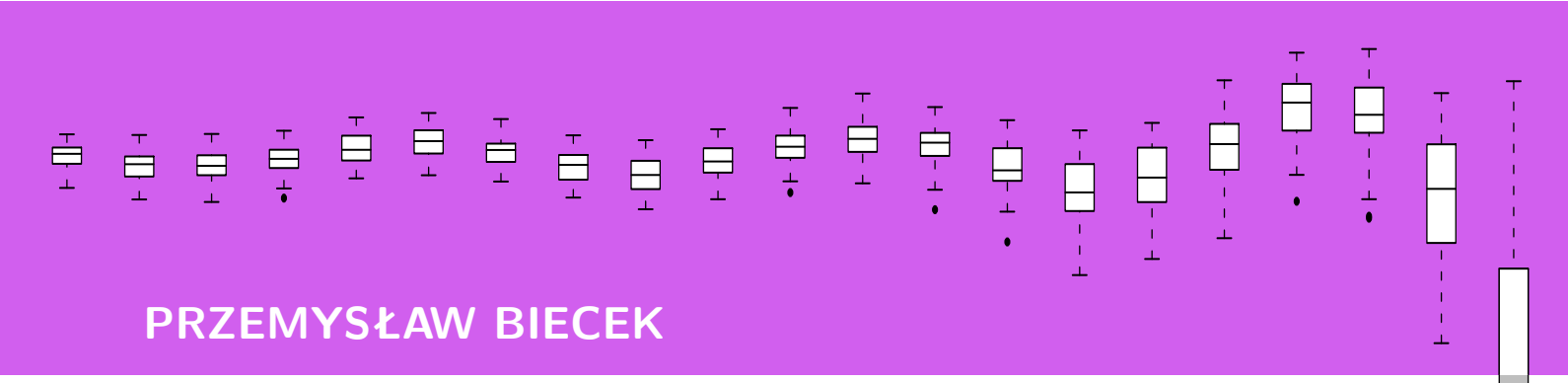
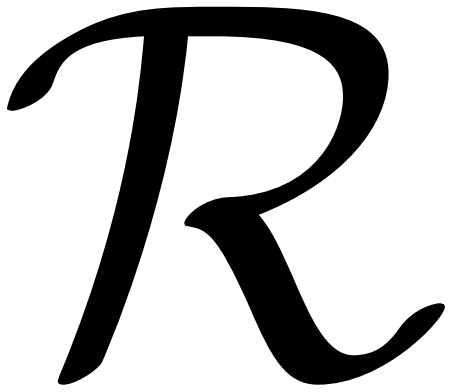
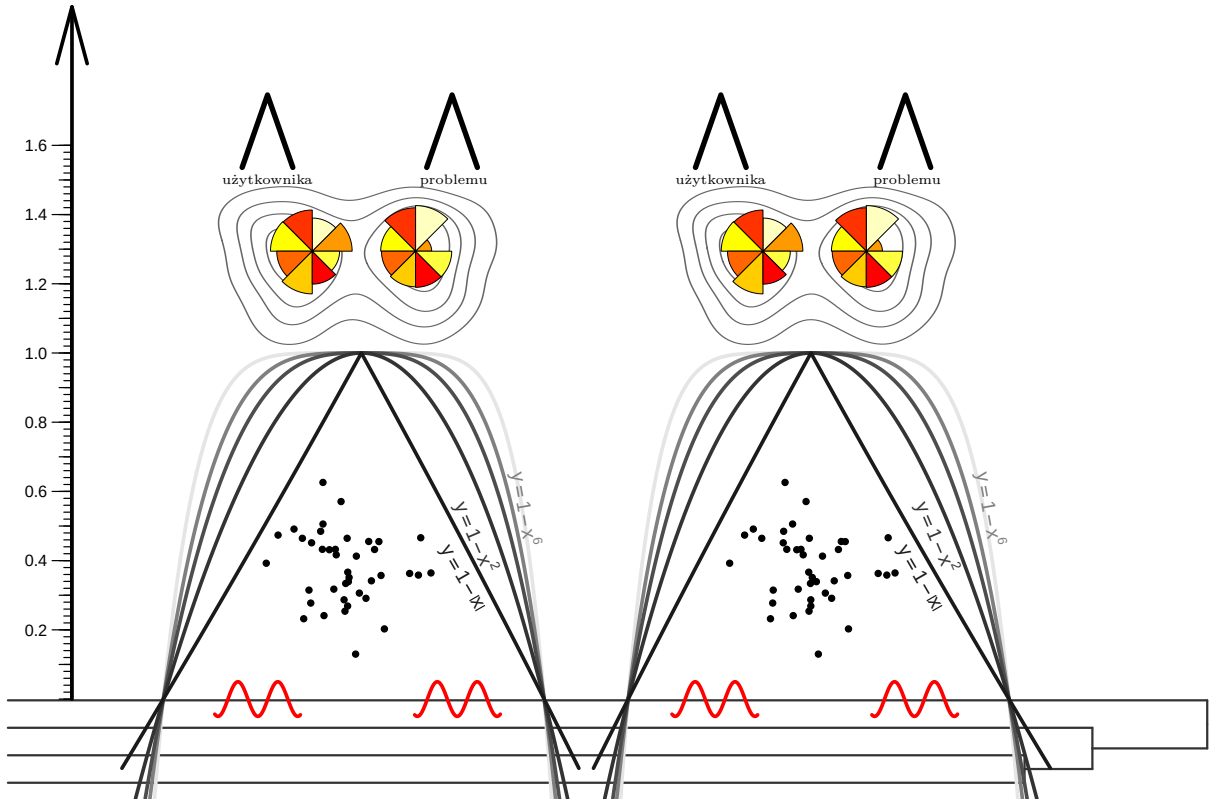


PRZEMYSŁAW BIECEK

Na przełaj przez Data Mining z pakietem



wersja 0.2, robocza



Spis treści

Rozdział 1

Kilka słów zamiast wstępu

na przełaj „najkrótszą drogą, nie trzymając się wytyczonej trasy”
Słownik Języka Polskiego PWN

„Kto chodzi na skróty, ten nie śpi w domu”
Mądrość ludowa ;-)

„Nie ma drogi na skróty, do miejsca do którego, warto dojść”
Edith

Notatki zatytułowane „na przełaj” przygotowane są jako materialy pomocnicze do moich zajęć oraz jako materialy pomocnicze dla mnie. Każdy może z nich korzystać, pamiętając jednak że:

- Notatki zostaly przygotowane tak, bym ułatwić wykonywanie pewnych analiz w R, nacisk został położony na poznanie pakietu R jako narzędzia do wykonywania danych analiz.
- Notatki NIE ZOSTAŁY przygotowane tak by uczyć się z nich metodologii. NIE SĄ tutaj dyskutowane zagadnienia teoretyczne, nie ma tu wyprowadzeń, nie ma sprawdzenia niektórych założeń. Od tego są książki, staram się w bibliografii zamieszać lepsze pozycje które udało mi się znaleźć.
- Notatki przygotowuję ucząc się danych metod, NIE SĄ więc to materiały zebrane przez eksperta, czasami nie są to nawet materiały czytane po raz drugi. Chętnie usłyszę co w takich notakach powinno się znaleźć dodatkowo, co jest niejasne lub co jest błędem tak merytorycznym jak i językowym.

Rozdział 2

Na przełaj przez wybrane metody Data Mining

Analiza danych to nie tylko klasyczna statystyka z zagadnieniami estymacji i testowania (najczęściej wykładanymi na standardowych kursach statystyki). Znaczny zbiór metod analizy danych nazywany technikami eksploracji danych lub data mining dotyczy zagadnień klasyfikacji, identyfikacji, analizy skupień oraz modelowania złożonych procesów. Właśnie te metody będą nas interesować w poniższym rozdziale.

To paraphrase provocatively, 'machine learning is statistics minus any checking of models and assumptions'.

Brian D. Ripley (about the difference between machine learning and statistics) fortune(50)

Data mining to szybko rosnąca grupa metod analizy danych rozwijana nie tylko przez statystyków ale głównie przez biologów, genetyków, cybernetyków, informatyków, ekonomistów, osoby pracujące nad rozpoznawaniem obrazów, myśli i wiele innych grup zawodowych. Podobnie jak w poprzednich rozdziałach nie będziemy dokładnie omawiać poszczególnych algorytmów ani szczegółowo dyskutować kontekstu aplikacyjnego danej metody. Zakładamy, że czytelnik zna te metody, zna ich założenia i podstawy teoretyczne ale jest zainteresowany w jakich funkcjach pakietu R są one zaimplementowane. Stąd też skrótowe traktowanie wielu zagadnień. Osoby szukające więcej informacji o sposobie działania poszczególnych algorytmów znajdą z pewnością wiele pozycji poświęconym konkretnym metodom, szczególnie polecam książki [26] oraz [25].

Przedstawione metody zostały podzielone na trzy grupy. Pierwsza z nich to metody związane z zagadnieniem redukcji wymiaru. Te metody są również wykorzystywane do ekstrakcji cech oraz do wstępnej obróbki danych, stanowią więc często etap pośredni w analizach. Druga grupa metod związana jest z zagadnieniem analizy skupień (w innej nomenklaturze nazywanym uczeniem bez nadzoru). Przedstawiane będą metody hierarchiczne oraz grupujące. Ostatnia grupa metod związana jest z zagadnieniem dyskryminacji (w innej nomenklaturze - uczeniem pod nadzorem lub klasyfikacją).

Rozdział 3

Redukcja wymiaru

Metody redukcji wymiaru umożliwiają przedstawienie obserwacji w przestrzeni o zadanym wymiarze, niższym niż w przypadku oryginalnych danych.

Przedstawimy wybrane metody na przykładzie danych z różnych województw (zbiór danych daneGUS, patrz ostatni rozdział). Dla każdego województw zebrano informację o liczbie studentów w tych województwach w rozbiciu na 16 grup kierunków. Każde województwo jest więc opisane wektorem 16 liczb.

Przypuśćmy, że chcemy przedstawić graficznie te województwa. Jak uwzględnić wszystkie 16 parametrów? Jednym z rozwiązań jest użycie metod redukcji wymiaru i liczby zmiennych z 16 do 2. Opisanei każdego z województw dwoma liczbami ułatwi przedstawianie graficzne tych województw.

Więc do dzieła!

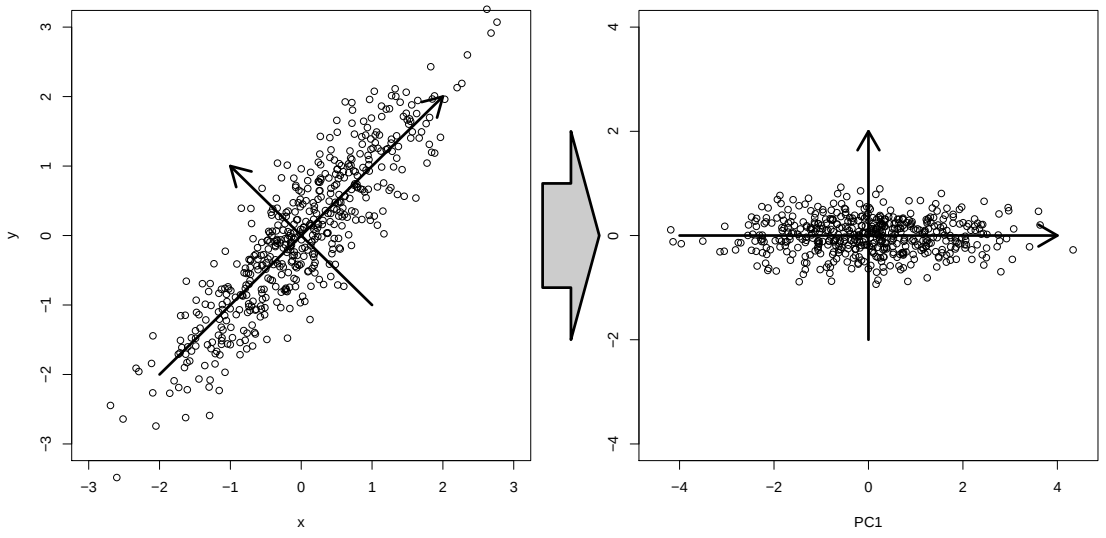
Redukcja wymiaru często jest pośrednim etapem w zagadnieniu klasyfikacji, analizy skupień czy regresji. W określonych sytuacjach pozwala na poprawę skuteczności tych metod, zwiększa stabilność a czasem pozwala na uwzględnienie w analizach dużej liczby zmiennych. Jest też popularnie wykorzystywaną metodą do wizualizacji wielowymiarowych zmiennych, dane są redukowane do przestrzeni dwuwymiarowej, w której już łatwo je przedstawić na wykresie. Metody z tej grupy są również nazywane metodami ekstrakcji cech, ponieważ w wyniku redukcji wymiaru tworzone są nowe cechy, które mogą być wykorzystane do innych zagadnień.

3.1 Analiza składowych głównych (PCA, ang. Principal Components Analysis)

Analiza składowych głównych służy do wyznaczania nowych zmiennych, których możliwie mały podzbiór będzie mówił możliwie dużo o całej zmienności w zbiorze danych. Nowy zbiór zmiennych będzie tworzył bazę ortogonalną w przestrzeni cech. Zmienne będą wybierane w ten sposób by pierwsza zmienna odwzorowywała możliwie dużo zmienności w danych (po zrzutowaniu obserwacji na ten wektor, chcemy by wariancja rzutów była najwyższa). Po wyznaczeniu pierwszej zmiennej wyznaczamy

drugą, tak by była ortogonalna do pierwszej, i wyjaśniała możliwie dużo pozostałej zmienności, kolejną zmienną wybieramy tak by była ortogonalna do dwóch pierwszych itd.

Tak uzyskany zbiór wektorów tworzy bazę ortogonalną w przestrzeni cech, a co więcej pierwsze współrzędne wyjaśniają większość zmienności w obserwacjach. Celem metody składowych głównych jest więc znalezienie transformacji układu współrzędnych, która lepiej opisze zmienność pomiędzy obserwacjami. Przykład takiej transformacji pokazujemy na rysunku 3.1. Przedstawiamy obserwacje w oryginalnym układzie współrzędnych (lewy rysunek) i w nowym układzie współrzędnych (prawy rysunek).



Rysunek 3.1: Przykład transformacji zmiennych z użyciem metody PCA.

W pakiecie R dostępnych jest kilka implementacji metody składowych głównych. Przedstawione poniżej najpopularniejsze wersje znajdują się w funkcjach `prcomp(stats)` i `princomp(stats)` z pakietu `stats`. Inne popularne wersje znaleźć można w funkcjach `PCA(FactoMineR)`, `cmdscale(MASS)` lub `pca(pcurve)`.

Poszczególne implementacje różnią się metodami wykorzystanymi do znalezienia nowego układu zmiennych. W przypadku funkcji `prcomp()` nowe zmienne wyznaczane są z z użyciem dekompozycji na wartości osobliwe SVD. Ten sposób wyznaczania składowych głównych jest zalecany z uwagi na dużą dokładność numeryczną. W funkcji `princomp()` składowe główne są wyznaczane poprzez wektory własne macierzy kowariancji pomiędzy zmiennymi, używana jest więc dekompozycja spektralna. Teoretyczne właściwości wyznaczonych składowych głównych będą identyczne, jednak w określonych sytuacjach otrzymane wyniki dla poszczególnych funkcji mogą się różnić.

Poniższy kod pokazuje w jaki sposób działa funkcja `princomp()`. Wyznaczane są wektory własne macierzy kowariancji, tworzą one macierz przekształcenia dla danych.

```
kowariancja = cov(dane)
eig = eigen(kowariancja)
noweDane = dane %*% eig$vectors
```

Poniższy kod pokazuje w jaki sposób działa funkcja `prcomp()`. Wykorzystywana jest dekompozycja SVD.

```
svdr = svd(dane)
noweDane = dane %*% svdr$v
```

Obu funkcji do wyznaczania składowych głównych używa się w podobny sposób, kolejne argumenty określają zbiór danych (można wskazać macierz, ramkę danych, lub formułę określającą które zmienne mają być transformowane) oraz informacje czy zmienne powinny być uprzednio wycentrowane i przeskalowane. To czy dane przed wykonaniem analizy składowych głównych mają być przeskalowane zależy od rozwiązywanego problemu, w większości sytuacji skalowanie pozwala usunąć wpływ takich artefaktów jak różne jednostki dla poszczególnych zmiennych. W obiektach przekazywanych jako wynik przez obie funkcje przechowywane są podobne informacje, choć w polach o różnej nazwie. Wynikiem funkcji `prcomp()` jest obiekt klasy `prcomp`, którego pola są wymienione w tabeli 3.1.

Tabela 3.1: Pola obiektu klasy *prcomp*.

`$sdev`	Wektor odchyleń standardowych dla obserwacji. Kolejne zmienne odpowiadają odchyleniom standardowym liczonym dla kolejnych składowych głównych.
`$rotation`	Macierz obrotu przekształcająca oryginalny układ współrzędnych w nowy układ współrzędnych.
`$center`	Wektor wartości wykorzystanych przy centrowaniu obserwacji.
`$scale`	Wektor wartości wykorzystanych przy skalowaniu obserwacji.
`$x`	Macierz współrzędnych kolejnych obserwacji w nowym układzie współrzędnych, macierz ta ma identyczne wymiary co oryginalny zbiór zmiennych.

Dla obiektów klasy `prcomp` dostępne są przeciążone wersje funkcji `plot()`, `summary()`, `biplot()`. Poniżej przedstawiamy przykładowe wywołanie tych funkcji. Graficzny wynik ich działania jest przedstawiony na rysunku 3.3. Lewy rysunek przedstawia wynik dla funkcji `plot()` a prawy przedstawia wynik funkcji `biplot()` wykonanych dla argumentu klasy `prcomp`. Na lewym rysunku przedstawione są warian-

cje wyjaśnione przez kolejne wektory nowego układu współrzędnych. Ta informacja jest podawana również przez funkcje `summary()`. Prawy rysunek przedstawia biplot, na którym umieszczone są dwa wykresy. Jeden przedstawia indeksy obserwacji przedstawione na układzie współrzędnych określonych przez dwie pierwsze składowe główne (w tym przypadku dwie współrzędne wyjaśniają około 50% całej zmienności). Drugi rysunek przedstawia kierunki w których działają oryginalne zmienne, innymi słowy przedstawiają jak wartość danej zmiennej wpływa na wartości dwóch pierwszych składowych głównych.

