Politechnika Warszawska
Wydział Elektroniki i Technik Informacyjnych
Instytut Informatyki

Rok akademicki 2003/2004

PRACA DYPLOMOWA MAGISTERSKA

Krzysztof Rządca

ALGORYTMY GRUPOWANIA DANYCH

Opiekun pracy
dr hab. Franciszek Seredyński

Ocena:

Podpis Przewodniczącego
Komisji Egzaminu Dyplomowego

Życiorys

Streszczenie

Zagadnienie grupowania polega na podziale zbioru wejściowego na mniejsze grupy, przy czym byty zaliczone do tej samej grupy powinny być do siebie jak najbardziej podobne, natomiast byty zaliczone do różnych grup – jak najbardziej różniące się od siebie. W tej pracy proponuję nową taksonomię algorytmów grupowania i charakteryzuję przy jej użyciu kilka znanych z literatury podejść do tego problemu. Następnie opisuję niedawno opublikowany algorytm grupowania maksymalnej wariancji (Maximum Variance Cluster algorithm, MVC) i proponuję algorytm IMVC, sposób na proste i wydajne wyznaczanie jedynego parametru algorytmu. Ważną częścią pracy są eksperymenty, w których badam działanie MVC, porównuję MVC z algorytmem k-środków (najczęściej używanym algorytmem grupowania), oraz analizuję IMVC. Badania przeprowadzam zarówno na wygenerowanych, jak i na rzeczywistych zbiorach danych. Uzyskane rezultaty pokazują wyraźną przewagę MVC w grupowaniu zbiorów wygenerowanych, natomiast nie ukazują istotnych różnic w grupowaniu zbiorów rzeczywistych.

Słowa kluczowe: grupowanie, badanie tendencji grupowania, MVC, klasteryzacja

---

Data Clustering Algorithms

Abstract

Data clustering problem consists of finding partition of given data in terms of similarity. A new hierarchy of clustering algorithms is proposed and some well-known algorithms are described using this hierarchy. Then, a recently proposed Maximum Variance Clustering algorithm (MVC) is described along with a straightforward and efficient way to discover clustering tendencies in data using MVC, called IMVC. The approach shares the benefits of the plain clustering algorithm with regard to other approaches for clustering. Experiments using both synthetic and real data have been performed in order to compare MVC with the well-known k-means algorithm and evaluate the differences between the proposed methodology and the plain use of the Maximum Variance Clustering algorithm. According to the results obtained, MVC is more accurate on syntethic data sets, but does not show advantages on real data sets. However, assessing clustering tendecy using IMVC constitutes an efficient and accurate alternative to MVC on almost all datasets considered.

Keywords: clustering, cluster tendency, MVC

Spis treści

Rozdział 1

Wstęp

> *"Forty-two!" yelled Loonquawl. "Is that all you've got to show for seven and a half million years' work?"*
> *"I checked it very thoroughly," said the computer, "and that quite definitely is the answer. I think the problem, to be quite honest with you, is that you've never actually known what the question is."*
> *"But it was the Great Question! The Ultimate Question of Life, the Universe and Everything!" howled Loonquawl.*
>
> „The Hitchhiker's Guide to the Galaxy", Douglas Adams.

W ciągu ostatnich kilkudziesięciu lat nastąpił ogromny wzrost mocy obliczeniowej komputerów. Pierwsze „maszyny liczące" – mechaniczny komputer Charles'a Babbage'a, czy pierwsze komputery z serii ENIAC – były w swej istocie trochę bardziej skomplikowanymi kalkulatorami. Oczekiwały pewnych symboli na wejściu i zwracały symbole na wyjściu, wykonując na nich szereg zapisanych wcześniej poleceń. Teoretycznym modelem obliczeń dla tych komputerów była (i nadal pozostaje) maszyna Turinga, wyposażona w taśmę na dane i zapis rezultatów, pewny określony zbiór prostych rozkazów i wykonująca operacje na **danych** – napisach, czyli ciągach symboli z zadanego alfabetu. Dane są ze swej istoty asemantyczne, dlatego też maszyna Turinga nie dokonuje (bo nie może dokonać) żadnej interpretacji dostarczonych wiadomości, oczekuje danych (napisów) na wejściu i zwraca dane (napisy) na wyjściu. Właściwe odczytanie tych danych

jest zadaniem operatora maszyny.

Moc obliczeniowa komputerów wzrastała, rosło również szeroko rozumiane obycie ludzi z nimi i w pewnym momencie nastąpił istotny, logiczny przełom w informatyce. Wraz z pierwszymi bazami danych komputery zaczęły operować na **informacji**. Informacja to dane, napisy złożone z symboli wcześniej zdefiniowanego alfabetu, ale również sposób ich interpretacji. „18041979" to napis nad pewnym alfabetem, możemy domniemywać, że nad alfabetem złożonym z cyfr dziesiętnych, natomiast ten sam napis umieszczony w tabeli bazy danych, w kolumnie „data urodzenia" jest już informacją. Inna definicja informacji mówi, że jest to coś, co zmniejsza niepewność odbiorcy. Po zadaniu dobrze sformułowanego zapytania do bazy danych, otrzymany rezultat zmniejsza naszą niepewność, na przykład poprzez przypomnienie czyjegoś dnia urodzin.

W dzisiejszych czasach bazy danych obecne są w tak wielu dziedzinach życia i przechowują tak wiele informacji, że stają się krytycznym zasobem wielu organizacji. Trudno wyobrazić sobie funkcjonowanie dużych firm bez baz danych, bo informacji w nich przechowywanych jest zbyt dużo, by przetwarzali je ludzie. Dlatego też, wraz z przyrostem informacji magazynowanej w bazach, a również i z rozwojem literatury SF, ludzie próbowali zaprogramować komputery w sposób taki, by nie tylko magazynowały one i w prosty sposób przetwarzały informacje, ale były również zdolne operować nimi na wyższym stopniu abstrakcji, wnioskując, uogólniając, przewidując. Tak narodził się mit sztucznej inteligencji, którym to mitem ludzie zafascynowani są od co najmniej lat siedemdziesiątych ubiegłego wieku. Sztuczna inteligencja pozwoli przejść komputerom na wyższy poziom. Będzie to przejście od operowania na danych przez maszyny Turinga, poprzez gromadzenie i przetwarzanie informacji w bazach, do operacji na **wiedzy**. Niestety nie powstała dobra definicja pojęcia wiedzy, intuicyjnie rozumiemy to jako coś, co różni ucznia od nauczyciela, doświadczonego pilota od pilota początkującego, ale również i szefa działu personalnego firmy od bazy danych o pracownikach.

Skonstruowanie bytu naśladującego inteligencję ludzką, tzw. silnej sztucznej inteligencji, okazało się zdumiewająco trudne. Z każdą nową próbą dowiadujemy się o kolejnych kłopotach, niemożliwych do wyobrażenia wcześniej. Dlatego

też informatycy próbują dziś, nie bez sukcesów, budować tzw. słabe sztuczne inteligencje, których działanie ograniczone jest do pewnych ściśle określonych zadań, np. automatycznej klasyfikacji nowych informacji w oparciu o informacje już sklasyfikowane, ekstrakcji informacji z artykułów, książek i stron internetowych, czy odkrywania asocjacji (zależności typu *jeżeli* ... *to* ...) w bazach danych.

Moja praca traktuje o jednym z działów sztucznej inteligencji, jakim jest grupowanie. W zadaniu tym na wejściu dany jest pewien zbiór bytów opisany przez określone atrybuty. Grupowanie polega na podziale tego zbioru na mniejsze grupy, przy czym byty zaliczone do tej samej grupy powinny być do siebie maksymalnie podobne, natomiast byty zaliczone do różnych grup – maksymalnie różniące się od siebie. Wynik grupowania może być cenny, ponieważ grupy stanowią **uogólnienie** informacji danej na wejściu. Przykładowo, kilka milionów klientów sieci telefonii komórkowej, liczba, z którą nie jest w stanie poradzić sobie żaden człowiek, może być podzielone na kilka lub kilkanaście grup, w których można znaleźć „klienta charakterystycznego". Takie uogólnienie pozwala ogarnąć dostępną informację i spojrzeć na nią z innej perspektywy, można zatem powiedzieć, że w procesie grupowania konieczna jest, być może sztuczna, inteligencja, a sam wynik jest wiedzą.

Praca ta skonstruowana jest następująco. W rozdziale 2 precyzuję zagadnienie grupowania, proponuję taksonomię dla istniejących algorytmów i omawiam wybrane podejścia. W rozdziale 3 opisuję algorytm grupowania Maximum Variance Clusterer (MVC) oraz stworzony przeze mnie algorytm IMVC, pozwalający w prosty i wydajny sposób na wyznaczanie parametrów dla MVC. W rozdziale 4 zawarte są wyniki przeprowadzonych eksperymentów oraz porównanie działania MVC z algorytmem k-środków. W rozdziale 5 podsumowuję pracę i przedstawiam propozycje do dalszych badań.

Rozdział 2

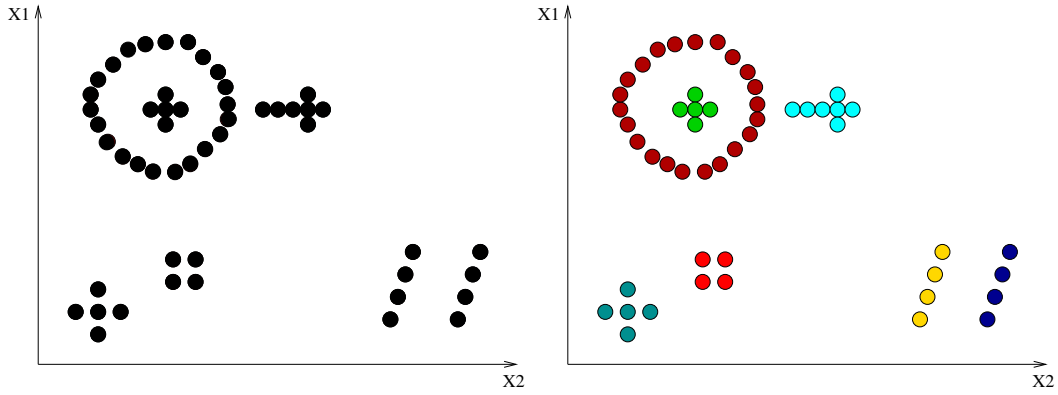
Wprowadzenie do zagadnienia grupowania

2.1 Wstęp

Zadaniem **grupowania** (uczenia się bez nadzoru, klasteryzacji, ang. clustering) jest podział danego zbioru elementów na podzbiory (kategorie, grupy). Elementy każdego z podzbiorów (kategorii) powinny być bardziej podobne do innych elementów z tego samego podzbioru (kategorii), niż do elementów innych podzbiorów([7]).

Przykładowy rezultat grupowania zbioru dwuwymiarowych danych przedstawiam na rysunku 2.1.

Ważne jest zrozumienie różnicy pomiędzy klasyfikacją a grupowaniem. W zadaniu klasyfikacji dany jest pewien zbiór etykietowanych przykładów, a trudność polega na sklasyfikowaniu (nadaniu etykiety) nowego przykładu. W grupowaniu zbiór, który mamy do dyspozycji, w ogóle nie zawiera etykiet. Grupowanie może być postrzegane jako nadanie etykiet nieetykietowanym elementom, ale trzeba pamiętać, że etykiety te wynikają tylko i wyłącznie z wiedzy *explicite* zapisanej w danych.



a) zbiór danych b) rezultat grupowania

Rysunek 2.1: Przykładowy rezultat grupowania zbioru danych o dwóch atrybutach: X_1 i X_2. Przykłady należące do tego samego klastra oznaczone zostały tym samym kolorem.

2.2 Zastosowania

Umiejętność grupowania podobnych przedmiotów jest jedną z fundamentalnych składowych szeroko rozumianej inteligencji. Zwykle pierwszym krokiem do zrozumienia jakiegoś zjawiska jest wyodrębnienie grupy zjawisk podobnych i próba uogólnienia ich cech. Praktycznie w każdej dziedzinie nauki znane są pewne grupy, często zhierarchizowane, które pomagają w ustaleniu dokładnego tematu debaty, artykułu czy wypowiedzi. Przykładem może tu być klasyfikacja gatunków w biologii. Każdy nowo odkryty gatunek jest przypisywany do określonej rodziny, rzędu, etc., co pomaga w ustaleniu cech wspólnych z innymi gatunkami oraz w przypadku prób klasyfikacji poszczególnych egzemplarzy. W tym biologicznym przykładzie grupowaniem można by nazwać stworzenie od nowa całej hierarchii gatunków.

Innym, ważnym zadaniem grupowania jest wyznaczenie przykładów o wartościach izolowanych, niepasujących do pozostałych. Podczas przeprowadzania eksperymentu składającego się z pewnej liczby pomiarów-przykładów, pojawienie się takich izolowanych przykładów może świadczyć o tzw. grubym błędzie pomiarowym, albo o zjawisku, które jest w pewien sposób ciekawe (odbiegające od reszty).

W informatyce grupowanie używane jest zarówno jako wstępny krok analizy danych, jak i pełnoprawne narzędzie badawcze. Do najczęstszych zastosowań należy m.in. (za [13]):

- eksploracja danych (*data mining*), gdzie grupowanie używane jest np. do podziału klientów na pewne podgrupy;
- segmentacja obrazu (*image segmentation*), czyli podział obrazu na regiony homogeniczne pod względem pewnej własności obrazu (kolor, tekstura, intensywność). Taki uproszczony obraz jest prostszy do obróbki np. przez algorytmy rozpoznawania obrazu;
- rozpoznawanie obrazu;
- ekstrakcja informacji (*information retrieval*), mająca za zadanie uporządkowanie i uproszczenie dostępu do informacji. Do klasycznych zastosowań należy stworzenie klasyfikacji książek, czy stron internetowych;
- grupowanie zadań w problemie harmonogramowania tak, by zadania intensywnie ze sobą komunikujące się trafiły do tej samej grupy. Taka grupa zostanie w następnym kroku przypisana do wykonania na jednym procesorze (bądź kilku, ale połączonych szybkimi kanałami komunikacyjnymi) [20].

2.3 Używane oznaczenia

Zakładam, że $X = \{x_1, x_2, \ldots, x_N\}$ jest zbiorem N przykładów (punktów, wektorów danych), które należy pogrupować. Każdy z przykładów jest wektorem d atrybutów $x_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,d})$. d to wymiar przestrzeni, w której dokonujemy grupowania.

Atrybuty przykładu mogą należeć do jednej z podanych grup:

1. Atrybuty ilościowe
   (a) o wartościach ciągłych (np. waga)
   (b) o wartościach dyskretnych (np. liczba okien)

2. Atrybuty jakościowe

   (a) nominalne (np. kolor)

   (b) porządkowe (np. wielkość opadów: bez opadów, małe opady, duże opady).

$C = \{C_1, C_2, \ldots, C_M\}$to zbiór M klastrów (grup), które będą wynikiem grupowania (podziału) zbioru X. W dalszej części pracy słów „grupa" i „klaster" używał będę wymiennie. Grupowanie to może być również opisane macierzą $U = [u_{ik}]$, przy czym wartość u_{ik} to stopień w jakim przykład x_k należy do klastra C_i. Gdy klastry nie są rozmyte, $u_{ik} = 1$, gdy $x_k \in C_i$, $u_{ik} = 0$ w przeciwnym przypadku.

2.4 Proponowana taksonomia

Zadanie grupowania jest zdefiniowane bardzo szeroko i nieprecyzyjnie. Istnieje wiele różnych typów i rodzin algorytmów, stosowanych w różnych sytuacjach. Ponieważ podziały między nimi zachodzą w wielu różnych płaszczyznach, trudno jest narzucić jedną uniwersalną klasyfikację lub hierarchię. Dwa klasyczne przeglądy algorytmów grupowania – Cichosza[7] i Jain'a[13] proponują zupełnie różny podział tych technik. Cichosz koncentruje się na typie danych wejściowych (dzieli dane na dyskretne i ciągłe), z kolei Jain podaje ogólną taksonomię i kilka przykładowych, najbardziej popularnych podejść do tej dziedziny, nie starając się pokazać zależności pomiędzy algorytmami.

W swojej książce[14] autorzy proponują następujący podział zadania grupowania na komponenty:

1. Wybór reprezentacji przykładów (włączając *feature extraction* lub selekcję atrybutów).

2. Zdefiniowanie funkcji będącej miarą podobieństwa pomiędzy przykładami.

3. Klasteryzacja lub grupowanie.

4. Abstrakcja rezultatów.

5. Ocena rezultatów.

W typowych zastosowaniach bardzo często występuje sprzężenie zwrotne. Po uzyskaniu wyników z kolejnego kroku wraca się do poprzednich, wybierając np. inną reprezentację przykładów lub inną funkcję odległości.

Podział ten niezbyt dobrze opisuje niektóre algorytmy grupowania, takie jak COBWEB czy CLUSTER/2. W tych algorytmach bardzo dużą rolę odgrywa wybór właściwej reprezentacji **klastra**, nie tylko przykładów. Innym istotnym mankamentem podanego wyżej podziału, który wynika w pewnej mierze z niedocenienia roli klastrów, jest to, że w większości algorytmów mierzy się **podobieństwo między klastrem a przykładem**, a nie pomiędzy dwoma przykładami. Oczywiście jeśli wybierzemy reprezentację klastrów sprowadzającą je do przestrzeni przykładów, te dwie miary są identyczne. Niemniej jednak w wielu algorytmach takiej projekcji stworzyć nie można lub jest ona niewygodna. Kolejną niedoskonałością, wynikającą z poprzedniej, jest nieoddzielenie przestrzeni rozwiązań w jakiej porusza się dany algorytm grupowania (czyli „pomysłu na algorytm") od sposobu jej przeglądania. Utrudnia to właściwą ocenę nowych algorytmów, bo algorytm proponujący całkowicie nowe podejście do problemu jest przecież dużo bardziej innowacyjny od algorytmu opisującego tylko lepszy sposób poszukiwania najlepszego rozwiązania, np. przy użyciu którejś ze znanych technik optymalizacji globalnej (algorytmów genetycznych, czy symulowanego wyżarzania).

Dlatego na potrzeby tej pracy proponuję następujący podział zadania grupowania:

1. Wybór reprezentacji przykładów (włączając w to *feature extraction* lub selekcję atrybutów)
2. Wybór modelu – reprezentacji klastrów i pewnych warunków, jakie muszą być spełnione w wyniku (ograniczeń)
3. Zdefiniowanie funkcji będącą miarą podobieństwa pomiędzy klastrem a przykładem
4. Zdefiniowanie funkcji oceny grupowania (i funkcji oceny klastra, z której korzysta funkcja oceny grupowania)

5. Grupowanie, czyli przeglądanie przestrzeni w jakiej opisane są klastry w celu znalezienia rozwiązania optymalnego z punktu widzenia funkcji zdefiniowanej w 3

6. Abstrakcja rezultatów

7. Ocena rezultatów

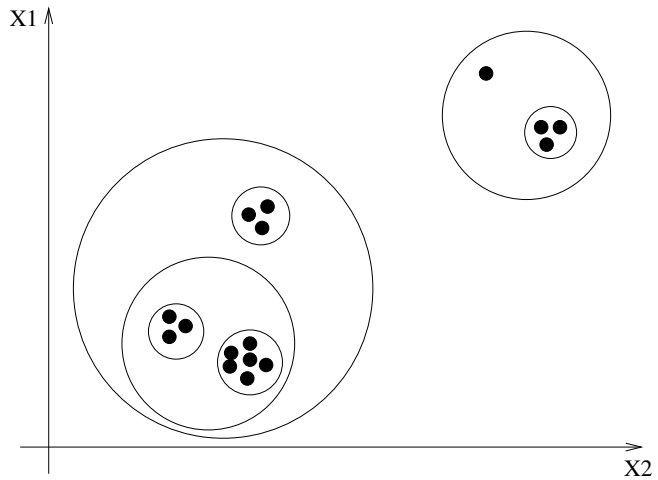
W tym podziale krok grupowania (5) jest wyraźnie wyodrębniony. Widać, że można używać zarówno klasycznych, zazwyczaj zachłannych, heurystyk, jak i znanych metod optymalizacji globalnej, jak algorytmy genetyczne, sieci neuronowe, czy symulowane wyżarzanie.

Zwykle funkcja oceny klastra (4) to suma odległości przykładów należących do klastra od jego środka, choć zdażają się również funkcje uwzględniające np. liczbę przykładów w klastrze (np. wariancja).

Taka taksonomia dobrze opisuje algorytmy, które grupują optymalizując pewną funkcję celu. Do tej grupy można zaliczyć większość współcześnie używanych i popularnych algorytmów. Niestety do tej taksonomii nie pasują algorytmy zaliczające się do grupy **algorytmów grafowych (graph theoretic)**, w większości opracowane i rozwijane w latach siedemdziesiątych XX w. Algorytmy te narzucają grafową reprezentację klastra, a ponieważ nie optymalizują żadnej funkcji, nie używają np. funkcji oceny klastra. Ze względu na te różnice zostaną one omówione w oddzielnym rozdziale (2.8). Należy przyjąć, że algorytm taki zastępuje kroki 2-5 w powyższej taksonomii.

Dla uproszczenia dalszego opisu zdefiniuję jeszcze dwa podziały, w pewnej mierze nakładające się na kroki 2 i 3. Można je oczywiście włączyć do któregoś z nich (albo do obydwu), ale wydaje mi się, że byłoby to niewygodne i w dużej mierze niepotrzebne. Podziały te można rozumieć jako kolejne wymiary proponowanej przeze mnie taksonomii.

Algorytmy grupowania można podzielić na algorytmy **hierarchiczne** i **dzielące (partitional)**. Algorytmy hierarchiczne dostarczają od razu całą hierarchię możliwych podziałów, w której grupy na poziomie n składają się zazwyczaj z kilku grup z poziomu $n-1$. Przykład działania takiego algorytmu przedstawiony jest



Rysunek 2.2: Przykładowy rezultat grupowania hierarchicznego. Klastry przedstawione są w postaci okręgów. Hierarchia klastrów reprezentowana jest przez zawieranie się okręgów. Na rysunku nie zaznaczono wszystkich klastrów.

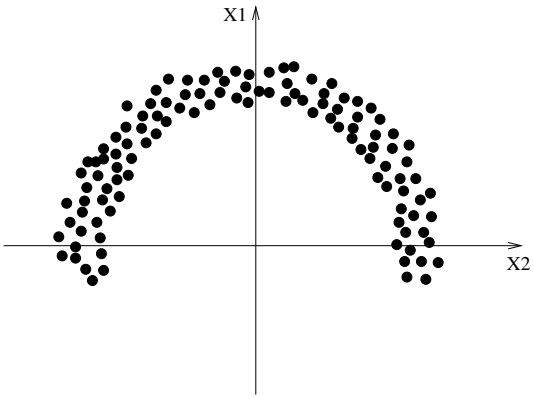
na rysunku 2.2. Algorytmy dzielące proponują jeden możliwy podział, będący lokalnym lub globalnym maksimum funkcji celu zdefiniowanej w kroku 4. Do tej grupy zalicza się najczęściej używany algorytm k-środków[17].

Proponowany przez algorytm podział zbioru na grupy może być **twardy** (*hard*) lub **rozmyty** (*fuzzy*). W grupowaniu twardym każdy przykład należy do dokładnie jednego klastra. W grupowaniu rozmytym przykłady mogą należeć do kilku klastrów. Stopień przynależności do każdego z klastrów określony jest współczynnikiem przynależności (z przedziału $[0, 1]$), przy czym dla każdego przykładu suma współczynników musi być równa 1. Przykładowy rezultat działania tych dwóch metod przedstawiony jest na rysunku 2.3.

Wybór właściwej reprezentacji przykładów może być kluczowy dla powodzenia całego algorytmu (rysunek 2.4), tym bardziej, że jest pierwszym krokiem w przedstawionym podziale zadania grupowania. Z wybranej reprezentacji korzystają wszystkie pozostałe kroki. Jest to zadanie bardzo trudne i zależne od rodzaju i postaci dostępnych danych, jak też i od doświadczenia osoby przeprowadzającej eksperyment. Z drugiej strony w wielu innych działach maszynowego uczenia się (np. w klasyfikacji, czy w poszukiwaniu reguł asocjacyjnych) również przeprowa-

a) hard b) fuzzy

Rysunek 2.3: Przykładowy rezultat grupowania twardego (a) i rozmytego (b). Na rysunku (b) im ciemniejszy jest kolor przykładu, tym większy jest współczynnik przynależności do klastra zawierającego wszystkie przykłady oznaczone na czarno.



Rysunek 2.4: Wybór właściwej reprezentacji przykładów może być kluczowy dla powodzenia algorytmu grupowania. Przy reprezentacji we współrzędnych euklidesowych tego zbioru danych wiele algorytmów zaproponowałoby grupowanie złe, dzieląc zbiór danych na kilka klastrów. Używając współrzędnych biegunowych, algorytmy zachowywałyby się poprawnie.

dza się podobne przekształcenia atrybutów, zwane konstruktywną indukcją ([7]). Z powyższych przyczyns wybór właściwej reprezentacji przykładów nie wchodzi w skład tego opracowania.

2.5 Reprezentacja klastrów i miara podobieństwa

W tym podrozdziale omówię najbardziej znane reprezentacje klastrów. W większości przypadków miara podobieństwa pomiędzy klastrem a przykładem (krok 3 proponowanej taksonomii) ściśle zależy od wyboru reprezentacji klastrów. Czasem istnieje kilka możliwych sposobów mierzenia podobieństwa w ramach jednej reprezentacji, ale bardzo rzadko daje się zastosować tę samą miarę dla różnych reprezentacji. Dlatego też te dwa kroki zadania grupowania zostaną opisane razem.

Wybór jednego z kilku możliwych sposobów mierzenia podobieństwa jest uzależniony od postaci danych wejściowych, poprzednich wyników grupowania przy użyciu innych miar i rodzaju rezultatów, które chcemy otrzymać.

2.5.1 Reprezentacja poprzez środek klastra

W tej, koncepcyjnie najprostszej, metodzie reprezentacji klastry przedstawiane są poprzez swoje **środki**. Reprezentacja ta jest bardzo chętnie używana, zwłaszcza gdy dane wejściowe są ciągłe, gdyż można wtedy łatwo wyznaczyć taki środek jako (zazwyczaj) średnią wartość atrybutów przykładów należących do klastra. Gdy operujemy na danych nominalnych, takich jak „kolor", trudno jest znaleźć właściwą interpretację środka. Co powinno być środkiem dla wartości „czerwony" i „zielony"? „Biały", „czarny", którakolwiek z podanych wartości, czy może powinien on być niezdefiniowany? Ale jak wtedy określić miarę odległości mówiącą, że „czerwony" jest bliżej takiego niezdefiniowanego środka niż np. „niebieski"?

Z powodu tych trudności ta reprezentacja jest używana szczególnie chętnie w zastosowaniach rozpoznawania wzorców (*pattern recognition*), gdzie z reguły nie występują dane nominalne.

Ta reprezentacja pozostawia najwięcej miejsca na wybór miary podobieństwa między klastrem a punktem. Najczęściej używane miary przedstawię w następnych podrozdziałach. Ponieważ w tej reprezentacji klastra jego środek można traktować jako przykład, podawane przeze mnie miary będą odległościami mię-

dzy dwoma przykładami (w tej formie występują w literaturze).

Miara Euklidesowa

W mierze Euklidesowej przyjmuje się, że im mniejsza odległość przykładu od środka klastra, tym bardziej przykład ten jest do klastra podobny. Odległość Euklidesowa d-wymiarowych przykładów x_i i x_j dana jest wzorem

$$d_2(x_i, x_j) = (\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2)^{\frac{1}{2}}$$

i jest to szczególny przypadek ($p = 2$) miary Minkowskiego:

$$d_p(x_i, x_j) = (\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^p)^{\frac{1}{p}}.$$

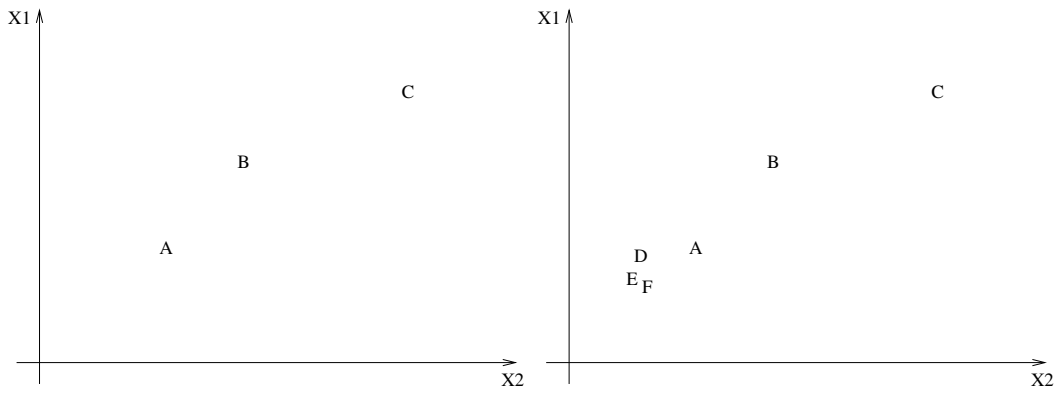
Miary te mają intuicyjną interpretację, jako że zwykle używamy ich, by ocenić odległość dwóch przedmiotów w dwu- lub trójwymiarowej przestrzeni. Dają dobre wyniki, jeśli w danych można wyróżnić klastry leżące daleko od siebie i mające kulisty kształt. Największą wadą stosowania takich miar jest tendencja do dominacji atrybutu o największych liczbowo wartościach nad pozostałymi, mniejszymi. Najprostszym rozwiązaniem jest normalizacja wszystkich atrybutów, polegająca na przeskalowaniu przedziałów, do których należą poszczególne atrybuty, na przedział $[0, 1]$. Dla k-tego atrybutu i-tego przykładu normalizacja ta wygląda następująco:

$$x_{i,k} := \frac{x_{i,k} - \min_j(x_{j,k})}{\max_j(x_{j,k}) - \min_j(x_{j,k})}, \tag{2.1}$$

gdzie $\min_j(x_{j,k})$ jest najmniejszą wartością k-tego atrybutu w danych, a $\max_j(x_{j,k})$ jest wartością największą.

Miary kontekstowe

Miary te biorą pod uwagę **kontekst** przykładu, zdefiniowany jako zbiór przykładów otaczających przykład. Miara kontekstowa pomiędzy dwoma przykładami x_i i x_j jest zdefiniowana jako:



a) A i B są bardziej podobne niż B i C

b) B i C są bardziej podobne niż A i B

Rysunek 2.5: Przykład działania miary MND. Na rysunku b przykład A ma przykłady bliższe niż B, w związku z czym stopień podobieństwa z B zmniejsza się. Za [13].

$$s(x_i, x_j) = f(x_i, x_j, \xi), \tag{2.2}$$

gdzie ξ jest kontekstem.

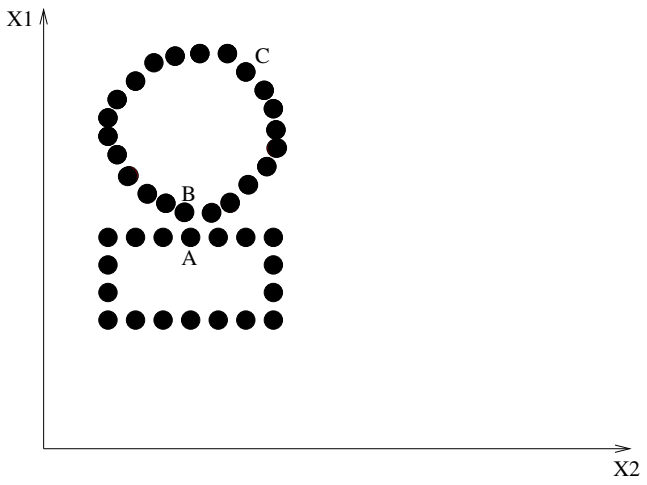
Przykładem takiej miary jest *mutual neighbor distance* (MND), zdefiniowana jako

$$MND(x_i, x_j) = NN(x_i, x_j) + NN(x_j, x_i), \tag{2.3}$$

gdzie $NN(x_i, x_j)$ jest indeksem przykładu x_j na uporządkowanej pod względem używanej miary odległości liście sąsiadów przykładu x_i.

Przykład działania MND przedstawiam na rysunku 2.5. Na części a przykład A jest najbliżej przykładu B, a przykład B najbliżej przykładu A, zatem $NN(A,B) = NN(B,A) = 1$, czyli $MND(A,B) = 2$. Najbliżej przykładu C leży przykład B, czyli $NN(C,B) = 1$, natomiast przykład C jest drugim co do odległości od przykładu B, zatem $NN(B,C) = 2$, co daje $MND(B,C) = 3$. Po dodaniu trzech nowych przykładów D,E i F MND dla przykładów A,B zwiększa się $MND(A,B) = 5$, dzięki czemu przykłady B i C są do siebie teraz bardziej podobne.

MND nie jest miarą w sensie topologicznym, gdyż nie spełnia nierówności trójkąta. Pomimo tego w grupowaniu używana jest z dużymi sukcesami.



