MOW Projekt 13

Porównanie algorytmów grupowania i klasyfikacji do detekcji anomalii.

Katarzyna Piórkowska, 259078
Tomasz Korzeniowski, 265753

3 kwietnia 2018

1 Zadanie

1.1 Treść

Nienadzorowana detekcja anomalii za pomocą odpowiednio opakowanych wybranych algorytmów grupowania dostępnych w R. Porównanie z nadzorowaną detekcją anomalii za pomocą dostępnych w R algorytmów klasyfikacji.

1.2 Interpretacja

W ramach projektu należy zbadać czy możliwe jest wykrycie anomalii w danych korzystając ze standardowych algorytmów grupowania dostępnych w R. W tym celu należy zapewnić mechanizm generowania modelu grupowania, który zwróci podział danych trenujących na grupy. Następnie, dla każdego nowego przykładu testowego, trzeba ocenić w jakim stopniu jest on podobny do którejś z wyznaczonych grup. Ocena polegać będzie na wyznaczeniu wskaźnika nietypowości, który jest miarą liczbową wskazującą na ile badany przykład jest niepodobny do rozważanych grup.

Do symulowania anomalii w danych przyjmiemy, że jedna z klas w nich występująca (np. najmniej liczna) będzie stanowiła anomalie. Przykłady należące do tej klasy nie zostaną wykorzystane do budowy modelu grupowania.

Częścią implementacji projektu będzie zapewnienie opakowania wybranych algorytmów grupowania w funkcję, która zwraca model pogrupowanych danych trenujących. Użytkownik będzie miał możliwość podania metody grupowania z jakiej chce skorzystać (wraz z jej niezbędnymi parametrami). Inna funkcja będzie miała za zadanie skorzystać z wyznaczonego modelu oraz dopasować dane testowe przez wyznaczenie zadanego wskaźnika nietypowości.

W części analitycznej zostanie przeprowadzona symulacja wykrywania anomalii na kilku zbiorach danych. Wyniki otrzymane przy użyciu zaimplementowanych funkcji zostaną porównane z wynikami uzyskanymi przez zastosowanie znanych algorytmów klasyfikacji dostępnych w R. W tym drugim przypadku algorytmy będą znały klasy do jakich należą obserwacje by wskazać anomalie jako jedną z klas. Porównanie wyników obu podejść będzie polegało na wyznaczeniu wskaźników jakości (dokładności) powyższych rozwiązań.

2 Algorytmy

Do realizacji zadania zostaną wykorzystane trzy algorytmy grupowania (k-średnich, k-medoidów,) oraz dwa algorytmy klasyfikacji (...). Każdy z nich zostanie pokrótce opisany wraz ze wskazaniem kluczowych parametrów.

2.1 Algorytm k-średnich

Algorytm k-średnich jest jednym z najprostszych algorytmów rozwiązujących zadanie grupowania. Ideą algorytmu jest przyporządkowanie pewnego zbioru N przykładów do przyjętej a priori liczby grup K. Każda grupa posiada dokładnie jeden centroid, czyli punkt reprezentujący wartość średnią grupy. Pojedynczą obserwację $x_i = (x_1, x_2, \ldots, x_N)$ można przyporządkować tylko do jednego z centroidów $c_j = (c_1, c_2, \ldots, c_K)$ Oznacza to minimalizację funkcji:

$$J = \sum_{j=1}^{K} \sum_{i=1}^{N} \|x_i - c_j\|^2 \tag{1}$$

Po zakończeniu pojedynczej iteracji grupowania należy uaktualnić położenie centroidów i przyporządkować obserwacje ponownie. Przebieg grupowania przedstawia algorytm 1.

Algorytm 1 k-średnich

1. Wyznacz początkowe położenie centroidów
2. Przyporządkuj każdej obserwacji najbliższy jej centroid.
3. Gdy wszystkie obserwacje zostaną przyporządkowane, wyznacz ponownie położenie centroidów, znajdując wartość średnią obserwacji przypisanych do centroidu:

$$c_{ji} = \frac{1}{M} \sum_{m=1}^{M} x_m \quad , gdzie\ M - liczba\ obserwacji\ w\ c_j$$

4. Powtarzaj kroki 2. i 3., dopóki centroidy zmieniają swoje położenie lub nie zostanie osiągnięta maksymalna liczba iteracji.