Jeżeli wektory mają przeciwne zwroty to dane zmienne są ujemnie skorelowane (nie można jednak ocenić wartości korelacji), jeżeli zwroty są prostopadłe to zmienne są nieskorelowane, a jeżeli zwroty są bliskie to zmienne są dodatnio skorelowane.

```
> # przygotowujemy dane, usuwamy zmienne jakościowe i brakujące przypadki
> dane  = na.omit(daneO[,-c(4,5,6,7)])
> # wykonujemy analizę składowych głównych, normalizując wcześniej zmienne
> wynik = prcomp(dane, scale=T)
>
> # jak wygląda obiekt z wynikami od środka
> str(wynik)
List of 5
 $ sdev    : num [1:5] 1.153 1.068 0.963 0.916 0.873
 $ rotation: num [1:5, 1:5]  0.5016  0.0935 -0.4244  0.4878 -0.5671 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:5] "Wiek" "Rozmiar.guza" "Wezly.chlonne" "Okres.bez.
     wznowy" ...
  .. ..$ : chr [1:5] "PC1" "PC2" "PC3" "PC4" ...
 $ center  : Named num [1:5]   45.417    1.271    0.417   37.406 2640.896
  ..- attr(*, "names")= chr [1:5] "Wiek" "Rozmiar.guza" "Wezly.chlonne" "
     Okres.bez.wznowy" ...
 $ scale   : Named num [1:5]    6.206    0.447    0.496    9.527 3616.045
  ..- attr(*, "names")= chr [1:5] "Wiek" "Rozmiar.guza" "Wezly.chlonne" "
     Okres.bez.wznowy" ...
 $ x       : num [1:96, 1:5] -1.5446  0.0105 -1.4565 -1.2352 -1.2541 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:96] "1" "2" "3" "4" ...
  .. ..$ : chr [1:5] "PC1" "PC2" "PC3" "PC4" ...
 - attr(*, "class")= chr "prcomp"
> # ten wykres przedstawia ile wariancji jest wyjaśnione przez kolejne
    zmienne
> plot(wynik)
> # zamiast obrazka możemy tę informację mieć przedstawioną jako ramkę
    danych
> summary(wynik)
Importance of components:
                          PC1   PC2   PC3   PC4   PC5
```

```
Standard deviation     1.153 1.068 0.963 0.916 0.873
Proportion of Variance 0.266 0.228 0.185 0.168 0.153
Cumulative Proportion  0.266 0.494 0.680 0.847 1.000
> # narysujmy biplot dla tych wyników
> biplot(wynik)
```

I jeszcze przykład dla danych GUSowskich

```
> # przygotowujemy dane, wybieramy tylko kolumny dotyczące studentów
> dane  = daneGUS[,5:19]

> # wykonujemy analizę składowych głównych
> wynik = prcomp(dane, scale=T)

> # ten wykres przedstawia ile wariancji jest wyjaśnione przez kolejne
    zmienne
> plot(wynik)

> # zamiast obrazka możemy tę informację mieć przedstawioną jako ramkę
    danych
> summary(wynik)
Importance of components:
                          PC1   PC2    PC3    PC4    PC5
Standard deviation     3.327 1.199 0.8443 0.7418 0.5666 ...
Proportion of Variance 0.738 0.096 0.0475 0.0367 0.0214 ...
Cumulative Proportion  0.738 0.834 0.8815 0.9182 0.9396 ...

> # narysujmy biplot dla tych wyników
> biplot(wynik)
```

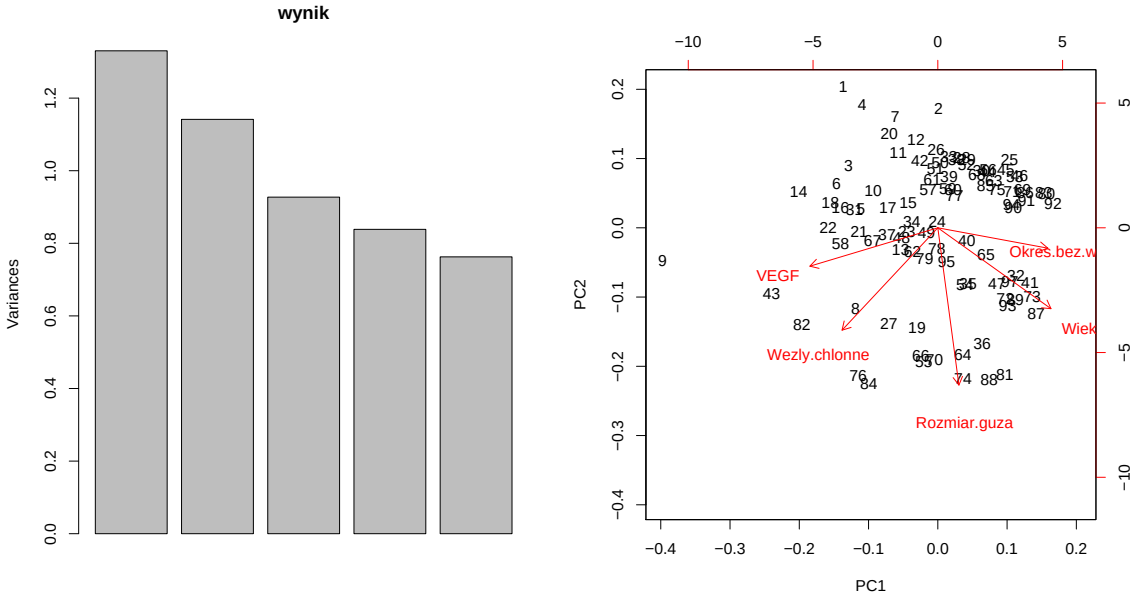
Zobaczmy jak wygląda macierz przekształcenia. Można z niej odczytać w jaki sposób poszczególne współrzędne wpływają na kolejne składowe.

W przypadku pierwszej składowej współczynniki przy praktycznie każdej zmiennej wynoszą około -0.25. W przybliżeniu oznacza to, że pierwsza składowa będzie odpowiadała -łącznej liczbie studentów w danym województwie. A więc im więcej studentów tym mniejsza wartość pierwszej składowej.

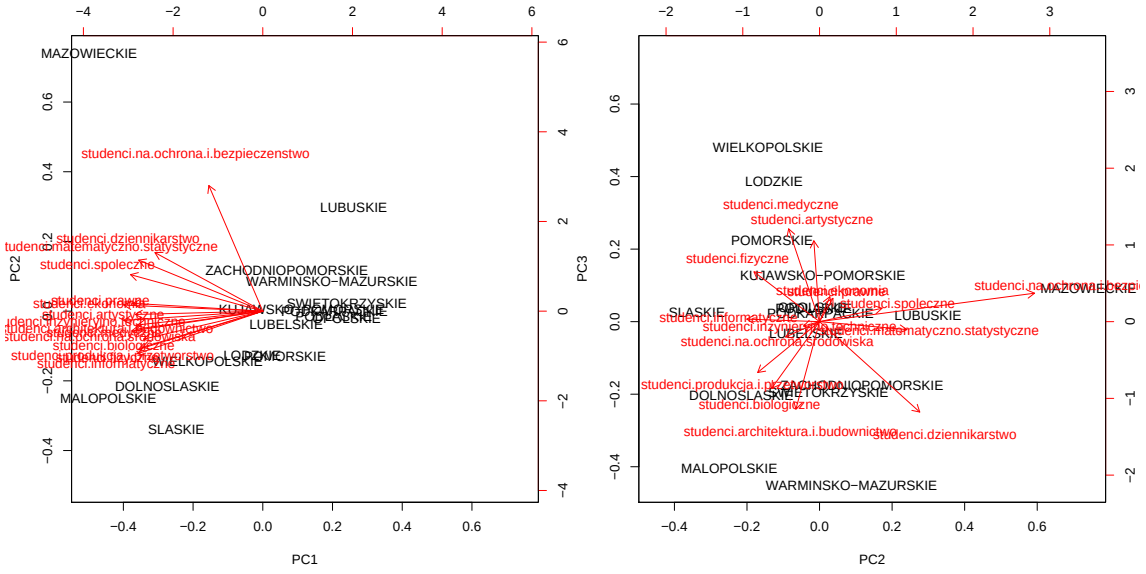
Druga składowa jest już ciekawsza, ponieważ różnym województwom odpowiadają różne współczynniki. Województwa o dużych wartościach na drugiej składowej to województwa z „nadreprezentacją" studentów z kierunków społecznych, dziennikarstwa, matematyki i ochrony a „niedomiarze" studentów z nauk biologicznych, fizycznych, informatycznych i produkcji.

```
> # macierz przekształcenia
```

```
> wynik$rotation[,1:4]
                                        PC1       PC2       PC3       PC4
studenci.artystyczne                -0.2668 -0.01935  0.39012  0.06339
studenci.spoleczne                  -0.2769  0.21209  0.06445 -0.18107
studenci.ekonomia                   -0.2908  0.03508  0.12031  0.01692
studenci.prawne                     -0.2724  0.04551  0.11257  0.27527
studenci.dziennikarstwo             -0.2258  0.34018 -0.43605  0.37212
studenci.biologiczne                -0.2496 -0.16285 -0.32214  0.40296
studenci.fizyczne                   -0.2604 -0.22242  0.24023  0.28846
studenci.matematyczno.statystyczne  -0.2599  0.29657 -0.03841  0.13851
studenci.informatyczne              -0.2621 -0.24703  0.01225 -0.33688
studenci.medyczne                   -0.2654 -0.10490  0.44763  0.09636
studenci.inzynieryjno.techniczne    -0.2913 -0.05414 -0.02117 -0.08996
studenci.produkcja.i.przetworstwo   -0.2521 -0.20996 -0.24515 -0.39126
studenci.architektura.i.budownictwo -0.2621 -0.08025 -0.42188 -0.15234
studenci.na.ochrona.srodowiska      -0.2744 -0.11530 -0.07394 -0.29601
studenci.na.ochrona.i.bezpieczenstwo -0.1129  0.72998  0.13728 -0.29845
```



Rysunek 3.2: Graficzna reprezenacja wyników funkcji *prcomp()*.



Rysunek 3.3: Graficzna reprezenacja wyników PCA dla danych GUSowych.

3.2 Nieliniowe skalowanie wielowymiarowe (Sammon Mapping)

W przypadku metody PCA nowe współrzędne konstruowano tak, by były one kombinacjami liniowymi oryginalnych danych. To oczywiście nie jest jedyna możliwość konstrukcji nowych zmiennych. Postawmy zagadnienie skalowania następująco.

Dane: mamy n obiektów oraz macierz odległości pomiędzy każdą parą obiektów. Oznaczmy przez d_{ij} odległość pomiędzy obiektem i-tym i j-tym.

Szukane: reprezentacja obiektów w przestrzeni k wymiarowej, tak by zminimalizować

$$stress = \frac{\sum_{i,j}(d_{ij} - \tilde{d_{ij}})^2/d_{ij}}{\sum_{i,j} d_{ij}},$$

gdzie $\tilde{d_{ij}}$ to odległość pomiędzy obiektami i i j w nowej k-wymiarowej przestrzeni.

Innymi słowy, szukamy (niekoniecznie liniowego) przekształcenia, które możliwie najwierniej (w sensie ważonego błędu kwadratowego) zachowa odległości pomiędzy obiektami. Takie przekształcenie poszukiwane jest iteracyjnie.

Jak to zrobić w R? Można np. używając funkcji `sammon(MASS)`. Pierwszym argumentem tej funkcji powinna być macierz odległości pomiędzy obiektami (np. wynik funkcji `dist()`) a argument k określa na iluwymiarową przestrzeń chcemy skalować dane (domyślnie $k = 2$).

```
> # wyznaczamy macierz odległości
```

```
> odleglosci = dist(dane)
> # wyznaczamy nowe współrzędne w przestrzeni dwuwymiarowej
> noweSammon = sammon(odleglosci, k=2, trace=FALSE)
> # jak wyglądają nowe współrzędne
> head(noweSammon$points)
                         [,1]       [,2]
DOLNOSLASKIE        15211.180 -3403.2348
KUJAWSKO-POMORSKIE  -7063.770 -3448.9048
LUBELSKIE           -4912.805 -1058.6175
LUBUSKIE           -12775.224 -3090.0001
LODZKIE              1251.418   916.5612
MALOPOLSKIE         21526.869 -1963.5498
```

Tabela 3.2: Pola obiektu będącego wynikiem funkcji *sammon()*.

`$points`	Macierz współrzędnych obiektów w nowej k-wymiarowej przestrzeni.
`$stress`	Uzyskana wartość optymalizowanego parametru stresu.

3.3 Skalowanie wielowymiarowe Kruskalla (MDS, ang. Multidimensional Scaling)

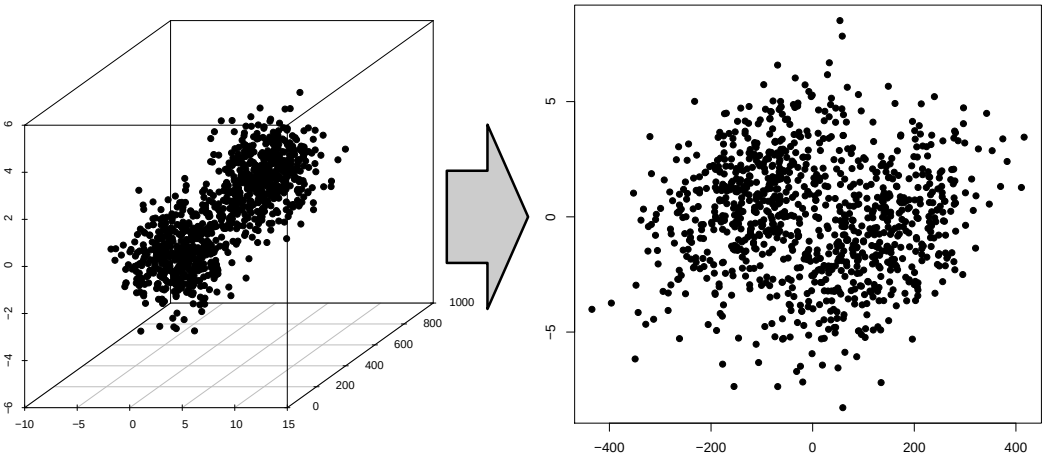
Jak już wspominaliśmy, metody redukcji wymiaru są często wykorzystywane do wizualizacji danych. W przypadku analizy składowych głównych po znalezieniu współrzędnych obiektów w nowej bazie wystarczy wziąć dwie pierwsze współrzędne by móc przedstawiać zbiór obserwacji na wykresie dwuwymiarowym, trzy pierwsze by móc przedstawić zbiór obserwacji na wykresie trójwymiarowym itp. Wadą analizy składowych głównych jest uwzględnianie wyłącznie zmiennych ilościowych. Kolejnym minusem jest konieczność posiadania wartości pomiarów dla kolejnych zmiennych, nie można tej metody użyć w sytuacji gdy mamy wyłącznie informacje o podobieństwie lub odległości pomiędzy obiektami.

Metody skalowania Sammona i Kruskala nie mają tych wad. Są to metody ekstrakcji cech, na podstawie macierzy odległości lub macierzy niepodobieństwa pomiędzy obiektami. Celem tych metod jest wyznaczenie współrzędnych w nowym układzie współrzędnych, w taki sposób by odległości pomiędzy obiektami w nowym układzie współrzędnych były podobne do oryginalnych odległości pomiędzy obiektami. Przykład skalowania wielowymiarowego przedstawiliśmy na rysunku 3.4.

W przypadku skalowania Kruskala minimalizowana jest wartość

$$stress = \frac{\sum_{i,j}(f(d_{ij}) - \tilde{d_{ij}})^2}{\sum_{i,j} f(d_{ij})^2},$$

gdzie $\tilde{d_{ij}}$ to odległość pomiędzy obiektami i i j w nowej k-wymiarowej przestrzeni a d_{ij} to oryginalne odległości pomiędzy obiektami przekształcone przez pewną monotoniczną funkcję $f()$ (więc d_{ij} i $\tilde{d}_{ij}$ mogą być w różnych skalach!).



Rysunek 3.4: Przykład skalowania wielowymiarowego.

Skalowanie wielowymiarowe jest w R dostępne w kilku funkcjach:

- funkcja `isoMDS(MASS)` wyznacza niemetryczne skalowanie Kruskala,
- funkcja `sammon(MASS)` wyznacza niemetryczne skalowanie Sammona (patrz poprzedni podrozdział [TODO: uspójnić!!]),
- funkcja `cmdscale(stats)` wyznacza skalowanie metryczne inaczej PCA (patrz poprzedni podrozdział [TODO: uspójnić!!]).

Poniżej przedstawimy przykłady użycia dla funkcji `isoMDS()`, z pozostałych korzysta się podobnie.

Najważniejszym argumentem wejściowym do algorytmu skalowania wielowymiarowego jest macierz odległości pomiędzy obserwacjami. Wyznaczyć ją można np. funkcją `dist(stats)`. W funkcji `dist()` zaimplementowane są wszystkie popularne metody liczenia odległości pomiędzy obserwacjami, w tym odległość euklidesowa (najpopularniejsza, odpowiadająca sumie kwadratów różnic poszczególnych współrzędnych, tej odległości odpowiada argument `method="euclidean"`), odległość mak-

This has been discussed before in this list, and Ripley said „no, no!". I do it all the time, but only in secrecy.

Jari Oksanen (about replacing zero distances with tiny values for isoMDS()) fortune(163)

simum (argument `method="maximum"`), odległość Manhattan, nazywana też odległością taksówkową lub miejską (suma modułów różnic pomiędzy współrzędnymi, argument `method="manhattan"`), odległość Mińkowskiego (argument `method="minkowski"`) oraz kilka innych mniej popularnych odległości. Jeżeli w zbiorze danych znajdują się zmienne jakościowe to funkcja `dist()` sobie z nimi nie poradzi. W takiej sytuacji lepiej wykorzystać funkcję `daisy(cluster)` wyznaczającą macierz niepodobieństwa pomiędzy obiektami. Funkcja `daisy()` uwzględnia również zmienne jakościowe (poniżej przedstawiamy przykład użycia). Macierz odległości jest obiektem klasy `dist()` i nie jest pamiętana jako macierz, a jedynie jako połowa macierzy (ponieważ odległość jest symetryczna szkoda pamięci na przechowywanie nadmiarowych danych). Jeżeli potrzebujemy przekształcić obiekt `dist()` na macierz to możemy wykorzystać funkcję `as.matrix()`.

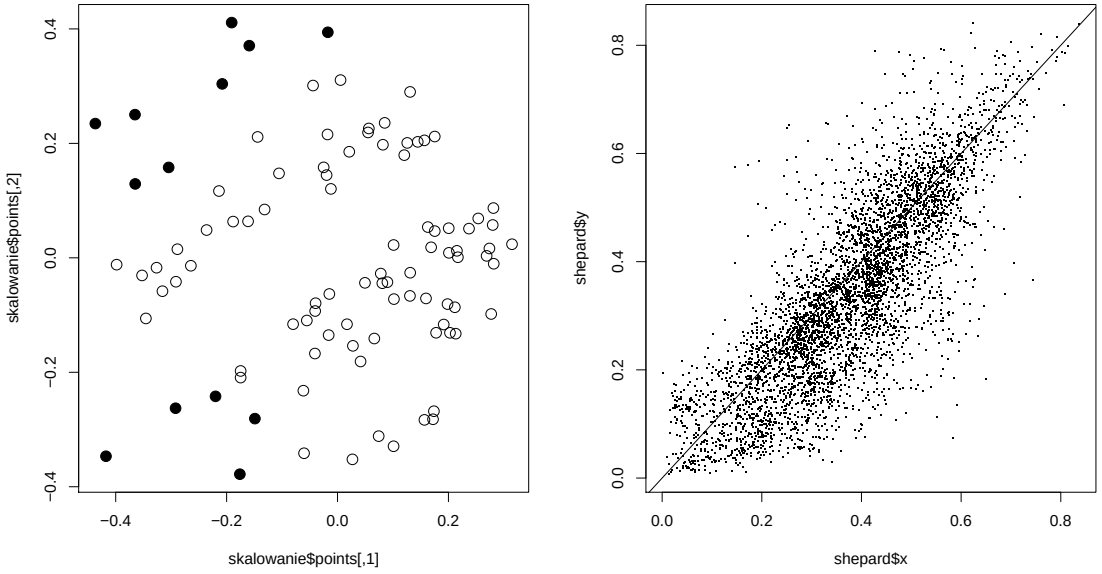
Wynikiem algorytmu skalowania wielowymiarowego są współrzędne obserwacji w pewnym nowym układzie współrzędnych. Możemy wybrać wymiar przestrzeni na jaką mają być przeskalowane dane (argument `k` funkcji `isoMDS`). Z pewnością po wykonaniu skalowania interesować nas będzie na ile skalowanie zachowało odległości pomiędzy obiektami, czy dużo jest znacznych zniekształceń. Do oceny wyników skalowania wykorzystać można wykres Sheparda przedstawiający na jednej osi oryginalne odległości pomiędzy obiektami a na drugiej osi odległości w nowym układzie współrzędnych. Do wyznaczenia obu wektorów odległości służy funkcja `Shepard(MASS)`, można też skorzystać z wrappera na tę funkcję, czyli z funkcji `stressplot(vegan)`.

Wrapper to funkcja agregująca i udostępniająca funkcjonalności innej funkcji.

Poniżej przedstawiamy przykład skalowania wielowymiarowego. Wykorzystamy tę metodę do przedstawienia za pomocą dwuwymiarowego wykresu podobieństw pomiędzy pacjentkami ze zbioru danych `dane0`. Graficzny wynik tych analiz jest przedstawiony na rysunku 3.5. Lewy rysunek przedstawia pacjentki w nowym dwuwymiarowym układzie współrzędnych, w tym przypadku pacjentki przedstawiane są jako punkty. Wypełniony punkt oznacza dla niepowodzenie leczenia a więc wznowę, a pusty w środku w oznacza wyleczenie pozytywne (widzimy, że pacjentki z niepowodzeniami grupują się blisko siebie). Ciekawym było by naniesienie na ten wykres nazwisk pacjentek i porównanie, które pacjentki pod względem zmierzonych wartości były do siebie podobne. Prawy rysunek przedstawia dokładność skalowania, a więc jak oryginalne odległości mają się do odległości w nowym układzie współrzędnych.

