AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Wieloagentowe środowisko dla potrzeb systemu zarządzającego przedsiębiorstwem i do optymalizacji działań przedsiębiorstwa

Artur Wolak
Tomasz Nizio

Spis treści

1. Sformułowanie zadania projektowego

1.1 Temat i cel zadania projektowego

Tematem jest stworzenie wieloagentowego środowiska dla potrzeb systemu zarządzającego przedsiębiorstwem i do optymalizacji działań przedsiębiorstwa. Naszym przedsiębiorstwem będzie odlewnia metali, która produkuje różnego rodzaju stopy. Celem jest stworzenie środowiska, które pomoże zoptymalizować działanie odlewni.

Głównym zadaniem systemu jest możliwość przewidywania jakości stopu wybranego metalu, w zależności od podanych parametrów wejściowych, czyli m.in. temperatury i czasu wytapiania. System pozwala również wyznaczyć koszt produkcji danego stopu oraz uwzględnia wypalanie się metali w czasie produkcji, dlatego udostępnia też informacje na temat wsadu (ile danego metalu trzeba zużyć, aby otrzymać oczekiwany skład).

Założyliśmy, że stopy uzyskiwane są z następujących metali: aluminium, krzem, magnez, miedź, cynk, cyna, nikiel, żelazo i ołów.

1.2 Cele

Postawiliśmy następujące cele:

- implementacja w języku Java,
- wykorzystanie środowiska wieloagentowego z użyciem frameworku JADE[2],
- wykorzystanie biblioteki WEKA[1] służącej do uczenia maszynowego,
- stworzenie generatora danych treningowych oraz testowych,
- umożliwienie przewidywania jakości stopu, wyznaczenia kosztu oraz wsadu,
- umożliwienie uczenia maszynowego na nowych danych

1.3 Opis istniejących rozwiązań

Będziemy bazować na projektach studentów z poprzednich lat. Wykorzystane w nich są następujące biblioteki: WEKA[1] oraz JADE[2]. Zauważyliśmy, że w żadnym z projektów nie były wykorzystane te biblioteki równocześnie, dlatego staraliśmy się połączyć je w naszym systemie. Elementy istniejących projektów[3], z których będziemy korzystać:

- [Agenci] - model agentowy - kod odpowiedzialny za cykl życia agentów,
- [IO] - GUI (istniejący interfejs będzie edytowany),
- [Proj02 - IndustryAnalizer] - wykorzystanie biblioteki WEKA

2. Opis wizji rozwiązania

2.1 Model wizji systemu

System składa się z następujących warstw:

- **Warstwa 1 - interakcja z użytkownikiem**
  Ta warstwa odpowiedzialna jest za udostępnienie interfejsu graficznego użytkownikowi. Do implementacji tej warstwy wykorzystaliśmy technologię JavaFX[4]. Informacje z tej warstwy zarządzane są przez system agentowy.
- **Warstwa 2 - system agentowy**
  System agentowy zarządza systemem. Kieruje odpowiednie komunikaty do poszczególnych agentów. Implementacja w oparciu o framework JADE[2]
- **Warstwa 3 - uczenie maszynowe i generowanie danych**
  W tej warstwie występują operacje, które zajmują najwięcej czasu. Odbywa się tu trenowanie klasyfikatorów i generowanie danych. Wykorzystana została biblioteka WEKA[1]. Klasyfikatory[5], które będziemy brali pod uwagę przy testowaniu ich skuteczności na wygenerowanych danych treningowych to:
    - RandomForest
    - MultilayerPerceptron
    - M5P
    - KStar
    - DecisionTable
    - DecisionStump
    - Bagging
    - Vote

2.2 Opis i sposób przygotowania danych wejściowych

Dane wejściowe generowane są przez stworzony przez nas generator. Na jego wejście podawany jest procentowy udział metali w stopie. Wyróżniliśmy następujące gotowe stopy metali (odnośniki zawierają informacje o podanych stopach):

- aluminium[6]: (80%)
    - z krzemem (20%),
    - z krzemem (15%) i magnezem (5%),
    - z krzemem (12%), miedzią (6%), magnezem (2%),
- magnezu[7]: (90%)
    - z aluminium (8%) i krzemem (2%)
    - z aluminium (7%) i cynkiem (3%)
- miedzi[8]:
    - brązy: miedź (98%), cyna (2%)
    - mosiądz: miedź (70%), cynk (30%)

- niklu[9] (67%):
    - z miedzią (30%) i żelazem (3%)
- ołowiu[10] (95%):
    - z cyną (3%), miedzią (2%)

Dodatkowo użytkownik ma możliwość zdefiniowania własnego stopu metalu. Głównym czynnikiem wpływającym na postać generowanych danych jest temperatura topnienia metali. W stopie wybierany jest metal, który ma najwyższą temperaturę topnienia. Ma ona największe znaczenie, ponieważ od niej zależy, czy ten metal ulegnie stopieniu. Od niej również zależy proces krzepnięcia stopu.

Poniżej znajdują się temperatury topnienia[11] metali, które wykorzystaliśmy:
- Aluminium - 660°C
- Krzem - 1410°C
- Magnez - 650°C
- Miedź - 1084°C
- Cynk - 420°C
- Cyna - 232°C
- Nikiel - 1453°C
- Żelazo - 1500°C
- Ołów - 327°C

Działanie generatora, jego normy, wzory i zależności są opisane w punkcie 4.4, po tym jak przedstawimy proces produkcji stopu. Jest on istotny, by zrozumieć, jak generowane są dane.

3. Harmonogram

3.1 Kamienie milowe

1. Wygenerowanie danych treningowych, na których będziemy mogli przetestować działanie różnych algorytmów uczenia maszynowego
2. Przetestowanie różnych algorytmów uczenia maszynowego, wybranie tych które dają najlepsze rezultaty oraz ich odpowiednia implementacja
3. Utworzenie środowiska wieloagentowego
    a. ustalenie właściwej struktury środowiska agentowego
    b. implementacja odpowiednich agentów oraz ich zachowań
4. Przebudowanie oraz rozszerzenie GUI
5. Przeprowadzenie różnorodnych eksperymentów w aplikacji
6. Ukończenie dokumentacji

3.2 Tydzień 1 (30.07 - 3.08)

Ustalenie wizji systemu, założenia dotyczące programu:

- zapoznanie z tematem projektu
- stworzenie harmonogramu
- wygląd danych wejściowych i wyjściowych
- przygotowanie środowiska
- wstępne ustalenie algorytmów uczenia maszynowego, które wykorzystamy
- zdefiniowanie agentów
- zdefiniowania sposobu obliczania kosztów produkcji
- ustalenie zależności między parametrami

3.3 Tydzień 2 (6.08 - 10.08)

Praca nad uczeniem maszynowym:

- przygotowanie formatu danych wejściowych
- **stworzenie generatora danych i przygotowanie danych treningowych (1)**
- testowanie różnych klasyfikatorów
- **wybór klasyfikatorów dla uczenia jakości (2)**

3.4 Tydzień 3 (13.08 - 17.08)

GUI i środowisko agentowe:

- poprawki modułu uczenia maszynowego
- testowanie klasyfikatorów
- **ustalenie struktury agentów (3a)**
- zaprojektowanie wyglądu interfejsu graficznego
- **implementacja środowiska wieloagentowego (3b)**

3.5 Tydzień 4 (20.08 - 24.08)

Baza danych i dokumentacja:

- **implementacja GUI (modyfikacja obecnego - dodanie nowych elementów) (4)**
- połączenie z bazą danych
- tworzenie wykresów przedstawiających działanie systemu
- **przeprowadzanie eksperymentów (5)**
- ostateczne poprawki
- **tworzenie ostatecznej wersji dokumentacji (6)**

4. Koncepcja

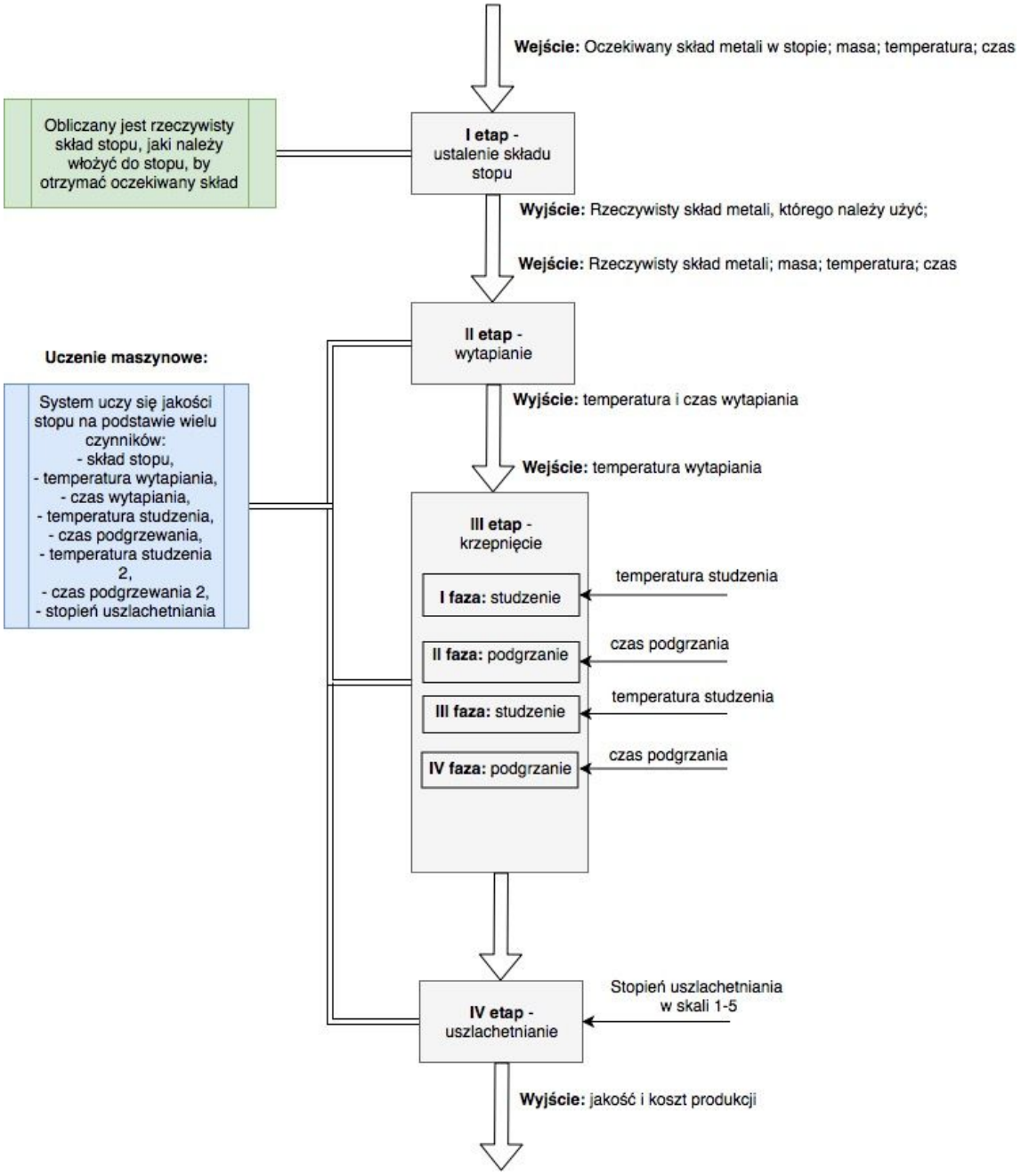
4.1 Opis problemu

Problemy jakie rozwiązuje nasz system to:

- obliczenie ilości metali jakie trzeba włożyć do pieca aby uzyskać docelowy stop,
- obliczenie przybliżonego kosztu wytworzenia danego stopu
- przewidywanie jakości stopu, przy parametrach podanych przez użytkownika

4.2 Proces produkcji stopu

Ustaliliśmy, że proces wygląda następująco: ustalenie składu stopu, wytapianie, krzepnięcie oraz uszlachetnianie. Poniżej znajduje się diagram ukazujący wyszczególnione przez nas etapy produkcji.



rys. 1 - etapy produkcji

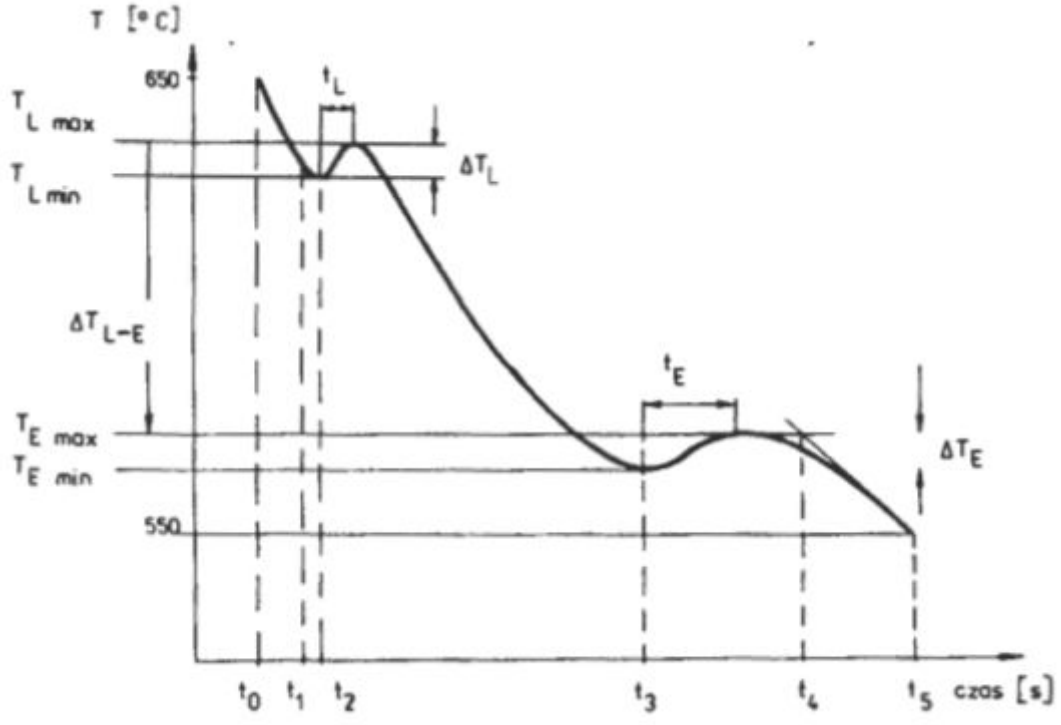
I etap - ustalanie składu stopu:

Na wejściu użytkownik wybiera stop metali z listy. Procentowy udział poszczególnych metali jest wczytywany z bazy, ale użytkownik może dowolnie zmieniać skład stopu. Użytkownik musi podać również masę stopu i temperaturę wytapiania. Założyliśmy, że optymalna temperatura wytapiania powinna być o 30% wyższa, od temperatury topnienia metalu o najwyższej temperaturze topnienia w

stopie. Na wyjście dostaje procentowy udział metali, jaki musi być w stopie aby uzyskać oczekiwany skład.