Rysunek 2.6: Stosowanie miary pojęciowej powinno prowadzić do podziału przedstawionego zbioru danych na dwie grupy: „owal” i „prostokąt”, co jest możliwe, gdy odległość przykładów B i C jest mniejsza od odległości przykładów A i B. Za [13].

Miary pojęciowe

Miary te biorą pod uwagę nie tylko kontekst przykładów, ale również zbiór pewnych *pojęć*. Ogólny wzór na odległość według takiej jest zdefiniowany jako:

$$s(x_i, x_j) = f(x_i, x_j, \zeta, \xi),$$

gdzie ζ jest zbiorem zadanych a priori pojęć. Znaczenie „pojęcia” może być bardzo różne w zależności od zastosowania.

Przykład stosowania tej miary prezentowany jest na rysunku 2.6. By zbiór danych przedstawiony na rysunku został podzielony na dwie grupy: „owal” i „prostokąt”, konieczne jest, by odległość pomiędzy przykładami B i C była mniejsza od odległości pomiędzy przykładami A i B, chociaż odległość euklidesowa B od C jest większa od odległości pomiędzy A i B. Jest to możliwe, jeżeli ustalimy, że B i C należą do tego samego pojęcia ζ_i (owalu), a przykład A należy do innego pojęcia ζ_j (prostokąta).

Z praktycznego punktu widzenia miary te właściwie nie są stosowane. Trudno jest bowiem zadać pewien zbiór pojęć, trudno jest również dopasować do tych pojęć przykłady.

2.5.2 Reprezentacja poprzez rozkłady prawdopodobieństwa

Na grupowanie można patrzeć też jako na zadanie modelowania mieszanin. Zakładamy, że przykłady pochodzą z pewnego skończonego zbioru prostych rozkładów prawdopodobieństwa o nieznanych parametrach. Badania empiryczne wykazały, że rozkład normalny (Gaussa) w wielu przypadkach najlepiej opisuje klastry istniejące w rzeczywistym świecie. Rozkładem tym można opisać zmienne losowe takie jak np. waga oraz wzrost osobników jednorodnych populacji ludzkich lub zwierzęcych, plon na jednakowych poletkach doświadczalnych, czy losowe błędy pomiarów ([15]).

Zmienna losowa X ma rozkład normalny o parametrach μ (średnia) oraz Σ (macierz kowariancji), oznaczany jako $N(\mu,\Sigma)$, jeśli jej funkcja gęstości ma następującą postać

$$f(x) = \frac{\exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}}{(2\pi)^{\frac{D}{2}}\,|\Sigma|^{\frac{1}{2}}}, \tag{2.4}$$

gdzie μ jest środkiem klastra, a Σ macierzą kowariancji, zaś D to liczba wymiarów przykładów $x_i \in X$.

Podobnie jak reprezentacja poprzez środek klastra (2.5.1), gdy używa się rozkładu Gaussa, ta reprezentacja również dostosowana jest raczej do danych ciągłych. Oczywiście można przystosować ją do danych nominalnych poprzez wybranie odpowiednich rozkładów prawdopodobieństwa.

Algorytmy używające takich reprezentacji obficie korzystają ze znanych i dopracowanych metod statystyki matematycznej.

W tej reprezentacji przyjmuje się, że odległość przykładu x od klastra C jest wartością funkcji gęstości klastra C w punkcie x przemnożonej przez wagę klastra p_c. Waga klastra ustalona jest globalnie, jest taka sama dla wszystkich przykładów. Funkcję taką przyjęło się nazywać w statystyce *funkcją wiarygodności* próbki.

Używanie funkcji wiarygodności jako miary odległości pomiędzy klastrem a przykładem ma wiele zalet. Przede wszystkim wynikowe klastry są rozmyte, dzięki czemu wynik zawiera dużo więcej informacji. Stopień przynależności $u_{i,c}$

przykładu x_i do klastra C_c określa się jako:

$$u_{i,c} = \frac{f_c(x_i) * p_c}{\sum_{d \in C} f_d(x_i) * p_d},$$

gdzie C to zbiór wszystkich klastrów.

2.5.3 Reprezentacja poprzez kompleksy

Kompleks jest zwięzłym opisem hipotezy, czyli w tym znaczeniu proponowanego przez algorytm klastra. Możemy wyobrazić sobie klaster stworzony przez dwa przykłady: („biały”, „okrągły”, „wysoki”) i („czarny”, „okrągły”, „wysoki”). Jedną z możliwych hipotez, czyli opisów takiego klastra, jest to, że zawiera on wszystkie przykłady „okrągłe” i „wysokie”, natomiast w zależności od tego czy atrybut kolor ma dwie, czy więcej wartości, może być on nieistotny (wtedy hipoteza o nim nie wspomni), albo mający dwie możliwe wartości – wtedy klaster opisany jest jako „biały” lub „czarny”, „okrągły” i „wysoki”.

Kompleks nadaje takim hipotezom formę zwięzłą i dającą się przetwarzać komputerowo. Każdy z atrybutów ma odpowiadający *selektor*, opisujący możliwe jego wartości. Istnieją cztery rodzaje warunków nakładanych na atrybut:

selektor pojedynczy: spełniony, jeśli odpowiadający atrybut ma wartość wskazaną przez selektor;

selektor dysjunkcyjny: określający zbiór możliwych wartości danego atrybutu;

selektor uniwersalny: dopuszczający wszystkie wartości danego atrybutu;

selektor pusty: nie dopuszczający żadnej wartości;

Reprezentacja poprzez kompleksy jest szczególnie skuteczna w zastosowaniach, w których występują atrybuty o stosunkowo małym zbiorze wartości, gdyż zbiory możliwych wartości w selektorze dysjunkcyjnym zazwyczaj określa się poprzez wyliczenie wszystkich elementów. Jest to reprezentacja chętnie używana w grupowaniu pojęciowym, części maszynowego uczenia się.

Funkcja podobieństwa pomiędzy klastrem określonym przez kompleks a przykładem jest zwykle binarna. Gdy kompleks jest z przykładem zgodny (wszystkie selektory są spełnione), funkcja przyjmuje wartość 1, co oznacza, że przykład należy do klastra. Gdy którykolwiek z selektorów jest niezgodny z odpowiadającą mu wartością atrybutu, przykład nie należy do klastra, a funkcja przyjmuje wartość 0. Można wyobrazić sobie funkcję przyjmującą wartości z przedziału $[0,1]$ w zależności od liczby spełnionych selektorów, lub nawet stopnia ich (nie-)spełnienia (np. kara za niespełnienie selektora pojedynczego jest większa, niż za niespełnienie selektora dysjunkcyjnego). Funkcja taka dawałaby klastry rozmyte. W praktyce jednak takie funkcje nie są używane, głównie z powodu zwiększenia rozmiaru przestrzeni możliwych rozwiązań, którą algorytm przeszukiwania musiałby przejrzeć.

2.6 Funkcja oceny grupowania

Zwykle jako funkcji oceny proponowanego grupowania używa się sumy miar odległości przykładów od klastrów, do których te przykłady zostały zaklasyfikowane:

$$fitness(C) = \sum_{c \in C} \sum_{x \in X} f(x,c) * u(x,c), \tag{2.5}$$

gdzie $f(x,c)$ jest wybraną funkcją odległości, a $u(x,c)$ określa stopień przynależności przykładu x do klastra c. Dla klastrów twardych $u(x,c) \in \{0,1\}$, dla rozmytych $u(x,c) \in [0,1]$.

Podstawową wadą funkcji postaci (2.5) jest to, że dla wielu funkcji odległości:

$$fitness(C_1) < fitness(C_2),$$

gdy

$$|C_1| > |C_2|,$$

gdzie C_1,C_2 są dwoma proponowanymi przez algorytm grupowaniami zbioru danych.

W krańcowym przypadku łatwo jest zauważyć, że grupowanie, w którym każdy przykład jest w odrębnym, jednoelementowym klastrze, jest oceniane najlepiej.

Dlatego w algorytmach, które nie mają ustalonej *a priori* liczby klastrów, czynnik ten trzeba włączyć w funkcję oceny grupowania. Jedną z takich funkcji uwzględniających liczbę klastrów jest **Bayesowskie Kryterium Informacji** (*Bayesian Information Criterion*, BIC) [21] [8]:

$$BIC(C) = -2 * fitness(C) + v(C) * \log N,$$

gdzie $v(C)$ jest liczbą parametrów modelu C, a N to wielkość zbioru danych.

Funkcja ta porównuje zysk, jaki uzyskujemy ze zwiększonej liczby klastrów, z wiążącym się z tym wzrostem skomplikowania opisu modelu.

2.7 Algorytmy grupowania

Używane algorytmy grupowania można podzielić na dwie grupy: klasyczne, zwykle zachłanne heurystyki, i metody optymalizacji globalnej, przystosowane w pewien sposób do zagadnienia grupowania. Ponieważ w mojej pracy stosowane są jedynie metody zachłanne, ograniczę się tutaj do omówienia tylko tego rodzaju podejść. W zadaniu grupowania zostały z powodzeniem użyte np. algorytmy genetyczne – odniesienia do ważniejszych prac podawał będę opisując poszczególne algorytmy zachłanne, które były inspiracją dla metod genetycznych.

2.7.1 EM

W tych metodach zakładamy, że każdy przykład pochodzi z pewnej mieszaniny prostych rozkładów o znanych postaciach, lecz nieznanych parametrach. Na początek omówię używane przeze mnie pojęcia z dziedziny rachunku prawdopodobieństwa i statystyki matematycznej. W tym rozdziale słów **klaster** i **rozkład** będę używał zamiennie, ponieważ w tym kontekście mają to samo znaczenie.

Podstawowe definicje

Definicje te podaję za [15]

Próbą losową prostą jest ciąg n zmiennych losowych $(X_1, X_2, \ldots, X_n)$ niezależnych, mających jednakowe rozkłady, takie jak rozkład zmiennej losowej X w populacji.

Statystyką (z próby) nazywamy zmienną losową Z_n będącą funkcją zmiennych $X_1, X_2, \ldots, X_n$ stanowiących próbę losową.

Estymatorem T_n parametru θ rozkładu populacji generalnej nazywamy statystykę z próby $T_n = t(X_1, X_2, \ldots, X_n)$, która służy do oszacowania wartości tego parametru.

Mówimy, że estymator T_n parametru θ jest **nieobciążony**, jeśli spełniona jest relacja: $E(T_n) = \theta$. W przeciwnym przypadku estymator T_n nazywamy **obciążonym**, a wyrażenie: $E(T_n) - \theta = b(T_n)$ nazywamy **obciążeniem estymatora**.

Funkcją wiarygodności próby nazywamy funkcję postaci:

$$L(x_1, x_2, \ldots, x_n \mid \theta) = \prod_{i=1}^{n} f(x_i \mid \theta), \tag{2.6}$$

gdzie f jest funkcją gęstości rozkładu X, a θ parametrami tego rozkładu.

Funkcji tej często używa się w postaci zlogarytmizowanej:

$$L_M(x_1, x_2, \ldots, x_n \mid \theta) = \sum_{i=1}^{n} \log(f(x_i \mid \theta)). \tag{2.7}$$

Estymacja parametrów

Do wyznaczania parametrów μ oraz Σ rozkładu normalnego (wzór 2.4) mając daną próbę losową X używa się ich estymatorów. W tej pracy posłużyłem się estymatorami największej wiarygodności. Wyznacza się je jako maksymalizujące zlogarytmizowaną funkcję największej wiarygodności L dla rozkładu $N(\mu, \Sigma)$ (jako że funkcja logarytmu jest rosnąca, ma maksimum w tym samym punkcie co funkcja największej wiarygodności). Takimi estymatorami są ([1]):

$$\hat{\mu} = \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{2.8}$$

oraz

$$\hat{\Sigma} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})'. \tag{2.9}$$

Modelowanie mieszanin

W modelowaniu mieszanin (*mixture-modeling*) zakłada się, że rozkład każdego elementu próby losowej jest „mieszaniną", czyli kombinacją pewnej liczby prostszych rozkładów. W tej pracy posługuję się mieszaninami, w których jako podstawowych rozkładów używam rozkładów Gaussa.

Zmienna losowa X pochodzi z mieszaniny Φ składającej się z K rozkładów, jeśli:

$$f(X \mid \Phi) = \sum_{i=1}^{K} p_k \phi(X \mid \theta_k), \tag{2.10}$$

gdzie p_k jest wagą rozkładu k, $\phi(X \mid \theta_k)$ jest k-tym rozkładem, a θ_k parametrami k-tego rozkładu.

Gdy k-ty rozkład jest rozkładem normalnym θ_k zawiera μ_k (średnia k-tego rozkładu) oraz Σ_k (macierz kowariancji k-tego rozkładu).

Wyznaczanie parametrów mieszanin

Ze względu na zbyt dużą liczbę zmiennych nie można ściśle wyznaczyć parametrów rozkładu, trzeba posłużyć się jakimś heurystycznym algorytmem. Najpopularniejszym, a przy tym prostym i dobrze znanym, jest algorytm GMM [18]. Jest on konkretyzacją znanego w statystyce matematycznej algorytmu EM (Expectation – Maximalization).

Algorytm EM jest uniwersalną metodą postępowania, polega ona na cyklicznym powtarzaniu dwóch kroków: przewidywaniu pewnych parametrów (krok E), a następnie wyliczeniu zmiennych maksymalizujących pewną funkcję celu (krok M). W każdym z kroków korzysta się z wielkości obliczonych w kroku poprzednim.

Pierwszym zastosowaniem algorytmu EM było zagadnienie przewidywania parametrów rozkładu dla danych, w których brakuje wartości dla niektórych atry-

butów w pewnych przykładach. Krok wstępny polega na założeniu jakiejś wartości dla każdej brakującej danej – np. średniej lub mediany (dla atrybutów nominalnych) odpowiedniego atrybutu. Korzystając z tych wartości, w kroku E estymuje się parametry rozkładu. Następnie w kroku M wcześniej przyjęte wartości brakujących atrybutów są zastępowane wartościami dającymi największą wartość funkcji wiarygodności. Funkcja wiarygodności zależy ściśle od parametrów rozkładu wyznaczonych w poprzednim kroku.

W GMM, kolejnej konkretyzacji algorytmu EM, zakłada się, że przykłady można opisać mieszaniną rozkładów normalnych. GMM w formie zaproponowanej w [12] zamieszczam jako algorytm 1. W algorytmie używam zlogarytmizowanej funkcji wiarygodności zdefiniowanej w 2.7.1 dla funkcji gęstości f określonej dla mieszanin (2.10).

Algorytm 1 GMM

1. Mając daną liczbę klastrów K, zainicjuj parametry mieszaniny: p_k^0, μ_k^0, oraz Σ_k^0 $(k = 1, \ldots, K)$. Ustal numer iteracji $j = 0$.

2. Krok E: mając dane parametry modelu, oblicz t_{ik}^j:

$$t_{ik}^j = \frac{p_k^j \phi(x_i \mid \mu_k^j, \Sigma_k^j)}{\sum_{l=1}^K p_l^j \phi(x_i \mid \mu_l^j, \Sigma_l^j)}$$

3. Krok M: mając dane t_{ik}^j, oblicz nowe parametry modelu dla $k = 1, \ldots, K$:

$$p_k^{j+1} = \frac{1}{N} \sum_{i=1}^K t_{ik}^j$$

$$\mu_k^{j+1} = \frac{\sum_{i=1}^K t_{ik}^j x_i}{N * p_k^{j+1}}$$

$$\Sigma_k^{j+1} = \frac{\sum_{i=1}^K t_{ik}^j (x_i - \mu_k^{j+1})(x_i - \mu_k^{j+1})^T}{N * p_k^{j+1}}$$

4. Jeśli $\left| L_M(\Phi^{j+1}) - L_M(\Phi^j) \right| \geq \varepsilon$, przejdź do kolejnej iteracji: $j = j + 1$ i skocz do 2. (ε to mała liczba dodatnia).

t_{ik}^j to stopień przynależności przykładu i do klastra k w iteracji j.

Implementacja algorytmu w tej formie może wiązać się z pewnymi nie ewi-

dentnymi problemami, widocznymi zwłaszcza w przypadkach granicznych. Załóżmy, że d jest wymiarem przestrzeni X, czyli że przykłady mają po d atrybutów. W pierwszych krokach opisywanego poniżej algorytmu HAC liczba klastrów porównywalna jest z liczbą przykładów. W takich przypadkach prawdopodobne jest, że do któregoś z klastrów zaliczonych będzie mniej niż d przykładów, czyli że parametry t_{ik}^{j} dla pozostałych $|X| - d$ przykładów będą albo bardzo bliskie, albo równe zeru. Wtedy rząd macierzy Σ takiego klastra będzie mniejszy niż d, a zatem jej wyznacznik, będący mianownikiem we wzorze 2.4 będzie równy 0. Niestety w literaturze nie spotkałem opisu postępowania w takiej sytuacji. Wydaje mi się, że najlepszym sposobem jest lokalna redukcja wymiarowości przestrzeni przykładów. Dla zbioru liniowo zależnych przykładów można wyznaczyć bazę i następnie przekonwertować przykłady do nowej przestrzeni korzystając z tej bazy. Wtedy przykłady zaliczane do tego klastra nie będą już liniowo od siebie zależne. Przy wyznaczaniu funkcji wiarygodności tego klastra dla jakiegoś przykładu x_i należy najpierw sprawdzić czy x_i należy do przestrzeni klastra, czyli czy x_i jest liniowo zależny od przykładów wchodzących w skład tego klastra. Jeżeli nie, wartością funkcji wiarygodności jest 0. Taki sposób postępowania z powodzeniem przetestowałem w mojej implementacji algorytmu GMM.

Kolejnymi przypadkami szczególnymi, wymagającymi osobnego traktowania, jest klaster złożony z jednego przykładu, oraz klaster, do którego nie należy żaden przykład.

Wszystkie opisane tu problemy komplikują implementację, utrudniają testowanie i zwiększają koszt obliczeniowy prostego na pozór algorytmu.

Algorytm GMM wymaga podania *a priori* liczby klastrów. Może być to dużym ograniczeniem w dziedzinach, w których nic nie wiadomo o danych. Oczywiście można powtarzać go dla różnej liczby parametrów, ale jest to niewygodne. GMM to algorytm zachłanny i czasem jest bardzo wrażliwy na inicjalizację parametrów. Warto więc wielokrotnie uruchamiać go dla tych samych parametrów wejściowych. By znaleźć zatem najlepszy podział danych, trzeba uruchomiać GMM z $K = 2, \ldots, N$, za każdym razem kilkakrotnie. Jest to kosztowne obliczeniowo, a poza tym niezbyt sensowne, ponieważ informacja o poprzednich

grupowaniach jest tracona za każdym uruchomieniem.

Ograniczenia te próbuje pokonać algorytm HAC (Hierarchical Agglomerative Clustering). Algorytm ten zaczyna działanie od dużej liczby klastrów, porównywalnej lub równej liczbie przykładów. W każdej iteracji tego algorytmu łączona jest para „najlepszych" klastrów. Funkcją oceny jest *maximum classification log-likelihood*. Zakładając, że $p_k\phi(x_k \mid \theta_k) \gg p_s\phi(x_k \mid \theta_s)$ dla przykładu x_k należącego do klastra C_k (czyli że wiarygodność przykładu w klastrze, do którego należy on, jest dużo większa, niż w innych klastrach), spełnione jest:

$$\begin{aligned} L_M(x_1, x_2, \ldots, x_n \mid \theta) &= \sum_{i=1}^{n} \log(f(x_i \mid \theta)) \\ &= \sum_{i=1}^{n} \sum_{k=1}^{K} \log(p_k\phi(x_i \mid \theta_k)) \\ &\simeq \sum_{k=1}^{K} \sum_{x_i \in c_k} \phi(x_i \mid \theta_k) \\ &= L_C(x_1, x_2, \ldots, x_n \mid \theta), \end{aligned} \tag{2.11}$$

czyli że funkcja L_c, *maximum classification log-likelihood*, dobrze aproksymuje L_M, a przy tym jest dużo prostsza do obliczenia, ponieważ jest to wartość funkcji wiarygodności dla przykładu i klastra, do którego ten przykład należy. Podczas działania algorytmu HAC, w każdym kroku łączona jest para klastrów, których połączenie powoduje najmniejszy spadek wartości funkcji *maximum classification log-likelihood*. HAC jest algorytmem zachłannym i zwraca suboptymalne wyniki, ale używany jest stosunkowo często z uwagi na łatwość implementacji i niezłą jakość wyników. Główną wadą algorytmu jest kwadratowa złożoność obliczeniowa i pamięciowa, ponieważ działanie rozpoczyna się od klastrów jednoelementowych.

Alternatywą dla zachłannego stosowania HAC jest algorytm genetyczny opisany w [12]. W tej pracy autorzy proponują wykorzystanie hybrydowego algorytmu genetycznego, łączącego lokalną optymalizację (hill climbing) przeprowadzaną przez GMM z algorytmem genetycznym. HAC używany jest jako jeden z operatorów krzyżowania.

2.7.2 k-środki

Algorytm k-środków minimalizuje funkcję błędu średniokwadratowego, będącą konkretyzacją 2.5:

$$\sum_{i=1}^{M} H(C_i) \tag{2.12}$$

gdzie

$$H(Y) = \sum_{x \in Y} dist(x, \mu(Y))$$

jest błędem klastra Y (*dist* to miara odległości, np. miara Euklidesowa), a

$$\mu(Y) = \frac{1}{|Y|} \sum_{x \in Y} x$$

jest środkiem klastra Y (zakładam użycie miary euklidesowej).

Funkcja (3.1) nie bierze pod uwagę ilości klastrów, zatem jej minimalizacja prowadzi do trywialnego grupowania, w którym każdy punkt danych jest w swoim własnym klastrze. Dlatego konieczne jest zdefiniowane ograniczeń, które nie pozwolą na uzyskanie takich rezultatów. Algorytmy typu k-środków jako tego ograniczenia używają liczby klastrów.

Klasyczny algorytm k-środków[17] podany jest jako algorytm 2.

Algorytm 2 k-środki

1. Losowo zainicjuj k środków klastrów wewnątrz przestrzeni określonej przez przykłady.
2. Przypisz każdemu przykładowi najbliższy środek klastra.
3. Przelicz środki klastrów używając aktualnego przypisania przykładów do klastrów (przy korzystaniu z miary euklidesowej środek klastra to średnia z przykładów).
4. Jeśli kryterium zbieżności nie zostało spełnione, wróć do kroku 2..

Używanymi kryteriami zbieżności są: brak zmian w przypisaniu punktów do klastrów lub spadek błędu średniokwadratowego mniejszy niż ε.

Algorytm ten ma wiele istotnych wad. Przede wszystkim jest zachłanny i nie przegląda efektywnie przestrzeni rozwiązań. W typowych zastosowaniach algorytm musi być uruchamiany wielokrotnie z różnymi punktami początkowymi. Poza tym wymaga on podania a priori liczby klastrów k, tymczasem w wielu

zastosowaniach jest to wartość, której oczekujemy na wyjściu algorytmu. Trudno jest również porównywać modele uzyskane przy różnej liczbie klastrów, jako że błąd średniokwadratowy J_l modelu z l klastrami jest prawie zawsze wyższy niż błąd J_{l-1} modelu z $l-1$ klastrami. Można oczywiście używać kryterium informacji Bayesa (BIC) [21], ale w wielu przypadkach nie prowadzi to do najlepszych rezultatów [14, 21].

Istnieje wiele prac, w których model k-środków optymalizowany jest przy użyciu algorytmów genetycznych. W pracach tych obecne są dwa podejścia do reprezentacji grupowania jako chromosomu: albo kodowane są środki klastrów [10], a przynależność przykładów do klastrów określana jest w trakcie obliczania funkcji celu, albo kodowana jest przynależność punktu do klastra [16], a środki klastrów obliczane są w trakcie obliczania funkcji celu (w [16] jest to połączone z zachłanną optymalizacją algorytmem k-środków).

Podstawową zaletą algorytmu k-środków jest szybkość działania. Algorytm ten jest podstawą dla algorytmów grupowania dużych zbiorów danych, np. w [6] opisywany jest algorytm grupowania oparty na k-środkach i działający na danych nie mieszczących się w pamięci RAM. K-środki mają małe wymagania pamięciowe, ponieważ przechowują tylko listę przykładów, przypisań przykład-klaster i środków klastrów.

2.7.3 ISODATA

ISODATA [2] [11] to algorytm nawiązujący do algorytmu k-środków, natomiast wykazujący większą tolerancję na inicjalizację. W czasie działania algorytmu, w zależności od uzyskanych wyników, klaster może być podzielony, gdy jego wariancja przekracza pewną zadaną wartość. Klaster może być również połączony z sąsiednim klastrem, gdy odległość pomiędzy środkami tych klastrów jest mniejsza niż zadana parametrem. Problemem jest właściwy dobór tych parametrów. Algorytm w uproszczonej postaci podaję jako algorytm 3.

Algorytm 3 Algorytm ISODATA

Parametry wejściowe:

- k ilość klastrów jakie chcemy otrzymać;
- n_0 najmniejsza ilość przykładów w klastrze;
- n_{max} największa liczba przykładów w klastrze;
- σ_0 progowe odchylenie standardowe w kroku dzielenia klastrów;
- d_0 progowa odległość między środkami klastrów w kroku scalania.

Algorytm kończy się po zadanej, maksymalnej liczbie iteracji.

1. Losowo zainicjuj k środków klastrów.
2. Przypisz każdemu przykładowi najbliższy środek klastra.
3. Usuń wszystkie klastry, które mają mniej przykładów niż n_0. Przykłady należące do usuniętych klastrów potraktuj jak w 2.
4. Przelicz środki klastrów używając aktualnego przypisania przykładów do klastrów (przy korzystaniu z miary euklidesowej środek klastra to średnia z przykładów).
5. Ustal, czy dzielić, czy scalać klastry (C – aktualna liczba klastrów):
   (a) jeśli $C \leq \frac{k+1}{2}$ lub $\frac{k+1}{2} < C < 2k$ i numer iteracji jest nieparzysty, spróbuj podzielić klastry, przejdź do 6,
   (b) jeśli $C \geq 2k$ lub $\frac{k+1}{2} < C < 2k$ i numer iteracji jest parzysty, spróbuj scalić klastry, przejdź do 9.
6. Oblicz wartości odchylenia standardowego $\sigma_{i,j}$ dla każdej cechy j w każdym klastrze i. Zapamiętaj maksymalną wartość σ_i^* w każdym klastrze.
7. Dla każdego klastra i: jeśli $\sigma_i^* \leq \sigma_0$ nie dziel, w przeciwnym przypadku: jeśli $C \leq \frac{k+1}{2}$ lub $d_j > D$ i $N_j > 2n_0$ podziel wzdłuż j-tej współrzędnej, gdzie $\sigma_i^* = \sigma_{i,j}$.
8. Przelicz środki klastrów i przejdź do 2.
9. Oblicz odległości $d_{i,j}$ pomiędzy środkami wszystkich klastrów. Uporządkuj je w porządku rosnącym. Niech $k^* = \min(k, n_{max})$.
10. Wykonaj k^* razy: weź kolejny $d_{i,j}$ i jeśli $d_{i,j} < d_0$ oraz żaden z klastrów nie był scalony z jakimkolwiek innym, scal te dwa klastry.
11. Przelicz środki klastrów i przejdź do 2.

2.7.4 CLUSTER/2

CLUSTER/2 [7] jest zachłannym algorytmem grupowania korzystającym z pojęcia kompleksów. W każdym kroku algorytmu spośród wszystkich N przykładów wybiera się k ziaren. Następnie wyznaczanych jest k kompleksów opisujących poszczególne grupy tak, by każdy kompleks pokrywał dokładnie jedno ziarno. Kompleksy te są następnie oceniane heurystyczną funkcją oceny i jeżeli jej wynik jest lepszy niż dotychczas znaleziony, zbiór optymalnych kompleksów K zastępowany jest przez zbiór kompleksów wyznaczony w aktualnym kroku. Kroki powtarzane są, aż nie zajdzie pewne kryterium końca, np. brak poprawy optymalnej wartości funkcji celu w ciągu ostatnich kilku iteracji.

W pierwszej iteracji ziarna generowane są losowo, przy czym korzystne jest, aby były od siebie maksymalnie odległe, czyli by wartości poszczególnych atrybutów w miarę możliwości różniły się od siebie. W kolejnych iteracjach na ziarna wybierane są przykłady należące do kompleksów wygenerowanych w poprzedniej iteracji (przy czym obowiązuje zasada, że jeden kompleks generuje jedno ziarno).

Funkcja oceny grupowania powinna uwzględniać wiele czynników, m.in. stopień pokrycia zbioru trenującego X i złożoność opisu kategorii – im prostszy opis (im więcej selektorów uniwersalnych), tym lepiej.

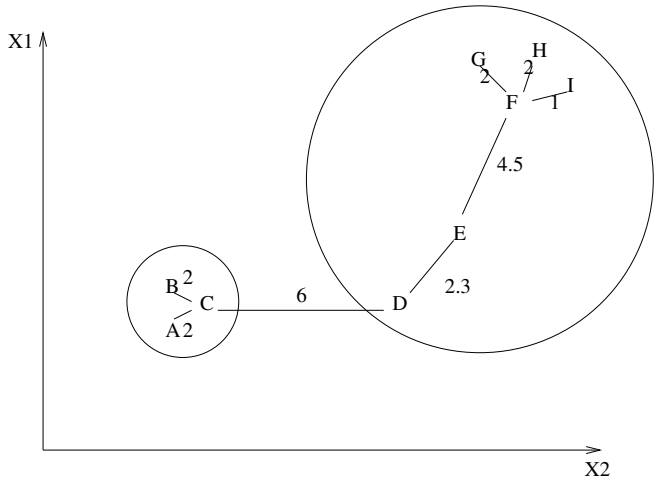
2.8 Algorytmy grafowe

Zbiór danych można przedstawić jako graf $G = V,E$, gdzie zbiór wierzchołków jest tożsamy ze zbiorem przykładów $V \equiv X$. Każde dwa przykłady łączy krawędź o wadze równej odległości między tymi przykładami:

$$E = e = (x_1, x_2) : x_1 \neq x_2$$

$$\forall e = (x_1, x_2) \ni E : w(e) = dist(x_1, x_2)$$

gdzie *dist* jest dowolną funkcją, której wartości reprezentują miarę odległości (podobieństwa) między dwoma przykładami. W zależności od reprezentacji przykła-



Rysunek 2.7: Przykładowy rezultat grupowania za pomocą algorytmu grafowego. Graf przedstawiony na rysunku to MST dla zbioru danych.

dów, a w szczególności od zbioru wartości ich artybutów, można wybrać dowolną z funkcji opisywanych w 2.5.1 lub 2.5.3.

Znanych jest kilka algorytmów działających na tak zdefiniowanym grafie, m.in. *single-link*, *complete-link*, czy opisany poniżej algorytm minimalnego drzewa rozpinającego.

2.8.1 Algorytm minimalnego drzewa rozpinającego (MST)

W pierwszym kroku algorytmu wyznaczany jest graf będący minimalnym drzewem rozpinającym (*minimal spanning tree*, MST) zbioru przykładów. Następnie porządkuje się malejąco pod względem wag krawędzie tego grafu. Usuwając kolejne krawędzie otrzymujemy spójne części tego grafu, będące kolejnymi hierarchicznymi grupowaniami zbioru danych. Przykład grupowania za pomocą tego algorytmu przedstawiony jest na rysunku 2.7. Graf na rysunku to MST dla zbioru danych. Po usunięciu krawędzi o największym koszcie ($C \leftrightarrow D$) uzyskujemy podział danych na klastry: $\{A,B,C\}$ i $\{D,E,F,G,H,I\}$. Kolejną krawędzią pod względem kosztu jest $E \leftrightarrow F$, po usunięciu której otrzymujemy 3 klastry: $\{A,B,C\}$, $\{D,E\}$ i $\{F,G,H,I\}$.

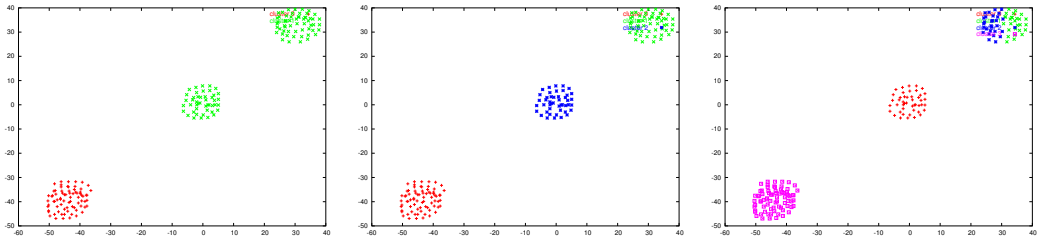
2.9 Abstrakcja rezultatów

Zrozumiały dla użytkownika opis klastrów może być kluczowy w interpretacji wyników działania algorytmu grupowania. Często algorytm narzuca sposób symbolicznego opisu klastrów (np. COBWEB, CLUSTER/2, czy algorytm EM), który może okazać się wygodny i łatwy do interpretacji dla osoby znającej dziedzinę problemu. Jeśli natomiast opisem takim nie dysponuje się, w [13] proponowane są następujące, uniwersalne opisy wiedzy, jaką uzyskaliśmy w wyniku grupowania:

- Opis klastra przez punkt będący jego środkiem, lub zbiór punktów najbardziej od środka oddalonych.
- Opis grupowania poprzez węzły w drzewie klasyfikacji.
- Opis grupowania poprzez koniunkcyjne wyrażenia logiczne, nakładające warunki na wybrane atrybuty (np. $[wiek > 20][waga < 70]$ może oznaczać grupę ludzi w wieku powyżej 20 lat i ważących mniej niż 70 kg).

