Szymon Wojciechowski, Szymon Wilk

Generator sztucznych danych wielowymiarowych: projekt i implementacja (Raport Badawczy RB-16/14)

Instytut Informatyki, Politechnika Poznańska, 2014

1. Wprowadzenie

W raporcie opisano projekt, implementacje i sposób użycia nowej wersji generatora sztucznych danych przeznaczonego do tworzenia wielowymiarowych i wieloklasowych zbiorów danych. Tworząc nowy generator wykorzystano doświadczenia związane z jego pierwszą wersją zrealizowaną w ramach pracy magisterskiej [1]. Pierwsza wersja generatora była ograniczona do zbiorów dwuwymiarowych i dwuklasowych i nie uwzględniała wszystkich rozważanych typów obiektów. Pozwoliła ona jednak na przeprowadzenie serii eksperymentów opisanych m.in. w [3].

Dla nowego generatora zdefiniowano następującą listę wymagań:

— generowanie zbiorów wielowymiarowych i wieloklasowych (z praktycznego punktu widzenia stosowane są 3-4 klasy oraz 3-10 atrybutów warunkowych odpowiadających poszczególnym wymiarom),
— definiowanie klas składających się z jednego lub wielu obszarów (regionów) – meta-kul oraz meta-kostek z możliwością wskazania klas „wypełniających" dostępną przestrzeń między regionami (takie wypełnienia zazwyczaj związane z klasami większościowymi),
— możliwość różnicowania względnego rozkładu obiektów między regionami oraz kontrolowania „gęstości" obiektów wewnątrz poszczególnych regionów (wybór jednego z dwóch możliwych rozkładów obiektów – jednostajnego i normalnego),
— generowanie obiektów wszystkich rozważanych typów [2]:
  — safe – obiekty „bezpieczne" leżące wewnątrz poszczególnych regionów i posiadające jednorodne sąsiedztwo składające się głównie z obiektów tej samej klasy,
  — borderline[1] – obiekty brzegowe leżące na obrzeżu poszczególnych regionów i przemieszane z obiektami z innych klas,
  — rare – obiekty tworzące małe „wyspy" (składające się z 2-3 obiektów) leżące daleko od regionów tworzących daną klasę, odpowiadające one rzadkim, ale poprawnym obserwacjom,
  — outlier – pojedyncze obiekty leżące daleko od innych obiektów z tej samej klasy,odpowiadające obserwacjom „odstającym",
— generowanie par zbiorów uczący-testujący z zachowaniem położenia obiektów typu rare i outlier (w poszczególnych parach zbiorów powinny one występować w zbliżonych lokalizacjach),
— implementacja w języku Java pozwalająca na zapis danych w formacie ARFF wykorzystywanym w środowisku WEKA.

2. Sposób uruchomienia generatora

Generator nie posiada specjalizowanego interfejsu użytkownika – jego wywołanie odbywa się z linii poleceń i wygląda następująco:

```
> java -jar DataGenSW -config <pliku-konfiguracyjny>
```

W powyższym przykładzie zakłada się, że plik `DataGenSW.jar` oraz dodatkowe biblioteki (umieszczone w podkatalgu `./lib`) znadują się w aktualnym katalogu. Jeśli zostały one zapisane w innej lokalizacji, wówczas należy podać pełną ścieżkę dostępu do tego pliku, np.

```
> java -jar \Projects\Generator\DataGenSW -config flower3-2d.conf
```

[1] W implementacji stosowany jest często skrócony termin border.

W przypadku poprawnego działania generator wyświetli krótki raport z listą wygenerowanych plików (z przykładu poniżej oraz kolejnych usunięto nazwy pakietów z poszczególnych wpisów logu aby zachować zwięzłość prezentacji):

```
> java -jar ..\dist\DataGenSW.jar -config paw3-2d.conf
[main] INFO Generator - Generating data set(s)
[main] INFO Generator - Pass 1...
[main] INFO Generator - Learning set
[main] INFO ARFFWriter - Saving file paw3-2d.arff...
```

Jeśli w pliku konfiguracyjnym pojawi się błąd, wówczas zgłoszony zostaje wyjątek i wyświetlany jest komunikat o błędzie. Podobna sytuacja ma miejsce, jeśli wystąpi błąd podczas generowania obiektów (np. generator z uwagi na przyjęte ustawienia nie będzie w stanie rozmieścić niektórych typów obiektów). Poniżej przedstawiono błąd wynikający z niewłaściwej liczby wymiarów w definicji domyślnego rozmiaru regionu (3 wartości zamiast dwóch):

```
> java -jar ..\dist\DataGenSW.jar -config paw3-2d.conf
defaultRegion.radius: Invalid number of entries (3)
Exception in thread "main" java.lang.IllegalArgumentException:
        defaultRegion.radius: Invalid number of entries (3)
        at org.apache.commons.lang3.Validate.isTrue(Validate.java:155)
        ...
```

Istnieje możliwość napisania części parametrów z pliku konfiguracyjnego podając ich nowe wartości w linii poleceń (te ostatnie nadpisują lub uzupełniają wartości – teoretycznie istnieje więc możliwość podania wszystkich parametrów w linii poleceń bez konieczności tworzenia pliku konfiguracyjnego, jednak może być to uciążliwe z praktycznego punktu widzenia). Funkcjonalność taka pozwala na łatwe wprowadzanie tymczasowych modyfikacji bez konieczności tworzenia nowych plików konfiguracyjnych:

```
> java -jar DataGenSW.jar -config <plik-konfiguracyjny> -Dparam1=wart1
    -Dparam2=wart2 -Dparam3=wart3 ...
```

W przykładzie poniżej całkowita liczba obiektów oraz ich rozkład w klasach zdefniowany w pliku zostanie nadpisany przez ustawienia z linii poleceń. Uwaga – podając w taki sposób parametry nie należy wstawiać spacji po opcji `-D`, wokół znaku '=' ani w samych wartościach:

```
> java -jar ..\dist\DataGenSW.jar -config paw3-2d.conf
        -Dexamples=1800 -DclassRatio=1:2
```