II etap - wytapianie:
Temperatura wytapiania powinna być wystarczająca aby stopić wszystkie składniki, ale również osiągalna w piecu. Zakładamy że piec może się rozgrzać do ok. 2000°C. Użytkownik podaje dodatkowo jeszcze czas wytapiania. Założyliśmy, że optymalny czas wytapiania to 50 minut.

III etap - krzepnięcie:
Krzepnięcie podzieliliśmy na 4 fazy. Poniższe informacje są ustalone na podstawie dokumentu[12]: *"Przydatność analizy termicznej do oceny procesu krystalizacji"* autorstwa Jadwigi Skawy. Poniżej przedstawiona jest wzorcowa krzywa krzepnięcia:



rys. 2 - krzywa krzepnięcia

Z powyższego wykresu i informacji zawartych w dokumencie opisującym proces krzepnięcia możemy wyszczególnić następujące fazy:

- I faza (stygnięcie - t_0 - t_1) - temperatura stygnięcia powinna wynosić około ⅔ temperatury wytapiania,
- II faza (podgrzanie - t_1 - t_2) - podgrzanie o 10 °C. Czas podgrzewania to około 15 minut,
- III faza (stygnięcie - t_2 - t_3) - kolejna temperatura stygnięcia powinna wynosić około ½ temperatury pierwszego stygnięcia,

- IV faza (podgrzanie - t_3 - t_4) - podgrzanie o 10 °C. Czas podgrzewania to około 25 minut

IV etap - uszlachetnianie:

Użytkownik podaje stopień uszlachetniania w skali 1-5.

1 - brak uszlachetniania
2 - słabe
3 - średnie
4 - dobre
5 - bardzo dobre

4.2.1 Ustalenie składu stopu

Aby ustalić skład stopu musimy wiedzieć jak wypalają się dane metale w zależności od temperatury. Do tego potrzebny jest współczynnik wypalania. Założyliśmy, że współczynnik wypalania rośnie wraz z temperaturą wytapiania i jest inny dla każdego metalu. Przyjęte przez nas podatności poszczególnych metali na wypalenie w stosunku do żelaza, w przybliżeniu określone na podstawie temperatur topnienia poszczególnych metali:

- Aluminium - 2
- Krzem - 1
- Magnez - 2
- Miedź - 1,5
- Cynk - 2,3
- Cyna - 2,8
- Nikiel - 1
- **Żelazo - 1**
- Ołów - 2,5

Dodatkowo aby uzależnić go od temperatury postanowiliśmy powiększyć go o temperaturę wytapiania/2000. Założyliśmy że określa nam ona jaki procent metalu ulega wypaleniu. Ilość metalu, którego potrzebujemy, wyliczana jest poprzez przemnożenie oczekiwanej masy przez współczynnik i dodanie jej do oczekiwanej masy.

4.2.2 Obliczanie przybliżonego kosztu produkcji

Ceny[13] poszczególnych metali za 1kg:

- Aluminium - 7,5 zł
- Krzem - 40 zł
- Magnez - 32 zł
- Miedź - 22 zł
- Cynk - 9,5 zł
- Cyna - 73 zł
- Nikiel - 48 zł

- Żelazo - 0,5 zł
- Ołów - 8 zł

Przyjęliśmy, że koszt uszlachetniania to poziom_uszlachetaniania*masa_wsadu. Koszty energii potrzebnej do wytapiania rosną wraz z czasem i temperaturą wytapiania, założyliśmy więc, że wytapianie przez 1min w 1000°C kosztuje 10 zł. Koszt ogrzewania, dla uproszczenia równy czas*temp/1000*10zł+masa*stopien_uszlachetniania. Ostateczny koszt to suma ceny metali, ceny uszlachetniania i ceny wytapiania.

4.2.3 Przewidywanie jakości stopu

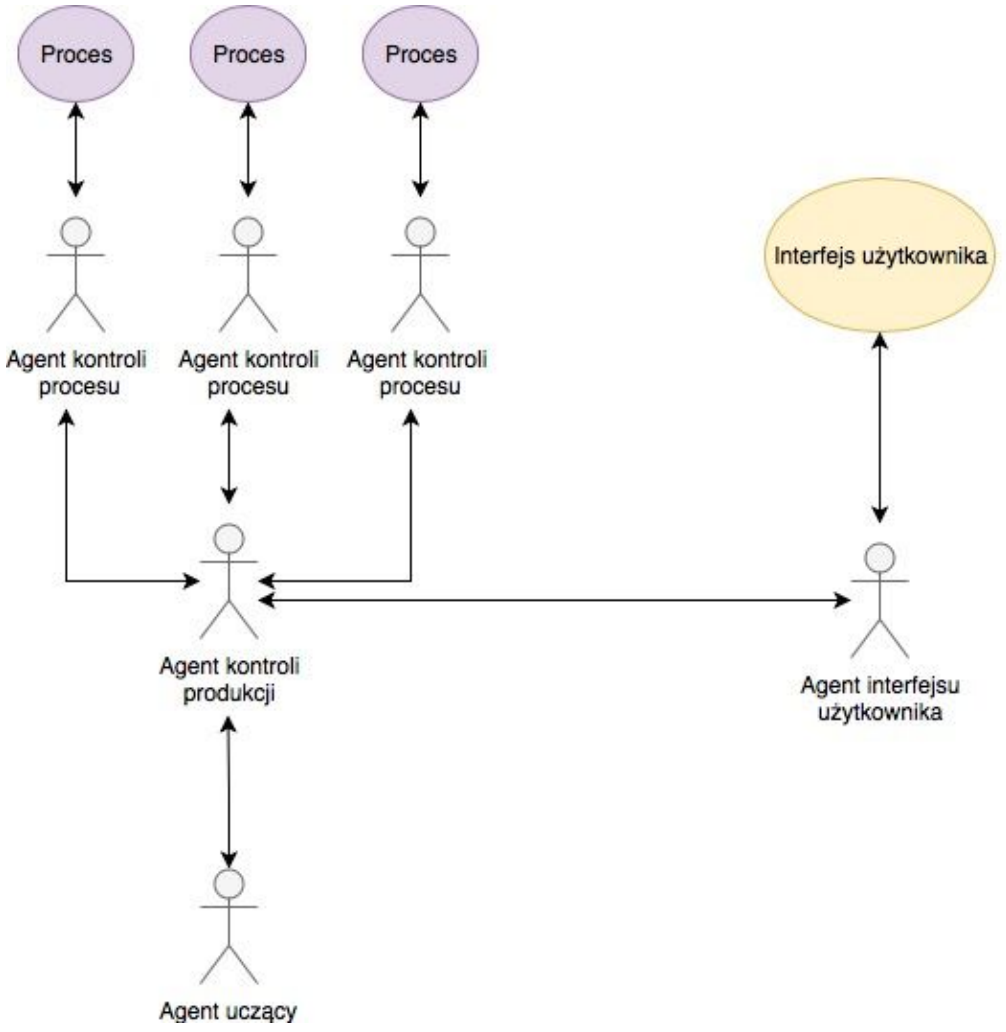
Założyliśmy że najlepszą jakość uzyskamy gdy:

- temperatura wytapiania będzie wyższa o 30% od temperatury topnienia metalu, ponieważ jest to temperatura, w której wszystkie metale ulegną wytopieniu w krótkim czasie,
- czas wytapiania będzie równy 50 minut,
- pierwsza temperatura studzenia będzie równa ⅔ temperatury wytapiania,
- pierwszy czas podgrzewania będzie równy 15 minut,
- druga temperatura studzenia będzie równa ½ pierwszej temperatury studzenia,
- drugi czas podgrzewania będzie równy 25 minut,
- poziom uszlachetniania będzie wynosił 5

4.3 Model agentowy

W utworzonym przez nas modelu agentowym wyszczególniamy:

- agenta odpowiedzialnego za interfejs użytkownika
- agenta odpowiedzialnego za trenowanie klasyfikatorów
- agent kontroli produkcji, który pełni rolę zarządcy
- agent kontroli procesu odpowiedzialny za uzyskanie wyników



rys. 3 - model agentowy

- Świat zewnętrzny:
    - **Proces** - proces produkcyjny kontrolowany przez system agentowy. Proces składa się z trzech etapów. Otrzymany wynik przekazywany jest do agenta kontroli procesu, a następnie do agenta kontroli produkcji.
    - **Użytkownik** - użytkownik otrzymujący dane z systemu na temat procesu i kontrolujący proces poprzez panel kontrolny.

- System agentowy:
    - **Agent kontroli procesu** - agent zajmujący się bezpośrednią komunikacją z procesem. Pobiera dane z sensorów procesu, a następnie wysyła je agenta zajmującego się kontrolą produkcji.

Zajmuje się także kontrolą procesu. Jego cykl życia jest ograniczony przez cykl życia samego procesu produkcyjnego, po jego zakończeniu jest kończony.

- **Agent kontroli produkcji** - agent zajmujący się kontrolą produkcji poszczególnych procesów i zarządzaniem danych, które od nich otrzymuje. Jego czas życia jest zależny od czasu życia całego systemu. Jest startowany na początku i kończy cykl życia na sam koniec.
- **Agent uczący** - agent inicjalizowany przez agenta kontorli produkcji. Jego zadaniem jest nauczenie klasyfikatorów przewidywania wyników na nowych danych. Uczenie następuje na żądanie użytkownika po wygenerowaniu nowych danych.
- **Agent interfejsu użytkownika** - zajmuje się bezpośrednią interakcją z użytkownikiem (opiera się na nim serwis interfejsu użytkownika) i dalszą komunikacją z innymi agentami. Podobnie jak agent kontroli produkcji cykl życia jest tak długi jak cykl życia systemu. Jest inicjalizowany przez agenta produkcji.

4.4 Dane treningowe

4.4.1 Format danych

Dane treningowe wykorzystywane do trenowania jakości mają następującą postać:

Al	Si	Mg	Cu	Sn	Zn	Ni	Fe	Pb	t_1	t_2	t_{s1}	t_{p1}	t_{s2}	t_{p2}	u	j

Al-Pb - procentowy udział poszczególnych składników stopu
t_1 - temperatura wytapiania
t_2 - czas wytapiania
t_{s1} - temperatura studzenia I
t_{p1} - czas podgrzewania I
t_{s2} - temperatura studzenia II
t_{p2} - czas podgrzewania II
u - poziom uszlachetniania
j - jakość

Zapisane są w pliku o formacie .arff w formacie obsługiwanym przez bibliotekę WEKA.
Przykładowy fragment danych treningowych wraz z odpowiednim nagłówkiem znajduje się poniżej:

```
@RELATION MgAlZn.arff

@ATTRIBUTE Aluminium numeric
@ATTRIBUTE Krzem numeric
@ATTRIBUTE Magnez numeric
@ATTRIBUTE Miedz numeric
@ATTRIBUTE Cynk numeric
@ATTRIBUTE Cyna numeric
@ATTRIBUTE Nikiel numeric
@ATTRIBUTE Zelazo numeric
@ATTRIBUTE Olow numeric
@ATTRIBUTE TemperaturaWytapiania numeric
@ATTRIBUTE CzasWytapiania numeric
@ATTRIBUTE TemperaturaStudzenia numeric
@ATTRIBUTE CzasPodgrzewania numeric
@ATTRIBUTE TemperaturaStudzenia2 numeric
@ATTRIBUTE CzasPodgrzewania2 numeric
@ATTRIBUTE StopieńUszlachetniania numeric
@ATTRIBUTE Jakość numeric

@DATA
7,0,90,0,3,0,0,0,0,636,66,345,18,173,28,2,27
7,0,90,0,3,0,0,0,0,496,45,315,19,183,15,4,35
7,0,90,0,3,0,0,0,0,330,44,253,12,111,25,4,6
7,0,90,0,3,0,0,0,0,484,30,358,19,142,19,3,14
7,0,90,0,3,0,0,0,0,538,57,394,19,218,16,5,38
7,0,90,0,3,0,0,0,0,566,31,415,24,178,35,4,24
7,0,90,0,3,0,0,0,0,488,55,375,20,158,37,2,25
7,0,90,0,3,0,0,0,0,404,40,305,24,164,21,5,16
7,0,90,0,3,0,0,0,0,488,31,374,22,199,25,5,18
7,0,90,0,3,0,0,0,0,531,61,361,20,180,30,4,36
7,0,90,0,3,0,0,0,0,408,39,284,24,138,33,5,16
7,0,90,0,3,0,0,0,0,398,35,284,20,136,29,1,4
7,0,90,0,3,0,0,0,0,591,61,357,17,201,35,2,38
7,0,90,0,3,0,0,0,0,764,60,581,18,199,37,1,43
7,0,90,0,3,0,0,0,0,743,53,433,8,212,14,1,57
7,0,90,0,3,0,0,0,0,337,64,237,15,104,23,2,2
7,0,90,0,3,0,0,0,0,359,35,251,15,138,15,4,6
7,0,90,0,3,0,0,0,0,681,38,369,11,246,27,4,41
```

rys. 4 - fragment danych treningowych

Dane wykorzystane do trenowania mają rozmiar około 11 tys wpisów.

