Sieci Społeczne i Eksploracja Danych

Sprawozdanie z projektu

Skład grupy:
Wojciech Bożek
Korneliusz Caputa
Grzegorz Cerowski
Michał Dydak
Paweł Garbacik
Michał Głowacki
Agnieszka Gondek
Grzegorz Holewa
Marek Kojder
Michał Kowal
Dawid Mazur
Krystian Pypłacz
Anna Romik
Dominik Samociuk
Paweł Szopa

1. Wstęp

1.1. Opis projektu

Projekt z przedmiotu Sieci Społeczne i Eksploracja Danych zakłada analizę zadanego zbioru danych pod kątem klasyfikacji za pomocą dostarczonych algorytmów. Dane wejściowe zostały przygotowane przez firmę zewnętrzną i zostały wykorzystane w pracy naukowej, o którą projekt został oparty. Zestaw algorytmów zaawansowanej alalizy danych składa się m.in. z algorytmów undersamplingu i oversamplingu, metody SMOTE (Synthetic Minority Oversampling Technique), metody WRF (Weighted Random Forest) oraz metody BRF (Balanced Random Forest).

1.2. Geneza projektu

Dane wejściowe wykorzystane w projekcie zostały zebrane przez firmę Lincoln Labs. Przed dziewięć tygodni gromadzono surowe dane TCP z typowej sieci LAN, jednocześnie przeprowadzając na sieć szereg ataków. Następnie na zebranych danych zostały zagregowane odpowiednie połączenia. Wykorzystując wiedzę na temat ataków sieciowych, do każdego rekordu dołączone zostały dodatkowe informacje, co pozwoliło podzielić ataki na 38 różnych typów i 4 główne kategorie.

Oprogramowanie do wykrywania włamań sieciowych chroni sieć komputerową przed nieautoryzowanymi użytkownikami. Zadaniem algorytmu detekcji jest zbudowanie modelu prognostycznego (tj. klasyfikatora) zdolnego do rozróżniania „złych" połączeń, zwanych włamaniami lub atakami, od „dobrych", normalnych połączeń.

W 1998 roku forma Lincoln Labs przygotowała program rządowy pod nazwą Intrusion Detection Evaluation, którego celem było badanie oraz ocena wykrywania włamań sieciowych. Specjalnie dla tego programu dostarczono dane symulujące ruch w wojskowym otoczeniu sieciowym. Zbiór treningowy to ok. cztery GB skompresowanych danych oraz siedem tygodni ruchu w sieci, co w sumie dało ok. pięć milionów rekordów. Z kolei dane testowe reprezentowane są przez zrzut rekordów z dwutygodniowej pracy sieci.

Wartym odnotowania jest fakt, że dane testowe nie mają tego samego rozkładu prawdopodobieństwa co dane treningowe i zawierają pewne specyficzne typy ataków, które nie występują w zbiorze treningowym. To sprawia, że zadanie klasyfikacji zyskuje na realistyczności. Niektórzy eksperci potwierdzają, że większość nowych typów ataków jest jedynie modyfikacją istniejących, co pozwala na ich rozpoznanie za pomocą analizy starszych danych. Zestaw danych zawiera w sumie 24 rodzaje ataków treningowych i 14 rodzajów występujących tylko w danych testowych.

2. Spotkania projektowe

2.1. 10 grudnia 2013 r.

Pierwsze spotkanie projektowe dotyczyło podstawowych aspektów przygotowania do wykonania projektu, tj. opisu danych, sposobu wykonania własnych operatorów dla oprogramowania RapidMiner oraz opisu metod klasyfikacji. Sekcja nr 1 w składzie Agnieszka Gondek oraz Michał Dydak przygotowała krótkie wyjaśnienie czego otrzymane dane dotyczą, jakich wyników powinno się oczekiwać oraz dlaczego nie są one zalecane do stosowania przez osoby zajmujące się analizą danych. Zaznaczono również, że plik z danymi w obecnej formie jest niemożliwy do przetworzenia i na jego podstawie należy spreparować nowe dane wejściowe, co wymaga określonych ram czasowych.

Sekcja nr 2 w składzie Paweł Garbacik i Marek Kojder przedstawiła metodologie udostępnioną przez twórców oprogramowania RapidMiner, które w zamyśle miały umożliwić przygotowanie własnych operatorów. Sekcja położyła nacisk na zamknięty dostęp do kodu obecnej wersji programu, wysoce ograniczone wsparcie dla wersji OpenSource oraz na nieaktualne i niekompletne rozwiązania proponowane przez użytkowników.

2.2. 17 grudnia 2013 r.

Kolejne spotkanie projektowe dotyczyło ponownego zapoznania się z tworzeniem operatorów dla oprogramowania RapodMiner oraz opisu metod klasyfikacji danych. Sekcja nr 3 w składzie Anna Romik, Michał Głowacki oraz Grzegorz Holewa przedstawiła podstawowe pojęcia stosowane przy klasyfikacji, takie jak zbiór danych niezbalansowanych, oversamling oraz undersampling. Wyjaśniono, że dane niezbalansowane to takie, w których liczebność jednej klasy jest znacznie większa od liczebności pozostałych klas, a oversampling i undersampling to odpowiednio: zwiększenie liczebności klasy zdominowanej oraz zmniejszenie liczebności klasy dominującej.

Sekcja nr 4 w składzie Wojciech Bożek, Dawid Mazur, Krystian Pypłacz i Paweł Szopa przygotowała prezentację na temat metody SMOTE (Synthetic Minority Oversampling Technique). Odniesiono się do wcześniejszych informacji dotyczących oversamplingu oraz danych niezbalansowanych oraz wyjaśniono, że metoda polega na syntetycznym generowaniu dodatkowych przykładów z klasy zdominowanej na podstawie odległości euklidesowej od k najbliższych sąsiadów rozpatrywanego obiektu oraz czynnika losowego.

Sekcja nr 2 w składzie Marek Kojder i Paweł Garbacik przedstawiła wszystkie możliwe sposoby tworzenia operatorów w RapidMinerze. Niestety żadna z nich nie okazała się trafna i nie generowała poprawnego rezultatu. Ustalono z prowadzącym, że należy rozważyć wykonanie projektu w innym środowisku (Weka, Matlab).

Sekcje nr 5 i 6 wyjaśniły, że zaproponowane metody wykorzystujące lasy losowe są teoretycznymi rozważaniami autorów pracy naukowej, na której jest oparty projekt i sposób implementacji zależy wyłącznie on grupy. Zaznaczono, że teoria została podparta przykładami, których dobór i sposób badania nie zostały w pracy wyjaśnione i należy brać pod uwagę, że autorska implementacja grupy może nie dawać tych samych rezultatów.

2.3. 22 stycznia 2014 r.

Na spotkaniu projektowym przedstawiona została prezentacja dotycząca metody BRF (Balanced Random Forest), która wyjaśniła genezę jej powstania oraz podstawowe założenia. Przedstawiono obszar zastosowań metody (dane niezrównoważone) oraz przykład jej implementacji w oprogramowaniu RapidMiner.

Pokazano również wyniki działania operatorów undersamplingu i oversamplingu na danych wejściowych. Zarówno sekcja nr 1 jak i sekcja nr 3 wskazały na problem dużej liczby danych, która nie każda maszyna jest w stanie sobie poradzić. Zaznaczono, że konieczna jest odpowiednia konfiguracja maszyny wirtualnej Java w celu przyporządkowania odpowiedniej liczby pamięci RAM tak, aby obliczenia wykonywały się w skończonym czasie.

2.4. 29 stycznia 2014 r.

Ostatnie spotkanie projektowe zostało przeznaczone na prezentację wyników działania implementacji algorytmu SMOTE na testowych danych wejściowych. Wyniki działania algorytmu były poprawne, algorytm wykonywał oversampling na klasie zdominowanej korzystając z metody generacji sztucznych przykładów. Zaprezentowane zostały również źródła w języku Java, które w dużej mierze były oparte o istniejące rozwiązania napisane wcześniej przez sekcję nr 3.

Przygotowana została również prezentacja wyjaśniająca podstawowe zagadnienia dot. metody WRF.
Wyjaśniono pojęcie Gini Impurity oraz przedstawiono problem doboru odpowiednich wag do klas.

3. Dane wejściowe

3.1. Przedstawienie danych

Dane wejściowe pochodzą z konkursu KDD Mining Cup 1999. KDD Cup jest światowym konkursem Eksploracji Danych i Odkrywania Wiedzy organizowanym przez ACM SIGKDD - Special Interest Group on Knowledge Discovery and Data Mining. Zadaniem konkursu było zbudowanie modelu prognostycznego (tj. klasyfikatora) zdolnego do rozróżniania połączeń „złych", zwanych atakami lub włamaniami, oraz „dobrych" (normalnych) połączeń. Zestaw obejmuje różnego rodzaju ataki symulowane na wojskowym środowisku sieciowym. Są to między innymi:

- Denial of Service Attack (DoS - odmowa usługi)
- Remote to Local (R2L)
- User to Root (U2R)
- Probe

Zbiór danych wejściowych w liczbach:

- 24 treningowe rodzaje ataków
- 14 rodzajów ataków w danych testowych
- 8 050 290 rekordów (1 173 MB)
- 4 940 000 rekordów treningowych (734 MB)
- 3 110 290 rekordów testowych (430 MB)
- 42 zmienne opisu pojedynczego rekordu
- 5 zmiennych dotyczących tego samego hosta
- 13 zmiennych dotyczących tej samej usługi

3.2. Opis rekordu danych

3.2.1. 42 zmienne opisujące rekord

duration	czas połączenia w sekundach
protocol_type	rodzaj protokołu
service	rodzaj usługi
flag	połączenie normalne lub błędne
src_bytes	liczba bajtów przesłanych od źródła do celu
dst_bytes	liczba bajtów przesłanych od celu do źródła
land	1 jeżeli połączenie jest z tego samego hosta lub portu, 0 w przeciwnym wypadku
wrong_fragment	liczba błędnych fragmentów
urgens	liczba pakietów z flagą „urgent”
hot	liczba wskaźników „hot”
num_failed_logins	liczba logowań zakończonych niepowodzeniem
logged_in	1 jeżeli użytkownik zalogowany, 0 w przeciwnym wypadku
num_compromised	liczba skompromitowanych warunków
root_shell	1 jeżeli uzyskano dostęp do powłoki Root, 0 w przeciwnym wypadku
su_attempted	1 jeżeli użyto komendy “su root”, 0 w przeciwnym wypadku
num_root	liczba dostępów do konta “Root”
num_file_creations	liczba operacji tworzenia pliku
num_shells	liczba monitów powłoki
num_access_files	liczba operacji kontroli dostępu na plikach
num_outbound_cmds	liczba komend wychodzących w sesji ftp
is_host_login	1 jeżeli zalogowany na konto “hot”, 0 w przeciwnym wypadku
is_guest_login	1 jeżeli zalogowano na koncie gościa, 0 w przeciwnym wypadku
count	liczba połączeń do tego samego hosta w przeciągu ostatnich 2 sekund
class	klasa ruchu

3.2.2. 5 zmiennych dotyczących tego samego hosta

serror_rate	procent połączeń z błędem “SYN”
rerror_rate	procent połączeń z błędem “REJ”
same_srv_rate	procent połączeń do tego samego serwisu
diff_srv_rate	procent połączeń do innego serwisu
srv_count	liczba połączeń do tej samej usługi w przeciągu ostatnich 2 sekund