2.10 Analiza tendencji grupowania

Na rysunku 2.8 b) przedstawiony jest przykładowy rezultat grupowania algorytmem k-środków. Oglądając ten rysunek można stwierdzić, że grupowanie było udane – uformowane zostały trzy, dobrze oddzielone od siebie i wizualnie „pasujące" (*visually attractve*) do danych klastry. Rysunki 2.8 a) i 2.8 c) przedstawiają rezultaty grupowania przy zadaniu algorytmowi odpowiednio dwóch i czterech klastrów. Można stwierdzić, że wyniki te są gorsze, ponieważ trzy wyraźnie istniejące w danych klastry zostały w dość przypadkowy sposób podzielone. Problem zaczyna się, gdy dane mają nie dwa lub trzy, ale cztery i więcej wymiarów. Wtedy, przy niemożliwości pełnego przedstawienia graficznego takiego zbioru danych, człowiekowi trudno jest zauważyć klastry nawet tak dobrze oddzielone, jak na rysunku 2.8 b), a co za tym idzie, właściwie ocenić rezultat grupowania. Konieczne są zatem pewne miary będące w stanie ocenić i porównać wyniki działania algorytmu.



a) dwa klastry b) trzy klastry c) cztery klastry

Rysunek 2.8: Kilka możliwych grupowań tego samego zbioru danych w zależności od liczby klastrów.

2.10.1 Terminologia

W pełnym procesie odkrywczej analizy danych (*exploratory data analysis*), zaproponowanym w [4], wyróżnione są trzy kroki:

1. Ocena możliwości grupowania (*assessment of clustering tendency*), krok, w którym to próbuje się ocenić, czy w danych są jakieś klastry bez przeprowadzania grupowania.

2. Grupowanie, czyli wybór reprezentacji przykładów, klastrów i algorytmu grupowania, a następnie wykonanie tego algorytmu.

3. Walidacja klastrów (*cluster validity*), kiedy to ocenia się proponowane przez algorytm grupowanie.

W literaturze istnieje pewna niekonsekwencja w nazywaniu pierwszego i trzeciego z podanych kroków. W pracy [22] przeprowadzone analizy nazywane są *cluster tendency assessment*, mimo iż wymagają do swojego działania rezultatu grupowania, czyli według [4] jest to raczej badanie *cluster validity*. W mojej pracy (podobnie jak w [4] i w [22]) nie będę zajmował się krokiem pierwszym, a z powodu intensywnego wykorzystania pracy [22] przyjmę proponowane w niej nazewnictwo, nawet przy referowaniu [4].

2.10.2 Badanie tendencji

Algorytmy używane do grupowania mają zazwyczaj wiele parametrów, wśród których można wyznaczyć parametr kluczowy, od którego wynik zależy w największym stopniu. Bardzo często parametrem takim jest liczba klastrów (np. w algorytmie k-środków, czy algorytmie EM). Pozostałe parametry mają zazwyczaj znaczenie drugorzędne, kontrolując np. sposób przeglądania przestrzeni rozwiązań. **Analizą tendencji grupowania** (*cluster tendency assessment*) można nazwać badanie działania algorytmu w zależności od wartości kluczowego parametru, przy ustalonych wartościach innych parametrów.

Na podstawie takiej analizy można dużo lepiej zrozumieć zależności w zbiorze danych, który badamy. Możliwe jest również wyznaczyć najwłaściwszą wartość kluczowego parametru. Wreszcie, gdy chcemy zbadać nie dane, a algorytm, analiza daje nam możliwość dokładnego jego przetestowania dla wszystkich istotnych wartości tego parametru.

2.10.3 Funkcje oceny grupowania

By przeprowadzić analizę tendencji, trzeba umieć porównać wyniki działania algorytmu dla różnych parametrów. Najprościej jest to zrobić wybierając pewną **funkcję oceny grupowania** (*cluster validity index*) [4], która danemu grupowaniu przypisuje pewną ocenę – liczbę rzeczywistą. Najlepiej jest, gdy funkcja ta ma globalne ekstremum dla najlepszej wartości parametru.

W tym rozdziale przedstawię trzy takie funkcje, opisane w [4].

Zmodyfikowana statystyka Huberta.Statystyka Γ Huberta mierzy dopasowanie pomiędzy danymi a grupowaniem wyrażoną przez macierz przynależności U. Niech macierz $P = [p_{ij}]$ wyraża obserwowalne podobieństwo między przykładami. p_{ij} jest podobieństwem między przykładami x_i i x_j, jego wartość jest równa odległości (w sensie dowolnej miary odległości z rozdziału 2.5.1, byle posiadającej własność symetrii) między tymi dwoma przykładami. Macierz $Q = [q_{ij}]$ jest

macierzą zdefiniowaną jako

$$q_{ij} = \begin{cases} 0 & \text{gdy } \exists k : u_{ik} = u_{kj}, \\ & \text{czyli } x_i \text{ i } x_j \text{ należą do tego samego klastra,} \\ 1 & \text{w przeciwnym przypadku.} \end{cases}$$

Statystyką Huberta jest korelacja (*point serial correlation coefficient*) między dwoma dowolnymi macierzami. Ponieważ macierze Q i P są symetryczne (bo odległość jest symetryczna, $p_{ij} = p_{ji}$), statystykę Γ można wyrazić poprzez

$$\Gamma(P, Q(U)) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} p_{ij} q_{ij} \tag{2.13}$$

W formie znormalizowanej [4] wartości tej statystyki zawierają się pomiędzy $-1 \leq \hat{\Gamma} \leq 1$. Statystyka ta mierzy stopień liniowej zależności między elementami P i Q, wyraża stopień wzajemnej korelacji tych macierzy.

Wartość statystyki $\hat{\Gamma}$ lub Γ mówi o sile zależności, natomiast by zbadać, czy ta zależność jest nadzwyczajnie duża (w porównaniu do innych, losowych grupowań), należałoby zbadać rozkład tych statystyk. Takie badanie wiąże się z obliczeniem jej wartości dla wszystkich możliwych grupowań, co jest w praktycznych zastosowaniach niewykonalne.

Z tych powodów wprowadzono **zmodyfikowaną statystykę Huberta** (MH). Niech $L(i) = k$ jeśli i-ty przykład należy do k-tego klastra $x_i \in C_k$, a $\|v_i - v_j\|_2$ będzie euklidesową odległością między środkami klastrów C_i i C_j. Macierz $Q(U, V(U)) = [q_{(U,V(U)),i,j}]$ ($V(U)$ to macierz zawierająca środki klastrów) można określić jako

$$q_{(U,V(U)),i,j} = [\|v_{L(i)} - v_{L(j)}\|_2]. \tag{2.14}$$

Taką postać macierzy Q używa się w wyrażeniu 2.13, uzyskując MH albo znormalizowaną MH. Z badań eksperymentalnych wiadomo, że statystyki te zwiększają swoją wartość wraz ze wzrostem liczby klastrów, dlatego też w celu znalezienia optymalnej wartości parametru, szuka się dużych zmian w wartości MH dla sąsiedniej liczby klastrów.

Davies-Bouldin Index: Dobre klastry mają małe rozproszenie wewnętrzne i są od siebie daleko. **Miara Davies'a-Bouldin'a (DB)** próbuje złączyć te dwa kryteria oceny.

Rozproszenie przykładów wewnątrz i-tego klastra można wyrazić przez

$$S_{i,q} = (\frac{1}{|X_i|} \sum_{x \in X_i} \|x - v_i\|_2^q)^{\frac{1}{q}}, \tag{2.15}$$

co jest pierwiastkiem q-tego stopnia z q-tego momentu przykładów w klastrze. $S_{i,1}$ to średnia odległość przykładu od klastra, $S_{i,2}$ to pierwiastek błędu średniokwadratowego klastra (środek klastra możemy wtedy interpretować jako np. wartość mierzoną, a przykłady do niego należące jako pomiary).

Jednym ze sposobów na mierzenie odległości pomiędzy klastrami C_i i C_j jest pomiar odległości między ich środkami

$$d_{ij,t} = \sum_{s=1}^{d} \|v_{si} - v_{sj}\|^{\frac{1}{t}} = \|v_i - v_j\|, \tag{2.16}$$

co jest odległością Minkowskiego rzędu t.

Dla i-tego klastra można znaleźć inny klaster, który jest w pewnym sensie najbardziej do niego podobny – leży blisko niego lub ma duże rozproszenie:

$$R_{i,qt}(U,V) = \max_{j, j \neq i} \frac{S_{i,q}(U) + S_{j,q}(U)}{d_{ij,t}(U)}. \tag{2.17}$$

Miara Davies'a-Bouldin'a definiowana jest jako

$$\nu_{DB,qt}(U,V) = \frac{1}{c} \sum_{i=1}^{c} R_{i,qt}(U,V). \tag{2.18}$$

Ponieważ chcemy dostawać klastry o możliwie małym rozproszeniu wewnętrznym i leżące możliwie daleko od siebie, szukamy parametrów, które *minimalizują* $\nu_{DB,qt}$.

Miara Dunn'a (DI)*Dunn's Index* jest podobna do miary Davies'a-Bouldin'a. Niech S i T będą niepustymi podzbiorami $\Re^d$, a $d : \Re^d \times \Re^d \to \Re^+$ będzie dowolną miarą. *Średnicę* Δ (diameter) zbioru S można zdefiniować jako

$$\Delta(S) = \max_{x,y \in S} d(x,y), \tag{2.19}$$

czyli największą odległość pomiędzy elementami tego zbioru. *Odległość* δ między zbiorami S i T można wyrazić poprzez

$$\delta(S,T) = \min_{x \in S, y \in T} d(x,y), \tag{2.20}$$

czyli najmniejszą odległość pomiędzy elementami z dwóch zbiorów.

Oczywiście tak zdefiniowane miary mają szereg wad, największą jest duża wrażliwość na pojedyncze punkty niepasujące do żadnego klastra, występujące zazwyczaj przy dużym zaszumieniu danych. Miary te *de facto* ignorują „centralizm" klastrów, skupiając się na najgorszych przykładach. W [4] autorzy proponują inne miary, analogiczne do podanych powyżej, natomiast zachowujące się lepiej przy zaszumionych danych.

Miarą separacji grupowania U jest

$$\nu_D(U) = \min_{1\leq i\leq c} \min_{1\leq j\leq c, j\neq i} \frac{\delta(X_i, X_j)}{\max_{i\leq k\leq c} \Delta(X_k)}. \tag{2.21}$$

Globalne maksimum $\nu_D(U)$ odpowiada optymalnym z punktu DI parametrom grupowania. W tej postaci widać wyraźnie analogie pomiędzy DI a DB. Rolę czynnika karzącego za rozproszenie klastrów spełnia w DI średnica klastra Δ, zaś δ nagradza za duże odległości pomiędzy klastrami.

Dunn zdefiniował dwa pojęcia charakteryzujące grupowanie. Przy danej mierze d grupowanie jest **zwarte i separowalne** (*compact and separated*, CS) jeżeli spełnione jest:

$$\forall s, q, rq \neq r, \forall x, y \in C_s, u \in C_q, v \in C_r : d(x, y) < d(u, v), \tag{2.22}$$

czyli że odległość między dowolnymi dwoma przykładami z jednego klastra jest mniejsza niż odległość między dowolnymi dwoma przykładami z dwóch różnych klastrów. Dunn udowodnił, że zbiór danych X może zostać pogrupowany w taki sposób tylko wtedy, gdy $\max_U \nu_D(U) > 1$, czyli że DI dla najlepszego grupowania musi być większa niż 1.

2.10.4 Problemy z badaniem tendencji

Niestety badanie analizy tendencji jest zazwyczaj bardzo kosztowne obliczeniowo, wiąże się bowiem z wielokrotnym uruchamianiem algorytmu dla nieznacznie tylko zmienionego parametru. By właściwie ją zbadać, należy próbkować funkcję oceny grupowania – musimy wybrać więc wartość najmniejszą, najwięk-

szą oraz krok, o jaki będziemy zwiększać wartość parametru pomiędzy poszczególnymi uruchomieniami. W ten sposób dla każdego parametru oryginalnego algorytmu uzyskujemy aż 3 parametry analizy tendencji. Co gorsza wartości tych 3 parametrów zazwyczaj muszą być ustalone eksperymentalnie, w wyniku mniej lub bardziej udanych prób badania analizy tendencji, co zwiększa jeszcze bardziej koszt obliczeniowy tego badania.

2.11 Sformułowanie problemu

W mojej pracy pragnę zająć się badaniem algorytmów grupowania. Używając zaproponowanej w rozdziale 2.4 taksonomii, skupię się na krokach 2 i 5, a więc na wyborze reprezentacji klastrów i wyborze algorytmu przeglądania przestrzeni rozwiązań. W celu pełnego przetestowania algorytmów będę badał również tendencję grupowania. Używał będę zbiorów danych z przykładami mającymi wszystkie atrybuty ciągłe. Danych nie poddam żadnemu procesowi wstępnego obrabiania czy wybierania właściwej reprezentacji. Nie będę również zajmował się zagadnieniem abstrakcji rezultatów.

Rozdział 3

Algorytm grupowania maksymalnej wariancji

W opisanym jako algorytm 2 algorytmie k-środków klastry reprezentowane były przez wektory środków, a algorytm minimalizował błąd średniokwadratowy:

$$\sum_{i=1}^{M} H(C_i), \tag{3.1}$$

gdzie

$$H(Y) = \sum_{x \in Y} dist(x, \mu(Y))$$

jest błędem klastra Y (*dist* to miara odległości, np. odległość euklidesowa), a

$$\mu(Y) = \frac{1}{|Y|} \sum_{x \in Y} x$$

jest środkiem klastra.

Ponieważ wynikiem minimalizacji bez ograniczeń funkcji postaci 3.1 jest podział zbioru danych X na klastry jednoelementowe, potrzebne są jakieś ograniczenia. W algorytmie k-środków, jak i w wielu innych algorytmach tego typu [13, 10] tym ograniczeniem jest ustalona liczba klastrów M. Takie podejście ma szereg wad opisanych w rozdziale 2.7.2, tu przypomnieć należy trudności z ustaleniem wartości tego parametru oraz z porównywaniem wyników działania algorytmu dla różnych wartości.

3.1 Model przestrzeni

Autorzy [22] proponują zupełnie inne podejście do tego problemu. Zamiast ustalać liczbę klastrów M, postulują oni, by wariancja sumy dowolnych dwóch klastrów była wyższa niż pewien ustalony próg σ^2_{max}

$$\forall C_i, C_j, i \neq j : Var(C_i \cup C_j) \geq \sigma^2_{max}, \tag{3.2}$$

gdzie wariancja klastra wyrażona jest przez:

$$Var(Y) = \frac{H(Y)}{|Y|}.$$

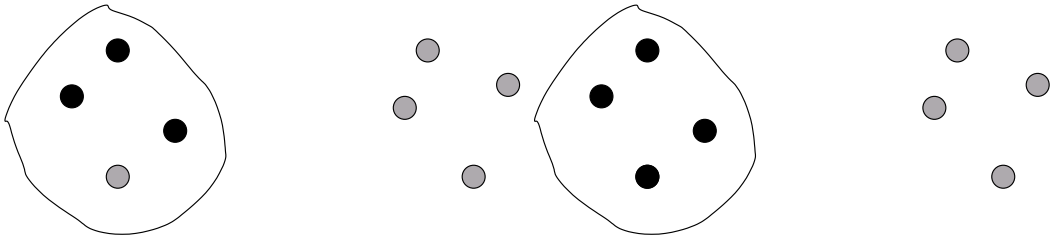
Wariancja klastrów będących rezultatem minimalizacji przy takim ograniczeniu zazwyczaj jest mniejsza niż σ^2_{max}. Jeżeli dla jakiegoś klastra C_a: $Var(C_a) \geq \sigma^2_{max}$, to można udowodnić [23], że dla każdego punktu tego klastra leżącego w odległości większej niż σ^2_{max} od jego środka, stworzenie nowego klastra zawierającego tylko ten punkt spowodowałoby pogwałcenie ograniczenia 3.2. Ten nowy klaster mógłby złączyć się z jednym z klastrów sąsiadujących z C_a:

$$Var(C_a) \geq \sigma^2_{max} \Rightarrow \forall x \in C_a : (\|x - \mu(C_a)\|^2 < \sigma^2_{max} \vee \exists C_b : Var(C_b \cup x) < \sigma^2_{max}).$$

Oczywiście nie znaczy to, że ograniczenie postaci 3.2 można zastąpić ograniczeniem na wariancję każdego klastra $\forall C_i : Var(C_i) \leq \sigma^2_{max}$. Takie ograniczenie poprowadziłoby do rozwiązania z jednym przykładem w każdym klastrze.

3.2 Algorytm grupowania maksymalnej wariancji

W pracy [22] autorzy proponują algorytm przeszukujący przestrzeń rozwiązań przedstawioną w rozdziale 3.1. **Algorytm grupowania maksymalnej wariancji** (*Maximum Variance Cluster Algorithm*, MVC) przeprowadza stochastyczną minimalizację odległości przykładów od środków klastrów (więc również i błąd średniokwadratowego określonego równaniem 3.1) przy ograniczeniu na wariancję sumy klastrów zdefiniowanym równaniem 3.2.



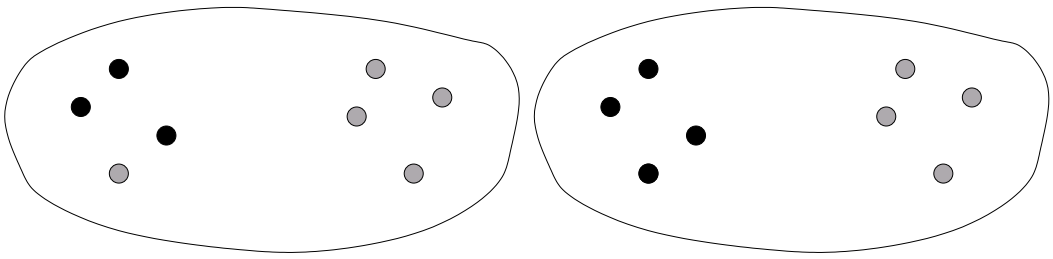
a) 1 kandydat do rozszerzenia b) brak kandydatów do rozszerzenia

Rysunek 3.1: Zbiór przykładów (zaznaczony obwiednią) przeglądanych w celu znalezienia przykładów do rozszerzenia klastra (zawierającego wszystkie czarne punkty) dla list zawierających 2 przykłady. Na rysunku a) klaster ma 3 elementy i 1 kandydata do rozszerzenia. Na rysunku b) klaster ma 4 elementy, a zbiór kandydatów na rozszerzenie klastra jest pusty.

Intuicyjnie można stwierdzić, że przykłady z innych klastrów leżące „blisko" przykładów należących do klastra mogą być kandydatami na powiększenie tego klastra. Na tej samej zasadzie można spróbować usunąć z klastra przykłady „odległe", to znaczy takie, które należą do tego samego klastra i leżą daleko od innych przykładów z klastra. Należy jeszcze znaleźć sposób znajdowania przykładów o takich własnościach.

Rozważmy na początek problem znajdowania kandydatów do rozszerzenia klastra. Najprostszym sposobem jest stworzenie dla każdego przykładu listy k najbliższych (w sensie używanej miary odległości) przykładów i przeglądanie jej w celu znalezienia kandydatów. Niestety trudno wyznaczyć wtedy właściwą wartość k, która staje się de facto kolejnym parametrem modelu. Jeżeli k jest zbyt małe, wszystkie przykłady z listy należą do tego samego klastra, w związku z tym zbiór kandydatów jest pusty (rysunek 3.1). Jeżeli jest zbyt duże, lista zawiera dużą część zbioru danych, należą do niej kandydaci obiektywnie zarówno „dobrzy" jak i „źli" (rysunek 3.2). W ten sposób tracimy cały zysk „lokalności" przeszukiwań i przeglądamy praktycznie cały zbiór danych.

W [22] autorzy proponują inne podejście do tego problemu. Dla danego przykładu x należącego do klastra C_a **zewnętrzna granica k-tego rzędu** $F(x,k,C_a,X)$



a) klaster 3 elementowy b) klaster 4 elementowy

Rysunek 3.2: Zbiór przykładów (zaznaczony obwiednią) przeglądanych w celu znalezienia przykładów do rozszerzenia klastra (zawierającego wszystkie czarne punkty) dla list zawierających 7 przykładów. Chociaż na rysunku a) klaster ma 3 elementy, a na rysunku b) klaster ma 4 elementy, zbiór kandydatów na rozszerzenie klastra jest taki sam i zawiera wszystkie przykłady ze zbioru.

to zbiór k najbliższych przykładów z innych klastrów:

$$F(x,k,C_a,Y) = \begin{cases} nf(x,C_a,Y) \cup F(x,k-1,C_a,Y-nf(x,C_a,Y)) \text{ jeżeli } k>0, \\ \emptyset \text{ jeżeli } k=0, \end{cases} \tag{3.3}$$

gdzie $nf(x,C_a,Y)$ jest przykładem leżącym w najmniejszej odległości od x i nie należącym do klastra C_a:

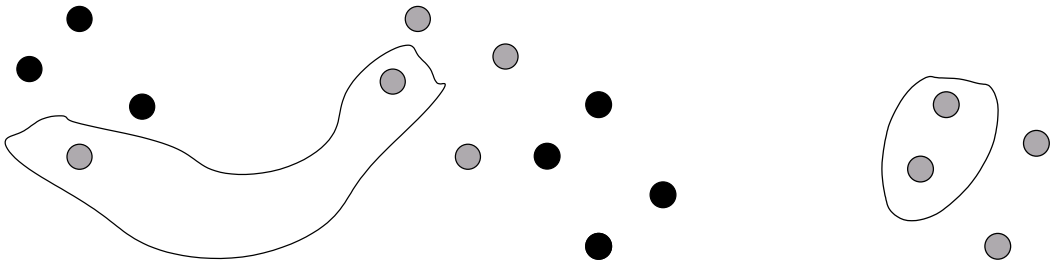
$$nf(x,C_a,Y) = \arg\min_{y \in Y-C_a} \|y-x\|^2. \tag{3.4}$$

Używając pojęcia zewnętrznej granicy dla przykładu można zdefiniować **zewnętrzną granicę k-tego rzędu B_a klastra C_a** jako sumę zewnętrznych granic przykładów należących do tego klastra:

$$B_a(k) = \bigcup_{x \in C_a} F(x,k,C_a,X). \tag{3.5}$$

B_a to naturalny zbiór kandydatów do powiększenia klastra. Taka granica ma podstawową zaletę polegającą na tym, że rośnie wraz ze wzrostem klastra. Zawsze więc algorytm będzie miał przykłady, które będzie próbował włączyć do klastra (oczywiście oprócz zdegenerowanego przypadku, gdy klaster zawiera wszystkie punkty danych).

Zupełnie analogicznie definiuje się **wewnętrzną granicę q-tego rzędu** $G(x,q,Y)$ dla danego przykładu $x \in C_a$ jako zbiór najdalszych przykładów należących do



a) klaster 3 elementowy b) klaster 4 elementowy

Rysunek 3.3: Zewnętrzna granica będąca zbiorem kandydatów do rozszerzenia klastra (zawierającego wszystkie czarne punkty) Na rysunku a) klaster ma 3 elementy i 2 kandydatów do rozszerzenia. Na rysunku b) klaster ma 4 elementy, a granica została zmodyfikowana i znów zawiera 2 kandydatów.

klastra C_a ($Y = C_a$) :

$$G(x,k,Y) = \begin{cases} fn(x,Y) \cup G(x,k-1,Y - fn(x,Y)) \text{ jeżeli } k > 0, \\ \emptyset \text{ jeżeli } k = 0, \end{cases} \tag{3.6}$$

gdzie $fn(x,Y)$ jest przykładem leżącym w największej odległości od x i należącym do klastra C_a ($Y = C_a$):

$$fn(x,Y) = \arg\max_{y \in Y} \|y - x\|^2 \tag{3.7}$$

Wewnętrzna granica klastra q-tego rzędu $I_a(q)$ jest sumą wewnętrznych granic przykładów należących do tego klastra:

$$I_a(k) = \bigcup_{x \in C_a} G(x,k,C_a) \tag{3.8}$$

Taka granica stanowi naturalny zbiór przykładów, które być może warto usunąć z klastra C_a.

By wydajnie wyznaczać zewnętrzne i wewnętrzne granice w trakcie kolejnych iteracji, algorytm zaczyna swoje działanie od przygotowania dla każdego przykładu x uporządkowanej rosnąco pod względem odległości listy sąsiadów, zawierającej wszystkie inne przykłady ze zbioru danych. Mając do dyspozycji takie listy, obie granice można wyznaczyć w czasie liniowym $O(n)$.

Oczywiście listy takie stanowią dość znaczne obciążenie pamięci, ponieważ do grupowania n przykładów potrzebujemy n^2 pamięci. Konstrukcja tych list jest

również problemem obliczeniowym, ponieważ stworzenie jednej listy wymaga obliczenia odległości przykładu od wszystkich innych przykładów a następnie posortowania takiej nieuporządkowanej listy, co wiąże się z kosztem $O(n+n\log(n)) = O(n\log(n))$. Stworzenie n list to już $O(n^2\log(n))$, co dla niektórych zbiorów danych przekracza koszt całego algorytmu k-środków $O(nkl)$, gdzie k jest liczbą klastrów, a l to liczba iteracji.

Dla dużych zbiorów danych, kiedy opisane problemy są szczególnie istotne, w [22] proponuje się ograniczenie listy sąsiadów do tych, którzy leżą bliżej niż $d_{max} = 2sigma^2_{max}$, ponieważ dla „dużego" klastra o wariancji w przybliżeniu równej σ^2_{max}, większość przykładów leży bliżej niż σ^2_{max} od środka, czyli odległość pomiędzy dwoma przykładami jest zazwyczaj mniejsza niż $2sigma^2_{max}$. Dla klastrów o wariancjach większych niż σ^2_{max} algorytm również powinien działać, bo kandydaci do rozszerzenia, bądź zmniejszenia klastra, będą brani z list sąsiedztwa przykładów leżących „bliżej" nich, np. po tej samej stronie od środka klastra.

Takie ograniczenie nie jest skuteczne, gdy w zbiorze danych dopuszcza się dosyć dużą wariancję i szuka się możliwie dużych klastrów. Niech σ^{*2}_{max} oznacza wariancję największego możliwego klastra, czyli klastra zawierającego wszystkie przykłady. Wtedy już dla $\sigma^2_{max} = \frac{1}{2}\sigma^{*2}_{max}$ „ograniczone" listy sąsiedztwa zawierają wszystkie przykłady ze zbioru danych.

Pełen algorytm przedstawiam jako algorytm 4. Zaczyna on działanie od klastrów jednoelementowych. W każdej iteracji każdy z klastrów przechodzi serię testów. W zależności od ich wyników klaster jest modyfikowany poprzez scalanie z innym klastrem albo poprzez izolację najdalszego przykładu, albo przez przepisanie przykładu z sąsiedniego klastra.

Ze względu na możliwość losowego przepisania przykładu w kroku perturbacji (P_d), algorytm nie zbiega. W związku z tym po pewnej liczbie iteracji E_{max} zeruje się prawdopodobieństwo $P_d = 0$. Możliwe jest również, że w optymalnym rozwiązaniu występują klastry o wariancji większej niż σ^2_{max} (problem ten opisany jest szczegółowo w rozdziale 3.1). Z tego powodu po E_{max} iteracjach nie wykonuje się izolacji. Po takich modyfikacjach algorytm jest zbieżny. Algorytm kończy się, gdy po ustalonej liczbie iteracji nie było żadnych zmian w przypisaniu

Algorytm 4 Maximum Variance Cluster Algorithm

1. Zainicjuj zbiór jednoelementowych klastrów C:

$$\forall x \in X \exists C_x : x \in C_x$$

$$\forall C_x \in C : |C_x| = 1$$

2. Dla każdego klastra $C_a \in C$ (kolejność losowa lub wykonuj równolegle):

   (a) *izolacja*. Jeśli wariancja klastra C_a przekracza wartość progową σ^2_{max} (będącą parametrem algorytmu), to: weź losowy podzbiór i_a wewnętrznej granicy I_a klastra i stwórz nowy klaster z przykładu $x^* \in i_a$, który leży najdalej od środka klastra μ_a. Wielkość podzbioru $|i_a|$ jest proporcjonalna do wielkości granicy I_a;

   (b) *scalanie*. Jeżeli wariancja C_a jest poniżej maksymalnej dopuszczalnej wartości, to: oblicz wariancję sumy klastra C_a z każdym klastrem sąsiadującym. Jeżeli najniższa wartość pozostaje poniżej σ^2_{max}, scal klastry. Klastrem sąsiadującym jest każdy klaster, który zawiera przykład z zewnętrznej granicy klastra C_a;

   (c) *perturbacja*. Jeżeli C_a nie został zmodyfikowany w żadnym z powyższych kroków, spróbuj zmniejszyć wartość kryterium optymalizacyjnego (optymalizacja lokalna) poprzez próbę przepisania przykładów z sąsiadujących klastrów do C_a. W celu znalezienia tych przykładów weź losowy podzbiór b_a zewnętrznej granicy B_a (znów $|b_a|$ jest proporcjonalne do $|B_a|$). Dla każdego przykładu $x \in b_a$, $x \in C_b$ oblicz zysk z przepisania tego przykładu z klastra C_b do klastra C_a dany wzorem:

$$G_{a,b} = H(C_a) + H(C_b) - H(C_a \cup x) - H(C_b - x).$$

   Jeżeli maksymalny zysk $G^*_{a,b}$ dla przykładu x^* jest dodatni, przepisz x^* z C_b do C_a. W przeciwnym wypadku x^* może być przepisany z pewnym małym prawdopodobieństwem P_d (occasional defect probability) bez względu na wartość $G^*_{a,b}$.

3. Jeśli nie zaszedł warunek końca, przejdź do 2.

przykładów do klastrów (czeka się kilka iteracji, a nie jedną, bo podzbiory granic w kroku perturbacji są nadal wyznaczane stochastycznie).

3.3 Badanie tendencji klastrów przy użyciu MVC

Kluczowym parametrem dla działania algorytmu MVC jest maksymalna wariancja σ^2_{max} [22]. Jeżeli rozwiązanie proponowane dla pewnej wartości maksymalnej wariancji σ^2_1 na prawdę dobrze dzieli zbiór danych na klastry, to powinniśmy dostać ten sam wynik nawet po zwiększeniu wartości wariancji. W algorytmie MVC parametr wariancji steruje scalaniem klastrów, więc jeśli proponowane przez algorytm środki klastrów są „daleko" od siebie, nawet dosyć znaczne zwiększenie dopuszczalnej wariancji nie zmieni wyniku. Jeżeli otrzymamy ten sam wynik, średnia odległość przykładu od środka klastra (J_e) również powinna być taka sama. Dlatego też badając tendencję klastrów, bada się średnią odległość przykładu od środka klastra J_e w funkcji σ^2_{max}. Na wykresie tej funkcji można wyróżnić rejony, w których wartość funkcji jest stała. Są to wartości wariancji dla których, pomimo wzrostu wariancji, średnia odległość przykładu od środka klastra nie zmienia się. Intuicyjnie, im dłuższy jest taki odcinek, tym odpowiadający mu wynik (przypisanie przykładów do klastrów) ma większe odzwierciedlenie w danych, jest „prawdziwszy". Takie rejony nazywane są **plateau**.

Siłę S (*strength*) plateau zaczynającego się w punkcie odpowiadającym wariancji σ^2_a a kończącego w σ^2_b definiuje się jako stosunek tych dwóch wartości, $S = \frac{\sigma^2_b}{\sigma^2_a}$. Plateau nazywamy **znaczącym** (*significant*) jeśli jego siła jest większa niż 2. Jeżeli połączymy dwa podobnej wielkości klastry, środek nowego klastra będzie w przybliżeniu na linii łączącej środki starych klastrów i w połowie odległości od każdego z nich. Średnia odległość wzrośnie zatem co najmniej dwukrotnie. Autorzy [22] przebadali tę hipotezę doświadczalnie, generując zbiory danych z jednorodnie rozmieszczonymi przykładami. Plateau o sile większej niż 2 znajdowane były tylko dla niewielkich zbiorów (zawierających do 50 przykładów). Powtórzenie tego eksperymentu znajduje się w rozdziale 4.6.1. Można zatem przyjąć, że znaczące plateau odpowiadają grupowaniom, które nie są artefaktami stworzony-

mi przez algorytm.