4.4.2 Generowanie danych treningowych:

Stworzyliśmy generator, który pozwala na ustawienie parametrów dotyczących składów metali, temperatur, czasów, a także poziomu uszlachetniania. Dzięki temu, przyjmując wyżej wymienione normy i ograniczenia, jesteśmy w stanie ustalić przybliżoną jakość stopu, którą podajemy generatorowi. Pliki są generowanie w formacie .arff w postaci opisanej w powyżej.

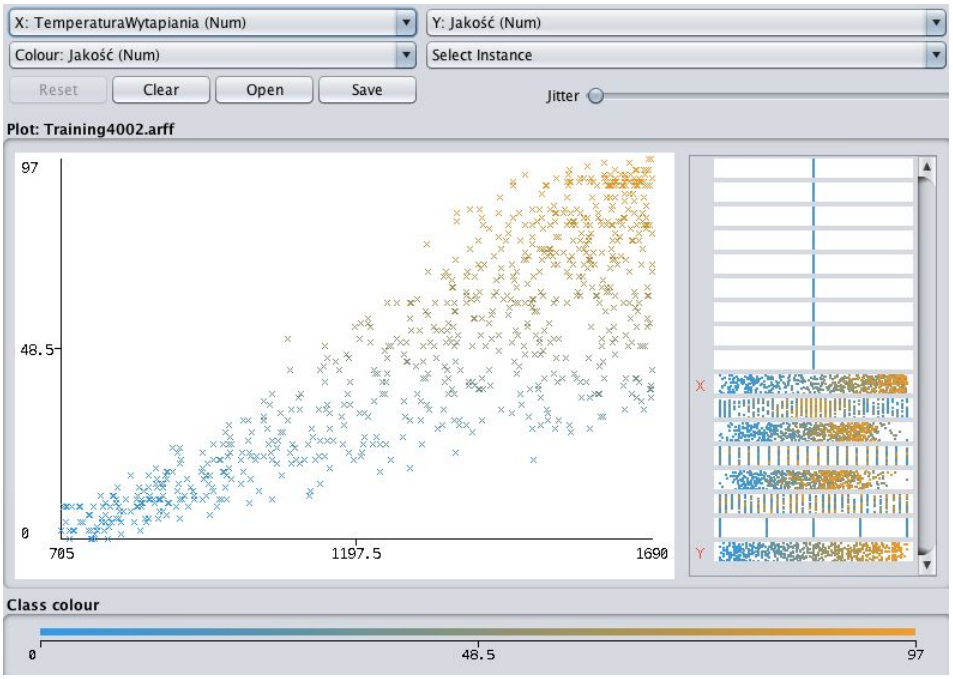
4.4.3 Wzory i zależności:

Losowane są następujące wartości:

- temperatura wytapiania, czas wytapiania, temperatury studzenia, czasy studzenia, poziom uszlachetniania

Wartością wyliczoną na podstawie podanych wyżej wartości jest jakość stopu. Zakresy losowanych wartości i wyznaczanie jakości jest przedstawione poniżej:

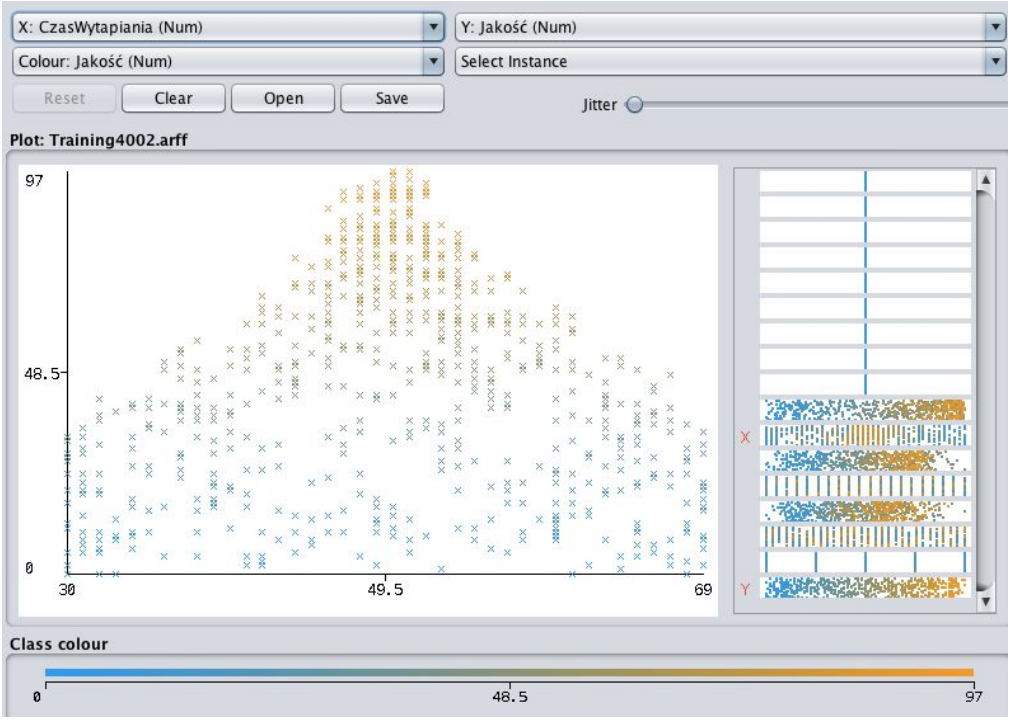
- Temperatura wytapiania:
  - pobierana w zależności od składu metali w stopie; jest to najwyższa temperatura wytapiania spośród metali znajdujących się w stopie,
  - zakres temperatur losowanych przez generator to (t - temperatura topnienia): [0.5*t; 0.5*t + 0.7*t], czyli dla temperatury topnienia = 1000°C mamy zakres [500°C; 1200°C]; wybraliśmy taki przedział temperatur, bo stwierdziliśmy, że nie jest potrzebne generowanie danych dla niższych temperatur, które i tak dadzą jakość 0.
  - losowana jest temperatura wytapiania,
  - zależność od jakości (jakość startowa = 90):
    - t_t - temperatura topnienia,
    - t_w - temperatura wylosowana,
    - t_{min} - minimalny zakres temperatury = $0.5*t_t$,
    - t_z - zakres temperatur = $0.7*t_t$,
    - jeśli $t_w >= t_t$ to jakość += $(t_w - t_t)/(t_{min} + t_z - t_t)/10$ (jakość może wzrosnąć o 10 jednostek)
    - w p.p. jakość -= $(t_t - t_w)/(t_t*0.5/90)$ (jakość może spaść o 90 jednostek)



rys. 5 - wykres zależności jakości od temperatury wytapiania dla stopu aluminium

Wykres przedstawia, jak temperatura wytapiania wpływa na jakość. Widzimy, że zbyt niska temperatura potrafi obniżyć jakość nawet do 0.

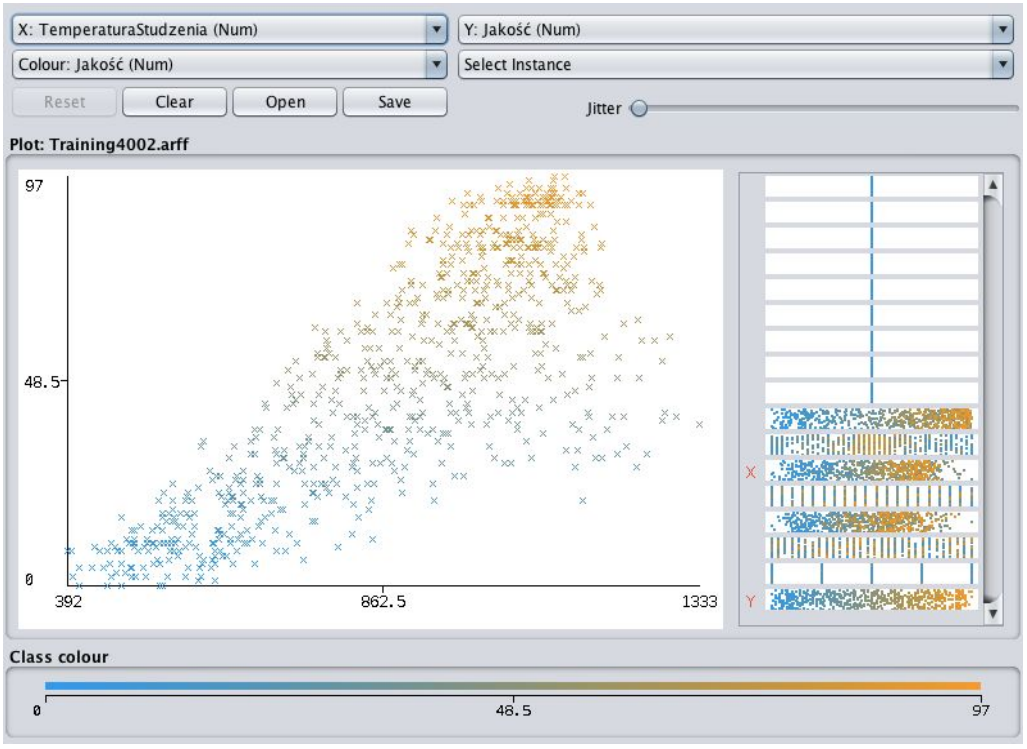
- Czas wytapiania:
    - optymalny czas przyjęliśmy jako 50 minut
    - generator losuje wartości z zakresu 30 minut - 70 minut
    - jakość jest obniżana zgodnie z poniższym wzorem, jeśli czas nie jest równy 50 minut
    - jakość -= |czas - 50|/(20/jakość)



rys. 6 - wykres zależności jakości od czasu wytapiania dla stopu aluminium

Wykres przedstawia, jak czas wytapiania wpływa na jakość. Widać, że optymalny czas to 50 minut. Wraz ze zmianą tego czasu jakość maleje.

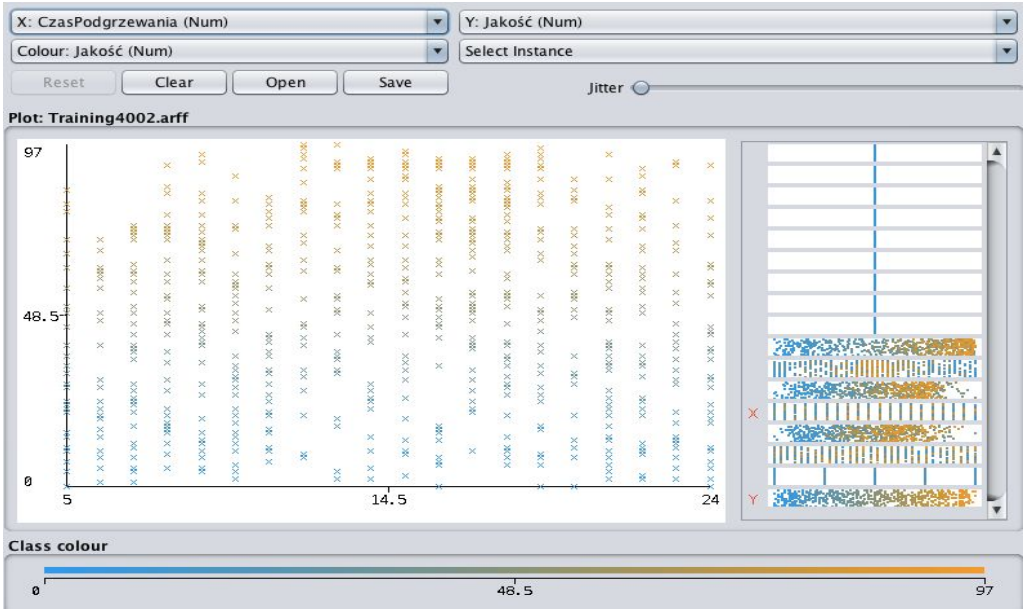
- Temperatura studzenia:
    - optymalna temperatura to ⅔ temperatury topnienia,
    - t_o - optymalna temperatura studzenia,
    - t_s - wylosowana temperatura studzenia,
    - jakość -=$|t_s-t_o|/(((0.25*t_t)/2)/(0.3*\text{jakość}))$;



rys. 7 - wykres zależności jakości od temperatury studzenia dla stopu aluminium

Wykres przedstawia, że najlepszą jakość uzyskamy, gdy temperatura studzenia będzie stanowiła ⅔ temperatury topnienia (w tym przypadku 1410°C), czyli dla około 940°C jakość będzie najlepsza.