3.2.3. 13 zmiennych dotyczących tego samego hosta

srv_serror_rate	procent połączeń z błędem “SYN”
srv_rerror_rate	procent połączeń z błędem “REJ”
srv_diff_host_rate	procent połączeń do innego hosta
dst_host_rate	procent połączeń do tego samego hosta docelowego
dst_host_count	liczba połączeń do tego samego hosta docelowego
dst_host_srv_count	liczba połączeń do tego samego hosta docelowego i tej samej usługi
dst_host_same_srv_rate	procent połączeń do tego samego hosta docelowego i do tej samej usługi
dst_host_diff_srv_rate	procent różnych usług połączonych do danego hosta
dst_host_same_src_port_rate	procent połączeń do hosta z tym samym portem źródłowym
dst_host_srv_diff_host_rate	procent połączeń do tej samej usługi od różnych hostów
dst_host_serror_rate	procent błędnych połączeń do hosta z błędem „SYN”
dst_host_srv_serror_rate	procent błędnych połączeń do hosta I tej samej usługi z błedem „SYN”
dst_host_rerror_rate	procent błędnych połączeń do hosta z błędem „REJ”

3.2.3. 4 Rodzaje ataków

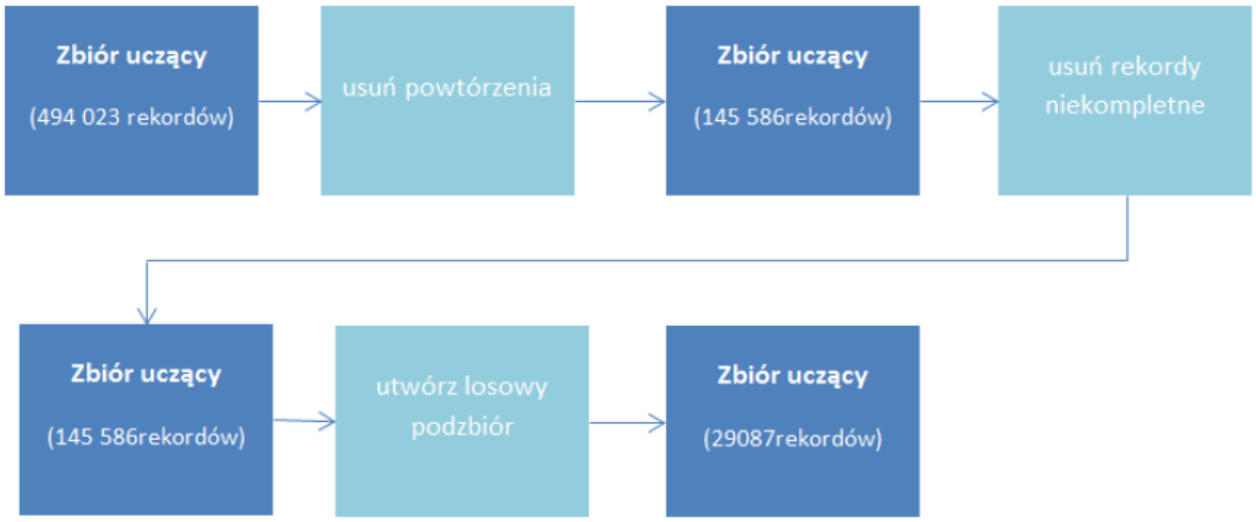
dos	smurf, pod, neptune, land, back, httptunnel
probe	ipsweep, nmap, satan, portsweep, teardrop
r2l	warezclient, phf, imap, guess_passwd, warezmaster, ftp_write, multihop, spy
u2r	buffer_overflow, loadmodule, perl, rootkit

3.3. Przygotowanie danych

Dane dostarczone z projektem okazały się zbyt obszerne dla dalszej analizy. Aby wykonać zadanie klasyfikacji, należało przeprowadzić redukcję oraz eksplorację danych, w czym pomogło oprogramowanie STATISTICA 10. Za jego pomocą wykonano:

- odczyt danych
- przypisanie danych właściwych etykiet
- usunięcie z arkusza powtarzających się rekordów
- minimalizację ryzyka wystąpienia problemu, w którym model nauczy się dobrze rozpoznawać jedynie obiekt najliczniej występujący w zbiorze uczącym
- usunięcie rekordów niekompletnych

W ramach dodatkowej redukcji danych przeprowadzono losowanie podzbiorów przypadków, odpowiednio 20% rekordów ze zbioru uczącego oraz 5% rekordów ze zbioru testowego.



Rys. 1. Proces eksploracji danych zbioru uczącego.

Po przeprowadzeniu powyższych operacji zbiór uczący zawiera 29087, a testowy 7196 rekordów.

3.4. Opis danych

3.4.1. Przygotowania

Do wczytania danych wykorzystano środowisko RapidMiner działające w otoczeniu maszyny wirtualnej Java w wersji 64-bitowej. Aby zapewnić skończoność wykonywania operacji skonfigurowano maszynę tak, aby jej zajętość pamięciowa wynosiła 3GB. W tym celu posłużono się opcją:

```
-Xmx3072m
```

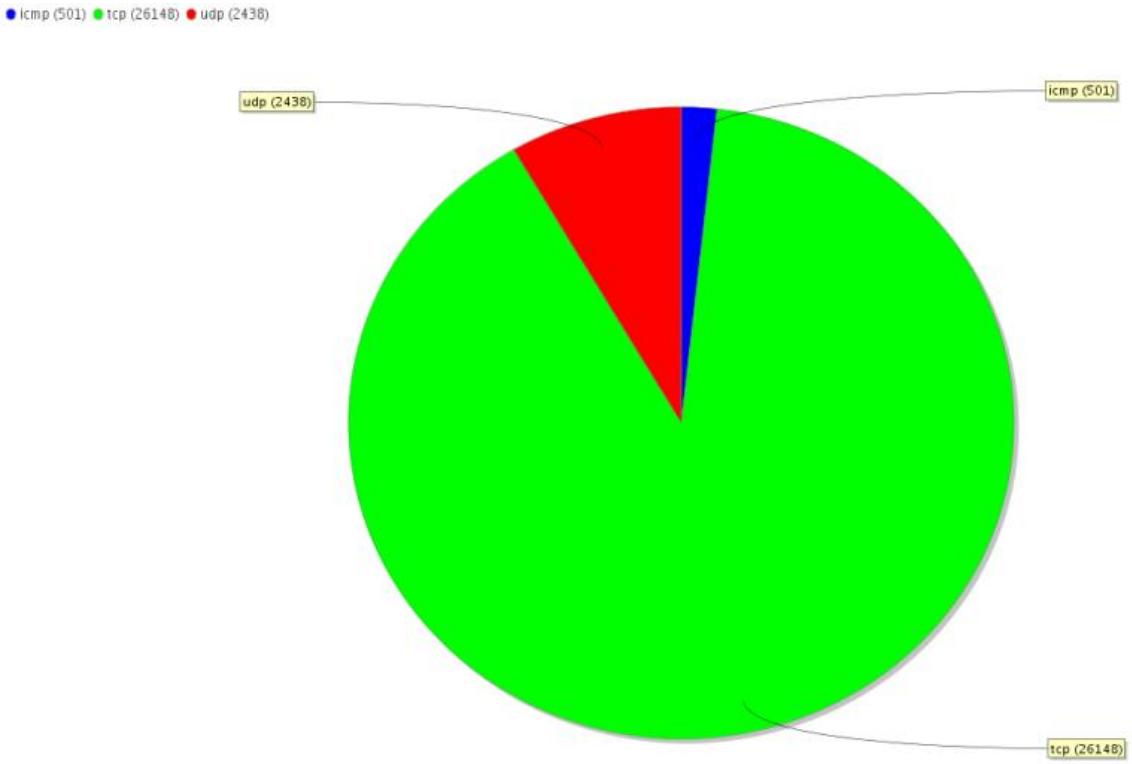
Również w oprogramowaniu RapidMiner (plik RapidMinerGUI.bat) należało wprowadzić odpowiednią opcję:

```
if "%MAX_JAVA_MEMORY%"=="" set MAX_JAVA_MEMORY=3072
```

Dane zostały załadowane za pomocą bloku Read Excel oraz zapewnionego kreatora. Proces ten po powyższej konfiguracji zajął 32 sekundy.

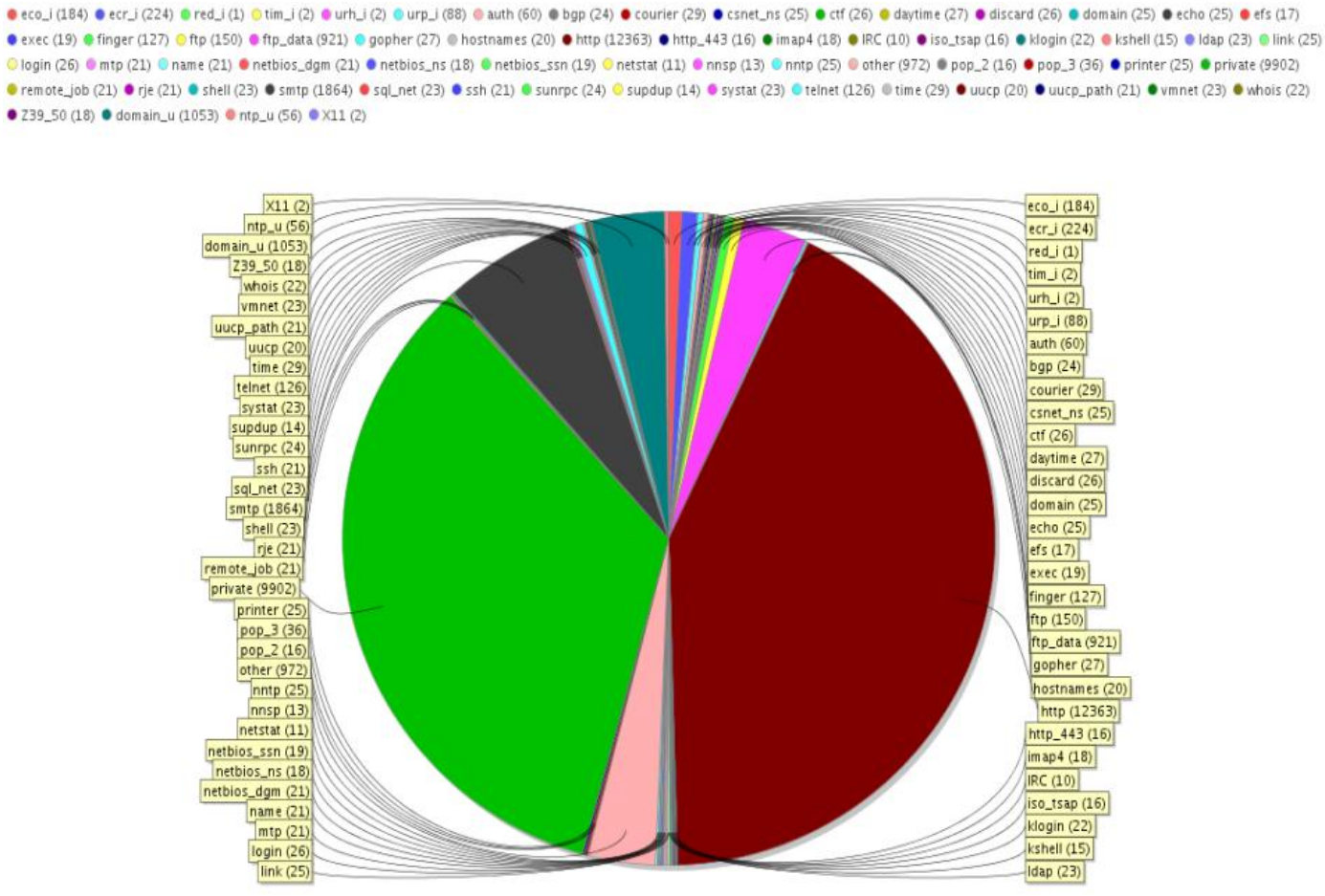
3.4.2. Prezentacja danych - zbiór treningowy