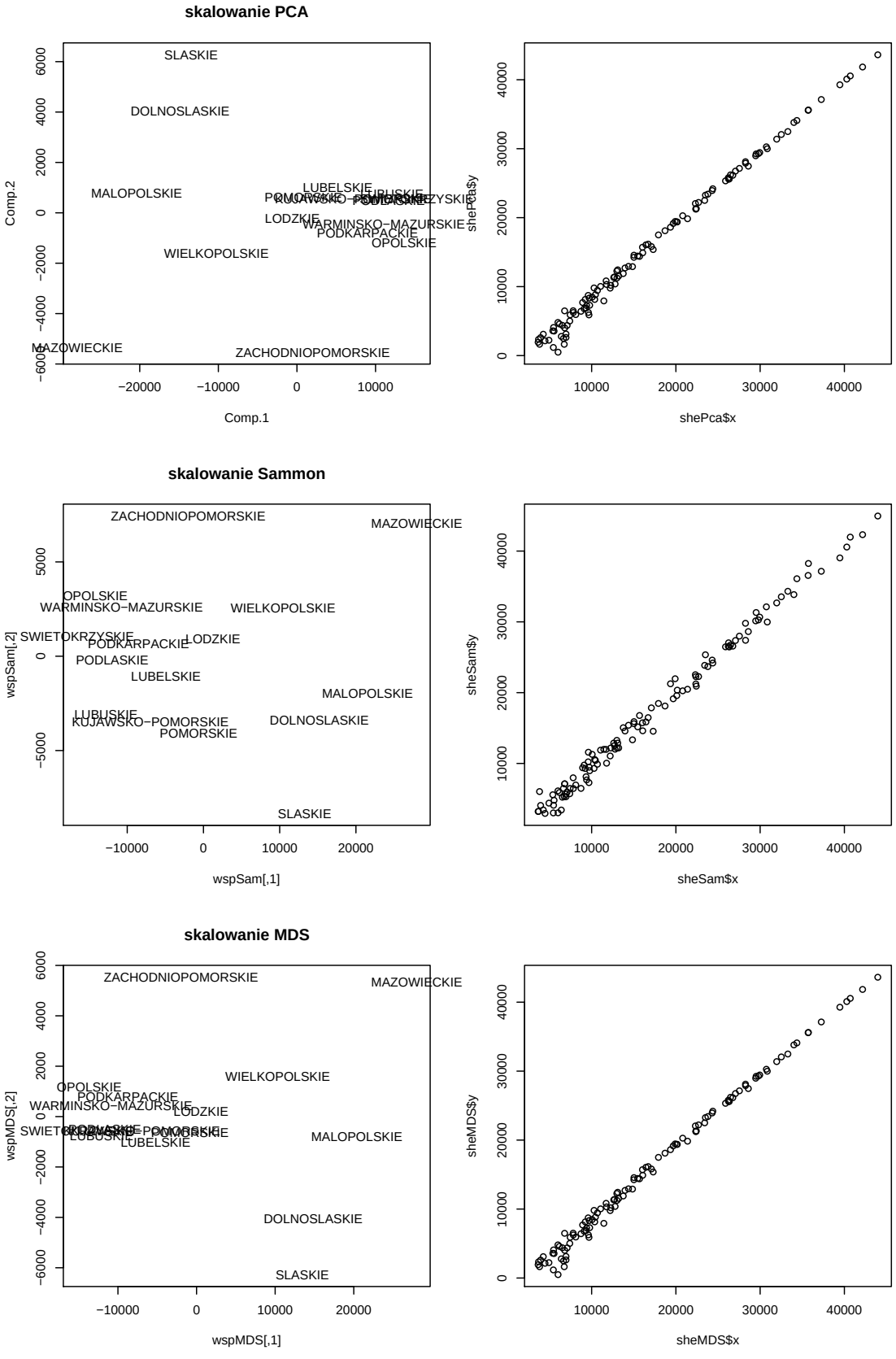
```
> # konstruujemy macierz niepodobieństwa pomiędzy pacjentkami, również
    zmienne jakościowe są uwzględnione
> niepodobienstwa = daisy(dane0)
Warning message:
In daisy(dane0) : binary variable(s) 2, 3 treated as interval scaled
> # przeprowadzamy skalowanie niemetryczne, skalujemy do przestrzeni o~
    dwóch wymiarach
> skalowanie = isoMDS(niepodobienstwa, k=2)
initial  value 30.138174
```

```
iter   5 value 25.846808
final  value 25.531983
converged
> # obiekt wynikowy zawiera współrzędne obserwacji w~nowym układzie
    współrzędnych
> str(skalowanie)
List of 2
 $ points: num [1:97, 1:2]  0.1011  0.3147 -0.1055  0.2811 -0.0604 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:97] "1" "2" "3" "4" ...
  .. ..$ : NULL
 $ stress: num 25.5
>
> # konstruujemy wektor pomocniczy do rysowania
> ksztalty = ifelse(daneO$Niepowodzenia=="brak", 21, 19)
> # rysujemy pacjentki w~nowym układzie współrzędnych
> plot(skalowanie$points, type = "p", pch=ksztalty, cex=1.5)
>
> # rysujemy również diagram Sheparda
> shepard <- Shepard(niepodobienstwa, skalowanie$points)
> plot(shepard, pch = ".")
> abline(0,1)
```



Rysunek 3.5: Graficzna reprezentacja wyników funkcji *isoMDS()* i *Shepard()*.

Zobaczmy jak wyglądają dane o województwach po przeskalowaniu różnymi metodami. Na wykresie 3.6 po lewej stronie przedstawiamy przeskalowane dane a po prawej wykresy Sheparda. [TODO: czy ten obrazek jest przydatny? Opisac dokladniej lub usunac]



Rysunek 3.6: Graficzna reprezentacja wyników różnych funkcji skalowania, na przykładach danych GUS.

Rozdział 4

Analiza skupień

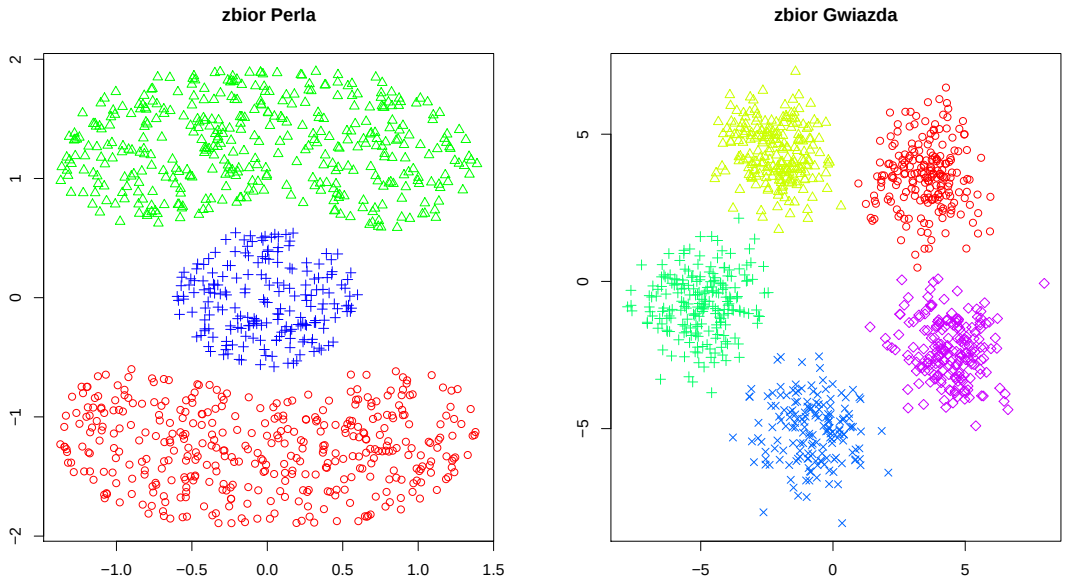
Analiza skupień to zbiór metod pozwalających na wyróżnienie zbiorów obserwacji (nazywanych skupieniami lub klastrami) podobnych do siebie. Proces szukania podziału na grupy, nazywany jest czasem klastrowaniem. W pakiecie R dostępnych jest bardzo wiele metod do przeprowadzania analizy skupień. Poniżej omówimy jedynie kilka wybranych funkcji z pakietów `cluster` i `stats`. Osoby zainteresowane tym tematem powinny przyjrzeć się również funkcjom z pakietów `flexclust` oraz `mclust02`.

Wyniki działania wybranych procedur analizy skupień przedstawimy ma przykładzie zbioru danych benchmarkowych. W pakiecie `mlbench` (skrót od Machine Learning Benchmark Problems) umieszczonych jest wiele ciekawych zbiorów danych wykorzystywanych do testowania właściwości algorytmów dyskryminacji lub analizy skupień. W tym pakiecie znajdują się zbiory rzeczywistych danych, jak również funkcje do generowania zbiorów danych o określonych kształtach lub właściwościach. Dwa zbiory wygenerowanych danych na których będziemy testować metody analizy skupień przedstawione są na rysunku 4.1. Zostały one wygenerowane funkcjami `mlbench.cassini(mlbench)` i `mlbench.2dnormals(mlbench)`.

4.1 Metoda k-średnich

Celem tej metody jest podział zbioru danych na k klastrów. Dobry podział to taki, w którym suma odległości obserwacji należących do klastra jest znacznie mniejsza od sumie odległości obserwacji pomiędzy klastrami. Metoda k-średnich polega na wyznaczeniu współrzędnych k punktów, które zostaną uznane za środki klastrów. Obserwacja będzie należała do tego klastra, którego środek jest najbliżej niej.

Metoda k-średnich jest zaimplementowana w funkcji `kmeans(stats)`. Pierwszym argumentem tej funkcji jest ramka danych określająca wartości zmiennych dla kolejnych obserwacji. Drugim argumentem może być pojedyncza liczba określająca ile klastrów chcemy identyfikować (w tym przypadku środki klastrów będą wyznaczone iteracyjnym algorytmem) lub wektor środków klastrów. Algorytm wyboru środków klastrów jest algorytmem zrandomizowanym, może też dawać różne wyniki nawet



Rysunek 4.1: Dane, na których będziemy przedstawiać metody analizy skupień.

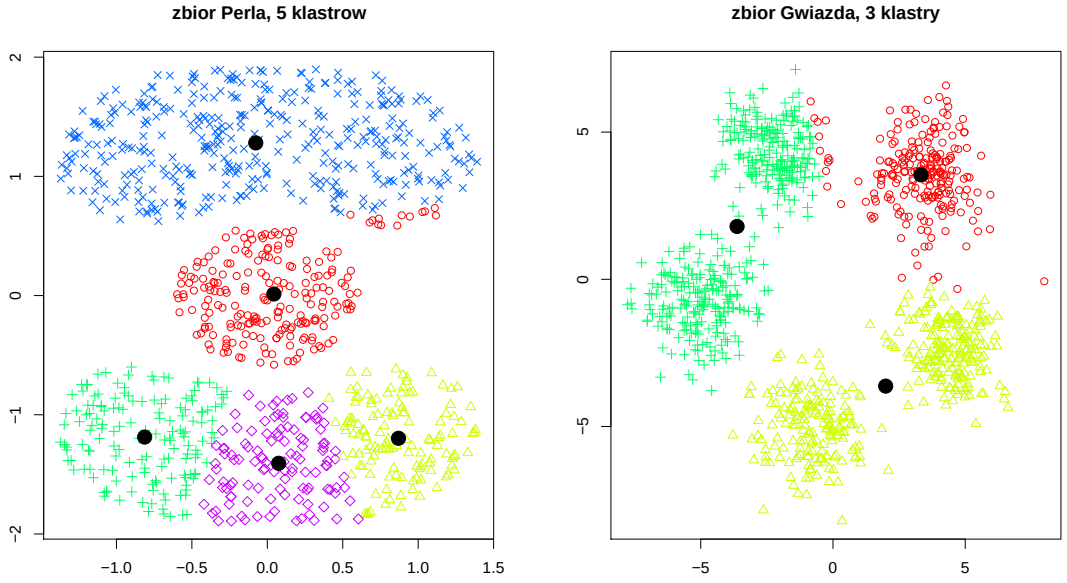
na tym samym zbiorze danych! Dlatego też zalecane jest uruchomienie kilkukrotne tego algorytmu oraz wybranie najlepsze go podziału na klastry. Można to zrobić też automatycznie, określając argument `nstart` funkcji `kmeans()`.

Poniżej prezentujemy przykład użycia funkcji `kmeans()`. Wyniki analizy skupień przedstawione są graficznie na rysunku 4.2. Różne klastry zaznaczono punktami o różnych kształtach. Czarne pełne punkty wskazują na środki znalezionych klastrów. Oczywiście właściwszym byłoby dopasowanie do lewego przykładu 3 klastrów, a do prawego 5 klastrów. Na przedstawionych przykładach możemy prześledzić ci się dzieje, jeżeli źle określimy liczbę klastrów (czytelnik powinien spróbować powtórzyć te analizy, wyniki najprawdopodobniej otrzyma inne!).

```
> # szukamy 5 klastrow, nie trafiło się nam najlepsze dopasowanie
> klaster = kmeans(zbiorPerla,5)
> # jak wygląda wynik w środku?
> # pole $cluster określa numer klastra dla kolejnych punktów, $centers
    określa współrzędne środków klastrów
> str(klaster)
List of 4
 $ cluster : int [1:1000] 3 3 3 3 3 3 3 3 3 3 ...
 $ centers : num [1:5, 1:2]  0.03203 -0.00749 -0.08380 -0.81601  0.91808
    ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:5] "1" "2" "3" "4" ...
```

```
  .. ..$ : NULL
 $ withinss: num [1:5]  22.4  10.9 126.9  11.5   9.8
 $ size    : int [1:5] 103 69 197 70 61
 - attr(*, "class")= chr "kmeans"
> # rysujemy punkty, różne klastry oznaczamy innymi kształtami punktów
> plot(zbiorPerla, pch=klaster$cluster)
> # dorysujmy środki klastrów
> points(klaster$centers, cex=2, pch=19)

> klaster = kmeans(zbiorGwiazda,2)
> plot(zbiorGwiazda, pch=klaster$cluster)
> points(klaster$centers, cex=2, pch=19)
```

Na powyższym przykładzie przedstawiliśmy pola w obiektach przekazanych przez funkcję `kmeans()`. Pole `$cluster` określa do jakiego klastra została przyporządkowana dana obserwacja, a pole `$centers` to współrzędne środków poszczególnych klastrów.



Rysunek 4.2: Graficzna prezentacja działania funkcji *kmeans()*.

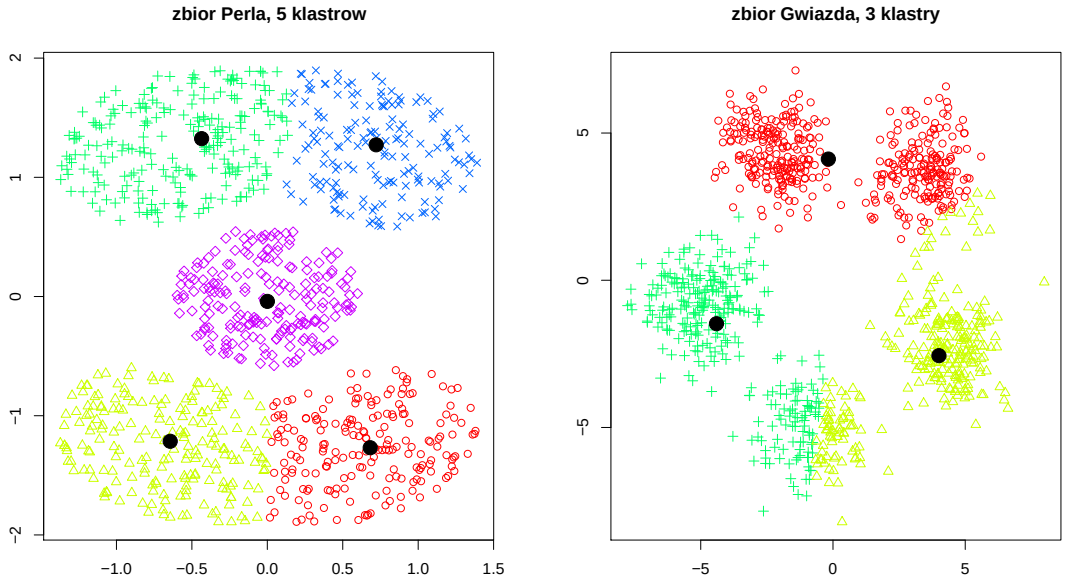
4.2 Metoda grupowania wokół centroidów (PAM, ang. Partitioning Around Medoids)

Metoda PAM działa na podobnej zasadzie jak k-średnich, z tą różnicą, że środkami klastrów są obserwacje ze zbioru danych (nazywane centroidami lub centrami klastrów). W metodzie PAM zbiór możliwych środków klastrów jest więc znacznie mniejszy, niż w metodzie k-średnich, zazwyczaj też wyniki działania metody PAM są stabilniejsze. Na rysunku 4.3 przedstawiony jest wynik działania poniższego przykładowego wywołania tej funkcji. Podobnie jak poprzednio różne klastry zaznaczono punktami o różnych kształtach. Czarne pełne punkty to środki klastrów (w tej metodzie odpowiadają przypadkom ze zbioru danych).

```
> # ponownie szukamy 5 klastrow
> klaster = pam(zbiorPerla,5)
> # jak wygląda wynik w środku?
> # pole $medoids określa współrzędne środków klastrów (wybranych
    przypadków), $in.med określa indeksy obserwacji, które są środkami
    klastrów, $clustering to wektor indeksów kolejnych klastrów, $silinfo
    to informacje o dopasowaniu danego obiektu do klastra w którym się
    znajduje (wartość silhouette)
> str(klaster)
List of 10
 $ medoids   : num [1:5, 1:2]  6.47e-01 -6.26e-01 -6.38e-01  5.87e-01
    1.59e-05 ...
 $ id.med    : int [1:5] 24 126 230 267 464
 $ clustering: int [1:1000] 1 1 2 1 1 2 1 1 1 2 ...
 $ objective : Named num [1:2] 0.526 0.461
  ..- attr(*, "names")= chr [1:2] "build" "swap"
 $ isolation : Factor w/ 3 levels "no","L","L*": 1 1 1 1 1
  ..- attr(*, "names")= chr [1:5] "1" "2" "3" "4" ...
 $ clusinfo  : num [1:5, 1:5]  87 113 101  99 100 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : NULL
  .. ..$ : chr [1:5] "size" "max_diss" "av_diss" "diameter" ...
 $ silinfo   :List of 3
  ..$ widths         : num [1:1000, 1:3] 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:500] "12" "96" "133" "155" ...
  .. .. ..$ : chr [1:3] "cluster" "neighbor" "sil_width"
  ..$ clus.avg.widths: num [1:5] 0.434 0.440 0.482 0.443 0.508
  ..$ avg.width      : num 0.462
 $ diss      : NULL
 $ call      : language pam(x = zb1$x, k = 5)
```

```
 $ data      : num [1:1000, 1:2]  0.0964  0.6938 -0.5325  1.2839  0.1743
    ...
 - attr(*, "class")= chr [1:2] "pam" "partition"
> # rysujemy punkty, różne klastry oznaczamy innymi kształtami punktów
> plot(zbiorPerla, pch=klaster$clustering)
> # dorysujmy środki klastrów
> points(klaster$meoids, cex=2, pch=19)
```

Obiekt, będący wynikiem funkcji `pam()` ma sporo pól, najistotniejsze to `$medoids` ze współrzędnymi medoidów, `$id.med` z indeksami medoidów i `$clustering` z indeksami klastrów, do których zostały przypisane kolejne obserwacje.



Rysunek 4.3: Graficzna prezentacja działania funkcji *pam()*.

