wessels, Random subspace method for multivariate feature selection, Pattern Recognition Letters, Vol. 27, No. 10, pp. 1067–1076, 2006.

[91] L. Lam, Classifier Combinations: Implementations and Theoretical Issue, Lecture Notes in Computer Science, No. 1857, pp. 77–87, 2000.

[92] Ch. Lampros, C. Papaloukas, T.P. Exarchos, Y. Golectsis, D.I Fotiadis, Sequence-based protein structure prediction using a reduced state-space hidden Markov model, Computers in Biology and Medicine, 37, pp. 1211–1224, 2007.

[93] Ch. Lampros, C. Papaloukas, K. Exarchos, D.I Fotiadis, Improving the protein fold recognition accuracy of a reduced state-space hidden Markov model, Computers in Biology and Medicine, 39, pp. 907–914, 2009.

[94] Y. Lee, C.K. Lee, Classification of Multiple Cancer Types by Multicategory Support Vector Machines Using Gene Expression Data, Bioinformatics, 19, pp. 1132–1139, 2003.

[95] C.L. Liu, K.Nakashima, H. Sako, H. Fujisava, Handwritten digit recognition: investigation of normalization and feature extraction techniques, Pattern Recognition 37, pp. 265–279, 2004.

[96] C.L. Liu, H. Fujisawa, Classification and Learning for Character Recognition: Comparison of Methods and Remaining Problems, Proc. Int. Workshop on Neural Networks and Learning in Document Analysis and Recognition, Seoul, Korea, 2005.

[97] H. Liu, X. Ding, Handwritten Character Recognition Using Gradient Feature and Quadratic Classifier with Multiple Discrimination Schemes, Proceedings of the Eighth ICDAR, pp. 19–25, 2005.

[98] S. Liu, Z. Liu, J. Sun, L. Liu, Application of Synergetic Neural Network in Online Writeprint Identification, International Journal of Digital Content Technology and its Applications, Vol. 5, No. 3, 2011.

[99] L. Lo Conte, B. Ailey, T.J.P. Hubbard, S.E. Brenner, A.G. Murzin, C. Chotchia, SCOP: a structural classification of protein database. Nucleic Acids Res. 28, pp. 257–259, 2000.

[100] P.C. Loizou, A.S. Spanias, High-Performance Alphabet Recognition, IEEE Transactions on Speech and Audio Processing, Vol. 4, No. 6, pp. 430–445, 1996.

[101] J. Lu, K.N. Plataniotis, A.N. Venetsanopoulos, Regularized discriminant analysis for the small sample size problem in face recognition, Pattern Recognition Letters 24, pp. 3079-3087, 2003

[102] G. Madzarov, D. Gjorgjevskij, I. Chorbev, A multi-class SVM classifier utilizing decision tree. Informatica 33, pp. 233–241, 2009.

[103] S. Maldonado, R. Weber, J. Basak, Simultaneous feature selection and classification using kernel-penalized support vector machines, Information Sciences 181, pp. 115-128, 2011.

[104] G. Mayraz, G.E. Hinton, Recognizing handwritten digits using hierarchical products of experts, IEEE Transations on Pattern Analysis and Machine Intelligence, Vol. 24, No. 2, pp. 189–197, 2002.

[105] T.K. Moon, Error Correction Coding: Mathematical Methods and Algorithms, John Wiley and Sons, New Jersey, 2005.

[106] D.S. Moore, G. McCabe, Introduction to the practice of statistics. New York: W.H.Freeman and Company, 2003.

[107] R. Mukundan, K.R. RamaKrishnan, Fast Computation of Legenfre and Zernike Moments, Pattern Recognition Vol. 28 (9), pp. 1433–1442, 1995.

[108] L. Nanni, A novel ensemble of classifiers for protein fold recognition, Neurocomputing 69, pp. 2434–2437, 2006.

[109] L. Nanni, S. Brahnam, A. Lumini, High performance set of PseAAC and sequence based descriptors for protein classification, Journal of Theoretical Biology, 266 (1), pp. 1–10, 2010.

[110] A. Narayanan, H. Paskov, N. Zhenqiang Gong, J. Bethencourt, E. Stefanov, E. Shin, D. Song, On the Feasibility of Internet-Scale Author Identification, IEEE Symposium on Security and Privacy 2012, pp. 300–314, 2012.

[111] A.Y. Ng, M.I. Jordan, On discriminative vs. generative classifiers: a comparison of logistic regression and naive bayes. In NIPS 14, pp. 841–848, 2001.

[112] A.Y. Ng, Feature selection, L1 vs. L2 regularization, and rotation invariance, Proceedings of the Twenty-first International Conference on Machine Learning, pp. 78–86, 2004.