W zbiorze treningowym o liczności 29087 rekordów wyróżniono trzy protokoły sieciowe:



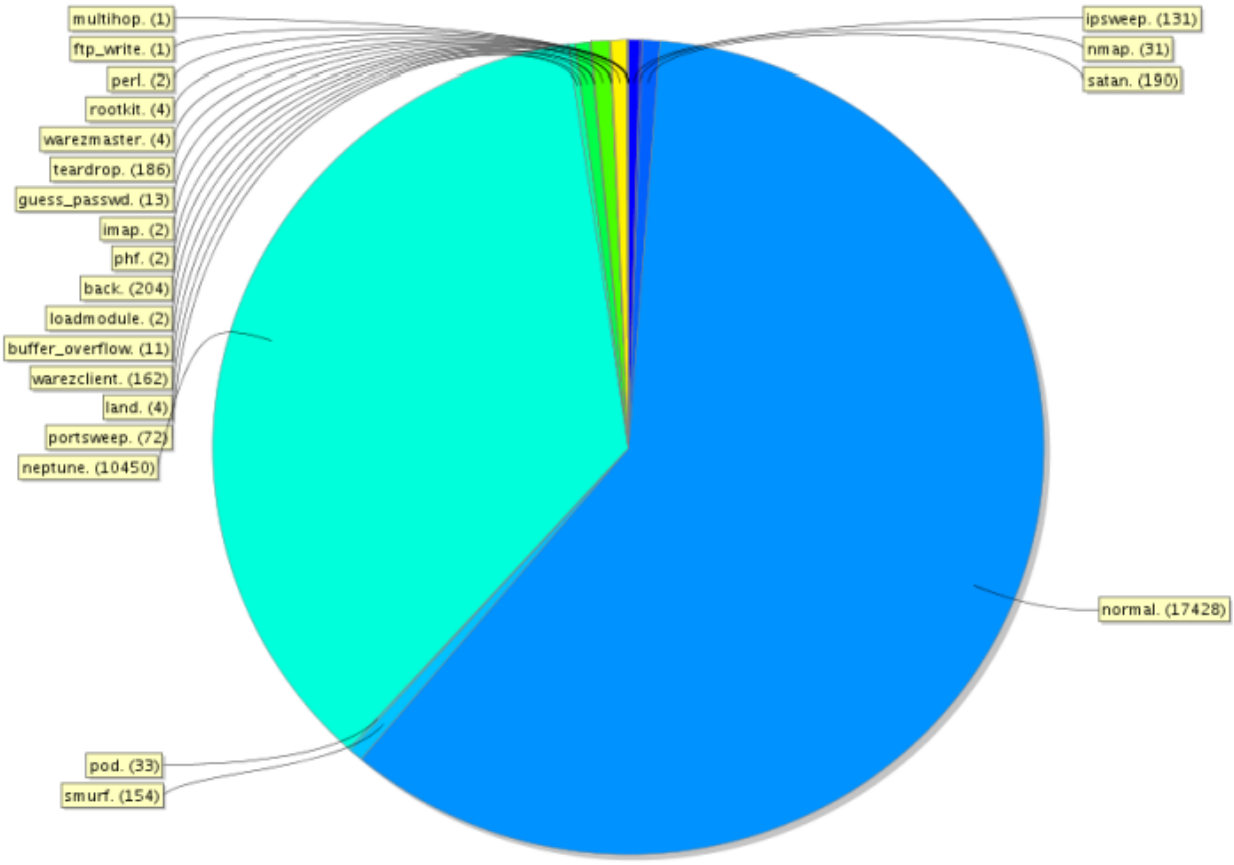
Rys. 2. Protokoły sieciowe w zbiorze uczącym.

Odnaleziono również 64 rodzaje usług, z jakich korzystali użytkownicy sieci:

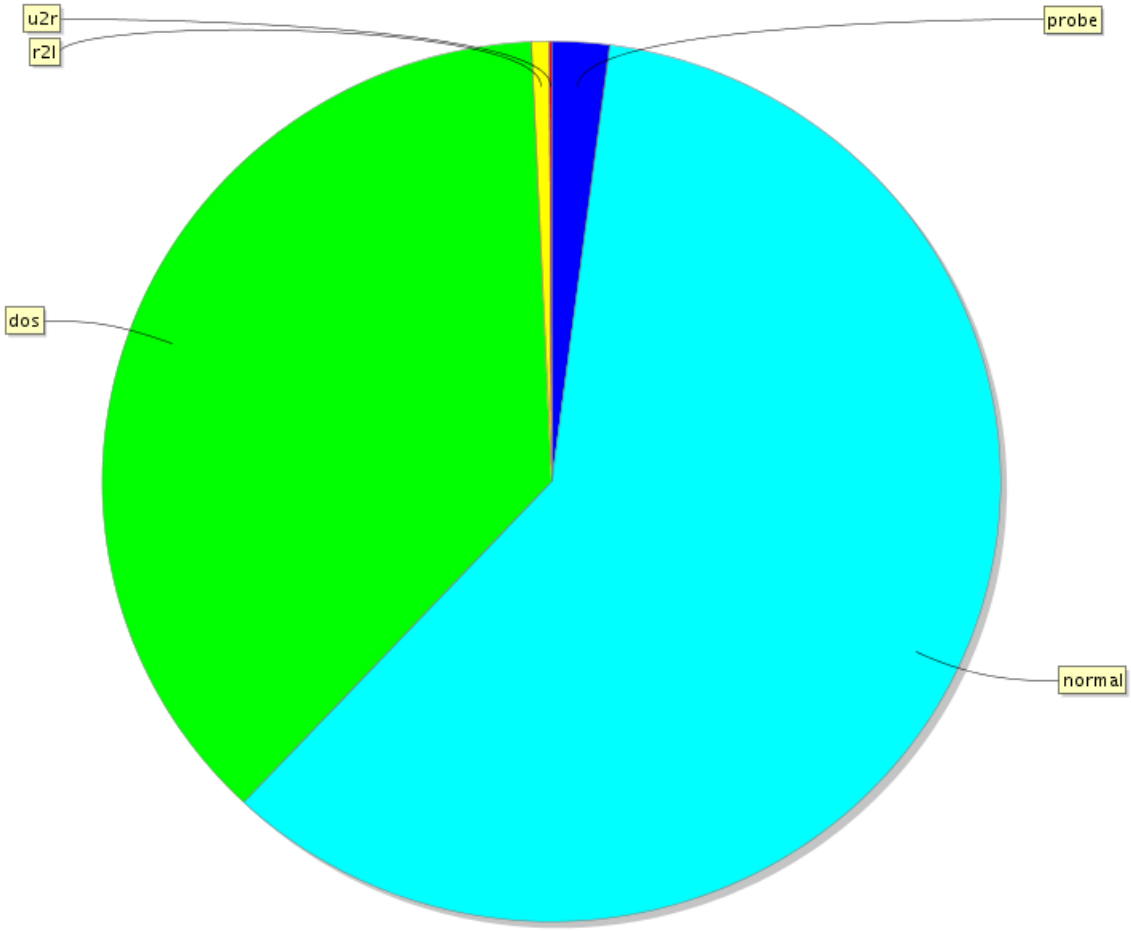


Rys. 3. Rodzaje usług sieciowych.

Dokonano również podziału pakietów na zwyczajne oraz ataki wraz z wyszczególnieniem rodzajów ataków:



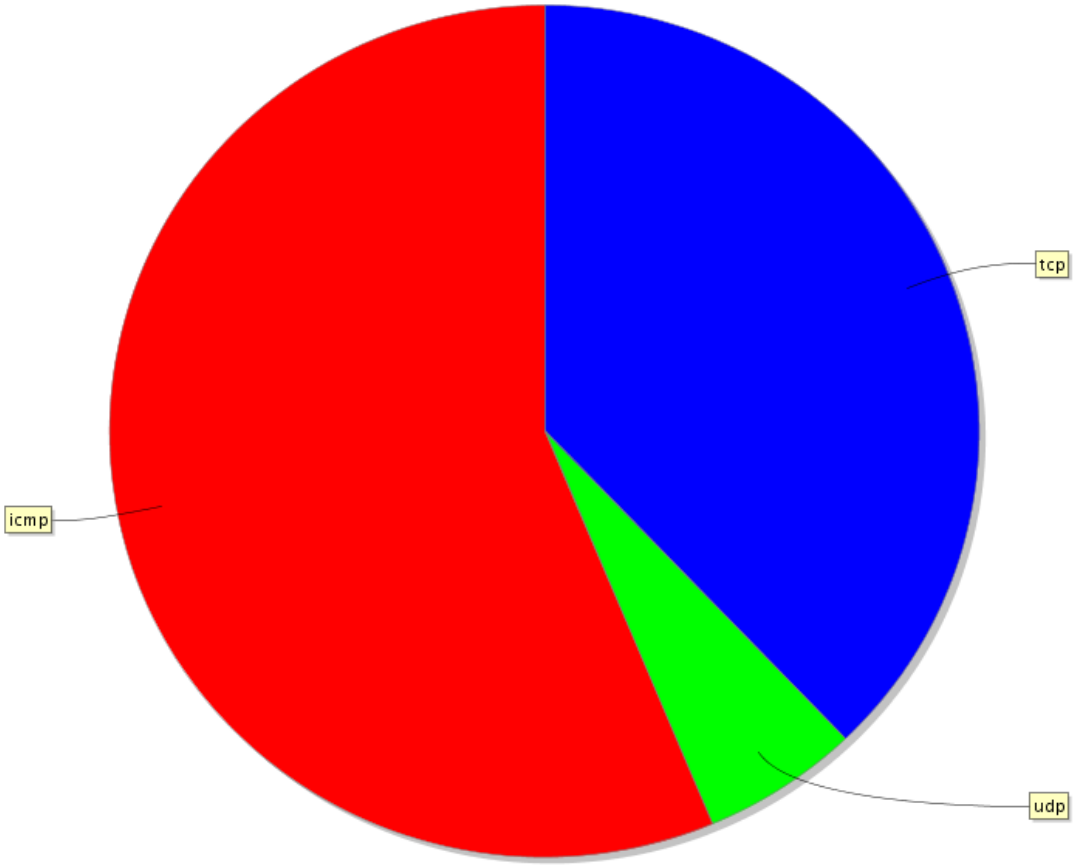
Rys. 4. Podział pakietów na zwykłe i poszczególne ataki.