Do zalet algorytmu należy proste znajdowanie podziału grup dobrze odseparowanych od siebie. Największą wadą algorytmu jest konieczność podania liczby grup na jakie chcemy podzielić dane. Ponadto początkowe położenie centroidów determinuje wynik grupowania. Algorytm nie radzi sobie również z danymi silnie zaszumionymi i/lub zawierającymi obserwacje odstające (zaburzenie średniej).

W środowisku R istnieje funkcja *kmeans* w standardowym pakiecie *stats*, która realizuje algorytm k-średnich.

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"), trace=FALSE)
```

Jej główne parametry wejściowe to:

x	–	zbiór danych (numerycznych) do pogrupowania
centers	–	wstępne położenie centroidów lub ich liczba oznaczająca na jak wiele grup należy podzielić dane
iter.max	–	maksymalna liczba iteracji algorytmu, warunek stopu

Pozostałe parametry związane są z wewnętrzną implementacją algorytmu w pakiecie R i nie będą rozpatrywane w ramach projektu. Wynikiem działania algorytmu jest model zawierający wektor przyporządkowania obserwacji do grup, środki wyznaczonych grup oraz pomocnicze miary (suma kwadratów odległości między przykładami w grupie, liczność grup, liczba wykonanych iteracji)

2.2 Algorytm k-medoidów

Struktura algorytmu k-medoidów jest niemal taka sama jak algorytmu k-średnich. Jedną różnicą jest przyjęcie, że środkiem grupy jest medoid, a nie wartość średnia. Medoid jest

najbardziej centralnym przykładem w grupie (średnia grupy może się nie pokrywać z żadnym przykładem należącym do wyznaczonej grupy). Powoduje to uodpornienie algorytmu na wartości odstające.

Aby wybrać nowy medoid, w kolejnych iteracjach algorytmu, rozważane są wszystkie przykłady p. Istnieją 4 możliwe przypadki, które należy sprawdzić by stwierdzić czy przykład p (niebędący medoidem) może zastąpić medoid m:

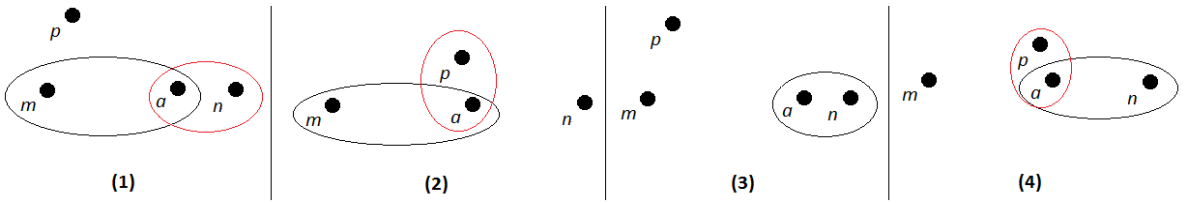
1. Przykład a należy do grupy medoidu m – jeśli zastąpimy m przykładem p oraz a znajduje się bliżej medoidu n to przydziel a do grupy n.

2. Przykład a należy do grupy medoidu m – jeśli zastąpimy m przykładem p oraz a znajduje się bliżej przykładu p to przydziel a do nowego medoidu p.

3. Przykład a należy do grupy medoidu n – jeśli zastąpimy medoid m przykładem p oraz a znajduje się bliżej medoidu n to nie zmieniaj przydziału przykładu a.

4. Przykład a należy do grupy medoidu n – jeśli zastąpimy medoid m przykładem p oraz a znajduje się bliżej p to przydziel a do nowego medoidu p.

Powyższe możliwości ilustruje wykres 1.



Wykres 1: Przypadki zamiany medoidu m przykładem p.

Przebieg grupowania przedstawia algorytm 2.

Algorytm 2 k-medoidów

1. Wybierz K przykładów jako medoidy.
2. Przyporządkuj każdej obserwacji najbliższy jej medoid.
3. Dla każdego medoidu m wybierz inny przykład p, który zmniejszy odległość między przykładami w obrębie nowej grupy.
4. Powtarzaj kroki 2. i 3., dopóki medoidy zmieniają się lub nie zostanie osiągnięta maksymalna liczba iteracji.

W języku R istnieje algorytm *pam* (ang. partitioning around medoids) w pakiecie cluster, który implementuje algorytm k-medoidów.