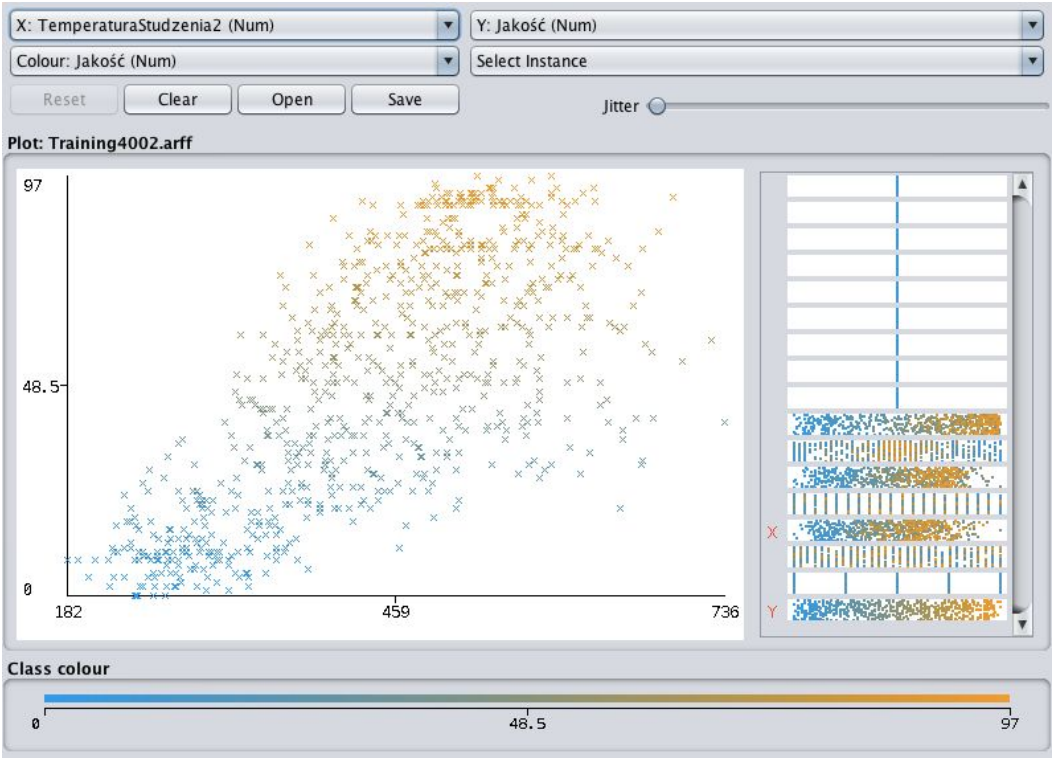
- Czas podgrzewania:
  - optymalny czas to 15 minut,
  - jakość -= |wylosowany czas-15|/(10.0/(0.1*jakość));



rys. 8 - wykres zależności jakości od czasu podgrzewania dla stopu aluminium

- Jakość dla drugiej temperatury studzenia i drugiego czas podgrzewania wyznaczanie są podobnie co dla pierwszych.

Optymalna temperatura drugiego studzenia to ½ pierwszej temperatury studzenia (940°C), co widać na poniższym wykresie - optymalna temperatura to 470°C.



rys. 9 - wykres zależności jakości od drugiej temperatury studzenia dla stopu aluminium

Założony przez nas optymalny czas drugiego podgrzewania to 25 minut. Jakość wyznaczana analogicznie jak dla pierwszego czasu podgrzewania.

- Uszlachetnianie:
  - poziom 1 - nie podwyższa jakości,
  - 2 - podwyższenie o 2,
  - 3 - podwyższenie o 4,
  - 4 - podwyższenie o 6,
  - 5 - podwyższenie o 8

4.4.4 Uczenie jakości stopu

➔ na wejściu dane treningowe
➔ uczymy się określać jakość na podstawie danych parametrów:
- ◆ procentowy skład stopu (oczekiwany)
- ◆ temperatura wytapiania

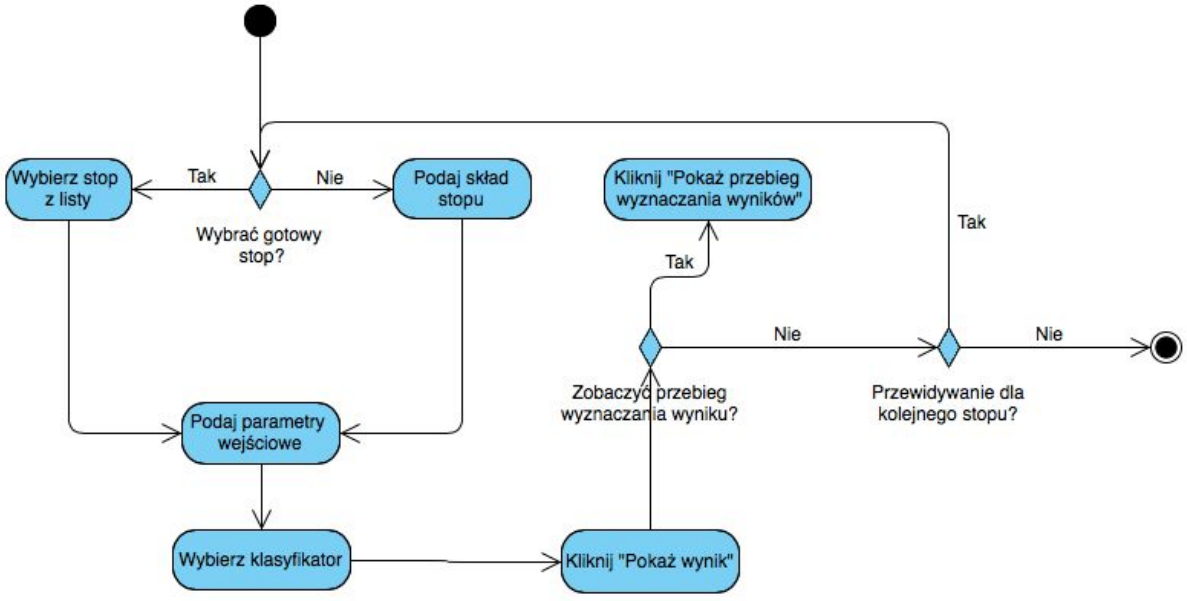
- czas wytapiania
- temperatura studzenia 1
- czas podgrzewania 1
- temperatura studzenia 2
- czas podgrzewania 2
- stopień uszlachetniania w skali 1 do 5

➔ na wyjściu ocena jakości stopu w skali od 1 do 100

5. Implementacja

Program został napisany w języku Java. Interfejs graficzny powstał przy pomocy biblioteki graficznej JavaFX. Moduł uczenia maszynowego został zaimplementowany przy pomocy biblioteki WEKA. Do stworzenia modelu agentowego wykorzystaliśmy bibliotekę JADE.

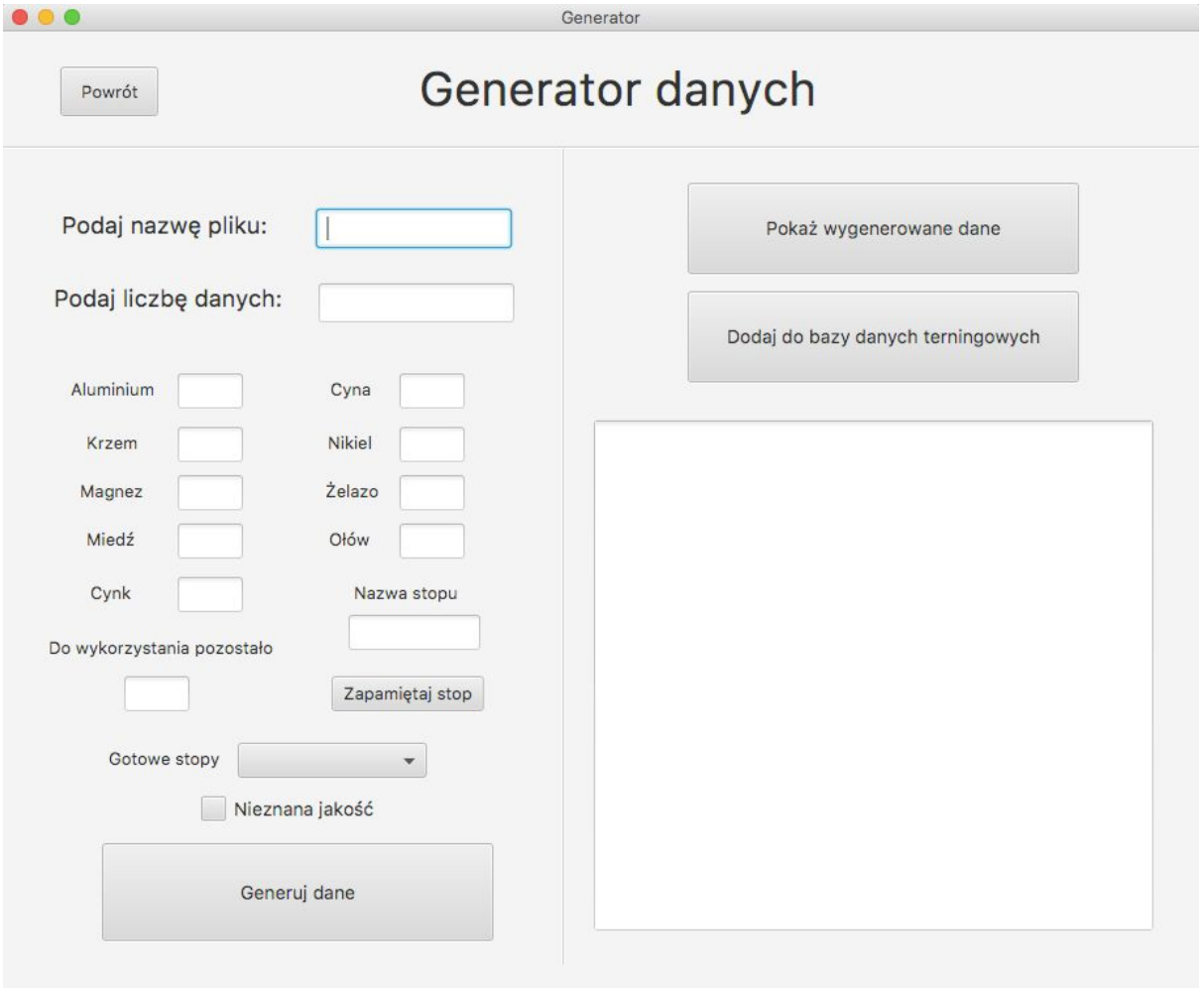
Diagram aktywności dla przewidywania:



rys. 10 - diagram aktywności dla przewidywania

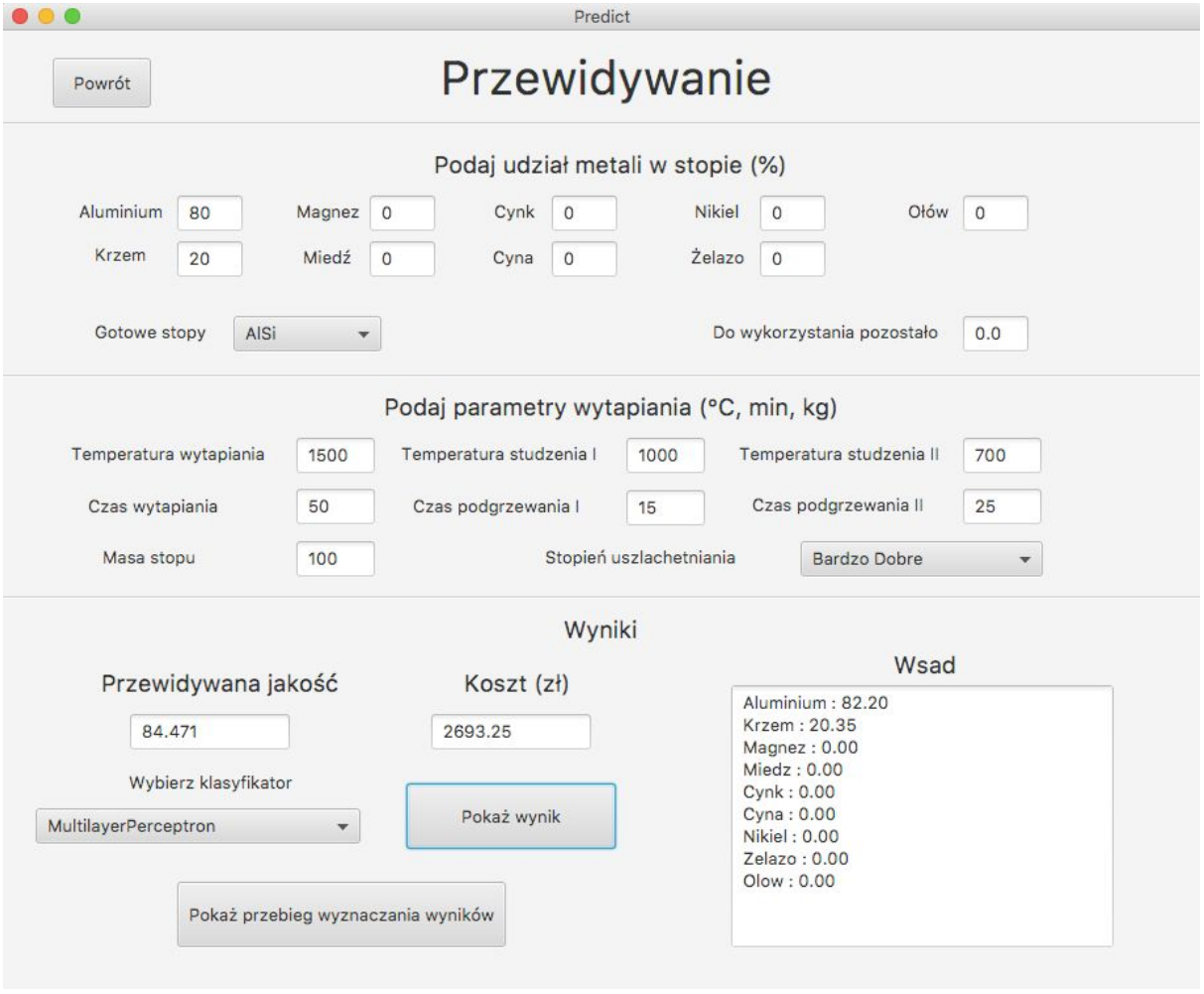
Użytkownik po wejściu w panel przewidywania może wybrać metale z listy lub wpisać własny stop. Następnie musi podać parametry dla procesu wytwarzania stopu. Po kliknięciu w “Pokaż wynik” wyświetli się informacja o otrzymanej jakości, koszcie produkcji oraz o wsadzie. Użytkownik może zobaczyć przebieg, jak wyglądało uczenie maszynowe.