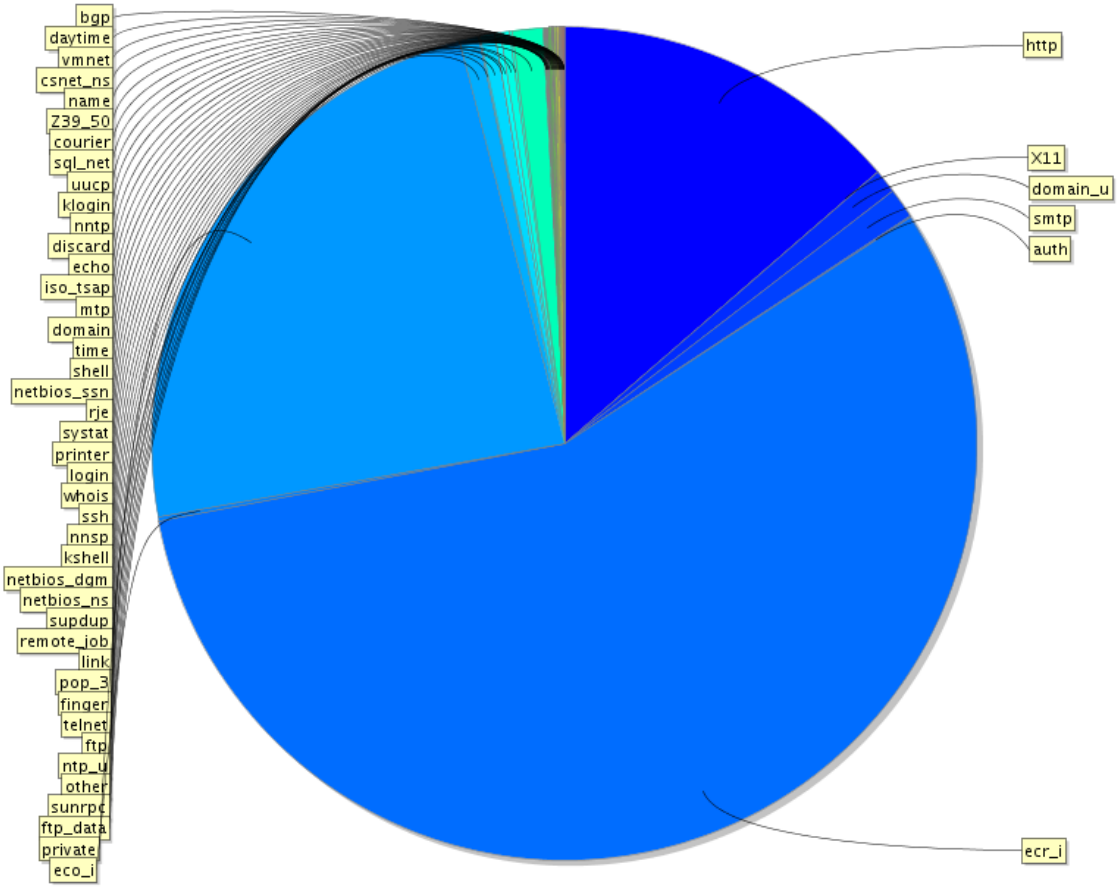
Rys. 5. Podział pakietów na zwykłe i rodzaje ataków (normal: 17428, dos: 10845, probe: 610, r2l: 185, u2r: 19).

3.4.2. Prezentacja danych - zbiór testowy

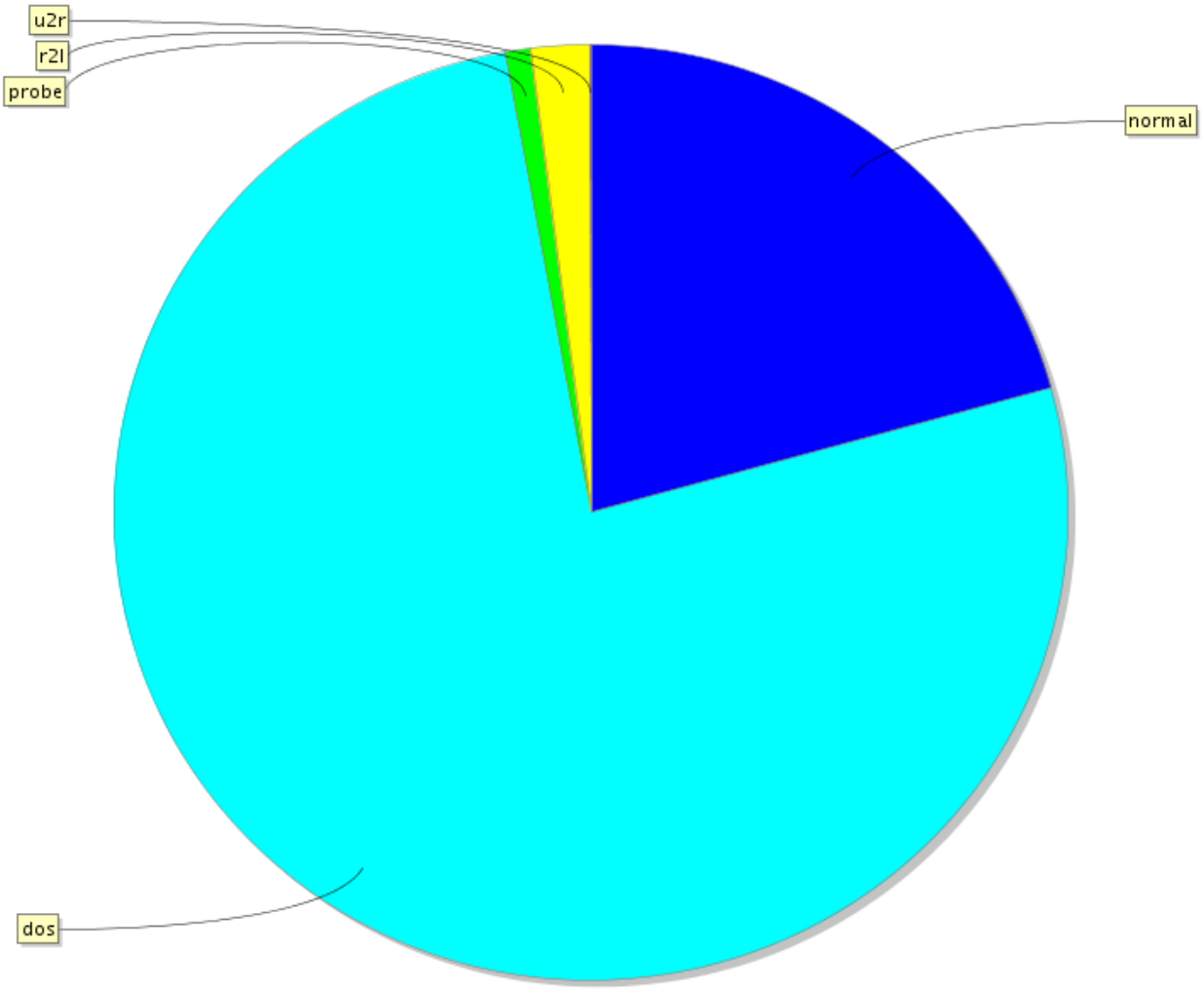
W zbiorze testowym o liczności 7196 rekordów wyróżniono trzy protokoły sieciowe:



Rys. 2. Protokoły sieciowe w zbiorze testowym.



Rys. 2. Usługi sieciowe w zbiorze testowym.



Rys. 5. Podział pakietów na zwykłe i rodzaje ataków (normal: 1491, dos: 5500, probe: 59, r2l: 146, u2r: 1).

4. Operatory klasyfikacyjne w środowisku RapidMiner

Oprogramowanie RapidMiner nie dostarczało odpowiednich operatorów, które pozwoliłyby na wykonanie projektu. Stąd niezbędne było wykonanie dodatkowych operatorów poprzez ingerencję w kod źródłowy. Aby to zrobić, należy do katalogu roboczego skopiować dwa projekty Eclipse:

- *RapidMiner_Extension_Template_Unuk*
- *Rapidminer_Unuk*

Powyższe projekty są wymagane do skompilowania rozszerzenia dla oprogramowania RapidMiner. Nowe operatory definiowane są w projekcie *RapidMiner_Extension_Template_Unuk* i można je wykonać w następujący sposób:

1) W katalogu *src/com/rapidminer/* definiujemy nową klasę dla operatora.
2) Nowa klasa powinna rozszerzać klasę `Operator` i przeciążać następujące metody:
    a. *doWork* - funkcje realizowane w trakcie przetwarzania danych
    b. *getParameterTypes* - parametry operatora

    Wejścia i wyjścia operatora definiowane są jako pola klasy:

    ```
    private final InputPort exampleSetInput = getInputPorts().createPort("nazwa wejscia");
    private final OutputPort exampleSetOutput = getOutputPorts().createPort("nazwa wyjscia");
    ```

    Konstruktor ma postać:

    ```
    public NowyOperator(OperatorDescription description) { super(description); }
    ```

    W ciele konstruktora typowe jest nadawanie warunków początkowych dla wejść / wyjść, ale tylko w przypadku, jeżeli oczekujemy ostrzeżenia gdy wejścia lub wyjścia nie są podłączone. Przykładowe odbieranie danych z portów / wysyłanie na port:

```
ExampleSet inputExampleSet = exampleSetInput.getData(ExampleSet.class);
exampleSetOutput.deliver(result);
```

Możemy posiłkować się istniejącym operatorem.

3) Po zdefiniowaniu klasy należy przejść do katalogu *resources/com/rapidminer/resources/* i w pliku *BalancingOperators.xml* dodać informacje o swoim operatorze:

```
<group key="">
  <group key="data_transformation">
    <group key="data_balancing">
      <operator>
        <key>unikalna_nazwa</key>
        <class>com.rapidminer.NowyOperator</class>
      </operator>
    </group>
  </group>
</group>
```

Znaczniki *<group key="nazwa">* informują w jakiej kategorii zostanie umieszczony nowy operator (panel *Operators* w rapidminerze).

4) Nazwa nadana w tagach *<key></key>* może zostać użyta do tłumaczenia nazwy operatora w pliku *OperatorsDocTemplate.xml* w katalogu *resources/com/rapidminer/resources/i18n/*.

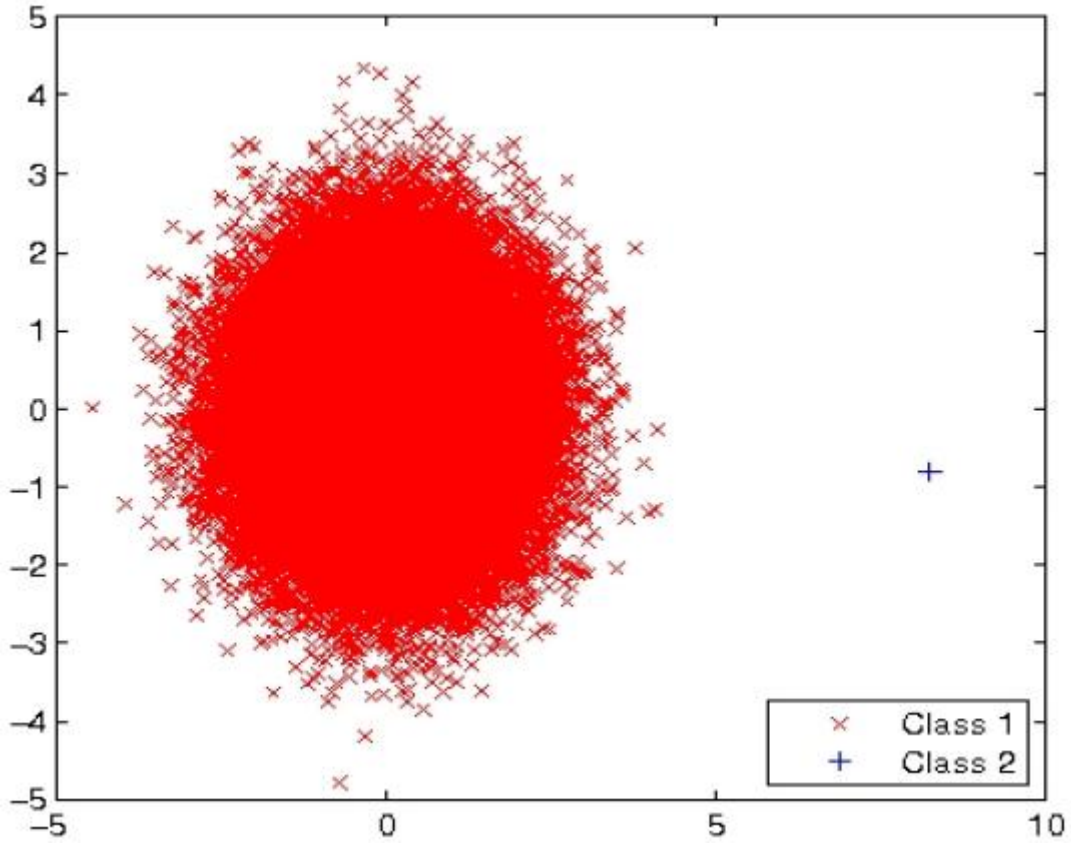
```
<operator>
  <key>unikalny_nazwa</key>
    <name>Nazwa widoczna w rapidmierze</name>
    <synopsis>Krótki opis</synopsis>
    <help>Długi opis</help>
</operator>
```

5) Budujemy projekt narzędziem Ant poprzez plik *build.xml* (PPM -> Run as -> Ant build) w głównym katalogu. Zbudowane rozszerzenie znajdzie się w projekcie *Rapidminer_Unuk* w folderze *lib/plugins*. Od teraz wystarczy uruchomić *lib/rapidminer.jar* i nowe operatory powinny działać. Rozszerzenie można też skopiować do analogicznego folderu w RapidMinerze 6.