[113] NIST Special Database 19, http://www.nist.gov/srd/nistsd19.cfm.

[114] O. Okun, Protein fold recognition with k-local hyperplane distance nearest neighbor algorithm, Proceedings of the Second European Workshop on Data Mining and Text Mining in Bioinformatics, 24 September, Pisa, Italy, pp. 51–57, 2004.

[115] S. Osowski, Sieci neuronowe do przetwarzania informacji, Oficyna Wydawnicza Politechniki Warszawskiej, Warszawa 2006.

[116] N.R. Pal, D. Chakraborty, Some new features for protein fold recognition, Artificial Neural Networks and Neural Information Processing ICANN/ICONIP, Vol. 2714, Turkey, Istanbul, June 26–29, pp. 1176–1183, 2003.

[117] J.C. Platt, Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In Advances in Large Margin Classifiers. MIT Press, 1999.

[118] J.C. Platt, N. Cristianini, J. Shawe-Taylor, Large Margin DAGs for Multiclass Classification. In Proceedings of Neural Information Processing Systems, pp. 547–553, 2000.

[119] L. Prevost, L. Oudot, A. Moises, Ch. Michel-Sendis, M. Milgram, Hybrid generative/disciminative classifier for unconstrained character recognition, Pattern Recognition Letters, Vol. 26, No. 12, pp. 1840–1848, 2005.

[120] Protein Data Bank, http://www.pdb.org/pdb/home/home.do.

[121] J.R. Quinlan, Discovering rules by induction from large collections of examples, in Expert Systems in the Micro Electronic Age, Edinburgh University Press, pp. 168–201, 1979.

[122] J.R. Quinlan, Induction of Decision Trees, Machine Learning 1(1), pp. 81–106, 1986.

[123] J.R. Quinlan, C4.5 Programs for Machine Learning, San Mateo, Morgan Kaufmann, 1993.

[124] J.R. Quinlan, Bagging, Boosting and C4.5, Proc. of Thirteenth national Conference on Artificail Intelligence and Eight Innovative Applications of Artificial Intelligence Conference, Vol. 1, pp. 725–730, 1996.

[125] P. Radivojac, Z. Obradovic, K. Dunker, S. Vucetic, Feature selection filters based on the permutation test, Lecture Notes in Computer Science, Springer, Vol. 3201, pp. 334–346, 2004.

[126] R.J. Ramteke, S.C. Mehrotra, Feature Extraction Based on Moment Invariants for Handwriting, Proc. of 2006 IEEE Conference on Recognition Cybernetics and Intelligent Systems, Bangkok, Thailand, 7-9 June 2006, pp. 1–6, 2006.

[127] J. Reunanen, Search strategies for wrapper methods In: Feature extraction, foundations and applications. Eds: I. Guyon, S. Gunn, M. Nikravesh, L. Zadeh, Studies in Fuzziness and Soft Computing, Physica-Verlag, Springer, pp. 89-118, 2006.

[128] R. Rifkin, A. Klautau, In Defense of One-vs-All Classification, Journal of Machine Learning Research 5, pp. 101–141, 2004.

[129] M. Romaszewski, P. Glomb, S. Opozda, A. Sochan, Choosing and modelling gesture database for natural user interface, Computer Recognition Systems 4, 2011.

[130] B. Rost, C. Sander, Prediction of protein secondary structure at better than 70% accuracy, Journal of Molecular Biology 232, pp. 584–599, 1993.

[131] V. Roth, The Generalized LASSO, IEEE Transactions on Neural Networks, Vol. 15(1), pp. 16–28, 2004.

[132] L. Rychlewski, J. Bujnicki, D. Fischer, Protein Fold-Recognition and Experimental Structure Determination, chapter in The New Avenues in Bioinformatics, 2003.

[133] R.E. Schapire, The Strength of Weak Learnability, Machine Learning 5, pp. 197–227, 1990.

[134] B. Scholkopf, A.J. Smola, Learning with Kernels. Support Vector Machines, Regularization, Optimization, and Beyond, The MIT Press 2002.

[135] R. Setiono, H. Liu, Chi2: Feature selection and discretization of numeric values, in: proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence, pp.388–391, 1995.

[136] H.B. Shen, K.C. Chou, Ensemble classifier for protein fold pattern recognition, Bioinformatics, 22, pp. 1717–1722, 2006.

[137] H.B. Shen, K.C. Chou, Hum-mPLoc, An ensemble classifier for large-scale human protein subcellular location prediction by incorporating samples with multiple sites, Biochemical and Biophysical Research Communications, 355, pp. 1006–1011, 2007.

[138] M. Shi, Y. Fujisava T. Wakabayashi, F. Kimura, Handwritten Numeral Recognition using gradient and curvature of gray scale image, Pattern Recognition 35 (10), pp. 2051–2059, 2002.