Metoda PAM jest obliczeniowo złożona. Może być niemożliwym obliczeniowo wykonanie jej na dużym zbiorze danych. Do klastrowania dużych zbiorów polecana jest metoda clara (Clustering Large Applications) zaimplementowana w funkcji `clara(cluster)`. Wykonuje ona wielokrotnie metodę PAM na mniejszych zbiorach danych i scala wyniki w jeden podział na klastry.

4.3 Metoda aglomeracyjnego klastrowania hierarchicznego

Klastrowanie hierarchiczne różni się od przedstawionych powyżej metod tym, że zamiast dzielić obserwacje na określoną liczbę klastrów, określa się stopień podobieństwa poszczególnych obiektów i wyznacza się drzewo odpowiadające tym podobieństwom. Do budowy takich drzew wykorzystywane są różne algorytmy.

Algorytm AGglomerative NESting (AGNES) jest metodą aglomeracyjną, co oznacza, że w pierwszym kroku każda obserwacja traktowana jest jak osobny klaster. W kolejnych krokach klastry najbardziej podobne do siebie są łączone w coraz większe klastry, tak długo aż nie powstanie tylko jeden klaster. Algorytm aglomeracyjnego klastrowania hierarchicznego jest dostępny w funkcji `agnes(cluster)`. Pierwszym argumentem może być macierz danych (podobnie jak w przypadku innych algorytmów klastrowania) określająca współrzędne poszczególnych obserwacji lub też macierz odległości pomiędzy obserwacjami, a więc obiekt klasy `dist` (jak tworzyć takie obiekty pisaliśmy w poprzednim podrozdziale). Duże znaczenie ma metoda liczenia odległości pomiędzy obserwacjami. Z reguły zmiana metody liczenia odległości (np. z euklidesowej na taksówkową) prowadzi do otrzymania zupełnie innego wyniku.

Kolejnym istotnym argumentem jest argument `method`. Określa on kolejność łączenia małych klastrów w coraz większe klastry. W każdym kroku łączone są najbliższe klastry, ale odległość pomiędzy dwoma klasterami można liczyć na trzy sposoby:

- `method="single"`, liczona jest odległość pomiędzy najbliższymi punktami każdego z klastrów, do jednego klastra dołączany jest klaster którego dowolny element jest najbliżej,
- `method="average"`, liczona jest średnia odległość pomiędzy punktami każdego z klastrów, łączone są więc klastry średnio podobnych obserwacji,
- `method="complete"`, liczona jest odległość pomiędzy najdalszymi punktami każdego z klastrów.

Użycie każdej z tych metod prowadzi do wygenerowania innego drzewa. Wyniki dla każdej z tych trzech wymienionych metod łączenia klastrów oraz dla obu zbiorów danych przedstawiamy na rysunku 4.4. Na tym rysunku przedstawione są wyniki analizy skupień dla 1000 obiektów, jeżeli analizujemy mniejszą liczbę obiektów, to na osi poziomej można odczytać nazwy poszczególnych obiektów a tym samym wizualizować, które obiekty są do siebie bardziej, a które mniej podobne (przykład takiego drzewa przedstawiliśmy na rysunku 4.5).

Wracając do rysunku 4.4 w przypadku zbioru Gwiazda sensowniejsze wyniki otrzymuje się dla metod łączenia `average` i `complete` (na drzewie można wydzielić 5 podgałęzi odpowiadającym spodziewanym skupiskom). Dla zbioru Perła najlepiej radzi sobie metoda łączenia `single` wyodrębniająca dosyć szybko trzy rozłączne

skupiska. Najczęściej wykorzystywaną metodą łączenia jest `average`, nie oznacza to że zawsze daje najlepsze wyniki.

Aby na podstawie hierarchicznego klastrowania przypisać obserwacje do określonej liczby klastrów należy drzewo przyciąć na pewnej wysokości. Do przycinania drzewa służy funkcja `cutree(stats)`, jej pierwszym argumentem jest obiekt będący wynikiem metody hierarchicznej. Kolejnym argumentem, który należy wskazać jest `k` (określa do ilu klastrów chcemy przyciąć drzewo) lub `h` (określa na jakiej wysokości chcemy przyciąć drzewo). Wysokość drzewa na której chcemy odciąć klastry można odczytać z rysunków wygenerowanych dla tego drzewa.

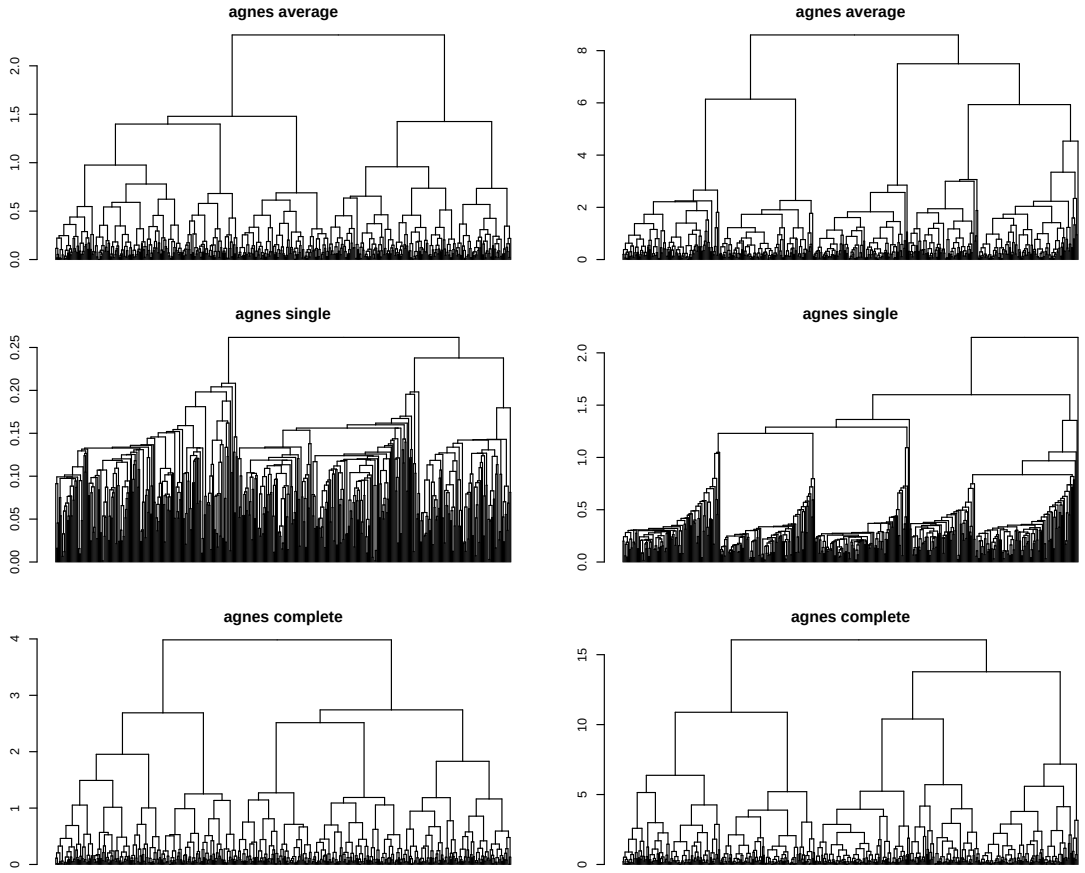
Poniżej przedstawiamy przykład wywołania funkcji `agnes()` i `cuttree()`.

```
> # wywołanie funkcji AGNES
> klaster = agnes(zbiorPerla, method="average")
> # wynik możemy narysować przeciążoną funkcją plot
> plot(klaster)
> # otrzymane drzewo możęmy przyciąć do określonej liczby klastrów
> etykietkiKlastrow = cutree(klaster, k=2)
```

4.4 Inne metody analizy skupień

Poza wymienionymi powyżej trzema metodami, w pakiecie R dostępnych jest wiele innych metod do analizy skupień. Poniżej wymienimy inne popularne.

- **Metoda hierarchiczej analizy skupień przez dzielenie.** Metoda hierarchicznego klastrowania przez łączenie działała w ten sposób, że zaczynając od małych, jednopunktowych klastrów w procesie łączenia klastrów otrzymywało się hierarchiczną zależność. Metoda analizy skupień przez dzielenie działa w przeciwnym kierunku. Zaczynając od jednego dużego klastra, w kolejnych krokach ten klaster jest dzielony na mniejsze klastry, aż do otrzymania jednoelementowych klastrów. Ta metoda jest zaimplementowana w funkcji `diana(cluster)`.
- **Metoda klastrowania rozmytego.** Jeżeli w procesie klastrowania dopuszczamy rozmytą przynależność do klastra (np. obserwacja może z pewnymi współczynnikami przynależeć do różnych klastrów) to uzasadnionym jest użycie metody klastrowania rozmytego. Ta metoda jest zaimplementowana w funkcji `fanny(cluster)`.
- **Inne metody hierarchiczego klastrowania.** Podobna w działaniu do `agnes()` metoda klastrowania hirarchicznego dostępna w funkcji `hclust(stats)`. Umożliwia ona większy wybór metody łączenia klastrów. Argumentem `method`

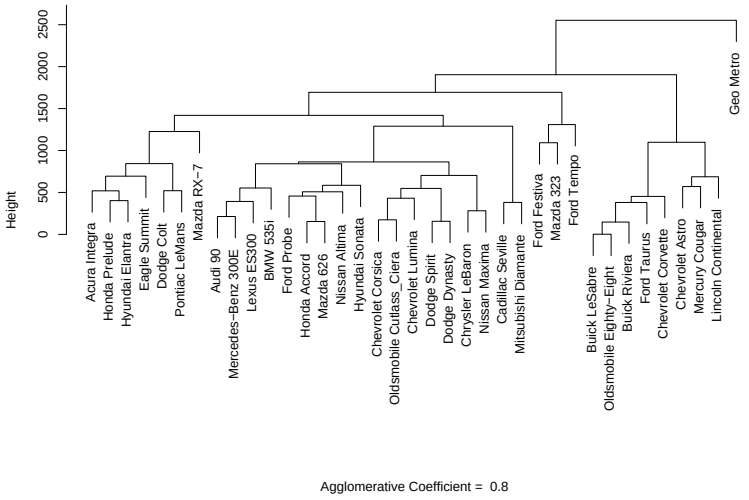


Rysunek 4.4: Graficzny przykład wyników funkcji *agnes()*.

należy wskazać jeden z wielu dostępnych indeksów do określania odległości pomiędzy klastrami, dostępne są indeksy znane z funkcji `agnes()` oraz `"mcquitty"`, `"ward"` i `"centroid"`.

Wykonać klastrowanie jest stosunkowo prosto, jednak to jeszcze nie jest koniec pracy, potrzebne są metody oceny dobroci podziału na skupiska. Do oceny jakości klastrowania można wykorzystać współczynnik silhouette, określający podobieństwo obiektu do innych obiektów w tym samym klastrze. Ten indeks jest wyznaczany przez funkcje `silhouette(cluster)`. Poniżej przedstawiamy przykład użycia tej funkcji, a na rysunku 4.6 przedstawiamy dopasowania poszczególnych punktów do klastrów. Wiele innych metod do oceny wyników klastrowania oraz badania zgodności dwóch podziałów na klastry jest dostępnych w pakiecie `clv`.

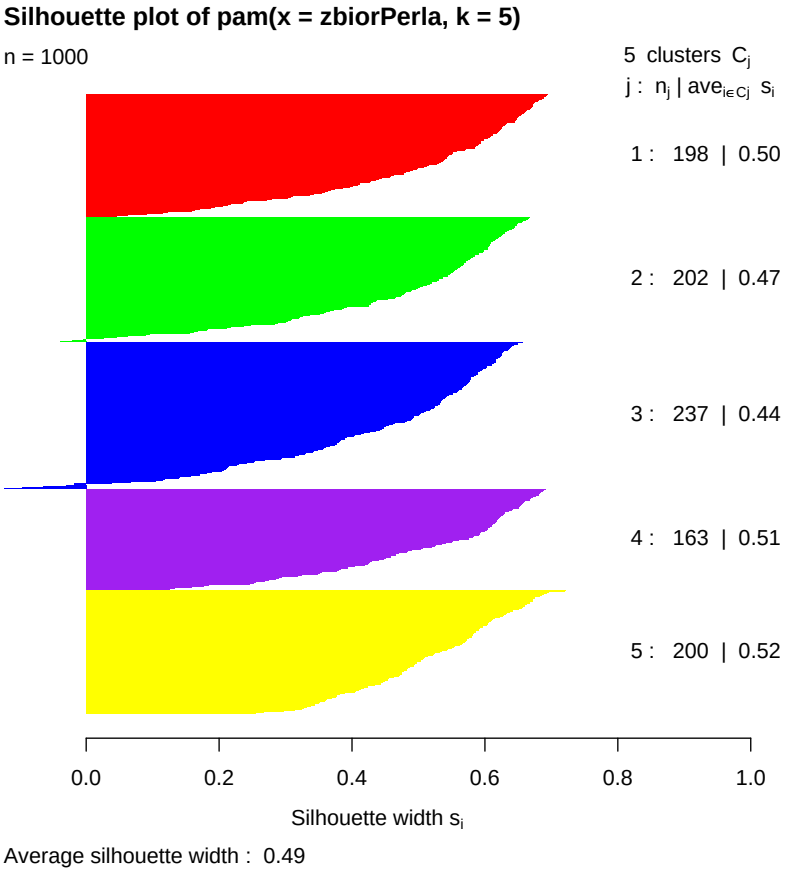
```
> kluster <- pam(zbiorPerla, 5)
> sil <- silhouette(kluster)
> summary(sil)
```



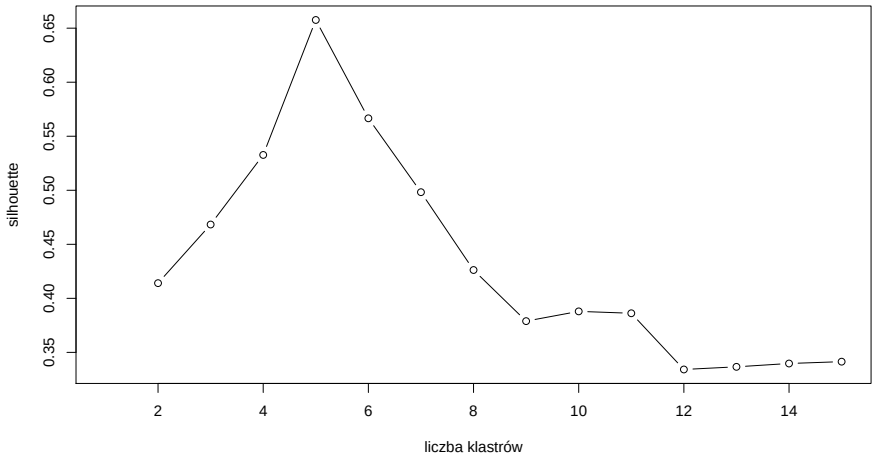
Rysunek 4.5: Drzewo dla wybranych modeli samochodów na bazie zbioru danych *(*Cars93(MASS)).

```
Silhouette of 1000 units in 5 clusters from pam(x = zbiorPerla, k = 5) :
Cluster sizes and average silhouette widths:
      198       202       237       163       200
0.5005513 0.4711742 0.4402300 0.5146160 0.5240964
Individual silhouette widths:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-0.1238  0.3895  0.5330  0.4873  0.6111  0.7220
> plot(sil, col = c("red", "green", "blue", "purple","yellow"))
```

Miara silhouette może użyta do wyznaczeniu liczby klastrów, na które należy podzielić dane. Na rysunku 4.7 przedstawiono zależność pomiędzy średnim współczynnikiem silhouette a liczbą klastrów wyznaczonych algorytmem PAN dla zbioru danych Gwiazda. W tym przypadku wyraźnie najlepszym wyborem jest 5 klastrów. Niestety w rzeczywistych problemach wybór klastrów nie jest tak prosty.



Rysunek 4.6: Wykres dopasowania punktów do poszczególnych klastrów z użyciem miary silhouette.



Rysunek 4.7: Wartości funkcji silhouette dla różnych liczb klastrów wyznaczonych algorytmem PAM na zbiorze danych Gwiazda.

Rozdział 5

Analiza dyskryminacji

W wielu dziedzinach potrzebne są metody, potrafiące automatycznie przypisać nowy obiekt do jednej z wyróżnionych klas. W medycynie interesować nas może czy pacjent jest chory, a jeżeli tak to na co (zbiorem klas do którego chcemy przypisać mogą być możliwe choroby, lub tylko informacja czy jest chory czy nie). W analizie kredytowej dla firm chcemy przewidzieć czy firma spłaci kredyt czy nie. W analizie obrazów z fotoradarów policyjnych będzie nas interesowało określenie numeru rejestracji samochodu który przekroczył prędkość a również typu pojazdu (w końcu ograniczenia dla ciężarówek są inne niż dla samochodów).

W rozdziale **??** używaliśmy regresji logistycznej do znalezienia parametrów, które można by wykorzystać w określeniu ryzyka pojawienia się wznowienia choroby u operowanych pacjentek. Okazuje się wiec, że regresja logistyczna jest klasyfikatorem, pozwalającym na przypisanie nowego obiektu do jednej z dwóch klas. Poniżej przedstawimy szereg funkcji implementujących inne popularne metody analizy dyskryminacji.

Celem procesu dyskryminacji (nazywanego też klasyfikacją, uczeniem z nauczycielem lub uczeniem z nadzorem) jest zbudowanie reguły, potrafiącej przypisywać możliwie dokładnie nowe obiekty do znanych klas. W przypadku większości metod możliwe jest klasyfikowanie do więcej niż dwie klasy.

5.1 Dyskryminacja liniowa i kwadratowa

Dyskryminacja liniowa, a więc metoda wyznaczania (hiper)płaszczyzn separujących obiekty różnych klas, jest dostępna w funkcji `lda(MASS)`. Rozszerzeniem tej metody jest dyskryminacja kwadratowa, umożliwiająca dyskryminacje powierzchniami, w opisie których mogą pojawić się człony stopnia drugiego. Metoda klasyfikacji kwadratowej jest dostępna w funkcji `qda(MASS)`. Wynikami obu funkcji jest klasyfikator, wyznaczony na zbiorze uczącym. Aby użyć go do predykcji klas dla nowych obiektów możemy wykorzystać przeciążoną funkcje `predict()`.

Poniżej przedstawiamy przykład użycia funkcji `lda()`. Z funkcji `qda()` korzysta się w identyczny sposób. Na potrzeby przykładu wykorzystaliśmy zbiór danych do-

tyczących występowania cukrzycy u Indian Pima, ten zbiór danych jest dostępny w zbiorze `PimaIndiansDiabetes2(mlbench)`. Interesować nas będą dwie zmienne z tego zbioru danych, opisujące poziom glukozy i insuliny, w zbiorze danych jest znacznie więcej zmiennych, ale na potrzeby wizualizacji wybraliśmy tę parę. Na bazie tych dwóch zmiennych będziemy badać skuteczność oceny czy dana osoba jest cukrzykiem z wykorzystaniem obu algorytmów klasyfikacji. Zbiór danych podzielimy na dwie części, uczącą i testową. Klasyfikator zostanie „nauczony" na zbiorze uczących, a później będziemy weryfikować jego właściwości na zbiorze testowym.