Plateau na wykresie średniej odległości przykładu J_e odpowiadają plateau na wykresie liczby klastrów M w zależności od σ^2_{max}, przy czym plateau na wykresie M są zazwyczaj dłuższe. Pozornie J_e nie powinien zmienić się aż do zmiany liczby klastrów. Przecież niewielkie jego zmiany odpowiadają zmianie przypisania pojedynczych przykładów pomiędzy zbiorami danych w kroku perturbacji, który nie zależy w żaden sposób od wariancji. Różnice te wynikają jednak z innego zachowania się algorytmu w pierwszych iteracjach, kiedy to zwiększona wariancja (należąca do przedziału, w którym widać różnicę na wykresach) może pozwolić na inne połączenie się małych, kilkuelementowych klastrów. Mechanizm tego zachowania dokładniej przedstawiam w rozdziale 4.2.

Pojęcie siły plateau pozwala nam w elegancki sposób porównać jakość proponowanych przez algorytm MVC grupowań dla różnych wartości parametrów, a co za tym idzie w łatwy sposób ocenić różne rozwiązania i wybrać najlepsze z nich.

Badanie tendecji klastrów przy użyciu MVC wiąże się z dość znacznymi nakładami obliczeniowymi. Trzeba uruchamiać od nowa algorytm dla każdej możliwej wartości σ^2_{max}. Występują tu także dwa z trzech zjawisk opisane w rozdziale 2.10: trudność w dobrze początkowej wartości wariancji i kroku, o jaki ją zwiększamy. Wartość końcowa σ^2_{end} to wariancja obliczona dla klastra zawierającego wszystkie przykłady. Jeżeli szukamy tylko znaczących plateau, de facto badanie tendecji można skończyć już po znalezieniu pierwszego końca plateau z przedziału $[\frac{\sigma^2_{end}}{2}, \sigma^2_{end}]$, bo przecież siła kolejnego plateau mogłaby być równa 2 tylko wtedy, gdyby zaczynało się dokładnie w punkcie $\frac{\sigma^2_{end}}{2}$.

3.4 Przyrostowe badanie tendencji klastrów

Jedną z własności MVC jest dosyć szybka zbieżność. Zazwyczaj już po około 10 iteracjach algorytmu otrzymuje się wynik bardzo bliski końcowemu. Podczas tych początkowych iteracji największą rolę odgrywa operator scalania klastrów. Po pierwszej iteracji jednoelementowe klastry łączone są w grupy zawierające w przybliżeniu 3 elementy. W kolejnych iteracjach algorytm scala te grupy do

momentu, w którym wariancja klastrów zaczyna być bliska σ^2_{max}. Zazwyczaj już po tej iteracji otrzymujemy wynik podobny do końcowego, gdyż operator izolacji używany jest stosunkowo rzadko, a perturbacja dotyczy zwykle ograniczonej liczby punktów.

Takie zachowanie algorytmu sugeruje nowy sposób znajdowania istotnych plateau bez konieczności uruchamiania MVC dla każdej możliwej wartości σ^2_{max}.

Załóżmy, że uruchomiliśmy MVC dla pewnej wartości σ^2_A i po określonej liczbie iteracji otrzymaliśmy stabilne rozwiązanie będące przypisaniem przykładów do klastrów. Załóżmy ponadto, iż mieliśmy szczęście i σ^2_A odpowiada początkowi plateau. Zadanie nasze polega na znalezieniu wartości σ^2_B odpowiadającej końcu plateau.

Załóżmy, że znamy σ^2_B i uruchamiamy MVC dla tej wartości, startując jednak nie z klastrami jednoelementowymi, ale z przypisaniem klastry-przykłady będącym wynikiem działania MVC dla σ^2_A. Jak zachowa się algorytm w pierwszej iteracji? Na pewno nie zostanie użyty operator izolacji, bo gdyby mógłby być użyty dla σ^2_B, mógłby być użyty również dla poprzedniej wartości wariancji σ^2_A, a zatem poprzednie rozwiązanie nie byłoby stabilne, co przeczy założeniom. Zakładając zerowe prawdopodobieństwo perturbacji losowej $P_d = 0$, nie zostanie również użyty operator perturbacji, bo jego działanie zależy jedynie od wartości funkcji celu J_e. Jedynym operatorem, który może skorzystać na zwiększeniu wariancji, jest operator scalania klastrów, którego działanie w prosty sposób zależy od maksymalnej dopuszczalnej wartości wariancji w klastrze σ^2_{max}.

Powyższe rozumowanie pozostaje prawdziwe dla $\sigma^2_{max} \in (\sigma^2_A, \sigma^2_B)$. Dlatego też można założyć, że minimalna wartość wariancji σ^2_B, która prowadzi do jakichkolwiek zmian w wyniku równa jest minimalnej wartości wariancji potrzebnej na złączenie dwóch klastrów w przypisaniu odpowiadającym σ^2_A:

$$\sigma^2_B = \min_{i,j=1\ldots M, i\neq j} Var(C_i \cup C_j) \tag{3.9}$$

W celu bezpośredniego uzyskania wartości σ^2_B przy danym σ^2_A i wyniku grupowania dla σ^2_A, należy próbować złączyć każde dwa sąsiadujące ze sobą klastry i obliczyć wspólną wariancję takiego klastra. Relację sąsiedztwa można zdefinio-

wać tak samo, jak dla operatora łączenia klastrów. Klaster C_b jest sąsiadem klastra C_a, gdy C_b zawiera jakiś przykład z zewnętrznej granicy B_a klastra C_a. Najmniejsza wspólna wariancja jest wartością σ_B^2.

Po zakończeniu algorytmu MVC dla σ_B^2 możliwy jest jeden z trzech następujących przypadków:

1. Jeśli MVC dla σ_B^2 zbiegnie do rozwiązania, które ma dokładnie jeden klaster mniej, znaczy to, że wszystkie poczynione założenia były prawdziwe, a wartość σ_B^2 odpowiada początkowi nowego plateau. Koniec tego plateau znaleźć można w analogiczny do znalezienia σ_B^2 sposób, rozpoczynając MVC z przypisaniem klastry-przykłady, które jest wynikiem obliczonym dla σ_B^2.

2. Jeśli MVC dla σ_B^2 zbiegnie do rozwiązania, w którym jest więcej niż jeden klaster mniej, prawdziwy koniec plateau leży w przedziale (σ_A^2, σ_B^2). W takim wypadku MVC uruchamiany jest jeszcze raz dla σ_A^2, ale zaczynając od grupowania obliczonego dla σ_B^2. Z dość dużym prawdopodobieństwem można przyjąć, że algorytm zbiegnie wtedy do wyniku innego niż oryginalnie uzyskany dla σ_A^2. Rozpoczynając od tego rozwiązania zaczyna się znów poszukiwanie końca plateau. Oczywiście może się zdarzyć, że algorytm postępując w ten sposób wpadnie w pętlę. W takich przypadkach algorytm uznaje całą strefę (σ_A^2, σ_B^2) za **niestabilną** i kontynuuje działanie, zaczynając od σ_B^2.

3. Jest również możliwe, choć mało prawdopodobne, że MVC zbiegnie do rozwiązania z większą liczbą klastrów. W takim przypadku by przedział (σ_A^2, σ_B^2) uznawany jest za **niestabilny**, a algorytm kontynuowany jest od σ_B^2. W tym wypadku uznajemy, że żadne znaczące plateau nie może zaczynać się w σ_B^2.

Powyższy algorytm nazywany jest przyrostowym algorytmem grupowania maksymalnej wariancji [19] (Incremental Maximum Variance Clustering algorithm, IMVC). Wynikiem działania algorytmu jest lista wartości σ_i^2 odpowiadających początkom i końcom plateau. Niektóre z tych plateau mogą zostać oznaczone

jako niestabilne. Algorytm zaczyna działanie uruchamiając oryginalny MVC z jednopunktowymi klastrami dla pewnej małej wartości σ_A^2. Grupowanie dla σ_i^2 przeprowadzane jest w oparciu o wynik grupowania dla σ_{i-1}^2. Koszt obliczeniowy algorytmu jest ograniczony z góry przez koszt jednego uruchomienia MVC pomnożonego przez liczbę znalezionych plateau.

Rozdział 4

Badania eksperymentalne

W tym rozdziale przedstawiam rezultaty badań eksperymentalnych nad algorytmem MVC oraz wyznaczaniem tendencji klastrów przy użyciu MVC i IMVC.

4.1 Parametry algorytmu i używane zbiory danych

Jeżeli nie podano inaczej, algorytm MVC był uruchamiany z parametrami takimi jak w [22]:

- prawdopodobieństwo losowej perturbacji $P_d = 0.001$;
- ilość iteracji, po których zerowano P_d i zabraniano izolacji $E_{max} = 100$;
- Wielkość zewnętrznej granicy przykładu $k = 3$;
- Wielkość wewnętrznej granicy przykładu $q = 1$;
- Wielkość zewnętrznej granicy klastra $b_a = \lfloor \sqrt{|B_a|} \rfloor$;
- Wielkość wewnętrznej granicy klastra $i_a = \lfloor \sqrt{|I_a|} \rfloor$.

Podawany w wynikach błąd J_e to średnia odległość przykładu od środka klastra, do którego został ten przykład zaliczony

$$J_e = \frac{\sum_{i=1}^{M} H(C_i)}{N}$$

$$= \frac{\sum_{i=1}^{M} \sum_{x \in C_i} dist(x, \mu(C_i))}{N},$$

gdzie $\mu(C_i)$ jest środkiem klastra C_i.

Używałem następujących zbiorów danych:

- Zbiór R15 (rysunek 4.1) składający się z 15 klastrów o dwuwymiarowym rozkładzie Gaussa. Środki klastrów ułożone są w dwa koncentryczne pierścienie. Każdy z klastrów zawiera 40 przykładów.

- Zbiór O3 (rysunek 4.2) składający się z 3 klastrów o dwuwymiarowym rozkładzie Gaussa i 3 przykładów izolowanych, leżących w znacznej odległości od tych klastrów (*outliers*). Każdy z klastrów składa się z 30 przykładów.

- Zbiór D31 (rysunek 4.3) składający się z 31 losowo rozmieszczonych, dwuwymiarowych klastrów o rozkładzie Gaussa. Każdy z klastrów zawiera 100 przykładów.

- Zbiór „iris" opisujący 150 przykładów należących do jednej z trzech klas będących różnymi typami roślin irysów. Każdy typ irysa reprezentowany jest przez 50 przykładów. Każdy z przykładów opisany jest przez 4 atrybuty, określające wymiary składowych kwiatów. Podobnie jak reszta rzeczywistych zbiorów danych, zbiór ten w oryginalnej postaci jest etykietowany, więc do analiz używać będę zbioru z usuniętymi etykietami.

- Zbiór „bupa" składa się z 345 przykładów, każdy z przykładów opisany jest przez 6 atrybutów o wartościach rzeczywistych i dwuwartościowy atrybut klasy. Zbiór ten opisuje wyniki badań pacjentów z podejrzeniem choroby wątroby. Wartości pięciu atrybutów opisują wyniki testów krwi przeprowadzonych na pacjentach, szósty atrybut to średnia liczba standardowych porcji alkoholu wypijanych w ciągu dnia.

- Zbiór „ecoli" zawiera dane dotyczące białek bakterii *e. coli*. Zbiór składa się z 336 białek charakteryzowanych przez 7 atrybutów. Każde z białek jest zaklasyfikowane do jednej z 8 klas, opisujących fragment komórki, w których białko to występuje.

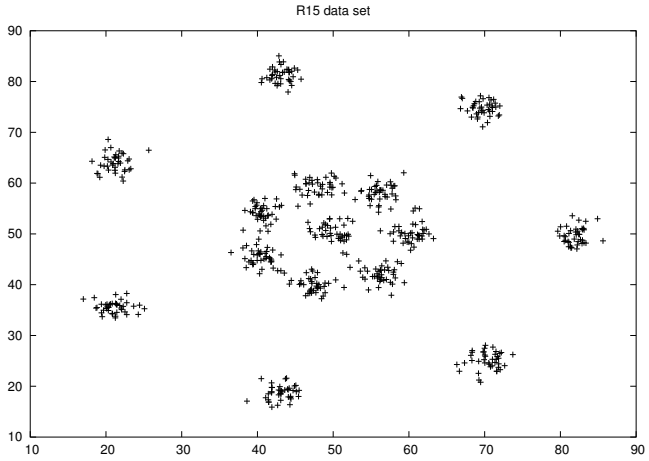
- Zbiór „glass” zawiera dane o próbkach szkła. Każdy przykład, pewna próbka szkła, opisany jest 9 atrybutami, interpretowanymi jako zawartość 9 pierwiastków w próbce. Każda z próbek jest zaklasyfikowana do jednej z sześciu kategorii (jedna z wymienionych w dokumentacji zbioru kategorii nie ma żadnego przykładu), kategorie te są również zhierarchizowane tak, że zbiór danych można dzielić na dwie, trzy lub siedem kategorii.

- Zbiór „Pima Indians Diabetes” („pima”) zawiera dane dotyczące 768 kobiet z indiańskiego plemienia Pima. Każdy z przykładów charakteryzowany jest przez 8 atrybutów oraz klasę określającą czy kobieta opisywana tymi atrybutami jest diabetykiem.

- Zbiór „segmentation” powstał po podzieleniu kolorowego obrazu na regiony zawierające pewną liczbę pikseli a następnie zaklasyfikowanie każdego regionu do jednej z siedmiu klas (cegła, niebo, liście, beton, okno, ścieżka, trawa). Każdy z przykładów opisany jest przez 19 atrybutów, określających m.in. współrzędne środka, średnią wartość poszczególnych barw składowych, średnie nasycenie, jasność i barwę (w modelu HSB) dla regionu.

- Zbiór „wine” opisuje skład chemiczny próbek wina pochodzących z tego samego regionu Włoch i uprawianych przez trzech producentów. Zbiór składa się z 178 przykładów scharakteryzowanych przez 13 rzeczywistych atrybutów. Każdy z przykładów zaliczony jest do jednej z trzech klas.

- Zbiór „yeast” jest największym z analizowanych przeze mnie zbiorów danych. Jest to zbiór analogiczny do zbioru „ecoli”, zadanie polega tu na przewidzeniu położenia białka w komórce drożdży w zależności od wartości pewnych testów przeprowadzonych na tym białku. Zbiór zawiera 1484 przykłady, z których każdy charakteryzowany jest przez 8 atrybutów i zaliczony do jednej z 10 klas. Klasy rozłożone są w zbiorze bardzo nierównomiernie, najmniej liczna reprezentowana jest przez 5 przykładów, najliczniejsza przez 463 przykłady. Zbiór ten używany jest do testowania algorytmów klasyfikacji, przy czym autorzy tego zbioru osiągnęli niezbyt zadowalające rezultaty, ponieważ jedynie 55% przykładów zostało poprawnie

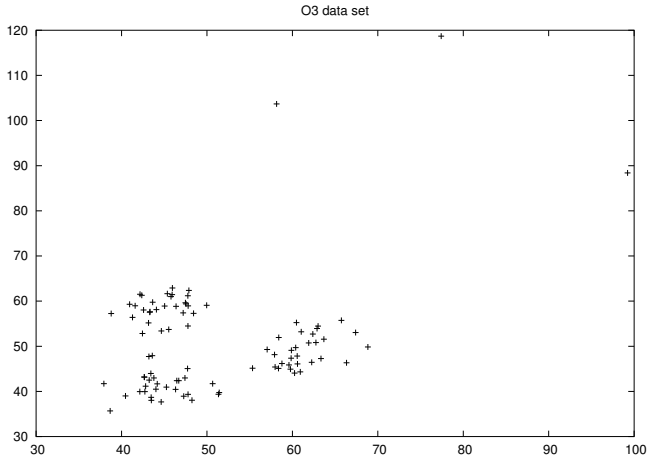


Rysunek 4.1: Zbiór danych R15.

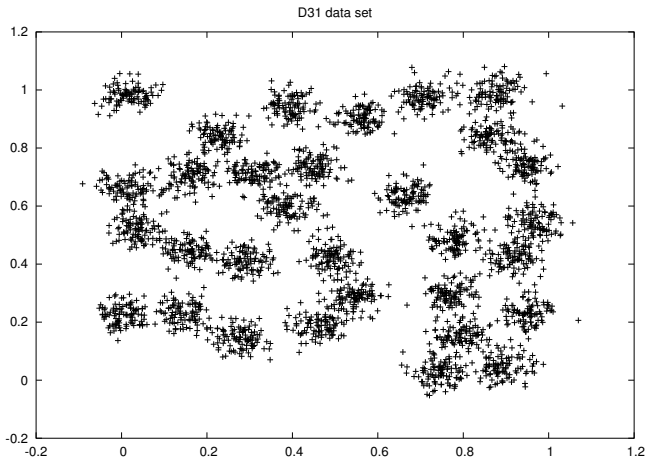
sklasyfikowanych (ten rezultat dawały różne metody: binarne drzewa decyzyjne, naiwna klasyfikacja bayesowska, ustrukturalizowany model probabilistyczny).

4.2 Analiza działania algorytmu MVC w poszczególnych iteracjach

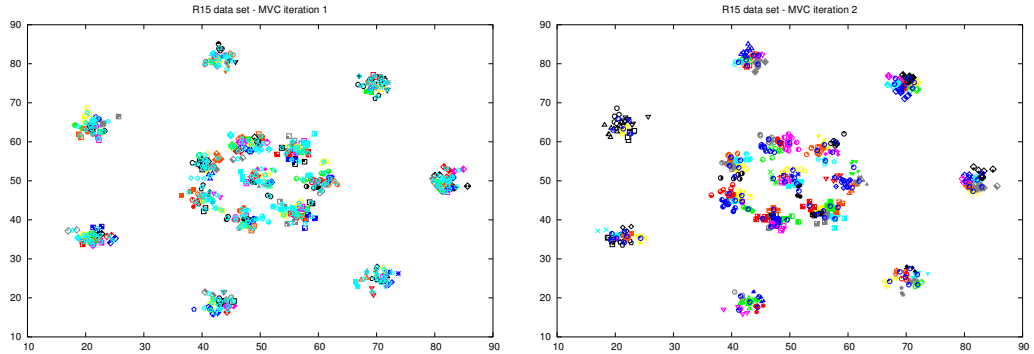
W pierwszym eksperymencie postanowiłem przeanalizować wyniki działania algorytmu MVC w poszczególnych iteracjach. Eksperymenty przeprowadzone były na zbiorze R15, maksymalna wariancja $\sigma^2_{max} = 10$. W pierwszej iteracji działał tylko operator łączenia klastrów. Po pierwszej iteracji klastry miały po 2-3 elementy (rysunek 4.4 a). W kolejnych iteracjach (rysunek 4.4 b i c) takie klastry łączyły się. Cały proces dział się bardzo szybko, już po 4 iteracji otrzymuje się wynik identyczny z rezultatem działania całego algorytmu (rysunek 4.4 d). Wyraźnie widać tu jak ważny dla algorytmu jest operator łączenia klastrów. Pozostałe operatory służą tylko do „naprawiania” spowodowanych zbyt pochopnym łączeniem.



Rysunek 4.2: Zbiór danych O3.

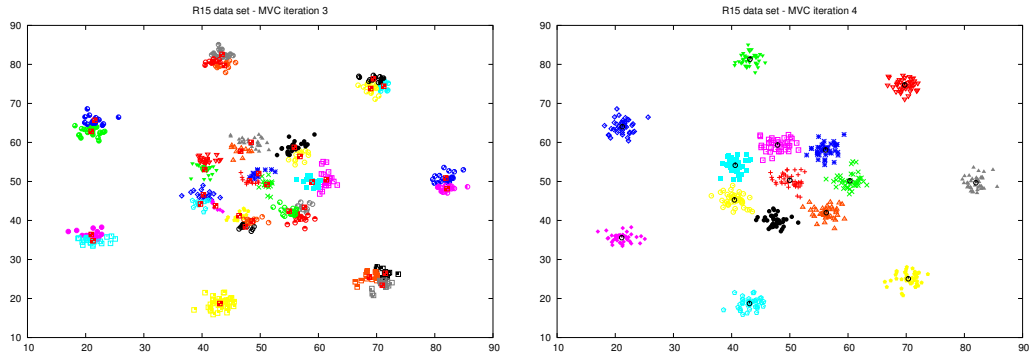


Rysunek 4.3: Zbiór danych D31.



a) po 1 iteracji

b) po 2 iteracji



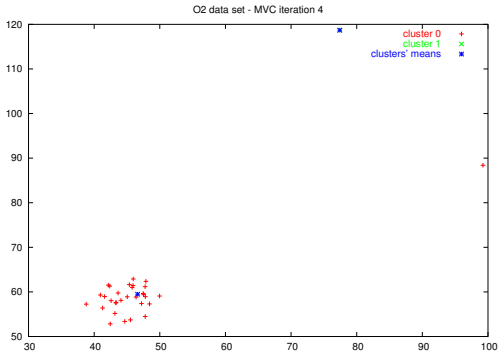
a) po 3 iteracji

b) po 4 iteracji

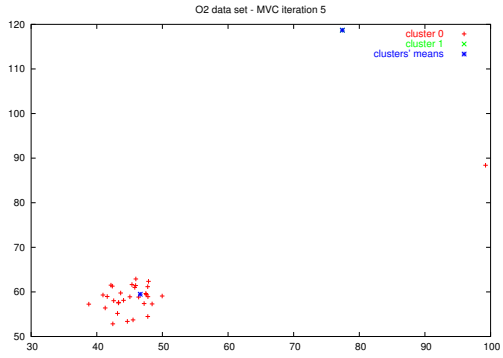
Rysunek 4.4: Działanie MVC w 4 pierwszych iteracjach. Na rysunkach przedstawione są wyniki działania algorytmu po każdej iteracji.

Kolejny eksperyment przeprowadziłem na fragmencie zbioru danych O3. Fragment ten, nazwany zbiorem O2, zawierał jeden gaussowski klaster i dwa przykłady izolowane, leżące daleko od klastra i do niego nie pasujące. Operowałem na fragmencie oryginalnego zbioru danych, by lepiej pokazać następujący mechanizm. Maksymalną wariancję postanowiłem ustalić tak, by duży klaster gaussowski mógł złączyć się z jednym z izolowanych przykładów. Ponieważ klaster zawierający klaster gaussowski i przykład o współrzędnych $(99.23, 88.40)$ miał niższą wariancję $\sigma^2 = 133.55$, maksymalna wariancja dla algorytmu została ustalona na $\sigma^2_{max} = 134$. Tak jak w poprzednim eksperymencie, już po czwartej iteracji (rysunek 4.5 a) otrzymujemy wynik, jaki zakładaliśmy: duży klaster gaussowski połączony jest z przykładem $(99.23, 88.40)$, a drugi z przykładów jest singletonem (klastrem jednoelementowym). Jednak w następnych iteracjach algorytm wpada w oscylacje. Po szóstej iteracji (rysunek 4.5 b) formowane są dwa klastry, jeden zawierający klaster gaussowski, drugi obydwa przykłady niepasujące. Po kolejnej iteracji (rysunek 4.5 c) algorytm wraca do stanu wejściowego, duży klaster gaussowski połączony z jednym z przykładów, drugi przykład jest singletonem.

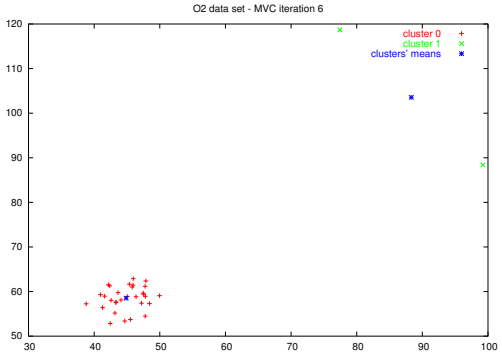
Takie zachowanie można wytłumaczyć sprzecznością jaka pojawia się dla tej wartości wariancji i dla tego zbioru danych pomiędzy operatorem scalania a operatorem permutacji. Załóżmy, że mamy sytuację, która mogła zaistnieć w trakcie czwartej iteracji, w zbiorze danych są 3 klastry: gaussowski oraz dwa singletony. Operator scalania połączy jeden z singletonów z klastrem gaussowskim (przy czym może być to operator wywołany albo dla singletonu, albo dla klastra gaussowskiego, nie ma to znaczenia), otrzymujemy grupowanie jak na rysunku 4.5 a. Problem polega jedynie na tym, że błąd jaki występuje w tym klastrze jest stosunkowo wysoki, wynosi on 129.38 (tabela 4.1). W którejś z kolejnych iteracji algorytm na pewno spróbuje zmniejszyć ów błąd operatorem permutacji, dzięki czemu przykład leżący poza klastrem gaussowskim zostanie przyłączony do singletonu. Takie grupowanie daje mniejszy błąd (34.78), ale klaster złożony z dwóch przykładów ma dużą wariancję, aż 348.40. W kolejnej iteracji działa w nim operator izolacji i jest on dzielony na dwa singletony, z których jeden prawie od razu zostaje włączony do dużego klastra.



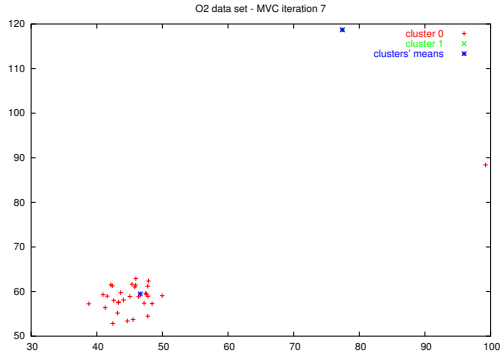
a) po 4 iteracji



b) po 5 iteracji



a) po 6 iteracji



b) po 7 iteracji

Rysunek 4.5: Oscylacje MVC w zbiorze O2 dla $\sigma^2_{max} = 134$. Na rysunkach przedstawione są wyniki działania algorytmu po każdej iteracji.

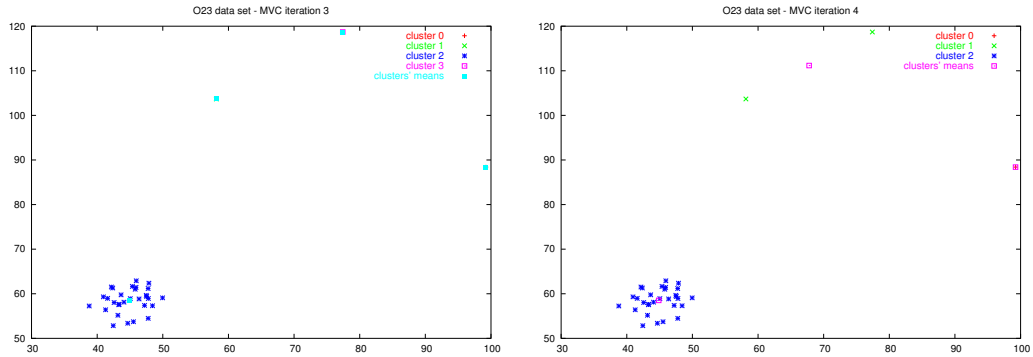
Tabela 4.1: Wyniki działania algorytmu MVC dla zbioru O2 po poszczególnych iteracjach

nr iteracji	liczba klastrów	max wariancja klastra	błąd J_e
1	15	2.2397	0.8034
2	6	4.7510	3.1076
3	4	10.4508	7.3913
4	2	133.5538	129.3803
5	2	133.5538	129.3803
6	2	348.3991	34.7876
7	2	133.5538	129.3803

Oscylacje te przerywane są dopiero w momencie, gdy po E_{max} iteracjach MVC nie może wykonywać izolacji. Algorytm zbiega wtedy do rozwiązania z większą wariacją, ale mniejszym błędem. We wszystkich dziesięciu przeprowadzonych przeze mnie eksperymentach, MVC jako rezultat zawsze zwracał grupowanie jak na rysunku 4.5 b.

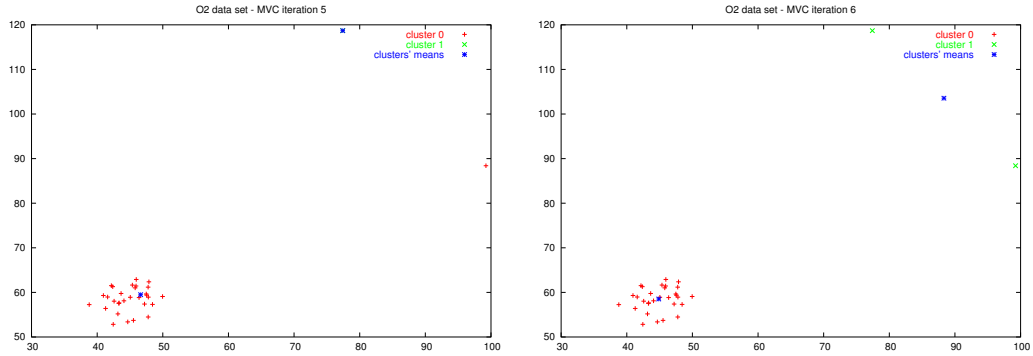
Jeżeli zbiór danych ma więcej niż dwa „niepasujące" przykłady, np. tak jak zbiór O3, dla pewnych wartości wariancji zachodzą podobne zjawiska, z tym że pojawiające się oscylacje dotyczą wszystkich trzech punktów. Po dodaniu trzeciego punktu do zbioru O2 (zbiór ten nazywał będę O23) najmniejsza możliwa wariancja pozwalająca na połączenie klastra gaussowskiego z najbliższym singletonem wynosiła $\sigma^2 = 82.66$. Po uruchomieniu MVC z $\sigma^2_{max} = 83$ zaobserwowałem następujące zjawisko. Po trzech pierwszych iteracjach algorytm dzieli dane na 3 singletony i jeden klaster gaussowski (rysunek 4.6 a). W kolejnej iteracji najbliższy z singletonów zostaje włączony do dużego klastra, ale jeszcze w tej samej iteracji jest on przepisany operatorem perturbacji do jednego z singletonów (rysunek 4.6 b). Klaster zawierający 2 singletony może następnie zostać podzielony operatorem izolacji (jego wariancja to $149,06$), a jeden z singletonów włączony z powrotem do dużego klastra (w tej samej lub kolejnej iteracji). W rezultacie otrzymuje się układ z rysunku 4.6 c. Możliwe jest również, że operator perturbacji przypisze singleton do innego klastra (rysunek 4.6 d). Mimo iż klaster taki ma większy błąd (960.52) i większą wariancję (480.26), jest to możliwe, gdy poprzedni wariant połączenia dwóch signetonów nie był brany pod uwagę z powodu innej kolejności rozpatrywania klastrów Klaster taki w następnej iteracji perturbuje i otrzymuje się albo układ jak na rysunku 4.6 b, albo taki, w którym połączone w jeden klaster są dwa singletony o największych współrzędnych X.

Moment przerwania oscylacji (czyli stan algorytmu w iteracji E_{max}) wpływa na wynik. MVC uruchomiony z wariacja $\sigma^2_{max} = 83$ zbiegał albo do rozwiązania z dwoma (rysunek 4.7 a), albo z trzema klastrami (rysunek 4.7 b). Na dziesięć uruchomień algorytmu rozwiązanie z dwoma klastrami otrzymałem dwukrotnie. Ma ono większą wariancję i większy błąd w zbiorze danych, niż rozwiązanie z 3 klastrami (tabela 4.2). Rozwiązanie to otrzymuje się, gdy w iteracji E_{max} dane



a) po 3 iteracji

b) po 4 iteracji



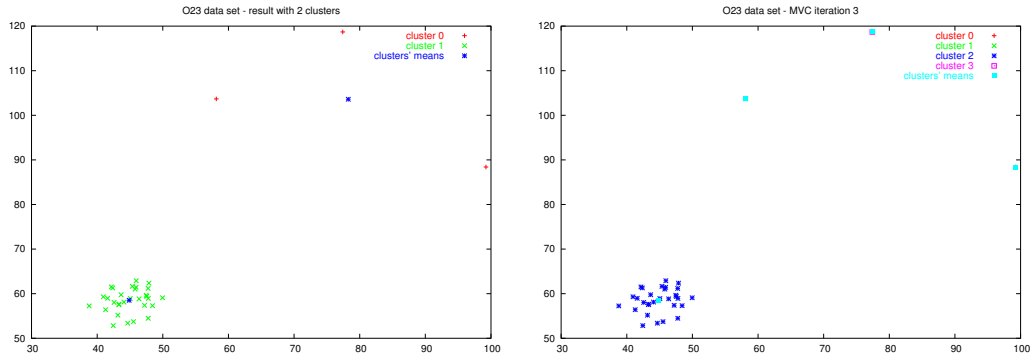
c) po 5 iteracji

d) po 15 iteracji

Rysunek 4.6: Oscylacje MVC w zbiorze O23 dla $\sigma^2_{max} = 83$. Na rysunkach przedstawione są wyniki działania algorytmu po każdej iteracji.

Tabela 4.2: Porównanie dwóch grupowań zwracanych przez MVC dla zbioru O23 i uruchomionego z wariancją $\sigma^2_{max} = 83$

wynik	max wariancja w zbiorze danych	błąd w zbiorze danych J_e
2 klastry	434.54	52.12
3 klastry	149.06	21.65



a) 2 klastry b) 3 klastry

Rysunek 4.7: Oscylacje MVC w zbiorze O23 dla $\sigma^2_{max} = 83$ prowadzą do dwóch różnych wyników.

grupowane są jak na rysunku 4.6 d, albo gdy połączone w jeden klaster są dwa przykłady o największych współrzędnych X (co jest jednym z możliwych stanów po sytuacji przedstawionej na rysunku 4.6 d)

Trzeba podkreślić, że mimo wyzerowania prawdopodobieństwa losowej perturbacji P_d, algorytm w dalszym ciągu zawiera elementy stochastyczne. Z tego powodu podane przeze mnie numery iteracji należy traktować orientacyjnie, jako ilustrację kolejności przedstawionych zachowań.