Domyślnie, informacje wyświetlane w trakcie pracy generatora są bardzo ograniczone. Jeśli zaistnieje potrzeba dokładnego wglądu w poszczególne etapy działania, a w szczególności w ostatecznie uwzględnione ustawienia oraz w wyznaczone bezwzględne liczby obiektów w poszczególnych klasach, regionach, itp., wówczas można włączyć dokładne raportowanie podając dodatkowy przełącznik `-Dorg.slf4j.simpleLogger.defaultLogLevel=debug`, który musi zostać umieszczony przed parametrem `-jar`. Wyniki jego zastosowania są następujące:

```
> java -Dorg.slf4j.simpleLogger.defaultLogLevel=debug -jar ..\dist\DataGenSW.jar
        -config paw3-2d-5pairs.conf -DlearnTestPairs=1 -Dexamples=1000 -DlearnTestRatio=1:1

[main] DEBUG GeneratorSettings - Final configuration:
attributes = 2
classes = 2
examples = 1000
names.attributes = A1, A2
names.classes = MIN, MAJ
names.decision = DEC
learnTestRatio = 1.0:1.0
learnTestPairs = 1
fileName.learn = paw3-2d-learn-%d.arff
fileName.test = paw3-2d-test-%d.arff
minOutlierDistance = 1.0
classRatio = 1.0:9.0
class.1.exampleTypeRatio = 40.0:20.0:30.0:10.0
class.1.regions = 3
class.1.region.1.weight = 1.0
class.1.region.1.shape = CIRCLE
class.1.region.1.distribution = UNIFORM
class.1.region.1.center = [5.0, 5.0]
class.1.region.1.radius = [2.0, 1.0]
class.1.region.1.borderZone = 1.0
class.1.region.1.noOutlierZone = 1.5
```

```
class.1.region.1.rotations = [(1, 2) --> 45.0]
class.1.region.2.weight = 1.0
class.1.region.2.shape = CIRCLE
class.1.region.2.distribution = UNIFORM
class.1.region.2.center = [-5.0, 3.0]
class.1.region.2.radius = [2.0, 1.0]
class.1.region.2.borderZone = 1.0
class.1.region.2.noOutlierZone = 1.5
class.1.region.2.rotations = [(1, 2) --> -45.0]
class.1.region.3.weight = 1.0
class.1.region.3.shape = CIRCLE
class.1.region.3.distribution = UNIFORM
class.1.region.3.center = [0.0, -5.0]
class.1.region.3.radius = [2.0, 1.0]
class.1.region.3.borderZone = 1.0
class.1.region.3.noOutlierZone = 1.5
class.1.region.3.rotations = []
class.2.exampleTypeRatio = 100.0:0.0:0.0:0.0
class.2.regions = 1
class.2.region.1.weight = 1.0
class.2.region.1.shape = INTEGUMENTAL
class.2.region.1.distribution = UNIFORM
class.2.region.1.center = [0.0, 0.0]
class.2.region.1.radius = [10.0, 10.0]
class.2.region.1.borderZone = 1.0
class.2.region.1.noOutlierZone = 1.5
class.2.region.1.rotations = []
[main] DEBUG GeneratorSettings - Distributing all examples into learning and testing parts:
                                 #1000 (1.0:1.0) ==> #500.0:500.0
[main] DEBUG GeneratorSettings - Learning part
[main] DEBUG GeneratorSettings -   Distributing examples into classes:
                                   #500.0 (1.0:9.0) ==> #50.0:450.0 (initial attempt)
[main] DEBUG GeneratorSettings -     Distributing class 1 into examples types:
                                     #50 (40.0:20.0:30.0:10.0) ==> #19.0:10.0:16.0:5.0
[main] DEBUG GeneratorSettings -     Distributing class 2 into examples types:
                                     #450 (100.0:0.0:0.0:0.0) ==> #450.0:0.0:0.0:0.0
[main] DEBUG GeneratorSettings -   Distributing examples into classes:
                                   #500.0 (1.0:9.0) ==> #50.0:450.0 (after possible correction)
[main] DEBUG GeneratorSettings -     Distributing class 1 into examples types:
                                     #50 (40.0:20.0:30.0:10.0) ==> #19.0:10.0:16.0:5.0
[main] DEBUG GeneratorSettings -       Ratio of SAFE:BORDER examples: 0.6666:0.3333
[main] DEBUG GeneratorSettings -       Distributing region 1 into examples types: #9 ==> 6.0:3.0
[main] DEBUG GeneratorSettings -       Distributing region 2 into examples types: #10 ==> 6.0:4.0
[main] DEBUG GeneratorSettings -       Distributing region 3 into examples types: #10 ==> 7.0:3.0
[main] DEBUG GeneratorSettings -     Distributing class 2 into examples types:
                                     #450 (100.0:0.0:0.0:0.0) ==> #450.0:0.0:0.0:0.0
[main] DEBUG GeneratorSettings -       Ratio of SAFE:BORDER examples: 1.0:0.0
[main] DEBUG GeneratorSettings -       Distributing region 1 into examples types: #450 ==> 450.0:0.0
[main] DEBUG GeneratorSettings - Testing part
[main] DEBUG GeneratorSettings -   Distributing examples into classes:
                                   #500.0 (1.0:9.0) ==> #50.0:450.0 (initial attempt)
[main] DEBUG GeneratorSettings -     Distributing class 1 into examples types:
                                     #50 (40.0:20.0:30.0:10.0) ==> #19.0:10.0:16.0:5.0
[main] DEBUG GeneratorSettings -     Distributing class 2 into examples types:
                                     #450 (100.0:0.0:0.0:0.0) ==> #450.0:0.0:0.0:0.0
[main] DEBUG GeneratorSettings -   Distributing examples into classes:
                                   #500.0 (1.0:9.0) ==> #50.0:450.0 (after possible correction)
[main] DEBUG GeneratorSettings -     Distributing class 1 into examples types:
                                     #50 (40.0:20.0:30.0:10.0) ==> #19.0:10.0:16.0:5.0
[main] DEBUG GeneratorSettings -       Ratio of SAFE:BORDER examples: 0.6666:0.3333
[main] DEBUG GeneratorSettings -       Distributing region 1 into examples types: #9 ==> 6.0:3.0
[main] DEBUG GeneratorSettings -       Distributing region 2 into examples types: #10 ==> 6.0:4.0
[main] DEBUG GeneratorSettings -       Distributing region 3 into examples types: #10 ==> 7.0:3.0
[main] DEBUG GeneratorSettings -     Distributing class 2 into examples types:
                                     #450 (100.0:0.0:0.0:0.0) ==> #450.0:0.0:0.0:0.0
[main] DEBUG GeneratorSettings -       Ratio of SAFE:BORDER examples: 1.0:0.0
[main] DEBUG GeneratorSettings -       Distributing region 1 into examples types: #450 ==> 450.0:0.0
[main] INFO Generator - Generating data set(s)
[main] INFO Generator - Pass 1...
[main] INFO Generator - Learning set
[main] INFO ARFFWriter - Saving file paw3-2d-learn-1.arff...
[main] INFO Generator - Testing set
[main] INFO ARFFWriter - Saving file paw3-2d-test-1.arff...
```

3. Parametry konfiguracyjne

Jak wspomniano w poprzednim rozdziale, działanie generatora kontrolowane jest przez zestaw parametrów przekazanych w pliku konfiguracyjnym oraz w linii poleceń (parametry mogą być specyfikowane w obu lokalizacjach – wartości z linii poleceń nadpisują wartości z pliku konfiguracyjnego). Przykładowy plik znajduje się na listingu 1. Pozwala on na wygenerowanie trójwymiarowego zbioru *paw3*, w którym

obiekty z klasy mniejszościowej zostały oznaczone etykietami wskazującymi na ich typ (safe, borderline, rare oraz outlier).