Pierwszym argumentem funkcji `lda()` jest zbiór zmiennych na bazie których budowany będzie klasyfikator (ramka danych lub macierz). Drugim argumentem `grouping` jest wektor określający klasy kolejnych obiektów. Kolejnym wykorzystanym poniżej argumentem jest `subset`, określający indeksy obiektów, na bazie których budowany ma być klasyfikator. Jeżeli argument `subset` nie będzie podany, to do konstrukcji klasyfikatora wykorzystane będą wszystkie obiekty. Jako pierwszy argument funkcji `lda()` można również podać również formułę, określającą, które zmienne mają być użyte do konstrukcji klasyfikatora a która zmienna opisuje klasy.

```
> # wczytujemy zbiór danych z pakietu mlbench, usuwamy brakujące dane i
    logarytmujemy poziom insuliny
> data(PimaIndiansDiabetes2)
> dane = na.omit(PimaIndiansDiabetes2)[,c(2,5,9)]
> dane[,2] = log(dane[,2])
> # zbiór danych chcemy podzielić na dwie części, uczącą i~testową,
    funkcją sample wylosujemy indeksy obiektów, które trafia do zbioru
    uczącego
> zbior.uczacy    = sample(1:nrow(dane), nrow(dane)/2, F)
> # wywołujemy funkcję lda
> klasyfikatorLDA = lda(dane[,1:2], grouping = dane[,3], subset=zbior.
    uczacy)
> # jak wygląda wynik w~środku?
> str(klasyfikatorLDA)
List of 8
 $ prior  : Named num [1:2] 0.643 0.357
  ..- attr(*, "names")= chr [1:2] "neg" "pos"
 $ counts : Named int [1:2] 126 70
  ..- attr(*, "names")= chr [1:2] "neg" "pos"
 $ means  : num [1:2, 1:2] 110.42 146.50   4.57   5.20
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "neg" "pos"
  .. ..$ : chr [1:2] "glucose" "insulin"
 $ scaling: num [1:2, 1] 0.0316 0.3227
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:2] "glucose" "insulin"
  .. ..$ : chr "LD1"
```

```
 $ lev    : chr [1:2] "neg" "pos"
 $ svd    : num 9
 $ N      : int 196
 $ call   : language lda(x = dane[, 1:2], grouping = dane[, 3], subset =
    zbior.uczacy)
 - attr(*, "class")= chr "lda"
```

Zbudowanie klasyfikatora to dopiero pierwszy krok, kolejnym jest jego ocena. Na bazie obserwacji niewykorzystanych do budowania klasyfikatora zbadamy jaka była zgodność klasyfikatora z rzeczywistymi danymi (ponieważ zbiór uczący wybraliśmy losowo, to dla różnych powtórzeń otrzymalibyśmy inny błąd klasyfikacji). Do klasyfikacji nowych obiektów użyjemy funkcji `predict()`. Jest to funkcja przeciążona, działająca dla większości klasyfikatorów. Wynikiem tej funkcji mogą być prognozowane klasy dla nowych obserwacji, lub też prawdopodobieństwa a posteriori przynależności do danej klasy.

```
> # używając metody predict wykonujemy klasyfikacje obiektów ze zbioru
    testowego
> oceny = predict(klasyfikatorLDA, newdata=dane[-zbior.uczacy,1:2])
> # jak wyglądają wyniki? Pole $class wskazuje na przewidzianą klasę, pole
     $posterior określa wyznaczone prawdopodobieństwo przynależności do
    każdej z klas, na podstawie tej wartości obiekt był przypisywany do
    bardziej prawdopodobnej dla niego klasy
> str(oceny)
List of 3
 $ class    : Factor w/ 2 levels "neg","pos": 1 1 2 1 1 1 1 1 2 1 ...
 $ posterior: num [1:196, 1:2] 0.5302 0.9480 0.0506 0.8663 0.7852 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:196] "5" "7" "9" "19" ...
  .. ..$ : chr [1:2] "neg" "pos"
 $ x        : num [1:196, 1]  0.540 -1.533  2.814 -0.762 -0.336 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:196] "5" "7" "9" "19" ...
  .. ..$ : chr "LD1"

> # porównajmy macierzą kontyngencji oceny i rzeczywiste etykietki dla
    kolejnych obiektów
> table(predykcja = oceny$class, prawdziwe = dane[-zbior.uczacy,3])
         prawdziwe
predykcja neg pos
      neg 115  29
      pos  21  31
```

W powyższym przykładzie w ostatnim poleceniu wyznaczyliśmy tablice kontyn-

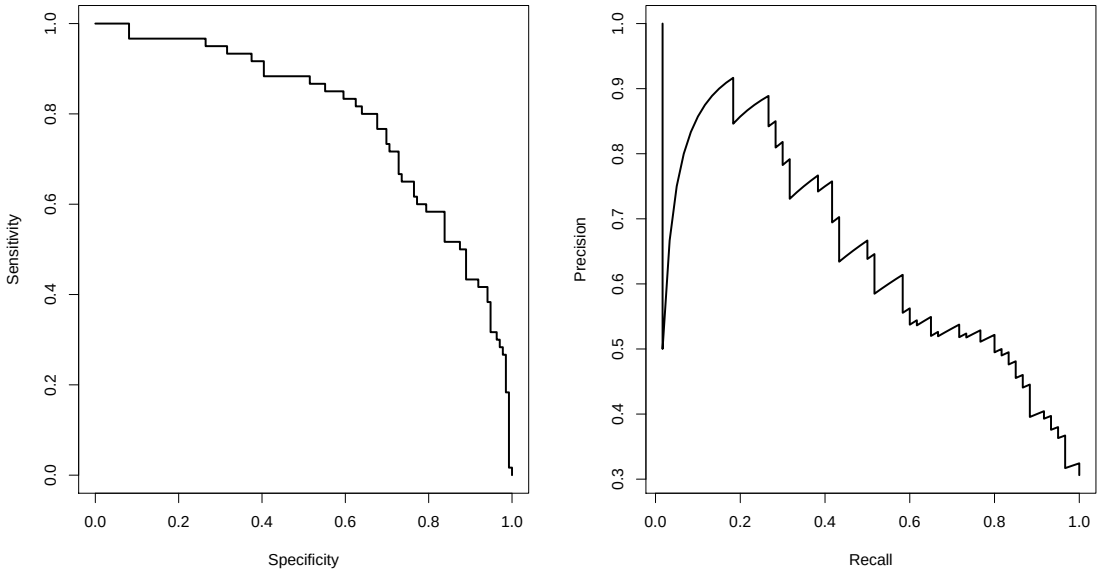
gencji dla wyników. Na bazie tak otrzymanej tablicy kontyngencji można wyznaczyć błąd predykcji. Jest wiele wskaźników opisujących błąd predykcji, począwszy od popularnych: czułość (ang. sensitivity `TP/(TP+FN)` opisuje jaki procent chorych zostanie poprawnie zdiagnozowanych jako chorzy), specyficzność (ang. (Specifity) `TN/(TN+FP)` określa jaki procent zdrowych zostanie poprawnie zdiagnozowanych jako zdrowi), przez mniej popularne aż po takie o których mało kto słyszał (np. mutual information, współczynnik ϕ itp.). Bogata lista takich współczynników wymieniona jest w opisie funkcji `performance(ROCR)` (definiując błąd klasyfikacji używa się oznaczeń `TP`, `TN`, `FP`, `FN` określających kolejno liczbę poprawnie wykrytych sygnałów pozytywnych, poprawnie wykrytych braków sygnału, fałszywie wykrytych sygnałów pozytywnych oraz fałszywie wykrytych braków sygnałów. W powyższym przypadku czułość wyniosła $31/(31+29) \approx 0.517$ a specyficzność $115/(115+21) \approx 0.846$.

Jeżeli już jesteśmy przy pakiecie `ROCR` to na przykładzie przedstawimy w jaki sposób wyznaczać krzywe ROC dla klasyfikatorów. Proszę zauważyć, że funkcja `predict()` poza ocenionymi klasami jako wynik przekazuje również prawdopodobieństwo przynależności do jednej z klas (pole `posterior` wyniku funkcji `predict()`). Krzywa ROC to zbiór punktów wyznaczonych dla różnych poziomów odcięcia (ang. threshold) dla wspomnianego prawdopodobieństwa przynależności do jednej z klas. Współrzędne każdego punktu to czułość i specyficzność (dokładniej rzecz biorąc 1-specyficzność) otrzymana dla zadanego punktu odcięcia.

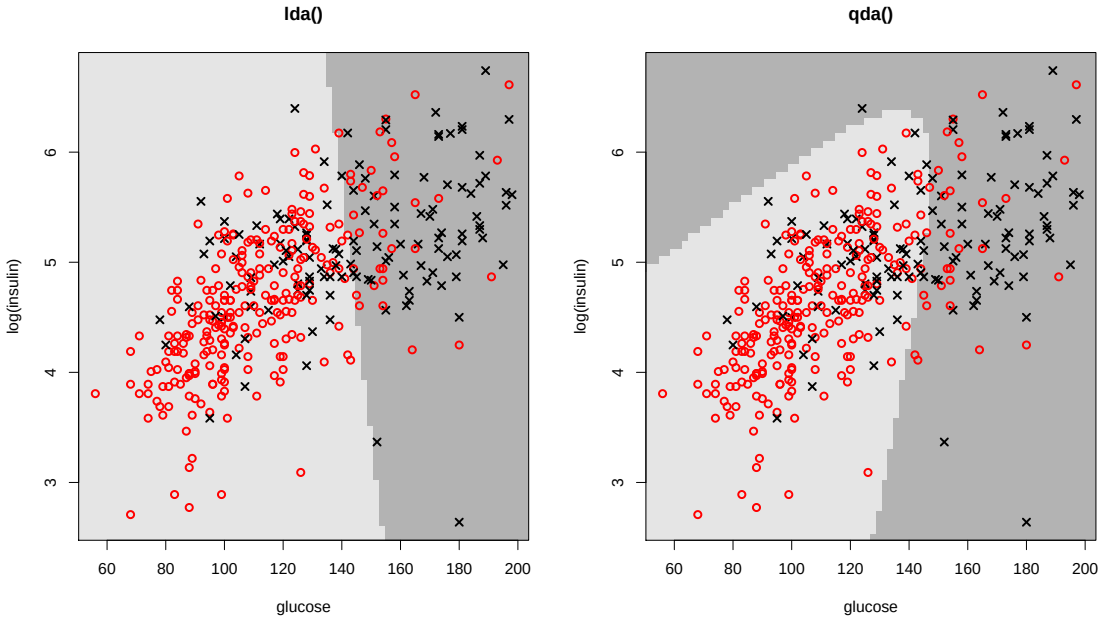
Poniżej przykład użycia funkcji z pakietu `ROCR`, służącej do rysowania krzywych ROC i innych o podobnych właściwościach. Wynik graficzny przedstawiony jest na rysunku 5.1. Na osiach tego wykresu mogą być przedstawiane różne miary dokładności klasyfikacji, w zależności od argumentów funkcji `performance()`.

```
# wyznaczamy obiekt klasy prediction, zawierający informacje o prawdziwych
    klasach, i prawdopodobieństwu przynależności do wybranej klasy
pred <- prediction(oceny$posterior[,2], dane[-zbior.uczacy,3])
# wyznaczamy i wyrysowujemy wybrane miary dobroci klasyfikacji
perf <- performance(pred, "sens", "spec")
plot(perf)
perf <- performance(pred, "prec", "rec")
plot(perf)
```

Na rysunku 5.2 przedstawiamy kształty obszarów decyzyjnych dla obu klasyfikatorów (do wyznaczania obszarów decyzyjnych można się posłużyć funkcją `partimat(klaR)` lub `drawparti(klaR)`). Obszary decyzyjne wyznaczane są dla wytrenowanego klasyfikatora, przedstawiają do której klasy zostałby przypisany punkt o określonych współrzędnych. Osoby chore na cukrzyce oznaczane są czarnymi krzyżykami, osoby zdrowe czerwonymi okręgami. Punkty w których przewidzianą klasą była by cukrzyca zaznaczone są ciemnoszarym kolorem a punkty dla których klasyfikowalibyśmy do grona osób zdrowych zaznaczono jaśniejszym szarym kolorem.



Rysunek 5.1: Wykres zależności czułości od specyficzności oraz miary prediction od recall dla klasyfikacji z użyciem funkcji *lda()*.



Rysunek 5.2: Przykładowe obszary decyzyjne dla liniowej i kwadratowej dyskryminacji dostępne w funkcjach *lda()* o *qda()*.

5.2 Metoda najbliższych sąsiadów

Bardzo popularną metodą klasyfikacji jest metoda k-sąsiadów. Idea jej działania jest prosta i intuicyjna. Nowemu obiektowi przypisuje się klasę, która występuje najczęściej wśród jego k sąsiadów (k najbliższych obiektów znajdujących się w zbiorze uczącym, najbliższych w sensie określonej miary odległości). Ten klasyfikator dostępny jest w różnych funkcjach, począwszy od `knn(class)` (zwykły klasyfikator najbliższych sąsiadów), przez `kknn(kknn)` (ważony klasyfikator k-sąsiadów) oraz `knncat(knncat)` (klasyfikator k-sąsiadów, również dla zmiennych jakościowych).

Poniżej przedstawimy implementację metody k-sąsiadów z funkcji `ipredknn(ipred)`. Na rysunku 5.3 prezentujemy obszary decyzyjne wyznaczone dla różnej liczby sąsiadów (odpowiednio k=3 i k=21) w omawianym zagadnieniu klasyfikacji na osoby zdrowe i chore na cukrzycę. Ponieważ metoda k-sąsiadów bazuje silnie na odległościach pomiędzy obiektami (jakoś trzeba mierzyć odległość od sąsiadów), przed rozpoczęciem obliczeń wykonamy skalowanie danych.

```
> # zaczynamy od przeskalownia danych
> dane[,1:2] = scale(dane[,1:2])
> # budujemy klasyfikator k-sąsiadów, dla 3 sąsiadów
> klasyfikatorKNN = ipredknn(diabetes~glucose+insulin, data = dane, subset
   =zbior.uczacy, k=3)
> # wykonujemy predykcję klas i wyświetlamy macierz kontyngencji
> oceny = predict(klasyfikatorKNN, dane[-zbior.uczacy, ], "class")
> table(predykcja = oceny, prawdziwe = dane[-zbior.uczacy,3])
         prawdziwe
predykcja neg pos
      neg  98  25
      pos  38  35
```

Błąd klasyfikacji można liczyć na palcach (powyższą procedurę należało by uśrednić po kilku wstępnych podziałach na zbiór uczący i testowy). Można też błąd klasyfikacji wyznaczyć wykorzystać funkcję `errorest(ipred)`. Wylicza ona błąd klasyfikacji (określony jako procent źle zaklasyfikowanych obiektów) używając różnych estymatorów tego błędu, w tym opartego na walidacji skrośnej (ocenie krzyżowej, ang. cross validation, domyślnie z podziałem na 10 grup), metodzie bootstrap lub estymatorze 632+. Estymator błędu możemy wybrać określając argument `estimator`. W funkcji `errorest()` jako kolejne argumenty należy wskazać zbiór danych, metodę budowy klasyfikatora (argument `model`) oraz metodę wyznaczania ocen dla zbioru testowego (argument `predict`). Funkcja `errorest()` pozwala na jednolity sposób wyznaczenia błędu dla dowolnej metody klasyfikacji. Przedstawimy poniżej przykład dla metody najbliższych sąsiadów.

```
> # wyznaczmy błąd klasyfikacji dla metody 3 sąsiadów
```

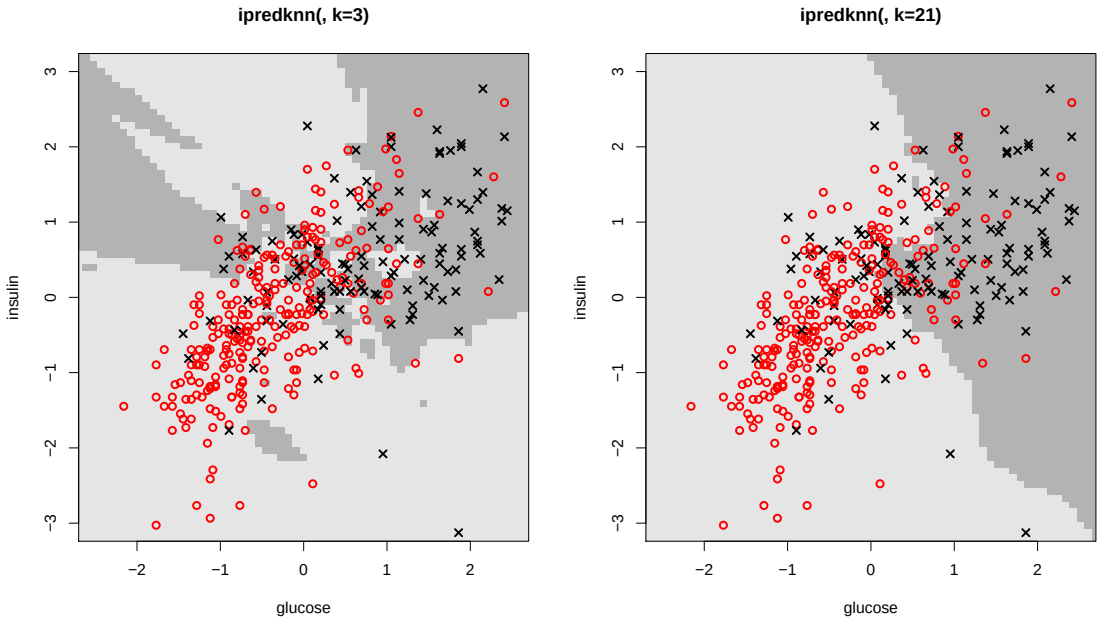
```
> errorest(diabetes~glucose+insulin, data = dane, model=ipredknn, k=3,
+          estimator = "632plus", predict= function(ob, newdata)  predict(
    ob, newdata, "class"))

Call:
errorest.data.frame(formula = diabetes ~ glucose + insulin, data = dane,
    model = ipredknn, predict = function(ob, newdata)  predict(ob, newdata
        , "class")
    , estimator = "632plus", k = 3)

         .632+ Bootstrap estimator of misclassification error
         with 25 bootstrap replications

Misclassification error:  0.2874

>
> # wyznaczmy błąd klasyfikacji dla metody 21 sąsiadów, powinna być
    stabilniejsza
> blad = errorest(diabetes~glucose+insulin, data = dane, model=ipredknn, k
    =21,
+          estimator = "632plus", predict= function(ob, newdata)  predict(
    ob, newdata, "class"))
> blad$error
[1] 0.2599233
```

5.3 Naiwny klasyfikator Bayesowski

Do klasyfikacji wykorzystać można szeroką grupę metod bazujących na ocenie prawdopodobieństwa przynależności do określonej grupy. Dla każdej z klas ocenia się częstość (w przypadku ciągłym gęstość) występowania obiektów o określonych parametrach. Następnie dla nowego obiektu wyznacza się częstości występowania obiektów poszczególnych klas i wybiera się klasę występującą dla tych parametrów najczęściej.

Do tej grupy metod należy naiwny klasyfikator Bayesowski. Bayesowski, ponieważ bazuje na regule Bayesa użytej do wyznaczenia prawdopodobieństwa a posteriori należenia do poszczególnych klas. Naiwność w tym kontekście oznacza przyjęte założenie, że łączna gęstość występowania obiektów jest iloczynem gęstości brzegowych. Naiwny klasyfikator Bayesowski jest dostępny w funkcjach `naiveBayes(e1071)` i `NaiveBayes(klaR)`. Poniżej przedstawimy tą drugą implementację.

Sposób użycia tej funkcji jest podobny do użycia innych opisanych powyżej klasyfikatorów. Na rysunku 5.4 przestawione są warunkowe brzegowe oceny gęstości dla obu zmiennych dla każdej z klas, na bazie tych gęstości wykonywana jest kalsyfikacja. Na rysunku 5.5 przedstawiamy przykładowe obszary decyzyjne dla naiwnego klasyfikatora Bayesowskiego.



Rysunek 5.3: Przykładowe obszary decyzyjne dla metody k-sąsiadów z parametrami k=3 i k=21.

```
> # konstruujemy naiwny klasyfikator Bayesowski
> mN    <- NaiveBayes(diabetes~glucose+insulin, data=dane, subset=zbior.
    uczacy)
> # wyznaczamy oceny
> oceny <- predict(mN, dane[-zbior.uczacy,])$class
> table(predykcja = oceny, prawdziwe = dane[-zbior.uczacy,3])
         prawdziwe
predykcja neg pos
      neg 113  27
      pos  23  33
> plot(mN)
```

5.4 Drzewa decyzyjne

Inną klasą klasyfikatorów są cieszące się dużą popularnością metody bazujące na drzewach decyzyjnych (klasyfikacyjnych). W tym przypadku klasyfikator jest reprezentowany przez drzewo binarne, w którego węzłach znajdują się pytania o wartości określonej cechy, a w liściach znajdują się oceny klas. Przykład drzewa klasyfikacyjnego przedstawiamy na rysunku 5.6. Dodatkowo w liściach przedstawiono proporcję obiektów z obu klas, które znalazły się w danym węźle, a wiec spełniły warunki

określone w poszczególnych węzłach drzewa. Jeżeli nowe obiekty będziemy przypisywać do klasy, która w danym liściu występowała najczęściej, to dla trzech liści (czwartego, szóstego i siódmego) klasyfikować będziemy do grupy osób chorych, a w pozostałych liściach będziemy klasyfikować do grupy osób zdrowych.