```
pam(x, k, diss = inherits(x, "dist"), metric = "euclidean",
    medoids = NULL, stand = FALSE, cluster.only = FALSE,
    do.swap = TRUE,
    keep.diss = !diss && !cluster.only && n < 100,
    keep.data = !diss && !cluster.only,
    pamonce = FALSE, trace.lev = 0)
```

(opis parametrów wejściowych i wyjściowych)

2.3 Algorytm hierarchiczny

Algorytmy hierarchiczne pozwalają na graficzną reprezentację struktury klasteryzacji w postaci drzewa. Można wyróżnić tu dwa podejścia: aglomeracyjne, gdzie każda z obserwacji stanowi na początku oddzielną grupę oraz deglomeracyjne - rozpoczynające od jednej grupy zawierającej wszystkie próbki. Przebieg aglomeracji przedstawia algorytm 3.

Algorytm 3 hierarchiczny

1. Utwórz jednoelementowe grupy.
2. Zbuduj macierz odległości pomiędzy rozpatrywanymi elementami.
3. Znajdź parę elementów, między którymi odległość jest najmniejsza.
4. Połącz znalezione grupy w jedną i wyznacz ich nowy środek ciężkości jako średnią środków cieżkości grup składowych.
5. Powtarzaj kroki 2-4 aż do uzyskania jednego skupiska zawierającego wszystkie próbki.

Do zalet algorytmów hierarchicznych należy brak konieczności początkowego określenia liczby klas w przeciwieństwie do opisanych wyżej algorytmów k-średnich i k-medoidów. Do wad można zaliczyć dość duże zróżnicowanie wyników w zależności od wybranych metod łączenia grup, wśród których znajdują się m.in. metody najbliższych i najdalszych sąsiadów, metoda średnich (centroidów) oraz minimalnej wariancji Warda.

Algorytmem aglomeracyjnym dostępnym w języku R jest agnes (ang. agglomerative nesting) w pakiecie cluster.

```
agnes(x, diss = inherits(x, "dist"), metric = "euclidean",
      stand = FALSE, method = "average", par.method,
      keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

Jego najważniejsze parametry wejściowe to:

x	–	zbiór danych (numerycznych) do pogrupowania
metric	–	metryka, według której obliczana jest odległość
method	–	metoda łączenia grup

Wynikiem działania algorytmu jest graficzna reprezentacja struktury grupowania - dendrogram. Ponadto obliczany jest współczynnik aglomeracji, który charakteryzuje wygląd dendrogramu: niski współczynnik oznacza węższe struktury.

2.4 Drzewa decyzyjne

Pierwszą z metod klasyfikacji stosowaną w celu porównania z algorytmami grupowania są drzewa decyzyjne. W języku R istnieje wiele jej implementacji, z których wykorzystany zostanie algorytm C4.5 przedstawiony poniżej.

Algorytm 4 Algorytm C4.5

1. Utwórz zbiór treningowy T.
2. Wybierz taki atrybut, który najlepiej różnicowałby przykłady ze zbioru T.
3. Utwórz węzeł drzewa odpowiadający wybranemu atrybutowi.
4. Do węzła dodaj podwęzły, z których każdy reprezentuje pewną wartość badanego atrybutu.
5. Powtarzaj podziały dla kolejnych podwęzłów.

Do niewątpliwych zalet drzew decyzyjnych należy czytelna forma reprezentacji, efektywność pamięciowa oraz wszechstronność metody. Konieczny jest w niej jednak

kompromis pomiędzy wielkością drzewa a jakością klasyfikacji. Drzewa mogą być również podatne na zjawisko nadmiernego dopasowania.

Wykorzystaną w projekcie funkcją będzie J48 z pakietu RWeka.

```
J48(formula, data, subset, na.action,
    control = Weka_control(), options = NULL)
```

Jego najważniejsze parametry wejściowe to:

formula – symboliczny opis modelu
data – dane treningowe

Wynikiem działania funkcji jest schemat drzewa decyzyjnego wraz z opisem jego węzłów oraz dane o jego wielkości.