Listing 1. Przykładowy plik konfiguracyjny pozwalający na wygenerowanie jednego pliku

```
# paw3-2d
attributes = 2
classes = 2

classRatio = 1:9
minOutlierDistance = 1

defaultRegion.weight = 1
defaultRegion.distribution = U
defaultRegion.borderZone = 1
defaultRegion.noOutlierZone = 1.5
defaultRegion.shape = C
defaultRegion.radius = 2, 1

defaultClass.exampleTypeRatio = 100:0:0:0

class.1.exampleTypeRatio = 40:20:30:10
class.1.regions = 3
class.1.region.1.center = 5,5
class.1.region.1.rotation = 1, 2, 45

class.1.region.2.center = -5,3
class.1.region.2.rotation = 1, 2, -45

class.1.region.3.center = 0,-5

class.2.regions = 1
class.2.region.1.shape = I
class.2.region.1.center = 0,0
class.2.region.1.radius = 10, 10

examples=1500
fileName=paw3-2d.arff

exampleTypeLabels.classes = 1
```

Znacznie poszczególnych parametrów jest następujące:

`attributes` liczba atrybutów warunkowych ($\geq 1 \wedge \leq 40$) – górne ograniczenie wynika z zastosowanego generatora wektorów quasi-losowych,[2]; wartość wymagana,

`classes` liczba klas decyzyjnych (≥ 1); wartość wymagana,

`names.classes` lista nazw poszczególnych klas decyzyjnych (łańcuchy znakowe oddzielone przecinkami), lista musi zawierać tyle elementów, ile wynosi liczba klas (`classes`); wartość opcjonalna – jeśli nie zostanie podana, poszczególnym klasom zostaną nadane nazwy 1, 2, ... `,n` (gdzie `n==classes`),

`names.attributes` lista nazw atrybutów warunkowych (łańcuchy znakowe oddzielone przecinkami), lista musi zawierać tyle elementów, ile wynosi liczba atrybutów (attributes); wartość opcjonalna – jeśli nie zostanie podana, poszczególnym klasom zostaną nadane nazwy X1, X2, ...,Xm (gdzie m == `attributes`),

`names.decision` nazwa atrybutu decyzyjnego; wartość opcjonalna – jeśli nie zostanie podana, przyjęta zostanie nazwa D,

`classRatio` rozkład obiektów w poszczególnych klasach (lista wartości oddzielonych znakiem :), poszczególne wartości muszą być liczbami rzeczywistymi≥ 0, nie muszą się one sumować do 1.0 lub 100.0 – generator wyznacza proporcję oraz przelicza liczby obiektów w poszczególnych klasach, lista musi zawierać tyle elementów, ile jest klas (`classes`); wartość wymagana,

[2] W obecnej wersji generatora (biblioteka Apache Commons Math) jet ona ograniczona do 40, jednak w razie potrzeby można obejść ten limit przygotowując własny zestaw wartości początkowych.

`minOutlierDistance` minimalna odległość między obiektami typu rare lub outlier należącymi do jednej klasy(≥ 0)[3], wyrażona w sposób bezwzględny (odległość euklidesowa między obiektami); odległość ta nie jest sprawdzana dla obiektów typu rare występujących w jednej „wyspie" – mogą wystąpić bardzo blisko siebie; wartość wymagana,

`defaultRegion.*` zgrupowanie domyślnych ustawień dla regionu, które mogą być współdzielone przez regiony definiujące poszczególne klasy – dzięki czemu nie trzeba ich wielokrotnie powtarzać, wszystkie te ustawienia domyślne mogą być nadpisane w definicjach specyficznych regionów. Poza tym wartości wymagane charakteryzujące obszar mogą pojawić się tylko w definicji domyślnej – nie trzeba powtarzać ich w kolejnych definicjach. Z drugiej strony, jeśli wartość wymagana nie pojawi się w definicji domyślnej, wówczas musi się ona pojawić w definicji konkretnego regionu,

`defaultRegion.weight` waga regionu określająca jaka część obiektów z danej klasy powinna znaleźć się w regionie (> 0), suma wag w poszczególnych regionach nie musi się sumować do 1.0 lub 100.0 – generator wyznacza proporcję oraz przelicza liczby obiektów w poszczególnych regionach; wartość wymagana,

`defaultRegion.shape` kształt regionu, dopuszczalne wartości to C, R i I – C (*circle*) oznacza meta-kulę, R (*rectangle*) meta-kostkę, a I (*integumental*) jest szczególnym przypadkiem meta-kostki, który wypełnia przestrzeń obiektami ze wskazanej klasy (w praktyce taki obszar jest używany do wypełnienia przestrzeni danych obiektami z klasy większościowej). Obiekty z regionu typu I nie mogą pojawiać się w „rdzeniu" regionów typu C i R (proces rozmieszczania obiektów jest kontrolowany przez generator), poza tym regionu typu I nie można obracać[4] i nie może on zawierać obiektów typu borderline, rare lub outlier (problemy z interpretacją typu obiektu) – innymi słowy, regiony typu I można definiować dla tych klas, które zawierają tylko i wyłącznie obiekty typu safe; wartość wymagana,

`defaultRegion.distribution` rozkład obiektów w danym regionie, dopuszczalne wartości to U i N – U oznacza rozkład jednostajny (*uniform*)[5], natomiast N to rozkład normalny; wartość opcjonalna – jeśli nie zostanie podana, przyjmowany jest rozkład jednostajny (U). W przypadku rozkładu normalnego (`N`) istnieje możliwość podania liczby odchyleń standardowych mieszczących się w regionie poprzez dodanie wartości liczbowej (> 0) po znaczniku `N` (np. „`N, 3`"). Liczba odchyleń standardowych jest opcjonalna – jeśli nie zostanie jawnie określona, przyjmowana jest wielkość 1.0. Ustawienie dotyczy tylko obiektów typu safe – obiekty typu borderline są generowane z wykorzystaniem rozkładu jednostajnego (U). Ustawienie to jest ignorowane w przypadku regionów o kształcie I – w tym wypadku stosowany jest zawsze rozkład `U`,

`defaultRegion.center` punkt środkowy regionu, podany jako lista liczb rzeczywistych (> 0), przy czym długość listy musi być równa liczbie atrybutów (`atttributes`); wartość wymagana,

`defaultRegion.border` sposób przeliczania wymiarów regionu dla obszarów zawierających obiekty typu safe oraz borderline, dopuszczalne wartości to fixed oraz auto. Wartość fixed oznacza, że generator wykorzystuje wartości parametrów radius oraz borderZone (opisane poniżej) podane przez użytkownika i nie są one w żaden sposób modyfikowane. Natomiast wartość auto oznacza, że generator ustala wymiary obszarów dla obiektów typu safe oraz borderline automatycznie na podstawie rozkładu typów obiektów (parametr exampleTypeRatio opisany poniżej) – jako wartości bazowe wykorzystywane są rozmiary regionu (radius), a sposób ich przeliczania przedstawiono w punkcie 4; wartość opcjonalna – domyślnie zakładana jest wartość fixed,

`defaultRegion.radius` rozmiar regionu podany jako promienie w przypadku region typu C oraz połowy długości boków dla obszarów typu R i I (szczegóły przedstawiono na rys. 1 i 2), podany jako lista liczb rzeczywistych (> 0), przy czym długość listy musi być równa liczbie atrybutów (`atttributes`); jeśli parametr border ma wartość fixed (brak przeliczania) obszar ten stanowi „rdzeń" regionu – tutaj umieszczane są obiekty typu safe, w przeciwnym razie (border=auto) w obszarze tym znajdują się obiekty safe oraz borderline (szczegóły w punkcie 4); wartość wymagana,

`defaultRegion.borderZone` bezwzględna szerokość strefy granicznej obszaru, w której mogą być rozmieszczone obiekty typu borderline (≥ 0, przy czym musi być to wartość > 0, jeśli w klasie,

[3] Uwaga – dla pewnych konfiguracji (duża liczba obiektów rare i outlier oraz duże odległości między nimi) generator może nie być w stanie wygenerować obiektów. Jeśli po pewnej liczbie prób (obecnie 10000) nie uda się wygenerować obiektów, wówczas generator kończy działanie. Aby rozwiązać taki problem, należy zmniejszyć wartość parametru `minOutlierDistance`.