Dlaczego nas to nie dziwi?

W pakiecie R metoda wyznaczania drzew decyzyjnych dostępna jest w wielu różnych funkcjach. Popularnie wykorzystywane są funkcje `tree(tree)`, `rpart(rpart)` oraz `cpart(party)`. Poniżej przedstawimy tylko tą ostatnią, ponieważ są dla niej opracowane najbardziej atrakcyjne funkcje do prezentacji graficznej. Z wszystkich wymienionych funkcji do konstrukcji drzew korzysta się podobnie. Wymienione funkcje mogą służyć zarówno do wyznaczania drzew regresyjnych jak i klasyfikacyjnych, ale w tym miejscu przedstawimy tylko ich klasyfikacyjną naturę. Do wizualizacji drzew klasyfikacyjnych można wykorzystać (w zależności od tego jaką funkcją wyznaczyliśmy drzewo) funkcje `plot.BinaryTree(party)`, `plot.tree(tree)`, `text.tree(tree)`, `draw.tree(maptree)`, `plot.rpart(rpart)` oraz `text.rpart(rpart)`.

Aby zbudować drzewo klasyfikacyjne należy określić kryterium podziału, a więc na jaka wartość ma być minimalizowana przy tworzeniu kolejnych gałęzi (najczęściej jest to błąd klasyfikacji) oraz kryterium stopu (a wiec jak długo drzewo ma być dzielone). Różne warianty drzew umożliwiają kontrolę różnych kryteriów, w przypadku metody `ctree()` zarówno kryterium stopu jak i podziału można określić argumentem `control` (patrz opis funkcji `ctree_control(party)`). Poniżej przedstawiamy przykładową sesję z budową drzewa klasyfikacyjnego.

```
> # określamy kryteria budowy drzewa klasyfikayjnego
> ustawienia <- ctree_control(mincriterion = 0.5, testtype = "
   Teststatistic")
> # uczymy drzewo
> drzewo     <- ctree(diabetes~glucose+insulin, data=dane, subset = zbior.
   uczacy, controls = ustawienia)
> # narysujmy je
> plot(drzewo)
> # w standardowy sposób przeprowadzamy klasyfikacje
> oceny = predict(drzewo, dane[-zbior.uczacy,])
> table(predykcja = oceny, prawdziwe = dane[-zbior.uczacy,3])
         prawdziwe
predykcja neg pos
      neg 111  26
      pos  25  34
```

Zaletą drzew jest ich łatwość w interpretacji. Narysowany klasyfikator może być oceniony i poprawiony, przez eksperta z dziedziny, której problem dotyczy.

Dla drzew zapisanych jako obiekty klasy `tree` lub `rpart` dostępne są dwie przydatne funkcje umożliwiające modyfikację drzewa. Pierwsza to `prune.tree(tree)` (`prune.rpart(rpart)`). Pozwala ona na automatyczne przycinanie drzewa, poszcze-

gólnymi argumentami tej funkcji możemy ustalić jak drzewo ma być przycięte. Można określić pożądaną liczbę węzłów w drzewie, maksymalny współczynnik błędu w liściu drzewa oraz inne kryteria. Nie zawsze przycinanie automatyczne daje satysfakcjonujące rezultaty. W sytuacji gdy drzewo potrzebuje ludzkiej ingerencji można wykorzystać funkcję `snip.tree(tree)` (`snip.rpart(rpart)`) pozwalającą użytkownikowi na wskazanie myszką które węzły drzewa maja być usunięte. Inne ciekawe funkcje to `misclass.tree(tree)` (wyznacza błąd klasyfikacji dla każdego węzła z drzewa) oraz `partition.tree(tree)` (wyznacza obszary decyzyjne dla drzew).

5.5 Lasy losowe

Wadą drzew klasyfikacyjnych jest ich mała stabilność. Są jednak sposoby by temu zaradzić. Takim sposobem jest konstruowanie komitetu klasyfikatorów a wiec użycie metody bagging lub boosting. Nie wystarczyło tu miejsca by przedstawić te metody w ich ogólnej postaci, wspomnimy o nich na przykładzie lasów losowych.

Idea, która przyświeca metodzie lasów losowych możne być streszczona w zdaniu „Niech lasy składają się z drzew". Jak pamiętamy w metodzie bootstrap generowano replikacje danych, by ocenić zachowanie statystyki dla oryginalnego zbioru danych. Odmianą metody bootstrap w zagadnieniu klasyfikacji jest bagging. Na bazie replikacji zbioru danych konstruowane są klasyfikatory, które na drodze głosowania większością wyznaczają ostateczną klasą dla danej obserwacji. W przypadku lasów losowych komitet klasyfikatorów składa się z drzew klasyfikacyjnych, które są trenowane na replikacjach zbioru danych, dla ustalonego podzbioru zmiennych.

Ponieważ drzew w lesie jest dużo, do komitet głosujący demokratycznie charakteryzuje się większą stabilnością. Dodatkową zaletą drzew jest naturalny nieobciążony estymator błędu klasyfikacji. Generując replikacje losując metodą z powtórzeniami, średnio do replikacji nie trafia około jednej trzeciej obserwacji (w ramach ćwiczeń warto to sprawdzić). Te obserwacje, które nie trafiły do replikacji, można wykorzystać do oceny klasyfikatora nauczonego na danej replikacji. Taka ocena błędu określana jest błędem OOB (ang. out-of-bag). Jeszcze inną zaletą drzew losowych jest możliwość oceny zdolności dyskryminujących dla poszczególnych zmiennych (dzięki czemu możemy wybrać najlepszy zbiór zmiennych).

Lasy losowe dostępne są w funkcji `randomForest(randomForest)`. Poniżej przedstawiamy przykład jej użycia. Ponieważ przy konstrukcji lasów losowych generowane są losowe replikacje zbioru danych, to aby móc odtworzyć uzyskane wyniki warto skorzystać z funkcji `set.seed()`.

```
> # ustawiamy ziarno generatora, by można było odtworzyć te wyniki
> set.seed(1)
> # ćwiczymy las
> klasyfikatorRF <- randomForest(diabetes~glucose+insulin, data=dane,
    subset=zbior.uczacy, importance=TRUE, proximity=TRUE)
```

```
> # podsumowanie lasu
> print(klasyfikatorRF)

Call:
 randomForest(formula = diabetes ~ glucose+insulin, data = dane,
    importance = TRUE,      proximity = TRUE, subset = zbior.uczacy)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 21.94%
Confusion matrix:
    neg pos class.error
neg 106  20   0.1587302
pos  23  47   0.3285714
> # podobnie jak dla innych klasyfikatorów funkcją predict() możemy ocenić
    etykietki na zbiorze testowym i porównać błąd klasyfikacji z błędem z
    innych metod
> oceny = predict(klasyfikatorRF, dane[-zbior.uczacy,])
> table(predykcja = oceny, prawdziwe = dane[-zbior.uczacy,"diabetes"])
         prawdziwe
predykcja neg pos
      neg 114  25
      pos  22  35
```

Lasy losowe są uznawane za jedną z najlepszych metod klasyfikacji. Ich dodatkową zaletą jest możliwość użycia nauczonego lasu losowego do innych zagadnień niż tylko do klasyfikacji. Przykładowo na podstawie drzew z lasu można wyznaczyć ranking zmiennych, a tym samym określić które zmienne mają lepsze właściwości predykcyjne a które gorsze. Taką informację przekazuje funkcja `importance(randomForest)`, można tę informacje również przedstawić graficznie z użyciem funkcji `varImpPlot(randomForest)` (przykład dla naszego zbioru danych jest przedstawiony na rysunku 5.9).

Używając lasów losowych (regresyjnych) można również wykonać imputacje, a wiec zastąpić brakujące obserwacje w zbiorze danych. Do tego celu można posłużyć się funkcją `rfImpute(randomForest)`. Można również zdefiniować na podstawie lasów losowych miarę odstępstwa i wykrywać nią obserwacje odstające (do tego celu służy funkcja `outlier(randomForest)`). Można rónież używając lasów losowych przeprowadzać skalowanie wielowymiarowe, osoby zainteresowane tym tematem powinny zaznajomić się z funkcją `MDSplot(randomForest)`

5.6 Inne klasyfikatory

Powyżej wymienione metody to zaledwie początek góry lodowej funkcji do klasyfikacji dostępnych w pakiecie R. Poniżej wymienimy jeszcze kilka nazw popularnych

klasyfikatorów, oraz pokażemy w których funkcjach i pakietach dane metody są dostępne.

- **Sieci neuronowe**.

  Sieci neuronowe gotowe do użycia w zagadnieniach klasyfikacji są dostępne w funkcjach `nnet(nnet)` (prosta w użyciu sieć z jedną warstwą neuronów ukrytych) i `train(AMORE)` (bardziej zaawansowana funkcja do uczenia sieci neuronowych).

- **Metoda wektorów podpierających (SVM, ang. Support Vector Machines)**.

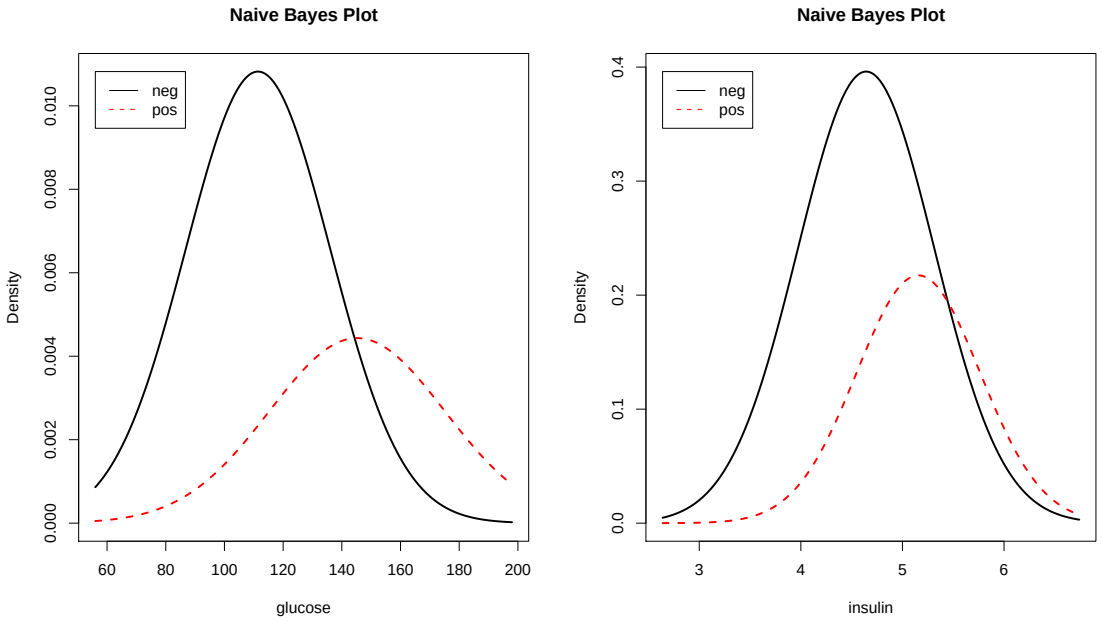
  Ideą wykorzystywaną w tej technice jest zwiększenie wymiaru przestrzeni obserwacji (przez dodanie nowych zmiennych np. transformacji zmiennych oryginalnych) tak by obiekty różnych klas można było rozdzielić hiperpłaszczyznami. Ta technika zyskała wielu zwolenników, funkcje pozwalające na jej wykorzystanie znajdują się w kilku pakietach np. `ksvm(kernlab)`, `svm(1071)`, `svmlight(klaR)`, `svmpath(svmpath)`.

- **Nearest mean classification i Nearest Shrunken Centroid Classifier** .

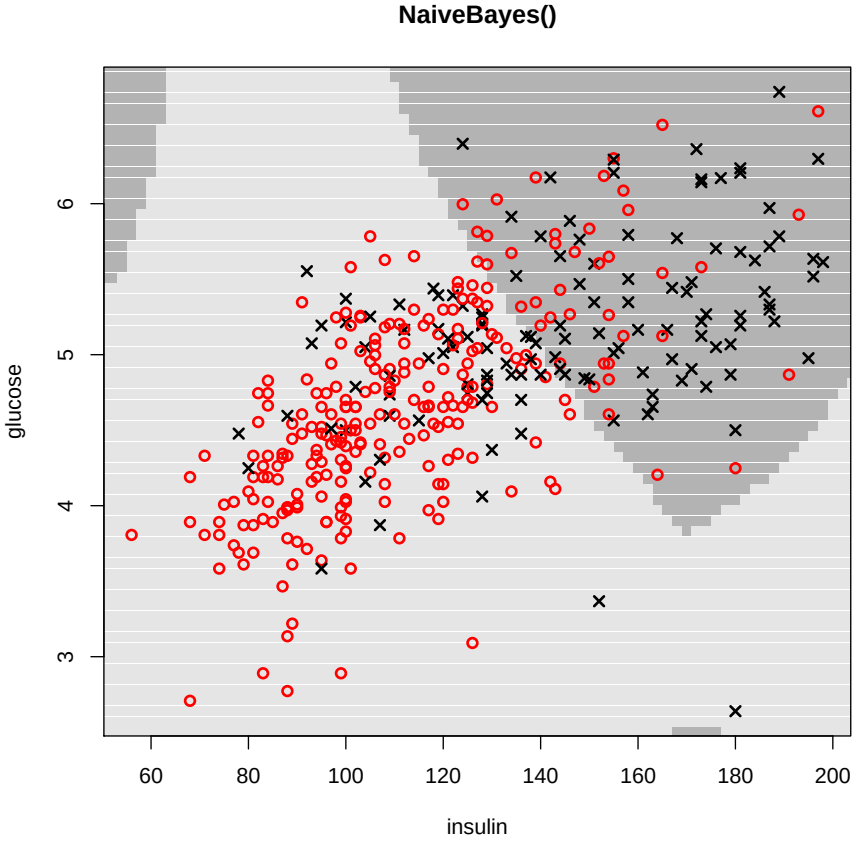
  Zasady działania podobne jak dla metod analizy skupień k-średnich i PAM. Metoda Nearest mean classification dostępna w funkcji `nm(klaR)` wyznacza średnie dla obiektów z danej klasy i nowe obiekty klasyfikuje do poszczególnych klas na podstawie wartości odległości nowej obserwacji od wyznaczonych średnich. Metoda Nearest Shrunken Centroid Classifier dostępna w funkcji `pamr.train(pamr)` działa podobnie, tyle, że zamiast średnich wyznacza centroidy.

- **Metody bagging i boosting**.

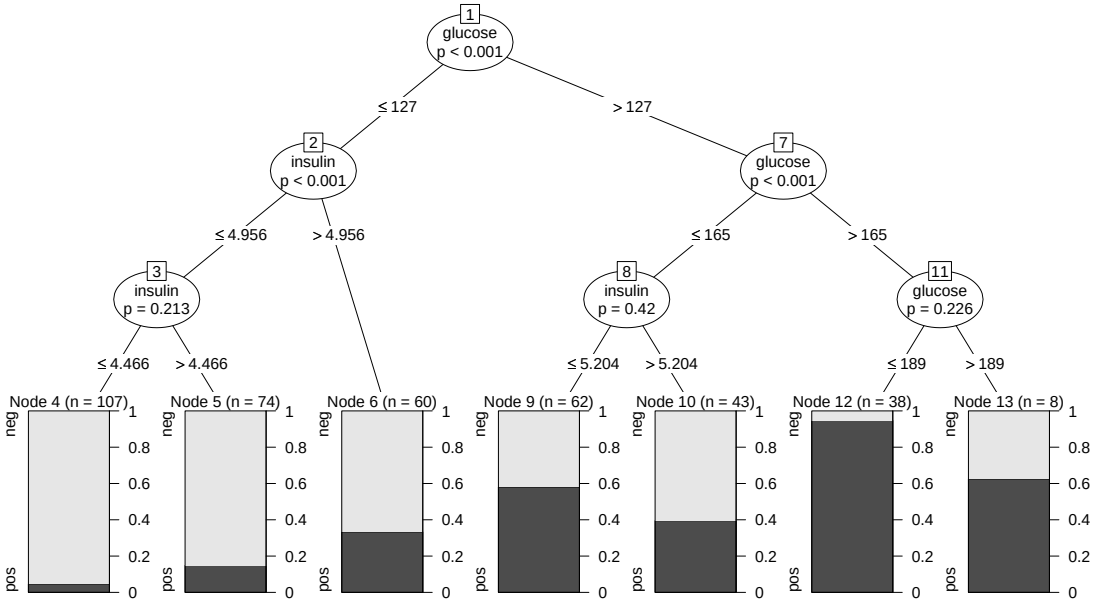
  Idea obu metod opiera się na tworzeniu replikacji zbioru danych, na których uczone są klasyfikatory. Wiele funkcji ze wsparciem dla tych metod znajduje się w pakiecie `boost` (boosting) i `ipred` (bagging), np. funkcje `adaboost(boost)`, `bagboost(boost)`, `l2boost(boost)`, `logitboost(boost)`, `ipredbagg(ipred)`.



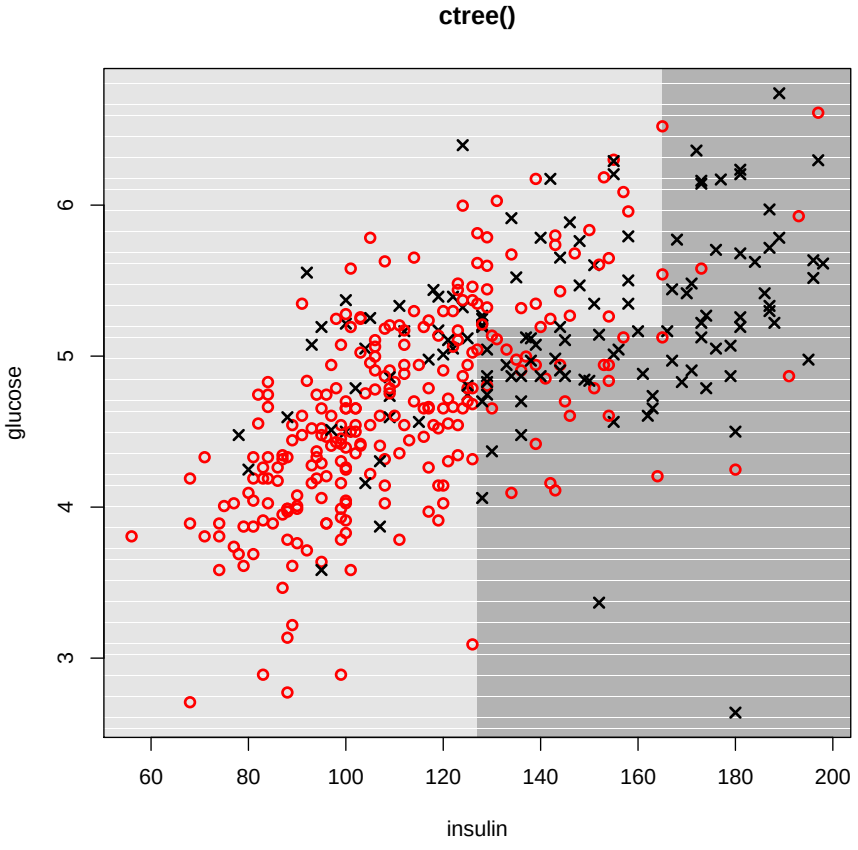
Rysunek 5.4: Warunkowe brzegowe oceny gęstości, na ich bazie funkcjonuje naiwny klasyfikator Bayesowski.