Rozważaniom o niestabilnym zachowaniu MVC poświęciłem dużo uwagi z dwóch powodów.

Po pierwsze nie wspominano o nich w [22], a zamieszczone tam wykresy mogą sugerować, że algorytm zawsze zachowuje się stabilnie, a przynajmniej, że zawsze dla takiej samej wartości wariancji daje takie samo rozwiązanie. Na prezentowanych tam wykresach nie widać charakterystycznego przeskakiwania pomiędzy dwoma różnymi rozwiązaniami w różnych uruchomieniach algorytmu, przedstawionego przeze mnie na rysunkach 4.7 a i 4.7 b.

Po drugie zachowanie to dość istotnie wpłynie na algorytm przyrostowego badania tendencji klastrów (IMVC) przedstawiony w następnych podrozdziałach.

4.3 Pomiar czasu działania algorytmu

Algorytm MVC składa się z dwóch dobrze wyodrębnionych faz. W pierwszej dla każdego przykładu konstruowana jest lista odległości od sąsiadów. Czas wykonania tej fazy zależy tylko od wielkości danych, nie od ich struktury. W drugiej fazie uruchamiany jest algorytm. Autorzy [22] w swoich pomiarach czasu pracy algorytmu mierzą tylko czas wykonania drugiej fazy, tłumacząc się, że pierwszą fazę wystarczy wykonać dla każdego zbioru danych tylko raz. Niestety, gdy wykonywaly program zawiera obie fazy, dokładne zmierzenie czasu wykonania drugiej fazy w wielozadaniowym systemie operacyjnym jest trudne. Standardowym sposobem jest zapisanie czasu startu i zatrzymania algorytmu, a następnie odjęcie tych wartości. Niestety, system operacyjny może przełączyć zadanie-algorytm na inne, i wtedy, mimo iż algorytm nie pracuje, czas pomiaru ciągle biegnie. Innym sposobem jest mierzenie czasu unixowym poleceniem time(1), które podaje czas pracy jądra systemu tylko nad procesem użytkownika. Bez ingerencji w kod polecenia time, niemożliwe jest mierzenie czasu od konkretnego momentu (startu drugiej fazy obliczeń), a nie od początku wykonania całego programu.

Biorąc pod uwagę powyższe trudności we wszystkich wynikach, które pokazują czas pracy algorytmu, podawał będę trzy wartości:

- *czas użytkownika (user time)* czas pracy systemu nad kodem użytkownika, będącego sumą czasu wykonania całego algorytmu, załadowania danych i inicjacji maszyny wirtualnej Javy;
- *procent (percentage)* procent czasu procesora, który dostał proces wykonujący algorytm;
- *czas algorytmu (algorithm time)* czas pracy algorytmu, będący różnicą dwóch wartości: momentu, w którym MVC zakończył algorytm i momentu początku algorytmu (początku drugiej fazy).

Im większy procent czasu procesora został przydzielony MVC, tym bardziej wiarygodny jest pomiar czasu pracy algorytmu. Nie można jednak czasu pracy algorytmu obliczyć jako $percentage * algorithm - time$, ponieważ nie jest pewne, czy przydział procesora w pierwszej i drugiej fazie algorytmu był taki sam.

Wszystkie pomiary w tej części przeprowadzane były na komputerze Toshiba Satellite 1400-503 z procesorem Intel Celeron 1333 MHz z 512 Mb pamięci RAM w systemie operacyjnym Red Hat Linux 9.0 działającym na jądrze 2.4.20 i JDK 1.4.2.

4.3.1 Szacowanie błędu pomiarowego czasu

Zmierzenie czasu wykonania programu w wielozadaniowym systemie operacyjnym nie jest możliwe bez ingerencji w dane jądra systemu. W celu oszacowania błędu pomiarowego, dla każdego zbioru danych dziesięciokrotnie powtórzyłem pomiar czasu, za każdym razem uruchamiając MVC z takim samym zestawem parametrów (wyniki przedstawiam w tabelach 4.3, 4.4 i 4.5). Błąd został wyznaczony jako stosunek różnicy średniej i pomiaru najbardziej odbiegającego od średniej i średniego pomiaru:

$$err = \frac{\max_i(|t_i - \hat{t}|)}{\hat{t}}, \tag{4.1}$$

gdzie $\hat{t}$ jest średnią z pomiarów:

$$\hat{t} = \frac{\sum_{i=1}^{n} t}{n}. \tag{4.2}$$

Największym zaobserwowanym błędem pomiaru czasu użytkownika jest 2.57% (dla zbioru O3) – przyjmuję, że błąd ten wynosi 3%. W pomiarze czasu działania algorytmu widać ogromny wpływ procentu CPU. W zbiorze R15, gdy dla jednego eksperymentu wielkość ta spadła o prawie 10 punktów procentowych, czas działania zdecydowanie zwiększył się. Gdyby brać ten pomiar pod uwagę, błąd wyniósłby aż 13%. Dlatego zdecydowałem się odrzucić ten pomiar i powtarzać eksperyment, jeśli sytuacja taka pojawiłaby się w przyszłości. Przyjmuję, że błąd pomiaru czasu działania algorytmu (przy założeniu podobnego do siebie wykorzystania procesora w pomiarach) wynosi 5%.

Tabela 4.3: Pomiar czasu pracy procesora nad kodem użytkownika dla zbioru D31 przy wariancji $\sigma^2_{max} = 0.004$. Kolejne wiersze przedstawiają wyniki pomiaru czasu w kolejnych powtórzeniach algorytmu.

powtórzenie	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
0	228.20	98%	59317
1	227.74	96%	61167
2	227.58	97%	59374
3	227.59	98%	59511
4	227.94	97%	59473
5	230.89	97%	63308
6	227.81	95%	62150
7	228.56	95%	60774
8	228.56	93%	59270
9	227.28	98%	59175
średnia	228.215	96.4%	60351
błąd	1.2%	–	4.9%

Tabela 4.4: Pomiar czasu pracy procesora nad kodem użytkownika dla zbioru R15 przy wariancji $\sigma^2_{max} = 10.0$. Kolejne wiersze przedstawiają wyniki pomiaru czasu w kolejnych powtórzeniach algorytmu.

powtórzenie	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
0	17.53	95%	11837
1	17.70	95%	11876
2	17.34	97%	11247
3	17.36	88%	13239[1]
4	17.41	97%	11338
5	17.38	96%	11582
6	17.52	95%	11466
7	17.31	97%	11272
8	17.33	97%	11352
9	17.43	97%	11395
średnia	17.438	96.2%	11485
błąd	1.50%	–	3.40%

Tabela 4.5: Pomiar czasu pracy procesora nad kodem użytkownika dla zbioru O3 przy wariancji $\sigma^2_{max} = 50.0$. Kolejne wiersze przedstawiają wyniki pomiaru czasu w kolejnych powtórzeniach algorytmu.

powtórzenie	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
0	4.11	78%	4010
1	3.99	75%	3806
2	4.02	81%	3808
3	4.15	82%	3851
4	4.04	80%	3868
5	4.02	81%	3834
6	4.15	84%	3842
7	4.06	81%	3860
8	4.12	82%	3835
9	4.19	83%	3835
średnia	4.085	80.7%	3854
błąd	2.57%	–	4%

4.4 Analiza wpływu poszczególnych parametrów na działanie algorytmu

W tym rozdziale opiszę, jak wpływają na działanie algorytmu i na uzyskiwane wyniki parametry uznane w [22] za drugorzędne: wielkości sąsiedztwa wewnętrznego q i zewnętrznego k, prawdopodobieństwo losowej perturbacji P_d. Dla każdego zbioru danych wariancja będzie ustalana w środku plateau o największej sile.

4.4.1 Wielkość sąsiedztwa wewnętrznego

Wielkość sąsiedztwa wewnętrznego q wpływa na liczbę kandydatów do ewentualnej izolacji, jeśli wariancja w klastrze została przekroczona. Parametr ten kontroluje liczbę najdalszych sąsiadów z tego samego klastra branych dla każdego przykładu do wewnętrznej granicy I_a klastra C_a.

Wyniki eksperymentów dla poszczególnych zbiorów danych przedstawiam w

Tabela 4.6: Analiza wpływu wielkości wewnętrznej granicy q na działanie algorytmu dla zbioru R15 przy wariancji $\sigma^2_{max} = 10.0$. W tabeli podaję czas pracy procesora nad kodem użytkownika (czas użytkownika).

wewnętrzna granica q	czas pracy algorytmu [s]
1	9.69
2	10.62
3	10.73
4	10.61
5	9.76

Tabela 4.7: Analiza wpływu wielkości wewnętrznej granicy q na działanie algorytmu dla zbioru O3 przy wariancji $\sigma^2_{max} = 50.0$. W tabeli podaję czas pracy procesora nad kodem użytkownika (czas użytkownika).

wewnętrzna granica q	czas pracy algorytmu [s]
1	2.49
2	2.47
3	2.47
4	2.41
5	2.51

tabelach:

- zbiór R15 w tabeli 4.6
- zbiór O3 w tabeli 4.7
- zbiór D31 w tabeli 4.8

Na podstawie wyników można wnioskować, że wielkość wewnętrznej granicy ma mały wpływ na czas wykonania algorytmu. Zaistniałe różnice mieszczą się w granicach błędu pomiarowego. Takich wyników można się spodziewać, ponieważ krok izolacji, w którym granice są przeglądane, zachodzi stosunkowo rzadko, przynajmniej dla wartości wariancji σ^2_{max} dobranej do zbioru danych.

Same wyniki grupowania również nie różnią się dla różnych wartości σ^2_{max}. Za wyjątkiem zbioru D31, algorytm po trzech lub czterech pierwszych iteracjach

Tabela 4.8: Analiza wpływu wielkości wewnętrznej granicy q na działanie algorytmu dla zbioru D31 przy wariancji $\sigma^2_{max} = 0.004$. W tabeli podaję czas pracy procesora nad kodem użytkownika (czas użytkownika).

wewnętrzna granica q	czas pracy algorytmu [s]
1	141.49
2	140.09
3	139.10
4	140.70
5	141.82

dochodził do prawidłowego rozwiązania. W kolejnych iteracjach zmiany praktycznie nie zachodziły.

Można było spodziewać się takich wyników, ponieważ za wyjątkiem operatora izolacji, wewnętrzna granica nie jest wykorzystywana. Z kolei sam operator izolacji również używany jest bardzo rzadko, szczególnie dla dopasowanego do zbioru danych parametru maksymalnej wariancji σ^2_{max}.

4.4.2 Wielkość sąsiedztwa zewnętrznego

Wielkość sąsiedztwa zewnętrznego k wpływa na liczbę kandydatów branych pod uwagę w operatorach scalania i perturbacji. Parametr ten kontroluje liczbę najbliższych sąsiadów z innych klastrów branych dla każdego przykładu do zewnętrznej granicy B_a klastra C_a.

Wyniki eksperymentów dla poszczególnych zbiorów danych przedstawiam w tabelach:

- zbiór R15 w tabeli 4.9
- zbiór O3 w tabeli 4.10
- zbiór D31 w tabeli 4.11

Zewnętrzna granica używana jest przez MVC stosunkowo często. Wykorzystywana jest zarówno w operatorze scalania, jak i w operatorze perturbacji. Jednak zwiększenie k nie powiększa znacząco czasu działania algorytmu (poza zbiorem

Tabela 4.9: Analiza wpływu wielkości zewnętrznej granicy k na działanie algorytmu dla zbioru R15 przy wariancji $\sigma^2_{max} = 10.0$.

zewnętrzna granica k	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
1	16.67	94%	10870
2	17.17	94%	11725
3	17.41	92%	12130
4	17.73	96%	11790
5	17.78	97%	11824

Tabela 4.10: Analiza wpływu wielkości zewnętrznej granicy k na działanie algorytmu dla zbioru O3 przy wariancji $\sigma^2_{max} = 50.0$.

zewnętrzna granica k	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
1	4.07	77%	4041
2	3.94	79%	3945
3	4.07	79%	3864
4	4.21	78%	3899
5	4.16	81%	3727

Tabela 4.11: Analiza wpływu wielkości zewnętrznej granicy k na działanie algorytmu dla zbioru D31 przy wariancji $\sigma^2_{max} = 0.004$.

zewnętrzna granica k	czas użytkownika [s]	procent CPU	czas algorytmu [ms]
1	221.80	91%	56804
2	223.89	97%	57281
3	227.96	96%	61713
4	230.77	97%	61607
5	232.60	98%	63404

D31, gdzie wzrosty zarówno czasu użytkownika jak i czasu algorytmu są nieznacznie większe od błędu pomiarowego). Wytłumaczeniem tego może być fakt, że zewnętrzne granice brane do obliczeń są stosunkowo małe. Dla zbioru O3 wielkość zewnętrznej granicy $|B_a|$ dla gaussowskich klastrów to 11-12 przykładów dla $k = 5$ i 6-7 dla $k = 3$. Różnica tych 6 przykładów nie powinna być widoczna w czasie działania algorytmu. Z kolei w przypadku zbioru D31 klastrów jest więcej, więc zwiększenie granicy może prowadzić do wzięcia pod uwagę większej liczby klastrów w operatorze scalania. Operator ten jest bardziej kosztowny niż perturbacja. Jednak również dla zbioru D31 różnica pomiędzy największym a najmniejszym czasem wykonania wynosi zaledwie 5%.

Podobnie jak w przypadku granicy wewnętrznej, zmiana k w żaden sposób nie wpływa na wyniki grupowania.

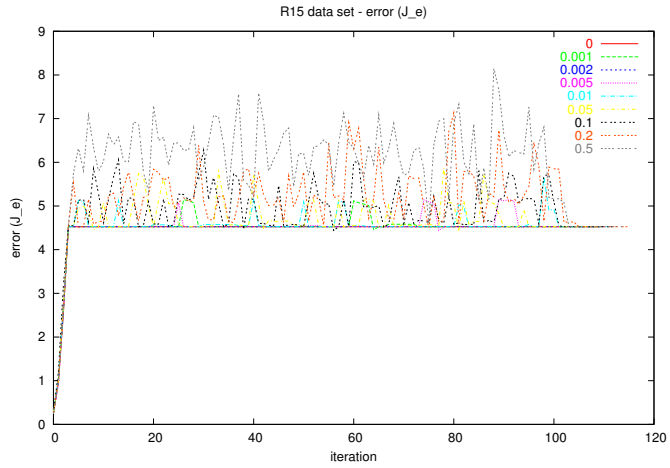
4.4.3 Prawdopodobieństwo losowej perturbacji

Parametr określający prawdopodobieństwo losowej perturbacji P_d kontroluje stopień losowości w przeglądaniu przestrzeni rozwiązań. Ponieważ liczba iteracji E_{max}, w których MVC zezwala na losową perturbację jest stała, wykonanie algorytmu za każdym razem zabiera ten sam czas. Z tego powodu w tym rozdziale przedstawię wykresy prezentujące błąd (średnią odległość przykładu od środka klastra) w funkcji iteracji algorytmu. Te wykresy pokazują sposób w jaki algorytm przegląda przestrzeń rozwiązań.

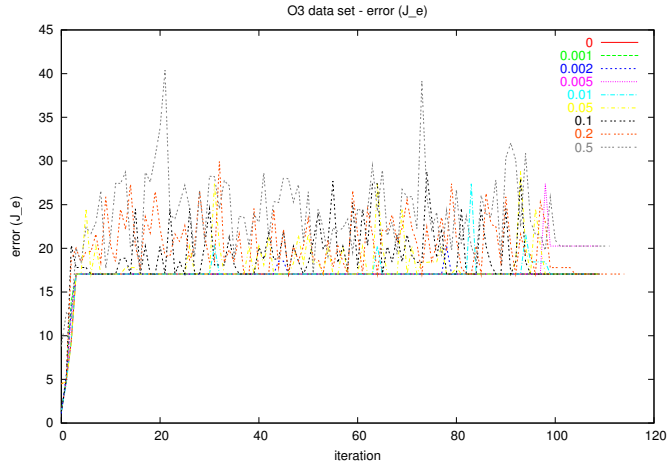
Wyniki eksperymentów dla poszczególnych zbiorów danych przedstawiam na wykresach:

- zbiór R15 na wykresie 4.8
- zbiór O3 na wykresie 4.9
- zbiór D31 na wykresie 4.10

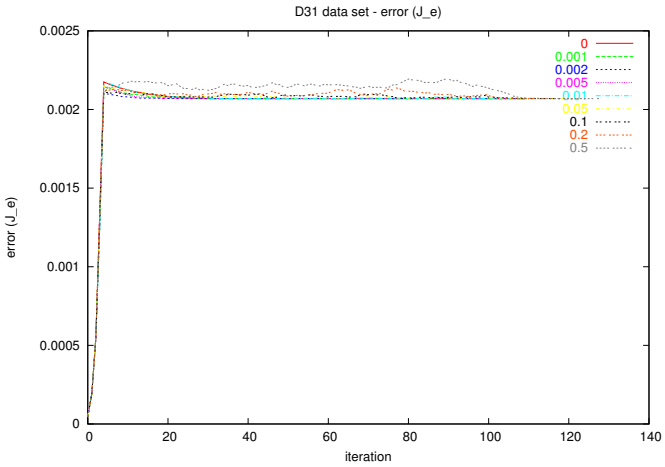
Analizując powyższe wykresy można stwierdzić, że im większe P_d, tym gorzej radzi sobie algorytm. Nawet dla sugerowanej w [22] wartości $P_d = 0.001$, raz na kilka iteracji ocena grupowania pogarsza się, dochodząc do optimum po kilku



Rysunek 4.8: Wykres średniej odległości od klastra w funkcji iteracji dla różnych wartości perturbacji w zbiorze R15.



Rysunek 4.9: Wykres średniej odległości od klastra w funkcji iteracji dla różnych wartości perturbacji w zbiorze O3.



Rysunek 4.10: Wykres średniej odległości od klastra w funkcji iteracji dla różnych wartości perturbacji w zbiorze D31.

kolejnych iteracjach. Drugą prawidłowością jest to, że zwiększanie P_d najmniej szkodzi wynikom uzyskiwanym dla największego zbioru danych – D31. W tym zbiorze klastry leżą stosunkowo blisko siebie, dlatego ewentualna zmiana przynależności któregoś z przykładów nie szkodzi tak bardzo, jak np. w przypadku O3, kiedy to losowa perturbacja często prowadzi do zmniejszenia liczby klastrów.

Losowa perturbacja jest operatorem analogicznym do operatora mutacji w algorytmach ewolucyjnych. Zbyt mała wartość perturbacji nie pozwala na dostateczne przejrzenie przestrzeni rozwiązań, zbyt duża powoduje zupełnie losowe zmiany w rezultacie. Jednak w odróżnieniu od algorytmów ewolucyjnych, w przypadku MVC nie ma pojęcia populacji możliwych wyników i dlatego jedna losowa perturbacja może prawie od razu popsuć dobre rozwiązanie, co będzie doskonale widoczne, jeśli zajdzie ona w jednej z ostatnich przed E_{max} iteracji algorytmu.

Używane przeze mnie zbiory danych nie pozwalają tak naprawdę na zbadanie optymalnej wartości tego parametru, gdyż lokalne optimum jest bardzo często tożsame z optimum globalnym. Dowodzą tego „idealne” wyniki dla zerowego prawdopodobieństwa permutacji. Okazuje się, że przestrzeń, którą definiuje MVC, jest stosunkowo prosta do przeglądania, więc starcza tu praktycznie deterministyczny algorytm. Trudno wymyślić zbiór danych posiadający wiele maksimów lo-

kalnych, będący odpowiednikiem *trap function* dla algorytmów ewolucyjnych. Z drugiej strony ta cecha jest niewątpliwie dużą zaletą algorytmu.

4.5 Porównanie algorytmu MVC z algorytmem k-środków

W tym rozdziale porównuję wyniki działania algorytmu MVC z algorytmem k-środków. Używałem nieznacznie zmodyfikowanego algorytmu k-środków z biblioteki Weka [24]. Oryginalna metoda wyznaczająca odległość pomiędzy dwoma przykładami normalizowała atrybuty i została zastąpiona funkcją używaną przez MVC. Dodałem również metodę pozwalającą na obliczenie błędu J_e, by móc porównywać wyniki MVC i k-środków tylko na podstawie J_e proponowanego grupowania. Po tych modyfikacjach z dużym prawdopodobieństwem można przyjąć, że jeżeli dwa algorytmy proponują grupowania o tym samym J_e, są to te same grupowania.

Ponieważ algorytm k-środków nie zawsze zatrzymuje się w minimum globalnym, dla każdego zbioru danych był on uruchamiany 100 krotnie. Dla większości zbiorów danych algorytm MVC również powtarzany był 100 krotnie w celu sprawdzenia, w jaki sposób niedeterminizm obecny w algorytmie wpływa na wyniki. Oba algorytmy uruchamiałem z parametrami pozwalającymi dla danego zbioru danych uzyskać najlepsze możliwe grupowanie. Liczba klastrów k w algorytmie k-środków odpowiadała liczbie klastrów w zbiorze danych. Parametr maksymalnej wariancji σ^2_{max} algorytmu MVC pochodził ze środka plateau odpowiadającego na wykresie tendencji grupowania odpowiedniej dla danego zbioru liczbie klastrów (wykresy te przedstawiam w rozdziale 4.6.2).

W przypadku algorytmu k-środków czas podawany w wynikach to różnica pomiędzy chwilą skończenia algorytmu a chwilą zainicjowania go. Czas ten uwzględnia ewentualne przełączenia zadań w systemie operacyjnym (nie jest to *user time*). Jednak ponieważ algorytm nie trwał długo i ponieważ nie było w tym czasie uruchomionych innych procesów intensywnie korzystających z procesora, można uznać te wyniki za wiarygodne, oczywiście jako wielkość orientacyjną.

Czas działania algorytmu MVC podawany w tym rozdziale zmierzony został tak samo jak w rozdziale 4.3 i jest średnią z 10 niezależnych przebiegów algorytmu. Jedyną różnicą jest używanie przeze mnie w tym rozdziale programu skompilowanego z opcją optymalizacji i bez opcji śledzenia (*debug*). Taki program był około 40% szybszy od wersji używanej w rodziale 4.3. Ponieważ w trakcie badań eksperymentalnych okazało się, że MVC jest kilka rzędów wielkości wolniejszy niż k-środki, podczas badania rzeczywistych zbiorów danych zrezygnowałem z tak ścisłego pomiaru czasu działania algorytmu MVC i mierzyłem go w ten sam sposób, co czas k-środków.

Efekt działania każdego z algorytmów uznawany był za poprawny (trafienie), gdy błąd J_e proponowanego przez algorytm grupowania odbiegał od błędu będącego globalnym minimum J_e^* o nie więcej niż pewna ustalona δ (zazwyczaj $\delta = 10^{-10}$ lub $\delta = 10^{-14}$). Algorytm MVC porusza się w nieco innej przestrzeni rozwiązań (inne są ograniczenia na minimalizację). Teoretycznie J_e^{*MVC} dla MVC może być inne niż $J_e^{*kmeans}$ dla k-środków, więc jeżeli oba algorytmy dają taką samą liczbę grup, to:

$$J_e^{*MVC} \geq J_e^{*kmeans}.$$

Jeżeli k-środki w 100 niezależnych wykonaniach rzeczywiście znalazły rozwiązanie o niższym błędzie niż J_e^{*MVC}, ten właśnie wynik (a nie J_e^{*MVC}) uznawał będę za poprawny (trafienie) dla k-środków. Używane wartości J_e^* i δ będę podawał przy każdym zbiorze danych.

Należy podkreślić, że grupowanie odpowiadające globalnemu minimum błędu J_e nie musi być tożsame z grupowaniem, które narzuca się dla zbioru danych. Zbiór R15 składa się z 15 gaussowskich klastrów, każdy z nich zawiera 40 przykładów. Błąd J_e wyliczony dla takiego grupowania to 4.5779. MVC uruchomiony z wariancją $\sigma^2_{max} = 10.0$ daje natomiast rozwiązanie, w którym dwa z pietnastu klastrów zawierają 39 przykładów, a dwa – 41. Klasteryzacja ta ma mniejszy błąd, który wynosi $J_e = 4.5258$. Podobnie w zbiorze D31, gdzie grupowanie „idealne" ma błąd $J_e = 0.002160$, natomiast MVC zwraca grupowanie z klastrami zawierającymi 99, 100, a czasem i 104 przykładów z błędem $J_e = 0.002069$.

Tabela 4.12: Zbiór O3: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	czas [ms] całość	czas [ms] użytkownika	czas [ms] algorytmu	CPU
k-środki	$M = 6, \delta = 10^{-10}$ $J_e^{*kmeans} = 16.489270818399046$	1	6	-	-	-
MVC	$\sigma^2_{max} = 50, \delta = 10^{-10}$ $J_e^{*MVC} = 17.05881200716846,$	99	-	1657	872	73%

4.5.1 Wygenerowane zbiory danych

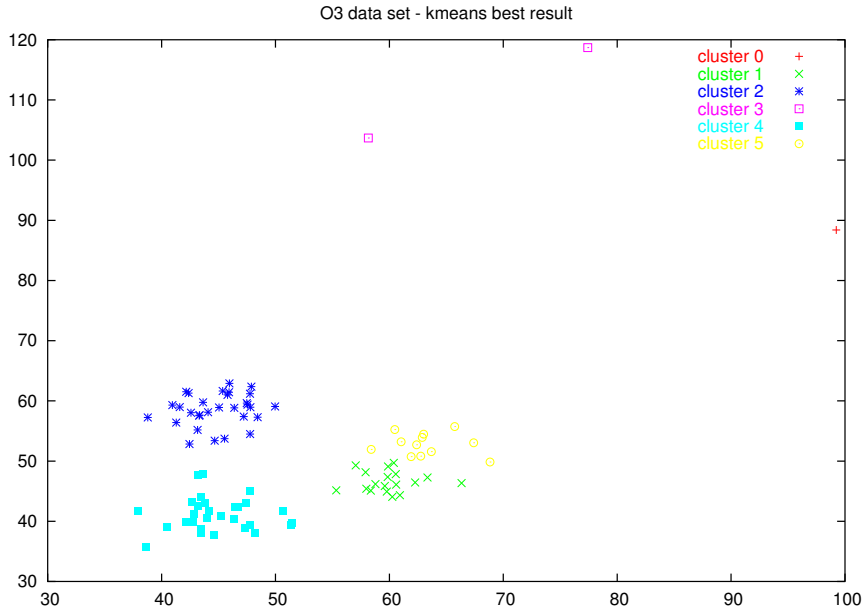
W pierwszych eksperymentach porównałem wyniki działania obu algorytmów na zbiorach danych opisanych w [22].

Zbiór O3

W zbiorze O3 (wyniki opisane w tabeli 4.12) algorytm k-środków nie był w stanie znaleźć poprawnej struktury 3 klastrów gaussowskich i 3 singletonów. Algorytm ten trafił na lokalne minimum o błędzie J_e niższym niż błąd przy grupowaniu algorytmem MVC. Jednak z analizy tego wyniku (rysunek 4.11) wynika, że grupowanie to nie pasuje do zbioru danych. W tym zbiorze danych istotne jest wyróżnienie 3 singletonów, jako niepasujących do żadnej z grup. Z tak postawionym zadaniem k-środki nie mogły sobie poradzić. Algorytm MVC działał prawie bezbłędnie, jedynie w jednym wykonaniu nie uzyskał właściwego wyniku. Średni czas wykonania k-środków jest prawie tysiąckrotnie mniejszy niż średni czas wykonania MVC.

Zbiór R15

W zbiorze R15 możliwe są dwa grupowania złożone z 15 i 8 klastrów. W ciągu 100 wykonań algorytm k-środków znalazł poprawną strukturę złożoną z 15 klastrów trzykrotnie, a z 8 klastrów – dwunastokrotnie (tabela 4.12). Algorytm k-środków nie znalazł rozwiązania lepszego pod względem błędu J_e. Algorytm MVC w każdym ze 100 powtórzeń zbiegł do poprawnego rozwiązania. Czas działania algorytmu MVC jest ponad stukrotnie większy niż czas działania algorytmu



Rysunek 4.11: Wynik grupowania zbioru O3 algorytmem k-środków o najmniejszym błędzie $J_e = 16.489270818399046$.

k-środków.

Zbiór D31

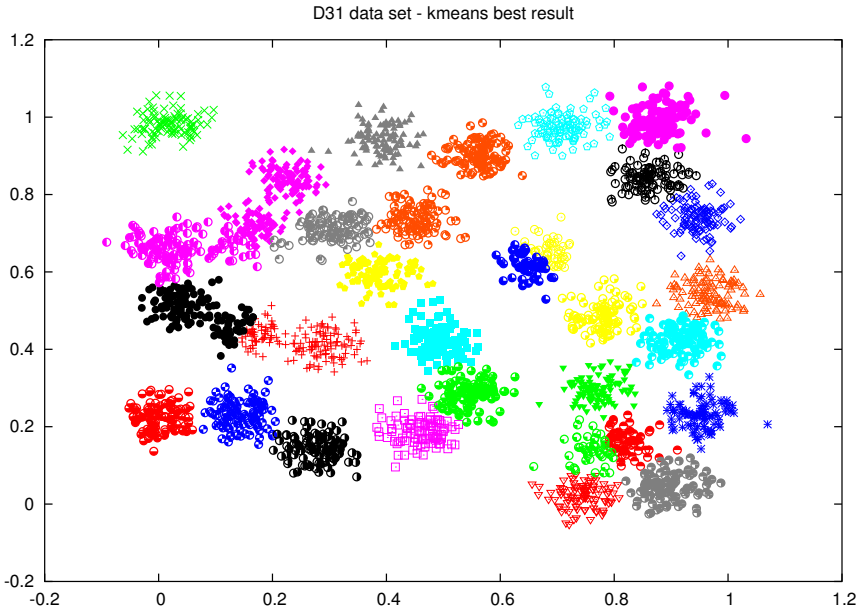
Zbiór D31 jest największym z badanych przeze mnie syntetycznych zbiorów danych. Algorytm k-środków w żadnej ze stu prób nie był w stanie znaleźć poprawnego grupowania (tabela 4.14). Najlepsze znalezione grupowanie z błędem $J_e = 0.0025797238457119674$ przedstawiam na rysunku 4.12. Algorytm MVC ze zbiorem D31 radzi sobie znacząco lepiej – na 20 powtórzeń algorytmu za każdym razem grupował dane właściwie. Niestety czas działania MVC jest duży. Chociaż sam algorytm trwał około 40 sekund, obliczanie list sąsiedztwa zajmowało około czterokrotnie więcej czasu. Warto jednak zauważyć, że czas wykonania algorytmu k-środków także rośnie i wynosi około 850 ms, co oznacza że przewaga kilku rzędów wielkości, jaką dla mniejszych zbiorów danych miały k-środki, maleje – dla zbioru D31 algorytm MVC jest około 60 krotnie wolniejszy.

Tabela 4.13: Zbiór R15: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	czas [ms]			CPU
			całość	użytkownika	algorytmu	
k-środki	$M = 15, \delta = 10^{-10}$, $J_e^* = 4.525793367224307$	3	56	-	-	-
k-środki	$M = 8, \delta = 10^{-10}$, $J_e^* = 53.28816445833333$	12	41	-	-	-
MVC	$\sigma_{max}^2 = 10, \delta = 10^{-10}$, $J_e^* = 4.525793367224307$	100	-	10532	3770	97%
MVC	$\sigma_{max}^2 = 100, \delta = 10^{-10}$, $J_e^* = 53.28816445833333$	100	-	19573	13130	97%

Tabela 4.14: Zbiór D31: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	czas [ms]			CPU
			całość	użytkownika	algorytmu	
k-środki	$M = 31, \delta = 10^{-17}$, $J_e^* = 0.002069185100796537$	1	851	-	-	-
MVC	$\sigma_{max}^2 = 0.004, \delta = 10^{-17}$, $J_e^* = 0.002069185100796537$	20	-	210502	39433	97%



Rysunek 4.12: Wynik grupowania zbioru D31 algorytmem k-środków o najmniejszym błędzie $J_e = 0.0025797238457119674$.

4.5.2 Zbiory standardowe

W tym rozdziale porównam wyniki działania algorytmu k-środków i MVC na pewnych standardowych zbiorach danych dostępnych w repozytorium maszynowego uczenia się w Internecie [5].

Ponieważ dotychczasowe eksperymenty przeprowadzone były na różnorodnych zbiorach danych i jednoznacznie wynikało z nich, że algorytm MVC jest o kilka rzędów wielkości wolniejszy od algorytmu k-środków, w następnych eksperymentach (z wyjątkiem zbioru *iris*) zdecydowałem się nie mierzyć ściśle czasu wykonania algorytmu. Czas podawany w kolejnych wynikach jest uśrednionym czasem jednego, pełnego wykonania algorytmu (obliczenia list sąsiedztwa i samego algorytmu) mierzonym z poziomu programu uruchamiającego algorytm.