[4] To ograniczenie może zostać zniesione w kolejnych wersjach generatora.

[5] W celu unikania „dziur" zamiast typowych generatorów liczby pseudo-losowych stosowane są sekwencje Haltona.

do której należy obszar, pojawiają się obiekty borderline); jedna wartość dla wszystkich atrybutów warunkowych/wymiarów; wartość wymagana jeśli border=fixed, w przeciwnym razie jest ignorowana,

`defaultRegion.noOutlierZone` bezwzględna szerokość strefy, w której nie mogą pojawić się obiekty typu rare oraz outlier (≥ 0), strefa ta jest rozmieszczona wokół strefy z obiektami typu borderline – wzajemne ułożenie obu stref przedstawiono na rys. 1 i 2;jedna wartość dla wszystkich atrybutów warunkowych/wymiarów; wartość wymagana,

`defaultRegion.rotation` obrót regionu zdefiniowany przez wskazanie dwóch wymiarów (unikalnych indeksów – liczonych od 1 – dwóch atrybutów warunkowych) oraz konta obrotu wyrażonego w stopniach. Dla jednego regionu można zdefiniować wiele obrotów – albo poprzez jeden wpis zawierający kilka trójek liczb, albo poprzez wiele wpisów; wartość domyślna – jeśli obroty nie zostaną zdefiniowane, wówczas regiony nie są przekształcane,

`defaultClass.*` zgrupowanie domyślnych ustawień dla wszystkich klas decyzyjnych – rozwiązanie analogiczne dla `defaultRegion`, którego celem jest uniknięcie wielokrotnego definiowania tych samych ustawień,

`defaultClass.exampleTypeRatio` rozkład typów obiektów w klasie zdefiniowany jako czwórka liczb rzeczywistych (≥ 0, przy czym ich suma musi być > 0) oddzielonych znakiem :, kolejne liczby podpowiadają obiektom typu safe, borderline, rare i outlier. Podane liczby nie muszą się sumować się do 1.0 lub 100.0 – generator wyznacza proporcje, a następnie ustala odpowiednie liczby obiektów. Jeśli klasa ma zawierać regiony typu l, wówczas wszystkie należące do niej obiekty muszą być typu safe; wartość opcjonalna – jeśli nie zostanie podana, przyjmowany jest rozkład 100:0:0:0 (tylko obiekty typu safe),

`class.`i`.region.`j`.*` definicja regionu j-tego ($\geq 1 \wedge \leq$`class.`i`.regions`) dla klasy i-itej ($\geq 1 \wedge \leq$`classes`), która nadpisuje oraz uzupełnia ustawienia domyślne wprowadzone dla `defaultRegion`,

`class.i.regions` liczba regionów (> 0) w klasie i-tej ($\geq 1 \wedge \leq$`classes`); wartość wymagana,

`class.i.*` definicja i-itej klasy ($\geq 1 \wedge \leq$`classes`), która nadpisuje oraz uzupełnia ustawienia domyślne wprowadzone dla `defaultClass`,

`examples` liczba obiektów w całym zbiorze danych, tzn. we wszystkich klas decyzyjnych i regionach (> 0); wartość wymagana,

`exampleTypeLabels.classes` wskazanie na klasy decyzyjne, dla których w wygenerowanym zbiorze mają pojawić się etykiety wskazujące na typ przykładu[6] (etykiety tworzone są przez sklejenie nazwy klasy z nazwą typu obiektu, np. MIN-BORDER), wskazanie klas następuje poprzez podanie indeksów – rozpoczynających się od 1 ($\geq 1 \wedge \leq$`classes`) – tych klas, dla których mają być włączone indeksy; wartość opcjonalna – jeśli nie zostanie podana, wówczas w wynikowym zbiorze zostaną zapisane tylko etykiety klas decyzyjnych (bez typów poszczególnych obiektów), a zatem generujące dane na potrzeby typowych eksperymentów obliczeniowych należy pominąć to ustawienie,

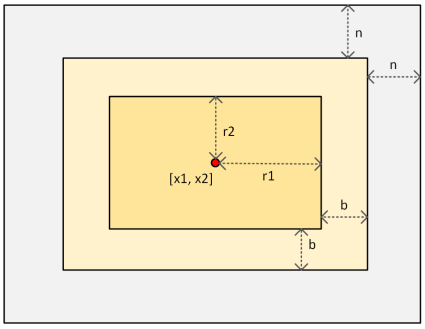
`fileName` nazwa pliku (w formacie ARFF), do którego zostaną zapisane wygenerowane dane, wartość wymagana, jeśli wygenerowany ma być tylko jeden plik (parametr `learnTestPairs`, opisany w dalszej części tekstu, nie został podany).

Generator pozwala też na wygenerowanie zestawów par plików z obiektami uczącymi i testującymi. Nie jest to jednak prosty schemat walidacji krzyżowej warstwowej – podczas generowania plików testowych generator uwzględnia położenie obiektów typu rare i outlier w zbiorach uczących i rozmieszcza obiekty testowej w zbliżonych obszarach. Dzięki temu została nie ma potrzeby ręcznej modyfikacji plików, jak to miało miejsce w przypadku pierwszej wersji generatora. Na listingu 2 przedstawiono konfigurację pozwalającą na uzyskanie 5 par zbiorów dla dwuwymiarowego kształtu *paw3*, użytego również w poprzednim przykładzie. Ponieważ definicja poszczególnych obszarów nie uległa zmianie, na listingu przedstawiono tylko zmodyfikowane części pliku. Pojawiają się w niej nowe parametry, opisane poniżej:

`learnTestRatio` rozkład obiektów między częścią uczącą, a testująca wyrażony jako para liczb (> 0[7]) oddzielonych oddzielonych znakiem :, poszczególne wartości nie muszą sumować się do 1.0 ani do 100.0 – generator wylicza proporcje i wyznacza liczby obiektów; wartość opcjonalna – jeśli nie zostanie podana, generowane są tylko przykłady uczące (tzn. przyjmowany jest rozkład 100:0),

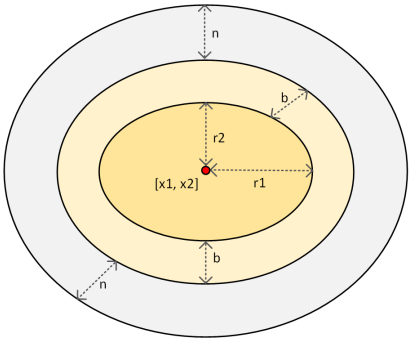
[6] Etykiety te wykorzystywane są do bardziej szczegółowej wizualizacji oraz weryfikacji wygenerowanych zbiorów.