5.1 Interfejs graficzny



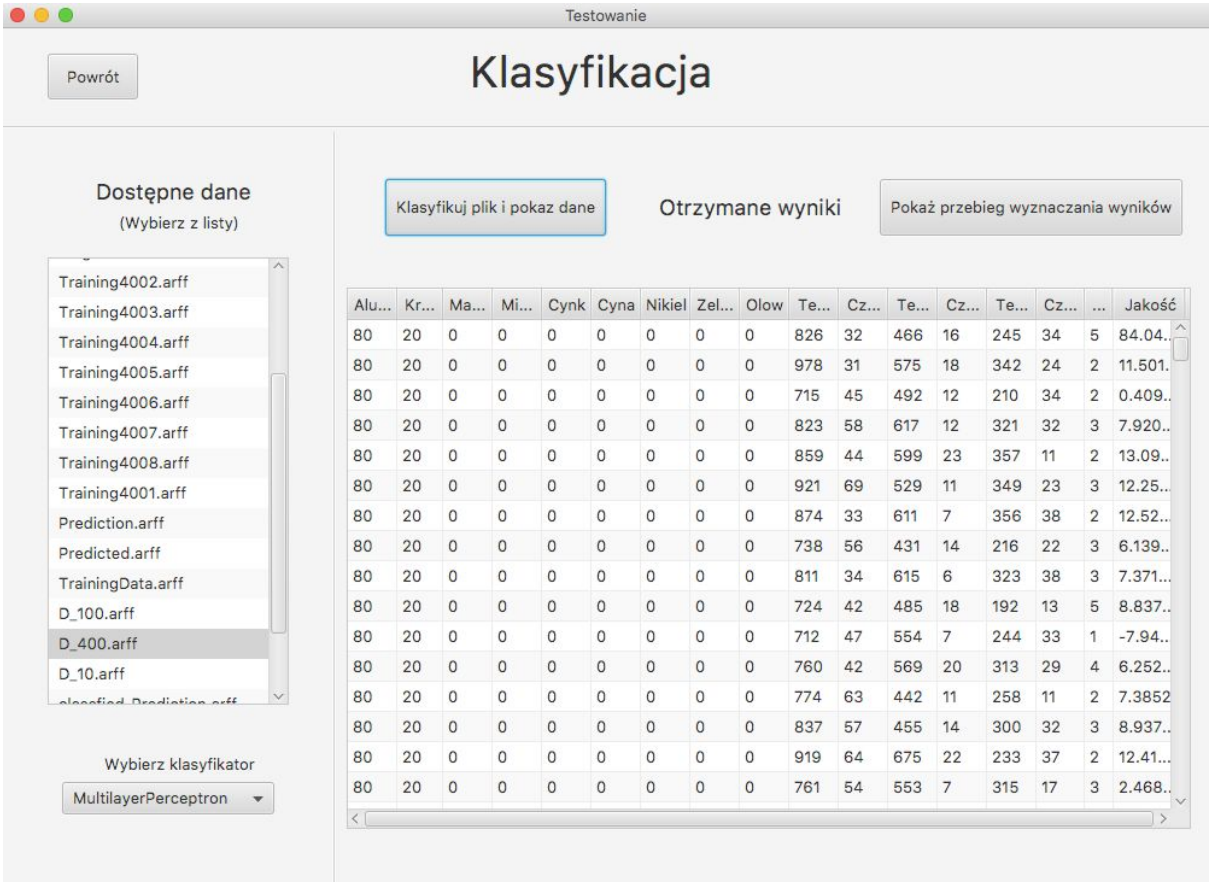
rys. 11 - panel generowania danych

Rysunek przedstawia interfejs do generowania danych. Użytkownik podaje nazwę pliku oraz liczbę danych do wygenerowania. Następnie wybiera gotowy stop lub wpisuje własny stop. Jeśli użytkownik chce żeby nowy stop został zapamiętany, musi kliknąć odpowiedni przycisk. Opcja “Nieznana jakość” powoduje utworzenie danych bez jakości, które potem można sklasyfikować.



rys. 12 - panel przewidywania

Rysunek przedstawia panel dla przewidywania. Użytkownik wybiera gotowy stop lub wpisuje własny stop. Następnie podaje parametry wejściowe. Po kliknięciu w “Pokaż wynik” otrzymuje przewidywaną jakość, koszt, oraz wsad.



rys. 13 - panel klasyfikacji

Rysunek przedstawia panel klasyfikowania, który umożliwia wybranie pliku do sklasyfikowania oraz klasyfikatora. Po kliknięciu “Klasyfikuj plik i pokaż dane” dany plik jest klasyfikowany, dane są zapisywane do nowego pliku z dopiskiem “classified” i wyświetlane w tabeli.

5.2 Uczenie maszynowe

Wykorzystaliśmy bibliotekę WEKA[1]. Dla użytkownika dostępne są klasyfikatory:

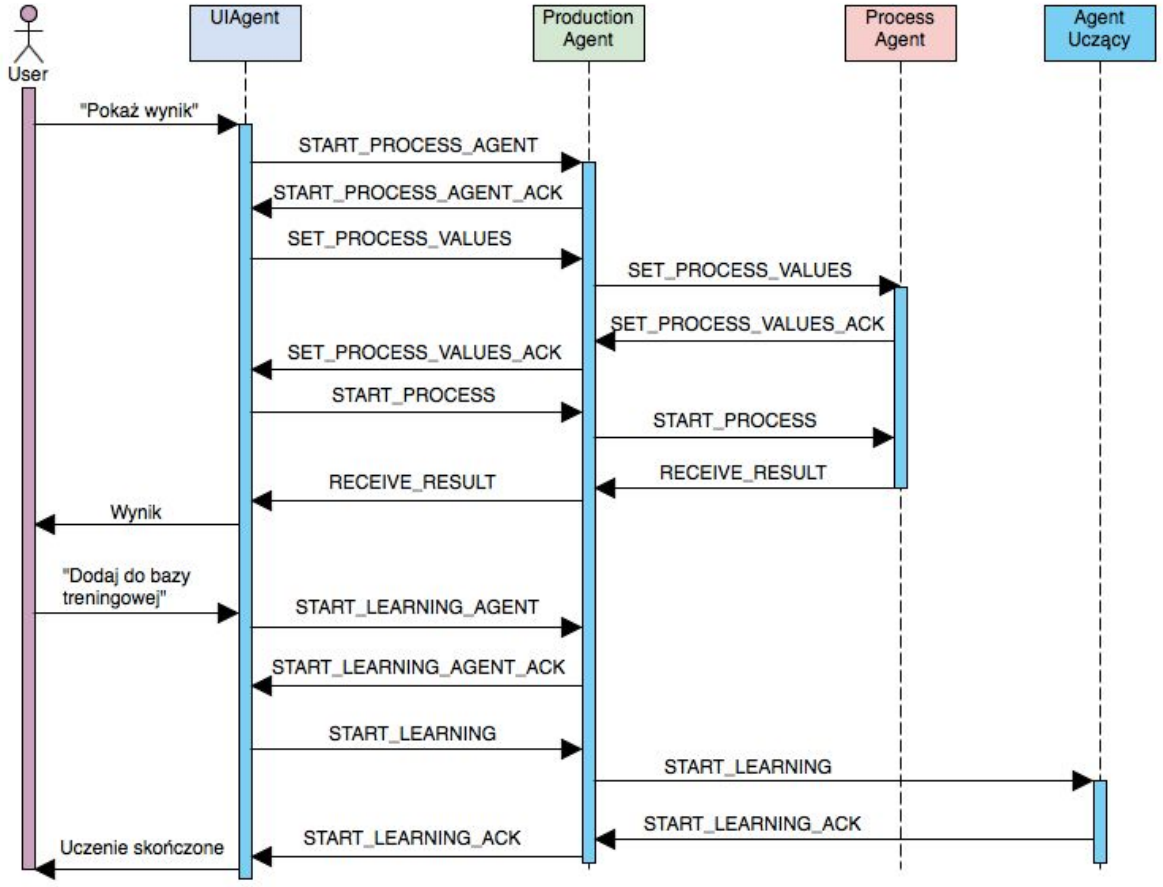
- MultilayerPerceptron
- M5P
- RandomForest
- Vote(średnia z powyższych)

Podczas uruchomienia klasyfikatory są trenowane na pliku z danymi treningowymi w formacie .arff. Podane parametry zapisywane są w pliku .arff, który następnie przetwarzany jest przez wybrany klasyfikator i zwraca nam przewidzianą jakość.

5.3 Model agentowy

Wykorzystaliśmy bibliotekę JADE i stworzyliśmy model opisany w punkcie 4.3.

Komunikacja w systemie agentowym:



rys. 14 - diagram sekwencyjny opisujący model agentowy

Diagram przedstawia komunikaty przesyłane przez agentów i co powoduje ich wysyłanie. Poniżej zostały opisane poszczególne komunikaty oraz zostały przedstawione pseudokody zachowań poszczególnych agentów.

- **Komunikaty agenta kontroli procesu:**

Komunikat	**Rola**	**Rodzaj**	**Powiązany agent**
CHECK_AGENT	Inicjalizacja i konfiguracja nowego procesu produkcyjnego	Przychodzący	Agent kontroli produkcji
START_PROCESS	Rozpoczęcie wcześniej skonfigurowanego procesu	Przychodzący	Agent kontroli produkcji
RECEIVE_RESULT	Informacje o skończonym procesie	Wychodzący	Agent kontroli produkcji
SET_PROCESS_VALUES	Zmiana parametrów produkcji	Przychodzący	Agent kontroli produkcji
SET_PROCESS_VALUES_ACK	Odpowiedź na zmianę parametrów produkcji	Wychodzący	Agent kontroli produkcji

Pseudokod:

```
CyclicBehaviour
      switch(msg)
            case(SET_PROCESS_VALUES)
                  setProcessValues()
                  send(SET_PROCESS_VALUES_ACK)
            case(START_PROCESS)
                  startProcess()
                  send(RECEIVE_RESULT)
```

- **Komunikaty agenta kontroli produkcji:**

Komunikat	**Rola**	**Rodzaj**	**Powiązany agent**
START_PROCESS_AGENT	Zapytanie o stworzenie nowego procesu	Przychodzący	Agent interfejsu użytkownika

START_PROCESS_AGENT_ACK	Odpowiedź na zapytanie o stworzenie nowego agenta	Wychodzący	Agent interfejsu użytkownika
START_PROCESS	Komenda wystartowania procesu	Przychodzący	Agent interfejsu użytkownika
START_PROCESS	Komenda wystartowania procesu	Wychodzący	Agent kontroli procesu
SET_PROCESS_VALUES	Zapytanie o ustawienie parametru procesu	Przychodzący	Agent interfejsu użytkownika
SET_PROCESS_VALUES_ACK	Potwierdzenie ustawienia parametrów	Wychodzący	Agent interfejsu użytkownika
SET_PROCESS_VALUES	Komenda ustawienia parametru procesu	Wychodzący	Agent kontroli procesu
SET_PROCESS_VALUES_ACK	Potwierdzenie ustawienia parametrów	Przychodzący	Agent kontroli procesu
START_LEARNING_AGENT	Komenda utworzenia agenta uczącego	Przychodzący	Agent interfejsu użytkownika
START_LEARNING_AGENT_ACK	Komenda potwierdzenia utworzenia agenta uczącego	Wychodzący	Agent interfejsu użytkownika
START_LEARNING	Komenda wystartowania uczenia maszynowego	Przychodzący	Agent interfejsu użytkownika
START_LEARNING	Komenda wystartowania uczenia maszynowego	Wychodzący	Agent uczący
START_LEARNING_ACK	Komenda potwierdzenia wykonania uczenia maszynowego	Przychodzący	Agent uczący
START_LEARNING_ACK	Komenda potwierdzenia wykonania uczenia maszynowego	Wychodzący	Agent interfejsu użytkownika

RECEIVE_RESULT	Informacje o skończonym procesie	Przychodzący	Agent kontroli procesu
RECEIVE_RESULT	Informacje o skończonym procesie	Wychodzący	Agent interfejsu użytkownika

Pseudokod:

```
CyclicBehaviour
      switch(msg)
            case(START_PROCESS_AGENT)
                  createProcessAgent()
                  send(START_PROCESS_AGENT_ACK)
            case(START_LEARNING_AGENT)
                  createLearningAgent()
                  send(START_LEARNING_AGENT_ACK)
            case(SET_PROCESS_VALUES)
                  send(SET_PROCESS_VALUES)
            case(SET_PROCESS_VALUES_ACK)
                  send(SET_PROCESS_VALUES_ACK)
            case(START_PROCESS)
                  send(START_PROCESS)
            case(START_LEARNING)
                  send(START_LEARNING)
            case(START_LEARNING_ACK)
                  send(START_LEARNING_ACK)
            case(RECEIVE_RESULT)
                  send(RECEIVE_RESULT)
```

- **Komunikaty agenta uczącego:**

Komunikat	**Rola**	**Rodzaj**	**Powiązany agent**
CHECK_AGENT	Sprawdzenie działania agenta uczącego	Przychodzący	Agent kontroli produkcji
START_LEARNING	Wystartowanie uczenia maszynowego	Przychodzący	Agent kontroli produkcji
START_LEARNING_ACK	Zakończenie uczenia maszynowego	Wychodzący	Agent kontroli produkcji