5. Oversampling i undersampling

5.1. Problem danych niezrównoważonych

Problem danych niezbalansowanych pojawia się wtedy, gdy liczność jednej klasy (klasy dominującej, przyjmuje się że jest to klasa negatywna) jest istotnie wyższa niż liczność drugiej klasy (klasy zdominowanej, pozytywnej). Istotą problemu niezbalansowania jest fakt, że zastosowanie klasycznych mechanizmów uczenia na niezrównoważonym zbiorze danych może prowadzić do faworyzowania przez wyuczony klasyfikator klasy dominującej kosztem klasy zdominowanej.



Rys. 6. Prezentacja problemu danych niezbalansowanych.

Do rozwiązania problemu niezbalansowania stosuje się metody przetwarzania danych, takie jak oversampling i undersampling.

5.2. Undersampling

Undersampling to technika analizy danych wykorzystywana do regulacji podziału klas zbioru danych. Odrzuca ona część próbek z klasy większościowej. Istnieją cztery główne metody undersamplingu:

- **Random Undersampling**
  Usuwanie losowych próbek.
- **Tomek Links**
  Polega na usuwaniu przykładów granicznych i szumu z klasy większościowej. Dla każdego przykładu z klasy większościowej i klasy mniejszościowej sprawdzany jest następujący warunek: jeżeli nie istnieje przykład El z pary (Ei, Ej) nazywanej „Tomek Link", dla którego spełniona jest zależność:

  `d(Ei, El) < d(Ei, Ej) lub d(Ej, El) < d(Ei, Ej)`

  to przykłady będące w Tomek Link są usuwane ze zbioru.
- **Edited Nearest Neighborhood**
  Dla każdego przykładu z klasy większościowej znajdowanych jest trzech najbliższych sąsiadów. Jeżeli klasy dwóch sąsiadów są różne od klasy przykładu to przykład jest usuwany.
- **Nearest Cleaning Rule**
  Dla każdego przykładu ze zbioru znajdowanych jest trzech najbliższych sąsiadów. Jeżeli przykład należy do klasy większościowej i sąsiedzi klasyfikują go inaczej to jest usuwany. Jeżeli przykład należy do klasy mniejszościowej i sąsiedzi klasyfikują go inaczej to usuwani są sąsiedzi z klasy większościowej

Problemy, jakie może rodzić metoda undersamplingu, to usunięcie zbyt wielu przykładów klasy większościowej oraz doprowadzenie do gwałtownego pogorszenia rozpoznawania klas większościowych.

5.3. Oversampling

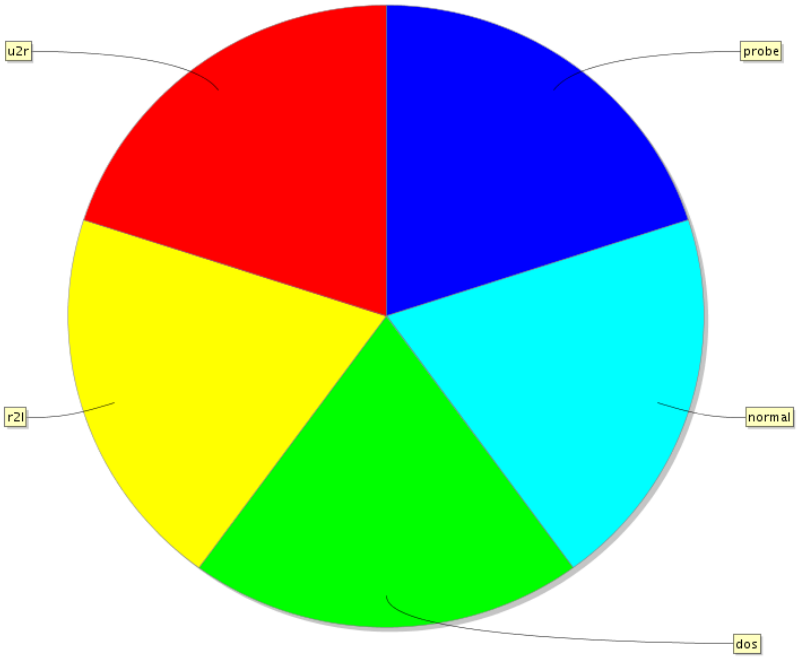
Oversampling to technika analizy danych wykorzystywana do regulacji podziału klas zbioru danych. Wybiera więcej próbek z klasy mniejszościowej (duplikuje dane). Replikacja zwiększa wagę próbek populacji mniejszościowej ale nie

zwiększa ilości informacji. Główną metodą oversampling jest metoda SMOTE, której zostanie poświęcony osobny rozdział.

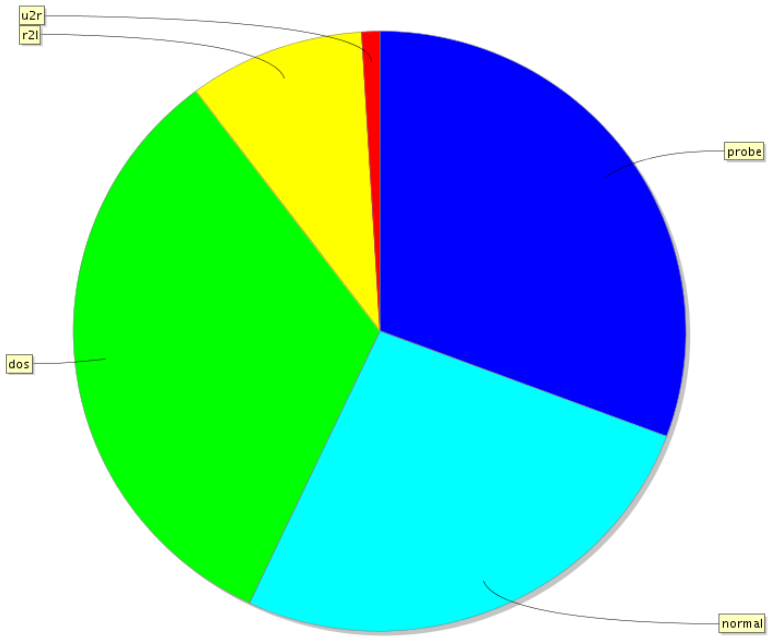
5.4. Przetwarzanie danych

Dane treningowe zostały przetworzone przez operatory oversamplingu i undersamplingu. Poniżej przedstawione są mnożniki oraz liczebności przykładów z poszczególnych klas. Poszczególne ataki zostały scalone do czterech typów ataków: dos, r2l, u2r oraz probe:

Typ pakietu	Liczność	Oversampling mnożnik	Oversampling liczność	Undersampling mnożnik	Undersampling liczność
normal	17428	-	17428	0,03	600
dos	10845	1,61	17428	0,06	600
probe	610	28,57	17428	-	610
r2l	185	94,21	17428	-	185
u2r	19	917,26	17428	-	19



Rys. 7. Wykres liczności typów pakietów po operacji oversamplingu.



Rys. 8. Wykres liczności typów pakietów po operacji undersamplingu.

5.5. Klasyfikacja (pakiety normalne i ataki)

Dane wejściowe reprezentowane przez rysunek nr 4 (podział na ataki i pakiety zwykłe) zostały poddane klasyfikacji za pomocą drzew decyzyjnych i lasów losowych (dobierając odpowiednio oversampling oraz undersampling) ze względu na występowanie i niewystępowanie ataku. Wyniki przedstawiają się następująco:

5.5.1. Drzewa decyzyjne (dane niezrównoważone)

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	1481	125	92.22%
	Attack	10	5581	99.82%
Class recall		99.33%	97.81%	

Acc	0,981242184
Acc+	0,993293092
Acc-	0,978093235
G-mean	0,985663864
Weighted Accuracy	0,985693164
Precision	0,922166874
Recall	0,993293092
F-measure	0,956409428

5.5.2. Drzewa decyzyjne (oversampling)

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	1480	125	92.21%
	Attack	11	5581	99.80%
Class recall		99.26%	97.81%	

Acc	0,981103237
Acc+	0,992622401
Acc-	0,978093235
G-mean	0,985331039
Weighted Accuracy	0,985357818
Precision	0,92211838
Recall	0,992622401
F-measure	0,956072351

5.5.3. Drzewa decyzyjne (undersampling)

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	1414	109	92.84%
	Attack	77	5597	98.64%
Class recall		94.84%	98.09%	

Acc	0,974155898
Acc+	0,948356808
Acc-	0,980897301
G-mean	0,96448983
Weighted Accuracy	0,964627054
Precision	0,928430729

Recall	0,948356808
F-measure	0,938287989

5.5.4. Lasy losowe (oversampling)

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	1486	4586	24.47%
	Attack	5	1120	99.56%
Class recall		99.66%	19.63%	

Acc	0,362095317
Acc+	0,996646546
Acc-	0,196284613
G-mean	0,442296712
Weighted Accuracy	0,596465579
Precision	0,244729908
Recall	0,996646546
F-measure	0,392965754

5.5.5. Lasy losowe (undersampling)

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	197	0	100.00%
	Attack	1294	5706	81.51%
Class recall		13.21%	100.00%	

Acc	0,820202862
Acc+	0,13212609
Acc-	1
G-mean	0,363491527
Weighted Accuracy	0,566063045
Precision	1
Recall	0,13212609
F-measure	0,233412322

5.6. Klasyfikacja (rodzaje ataków)

Dane wejściowe reprezentowane przez rysunek nr 5 (podział na rodzaje ataków i pakiety zwykłe) zostały poddane klasyfikacji za pomocą drzew decyzyjnych i lasów losowych (dobierając odpowiednio oversampling oraz undersampling) ze względu na rodzaj ataku. Wyniki przedstawiają się następująco:

5.6.1. Drzewa decyzyjne (dane niezrównoważone)

		TRUE					Class precision
		Normal	Dos	Probe	R2L	U2R	
PREDICTABLE	Normal	1480	6	0	105	1	92.96%
	Attack	1	5493	0	1	0	99.96%
	Dos	10	1	59	12	0	71.95%
	Probe	0	0	0	28	0	100.00%
	R2L	0	0	0	0	0	0.00%

	U2R	1480	6	0	105	1	92.96%
Class recall		99.26%	99.87%	100.00%	19.18%	0.00%	

5.6.2. Drzewa decyzyjne (oversampling)

		TRUE					**Class**
		Normal	**Dos**	**Probe**	**R2L**	**U2R**	**precision**
PREDICTABLE	**Normal**	1471	22	0	128	1	90.69%
	Attack	0	5448	0	0	0	100.00%
	Dos	20	30	59	18	0	46.46%
	Probe	0	0	0	0	0	0.00%
	R2L	0	0	0	0	0	0.00%
	U2R	1471	22	0	128	1	90.69%
Class recall		98.66%	99.05%	100.00%	0.00%	0.00%	