[7] Dopuszczalne jest także generowanie samych zbiorów uczących, a zatem w podanym rozkładzie mogą wystąpić tylko przykłady uczące.




```
class.i.region.j.shape = R
class.i.region.j.center = x1, x2
class.i.region.j.radius = r1, r2
class.i.region.j.borderZone = b
class.i.region.j.noOutlierZone = n
```

Rysunek 1. Region typu meta-kostka (R) i wybrane parametry opisujące jego położenie i wielkość (border=fixed)




```
class.i.region.j.shape = C
class.i.region.j.center = x1, x2
class.i.region.j.radius = r1, r2
class.i.region.j.borderZone = b
class.i.region.j.noOutlierZone = n
```

Rysunek 2. Region typu meta-kula (C) i wybrane parametry opisujące jego położenie i wielkość (border=fixed)

`learnTestPairs` liczba par zbiorów uczący-testujący (≥ 1); wartość opcjonalna – jeśli nie zostanie podana, generowany jest tylko zbiór uczący, poza tym, jeśli zostanie podany parametr `learnTestRatio`, a `learnTestPairs` zostanie pominięty, wówczas generator zgłosi błąd i przerwie działanie,

`fileName.learn` szablon nazwy zbioru z obiektami uczącymi, powinien zawierać on znacznik `%d`, który będzie zastąpiony przez kolejne indeksy par 1...`learnTestPairs`; wartość obowiązkowa, jeśli podano parametr `learnTestRatio`, w przeciwnym razie może zostać pominięta,

`fileName.test` szablon nazwy zbioru z obiektami testującymi, powinien zawierać on znacznik `%d`, który będzie zastąpiony przez kolejne indeksy par 1...`learnTestPairs`; wartość obowiązkowa, jeśli podano parametr `learnTestRatio`, w przeciwnym razie może zostać pominięta.

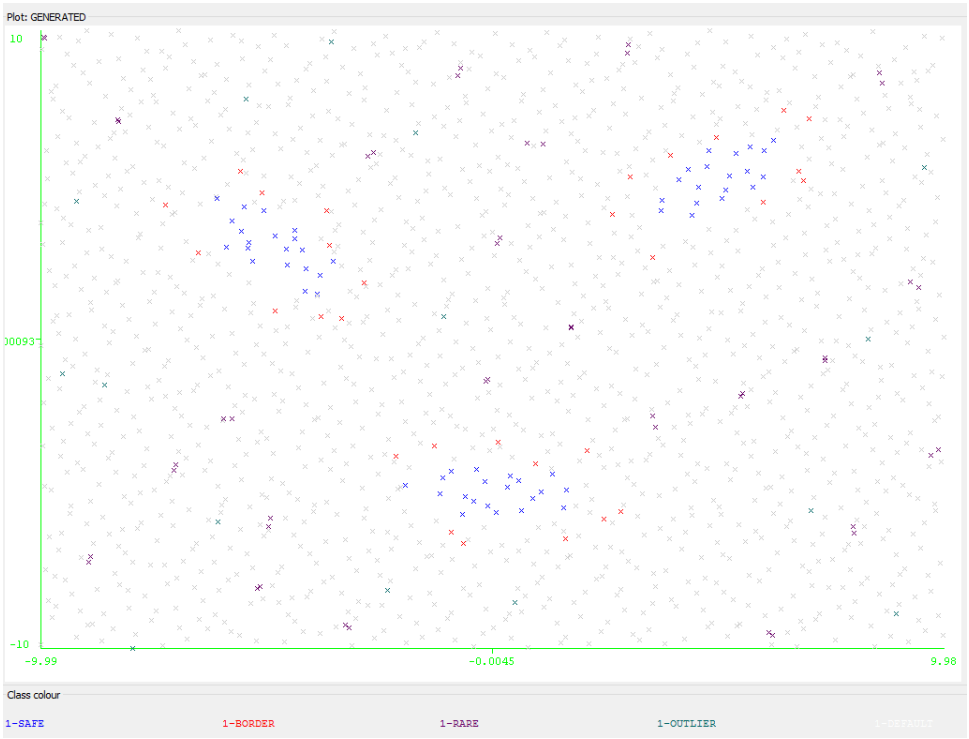
Na rys. 4oraz 5 przedstawiono odpowiednio zbiór uczący i testujący z pierwszej wygenerowanej pary. Widać na nich, że lokalizacje obiektów typu rare i outlier są zbieżne, dzięki czemu unika się sytuacji, gdzie obiekty testowe z danej klasy pojawiają się w takiej lokalizacji, w której nie wystąpiły żadne obiekty uczące. Oczywiście, możliwe jest wykonanie tradycyjnego podziału krzyżowego – należy wygenerować jeden plik, a następnie podzielić go na zbiory uczące i testowe za pomocą zewnętrznego narzędzia (np. filtra StratifiedRemoveFolds dostępnego w środowisku WEKA).

Listing 2. Przykładowy plik konfiguracyjny pozwalający na wygenerowanie 5 par plików uczący-testujący

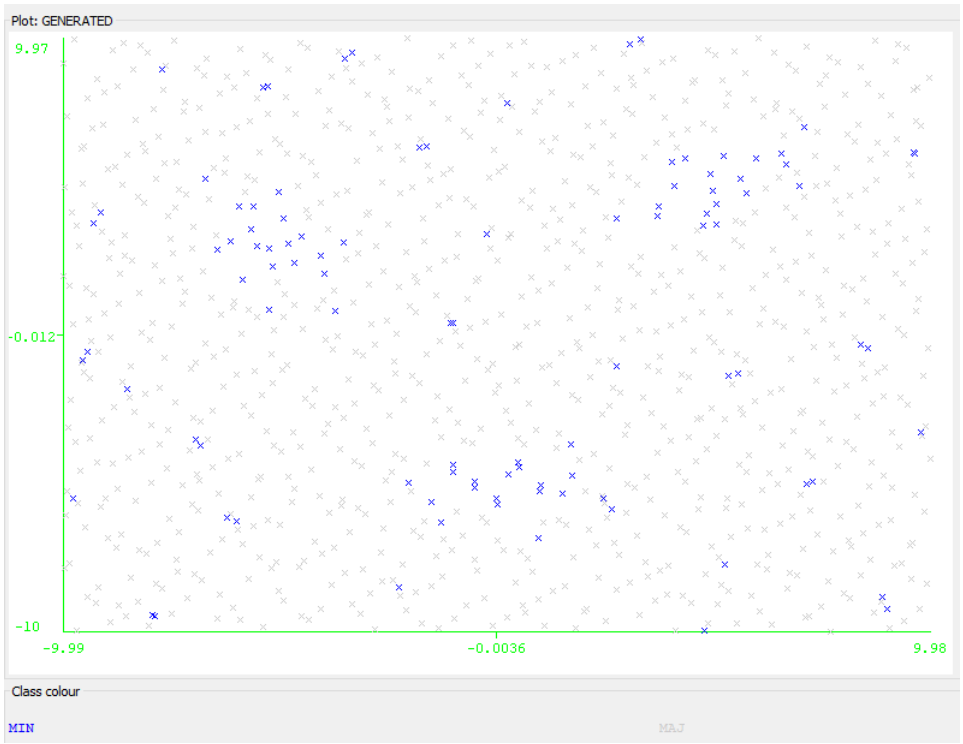
```
# paw3-2d-5-pairs
# [...]
# Większosc ustawien jest taka sama, jak dla paw3-2d, zmiany poniżej

learnTestRatio = 2:1
learnTestPairs = 5
fileName.learn = paw3-2d-learn-%d.arff
fileName.test = paw3-2d-test-%d.arff
```