Pseudokod:

```
CyclicBehaviour
      switch(msg)
            case(START_LEARNING)
                  startLearning()
                  send(START_LEARNING_ACK)
```

- **Komunikaty interfejsu użytkownika:**

Komunikat	**Rola**	**Rodzaj**	**Powiązany agent**
START_PROCESS_AGENT	Zapytanie o stworzenie rejestru dla nowego procesu	Wychodzący	Agent kontroli produkcji
START_PROCESS_AGENT_ACK	Odpowiedź na zapytanie o stworzenie rejestru dla nowego procesu	Przychodzący	Agent kontroli produkcji
SET_PROCESS_VALUES	Ustawienie parametrów procesu	Wychodzący	Agent kontroli produkcji
SET_PROCESS_VALUES_ACK	Odpowiedź na zapoytanie o ustawienie parametrów procesu	Przychodzący	Agent kontroli produkcji
START_LEARNING_AGENT	Komenda utworzenia agenta uczącego	Wychodzący	Agent kontroli produkcji
START_LEARNING_AGENT_ACK	Potwierdzenie utworzenia agenta uczącego	Przychodzący	Agent kontroli produkcji
START_LEARNING	Komenda wystartowania uczenia maszynowego	Wychodzący	Agent kontroli produkcji
START_LEARNING_ACK	Potwierdzenie zakończenia uczenia maszynowego	Przychodzący	Agent kontroli produkcji
RECEIVE_RESULT	Informacje o skończonym procesie	Przychodzący	Agent kontroli produkcji

Pseudokod:

```
runProcess()
      behaviour
            send(START_PROCESS_AGENT)
            receive(START_PROCESS_AGENT_ACK)
            send(SET_PROCESS_VALUES)
            receive(SET_PROCESS_VALUES_ACK)
            send(START_PROCESS)
            receive(RECEIVE_RESULT)


startTraining()
      behaviour
            send(START_LEARNING_AGENT)
            receive(START_LEARNING_AGENT_ACK)
            send(START_LEARNING)
            receive(START_LEARNING_ACK)
```

6. Testowanie

6.1 Dane treningowe i eksperymenty

Eksperymenty zostały przeprowadzone w środowisku Weka (explorer) na różnych zbiorach danych treningowych, które wygenerowaliśmy. Eksperymenty dotyczyły klasyfikatorów. Pierwszy zbiór treningowy zawierał ok. 11 tys. danych, drugi 3600, trzeci 900, a czwarty 90.

6.2 Klasyfikatory

Aby wybrać najlepsze klasyfikatory, dla naszych danych przeprowadziliśmy eksperymenty na różnych klasyfikatorach i różnych zbiorach danych.

θ_i - wartości docelowe
$\hat{\theta}_i$ - wartości przewidziane
$\bar{\theta}$ - średnia wartości θ_i

Wyniki[14], które uzyskaliśmy w testach to:

- współczynnik korelacji, który określa nam jak bardzo wartości docelowe i wartości przewidziane są powiązane(1- bardzo silna liniowa relacja)
- Mean absolute error:

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{\theta}_i - \theta_i|$$

- Root mean square error:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}$$

- Relative absolute error:

$$\mathrm{RAE} = \frac{\sum_{i=1}^{N}\left|\hat{\theta}_i - \theta_i\right|}{\sum_{i=1}^{N}\left|\bar{\theta} - \theta_i\right|}$$

- Root relative squared error:

$$\mathrm{RRSE} = \sqrt{\frac{\sum_{i=1}^{N}\left(\hat{\theta}_i - \theta_i\right)^2}{\sum_{i=1}^{N}\left(\bar{\theta} - \theta_i\right)^2}}$$

Uzyskaliśmy również wykresy prezentujące błędy (oś X - rzeczywista jakość, oś Y - przewidziana jakość, kolor żółty - najlepsza jakość, kolor niebieski - najsłabsza jakość).

I. Pierwszy eksperyment.

Eksperyment przeprowadzono na 11 tys. danych treningowych.

Zbiór danych został podzielony następująco: 70% - dane treningowe, 30% - dane testowe.

- **SimpleLinearRegression:**

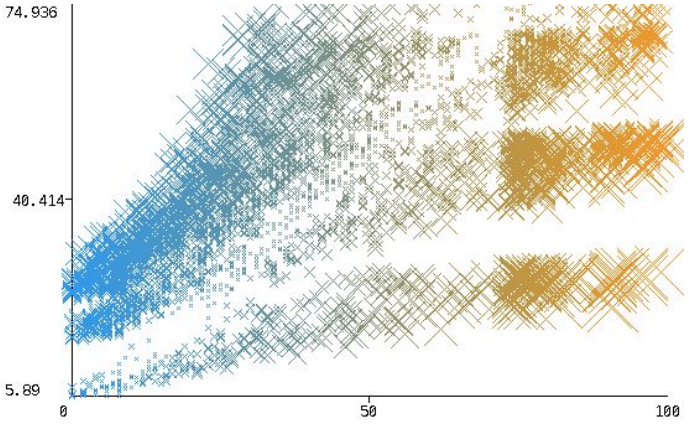
```
Time taken to build model: 0.08 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.6222
Mean absolute error                     17.1616
Root mean squared error                 21.1201
Relative absolute error                 74.8823 %
Root relative squared error             78.2865 %
Total Number of Instances             3387
```



- **LinearRegression:**

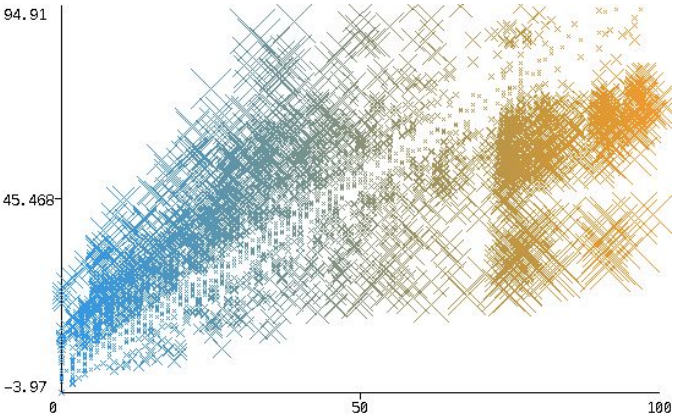
```
Time taken to build model: 0.04 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0 seconds

=== Summary ===

Correlation coefficient                  0.7512
Mean absolute error                     13.921
Root mean squared error                 17.8106
Relative absolute error                 60.7426 %
Root relative squared error             66.0191 %
Total Number of Instances             3387
```



- **MultilayerPerceptron:**

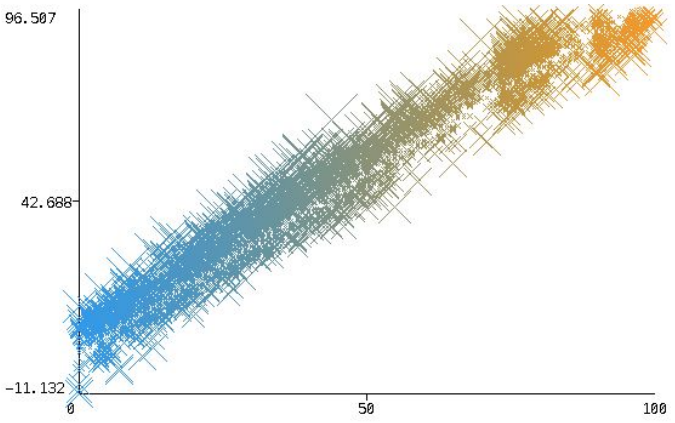
```
Time taken to build model: 16.17 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.982
Mean absolute error                      4.4588
Root mean squared error                  5.5661
Relative absolute error                 19.4553 %
Root relative squared error             20.6319 %
Total Number of Instances             3387
```



- **SMOreg:**

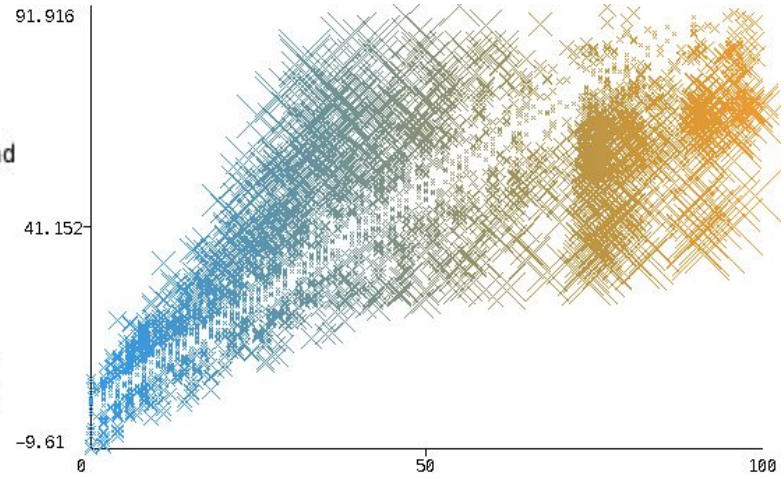
```
Time taken to build model: 279.35 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.06 second

=== Summary ===

Correlation coefficient                  0.8154
Mean absolute error                     11.8309
Root mean squared error                 15.6801
Relative absolute error                 51.6224 %
Root relative squared error             58.1218 %
Total Number of Instances             3387
```



- **IBk:**

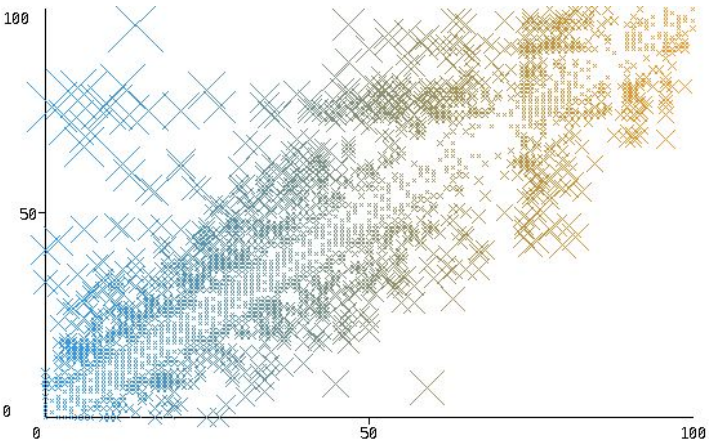
```
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 2.69 seconds

=== Summary ===

Correlation coefficient                  0.898
Mean absolute error                      9.2008
Root mean squared error                 12.6471
Relative absolute error                 40.1463 %
Root relative squared error             46.8793 %
Total Number of Instances             3387
```



- **KStar:**

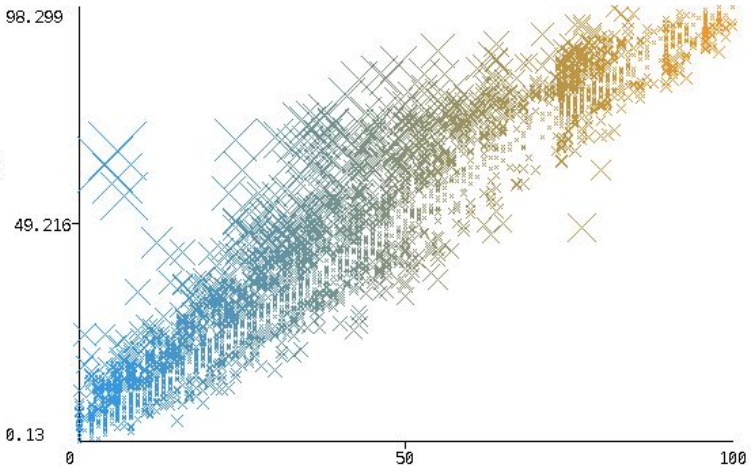
```
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 150.13 seconds

=== Summary ===

Correlation coefficient                  0.9553
Mean absolute error                      6.462
Root mean squared error                  9.0614
Relative absolute error                 28.196  %
Root relative squared error             33.588  %
Total Number of Instances             3387
```



- **LWL:**

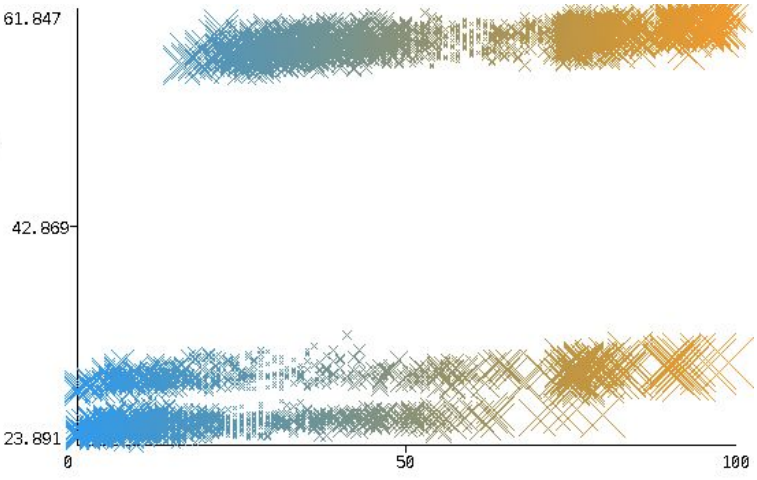
```
Time taken to build model: 0.01 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 70.33 seconds

=== Summary ===

Correlation coefficient                  0.6625
Mean absolute error                     16.9807
Root mean squared error                 20.3067
Relative absolute error                 74.093  %
Root relative squared error             75.2714 %
Total Number of Instances             3387
```