Ponieważ kolejne eksperymenty przeprowadzane będą na zbiorach danych o zróżnicowanych wartościach atrybutów, zmodyfikowałem funkcję wyznaczającą odległość między dwoma przykładami. W kolejnych eksperymentach używał bę-

Tabela 4.15: Zbiór *iris*: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	czas [ms]			CPU
			całość	użytkownika	algorytmu	
k-środki	$M = 3, \delta = 10^{-14}$, $J_e^* = 0.5262722761743065$	42	9	-	-	-
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^* = 1.0157913765155946$	97	1	-	-	-
MVC	$\sigma_{max}^2 = 1.0, \delta = 10^{-14}$, $J_e^* = 0.5262722761743065$	100	-	2711	1535	86%
MVC	$\sigma_{max}^2 = 2.0, \delta = 10^{-14}$, $J_e^* = 1.0157913765155946$	100	-	2597	1514	82%

dę znormalizowanej miary euklidesowej, opisanej w rozdziale 2.5.1. Podczas eksperymentów na zbiorze *iris* porównałem wyniki grupowania przy użyciu odległości euklidesowej i znormalizowanej odległości euklidesowej.

Zbiór iris

Pierwszym zbadanym zbiorem był wielokrotnie używany zbiór *iris*. Ze względu na liniową nieseparowalność trzech oryginalnie występujących w zbiorze klas (dokładniejszy opis tego zbioru zamieszczony jest w rozdziale 4.1) przeprowadziłem dwa eksperymenty, grupując dane na trzy i dwa klastry. Wyniki eksperymentów przedstawione są w tabeli 4.15. Przy podziale na trzy klastry algorytm k-środków dosyć często, w ponad 40% prób, znajduje rozwiązanie optymalne. Przy szukaniu dwóch klastrów k-środki działają jeszcze skuteczniej, zwracając rozwiązanie w prawie każdym uruchomieniu. Algorytm MVC w obu przypadkach był bezbłędny, za każdym uruchomieniem grupując właściwie. Czas działania algorytmu MVC jest o kilka rzędów wielkości większy, niż czas działania k-środków. Rezultaty grupowania, prezentujące przypisanie poszczególnych gatunków irysów do klas przedstawiam w tabelach: dla trzech klastrów w tabeli 4.17, dla dwóch w tabeli 4.16.

Dla dwóch klastrów znalezione przez algorytmy grupowanie dosyć dobrze pokrywa się z klasami. Do jednej z grup algorytmy przypisują irysy z gatunku *se-*

Tabela 4.16: Wynik grupowania dla zbioru *iris*: liczba przykładów z poszczególnych klas przypisana do grup przy podziale na dwa klastry.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
Iris-setosa	0	50
Iris-versicolor	47	3
Iris-virginica	50	0

Tabela 4.17: Wynik grupowania dla zbioru *iris*: liczba przykładów z poszczególnych klas przypisana do grup przy podziale na trzy klastry.

klasa	liczba przykładów przypisanych do		
	grupy I	grupy II	grupy III
Iris-setosa	0	0	50
Iris-versicolor	2	48	0
Iris-virginica	36	14	0

tosa, do drugiej z gatunków *versicolor* i *virginica*. W przypadku podziału na trzy grupy, irysy *setosa* nadal trafiają do osobnej grupy. W pozostałych dwóch grupach przykłady z każdej kategorii są bardziej wymieszane, choć w każdej z grup widać dominację pewnej odmiany, np. do grupy II algorytmy zaliczyły prawie wszystkie irysy z gatunku *versicolor*.

iris - znormalizowany

Kolejne eksperymenty przeprowadziłem używając zbioru *iris* i znormalizowanej odległości euklidesowej. Porównanie wyników działania algorytmów przedstawione są w tabeli 4.18.

Analizując wyniki w tabeli 4.18 widać dwa zachowania. Po pierwsze, algorytm k-środków działa skuteczniej. Zarówno dla dwóch, jak i dla trzech klastrów częściej znajduje rozwiązanie, niż gdy używa normy euklidesowej. Z drugiej strony skuteczność MVC dla dwóch klastrów spada. Przypisanie przykładów do klastrów nie zmieniło się znacznie (tabele 4.20 i 4.19). Dla dwóch klastrów jeden z nich zawierał wszystkie irysy z gatunku setosa, a drugi z dwóch pozostałych

Tabela 4.18: Zbiór *iris* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	czas całości [ms]
k-środki	$M = 3, \delta = 10^{-14}$, $J_e^* = 0.5262722761743065$	55	9
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^* = 0.124116102398678$	100	1
MVC	$\sigma_{max}^2 = 0.12, \delta = 10^{-14}$, $J_e^* = 0.09759617577099647$	84	1842
MVC	$\sigma_{max}^2 = 0.19, \delta = 10^{-14}$, $J_e^* = 0.124116102398678$	100	2147

Tabela 4.19: Wynik grupowania dla zbioru *iris* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do grup przy podziale na dwa klastry.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
Iris-setosa	0	50
Iris-versicolor	50	0
Iris-virginica	50	0

Tabela 4.20: Wynik grupowania dla zbioru *iris* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do grup przy podziale na trzy klastry.

klasa	liczba przykładów przypisanych do		
	grupy I	grupy II	grupy III
Iris-setosa	0	0	50
Iris-versicolor	3	47	0
Iris-virginica	36	14	0

Tabela 4.21: Zbiór *bupa* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^* = 0.10725658990643493$	1	2	39
MVC	$\sigma_{max}^2 = 0.115, \delta = 10^{-14}$, $J_e^* = 0.10725658990643493$	25	2	16698

Tabela 4.22: Wynik grupowania dla zbioru *bupa* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do grup przy podziale na dwa klastry

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
1	110	35
2	141	59

gatunków, zatem rezultat poprawił się.

bupa

Kolejnym analizowanym zbiorem był zbiór danych o chorobach wątroby *bupa*. Eksperymenty na tym zbiorze danych przeprowadzałem w analogiczny sposób do eksperymentów na zbiorze *iris*. Badane algorytmy otrzymały na wejściu zbiór bez wartości atrybutu klasy, a następnie porównywałem grupowanie proponowane przez algorytm z podziałem na klasy w oryginalnym zbiorze.

Najlepsze i najczęściej zwracane rozwiązanie znalezione przez algorytm k-środków miało błąd $J_e = 0.10764$, natomiast algorytm MVC był w stanie znaleźć rozwiązanie z nieco mniejszym błędem (tabela 4.21). Niestety, żadne z tych rozwiązań nie odpowiadało klasom na jakie podzielony był zbiór danych (tabela 4.22). Oba algorytmy zwracały dwie grupy o w przybliżeniu takim samym rozkładzie przykładów należących do pierwszej i do drugiej klasy.

ecoli

Zbiór *ecoli* analizowałem w analogiczny sposób jak poprzednie, usuwając atrybut klasy, a następnie porównując wyniki grupowania z klasami w zbiorze danych.

Grupowanie algorytmem MVC z wariancją, dla której algorytm zwraca 6 grup (przedstawione w tabeli 4.24) dosyć dobrze odpowiadają klasom na jakie podzielony był zbiór danych. Zakładając, że grupa odpowiada klasie, której najwięcej elementów do danej grupy zostało włączonych (np. w tabeli 4.24 grupa II odpowiada klasie *im*), jedynie 23% przykładów było źle zaklasyfikowanych. Najlepszy rezultat algorytmu k-środków to 25% przykładów źle zaklasyfikowanych. Zbiór *ecoli* był używany do testowania algorytmów klasyfikacji. Podobny błąd dawało użycie binarnych drzew decyzyjnych, klasyfikatora bayesowskiego i ustrukturalizowanego modelu probabilistycznego ([5], nagłówek pliku), a należy pamiętać, że metody te dysponują informacją o klasie.

Zbiór próbowałem również dzielić na dwie grupy, używając MVC z wariancją $\sigma^2_{max} = 16$. Uzyskane grupowanie (tabela 4.25) dzieli zbiór danych, do jednej z grup zaliczając większość przykładów z klas *cp*, *pp* i *om*, do drugiej *im*, *imU*. Pozostałe klasy podzielone zostały w przybliżeniu równo. Co ciekawe, w innych powtórzeniach algorytmu dawał on rozwiązania z podobnym błędem, ale innym podziałem przykładów na grupy, przy czym zazwyczaj w jednej z nich znajdowały się przykłady z klas *cp* i *pp*, a w drugiej *im* i *imU*. Wynik działania algorytmu k-środków miał mniejszy błąd, lecz wyglądał bardzo podobnie.

glass

Kolejnym analizowanym zbiorem danych jest zbiór *glass*. Algorytm MVC dla tego zbioru zachowywał się bardzo niestabilnie, co można odczytać z wykresu tendencji klastrów przedstawionego w następnym rozdziale. Dla wariancji $\sigma^2_{max} = 0.149$ (najlepsze plateau) MVC dzielił zbiór danych na dwie grupy, przy czym do jednej z nich przypisał jedynie dwa przykłady z piątej klasy. Jeden z możliwych wyników dla wariancji $\sigma^2_{max} = 0.12$ przedstawiony jest w tabeli 4.26. Również w tym przypadku nie widać jakiejś szczególnej zależności pomiędzy

Tabela 4.23: Zbiór *ecoli* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 6, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.08529371050366849$	3	6	101
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.12808694342058052$	37	2	38
MVC	$\sigma_{max}^2 = 0.11, \delta = 10^{-14}$, $J_e^* = 0.09376398285418744$	100	6	10086
MVC	$\sigma_{max}^2 = 0.16, \delta = 10^{-14}$, $J_e^* = 0.12824114688926053$	55	2	11388

Tabela 4.24: Wynik grupowania dla zbioru *ecoli* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 6 grup.

klasa	liczba przykładów przypisanych do					
	grupy I	grupy II	grupy III	grupy IV	grupy V	grupy VI
cp	137	0	0	0	6	0
im	8	66	0	1	2	0
pp	3	1	0	0	48	0
imU	1	33	0	1	0	0
om	0	0	0	0	19	1
omL	0	0	0	0	0	5
imL	0	0	1	1	0	0
imS	0	1	0	0	1	0

Tabela 4.25: Wynik grupowania dla zbioru *ecoli* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 2 grupy.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
cp	143	0
im	10	67
pp	50	2
imU	1	34
om	18	2
omL	3	2
imL	1	1
imS	1	1

Tabela 4.26: Wynik grupowania dla zbioru *glass* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 3 grupy.

klasa	liczba przykładów przypisanych do		
	grupy I	grupy II	grupy III
1	0	0	70
2	0	11	65
3	0	0	17
4	0	0	0
5	2	8	3
6	0	5	4
7	0	25	4

klasami a grupami. Rezultat podziału zbioru algorytmem k-środków bardziej odpowiada klasom w zbiorze danych. Przy podziale na dwa klastry (tabela 4.27), jeden z nich zawiera większość przykładów z trzech pierwszych grup, drugi zaś z kolejnych trzech, co odpowiada w przybliżeniu podziałowi na szkło okienne i inne (dwie klasy najwyższej hierarchii). Podział na trzy klastry jest podobny do zaproponowanego przez MVC.

Tabela 4.27: Wynik grupowania algorytmem k-środków dla zbioru *glass* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 3 grupy.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
1	70	0
2	65	11
3	17	0
4	0	0
5	3	10
6	5	4
7	4	25

pima

Grupy proponowane dla zbioru *pima* przez algorytm MVC dla $\sigma^2_{max} = 0.145$ (tabela 4.29) nie dzielą przykładów na diabetyków i zdrowych (błąd klasyfikacji wynosi około 33%). Ciekawa jest natomiast analiza środków dwóch zaproponowanych przez algorytm klastrów. Środkiem pierwszej grupy jest przykład opisujący kobietę po siedmiokrotnej ciąży, mającej 46 lat i o wyższym poziomie glukozy i ciśnieniu krwi. Środkiem drugiej grupy jest przykład opisujący kobietę młodszą (26 lat) i mającą za sobą dwie ciąże. Wartości dwóch atrybutów (bmi, pedigree) w obu grupach są bardzo podobne. Wynik grupowania algorytmem k-środków jest podobnie nieużyteczny w sensie podziału zbioru danych na zadane klasy (błąd klasyfikacji 33%) i o podobnych środkach klastrów. Porównanie zbieżności i czasu wykonania obu algorytmów znajduje się w tabeli 4.28.

segmentation

Kolejnym z analizowanych przeze mnie zbiorów danych był zbiór *segmentation*. Trzeba podkreślić, że żaden z testowanych przeze mnie algorytmów nie jest algorytmem segmentacji obrazu, bo nie uwzględnia ograniczenia na łączność (*spatial connectivity*) przykładów zaliczonych do każdego klastra. MVC zmodyfikowany by uwzględniać to ograniczenie opisany jest w [23].

Tabela 4.28: Zbiór *pima* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.13014727901180237$	84	2	64
MVC	$\sigma^2_{max} = 0.145, \delta = 10^{-14}$, $J_e = 0.12964734537043165$	1	2	50537

Tabela 4.29: Wynik grupowania dla zbioru *pima* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 2 grupy.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
0	126	374
1	135	133

Wyniki grupowania tego zbioru dla obu algorytmów i różnej liczby klastrów przedstawiam w tabelach 4.31, 4.32 i 4.33. Przy podziale na dwie grupy, rezultaty działania tych algorytmów są podobne, oba tworzą grupę, do której przypisane są wszystkie przykłady z regionu odpowiadającego na rysunku niebu i kilka przykładów z regionu beton (choć MVC popełnia mniejszy błąd). Przy trzech klastrach MVC dzieli zbiór na grupy odpowiadające klasom: {beton, ścieżka}, {niebo} i reszta, przy czym 13 przykładów zostało źle przypisanych. Algorytm k-środków działa podobnie, choć o jeden przykład trafniej. Przy czterech klastrach algorytmy dzielą zbiór na grupy odpowiadające klasom: beton, ścieżka; trawa; niebo; pozostałe. Oba algorytmy nieprawidłowo klasyfikują szesnaście przykładów. Porównania czasu działania algorytmów i procentu prób zakończonych sukcesem znajdują się w tabeli 4.30.

wine

Zbiór *wine* próbowałem dzielić na trzy i dwie grupy. Przy podziale na dwie grupy oba algorytmy znajdowały to samo optimum (tabela 4.36), choć algorytm k-środków dochodził do niego rzadziej (tabela 4.34). Klastry dosyć dobrze odpo-

Tabela 4.30: Zbiór *segmentation* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.15804115033091892$	12	2	20
k-środki	$M = 3, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.13488611025635905$	9	3	39
k-środki	$M = 4, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.11577812220344764$	66	4	65
MVC	$\sigma_{max}^2 = 0.18, \delta = 10^{-14}$, $J_e = 0.15802785581546358$	62	2	5531
MVC	$\sigma_{max}^2 = 0.16, \delta = 10^{-14}$, $J_e = 0.13486678726762585$	64	3	6412
MVC	$\sigma_{max}^2 = 0.14, \delta = 10^{-14}$, $J_e = 0.11570241083664505$	95	4	6535

Tabela 4.31: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma_{max}^2 = 0.18$) dla zbioru *segmentation* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 2 grupy.

klasa	k-środki: liczba przykładów przypisanych do		MVC: liczba przykładów przypisanych do	
	grupy I	grupy II	grupy I	grupy II
BRICKFACE	30	0	0	30
SKY	0	30	30	0
FOLIAGE	30	0	0	30
CEMENT	26	4	3	27
WINDOW	30	0	0	30
PATH	30	0	0	30
GRASS	30	0	0	30

Tabela 4.32: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma^2_{max} = 0.16$) dla zbioru *segmentation* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 3 grupy.

klasa	k-środki: liczba przykładów przypisanych do			MVC: liczba przykładów przypisanych do		
	grupy I	grupy II	grupy III	grupy I	grupy II	grupy III
BRICKFACE	1	29	0	0	1	29
SKY	0	0	30	30	0	0
FOLIAGE	3	27	0	0	3	27
CEMENT	25	5	0	0	25	5
WINDOW	2	28	0	0	3	27
PATH	29	1	0	0	29	1
GRASS	0	30	0	0	0	30

Tabela 4.33: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma^2_{max} = 0.14$) dla zbioru *segmentation* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 4 grupy.

klasa	k-środki: liczba przykładów przypisanych do				MVC: liczba przykładów przypisanych do			
	grupy I	grupy II	grupy III	grupy IV	grupy I	grupy II	grupy III	grupy IV
BRICKFACE	0	0	30	0	30	0	0	0
SKY	0	30	0	0	0	30	0	0
FOLIAGE	0	0	27	3	27	0	3	0
CEMENT	0	0	5	25	4	0	26	0
WINDOW	5	0	23	2	22	0	3	5
PATH	0	0	1	29	1	0	29	0
GRASS	30	0	0	0	0	0	0	30

Tabela 4.34: Zbiór *wine* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 2, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.16106504416278417$	32	2	26
k-środki	$M = 3, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.13820088182719859$	4	3	21
MVC	$\sigma_{max}^2 = 0.18, \delta = 10^{-14}$, $J_e = 0.16106504416278427$	100	2	4528
MVC	$\sigma_{max}^2 = 0.16, \delta = 10^{-14}$, $J_e = 0.13820088182719859$	3	3	4522

Tabela 4.35: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma_{max}^2 = 0.16$) dla zbioru *wine* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 3 grupy.

klasa	liczba przykładów przypisanych do		
	grupy I	grupy II	grupy III
1	0	0	59
2	4	65	2
3	48	0	0

wiadają klasom w zbiorze danych, do pierwszego klastra zaliczana jest cała klasa pierwsza i większość klasy drugiej, do drugiego cała klasa trzecia i część klasy drugiej. Przy podziale na 3 klastry oba algorytmy rzadko znajdowały optimum – k-środki w czterech próbach na 100, MVC w trzech. Optymalne grupowanie dobrze dzieli zbiór danych (tabela 4.35), klasy odpowiadają grupom i jedynie 6 przykładów jest źle zaklasyfikowanych.

yeast

Zbiór *yeast* próbowałem dzielić na sześć i osiem grup. Oba algorytmy zachowywały się bardzo niestabilnie, rzadko dając optymalny wynik (tabela 4.38). W obu przypadkach i dla obu algorytmów proponowane grupowania nie odpowiada-

Tabela 4.36: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma^2_{max} = 0.18$) dla zbioru *wine* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 2 grupy.

klasa	liczba przykładów przypisanych do	
	grupy I	grupy II
1	59	0
2	50	21
3	0	48

ło klasom w zbiorze danych. Optymalne wyniki zwrócone przez oba algorytmy dość znacznie różnią się. Dla sześciu klastrów (tabela 4.39) MVC tworzy dwie duże grupy, zawierające w sumie 98% przykładów. W pierwszej z grup wyraźną przewagę mają przykłady z klas *CYT*, *NUC*, *ME3*, w drugiej *MIT*, *ME2*, *ME1*, *EXC*, przy czym prawie wszystkie przykłady z tych trzech ostatnich klas trafiły do drugiej grupy. Wszystkie przykłady z klasy *ERL* trafiły do oddzielnego klastra. Algorytm k-środków tworzy z kolei klastry o dość podobnej wielkości, w których przykłady z najliczniejszych klas (*CYT*, *NUC*, *MIT*) rozkładane są dość równomiernie. Ponadto jeden z klastrów zawiera praktycznie wszystkie przykłady z klas *ME2*, *ME1*, *EXC* i *ERL*. Przy podziale na osiem grup (tabela 4.37) MVC tworzy trzy duże klastry, przy czym klaster I ma równomierny rozkład prawie wszystkich kategorii (ale z drugiej strony trafiają tam prawie wszystkie przykłady z klas *ME2*, *ME1*, *EXC*), w klastrze VI przeważają kategorie *CYT*, *NUC* i *ME3*, a w klastrze V kategoria *MIT*. Wszystkie przykłady z kategorii *ERL* trafiają do osobnego klastra. Pozostałe klastry zawierają po kilka przykładów z różnych grup. Algorytm k-środków klasy *CYT* i *NUC* dzieli prawie równomiernie na 6 grup, poza tym do grupy VI trafia większość przykładów z klasy *ME3*. Grupa IV zawiera prawie wszystkie przykłady z klas *ME2*, *ME1*, *EXC*. Podsumowując, można stwierdzić, że grupowanie zbioru *yeast* algorytmem MVC zachowuje więcej informacji i mniej miesza klasy, niż grupowanie algorytmem k-środków.

Tabela 4.37: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma^2_{max} = 0.085$) dla zbioru *yeast* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 8 grup.

klasa	k-środki: liczba przykładów przypisanych do grupy								MVC: liczba przykładów przypisanych do grupy							
	I	II	III	IV	V	VI	VII	VIII	I	II	III	IV	V	VI	VII	VIII
CYT	4	1	25	10	170	16	39	198	62	4	0	1	48	348	0	0
NUC	3	0	79	7	160	24	30	126	48	2	0	0	41	337	0	1
MIT	0	3	2	20	24	7	137	51	49	0	2	1	144	48	0	0
ME3	1	0	7	12	6	126	2	9	28	1	0	0	7	127	0	0
ME2	1	0	0	38	3	4	0	5	43	1	0	0	1	6	0	0
ME1	0	0	0	44	0	0	0	0	43	0	0	0	1	0	0	0
EXC	0	0	0	31	2	1	0	1	32	0	0	0	0	3	0	0
VAC	0	0	0	8	5	7	2	8	11	0	0	0	2	17	0	0
POX	0	11	0	2	2	1	1	3	2	0	9	2	1	6	0	0
ERL	5	0	0	0	0	0	0	0	0	0	0	0	0	0	5	0

Tabela 4.38: Zbiór *yeast* ze znormalizowaną miarą odległości: porównanie działania algorytmów k-środków i MVC.

algorytm	parametr	liczba trafień	liczba klastrów	czas całości [ms]
k-środki	$M = 6, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.07224133362137658$	1	6	659
k-środki	$M = 8, \delta = 10^{-14}$, $J_e^{kmeans*} = 0.06747095133633553$	1	8	1004
MVC	$\sigma^2_{max} = 0.095, \delta = 10^{-14}$, $J_e = 0.08599478519693154$	32	6	361442
MVC	$\sigma^2_{max} = 0.085, \delta = 10^{-14}$, $J_e = 0.07883227990755116$	2	8	186028

Tabela 4.39: Wynik grupowania algorytmem k-środków i algorytmem MVC ($\sigma^2_{max} = 0.095$) dla zbioru *yeast* ze znormalizowaną miarą odległości: liczba przykładów z poszczególnych klas przypisana do poszczególnych grup przy podziale na 6 grup.

klasa	k-środki: liczba przykładów przypisanych do grupy						MVC: liczba przykładów przypisanych do grupy					
	I	II	III	IV	V	VI	I	II	III	IV	V	VI
CYT	15	41	170	11	201	25	1	357	4	0	0	101
NUC	23	31	161	8	126	80	0	351	3	0	0	75
MIT	7	141	23	21	50	2	1	70	0	2	0	171
ME3	125	2	6	12	11	7	0	125	1	0	0	37
ME2	4	0	3	38	6	0	0	5	1	0	0	45
ME1	0	0	0	44	0	0	0	0	0	0	0	44
EXC	1	0	2	31	1	0	0	3	0	0	0	32
VAC	7	2	5	8	8	0	0	17	0	0	0	13
POX	2	2	8	4	4	0	2	6	0	9	0	3
ERL	0	0	0	5	0	0	0	0	0	0	5	0

4.5.3 Analiza wyników

Porównując czas działania algorytmu MVC i algorytmu k-środków wyraźnie widać, że MVC działa w najlepszym przypadku (dla dużych zbiorów: D31, *yeast*) dwa rzędy wielkości dłużej. Algorytmu k-środków prawie nigdy nie wykonuje się tylko raz, również istnieją znacznie lepsze algorytmy bazujące na k-środkach, ale wydajniej szukające optimum. Można zatem postawić tezę, że gdyby k-środkom dać tyle samo czasu, ile używał MVC, najlepszy wynik nie różniłby się.

Okazuje się to jednak nieprawdą, bo np. dla zbioru O3 k-środki znajdowały rozwiązanie o mniejszym błędzie J_e niż MVC (co zasadniczo nie jest wadą k-środków), ale jednocześnie mniej pasujące do zbioru danych. Ponadto dla zbioru D31, na 100 uruchomień, k-środki nie były w stanie znaleźć rozwiązania lepszego pod względem J_e niż MVC, z czego wynika, że mało prawdopodobne jest, by znalazły go w większej liczbie prób.

Inną kwestią jest to, że wyniki czasu działania algorytmu MVC w eksperymentach przeprowadzonych przeze mnie różnią się od wyników opublikowanych w [22], gdzie algorytm MVC był od 30 (dla zbioru O3) do 3 razy wolniejszy

od k-środków. Różnica ta wynika prawdopodobnie z różnicy pomiędzy moją, a oryginalną, implementacją algorytmu. Mój program (opisany w dodatku A) jest skonstruowany modułowo, z myślą o możliwości przetestowania jak największej liczby możliwych wariantów algorytmu. Jestem pewien, że implementacja, która kładzie nacisk na wydajność dałaby znacznie lepsze rezultaty czasowe.

By zmniejszyć różnice czasowe, można by również ustalić lepsze wartości parametrów MVC, a szczególnie zmniejszyć liczbę iteracji E_{max}.

Dla większości niewygenerowanych zbiorów danych wyniki podziału zbioru na grupy są mało skorelowane z kategoriami obecnymi w zbiorze. Takie wyniki niekoniecznie muszą świadczyć o złym działaniu algorytmów. Algorytmy grupowania dzielą zbiór danych na grupy, w których są podobne wartości atrybutów. Można znaleźć wiele etykietowanych zbiorów danych, w których klasa (etykieta) przykładu nie zależy od wszystkich atrybutów, a w skrajnych przypadkach zależy tylko od jednego z nich. Algorytmy klasyfikacji z takimi zbiorami powinny sobie poradzić, natomiast algorytmy grupowania, nie mając „zwrotnej" informacji o klasach, nie będą w stanie zwrócić grupowania skorelowanego z klasami zbioru danych.

Porównując wyniki obu algorytmów na wygenerowanych i rzeczywistych zbiorach danych trzeba stwierdzić, że przewagę MVC nad k-środkami widać tylko na zbiorach wygenerowanych. We wszystkich zbiorach rzeczywistych oba algorytmy radziły sobie bardzo podobnie, jedyną różnicą był procent optymalnych wyników – k-środki zazwyczaj optymalny wynik osiągały rzadziej, ale biorąc pod uwagę dużo niższy czas wykonania tego algorytmu, nie jest to znaczącą wadą (analizę k-środkami zawsze można kilkukrotnie powtórzyć i wziąść pod uwagę najlepszy wynik).

Definicja przestrzeni rozwiązań uwzględniająca wariancję klastra jest dobrym pomysłem i sprawdza się zwłaszcza w przypadku syntetycznych zbiorów danych. Dla zbiorów O3 i D31 MVC bez trudu znajdował właściwe, pasujące do struktury zbioru, rozwiązanie, co dla k-środków było praktycznie niemożliwe.

4.6 Badanie analizy tendencji klastrów przy użyciu MVC i IMVC

W tym rozdziale zajmę się problemem znalezienia odpowiedniej dla algorytmu wartości parametru σ^2_{max} poprzez wyznaczanie tendencji grupowania przy użyciu algorytmów MVC (opisany w rozdziale 3.3) i IMVC (opisany w rozdziale 3.4). Autorzy [22] stwierdzili, że grupowanie dla σ^2_{max} jest sensowne, gdy plateau, do którego należy σ^2_{max} ma siłę większą niż 2. Zweryfikuję to rozumowanie, tworząc zbiory danych o wartościach atrybutów wygenerowanych rozkładem jednorodnym $U(0,1)$, a następnie próbując grupować tak powstałe zbiory. Następnie porównam tendencję grupowania wyznaczaną poprzez powtarzanie algorytmu MVC dla każdej możliwej wartości wariancji, oraz poprzez wykonanie algorytmu przyrostowego IMVC.

4.6.1 Analiza zbiorów jednorodnych

W celu doświadczalnego sprawdzenia, czy plateau o sile większej niż 2 nie pojawiają się przypadkowo, powtórzyłem eksperyment opisany w [22]. Wygenerowanych zostało po 1000 zbiorów danych o 5, 10, 20, 40 i 80 przykładach. Zbiory te wygenerowane zostały dwuwymiarowym rozkładem jednorodnym $O(0,1)$, zatem na wykresie przykłady zawierały się w kwadracie jednostkowym. Następnie dla każdego zbioru danych została wyznaczona tendencja klastrów i zapisane największe plateau. W kolejnym kroku, oddzielnie dla zbiorów danych o różnych wielkościach, wyznaczyłem dystrybuantę siły największego plateau. Dla pewnej siły plateau S, wartość dystrybuanty w tym punkcie jest procentem zbiorów danych, w których plateau o największej sile ma siłę mniejszą niż S. Dystrybuanty te przedstawione są na rysunku 4.13.

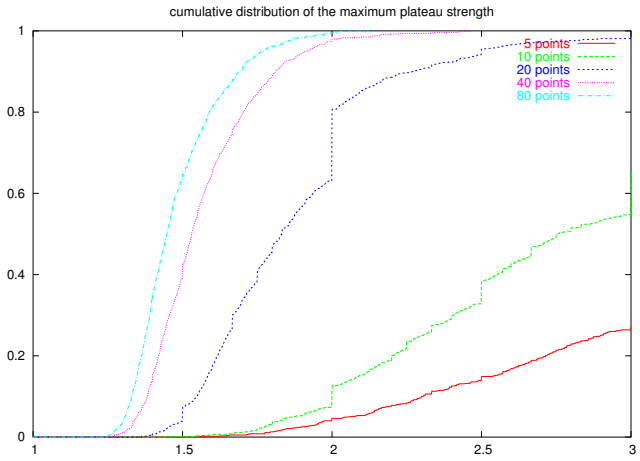
Właściwy dobór początkowej wariancji σ^2_{max0} i kroku $\Delta\sigma^2_{max}$ jest dosyć trudny. Jeżeli wartość początkowa σ^2_{max0} jest zbyt duża, ryzykujemy niezauważenie małych plateau, odpowiadających grupom złożonym z kilku przykładów. Podobnie, jeżeli $\Delta\sigma^2_{max}$ jest zbyt duże, możemy nad tymi niewielkimi plateau „przeskoczyć”. W celu ścisłego zbadania tendencji należałoby ustalić początkową warian-

cję $\sigma^2_{max0} = 0$, tak by zaczynać od liczby klastrów równej liczbie przykładów. Pierwsze plateau zaczynałoby się od wariancji pozwalającej na scalenie najbliższych dwóch klastrów (dlatego, że siła każdego plateau zawierającego $\sigma^2_{max0} = 0$ dążyłaby do nieskończoności). W podobny sposób należałoby wyznaczyć wartość kroku $\Delta\sigma^2_{max}$. Podczas badania tendencji, w każdym przypadku, w którym w dwóch kolejnych krokach ubyło ponad jeden klaster, należałoby zmniejszyć $\Delta\sigma^2_{max}$ w celu wyznaczenia plateau pomiędzy tymi dwoma punktami.

Nie zdecydowałem się na takie postępowanie z kilku powodów. Po pierwsze, wyznaczając ściśle wartości σ^2_{max0} i $\Delta\sigma^2_{max}$, otrzymalibyśmy dosyć dużą liczbę plateau odpowiadających klastrom zawierającym mało przykładów. Takie klastry zazwyczaj nic nie mówią o zbiorze danych, ponieważ spodziewamy się znalezienia grup co najmniej kilkuelementowych. Po drugie, po zmniejszeniu kroku $\Delta\sigma^2_{max}$, znacznie rośnie koszt obliczeniowy wyznaczenia analizy, co spowodowałoby znaczne wydłużenie czasu działania algorytmu. Trzecim powodem są trudności z wyznaczeniem kroku $\Delta\sigma^2_{max}$ w opisany przeze mnie sposób. Dla niektórych zbiorów danych zwiększenie wariancji w naturalny sposób powoduje spadek liczby klastrów o 2. Czerwona linia na rysunku 4.16 pokazuje tendencję wyznaczoną dla zbioru O3. Dla $\sigma^2_{max} = 113$ występuje spadek liczby klastrów o 2.

Eksperymentalnie ustaliłem wartość wariancji początkowej na $\sigma^2_{max0} = 0.001$, a wartość kroku na $\Delta\sigma^2_{max} = 0.001$. Badanie tendencji kończyło się, gdy dla pewnej σ^2_{maxend} w zbiorze był jeden klaster. Ponieważ przykłady pochodziły z kwadratu jednostkowego, średnia odległość przykładów od środka klastra jest mniejsza lub równa połowie przekątnej kwadratu, czyli $\sigma^2_{maxend} \leq \frac{\sqrt{2}}{2}$.

Rysunek 4.13 przedstawia wynik eksperymentu – dystrybuanty siły plateau dla różnej liczby przykładów w zbiorze danych. Wykres dystrybuanty dla większej liczby przykładów przesunięty jest na lewo od wykresu dystrybuanty dla mniejszej liczby przykładów, zatem im więcej przykładów w zbiorze, tym najlepsze plateau mają mniejszą siłę. Dla zbiorów o 40 i 80 przykładach sporadycznie zdarzało się, by najlepsze plateau było znaczące. Dla zbiorów o 80 przykładach wartość dystrybuanty w punkcie 2.0 to 0.996 (w punkcie 1.999 – 0.994), zatem na 1000 losowych zbiorów danych jedynie w 6 obecne były znaczące plateau.