[139] L. Sigal, A. Balan, M.J. Black, HumanEva: Synchronized video and motion capture dataset and baseline algorithm for evaluation of articulated human motion. International, Journal of Computer Vision 87(1-2) pp. 4–27, 2010.

[140] T.F. Smith, M.S. Waterman, Idetification of common molecular subsequences, Journal of Molecular Biology, 147, pp. 195–197, 1981.

[141] K. Stąpor, Automatyczna klasyfikacja obiektów, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2005.

[142] K. Stąpor, Wykłady z metod statystycznych dla informatyków, Wydawnictwo Politechniki Śląskiej, Gliwice, 2008.

[143] K. Stąpor, Protein fold classification based on machine learning paradigm - a review, Bio-Algorithms and Med-Systems Vol. 8, No. 1, pp. 53–76, 2012.

[144] D. Subramanian, R. Greiner, J. Pearl, The relevance of relevance, Artificial Intelligence, Vol. 97, No. 1, pp. 1–5, 1997.

[145] C. Sutton, A. McCallum, An introduction to conditional random fields for relational learning, In: Introduction to Statistical Relational Learning, MIT Press, Cambridge, pp. 93-128, 2006.

[146] R. Tadeusiewicz, M. Flasiński, Rozpoznawanie obrazów, PWN, Warszawa, 1991.

[147] M.R. Teague, Image analysis via the general theory of moments, Journal of the Optical Society of America, Vol. 70, No. 8, pp. 920–930, 1980.

[148] L.N. Teow, K.F. Loe, Robust vision-based features and classification schemes for offline handwritten digit recognition, Pattern Recognition 35 (11), pp. 2355–2364, 2002.

[149] The MNIST database of handwritten digits http://yann.lecun.com/exdb/mnist/

[150] S. Theodoridis, K. Koutroumbas: Pattern Recognition, Academic Press, 1999.

[151] R. Tibshirani, Regression Shrinking and Selection via Lasso, Journal of the Royal Statistical Society. Series B (Methodology), Vol. 58(1), pp. 267–288, 1996.

[152] X.J. Tong, S. Zeng, K. Zhou, Q. Jiang, Hand-written numeral recognition based on Zernike moment, Proceedings of the 2008 ICWAPR, pp. 368–372, 2008.

[153] K. Torkkola, Information-theoretic methods for feature selection and construction, in: Feature extraction, foundations and Applications, Springer, 2005.

[154] UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/datasets.html.

[155] Universal Protein Resource, http://www.uniprot.org/

[156] L.G. Valiant, A theory of the learnable, Communication of the ACM, Vol. 27, pp. 1134–1142, 1984.

[157] V. Vapnik, The Nature of Statistical Learning Theory, Springer, New York, 1995.

[158] V. Vural, J.G. Dy, A Hierarchical Method for Multi-Class Support Vector Machines, in: Proceedings of the 21st International Conference on Machine Learning, pp. 831–838, 2004.

[159] L. Wang, X. Shen, Multi-category support vector machines, feature selection and solution path, Statistica Sinica 16, pp. 617–633, 2006.

[160] L. Wang, J. Lu, K.N. Plataniotis, A.N. Venetsanopoulos, Kernel quadratic discriminant analysis for small sample size problem, Pattern Recognition, Vol. 41, Issue 5, pp. 1528–1538, 2008.

[161] T. Windeatt, R. Ghaderi, Coding and decoding for multiclass learning problems, Information Fusion 4(1), pp. 11–21, 2003.

[162] K. Woods, W.P. Kegelmeyer, K. Bowyer, Combination of multiple classifiers using local accuracy estimates, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 19, No. 4, pp. 405–410, 1997.

[163] M. Woźniak, Metody fuzji informacji dla komputerowych systemów rozpoznawania, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław 2006.

[164] D. Xu, H. Li, Geometric moment invariants, Pattern Recognition 41, pp. 240–249, 2008.

[165] L. Yu, H. Liu, I. Guyon, Efficient feature selection via analysis of relevance and redundancy, Journal of Machine Learning Research, 5, pp. 1205–1224, 2004.

[166] C.X. Zhang, J.S. Zhang, RotBoost: a technique for combining Rotation Forest and AdaBoost, Pattern Recognition Letters, Vol. 29, No. 10, pp, 1524–1536, 2008.

[167] J. Zhang, S. Gong, Action categorization with modified hidden conditional random field, Pattern Recognition 43, pp. 197–203, 2010.

[168] M. Zmyślony, M. Woźniak, K. Jackowski, Comparative Analysis of Classifier Fusers, International Journal of Artficial Intelligence & Applications, Vol. 3, No. 3, 2012.