- **DecisionTable:**

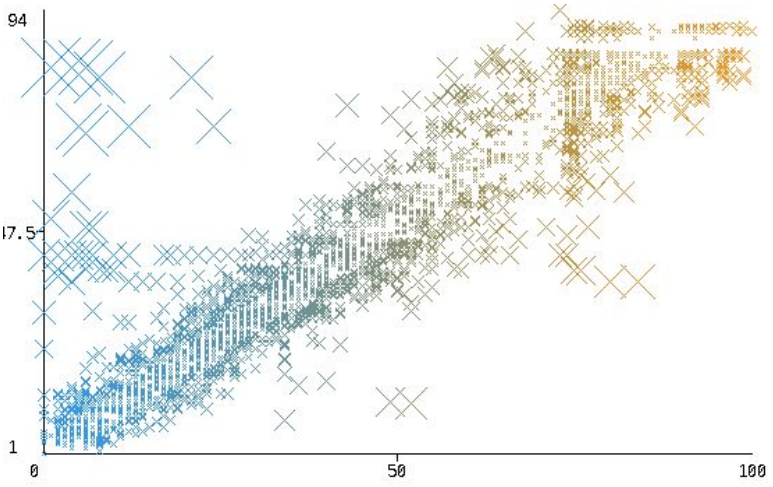
```
Time taken to build model: 1.6 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.04 seconds

=== Summary ===

Correlation coefficient                  0.9487
Mean absolute error                      5.73
Root mean squared error                  8.5345
Relative absolute error                 25.002  %
Root relative squared error             31.6352 %
Total Number of Instances             3387
```



- **M5Rules:**

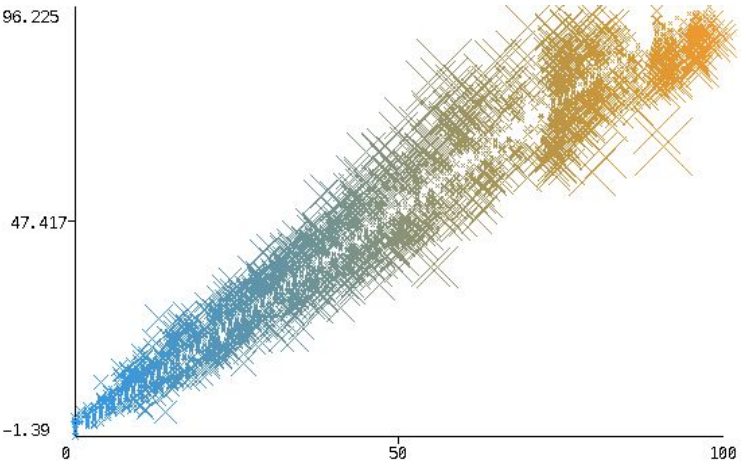
```
Time taken to build model: 31.49 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.18 seconds

=== Summary ===

Correlation coefficient                  0.9793
Mean absolute error                      4.0912
Root mean squared error                  5.467
Relative absolute error                 17.8514 %
Root relative squared error             20.2646 %
Total Number of Instances             3387
```



- **DecisionStump:**

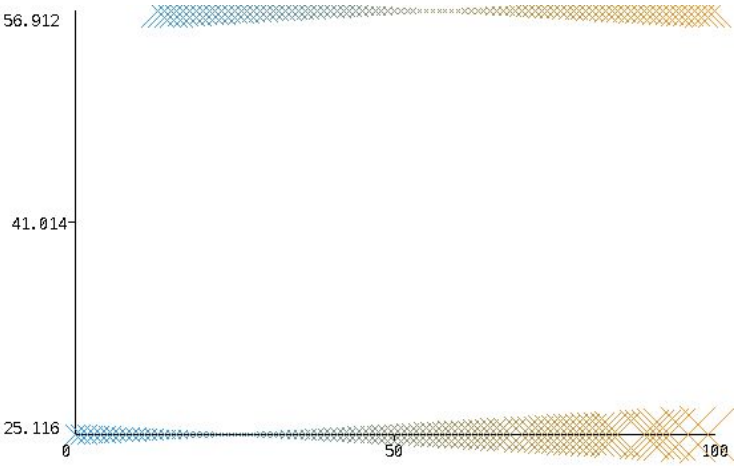
```
Time taken to build model: 0.16 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correlation coefficient                  0.5961
Mean absolute error                     17.9234
Root mean squared error                 21.6638
Relative absolute error                 78.2065 %
Root relative squared error             80.3017 %
Total Number of Instances             3387
```



- **M5P:**

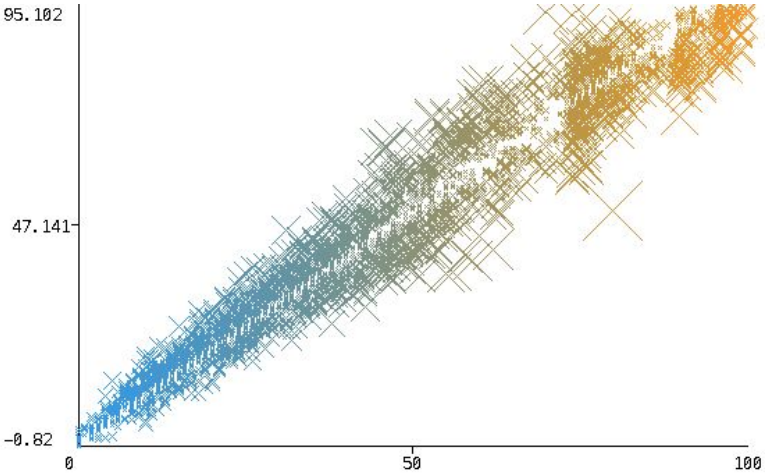
```
Time taken to build model: 2.19 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.982
Mean absolute error                      3.8599
Root mean squared error                  5.1078
Relative absolute error                 16.8422 %
Root relative squared error             18.9333 %
Total Number of Instances             3387
```



- **RandomForest:**

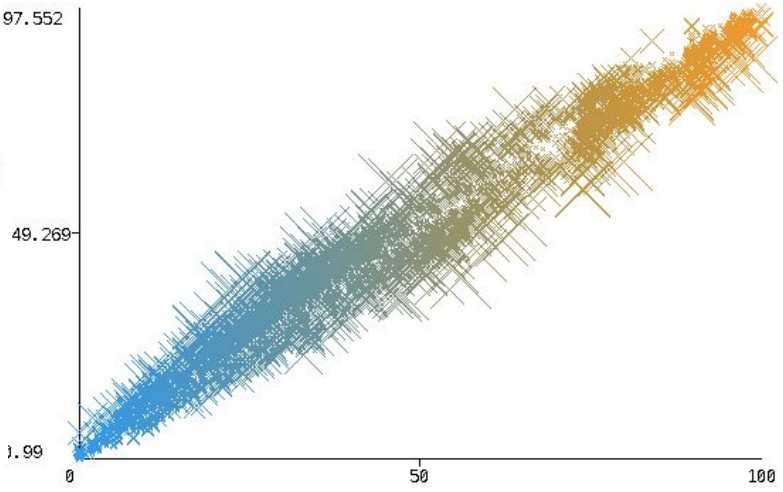
```
Time taken to build model: 6.42 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.49 seconds

=== Summary ===

Correlation coefficient                  0.9876
Mean absolute error                      3.3245
Root mean squared error                  4.3621
Relative absolute error                 14.5059 %
Root relative squared error             16.1693 %
Total Number of Instances             3387
```



- **RandomTree:**

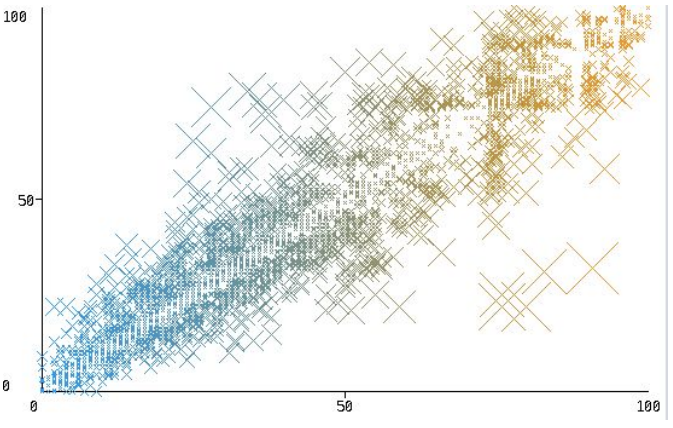
```
Time taken to build model: 0.23 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.9527
Mean absolute error                      5.8745
Root mean squared error                  8.3304
Relative absolute error                 25.6327 %
Root relative squared error             30.8785 %
Total Number of Instances             3387
```



- **REPTree:**

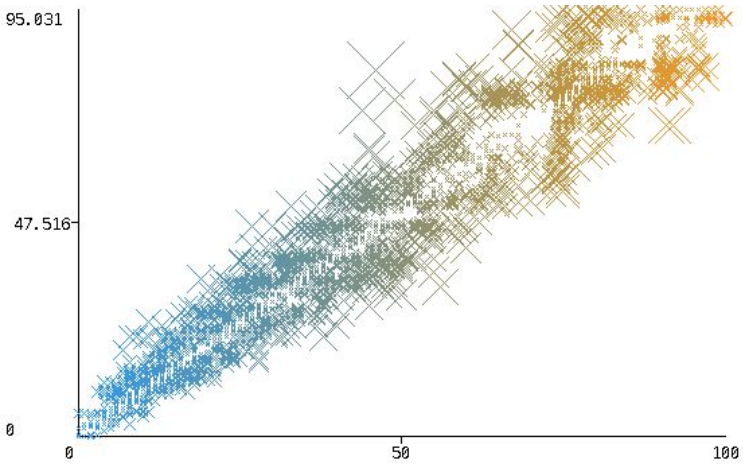
```
Time taken to build model: 0.54 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.02 seconds

=== Summary ===

Correlation coefficient                  0.9752
Mean absolute error                      4.4639
Root mean squared error                  5.9967
Relative absolute error                 19.4775 %
Root relative squared error             22.2282 %
Total Number of Instances             3387
```



- **AdditiveRegression:**

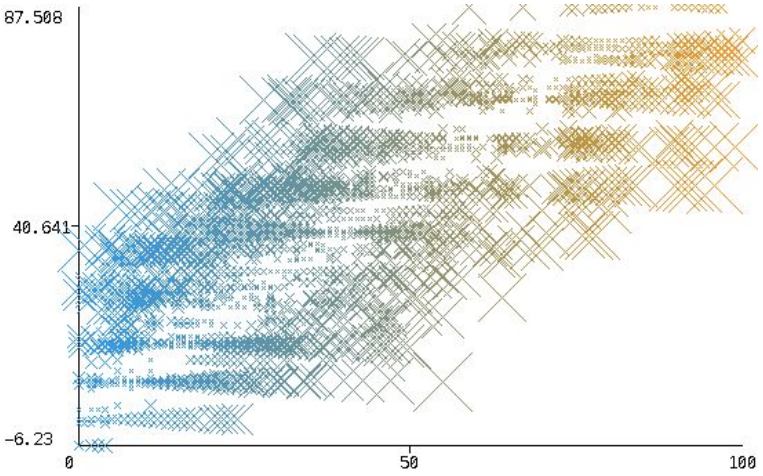
```
Time taken to build model: 0.31 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.01 seconds

=== Summary ===

Correlation coefficient                  0.8526
Mean absolute error                     11.2901
Root mean squared error                 14.1157
Relative absolute error                 49.2628 %
Root relative squared error             52.3232 %
Total Number of Instances             3387
```



- **Bagging:**

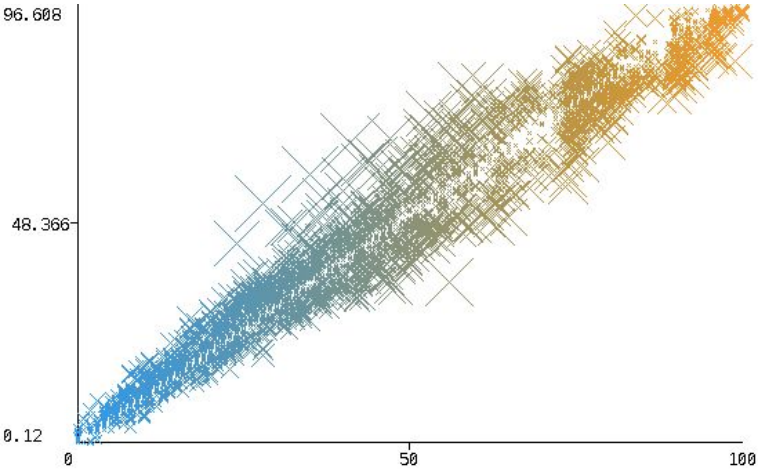
```
Time taken to build model: 1.21 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 0.05 seconds

=== Summary ===

Correlation coefficient                  0.9846
Mean absolute error                      3.5094
Root mean squared error                  4.7341
Relative absolute error                 15.313  %
Root relative squared error             17.5481 %
Total Number of Instances             3387
```



- **RandomizableFilteredClassifier:**

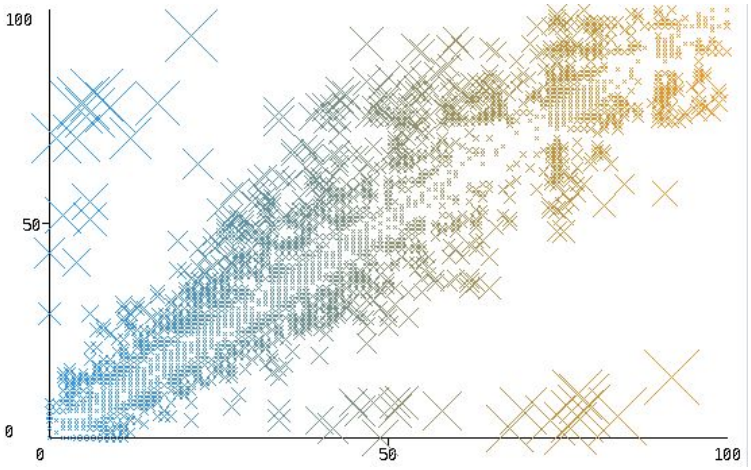
```
Time taken to build model: 0.07 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 1.66 seconds

=== Summary ===

Correlation coefficient                  0.9136
Mean absolute error                      7.8541
Root mean squared error                 11.5741
Relative absolute error                 34.2705 %
Root relative squared error             42.902  %
Total Number of Instances             3387
```