Rysunek 4.13: Wykres dystrybuanty siły największego plateau znalezionego w zbiorze danych dla losowych, dwuwymiarowych zbiorów danych o różnych wielkościach.

Dla zbiorów o 40 przykładach wartości te wynoszą: w punkcie 1.999 – 0.973, w punkcie 2.0 – 0.980, więc w tym przypadku liczba znaczących plateau jest odpowiednio większa i wynosi 27.

Nieciągłość dystrybuanty dla zbioru o 20 przykładach w punkcie 2.0 świadczy o dużej liczbie plateau o sile dokładnie równej 2.0. Ponieważ siła plateau to stosunek wariancji końca plateau do wariancji początku plateau $S = \frac{\sigma_B^2}{\sigma_A^2}$, plateau takie kończy się dokładnie dla $\sigma_B^2 = 2 * \sigma_A^2$. Oczywiście w losowym zbiorze danych taki wynik jest mało prawdopodobny, a opisane zjawisko świadczy o tym, że krok $\Delta\sigma_{max}^2$ był zbyt duży, a rzeczywista siła plateau jest nieco większa niż 2.

Z przytoczonych rezultatów wynika, że w przypadkowych zbiorach danych znaczące plateau pojawiają się bardzo rzadko, przy czym im większy jest zbiór danych, tym mniejsze jest prawdopodobieństwo, że takie plateau wystąpi. Można stwierdzić, że siła plateau jest dobrą miarą stopnia ustrukturalizowania zbioru. Jednocześnie należy się spodziewać, że siła ta będzie rosła dla zbiorów o bardziej widocznej strukturze. Dla zbiorów o dużej liczbie przykładów siła najlepszych plateau powinna się w istotny sposób wyróżniać od siły pozostałych.

4.6.2 Porównanie MVC i IMVC

W tym rozdziale porównana zostanie tendencja grupowania wyznaczona przez powtarzanie algorytmu MVC oraz poprzez wykonanie algorytmu przyrostowego IMVC.

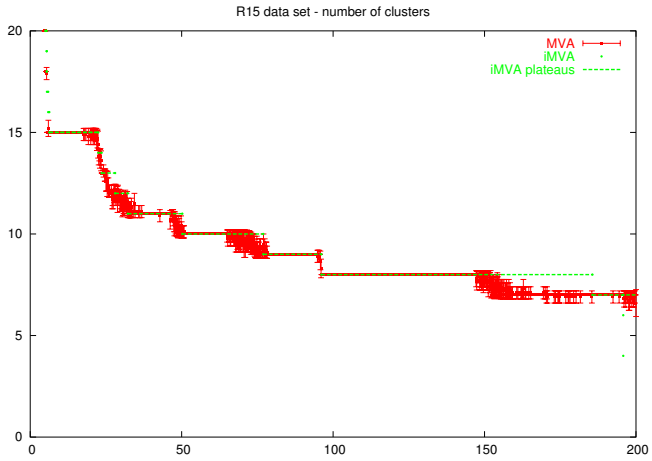
Wyznaczanie tendencji algorytmem MVC powtórzyłem dziesięciokrotnie. Na wykresach czerwona linia odpowiada średniej z wyników tych dziesięciu powtórzeń. Za pomocą pionowych słupków błędów (*y errorbars*) oznaczone zostały odchylenia standardowe mierzone jako odchylenie od średniej z dziesięciu pomiarów, wyznaczane niezależnie dla każdej wartości wariancji σ^2_{max}.

Plateau na wykresach odpowiadają rejonom w przybliżeniu stałym, otoczonym oscylacjami.

Okrągłe, zielone punkty odpowiadają wartościom σ^2_{max}, w których jako część IMVC uruchamiany był MVC. Łącząca je zielona linia wyznacza stabilne plateau proponowane przez IMVC.

R15

Uruchamiając MVC dla zbioru R15, przyjąłem wielkość kroku o jaki zwiększana jest wariancja $\Delta\sigma^2_{max} = 0.2$, a badanie tendencji zacząłem od $\sigma^2_{max0} = 2.0$, co dawało około 60 grup. Wyznaczając tendencję grupowania tego zbioru (przedstawioną na rysunkach 4.14 i 4.15), oba algorytmy znalazły znaczące plateau ($[6.23 \ldots 22.47]$) o sile 3.60, odpowiadające 15 grupom. Kolejne plateau znalezione przez IMVC ($[96.14 \ldots 185.62]$) ma siłę 1.93 i odpowiada 8 grupom. W tym przypadku plateau znalezione przez MVC jest nieco mniejsze ($[96.2 \ldots 147.0]$), ale nadal jest drugim co do siły. Kształt wykresów jest podobny, przy czym plateau znajdowane przez IMVC nigdy nie są krótsze, niż plateau znalezione dla MVC. Różnice widać szczególnie na krańcach plateau odpowiadających 9 i 10 grupom oraz w obszarze niestabilności odpowiadającemu przejściu pomiędzy 15 a 12 klastrami. Na wykresie liczby klastrów średnia obliczona dla 10 uruchomień MVC szybko, ale łagodnie spada, tworząc „zaokrąglenie" krawędzi wyznaczonej przez IMVC.



Rysunek 4.14: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru R15.

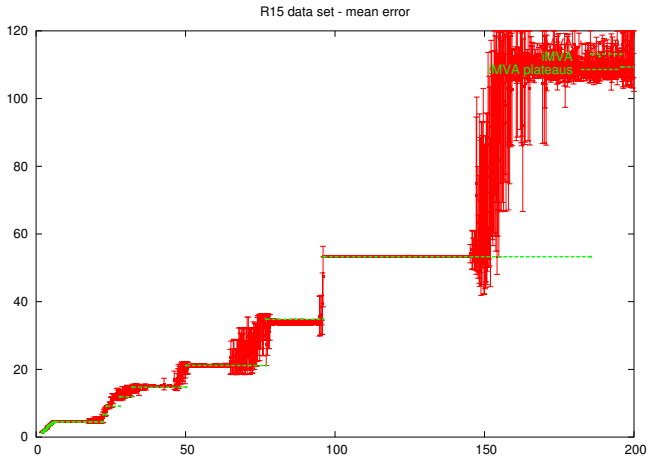
Dla wartości σ^2_{max} mniejszych niż 6, wykresy obu tendencji szybko i gwałtownie spadają.

Należy również zauważyć, że plateau na wykresach liczby grup w zależności od wariancji mają dość dobrze odpowiadające sobie plateau na wykresach błędu J_e. Zwiększenie wariancji σ^2_{max} zazwyczaj nie wpływa na zmianę wartości błędu, o ile nie zmienia się jednocześnie liczba klastrów. Odstępstwa od tej reguły zachodzą dla wartości σ^2_{max} bliskich krańcom plateau. W takich przypadkach zwiększona wariancja może mieć wpływ na inne łączenie się małych klastrów w pierwszych krokach działania algorytmu. Jeśli szybko sformują się duże klastry, perturbacja, przepisująca za każdą iteracją MVC co najwyżej 2 przykłady pomiędzy dwoma klastrami, nie będzie w stanie znaleźć lepszego pod względem błędu J_e rozwiązania, co wymagałoby jednoczesnego przepisania kilku przykładów.

O3

Badając tendencję grupowania dla zbioru O3, ustaliłem dla algorytmu MVC krok $\Delta\sigma^2_{max} = 0.2$ i wariancję początkową $\sigma^2_{max0} = 2.0$, co odpowiadało około 32 grupom. Wykresy tendencji przedstawione są na rysunkach 4.16 i 4.17.

Oba algorytmy znalazły to samo znaczące plateau odpowiadające 6 grupom



Rysunek 4.15: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru R15.

dla wariancji $[21.12 \ldots 82.61]$, o sile 3.91. Co ciekawe, na wykresie MVC widać, że dla wariancji z tego przedziału, algorytm ten nie zawsze dawał rozwiązanie o 6 grupach. Odpowiada to wynikom eksperymentu (rozdział 4.5.1), w którym MVC był powtarzany stukrotnie dla tej samej wartości σ^2_{max} – w zbiorze O3 na 100 powtórzeń jedno zakończyło się podaniem innego wyniku. W przedziale odpowiadającym temu plateau MVC powtarzany jest około 500 razy. Średnia liczba klastrów różni się od wartości 6 w jedenastu punktach (przy czym za każdym razem jeden wynik z dziesięciu odbiegał od średniej), co daje błąd algorytmu około 2%.

Dla wyższych wartości wariancji (z przedziału $(83, 127)$) MVC wpada w dosyć poważne niestabilności (opisane w rozdziale 4.2). Ciekawie w tym przedziale zachowuje się również IMVC – wartości σ^2_{max}, dla jakich IMVC włączał MVC z tego przedziału przedstawiam w tabeli 4.40.

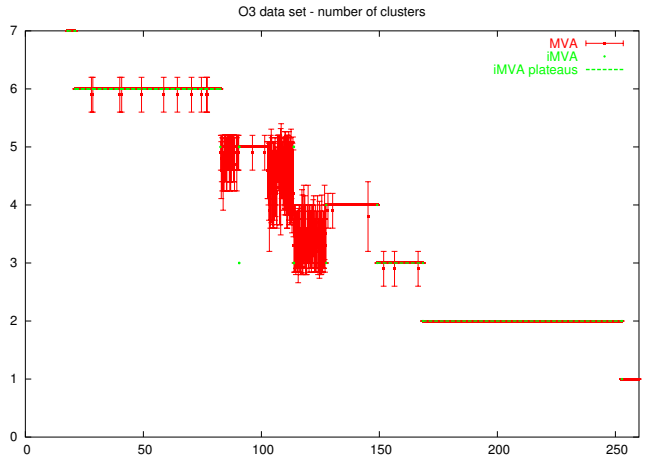
W przedziale tym widać dwa zachowania uwzględnione przy projektowaniu IMVC. Po pierwsze, dla wariancji $\sigma^2_{max} = 90.58$ zachodzi przypadek, w którym uruchamiane jest MVC, ale daje ono po stabilizacji taką samą liczbę grup, jaka była dla poprzedniej wartości $\sigma^2_{max} = 82.61$. IMVC oznacza taką wariancję jako niestabilną i kontynuuje swoje działanie. Kolejną wartością wariancji, dla której

Tabela 4.40: Fragment wyników algorytmu IMVC dla zbioru O3. Tabela przedstawia wartości σ^2_{max} odpowiadające początkom i końcom plateau znalezione przez algorytm.

początek plateau	koniec plateau	siła plateau
21.123278333333335	82.61678085327783	3.9111722882004263
82.61678085327783	90.58248944444439	niestabilne
90.58248944444439	113.77391841831431	niestabilne
113.77391841831431	127.50273882716048	niestabilne
127.50273882716048	149.06192499999992	niestabilne
149.06192499999992	168.47466583745924	1.1302327260127583

algorytm uruchamia MVC, jest $\sigma^2_{max} = 113.77$. Dla takiej wariancji MVC daje jako rozwiązanie 3 grupy (widać to również na wynikach badania tendencji grupowania dla algorytmu MVC, która dla tej wartości wariancji oscyluje w okolicach 3.3). Zmniejszenie liczby klastrów o dwa z reguły oznacza, że zostało przeoczone plateau odpowiadające czterem klastrom. IMVC próbuje je znaleźć, uruchamiając MVC jeszcze raz dla $\sigma^2_{max} = 90.58$, ale zaczynając od 3 klastrów. Niestety MVC zwraca rozwiązanie takie same, jak poprzednio, więc IMVC wpada w pętlę. W kolejnej iteracji MVC daje jednak jako wynik grupowanie o 5 klastrach (możliwe, choć mniej prawdopodobne). IMVC oznacza tę wariancję jako niestabilną i kontynuuje działanie. Jednak stan jest na tyle niestabilny, że kolejna analizowana wariancja różni się od poprzedniej zaledwie o $3 * 10^{-15}$. Dla tej wariancji MVC znów daje jako rozwiązanie 3 klastry. IMVC uruchamia, w ramach poszukiwania plateau dla 4 klastrów, MVC z poprzednią wartością wariancji, które tym razem daje inne rozwiązanie o 3 klastrach. Algorytm nie sprawdza, czy podczas powrotu do poprzedniej wartości wariancji znalazł rozwiązanie o tej samej liczbie klastrów. Podobne wynik otrzymalibyśmy, gdyby pętla ta została po prostu przerwana. Kolejna wartość wariancji ($\sigma^2_{max} = 127.50$) jest oznaczana jako niestabilna, ponieważ liczba klastrów rośnie.

Ponieważ żadna z wartości wariancji $\sigma^2_{max} = 90.58$, $\sigma^2_{max} = 113.77$, $\sigma^2_{max} = 127.50$ nie została oznaczona jako stabilna, łącząca je linia nie może być interpretowana jako plateau. A zatem wyniki IMVC nie odbiegają od wyników MVC, ponieważ oba algorytmy pokazują w tym przedziale niestabilne zachowanie i nie



Rysunek 4.16: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru O3.

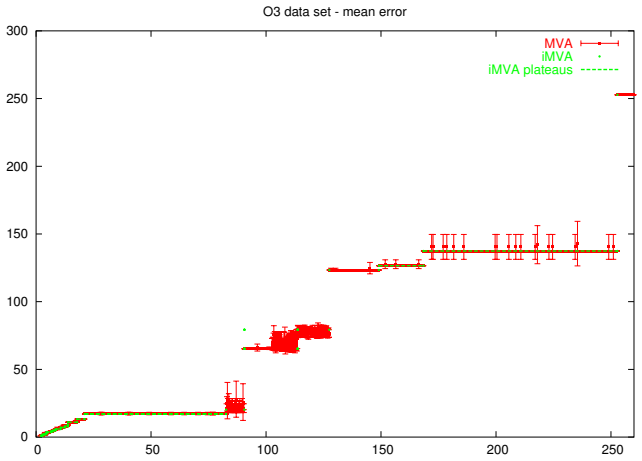
wskazują żadnego plateau. Istotne jest również, że IMVC bardzo dobrze wychodzi z obszaru niestabilności, ponieważ, zaczynając od $\sigma^2_{max} = 128$, wykresy tendencji grupowania dla obu metod są prawie identyczne.

D31

Analiza tendencji klastrów dla zbioru D31 algorytmem MVC była kosztowna obliczeniowo, dlatego też ustaliłem stosunkowo dużą wartość kroku, z jakim zwiększała się wariancja $\Delta\sigma^2_{max} = 0.005$. Wariancją początkową było $\sigma^2_{max0} = 0.005$. Wykresy tendencji przedstawione są na rysunkach 4.18 i 4.19

Również w tym zbiorze danych tak MVC jak i IMVC znalazły to samo plateau o największej sile 1.87 dla wariancji $[0.0033 \ldots 0.0063]$, odpowiadające 31 grupom.

Dla większych wartości wariancji σ^2_{max} MVC jest niestabilne, co odpowiada zwiększonym słupkom błędów. Podobnie jak w przypadku R15, IMVC jest bardziej „optymistyczny” i przedłuża każde plateau. Zaczynając od wariancji $\sigma^2_{max} = 0.002$, IMVC daje średnio jeden klaster więcej niż MVC, co w przybliżeniu mieści się w odchyleniu standardowym zmierzonym dla 10 niezależnych przebiegów MVC. Pomimo tych rozbieżności należy podkreślić, że żadne z tych plateau nie



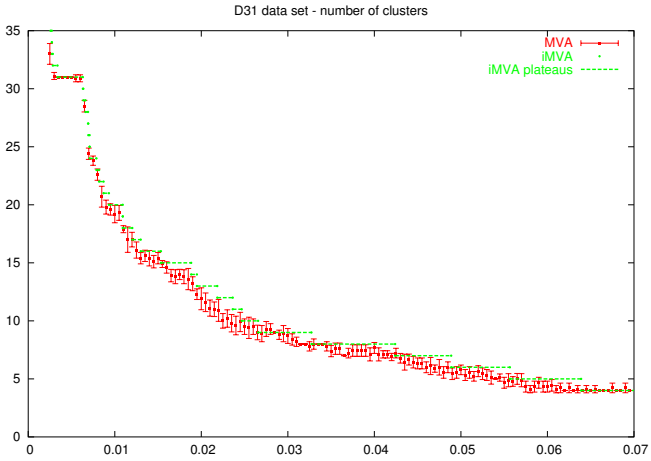
Rysunek 4.17: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru O3.

ma siły która odpowiadałaby sile plateau dla 31 grup. Trzy kolejne pod względem siły plateau odpowiadają czterem (siła 1.5257), trzem (siła 1.4124) i dwóm (siła 1.3607) grupom, które nie należą do tego obszaru niestabilności.

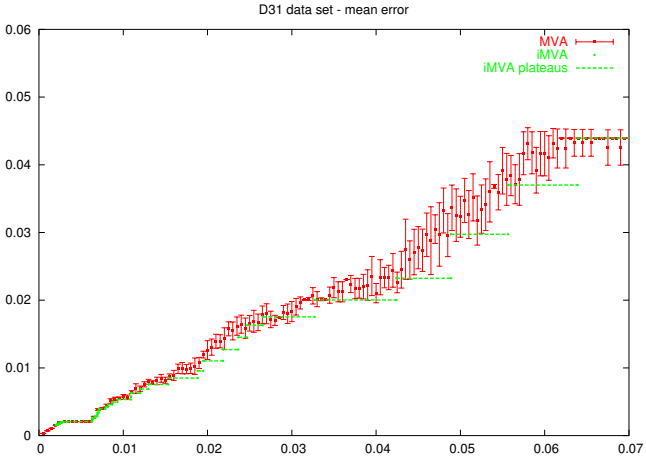
iris

Parametry algorytmu MVC, z jakimi analizowałem zbiór *iris*, to krok wariancji $\Delta\sigma^2_{max} = 0.05$ i $\sigma^2_{max0} = 0.1$, która to wartość odpowiadała 28 klastrom. Rezultaty działania algorytmu – wykresy tendencji przedstawione są na rysunkach 4.20 i 4.21.

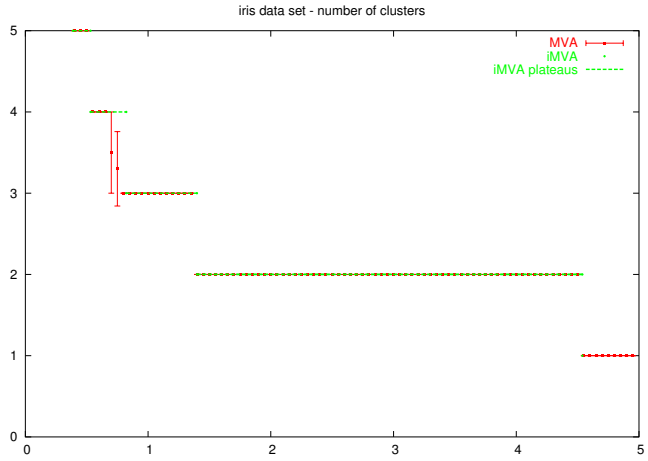
Znaczącym plateau o największej sile (3.25) znalezionym przez obydwa algorytmy jest $[1.40 \ldots 4.54]$. Wartości σ^2_{max} należące do tego plateau powodują podział zbioru danych na dwie grupy. Kolejnym pod względem siły (1.74) jest plateau dzielące zbiór *iris* na trzy grupy, nieodpowiadające niestety kategoriom w zbiorze. Szczegółowe omówienie wyników grupowania jest opisane w rozdziale 4.5.2. Wykresy IMVC i MVC są do siebie bardzo podobne.



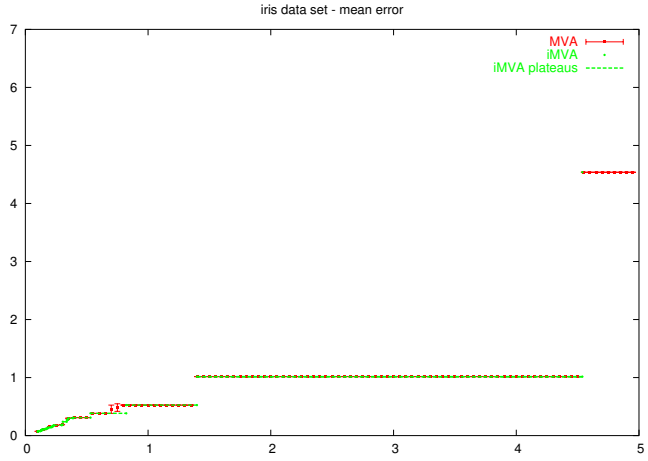
Rysunek 4.18: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru D31.



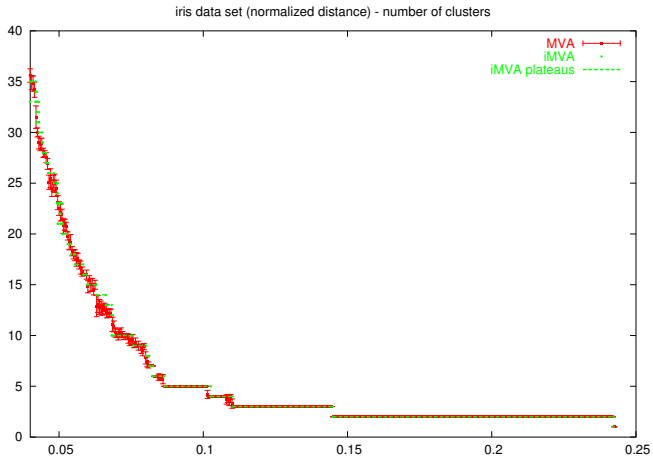
Rysunek 4.19: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru D31.



Rysunek 4.20: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *iris*.



Rysunek 4.21: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *iris*.



Rysunek 4.22: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru iris.

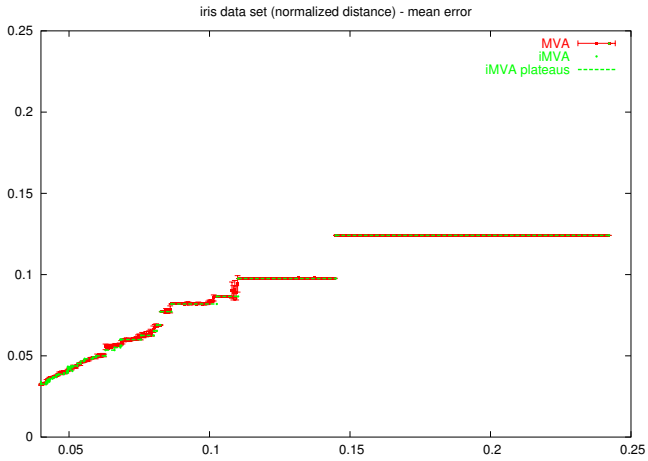
iris – znormalizowane

Zbiór *iris* analizowałem korzystając również ze znormalizowanej miary odległości. Parametry algorytmu MVC to: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.04$, która to wartość odpowiadała około 35 klastrom. Rezultaty działania algorytmu – wykresy tendencji przedstawione są na rysunkach 4.22 i 4.23.

Przedstawione wykresy są bardzo podobne do wykresów tendencji dla nieznormalizowanego zbioru *iris*. Podobnie jak tam, tendencja wyznaczona algorytmem IMVC ściśle pokrywa się z tendencją wyznaczoną MVC. Ważną różnicą jest zwężenie się wszystkich plateau. Najsilniejsze plateau w oryginalnym zbiorze *iris* miało siłę $S = 3.24$, w zbiorze znormalizowanym zaś – $S = 1.67$. Drugie plateau pod względem siły w zbiorze oryginalnym miało siłę $S = 1.74$, w zbiorze znormalizowanym – $S = 1.31$.

bupa

Analizę zbioru *bupa* algorytmem MVC przeprowadziłem z następującymi parametrami: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.06$, co odpowiadało około 40 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.24



Rysunek 4.23: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru iris.

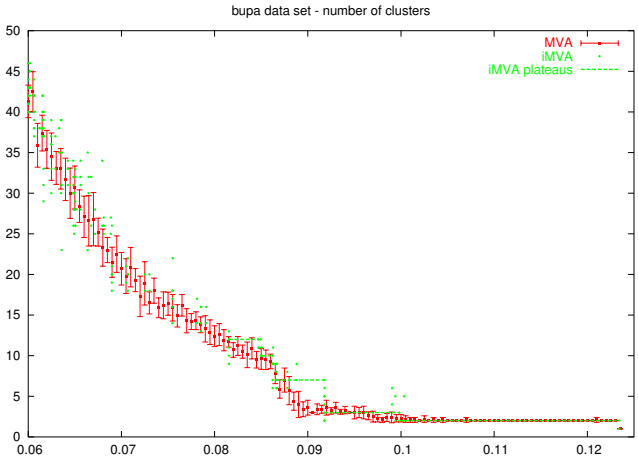
i 4.25.

Dla małych wartości wariancji MVC zachowuje się niestabilnie, co na obu rysunkach widać jako wysokie słupki błędu. Jedyne plateau widoczne na wykresie MVC odpowiadają trzem $((0.090, 0.098))$ i dwóm $((0.100, 0.123)$ o sile $1.23)$ klastrom, przy czym plateau dla trzech klastrów jest mniej widoczne. Oba plateau zostałe znalezione przez IMVC, ale plateau odpowiadające trzem klastrom jest przesunięte. Oprócz tych dwóch plateau, IMVC znalazło trzy małe plateau i dwa większe nie występujące w tendencji MVC. Zwraca uwagę duża liczba powtórzeń MVC przez IMVC dla pewnych wartości wariancji, zakończonych różnymi wynikami (na wykresie reprezentowane przez zielone kropki położone pionowo nad sobą).

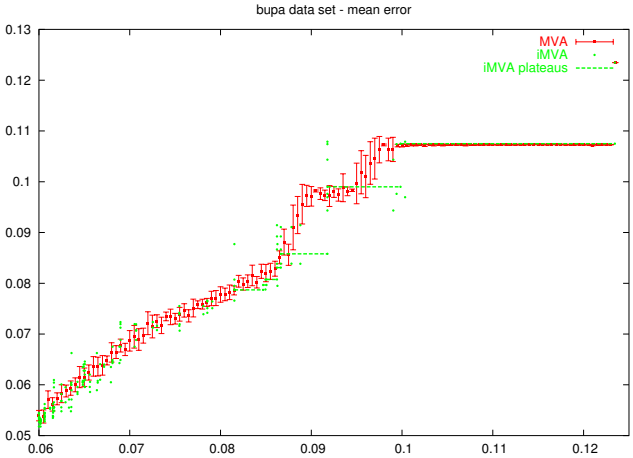
ecoli

Parametry MVC, z jakimi analizowałem zbiór *ecoli* to: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.06$, co odpowiadało około 30 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.26 i 4.27.

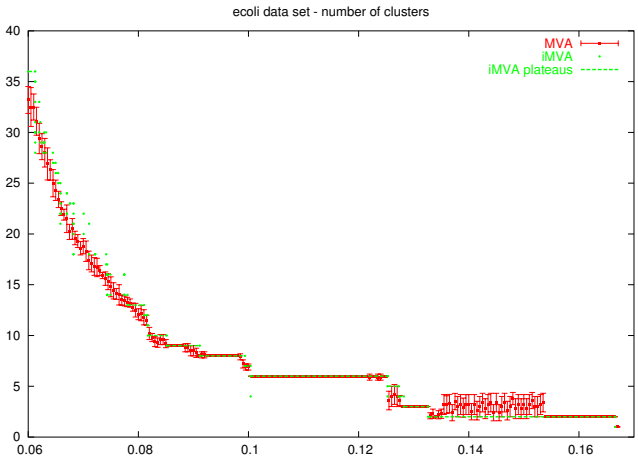
Dla wariancji mniejszej od około 0.08 następuje szybki spadek liczby klastrów. Dla wyższych wartości σ^2_{max0} na wykresie widoczne są coraz większe plate-



Rysunek 4.24: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *bupa*.



Rysunek 4.25: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *bupa*.



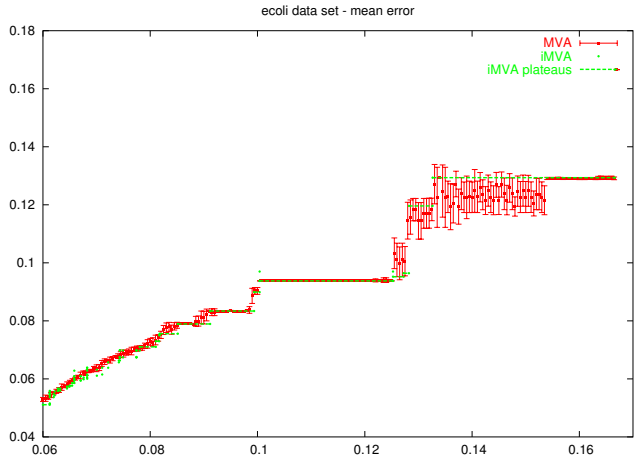
Rysunek 4.26: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *ecoli*.

au odpowiadające dziewięciu, ośmiu, sześciu (o największej sile, $S = 1.24$), trzem i dwóm klastrom. Wszystkie te plateau zostały wykryte przez IMVC, a ich granice są zgodne z granicami wyznaczonymi przez MVC. Największa różnica widoczna jest dla wariancji w przybliżeniu równej $\sigma^2_{max0} = 0.14$, kiedy to MVC jest niestabilny i oscyluje pomiędzy czterema, trzema i dwoma grupami, a IMVC w ogóle tego przedziału niestabilności nie zauważa.

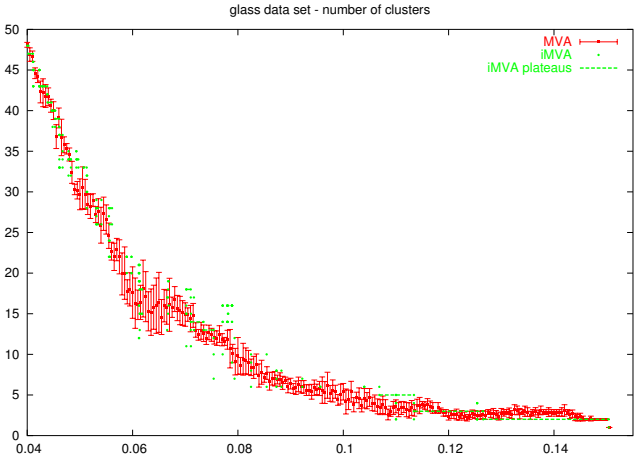
glass

Wariancją początkową dla analizy zbioru *glass* algorytmem MVC była $\sigma^2_{max0} = 0.04$, co odpowiadało około 40 klastrom. Krok ustalony został na $\Delta\sigma^2_{max} = 0.0005$. Wykresy tendencji grupowania przedstawione są na rysunkach 4.28 i 4.29.

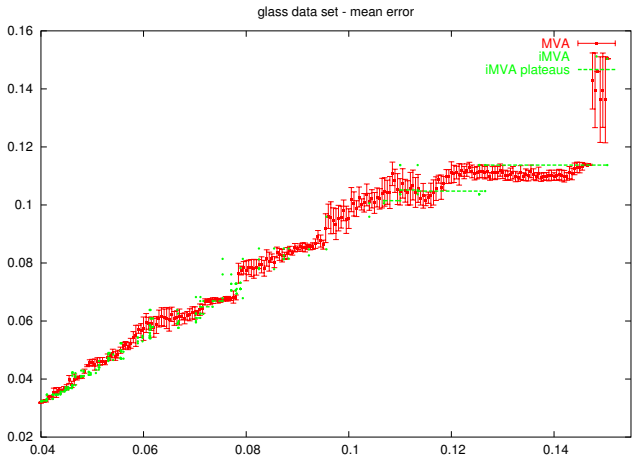
Dla prawie wszystkich wartości wariancji MVC zachowuje się bardzo niestabilnie, co na obu wykresach widać jako wysokie słupki błędów. Trudno tu mówić o jakimkolwiek plateau. Jedynie dla $\sigma^2_{max} \approx 0.15$ podczas dziesięciu uruchomień MVC algorytm zwrócił ten sam wynik. IMVC znajduje w tym zbiorze 3 większe plateau, przy czym tylko jedno z nich (odpowiadające dwóm klastrom) ma odpowiednik w tendencji MVC. Dla niektórych wartości wariancji IMVC wielokrotnie powtarzał MVC otrzymując wyniki różniące się liczbą klastrów.



Rysunek 4.27: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *ecoli*.



Rysunek 4.28: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *glass*.



Rysunek 4.29: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *glass*.

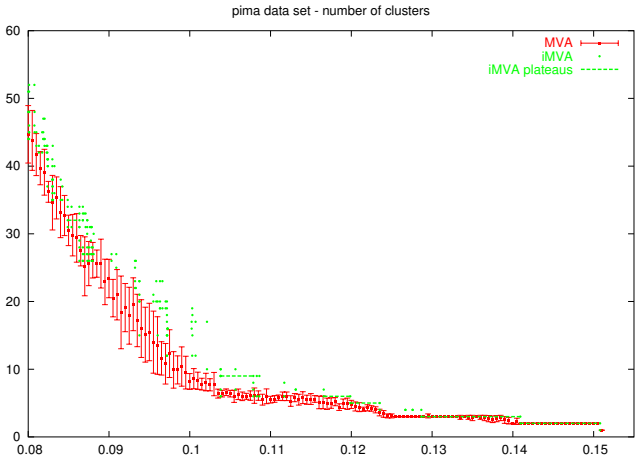
pima

Analizę zbioru *pima* algorytmem MVC przeprowadziłem z następującymi parametrami: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.08$, co odpowiadało około 45 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.30 i 4.31.