Wreszcie na listingu 3przedstawiono konfigurację pozwalającą na wygenerowanie trójwymiarowego kształtu *flower*, a jego wizualizacja za pomocą programu PlotViz3[8] znajduje się na rys. 6. W celu zwiększenia czytelności dla obiektów z klasy większościowej zostały wygenerowane etykiety wskazujące ich typ, a obiekty z klasy mniejszościowej zostały tymczasowo wyłączone.

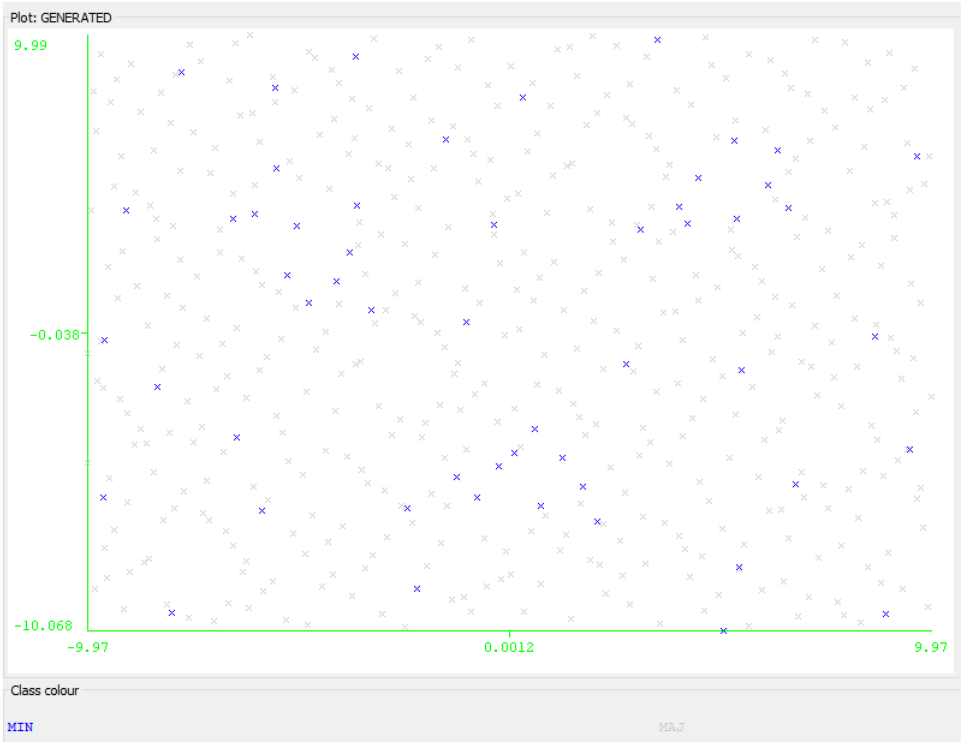
[8] http://salsahpc.indiana.edu/pviz3/



Rysunek 3. Wizualizacja zbioru danych uzyskanego dla konfiguracji z listingu 1 – dzięki etykietom wyróżniono poszczególne typy obiektów w klasie mniejszościowej (oznaczonej jako 1).



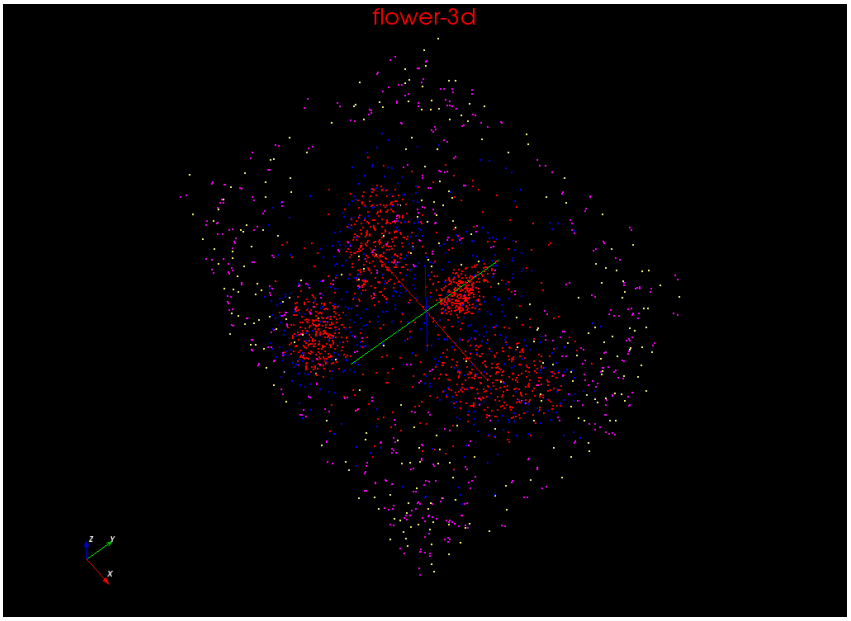
Rysunek 4. Wizualizacja uczącego zbioru danych uzyskanego dla konfiguracji z listingu 2



Rysunek 5. Wizualizacja testowego zbioru danych uzyskanego dla konfiguracji z listingu 2

Listing 3. Plik konfiguracyjny dla kształtu *flower*

```
# flower-3d
attributes = 3
classes = 2

names.classes = MIN, MAJ
names.attributes = A1, A2, A3
names.decision = CLASS
classRatio = 1:3
minOutlierDistance = 0.3

defaultRegion.weight = 1
defaultRegion.distribution = U
defaultRegion.borderZone = 0.5
defaultRegion.noOutlierZone = 0.5
defaultRegion.shape = C
defaultRegion.radius = 2, 1, 1

defaultClass.exampleTypeRatio = 100:0:0:0

class.1.exampleTypeRatio = 50:20:20:10
class.1.regions = 5
class.1.region.1.center = -3, 1.85, 0
class.1.region.1.radius = 2, 1, 2
class.1.region.1.rotation = 1, 2, -45

class.1.region.2.center = 0, 2.8, 0
class.1.region.2.radius = 1, 2, 2
class.1.region.2.distribution = N, 3

class.1.region.3.center = -1.5, -1.5, 0
class.1.region.3.radius = 1, 1, 2
class.1.region.3.distribution = N
```