5.6.3. Drzewa decyzyjne (undersampling)

		TRUE					**Class**
		Normal	**Dos**	**Probe**	**R2L**	**U2R**	**precision**
PREDICTABLE	**Normal**	1432	3	0	95	1	93.53%
	Attack	15	5445	2	0	0	99.69%
	Dos	31	50	57	27	0	34.55%
	Probe	13	2	0	24	0	61.54%
	R2L	0	0	0	0	0	0.00%
	U2R	1432	3	0	95	1	93.53%
Class recall		96.04%	99.00%	96.61%	16.44%	0.00%	

5.6.4. Lasy losowe (dane niezrównoważone)

		TRUE					**Class**
		Normal	**Dos**	**Probe**	**R2L**	**U2R**	**precision**
PREDICTABLE	**Normal**	1491	4090	34	146	1	25.88%
	Attack	0	1410	25	0	0	98.26%
	Dos	0	0	0	0	0	0.00%
	Probe	0	0	0	0	0	0.00%
	R2L	0	0	0	0	0	0.00%
	U2R	1491	4090	34	146	1	25.88%
Class recall		100.00%	25.64%	0.00%	0.00%	0.00%	

5.6.5. Lasy losowe (oversampling)

		TRUE					**Class**
		Normal	**Dos**	**Probe**	**R2L**	**U2R**	**precision**
PREDICTABLE	**Normal**	1455	29	1	98	0	91.91%
	Attack	10	5465	19	7	0	99.35%
	Dos	8	6	39	0	0	73.58%
	Probe	16	0	0	36	1	67.92%
	R2L	2	0	0	5	0	0.00%
	U2R	1455	29	1	98	0	91.91%
Class recall		97.59%	99.36%	66.10%	24.66%	0.00%	

5.6.5. Lasy losowe (undersampling)

		TRUE					Class precision
		Normal	Dos	Probe	R2L	U2R	
PREDICTABLE	Normal	1398	27	0	111	1	90.96%
	Attack	0	1397	2	0	0	99.86%
	Dos	93	4076	57	35	0	1.34%
	Probe	0	0	0	0	0	0.00%
	R2L	0	0	0	0	0	0.00%
	U2R	1398	27	0	111	1	90.96%
Class recall		93.76%	25.40%	96.61%	0.00%	0.00%	

6. Metoda SMOTE

6.1. Opis metody

Algorytm SMOTE jest metodą generowania syntetycznych próbek z klasy zdominowanej. Syntetyczne próbki umieszczane są na odcinku łączącym najbliżej położone siebie obiekty z jednej i drugiej klasy. Sposób działania algorytmu prezentuje pseudokod:

```
Wejście
  T - liczba przykładów z klasy zdominowanej
  N - procentowa liczebność zbioru SMOTEd względem zbioru oryginalnego
  k - liczba najbliższych sasiadów
Wyjście:
  (N/100)*T - elementowy zbiór przykładów SMOTEd z klasy zdominowanej

SMOTE(T, N, k)
begin
  /* Jeśli N jest mniejsze od 100, losowy zbiór przykładów zostanie zrównoważony */
  if N < 100 then
    Losuj przykłady z klasy T zdominowanej
    T = (N/100) * T
    N = 100
  endif
  N = (int)(N/100) /* N jest liczbą naturalną będącą wielokrotnością 100 */
  k = Liczba najbliższych sąsiadów
  numattrs = Liczba atrybutów
  Sample[ ][ ]: Macierz oryginalnych obiektów klasy zdominowanej
  newindex: zapamiętuje liczbę wygenerowanych syntetycznych przykładów, inizjalizowany jako 0
  Synthetic[ ][ ]: Macierz przykładów syntetycznych
  /* Odnajduje k njbliższych sąsiadów dla każdego obiektu klasy zdominowanej */
  for i <- 1 to T
    Odnajduje k najbliższych sasiadów dla i, zapisuje do tablicy nnarray
    Populate(N, i, nnarray)
  endfor
end

Populate(N, i, nnarray) /* Funkcja do generowania przykładów syntetycznych. */
begin
  while N != 0
    Wybierz losową liczbę ze zbioru od 1 do k i przypisz są do zmiennej nn. Ten krok wybiera
jednego z najbliższych sąsiadów i.
    for attr <- 1 to numattrs
      Oblicz: dif = Sample[nnarray[nn]][attr] - Sample[i][attr]
      Oblicz: gap = losowa liczba z przediału od 0 do 1
      Synthetic[newindex][attr] = Sample[i][attr] + gap * dif
    endfor
    newindex++
    N = N - 1
  endwhile
  return
end
```

Rozważmy przykład (6,4) i załóżmy, że (4,3) jest jego najbliższym sąsiadem. (6,4) jest przykładem, dla którego określono k najbliższych sąsiadów. (4,3) jest jednym z k najbliższych sąsiadów
f1_1 = 6, f2_1 = 4, f2_1 – f1_1 = -2
f1_2 = 4, f2_2 = 3, f2_2 – f1_2 = -1
Nowe przykłady zostaną wygenerowane według:
(f1',f2') = (6,4) + rand(0-1)*(-2,-1)
gdzie rand(0-1) oznacza losową liczbę z przedziału 0-1

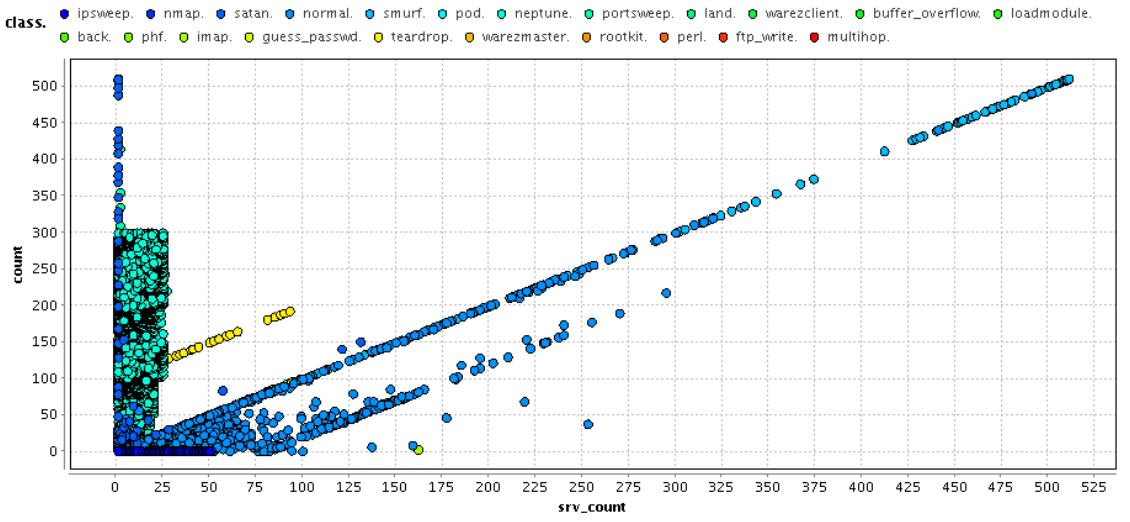
Często metodę SMOTE poprzedza się dodatkowo operacją under-samplingu. Polega to na tym, że z klasy dominującej usuwa się przykłady dopóki liczebność klasy zdominowanej nie stanie się pewnym określonym ułamkiem klasy dominującej.

6.2. Przetwarzanie danych

Poniżej przedstawiono wykresy zależności 2 parametrów, z wykorzystaniem zbioru uczącego zawierającego 29087 rekordów

- count - liczba połączeń do tego samego hosta w przeciągu ostatnich 2 sekund
- srv_count - liczba połączeń do tej samej usługi w przeciągu ostatnich 2 sekund

Wybór takich a nie innych parametrów spowodowany jest tym że obydwa parametry są typu liczbowego oraz że ich wartości w zbiorze danych są zróżnicowane, co umożliwia pokazania działania algorytmu SMOTE na wykresie. Dodatkowo kolor określa klasę danego pakietu.



Rys. 9. Wykres przedstawia oryginalny, niezmodyfikowany zbiór danych.

Przynależność klas do konkretnych rodzajów ataków pokazuje poniższa lista

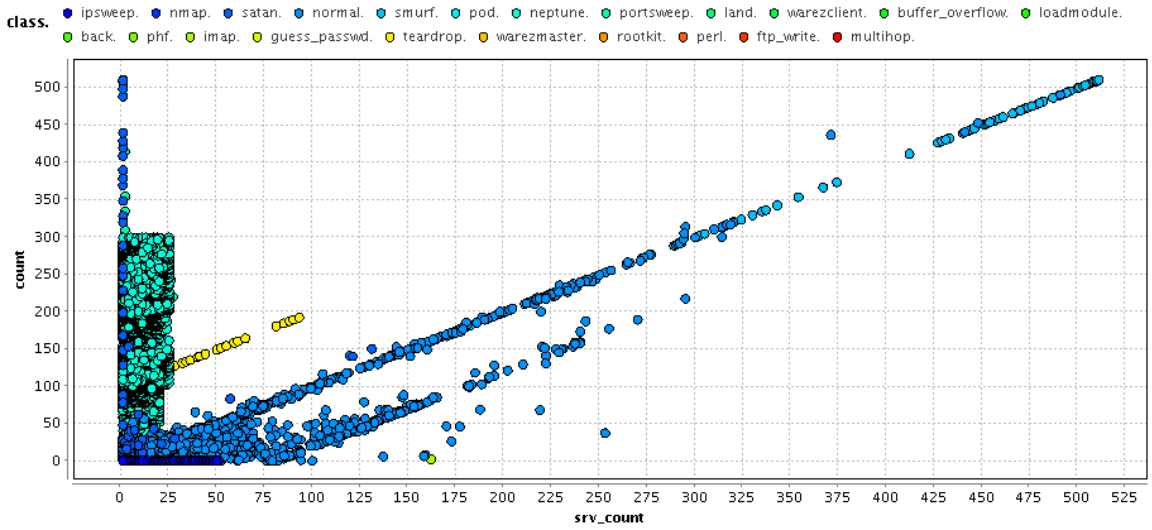
- probe: ipsweep, nmap, satan, portsweep, teardrop
- dos: smurf, pod, neptune, land, back
- r2l: warezclient, phf, imap, guess_passwd, warezmaster, ftp_write, multihop, spy
- u2r: buffer_overflow, loadmodule, perl, rootkit

Z wykresu można stwierdzić że istnieje liczna grupa pakietów która wykazuje relacje liniową między liczbą połączeń do tego samego hosta a liczbą połączeń do tej samej usługi w przeciągu ostatnich 2 sekund. Jest to grupa która składa się w większości z pakietów które nie były atakami.