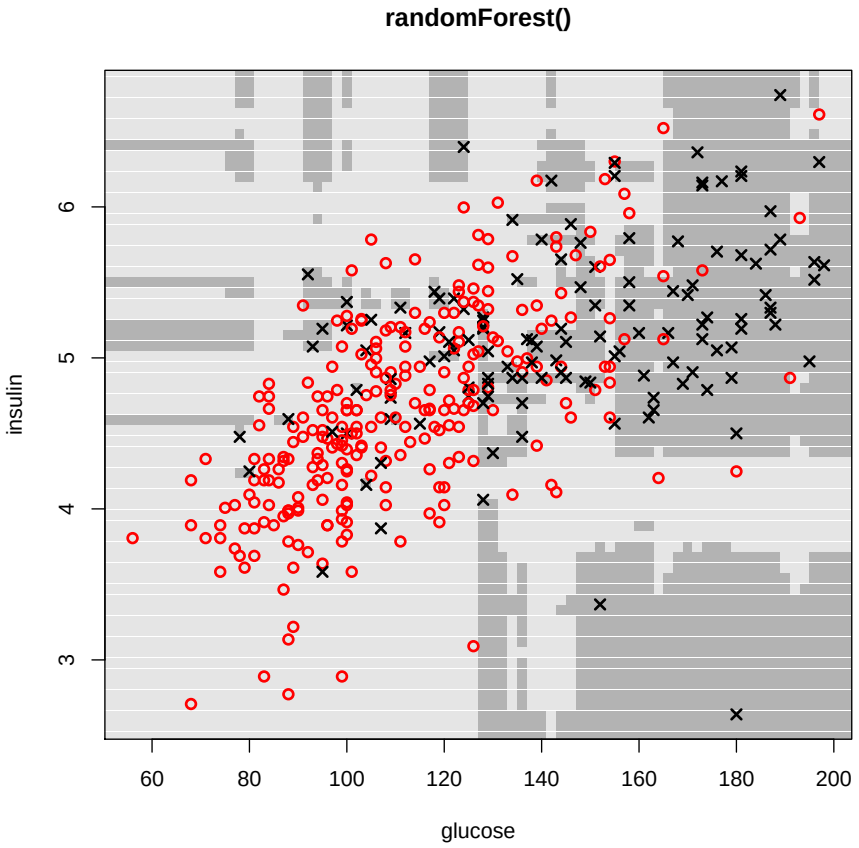
Rysunek 5.5: Przykładowe obszary decyzyjne dla naiwnego klasyfikatora Bayesowskiego.



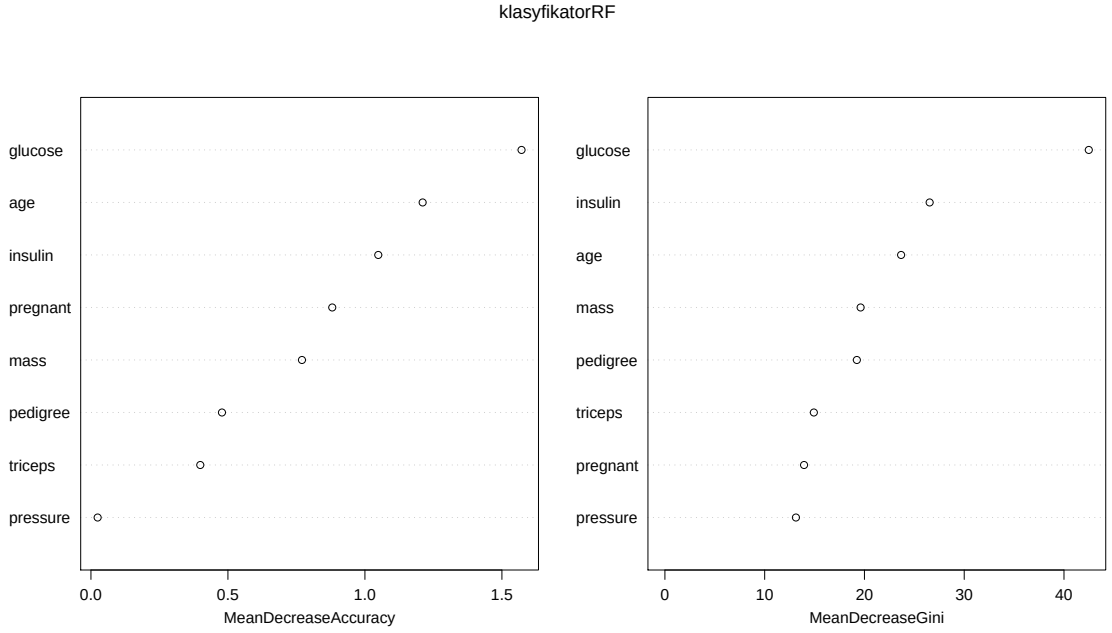
Rysunek 5.6: Przykładowe drzewo klasyfikacyjne wyznaczone funkcją *ctree()*.



Rysunek 5.7: Przykładowe obszary decyzyjne dla drzewa klasyfikacyjnego.



Rysunek 5.8: Przykładowe obszary decyzyjne dla lasów losowych.



Rysunek 5.9: Ranking zmiennych wykonany przez las losowy.

Rozdział 6

Analiza kanoniczna

Podstawowe problemy i wyniki analizy kanonicznej zostały sformułowane przez Harolda Hotellinga (wybitny ekonomista, matematyk, statystyk) w latach 1935-36.

Powstała jako metoda do badania zależności pomiędzy dwoma zbiorami zmiennych.

Do dziś doczekała się wielu uogólnień i rozszerzeń, np. na badanie relacji pomiędzy wieloma zbiorami zmiennych, na badane relacji w obecności współliniowych zmiennych (przez regularyzację) itp.

6.1 Problem

Mamy dwa zbiory zmiennych $\{X_1, ..., X_p\}$ i $\{Y_1, ..., Y_q\}$.

Chcemy znaleźć taką kombinację liniową zmiennych z pierwszego zbioru, aby korelowała ona możliwie najsilniej ze zmiennymi z drugiego zbioru.

Innymi słowy, szukamy wektorów współczynników a i b, takich, że

$$cor(a'X, b'Y)$$

jest możliwie największa.

6.2 Rozwiązanie

Wektor współczynników a to wektor własny odpowiadający największej wartości własnej macierzy

$$S_{22}^{-1} S_{21} S_{11}^{-1} S_{12} \tag{6.1}$$

a wektor współczynników b to wektor własny odpowiadający największej wartości własnej macierzy

$$S_{11}^{-1} S_{12} S_{22}^{-1} S_{21}. \tag{6.2}$$

Korelacja $cor(a'X, b'Y)$ to wartość największa wartość własna z powyższych macierzy.

[Wyprowadzenie na tablicy]

Nowe zmienne $u_1 = a'X$ i $v_1 = b'Y$ wyjaśniają największą część korelacji pomiędzy zbiorami wektorów X i Y, ale nie całą.

Kolejnym krokiem jest znalezienie kolejnych zmiennych $u_i = a_i X$ i $v_i = b_i Y$, tak by:

- wektory u_i są nieskorelowane pomiędzy sobą,
- wektory v_i są nieskorelowane pomiędzy sobą,
- korelacje $cor(u_i, v_i)$ tworzą nierosnący ciąg odpowiadający możliwie największym cząstkowym korelacjom.

Jeżeli obserwacje pochodzą z wielowymiarowego modelu normalnego $\mathcal{N}(\mu, \Sigma)$ to możemy testować:

$$H_0 : R_i = 0 \forall_i$$

Statystyka testowa dla testu ilorazu wiarogodności

$$LRT = -n \sum_{i=1}^{s} \log(1 - R_i^2)$$

ma asymptotyczny rozkład χ^2_{pq}.

$$H_0 : R_i = 0 \forall_{i>k}$$

Statystyka testowa dla testu ilorazu wiarogodności

$$LRT = -n \sum_{i=k+1}^{s} \log(1 - R_i^2)$$

ma asymptotyczny rozkład $\chi^2_{(p-k)(q-k)}$.

Wartość n w statystykach testowych zamienia się czasem na $n - \frac{1}{2}(p + q + 3)$, co poprawia test.

6.3 Założenia

- wielowymiarowa normalność,
- brak obserwacji odstających (miara Cooka, Leverage, test Grubbsa, test Dixona)
- brak współliniowości (reguła kciuka, wyznacznik $> 10^{-5}$)

Liczba obserwacji powinna być większa od około 20$*$liczba zmiennych.

6.4 Jak to zrobić w R

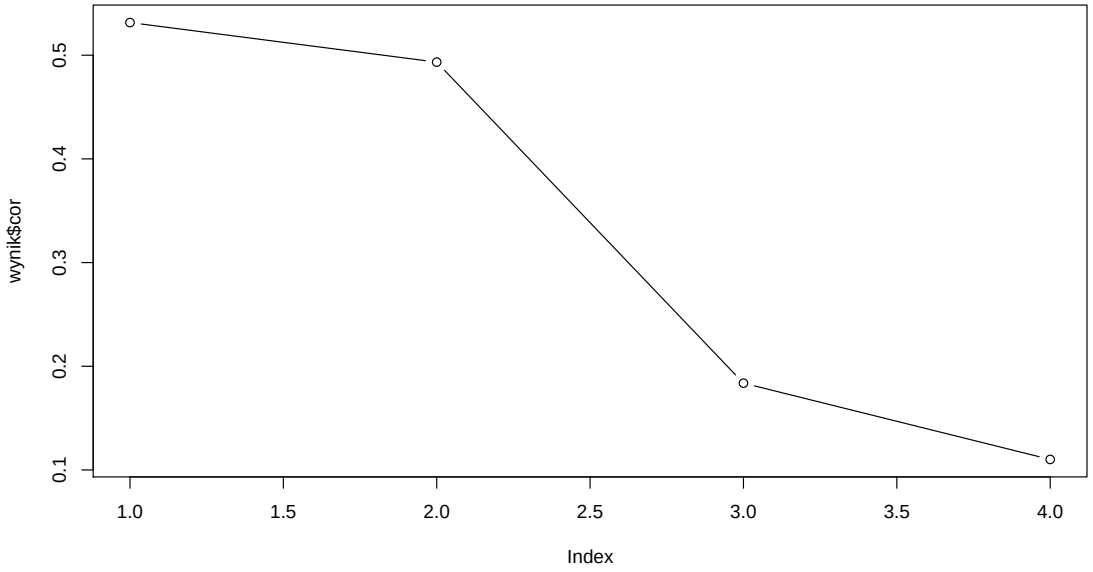
Analiza kanoniczna jest zaimplementowana między innymi w pakiecie CCA w funkcji $cc()$.

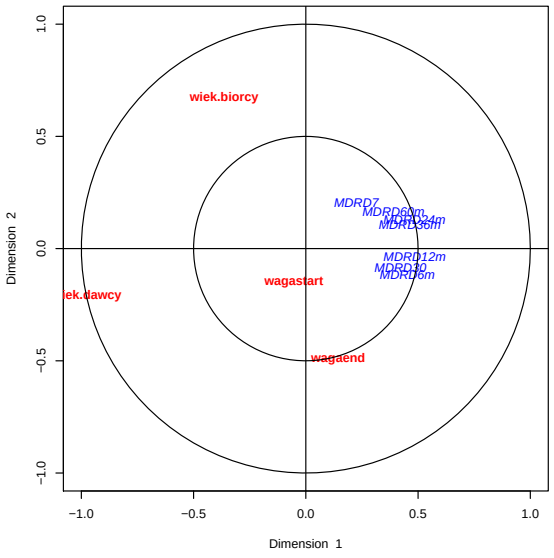
Prześledźmy poniższy kod

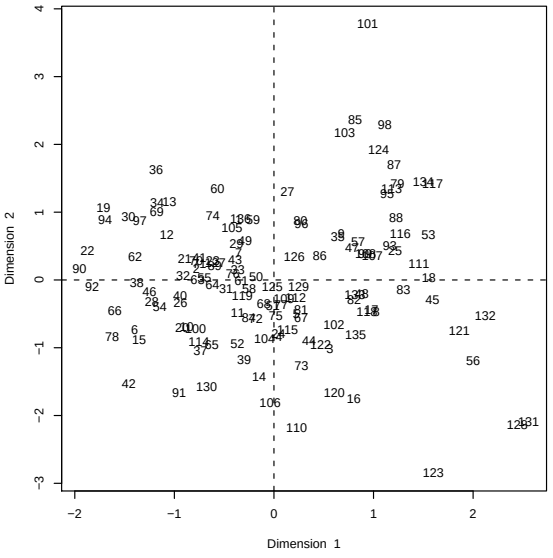
```
> library(CCA)
> dane = read.table("dane.csv",header=T,sep=";")
> X = dane[,c(9:10)]      # kolumny z waga
> Y = dane[,c(11:17)]     # kolumny z MDRD
> wynik = cc(X,Y)
> wynik$cor
[1] 0.3754946 0.1907164
```

```
> wynik$xcoef
                [,1]        [,2]
wagastart  0.1047822 -0.09276486
wagaend   -0.1154909  0.01404359
> wynik$ycoef
                [,1]        [,2]
MDRD7    0.056059823  0.05799373
MDRD30  -0.059196976 -0.03981322
MDRD6m  -0.006987328  0.02870234
MDRD12m -0.094082377  0.07732582
MDRD24m  0.119735985 -0.09688825
MDRD36m -0.024980200 -0.01744831
MDRD60m -0.007345604  0.04083270
> plot(wynik$cor,type="b")
> plt.cc(wynik,var.label=T)
```

6.5 Przykładowe wyniki







6.6 Studium przypadku

Rozdział 7

Analiza korespondencji (odpowiedniości)

7.1 Problem

Obserwujemy dwie zmienne jakościowe. Pierwsza zmienna przyjmuje wartości z n_1 poziomów, druga z n_2 poziomów.

Interesuje nas, czy zmienne te są od siebie niezależne, a jeżeli nie są niezależne to które kombinacje poziomów występują ze sobą znacznie częściej.

Dane z którymi mamy do czynienia to macierz kontyngencji (tabela wielodzielcza) o wymiarach $k \times l$ wyznaczona dla dwóch zmiennych o odpowiednio k i l poziomach. Jeżeli k i l są małe, to taką macierz można ogarnąć rzutem oka. Jednak dla zmiennych występujących na wielu poziomach potrzebne są już dodatkowe narzędzia.

7.2 Rozwiązanie

Aby ocenić niezależność dwóch zmiennych można wykorzystać test χ^2 z funkcji `chisq.test()` lub inny test dla macierzy kontyngencji (jeżeli poziomy są uporządkowane to dobrym rozwiązaniem będzie test Cochrana-Armitage). Testem tym zweryfikujemy hipotezę o niezależności częstości występowań poszczególnych zmiennych.

Jeżeli jednak odrzucimy tę hipotezę zerową, czyli przyjmiemy że jest JAKAŚ zależność, to naturalnym pytaniem będzie jaka to zależność i pomiędzy którymi poziomami.

Aby ocenić, które zmienne występują częściej ze sobą można wykonuje się tzw. analizę korespondencji. Jeżeli zmienne były by niezależne od siebie, to zachodziło by równanie

$$p_{ij} = p_{i\cdot} p_{\cdot j}, \qquad i \in \{1...k\},\ j \in \{1...l\}.$$

gdzie p_{ij} to prawdopodobieństwo zaobserwowania pierwszej zmiennej na poziomie i i jednocześnie drugiej na poziomie j, $p_{i\cdot}$ to prawdopodobieństwo zaobserwowa-

nia zmiennej pierwszej na poziomie i a $p_{.j}$ to prawdopodobieństwo zaobserwowania zmiennej drugiej na poziomie j.

Żeby ocenić, które zmienne występują częściej lub rzadziej niż wynikało by to z niezależności, wyznaczymy standaryzowane reszty Pearsonowskie, zastąpimy też prawdopodobieństwa ich częstościowymi ocenami (czyli $\hat{p}$ to liczba obserwowanych zdarzeń podzielona przez liczbę wszystkich obserwowanych zdarzeń)

$$\hat{e}_{ij} = \frac{\hat{p}_{ij} - \hat{p}_{i\cdot}\hat{p}_{\cdot j}}{\hat{p}_{i\cdot}\hat{p}_{\cdot j}}.$$

Duże dodatnie wartości $\hat{e}_{ij}$ odpowiadają wysokiemu współwystępowaniu, ujemne wartości odpowiadają występowaniu rzadszemu niż losowe. Możemy teraz macierz $E = [\hat{e}_{ij}]$ przedstawić w postaci graficznej używając tzw. biplotu. Innymi słowy wyznaczamy dekompozycję SVD macierzy E

$$E_{k\times l} = U_{k\times k}\Sigma_{k\times l}V^T_{l\times l}.$$

Kolumny macierzy $U_{k\times k}$ to wektory własne macierzy E^TE a kolumny macierzy V to wektory własne macierzy EE^T. Na przekątnej macierzy diagonalnej Σ znajdują się tzw. wartości singularne (osobliwe?) równe pierwiastkom z wartości własnych macierzy E^TE i EE^T. „Przy okazji" kolumny macierzy U rozpinają ortonormalną bazę na kolumnach macierzy E a kolumny macierzy V rozpinają ortonormlaną bazę na wierszach macierzy E. Można więc przedstawić macierz E (lub możliwie dużo informacji z tej macierzy) we nowej przestrzeni określonej na bazie współrzędnych wierszy i kolumn macierzy E.

7.3 Jak to zrobić w R?

Przedstawimy funkcje `ca(ca)` oraz `corresp(MASS)`. Obie służą do wykonania analizy korespondencji. Wykonują ją jednak w odrobinę odmienny sposób przez co wyniki końcowe też są inne.

7.4 Studium przypadku

Przedstawmy przykład bazujący na danych o zatrudnieniu w poszczególnych województwach. Zmienne które chcemy porównać to Województwo (16 poziomów) i Sektor pracy (4 poziomy: rolnictwo, przemysł, usługi, bezrobotni). Podejrzewamy, że struktura zatrudnienia różni się pomiędzy województwami.

```
> library(MASS)
> library(ca)
> library(RColorBrewer)
> # przygotujmy wektor kolorow
```

```
> kolory = rev(brewer.pal(11,"Spectral"))

> # wybierzmy interesujące nas kolumny
> # konwertujemy tabele na macierz, takiego formatu spodziewa się funkcja
    heatmap()
> dane = as.matrix(daneGUS[,c(22:25)])
> colSums(dane)
  pr.rolnictwo  pr.przemysl    pr.uslugi   bezrobotni
          2249          4680          8309          743
> rowSums(dane)
       DOLNOSLASKIE  KUJAWSKO-POMORSKIE             LODZKIE
               1218                 791                1304
          LUBELSKIE            LUBUSKIE         MALOPOLSKIE
               1018                 451                1336
        MAZOWIECKIE            OPOLSKIE        PODKARPACKIE
               2373                 379                 852
          PODLASKIE           POMORSKIE             SLASKIE
                478                 792                1845
     SWIETOKRZYSKIE WARMINSKO-MAZURSKIE       WIELKOPOLSKIE
                622                 573                1371
 ZACHODNIOPOMORSKIE
                578

> # jak wygląda macierz z danymi?
> head(dane)
                   pr.rolnictwo pr.przemysl pr.uslugi bezrobotni
DOLNOSLASKIE                 74         415       651         78
KUJAWSKO-POMORSKIE          128         245       369         49
LODZKIE                     220         384       636         64
LUBELSKIE                   328         197       447         46
LUBUSKIE                     43         151       244         13
MALOPOLSKIE                 205         381       689         61
```

Macierz 64 liczb jest trudno ogarnąć nieuzborojonym okiem. Możemy zauważyć, że biorąc pod uwagę, że najwięcej ludzi pracuje w sektorze usługowym (8 309 tys.) również łatwo zauważyć, że najludniejsze jest województwo Mazowieckie (2 373 tys.) ale z tych surowych danych ciężko wyciągnąć głębszy wniosek.

```
> # tę macierz można przedstawić w postaci graficznej
> # czerwony odpowiada dużym liczbom, niebieski małym
> heatmap(dane,scale="col",Colv=NA, col= kolory)

> # zobaczmy czy sa zaleznosci pomiedzy kolumnami i wierszami, wygląda na
    to że nie są to niezależne cechy
> chisq.test(dane)
```

```
        Pearson's Chi-squared test

data:  dane
X-squared = 1031.905, df = 45, p-value < 2.2e-16
```

Wykonaliśmy test χ^2 i otrzymaliśmy bardzo małą p-wartość, która upewniła nas w przekonaniu, że województwa mają inną strukturę zatrudnienia.

```
> # zobaczmy które kombinacje występują częściej niż w przypadku
    niezależności
> # policzymy residua Pearsonowskie
> P = dane/sum(dane)
> # macierz czestosci oczekiwanych
> PP = outer(rowSums(P),colSums(P))
> # macierz residuow Pearsonowskich
> E = (P-PP)/sqrt(PP)
> head(E)
                   pr.rolnictwo pr.przemysl pr.uslugi   bezrobotni
DOLNOSLASKIE           -0.05885     0.02442   0.00557     0.022465
KUJAWSKO-POMORSKIE      0.01250     0.00694  -0.01648     0.015945
LODZKIE                 0.02130     0.00086  -0.01275     0.003427
LUBELSKIE               0.12209    -0.04632  -0.02829    -0.001528
LUBUSKIE               -0.02032     0.01302   0.00491    -0.013765
MALOPOLSKIE             0.00979    -0.00409  -0.00168    -0.001118
> # tę macierz również można przedstawić za pomocą mapy ciepła
> heatmap(E,scale="none",Colv=NA,col= kolory)
```

Wyznaczyliśmy macierz reszt Pearsonowskich E (którą też przedstawiliśmy graficznie z użyciem mapy ciepła) i z tej macierzy możemy już odczytać w których województwach poszczególne sektory są popularniejsze. Największe reszty obserwujemy dla sektora rolnictwa, zarówno duże dodatnie wartości (w województwie lubelskim, podlaskim, podkarpackim i świętokrzyskim) jak i duże (co do modułu) ujemne wartości (w województwie śląskim i dolnośląskim).