Rysunek 6. Wizualizacja kształtu *flower* z listingu 3

```
class.1.region.4.center = 3, 1.85, 0
class.1.region.4.radius = 2, 1, 2
class.1.region.4.rotation = 1, 2, 45

class.1.region.5.center = 0, 1.5, 0
class.1.region.5.radius = 5.5, 4.5, 5

class.2.regions = 1
class.2.region.1.shape = I
class.2.region.1.center = 0, 1.5, 0
class.2.region.1.radius = 5.5, 4.5, 5

examples=10000
fileName=flower-3d.arff

exampleTypeLabels.classes = 1
```

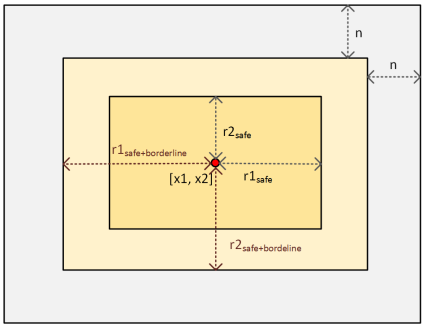
4. Przeliczanie rozmiarów obszarów dla obiektów safe i borderline

Automatyczne przeliczane rozmiarów obszarów regionu przeznaczonych dla obiektów typu safe oraz borderline w zależności od rozkłady typów obiektów w danej klasie pozwala na utrzymanie ich stałej gęstości. Unika się w ten sposób sytuacji, w której znaczne zmniejszenie udziału obiektów typu safe doprowadzi do „rozrzedzenia" obszaru z tymi obiektami (podobna uwaga dotyczy obiektów typu borderline). Można temu oczywiście zapobiec modyfikując jawnie opis wybranych regionów, jednak takie rozwiązanie może być uciążliwe w sytuacji, kiedy konieczne jest wygenerowanie wielu różnych wariantów zbioru danych. Przyjęty sposób przeliczania rozmiarów opiera się na następujących założeniach:

1. stosunek objętości obszaru z obiektami safe oraz borderline do objętości całego regionu powinien odpowiadać stosunkowi liczby obiektów obu tych typów w regionie do liczby wszystkich obiektów w danym regionie,
2. stosunek objętości obszaru z obiektami safe do objętości obszaru z obiektami safe i borderline powinien być równy stosunkowi liczby odpowiednich obiektów w danym regionie.

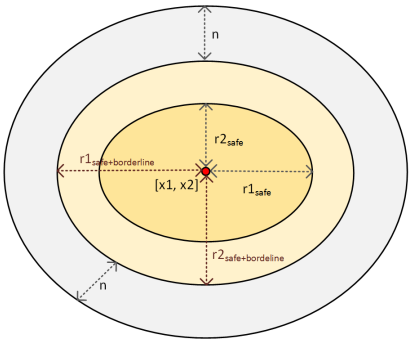
Prowadzi to do następujących zależności:

$$r_{safe+borderline} = r \cdot \sqrt[m]{\frac{n_{safe} + n_{borderline}}{n_{all}}},$$




```
class.i.region.j.shape = R
class.i.region.j.center = x1, x2
class.i.region.j.radius = r1, r2
~~class.i.region.j.borderZone = b~~
class.i.region.j.noOutlierZone = n
```

Rysunek 7. Region typu meta-kostka (R) i wybrane parametry opisujące jego położenie i wielkość (border=auto)




```
class.i.region.j.shape = C
class.i.region.j.center = x1, x2
class.i.region.j.radius = r1, r2
~~class.i.region.j.borderZone = b~~
class.i.region.j.noOutlierZone = n
```

Rysunek 8. Region typu meta-kula (C) i wybrane parametry opisujące jego położenie i wielkość (border=auto)

$$r_{safe} = r_{borderline} \cdot \sqrt[m]{\frac{n_{safe}}{n_{safe} + n_{borderline}}},$$

gdzie m – liczba atrybutów (wymiarów), $r_{safe+borderline}$ – rozmiar obszaru z obiektami safe oraz borderline, r_{safe} – rozmiar obszaru z obiektami safe, r – rozmiar danego regionu (parametr radius), n_{all}, n_{safe} i $n_{borderline}$ – liczba wszystkich obiektów, obiektów safe oraz borderline w danym regionie.

Na rysunkach 7 i 8 przedstawiono graficzną interpretację parametrów opisujących położenie i obszar regionu przy założeniu automatycznego przeliczania promieni (parametr borderZone jest ignorowany, dlatego został on przekreślony).

5. Główne pakiety i klasy generatora

Generator został zaimplementowany w języku Java (JDK 1.8), jego kod źródłowy został podzielony na 4 pakiety opisane poniżej. W każdym pakiecie wyszczególniono najważniejsze klasy, które biorą bezpośredni udział w procesie generowania danych.

1. `pl.put.poznan.cs.idss.generator` – klasy podstawowe „opakowujące" funkcjonalność generatora i dające do niej dostęp
   a) `Generator` – klasa główna (z metodą main) generująca dane zgodnie z przekazanymi ustawieniami i zapisująca je do plików,
   b) `ARFFWriter` – klasa obsługująca zapis danych do plików w formacie ARFF.
2. `pl.put.poznan.cs.idss.generator.factories` – klasy „fabryki" odpowiedzialne za tworzenie generatorów obiektów dla specyficznych typów obiektów oraz rozkładów danych (w ramach regionów)
   a) `RandomGeneratorFactory` – klasa tworząca odpowiednie generatory punktów (klasa `RandomGenerator` i pochodne, opisane poniżej) dla różnych typów obiektów i różnych typów rozkładów,
   b) `RegionGeneratorFactory` – klasa tworząca generatory obiektów związane z regionami (klasa `RegionGenerator` i pochodne) dla poszczególnych kształtów regionów,
   c) `DataSetGeneratorFactory` – klasa tworząca nowy obiekt `DataSetGenerator` na podstawie opisów poszczególnych regionów oraz opisów „wysp" zwierających obiekty typu rare i outlier,

d) `AdditionalPointGeneratorFactory` – klasa tworząca nowe obiekty `AdditionalOutlierPointGenerator`, które są odpowiedzialne za generowanie obiektów typu rare i outlier,
e) `RegionGenerators` – klasa tworząca i przechowująca listę generatorów dla poszczególnych regionów definiujących zbiór danych,