2.5 Metoda k najbliższych sąsiadów

Metoda k najbliższych sąsiadów zaliczana jest do grupy algorytmów tzw. leniwego uczenia. Nie tworzy ona żadnej reprezentacji danych w postaci modelu, a szuka rozwiązania dopiero w momencie pojawienia się przykładu do klasyfikacji. Aby zastosować metodę najbliższego sąsiada konieczne jest przedstawienie obiektów w n-wymiarowej przestrzeni po czym umieszczenie w niej obiektu testowanego. Klasyfikacja sprowadza się do sprawdzenia, do jakiej klasy należy obiekt najbliższy obiektowi testowanemu. Jeżeli wybrany został wariant metody z k sąsiadów to najpierw konieczne jest rozstrzygnięcie, która klasa dominuje wśród nich.

Algorytm 5 Algorytm k najbliższych sąsiadów

1. Oblicz odległości klasyfikowanego przykładu od przykładów ze zbioru treningowego.
2. Znajdź k najbliższych sąsiadów.
3. Sklasyfikuj przykład na podstawie klas sąsiadów.

Podstawową zaletą algorytmu kNN jest jego prostota. Ma on jednak szereg wad, do których należy długi czas obliczeń w przypadku licznych zbiorów treningowych, duże wymagi pamięciowe oraz konieczność wstępnej normalizacji danych. Język R oferuje tutaj funkcję kNN.

```
knn(train, test, cl, k = 1, prob = FALSE,
algorithm=c("kd_tree", "cover_tree", "brute"))
```

Jego najważniejsze parametry wejściowe to:

train – zbiór trenujący
test – testowany przykład
k – liczba sąsiadów

3 Opis badań

3.1 Planowane eksperymenty

dokładność dla różnych danych - czy któryś z wybranych algorytmów jest lepszy od innych, czy lepsze jest grupowanie czy klasyfikacja

miary nieprawidłowości z artykułu zamiast standardowych odległości przy badaniu danych testowych

$$CBLOF(p) = \begin{cases} |C_i| \cdot min(d(p, C_j)), & \text{jeśli } C_i \in SC, \text{ gdzie } p \in C_i \text{ oraz } C_j \in LC \\ |C_i| \cdot d(p, C_j), & \text{jeśli } C_i \in LC, \text{ gdzie } p \in C_i \end{cases} \quad (2)$$

$$u - CBLOF(p) = \begin{cases} min(d(p, C_j)), & \text{jeśli } p \in SC, \text{ gdzie } C_j \in LC \\ d(p, C_i), & \text{jeśli } p \in C_i \in LC \end{cases} \tag{3}$$

$$distance_{avg}(C) = \frac{\sum_{i \in C} d(i, C)}{|C|} \tag{4}$$

$$LDCOF(p) = \begin{cases} \frac{min(d(p,C_j))}{distance_{avg}(C_j}, & \text{jeśli } p \in C_i \in SC, \text{ gdzie } C_j \in LC \\ \frac{d(p,C_j)}{distance_{avg}(C_j}, & \text{jeśli } p \in C_i \in LC \end{cases} \tag{5}$$

czas wykonania?

3.2 Zbiory danych

wybór atrybutów przed dokonaniem grupowania (możliwe ograniczenie do mniejszego podzbioru z uzasadnieniem), wstępne przetwarzanie (funkcje tworzące model i klasyfikujące nie muszą same robić przekształceń)

atrybuty brakujące (NA) - sposób w jaki będziemy sobie z nimi radzić

decyzja o atrybucie klasy (np. ostatni atrybut)

przekształcenie na dane numeryczne już we własnej funkcji

opis wybranych zbiorów danych

3.3 Dobór parametrów algorytmów

które parametry będą znaczące opisane jest razem z algorytmami - co R przyjmuje i co głównie będziemy mu podawać, więc chyba nie ma sensu powtarzać (ewentualnie wyciągnąć wszystkie te parametry do tego podrozdziału)

raczej będziemy robić na standardowych parametrach, chyba, że sprawdzimy wcześniej, że któreś parametry dają lepsze wyniki i zastosujemy do nich konkretnie (uwzględniamy, że testy będą tylko na pewnych, wybranych danych)

co do parametrów, które będziemy podawać można napisać jakieś założenia ewentualnie

3.4 Ocena jakości

confusion matrix + podstawowe miary dokładności (accuracy, precision)

4 Kwestie otwarte

...

Literatura

[1] `http://wazniak.mimuw.edu.pl/images/8/86/ED-4.2-m11-1.01.pdf`