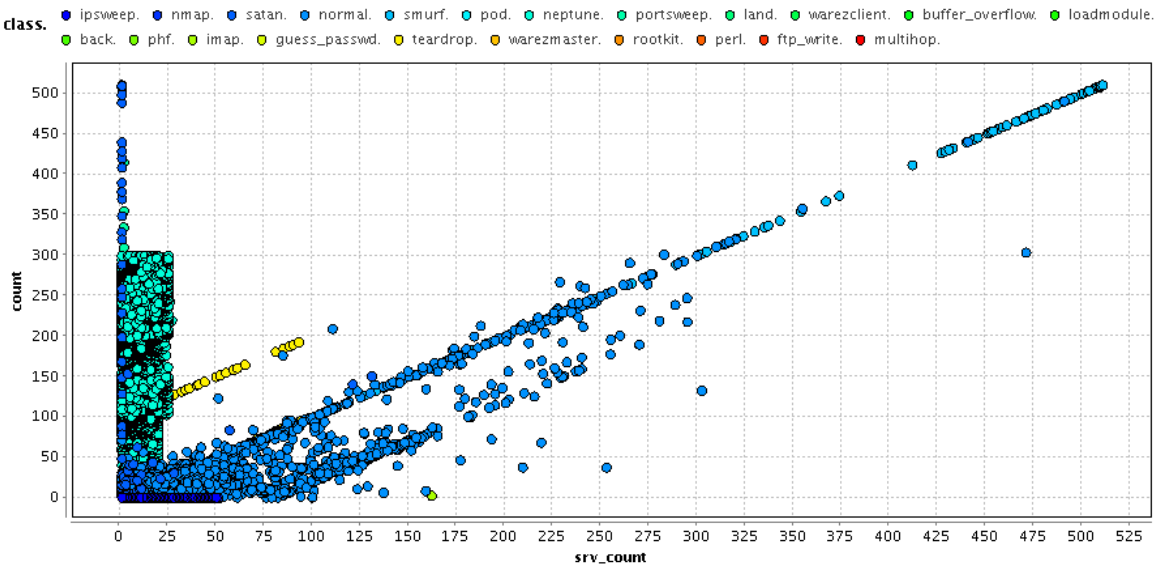
Inną liczną grupę stanowią pakiety, gdzie liczba połączeń do tego samego hosta zawierała się w przedziale od 0 do 300 a liczby połączeń do usługi w przedziale od 0 do 25. Były to głównie pakiety związane z atakami typu dos.

Na poniższych wykresach przedstawiono zbiór danych wzbogacony o syntetyczne dane stworzone przy użyciu metody SMOTE dla różnych parametrów wejściowych:

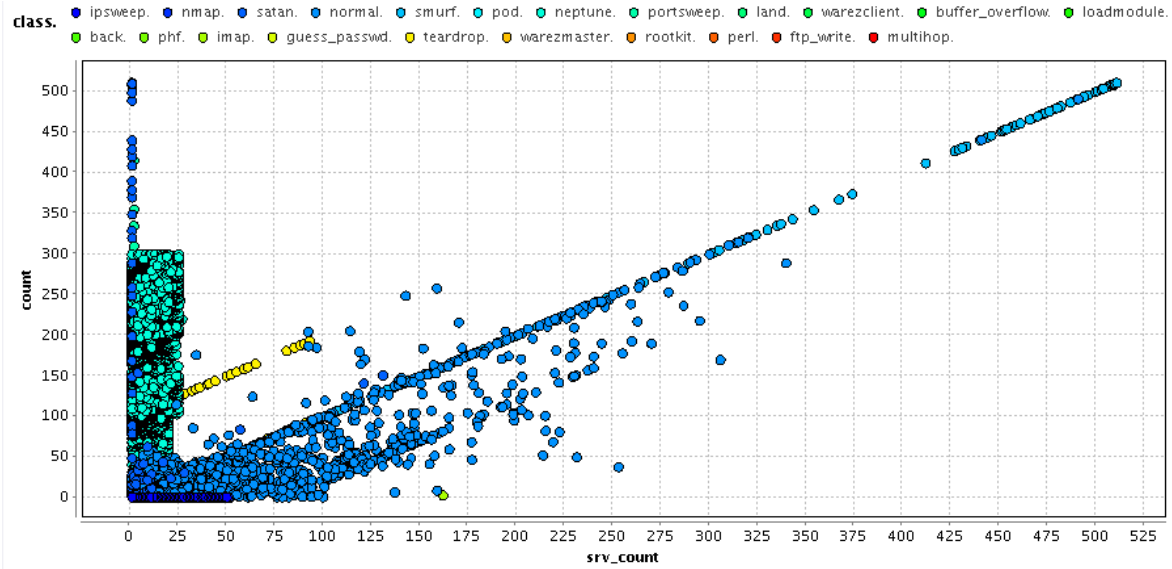
N - wartość metody SMOTE, będąca krotnością liczby 100
k - liczba najbliższych sąsiadów wziętych pod uwagę podczas tworzenia syntetycznego obiektu
Jako klasę mniejszościową przyjęto klasę normal - czyli pakiety nie będące atakami.



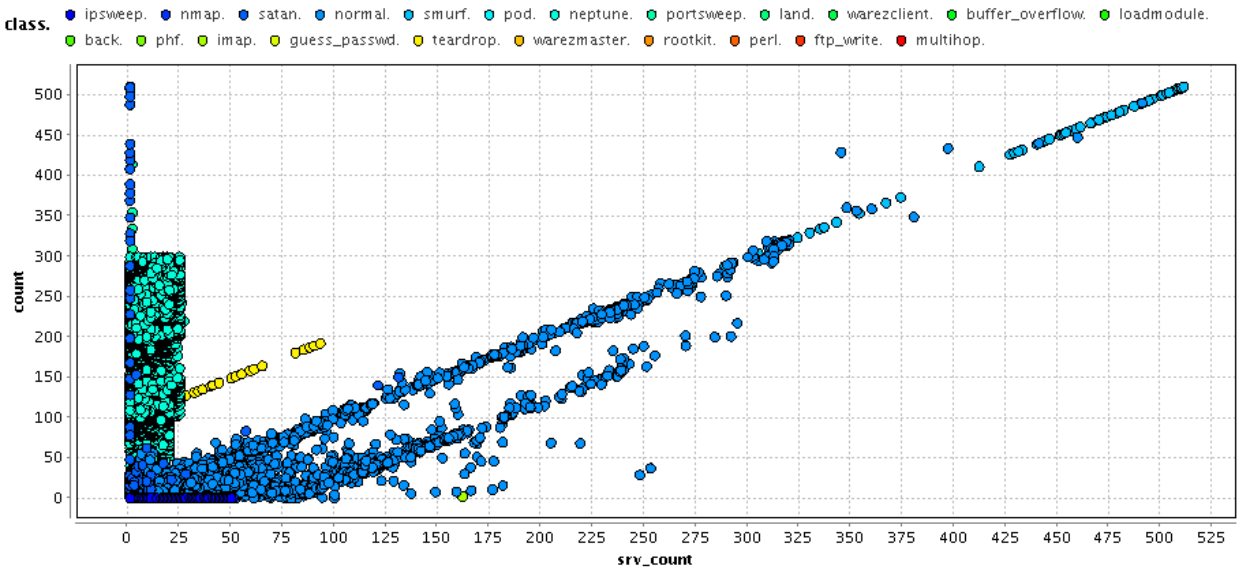
Rys. 10. N = 100, k = 3



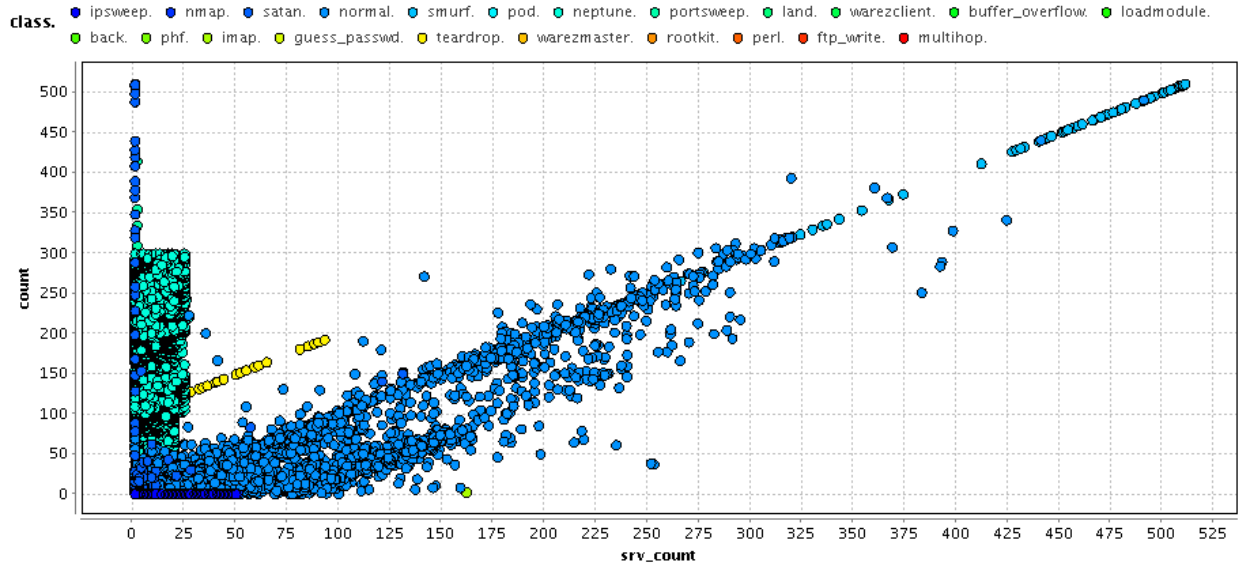
Rys. 11. N = 100, k = 30



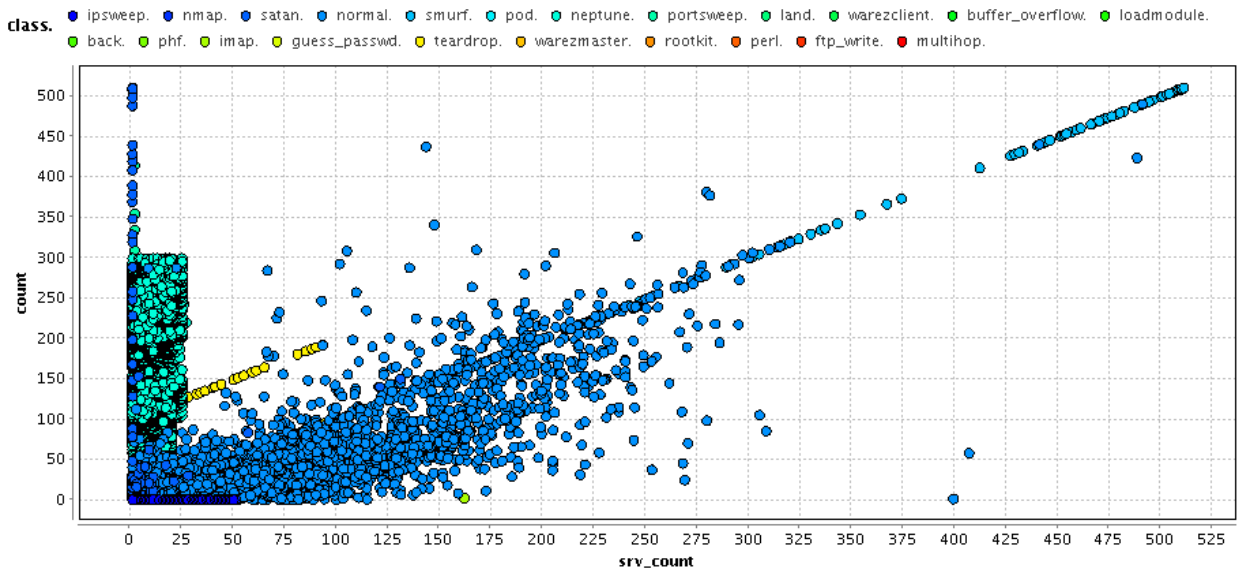
Rys. 12. N = 100, k = 300



Rys. 13. N = 500, k = 3



Rys. 14. N = 500, k = 30



Rys. 15. N = 500, k = 300

Na powyższych wykresach widać że algorytm działa prawidłowo, gdyż dla wyższych wartości N generuje więcej elementów, natomiast wraz ze wzrostem parametru k, elementy syntetyczne są bardziej oddalone od elementów ze zbioru danych, czyli znajdują się w dalszym sąsiedztwie.