3. `pl.put.poznan.cs.idss.generator.generation` – klasy odpowiedzialne za generowanie obiektów poszczególnych typów oraz należących do specyficznych regionów (większość z nich jest tworzona za pomocą klas z pakietu `.factories`)
   a) `RandomGenerator` – generator obiektów (punktów) zgodnych z pewnym rozkładem, jest to klasa bazowa dla `GaussianDistributionGenereator` oraz `LowDiscrepancySequenceGenerator`, które wykorzystują rozkład normalny oraz sekwencję Haltona symulującą rozkład jednostajny. Dostępna jest również klasa `UniformDistributionGenerator` wykorzystująca „zwykłe" liczby pseudo-losowe, jednak nie jest ona aktualnie wykorzystywana,
   b) `RegionGenerator` – generator obiektów tworzących specyficzny region, w zależności od kształtu regionu wykorzystuje jeden lub dwa generatory punktów – jeden do generowania obiektów typu safe stanowiących „rdzeń" obszaru oraz drugi do generowania obiektów typu borderline (więcej informacji w kolejnym rozdziale),
   c) `OutlierGenerator` – generator obiektów typu rare i outlier (w odróżnieniu od generatorów regionów, dla zbioru danych tworzony jest tylko jeden taki generator),
   d) `DataSetGenerator` – generator całego zbioru danych, który wykorzystuje obiekty klas `RegionGenerator` oraz `OutlierGenerator` do uzyskania obiektów należących do poszczególnych regionów oraz do „wysp".
4. `pl.put.poznan.cs.idss.generator.settings` – klasy odpowiedzialne za odczyt, przechowywanie i obsługę ustawień konfiguracyjnych dla generatora
   a) `GeneratorSettings` – klasa odpowiedzialna za odczyt ustawień z pliku konfiguracyjnego i z linii poleceń,
   b) `ParameterExtractor` – klasa służąca do obsługi poprzedniej składni ustawień konfiguracyjnych, obecnie stanowi opakowanie wokół klasy `GeneratorSettings` i pozwala istniejącym klasom z pakietów `factories` i `generation` na odczyt nowych ustawień.

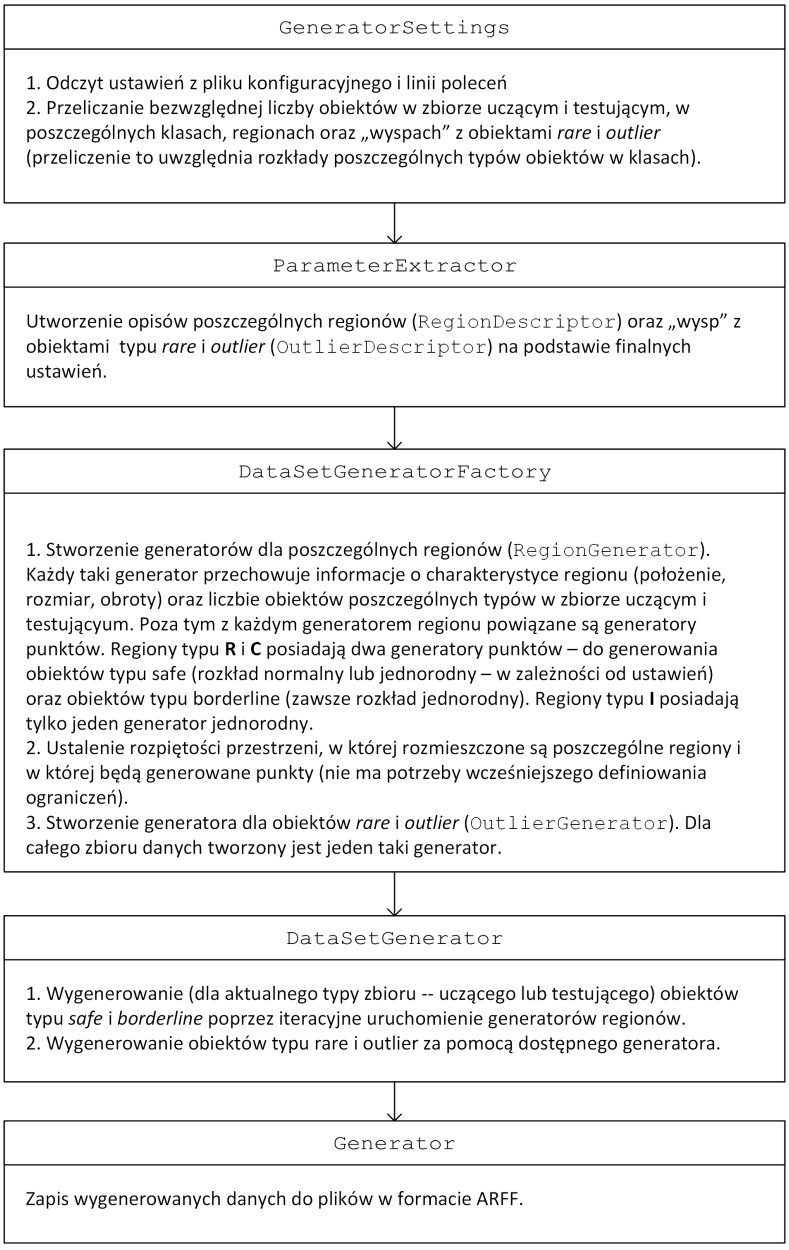
6. Schemat działania generatora

Na rys. 9 przedstawiono schemat działania generatora. Opisuje on poszczególne fazy procesu generacji danych oraz wskazuje na główne klasy realizujące te fazy. Obecnie dwie pierwsze fazy związane są z przetwarzaniem ustawień i wynikają ze zmian wprowadzanych do implementacji – w kolejnej wersji generatora mogą zostać one połączone. Warto też zauważyć, że obecnie generator zapisuje obiekty do pliku zaraz po ich wygenerowaniu i nie przechowuje dłużej w pamięci. W sytuacji, w której wymagane byłoby dodatkowe przetwarzanie albo długotrwałe utrzymywanie obiektów w pamięci (np. przelosowywanie i stopniowe przenoszenie do strumienia danych), konieczna byłaby niewielka modyfikacja klasy `Generator` (dodanie dodatkowych pól – kolekcji służących do przechowywania obiektów)

7. Podsumowanie

Nowa wersja generatora przedstawiona w niniejszym opracowaniu spełnia wszystkie postawione na wstępie wymagania i pozwala na uzyskanie złożonych zbiorów danych. Generator zostanie teraz przetestowany w serii eksperymentów obliczeniowych badający wpływ różnego rodzaju „niedoskonałości" (modelowanych przez różne rozkłady poszczególnych typów obiektów oraz różne układy regionów tworzących poszczególne klasy) na wpływ wybranych klasyfikatorów symbolicznych (drzewa decyzyjne, reguły), jak i statystycznych (naiwny klasyfikator Bayesa, sieci neuronowe RBF).

W ramach dalszego rozwoju generatora rozważane jest dodanie następujących funkcji:
— definiowanie bardziej złożonych kształtów regionów poprzez łączenie meta-kul i meta-kostek,
— obsługa atrybutów symbolicznych,
— generowanie szumu, zarówno na atrybutach warunkowych, jak i decyzyjnych,
— generowanie danych niekompletnych.



Rysunek 9. Schemat działania generatora

Podziękowania

Autorzy dziękują za wsparcie udzielone przez Narodowe Centrum Nauki w ramach grantu DEC-2013/-11/B/ST6/00963.

Literatura

[1] K. Kałużny. Metody dekompozycji w analizie niezrównoważonych liczebnie danych. praca magisterska, 2009.

[2] K. Napierała and J. Stefanowski. Types of minority class examples and their influence on learning classifiers from imbalanced data. *J. Intell. Inform. Syst.*, 2015 to appear.

[3] K. Napierała, J. Stefanowski, and Sz. Wilk. Learning from imbalanced data in presence of noisy and borderline examples. In *Proceedings of the 7th International Conference RSCTC 2010*, volume 6086 of *LNAI*, pages 158–167. Springer, 2010.