Możemy teraz przedstawić graficznie macierz E z użyciem biplotu (z wykorzystaniem dekompozycji SVD).

```
> # wykonujemy dekompozycje na wartosci osobliwe (singularne/szczegolne)
> A = svd(E)
> X = t(apply(A$u, 1, "*", sqrt(A$d)))
> Y = t(apply(A$v, 1, "*", sqrt(A$d)))
# zwykle współrzędne liczone są ze wzorow, w ktorych A jest dekompozycja
    innej macierzy
# [TODO: uzupełnić albo pominąć]
# A = Dr^(-1/2) A$u A$d
```

```
# B = Dc^(-1/2) A$v a$d
> plot(rbind(X[,1:2], Y[,1:2]), xlab="", ylab="", main="Bramka nr. 1", lwd
   =3)

> # analiza korespondencji z użyciem pakietu MASS
> biplot(corresp(dane, nf = 2))
> # analiza korespondencji z użyciem pakietu ca
> # argument mass powoduje że liczebności komórek odpowiadają wielkościom
   punktów na wykresie
> plot(ca(dane), mass=c(T,T))
> summary(ca(dane))

Principal inertias (eigenvalues):

     dim    value      %   cum%   scree plot
[1,] 1      0.054859  85.0  85.0  ************************
[2,] 2      0.007352  11.4  96.3  **
[3,] 3      0.002360   3.7 100.0
[4,]        -------- -----
[5,] Total: 0.064571 100.0


Rows:
     name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1  |  DOL |   76  918   71 | -225 837  70 |  -70  81  51 |
2  |  KUJ |   49  848   11 |   65 286   4 |  -91 563  56 |
3  |  LOD |   82 1000   10 |   80 838  10 |  -35 162  14 |
4  | LUBE |   64 1000  277 |  527 990 322 |   53  10  24 |
5  | LUBU |   28  723   12 | -142 719  10 |  -11   4   0 |
6  |  MAL |   84  995    2 |   37 960   2 |    7  34   1 |
7  |  MAZ |  148 1000   88 |  -95 238  25 |  171 762 587 |
8  |  OPO |   24  496    3 |   -5   2   0 |  -68 494  15 |
9  | PODK |   53  926   78 |  296 925  85 |  -12   2   1 |
10 | PODL |   30  994   56 |  342 974  64 |   49  20  10 |
11 |  POM |   50  939   24 | -171 919  26 |   25  20   4 |
12 |  SLA |  115  995  187 | -319 970 214 |  -52  25  42 |
13 |  SWI |   39  968  128 |  443 926 139 |  -94  41  47 |
14 |  WAR |   36  515    4 |  -42 246   1 |  -44 269  10 |
15 |  WIE |   86  808   15 |   -1   0   0 |  -97 808 109 |
16 |  ZAC |   36  806   33 | -203 702  27 |   78 103  30 |

Columns:
    name   mass  qlt  inr    k=1 cor ctr    k=2 cor ctr
1 | pr.r |  141  999  720 |  574 999 847 |   -4   0   0 |
2 | pr.p |  293  967  128 | -120 509  77 | -114 458 514 |
3 | pr.s |  520 1000  111 |  -90 587  77 |   75 413 402 |
```

```
4 |  bzr |   46  236   42 |   21   7   0 | -115 228  83 |
```

Przy opisie analizy korespondencji często wymienianym czynnikiem jest inercja, nazywana czasem bezwładnością (przez podobieństwo do inercji w fizyce). Aby opisać formalnie czym jest ta miara musimy zacząć od kilku innych pojęć.

Niech masa wiersza i masa kolumny będzie określona jako brzegowe częstości macierzy kontyngencji.

```
> (masa.kolumny = colSums(prop.table(dane)))
pr.rolnictwo  pr.przemysl    pr.uslugi   bezrobotni
  0.14072962   0.29284776   0.51992992   0.04649271
> (masa.wiersza = rowSums(prop.table(dane)))
       DOLNOSLASKIE  KUJAWSKO-POMORSKIE             LODZKIE
         0.07621551          0.04949628          0.08159690
          LUBELSKIE            LUBUSKIE         MALOPOLSKIE
         0.06370064          0.02822101          0.08359927
        MAZOWIECKIE            OPOLSKIE        PODKARPACKIE
         0.14848883          0.02371566          0.05331331
         PODLASKIE           POMORSKIE             SLASKIE
         0.02991052          0.04955885          0.11544960
     SWIETOKRZYSKIE WARMINSKO-MAZURSKIE       WIELKOPOLSKIE
         0.03892122          0.03585508          0.08578937
 ZACHODNIOPOMORSKIE
         0.03616795
```

Profilem wiersza (kolumny) będą wartości w wierszu (kolumnie) unormowane przez masę kolumny (wiersza).

```
> head(profil.kolumny <- dane/rowSums(dane))
                   pr.rolnictwo pr.przemysl pr.uslugi bezrobotni
DOLNOSLASKIE         0.06075534   0.3407225 0.5344828 0.06403941
KUJAWSKO-POMORSKIE   0.16182048   0.3097345 0.4664981 0.06194690
LODZKIE              0.16871166   0.2944785 0.4877301 0.04907975
LUBELSKIE            0.32220039   0.1935167 0.4390963 0.04518664
LUBUSKIE             0.09534368   0.3348115 0.5410200 0.02882483
MALOPOLSKIE          0.15344311   0.2851796 0.5157186 0.04565868
```

Tak unormowane profile wierszy i kolumn mogą być ze sobą porównywane. Im większa odległość pomiędzy profilami kolumn tym mniejsza zależność pomiędzy czynnikami opisanymi w tych kolumnach (oczywiście dla wierszy jest tak samo). Średni profil kolumnowy to masa wiersza a średni profil wierszowy to masa kolumn.

Inercja to odległość (w mierze χ^2) pomiędzy daną kolumną (wierszem, punktem) a średnią wartością dla kolumn (wierszy). Całkowita inercja to suma odległości dla

wszystkich kolumn (wierszy). **Im większa inercja tym punkty są oddalone dalej od średniego profilu wiersza/kolumny.**

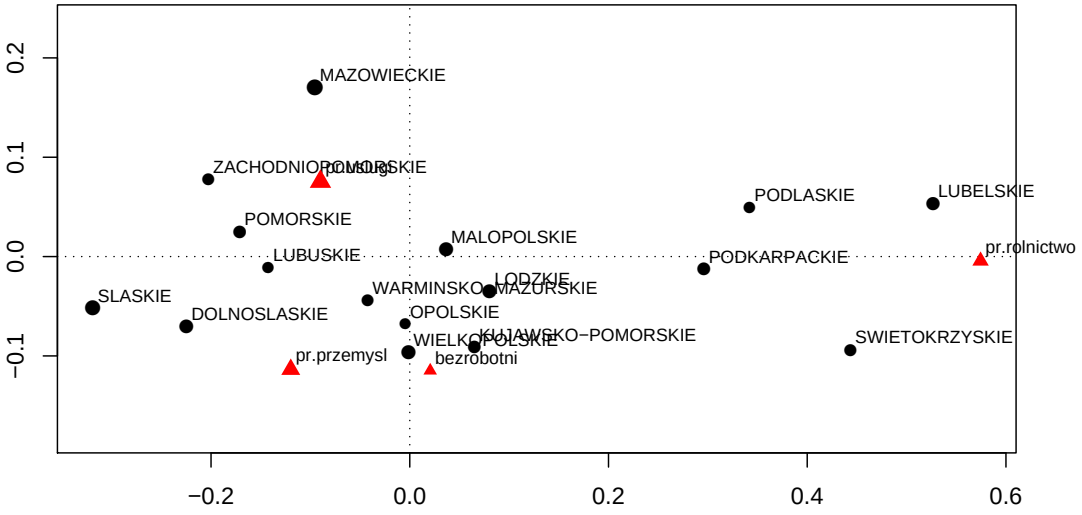
[TODO: uzupełnić. nawiązać do wyników funkcji summary.ca()]

Wyniki powyższych instrukcji oglądać można na wykresie 7.1 i kolejnych.

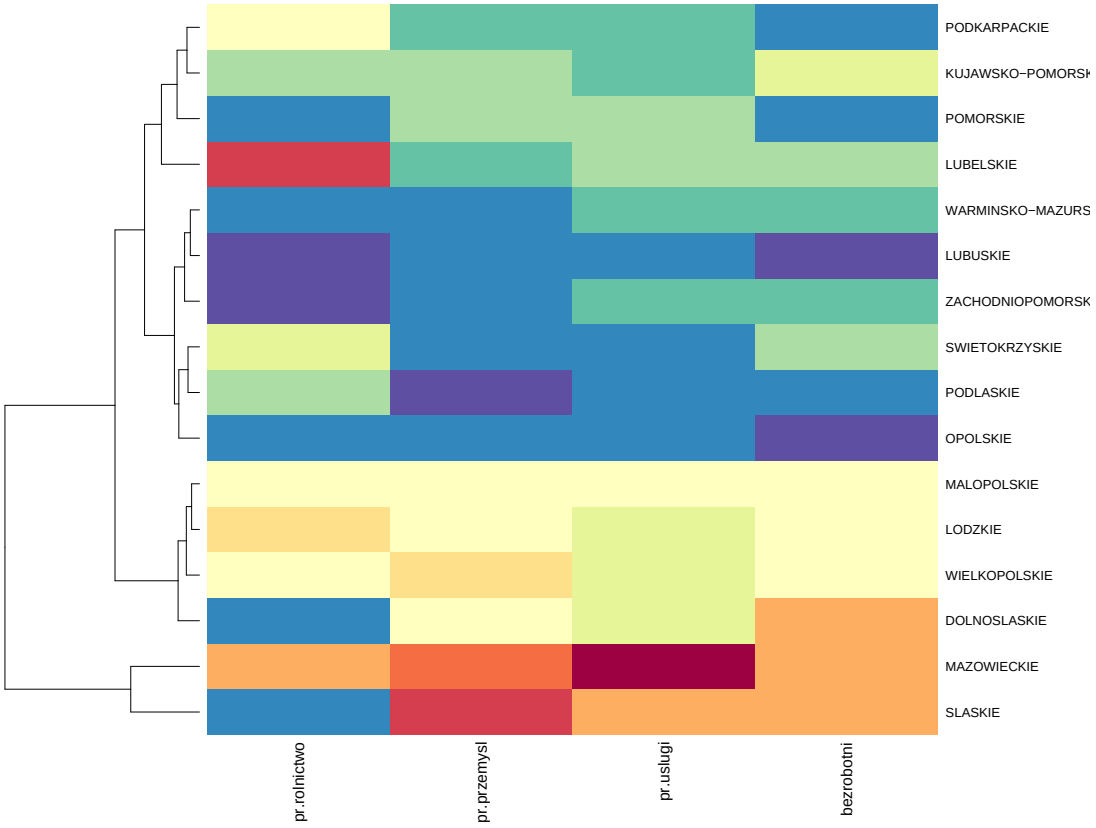
Analizując ten wykres można zauważyć ciekawe zróżnicowanie województw. Na osi X największą współrzędną ma zmienna `pr.rolnictwo`, a więc to ta zmienna odpowiada największemu zróżnicowaniu województw. Wysoki wartości na tej osi osiągają województwa gdzie zatrudnienie w rolnictwie było wyższe niż średnie. Druga oś rozróżnia województwa w których dominuje zatrudnienie w sektorze usług (dodatnie wartości) versus zatrudnieniu w przemyśle lub braku zatrudnienia (niestety profil zatrudnienia w przemyśle i braku zatrudnienia jest podobny).

Osoby w województwach Mazowieckim i Zachodniopomorskim częściej pracują w sektorze usług, w województwach Lubelskim, Podlaskim, Świętokrzyskim i Podkarpackim dominuje zatrudnienie w sektorze rolniczym, w województwach Śląskim i Dolnośląskim dominuje zatrudnienie w przemyśle te województwa mają też większe problemy z bezrobociem.

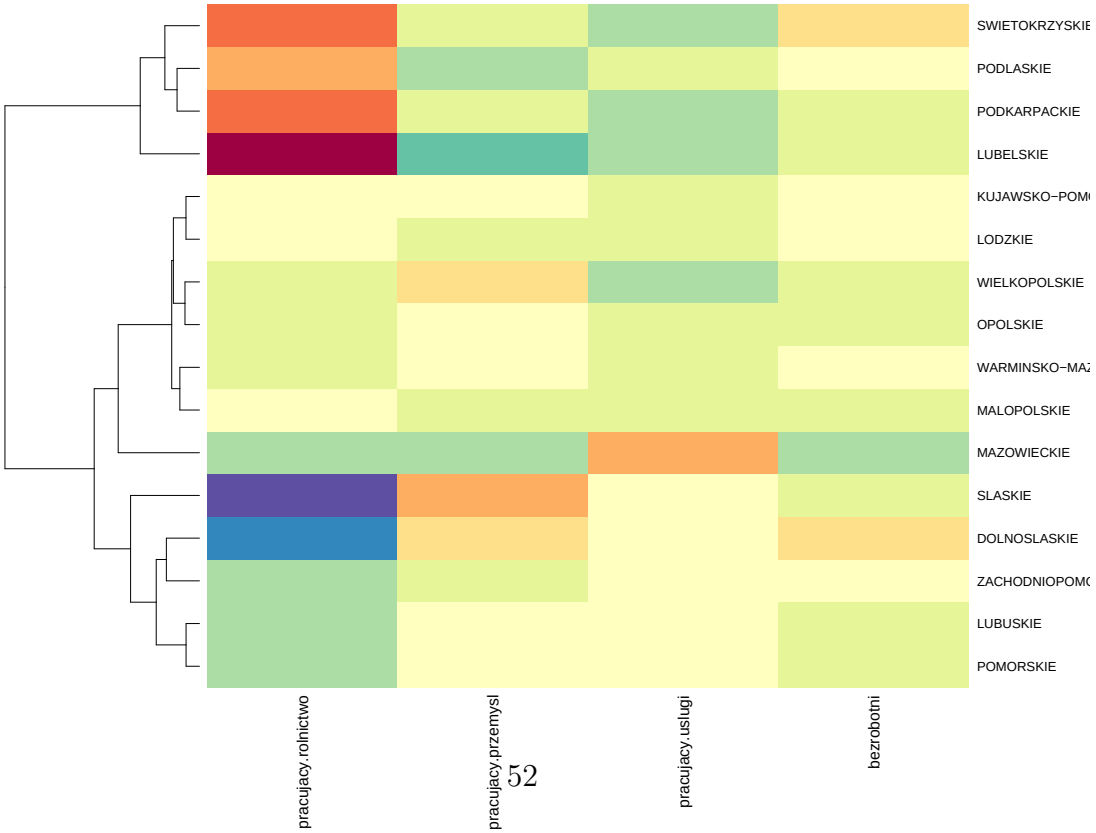
[TODO: opisać podobieństwa i różnice do PCA]



Rysunek 7.1: Przykład analizy korespondencji.



Rysunek 7.2: Mapa ciepła dla danych oryginalnych (kolory normalizowane po kolumnach).



Rysunek 7.3: Mapa ciepła dla reszt Pearsonowskich.

Rozdział 8

Zbiory danych

W tym miejscu przedstawimy zbiory danych wykorzystywane w poprzednich rozdziałach. Są one umieszczone w Internecie i można je ściągnąć podanymi poniżej poleceniami. Jeżeli nie mamy własnych danych, to warto na tych przećwiczyć omawiane w tej książce funkcje.

8.1 Zbiór danych GUSowskich

Korzystając z danych dostępnych na stronach GUSu `http://www.stat.gov.pl/` przygotowałem poniższy wybranych wskaźników w rozbiciu na województwa. Są to prawdziwe dane dotyczące roku 2007.

```
> daneGUS = read.table("http://www.biecek.pl/R/dane/Dane2007GUS.csv", sep=";"
      , h=T, dec=",")
```

W tej ramce danych dla 16 województw zebrane są następujące dane:

1. ludnosc.do.15.lat - liczba mieszkańców w wieku do 15 lat
2. ludnosc.15.60.lat - liczba mieszkańców w wieku od 15 do 60 lat
3. ludnosc..60.lat + - liczba mieszkańców w wieku od 60 lat
4. mieszkan.wyprodukowanych - liczba mieszkań wyprodukowanych w 2007 roku
5. studenci.artystyczne - liczba studentów kierunków artystycznych
6. studenci.spoleczne - liczba studentów kierunków społecznych
7. studenci.ekonomia - liczba studentów kierunków ekonomicznych
8. studenci.prawne - liczba studentów kierunków prawniczych

9. studenci.dziennikarstwo - liczba studentów kierunku dziennikarstwo
10. studenci.biologiczne - liczba studentów kierunków biologicznych
11. studenci.fizyczne - liczba studentów kierunków fizycznych
12. studenci.matematyczno.statystyczne - liczba studentów kierunków matematycznych
13. studenci.informatyczne - liczba studentów kierunków informatycznych
14. studenci.medyczne - liczba studentów kierunków medycznych
15. studenci.inzynieryjno.techniczne - liczba studentów kierunków inżynieryjno technicznych
16. studenci.produkcja.i.przetworstwo - liczba studentów kierunków produkcja i przetwórstwo
17. studenci.architektura.i.budownictwo - liczba studentów kierunków architektura i budownictwo
18. studenci.na.ochrona.srodowiska - liczba studentów kierunku ochrona środowiska
19. studenci.na.ochrona.i.bezpieczenstwo - liczba studentów kierunków ochrona i bezpieczeństwo
20. nauczycieli.akademickich - liczba nauczycieli akademickich
21. dochod.budzetu.na.mieszkanca - średni dochód do budżetu na jednego mieszkańca
22. pracujacy.rolnictwo - liczba (w tysiącach) osób pracujących w sekorze rolniczym
23. pracujacy.przemysl - liczba (w tysiącach) osób pracujących w przemyśle
24. pracujacy.uslugi - liczba (w tysiącach) osób pracujących w sekorze usługowym
25. bezrobotni - liczba (w tysiącach) bezrobotnych
26. powierzchnia.na.mieszkanie - średnia powierzchnia mieszkania
27. powierzchnia.na.osobe - średnia powierzchnia mieszkania na osobe
28. mieszkancow.na.sklep - średnia liczba mieszkańsców na sklep

Bibliografia

[25] **„Statystyczne systemy uczące się”**, Jacek Koronacki i Jan Ćwik. WNT 2005, ISBN: 83-204-3157-3.

[26] **„The Elements of Statistical Learning: Data Mining, Inference, and Prediction”**, Trevor Hastie, Robert Tibshirani, Jerom Friedman. Springer-Verlag 2001.

Skorowidz