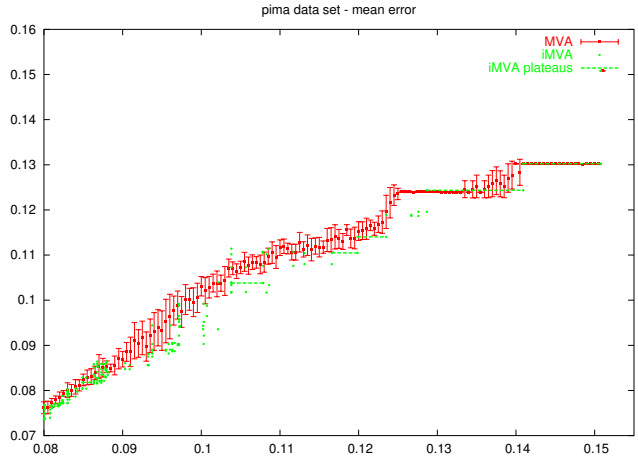
Także w tym zbiorze danych MVC dla wariancji mniejszej niż 0.1 zachowuje się niestabilnie. Na wykresie tendencji MVC widać dwa plateau, odpowiadające trzem i dwóm klastrom. IMVC „gubi się" przy małych wariancjach, wielokrotnie powtarzając algorytm MVC, który, jako niestabilny, daje w kolejnych powtórzeniach różne wyniki. Dla wyższych wartości σ^2_{max} IMVC identyfikuje w zbiorze pięć większych plateau, z których dwa odpowiadają plateau na wykresie MVC, choć przesunięte są w stosunku do nich nieco na prawo.

segmentation

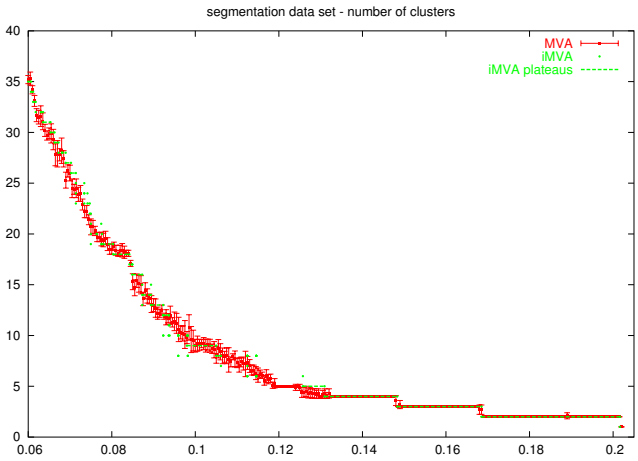
Zbiór *segmentation* analizowany był z następującymi parametrami: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.06$, co odpowiadało około 35 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.32 i 4.33.



Rysunek 4.30: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *pima*.



Rysunek 4.31: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *pima*.



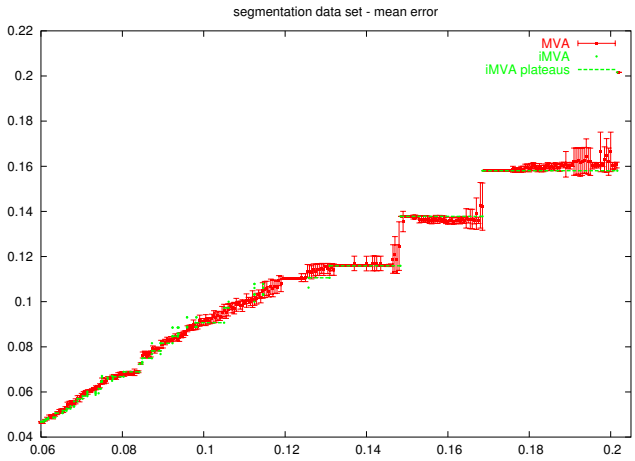
Rysunek 4.32: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *segmentation*.

Po początkowym okresie spadku i niestabilności, na wykresie tendencji liczby klastrów MVC widoczne są trzy wyraźne plateau, odpowiadające czterem, trzem i dwóm (o największej sile, w przybliżeniu równej 1.19) grupom. Ciekawe jest, że o ile na wykresie liczby klastrów plateau wyglądają na stabilne, nie są takimi na wykresie błędu J_e, a zatem dla wariancji należącej do danego plateau możemy otrzymać różne rozwiązania. Wszystkie trzy plateau znajdowane są przez IMVC, przy czym algorytm ten dość dobrze identyfikuje ich krańce. Oprócz tych plateau, IMVC znajduje trzy plateau o niewielkiej sile i nie mające odpowiedników na wykresie MVC.

wine

Parametry MVC, z jakimi analizowałem zbiór *wine* to: krok wariancji $\Delta\sigma^2_{max} = 0.0005$ i $\sigma^2_{max0} = 0.09$, co odpowiadało około 40 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.34 i 4.35.

Na wykresie tendencji liczby klastrów MVC widoczne są dwa wyraźne plateau, odpowiadające trzem i dwóm klastrom. Plateau te są poprawnie identyfikowane przez IMVC. IMVC znajduje ponadto kilka plateau nie mających odpowiedników na wykresie MVC, lecz wszystkie te plateau mają niewielką, w porównaniu



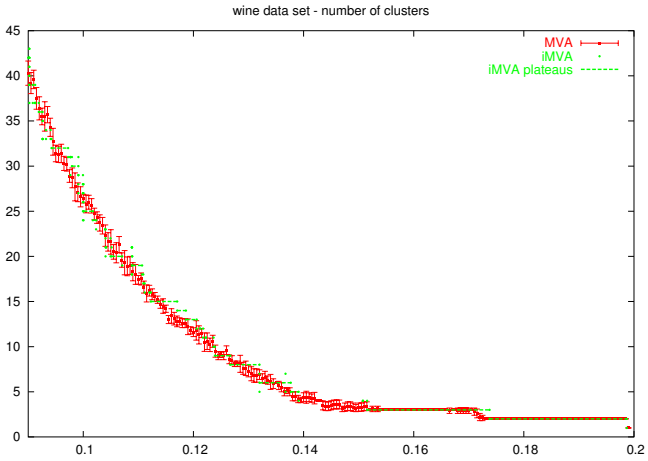
Rysunek 4.33: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *segmentation*.

z dwoma największymi, siłę. Dla wartości wariancji odpowiadających szybkiemu spadkowi liczby klastrów na wykresie MVC, IMVC dosyć dobrze aproksymuje tamtą krzywą.

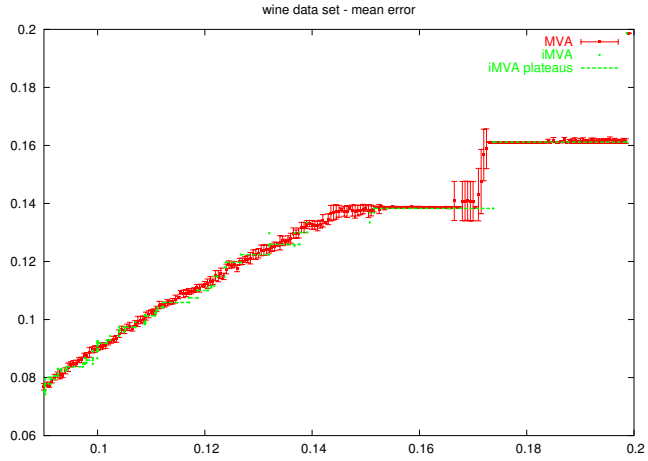
yeast

Yeast jest dosyć dużym zbiorem danych, co łączy się z długim czasem działania algorytmu MVC, dlatego też wybrałem stosunkowo dużą wartość kroku $\Delta\sigma^2_{max} = 0.001$. Wariancją początkową było $\sigma^2_{max0} = 0.06$, co odpowiadało około 25 klastrom. Wykresy tendencji grupowania przedstawione są na rysunkach 4.36 i 4.37.

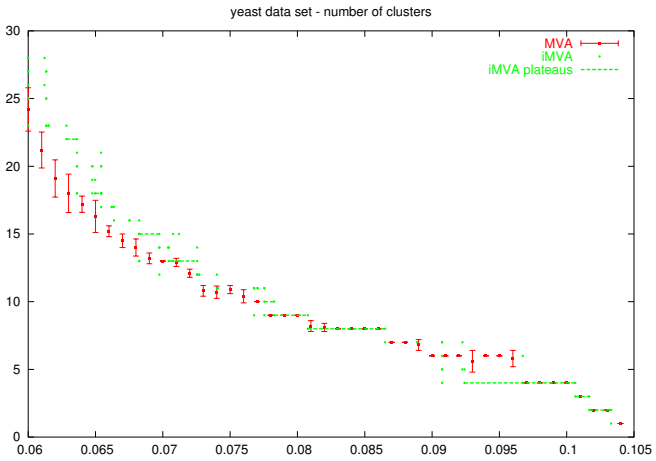
Funkcja tendencji grupowania MVC próbkowana jest dosyć rzadko, dlatego też ewentualne plateau są na wykresie gorzej widoczne. Z całą pewnością można stwierdzić, że nieobecne są żadne plateau o dużej sile. Z drugiej strony na wykresie są obszary, gdzie wszystkie dziesięć powtórzeń badania tendencji dało ten sam rezultat, a zatem w których MVC jest stabilny. Biorąc te wszystkie cechy pod uwagę, można wyznaczyć dwa największe plateau, odpowiadające siedmiu i pięciu klastrom (choć plateau odpowiadające pięciu klastrom jest dla pewnych wartości σ^2_{max} niestabilne), oraz kilka mniejszych, odpowiadających ośmiu, sze-



Rysunek 4.34: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *wine*.



Rysunek 4.35: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *wine*.



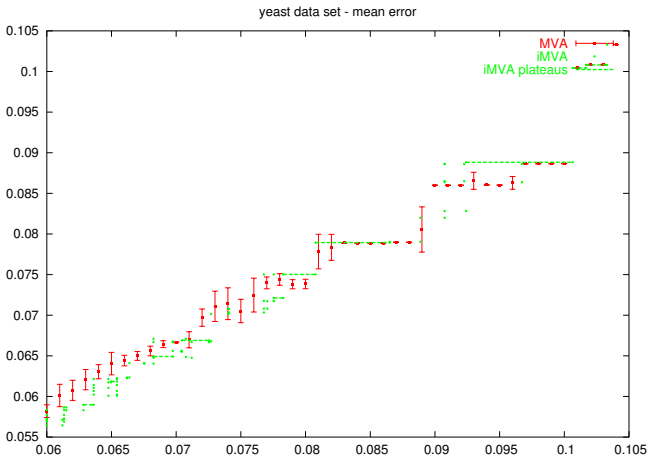
Rysunek 4.36: Wykresy tendecji grupowania (liczby grup w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *yeast*.

ściu i czterem klastrom. Tendencja wyznaczona przez IMVC różni się od tendencji MVC dość znacznie dla mniejszych wartości wariancji. IMVC nie dawał sobie rady z niestabilnością MVC i wielokrotnie powtarzał MVC dla tych samych wartości σ^2_{max}. Z opisanych plateau IMVC odkrył jedno większe i dwa mniejsze, przy czym plateau odpowiadające czterem klastrom zostało rozciągnięte kosztem w istocie silniejszego plateau o pięciu klastrach.

Wnioski

Porównując wyniki badania tendencji przez powtarzanie MVC i wykonanie IMVC można stwierdzić, że IMVC dosyć dobrze aproksymuje tendencję klastrów zwłaszcza na „prostszych” zbiorach danych, którymi okazały się w większości zbiory wygenerowane. We wszystkich eksperymentach, zarówno dla zbiorów wygenerowanych, jak i rzeczywistych, IMVC znajdował prawie wszystkie plateau znalezione przez MVC, natomiast niekiedy były one przesunięte lub dłuższe niż ich odpowiedniki na wykresie tendencji MVC.

Największym problemem algorytmu IMVC jest nieoptymalne działanie dla wariancji, dla których MVC jest niestabilny i daje różne wyniki w kolejnych wykonaniach. IMVC niepotrzebnie wielokrotnie powtarza MVC, starając się znaleźć



Rysunek 4.37: Wykresy tendecji grupowania (błędu J_e w zależności od wariancji) wyznaczonej przez algorytmy MVC i IMVC dla zbioru *yeast*.

wynik umożliwiający konstrukcję liniowo opadającej krzywej opisującej liczbę klastrów. Można wnioskować, że forma algorytmu zaproponowana w 3.4 jest zbyt prosta i że należy poprawić samo sterowanie algorytmu, tak, aby umożliwić wykrywanie i ignorowanie obszarów niestabilności MVC. Obszary takie mogą być zaznaczane jako niestabilne plateau. Należy jednak podkreślić, że jest to niewielka poprawa, a pomysł algorytmu generalnie sprawdził się.

Analizując wyniki obu algorytmów ze zdziwieniem zobaczyłem, że gdy używana jest znormalizowana miara odległości, siła plateau jest znacznie mniejsza, a znaczące plateau pojawiają się niezmiernie rzadko. W większości rzeczywistych zbiorów danych najsilniejsze plateau miało siłę w przybliżeniu równą $S^* \approx 1.25$, co dalekie jest od $S = 2$ wymaganego przez autorów [22], by plateau uznane zostało za znaczące. Wynik ten jest niebezpiecznie bliski wynikom uzyskiwanym na losowych zbiorach danych w eksperymentach opisywanych w rozdziale 4.6.1. Mogłoby to świadczyć o tym, że analizowane przeze mnie zbiory danych nie mają jakiejś dobrze widocznej wewnętrznej struktury, gdyby nie porównanie wyników uzyskanych dla zbioru *iris* przy użyciu miary znormalizowanej i zwykłej. Podczas analiz przy użyciu zwykłej miary euklidesowej, najsilniejsze plateau miało w tym zbiorze siłę $S = 3.24$. Przy miarze znormalizowanej to samo plateau, odpo-

wiadające dwóm klastrom, miało siłę $S = 1.67$. Z tych powodów niemożliwe jest niestety decydowanie o tym, czy w danym zbiorze w ogóle istnieje jakaś struktura wewnętrzna, biorąc pod uwagę jedynie siłę największego plateau. Wniosek ten podważa wyniki uzyskane w [22]. W moich wynikach wyraźna jest zależność pomiędzy *numeryczną wartością* poszczególnych atrybutów, a siłą uzyskiwanych plateau. Choć szczegółowe parametry rozkładów nie zostały w [22] podane, wydaje mi się, że autorzy, przeprowadzając eksperymenty na losowych zbiorach danych, generowali je z rozkładu $U(0,1)$. Z kolei w dalszej części artykułu używali nieznormalizowanych zbiorów danych. Z uzyskanych przeze mnie wyników można wnioskować, że postępowanie takie jest błędne.

Nawet przy użyciu miary znormalizowanej zachodzi zależność pomiędzy siłą plateau a dopasowaniem grupowania odpowiadającemu temu plateau do zbioru danych – w zbiorze *iris* kolejność plateau jest taka sama bez względu na używaną miarę odległości.

Rozdział 5

Podsumowanie

W pracy opisałem opublikowany niedawno algorytm grupowania Maximum Variance Clusterer (MVC). Algorytm ten ma bardzo ciekawe teoretyczne właściwości, ponieważ do konstrukcji grup używa pojęcia wariancji grupy, która jest miarą rozproszenia przykładów zaliczonych do tej grupy. Dzięki takiemu podejściu algorytm ma szanse na elastyczne manipulowanie liczbą grup, co z kolei jest ważne w wielu zastosowaniach, w których liczba ta nie jest znana a priori, a nawet jest jedną z istotnych części odpowiedzi algorytmu.

Jedynym istotnym parametrem MVC jest maksymalna wariancja σ^2_{max}, dzięki czemu korzystanie z tego algorytmu jest stosunkowo proste. Ponadto w łatwy sposób można znaleźć właściwe jego wartości przez wyznaczenie tendencji grupowania, wykorzystując do tego bądź powtarzany z różnymi wartościami σ^2_{max} MVC, bądź zaproponowany przeze mnie algorytm przyrostowy IMVC. Dzięki temu użytkownik algorytmu nie musi wiedzieć prawie nic ani o danych, ani o samym mechanizmie algorytmu. MVC może stanowić dobrą alternatywę dla wielu istniejących algorytmów grupowania.

Wyniki badań eksperymentalnych na syntetycznych zbiorach danych okazały się bardzo zachęcające. Wykorzystując szybkie badanie tendencji algorytmem przyrostowym IMVC, zostały wyznaczone właściwe wartości parametru σ^2_{max}, a następnie zbióry danych zostały podzielone na grupy najlepiej do tych zbiorów pasujące. Na tych zbiorach algorytm okazał się dużo lepszy od algorytmu

k-środków, który nierzadko w ogóle nie był w stanie znaleźć poprawnego rozwiązania, ponieważ z punktu widzenia k-środków rozwiązanie optymalne, dające minimalny błąd, nie pasowało do zbioru. Dlatego też dobrym pomysłem okazało się oparcie działania algorytmu, a zatem i zdefiniowanie przestrzeni rozwiązań poprzez określenie maksymalnej wariancji sumy dowolnych dwóch grup.

MVC zostało również przetestowane na kilku rzeczywistych zbiorach danych. Niestety na większości z nich algorytm zachowywał się dużo gorzej. W kilku przypadkach badanie tendencji w ogóle nie znalazło odpowiednich wartości dla parametru σ^2_{max}. Można oczywiście podejrzewać, że w tych zbiorach danych nie istnieje dobre grupowanie, bo niemożliwe jest wyznaczenie dobrze od siebie odseparowanych grup, gdyż zbiory te są w istocie bardzo jednorodne. Większym problemem wydaje się jednak to, że algorytm często zachowywał się niestabilnie, dając różne rozwiązania w kolejnych uruchomieniach z tą samą wartością σ^2_{max}, co praktycznie nie zdarzało się w syntetycznych zbiorach danych. Trzeba również zauważyć, że gdy algorytm używa znormalizowanej miary odległości, co konieczne jest w rzeczywistych zbiorach danych, siła plateau wskazująca na najlepsze wartości parametru jest mniejsza, niż gdy używana jest miara euklidesowa. Jest to zachowanie istotne dla wyników algorytmu, o którym nie wspomnieli autorzy MVC i które podważa sens badania stopnia ustrukturalizowania zbioru tylko przez badanie siły największego plateau. Otrzymane jako wynik grupy różniły się zazwyczaj od klas obecnych w zbiorach danych. Nie może być to jednak uznane za wadę algorytmu, ponieważ algorytmy grupowania biorą pod uwagę wartości wszystkich atrybutów i nie mają dostępnej w formie sprzężenia zwrotnego żadnej informacji o klasach. Oczywiście jeżeli grupy odpowiadają klasom, jest to informacja pozytywna, świadcząca z jednej strony o dobrym działaniu algorytmu, a z drugiej o tym, że podział zbioru na klasy wynika z wartości atrybutów.

Porównując algorytm MVC z algorytmem k-środków trzeba wspomnieć o dwóch dużych wadach pierwszego z nich. MVC działa do kilku rzędów wielkości wolniej niż k-środki. Takie zachowanie nie jest jeszcze dużą wadą, ponieważ działanie k-środków zazwyczaj trzeba kilka razy powtarzać. Nie należy również zapominać o tym, że autorzy MVC otrzymali znacznie lepszy czas działania al-

gorytmu, prawdopodobnie dlatego, że moja implementacja miała być łatwa do rozbudowy i wymiany poszczególnych elementów. Korzystałem również z wysokopoziomowej biblioteki Weka, której pominięcie mogłoby okazać się opłacalne jeśli chodzi o czas działania algorytmu. Drugą i dużo poważniejszą wadą MVC jest kwadratowa złożoność pamięciowa, która istotnie ogranicza rozmiar zbioru danych możliwego do analizy tym algorytmem.

Podsumowując, można stwierdzić że model przestrzeni, czyli oparcie grupowania o miarę rozproszenia elementów grupy, zastosowane w pomysłowy sposób w algorytmie MVC sprawdziło się. Sam algorytm jednak zdaje egzamin tylko dla dość dobrze ustrukturalizowanych i niewielkich zbiorów danych. Wydaje mi się, że największym wyzwaniem jest opracowanie algorytmu, który byłby wolny od wymienionych przeze mnie wad, a jednocześnie korzystał z modelu przestrzeni używanego przez MVC.

Spis literatury

[1] ANDERSON, T. *An Introduction to Multivariate Statistical Analysis*. John Wiley & Sons, 1984.

[2] BALL, G., AND HALL, D. A clustering technique for summarizing multivariate data. *Behavioral Sciences 12*, 3 (1967), 153–155.

[3] BECK, K. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, 1999.

[4] BEZDEK, J., AND PAL, N. Some new indexes of cluster validity. *IEEE Transactions on Systems, Man, And Cybernetics–Part B: Cybernetics 28*, 3 (1998), 301–315.

[5] BLAKE, C., AND MERZ, C. UCI repository of machine learning databases, 1998.

[6] BRADLEY, P. S., FAYYAD, U. M., AND REINA, C. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining* (1998), pp. 9–15.

[7] CICHOSZ, P. *Systemy uczące się*. WNT, 2000.

[8] FRALEY, C., AND RAFTERY, A. How many clusters? which clustering method? answers via model-based cluster analysis. *The Computer Journal 41*, 8 (1998), 578–588.

[9] GAMMA, E., HELM, R., JOHNSON, R., AND VLISSIDES, J. *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[10] HALL, L., OZYURT, B., AND BEZDEK, J. Clustering with a genetically optimized approach. *IEEE Transactions on Evolutionary Computation 3*, 2 (1999), 103–112.

[11] HARKIN, G. The isodata algorithm, 2002.

[12] HUI-DONG, J., KWONG-SAK, L., AND MAN-LEUNG, W. Genetic-guided model-based clustering algorithms. In *Proc. of the International Conference on Artificial Intelligence* (2001), H. Arabnia, Ed., vol. 2, pp. 653–659.

[13] JAIN, A., MURTY, M., AND FLYNN, P. Data clustering: A review. *ACM Computing Surveys 31*, 3 (1999), 265–323.

[14] JAIN, A. K., AND DUBES, R. C. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[15] JÓŹWIAK, J., AND PODGÓRSKI, J. *Statystyka od podstaw*. Polskie Wydawnictwo Ekonomiczne, 2001.

[16] KRISHNA, K., AND MURTY, M. N. Genetic k-means algorithm. *IEEE Transactions on Systems, Man And Cybernetics–Part B: Cybernetics 29*, 3 (1999), 433–439.

[17] MACQUEEN, J. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. Math. Statistics and Probability* (1967), L. L. Cam and J. Neyman, Eds., vol. 1, pp. 281–297.

[18] MITCHELL, T. *Machine Learning*. McGraw–Hill, 1997.

[19] RZADCA, K., AND FERRI, F. Incrementally assessing cluster tendencies. In *Pattern Recognition and Image Analysis* (2003), F. Perales, A. Campilho, N. P. de la Blanca, and A. Sanfeliu, Eds., no. 2653 in LNCS, Springer-Verlag, pp. 868–875.

[20] SARKAR, V. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. MIT Press, 1989.

[21] SCHWARZ, G. Estimating the dimension of a model. *The Annals of Statistics 6* (1978).

[22] VEENMAN, C., REINDERS, M., AND BACKER, E. A maximum variance cluster algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 9 (2002), 1273–1280.

[23] VEENMAN, C., REINDERS, M., AND BACKER, E. A cellular coevolutionary algorithm for image segmentation. *IEEE Transactions on Image Processing 12*, 3 (2003), 304–316.

[24] WITTEN, I., AND FRANK, E. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

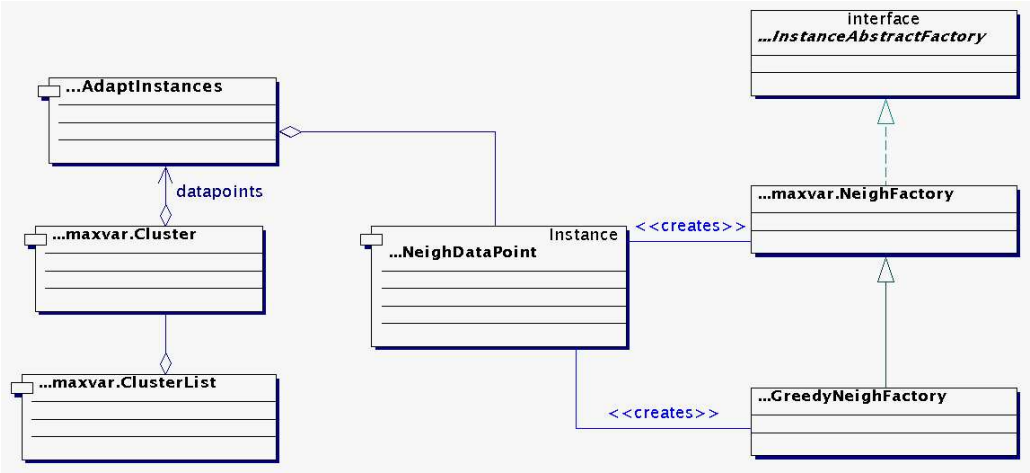
Dodatek A

Opis programu

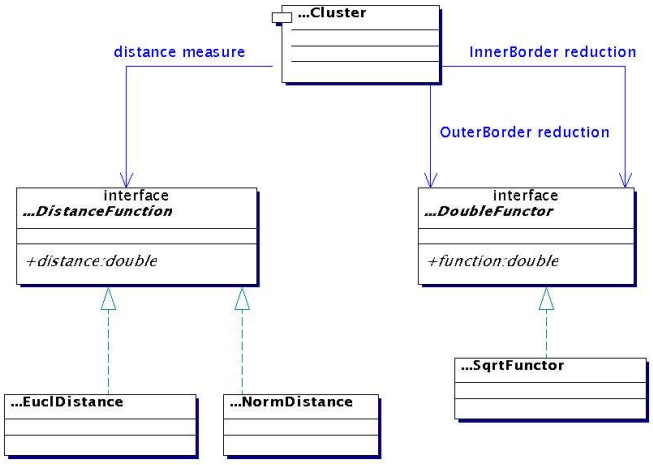
Program zrealizowany w ramach tej pracy został zaprojektowany jako rozszerzenie biblioteki Weka [24], napisanej w języku Java i obecnie najpopularniejszej biblioteki maszynowego uczenia się. Wszystkie klasy związane z realizacją algorytmów MVC i IMVC umieściłem w pakiecie `weka.clusterers.maxvar`. Pozostałe klasy, w większości pomocnicze i ułatwiające przeprowadzenie eksperymentów, dołączyłem do pakietu `weka.clusterers`. Do większości klas, w szczególności do wszystkich niskopoziomowych, zostały napisane testy wykorzystujące bibliotekę JUnit, przy czym większość testów pisana była przed stworzeniem kodu (zgodnie z metodologią Extreme Programming [3]). Cała biblioteka ma elastyczną strukturę dzięki intensywnemu wykorzystaniu wzorców projektowych [9]. Projekt jest automatycznie budowany i testowany przy pomocy programu `ant`.

Zależności pomiędzy poszczególnymi klasami programu przedstawione w notacji UML pokazane są na rysunkach A.1, A.2 i A.3. Szczegóły budowy i opis poszczególnych metod dostępny jest w automatycznie wygenerowanej dokumentacji Javadoc. Poniżej omówione zostaną klasy kluczowe dla zrozumienia projektu.

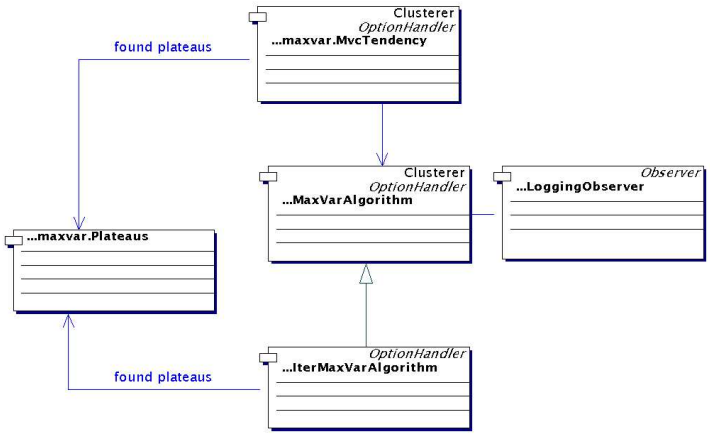
- `NeighDataPoint` implementuje punkt danych, który pamięta uporządkowaną pod względem odległości listę sąsiadów. Klasa udostępnia operacje pozwalające wyznaczyć wewnętrzną i zewnętrzną granicę danego punktu. Obiekty tej klasy tworzone są przy użyciu jednej z fabryk implementują-



Rysunek A.1: Diagram UML klas reprezentujących klastry i punkty danych.



Rysunek A.2: Diagram UML klas odpowiedzialnych za realizację algorytmu na poziomie klastra.



Rysunek A.3: Diagram UML klas zapewniających wysokopoziomową obsługę algorytmu.

cych interfejs `InstanceAbstractFactory` (wzorzec projektowy Factory Method).

- `weka.core.AdaptInstances` to adapter klasy `Instances` z biblioteki Weka, pozwalający korzystać z usług oferowanych przez tę klasę, jednocześnie pamiętając informacje o rzeczywistej klasie przykładów. `Instances` w momencie dodawania przykładu płytko go kopiuje (zakładając, że klasą przykładu jest `Instance`), dzięki czemu tracona jest informacja przechowywana w klasie `NeighDataPoint`.

- `Cluster` reprezentuje pojedynczy klaster składający się ze środka i pewnej liczby przykładów. Tu zrealizowane są podstawowe dla MVC operacje: scalania, dzielenia i perturbacji klastrów.

- `ClusterList` reprezentuje pewne grupowanie (listę obiektów klasy `Cluster`) i udostępnia operacje, które wykonywane są jednocześnie na wszystkich klastrach: inicjalizację, obliczanie błędu średniokwadratowego, czy zmianę parametrów klastrów.

- `DistanceFunction` jest interfejsem realizującym wzorzec projektowy Command. Klasy implementujące ten interfejs dostarczają różnych miar od-

ległości używanych w algorytmie. W moim projekcie zaimplementowane są dwie takie klasy: `EuclDistance`, wyznaczająca odległość euklidesową i `NormDistance`, wyznaczająca znormalizowaną odległość euklidesową i uwzględniająca atrybuty nominalne.

- `DoubleFunctor` jest interfejsem realizującym wzorzec projektowy Command, reprezentujący jednowymiarową funkcję o dziedzinie i przeciwdziedzinie na liczbach rzeczywistych. Implementująca ten interfejs klasa `SqrtFunctor` zwraca pierwiastek kwadratowy argumentu i używana jest do wyznaczania wielkości części zewnętrznej i wewnętrznej granicy klastra, branej pod uwagę w krokach scalania i perturbacji algorytmu MVC.

- `MaxVarAlgorithm` realizuje algorytm MVC i jako wynik zwraca grupowanie proponowane przez ten algorytm.

- `LoggingObserver`. W trakcie przebiegu algorytmu klasa `MaxVarAlgorithm` generuje pewne zdarzenia (takie jak koniec kolejnej iteracji), które są obserwowane przez obiekty klasy `LoggingObserver`. Klasa ta zapisuje wyniki algorytmu po każdej iteracji, może również eksportować aktualne grupowanie do postaci zrozumiałej przez program `gnuplot`.

- `Plateaus` reprezentuje listę plateau znalezionych w trakcie badania tendencji klasteryzacji.

- `MvcTendency` realizuje badanie tendencji przez powtarzanie algorytmu MVC ze stopniowo zwiększaną wariancją. Po zbadaniu tendencji wybierana jest wariancja ze środka plateau o największej sile i dla takiej wariancji wykonywane jest grupowanie algorytmem MVC.

- `IterMaxVarAlgorithm` bada tendencję algorytmem IMVC, a następnie wykonuje grupowanie algorytmem MVC z wariancją odpowiadającą środkowi plateau o największej sile.

Dodatek B

Przykładowe uruchomienie programu

Polecenie:

```
java -jar jars/mvc.jar
weka.clusterers.maxvar.IterMaxVarAlgorithm
-t datasets/iris.arff
-c last
-U IMVC_iris.data
-P iris_plats.txt
-V 0.06
-D
```

powoduje zbadanie tendencji algorytmem IMVC dla zbioru *iris*, przy czym informacja o klasie zawarta w ostatnim atrybucie jest przed badaniem usuwana. Punkty, w których uruchamiany był MVC, zostaną zapisane do zbioru `IMVC_iris.data`, a znalezione plateau do zbioru `iris_plats.txt`. Badanie tendencji rozpocznie się przy wariancji $\sigma^2_{max} = 0.06$. Używana będzie znormalizowana miara odległości. Po znalezieniu optymalnej wartości wariancji, algorytm przeprowadzi grupowanie, a uzyskane grupy porówna z kategoriami ze zbioru danych.