6.2. Symulacja

[PATRZ OVER- I UNDERSAMPLING]

6.2.3. Komentarz

Dla drzew decyzyjnych oraz klasy pod zauważono poprawę wyników oraz zwiększenie dokładności klasyfikacji drzewa. Dla klas satan oraz ipsweep klasyfikacja była mniej dokładna niż przed zastosowaniem metody SMOTE. W przypadku lasów losowych SMOTE okazał jedynym gwarantem uzyskania jakiegokolwiek podziału. Dla ipsweep dokładność zwiększyła się z 0 do 0,15, natomiast dla satan z 0 do 100. Nawet przy próbie wygenerowania aż 9-krotnie większej liczby przykładów syntetycznych z klasy pod nie zaobserwowano poprawy wyników.

7. Metoda BRF

7.1. Opis metody

Jedną z metod rozwiązujących problem niezrównoważenia danych jest zastosowanie Zrównoważonych Lasów Losowych. Metoda ta (Balanced Random Forest) została zaproponowana przez Leo Breiman-a w artykule „Using Random Forest to Learn Imbalanced Data”. Działanie algorytmu jest następujące:

- Dla każdego pojedynczego drzewa losuje się podpróbę bootstrapową - połowa przypadków jest losowana w sposób niezależny z klasy mniej licznej, a druga połowa – również w sposób niezależny – z klasy bardziej licznej
- Buduje się model drzewa przy pomocy algorytmu CART bez opcji przycinania; na każdym etapie podziału wykorzystuje się losowo dobrany zestaw predykatorów poddawanych ocenie.
- Pierwszy i drugi etap powtarza się zadaną liczbę razy
- Agreguje się pojedyncze drzewa w celu uzyskania finalnego modelu predykcyjnego.

7.2. Działanie algorytmu

[PATRZ OVER- I UNDERSAMPLING]

8. Metoda WRF

8.1. Działanie algorytmu.

Problem klasyfikacji danych silnie niezrównoważonych został opisany wyżej. Rozwiązanie tego problemu zostało zaproponowane przez Leo Breiman-a w jego publikacji „Using Random Forest to Learn Imbalanced Data”. W publikacji przedstawione zostały dwa algorytmy, a jednym z nich jest metoda Weighted Random Forest.

Breiman i współautorzy zakładają, że podczas wyboru poszczególnych drzew należy na podstawie wstępnej analizy danych tak wybrać kryterium podziału, żeby klasy mniejszościowe otrzymywały większy mnożnik kary za niepoprawną klasyfikacje.

Dla każdego potencjalnego podziału, waga dla każdego potencjalnego węzła dziecka jest definiowana jako:

$$t_c = \sum_i w_i * n_i$$

n_i to ilość obserwacji klasy i w c, a w_i to waga przypisana do klasy i.

Współczynnik nieczystości węzła dziecka c:

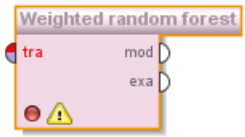
$$i_c = 1 - \sum_i (w_i * n_i / t_c)^2$$

Łączna nieczystość potencjalnego podziału będzie się równać:

$$\sum_c (t_c / t_p) * i_c$$

, gdzie t_p to suma wag wszystkich obserwacji w węźle rodzica, dla którego obliczany jest podział.

Według zalecenie Breimana trzeba tak zmodyfikować algorytm Random Forest, by korzystał on z poprawnie zaimplementowanego kryterium Gini Index. Na potrzeby symulacji stworzony został operator Weighted Random Forest, który pozwala na stworzenie modeli, uwzględniających duży współczynnik kar dla klas mniejszościowych.

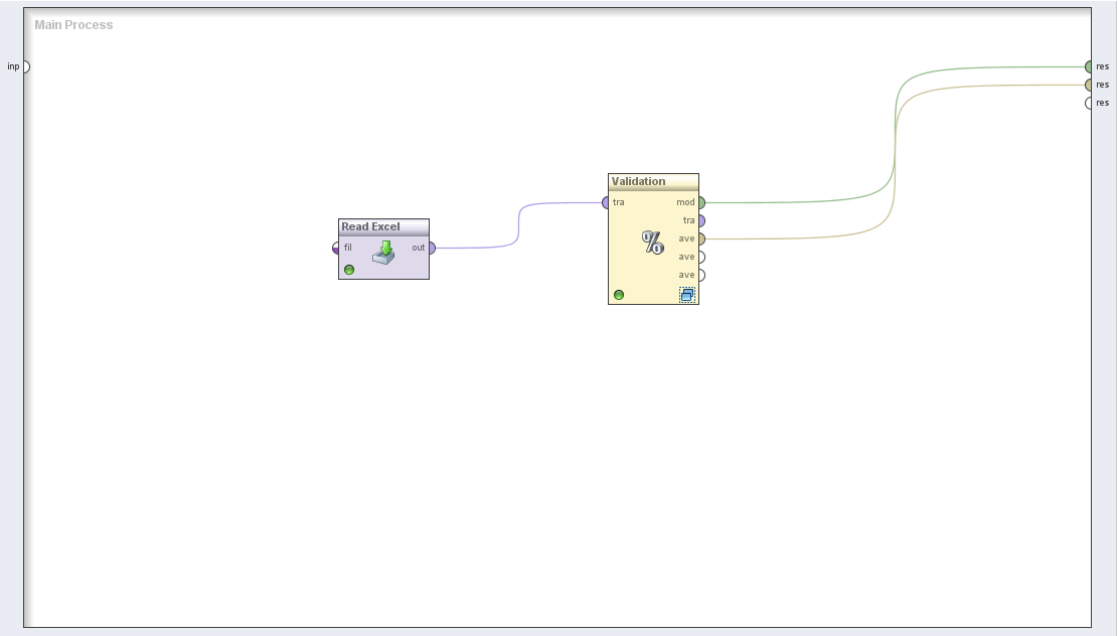


Wejście i wyjścia, oraz ustawialne parametry są identyczne jak w oryginalnym operatorze, a jedynie zostało zmodyfikowane kryterium podziału.

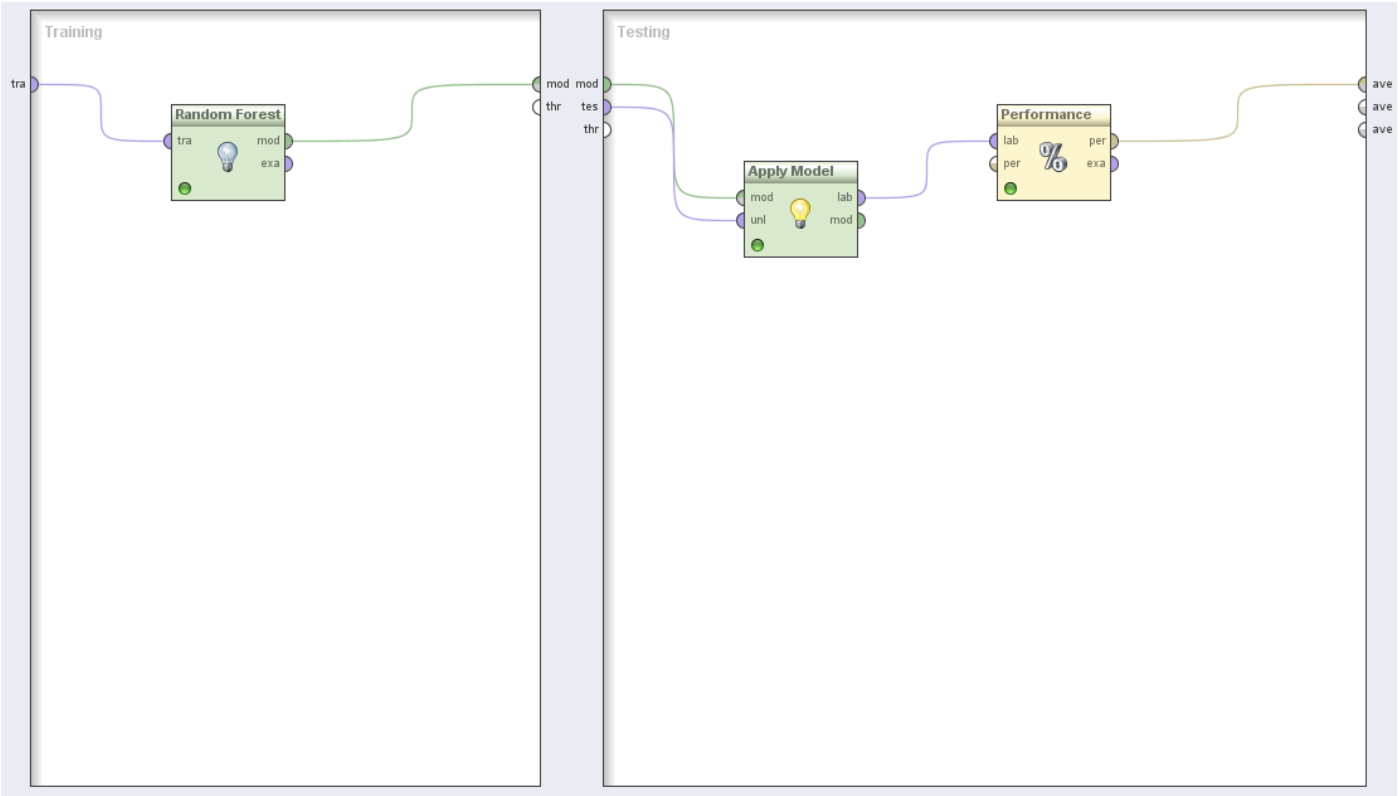
8.2. Symulacja.

W celu sprawdzenie wydajności działania algorytmu WRF zostały przeprowadzone analogiczne doświadczenie jak w punktach 5.5. x.

Główny proces:



Validation:



Metoda WRF NORMAL VS ATTACK

		TRUE		Class precision
		Normal	Attack	
PREDICTABLE	Normal	5200	177	96.71%
	Attack	12	3337	99.64%
Class recall		99.77%	94.96%	

Metoda WRF NORMAL VS 4 klasy ATTACK

		TRUE					Class precision
		Normal	**PROBE**	**DOS**	**R2L**	**U2R**	
PREDICTABLE	**Normal**	5212	75	180	56	7	94.25%
	PROBE	0	33	0	0	0	100%
	DOS	0	14	3149	0	0	99.56%
	R2L	0	0	0	0	0	0%
	U2R	0	0	0	0	0	0%
Class recall		100%	27.05%	94.59%	0%	0%	

9. Podsumowanie

Podczas pracy nad projektem grupa spotkała się z problemem selekcji danych testowych z ogromnej liczby danych uzyskanych z eksperymentu. Okazało się, że standardowe podejście z wykorzystaniem środowiska RapidMiner jest nieskuteczne i niezbędne było zastosowanie innych rozwiązań.

Metoda SMOTE na pierwszy rzut oka skutecznie radzi sobie z danymi niezbalansowanymi. Jednak eksperymenty przeprowadzone przez grupę pokazują, że tylko część algorytmów grupujących dokonuje dokładniejszego podziału po spreparowaniu dodatkowych przykładów syntetycznych. Dodając do tego zależność od wyboru klasy mniejszościowej nie można jednoznacznie stwierdzić, że SMOTE poprawia bądź pogarsza klasyfikację.

Metoda WRF, sprawdza się dla danych, w których występują klasy niezrównoważone, ale, dla których elementy klasy są do siebie podobne. W sytuacji, w której posiadamy klasy mniejszościowe, ale z bardzo różnymi elementami składowymi, a taki przypadek analizujemy dla ataków sieciowych, przyrost dokładności klasyfikacji nie jest zbyt wielki.

Dla doświadczeń przeprowadzanych w ramach projektu, dla standardowych drzew losowych, uzyskano dokładność na poziomie ~93%, a dla metody WRF 94,67% dla oryginalnego podziału na ilość ataków, oraz 95% (RF) i 96,7% (WRF) – podział na ruch normalny i 4 rodziny ataków.