- **RegressionByDiscretization:**

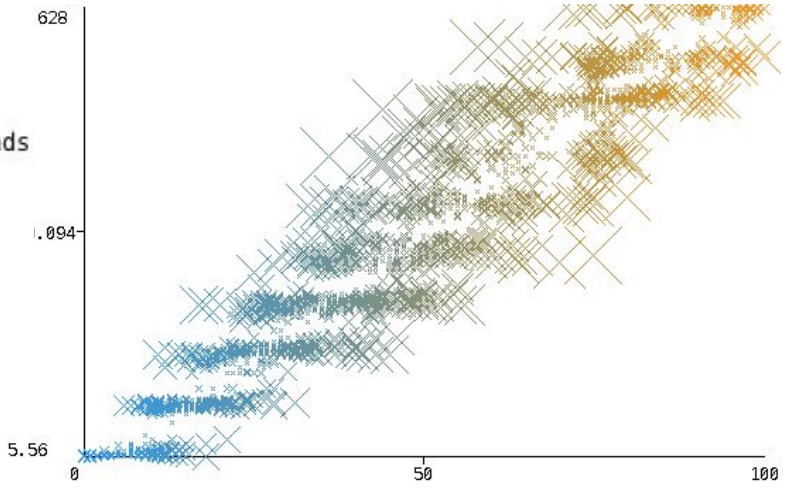
```
Time taken to build model: 1.43 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 19.06 seconds

=== Summary ===

Correlation coefficient                  0.9719
Mean absolute error                      4.7714
Root mean squared error                  6.3919
Relative absolute error                 20.8192 %
Root relative squared error             23.6929 %
Total Number of Instances             3387
```



- **Vote** (Na podstawie **RandomForest**, **Bagging**, **M5P**, **MultilayerPerceptron**):

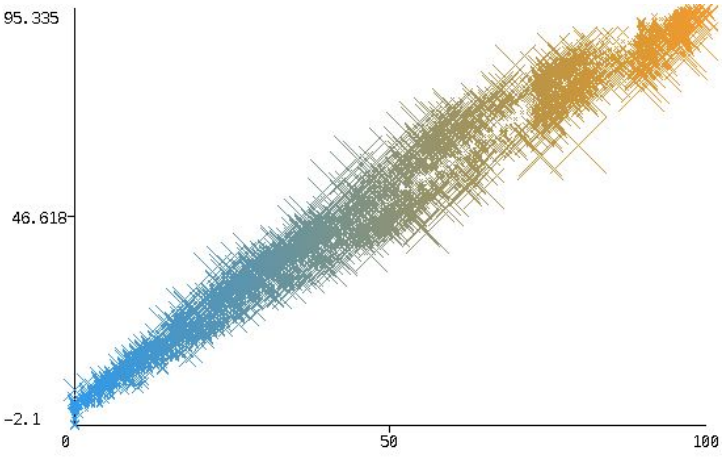
```
Time taken to build model: 35.34 seconds

=== Evaluation on test split ===

Time taken to test model on test split: 1.62 seconds

=== Summary ===

Correlation coefficient                  0.9907
Mean absolute error                      2.9481
Root mean squared error                  3.8178
Relative absolute error                 12.8636 %
Root relative squared error             14.1514 %
Total Number of Instances             3387
```



Wyniki posortowane wg współczynnika korelacji:

Klasyfikator	Współczynnik korelacji	Czas potrzebny na zbudowanie modelu [s]	Czas potrzebny na testowanie [s]
Vote	0.9907	35.34	1.62
RandomForest	0.9876	6.42	0.49
Bagging	0.9846	1.21	0.05
M5P	0.982	2.19	0.01
MultilayerPerceptron	0.982	16.17	0.01
M5Rules	0.9793	31.49	0.18
REPTree	0.9752	0.54	0.02
RegressionByDiscretization	0.9719	1.43	19.06
KStar	0.9553	0.01	150.13
RandomTree	0.9527	0.23	0.01
DecisionTable	0.9487	1.6	0.04
RandomizableFilteredClassifier	0.9136	0.07	1.66
IBk	0.898	0.01	2.69
AdditiveRegression	0.8526	0.31	0.01
SMOreg	0.8154	279.35	0.06
LinearRegression	0.7512	0.04	0
LWL	0.6625	0.01	70.33
SimpleLinearRegression	0.6222	0.08	0.01
DecisionStump	0.5961	0.16	0.02

II. Drugi eksperyment.

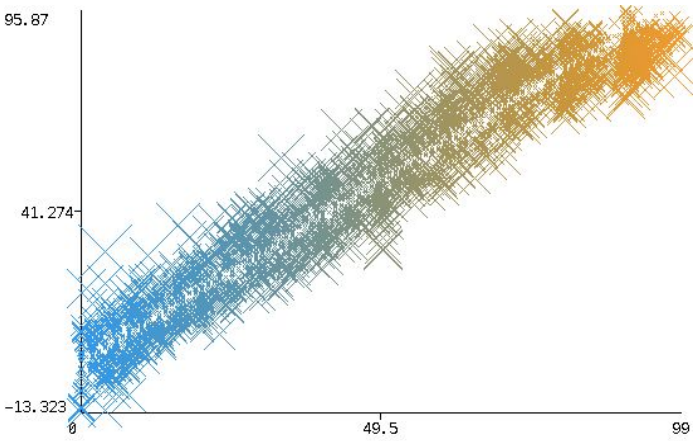
Eksperyment przeprowadzono na 3600 danych treningowych. W tym eksperymencie użyto 10 folds cross-validation.

- **MultilayerPerceptron:**

```
Time taken to build model: 7.68 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9764
Mean absolute error                      4.7997
Root mean squared error                  6.0409
Relative absolute error                 19.7769 %
Root relative squared error             21.6287 %
Total Number of Instances             3600
```

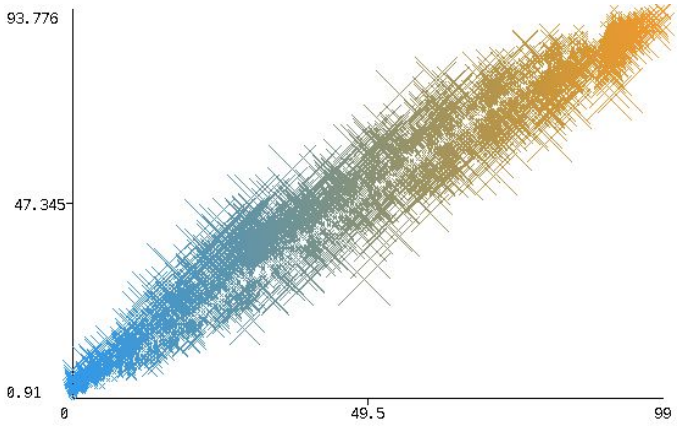


- **RandomForest:**

```
Time taken to build model: 2.36 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9834
Mean absolute error                      4.1626
Root mean squared error                  5.2982
Relative absolute error                 17.1518 %
Root relative squared error             18.9697 %
Total Number of Instances             3600
```

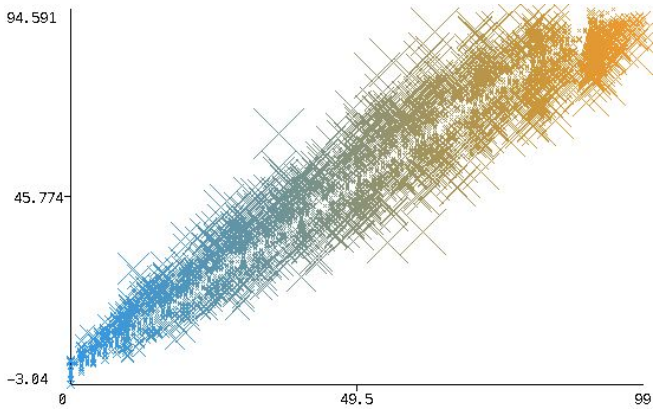


- **M5P:**

```
Time taken to build model: 1.46 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9769
Mean absolute error                      4.6197
Root mean squared error                  5.9907
Relative absolute error                 19.0351 %
Root relative squared error             21.4491 %
Total Number of Instances             3600
```

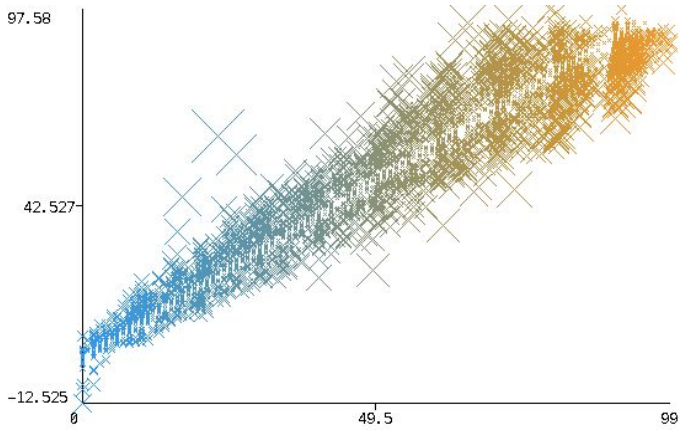


- **M5Rules:**

```
Time taken to build model: 3.33 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9721
Mean absolute error                      4.954
Root mean squared error                  6.5476
Relative absolute error                 20.4126 %
Root relative squared error             23.4428 %
Total Number of Instances             3600
```

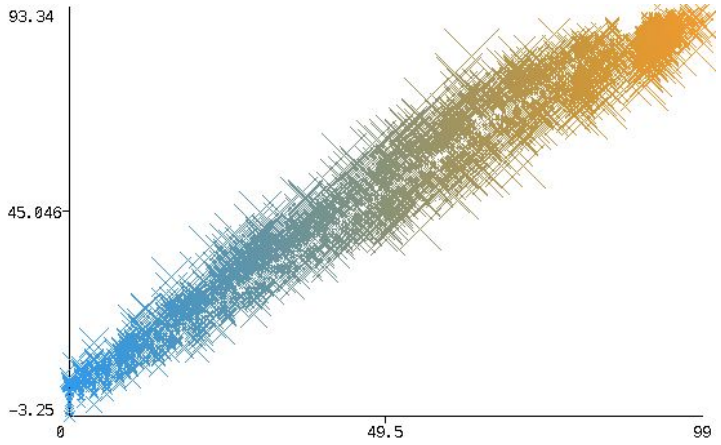


- **Vote:**

```
Time taken to build model: 9.34 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9867
Mean absolute error                      3.6234
Root mean squared error                  4.64
Relative absolute error                 14.9297 %
Root relative squared error             16.613  %
Total Number of Instances             3600
```



Klasyfikator	**Współczynnik korelacji**	**Czas potrzebny na zbudowanie modelu [s]**
Vote	0.9867	9.34
RandomForest	0.9834	2.36
M5P	0.9769	1.46
MultilayerPerceptron	0.9764	7.68
M5Rules	0.9721	3.33

III. Trzeci eksperyment.

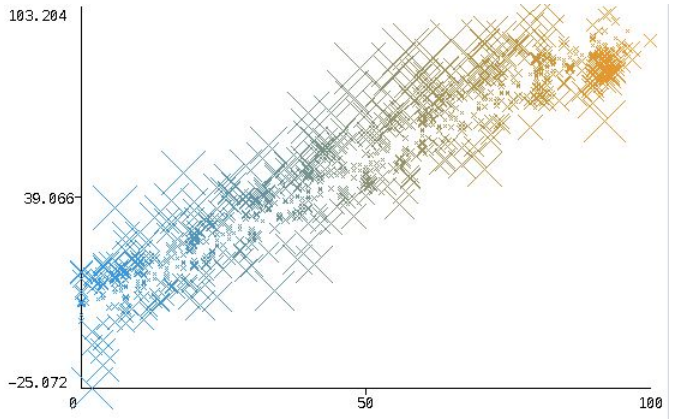
Eksperyment przeprowadzono na 900 danych treningowych. W tym eksperymencie użyto 10 folds cross-validation.

- **MultilayerPerceptron:**

```
Time taken to build model: 1.8 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9483
Mean absolute error                      7.0875
Root mean squared error                  9.1036
Relative absolute error                 29.0804 %
Root relative squared error             32.528  %
Total Number of Instances              900
```

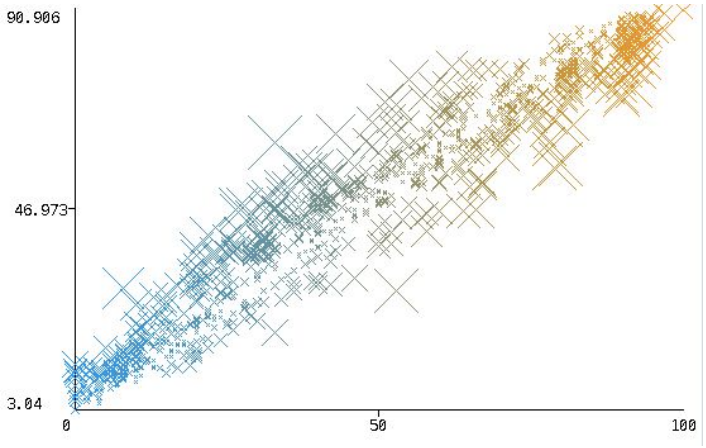


- **RandomForest:**

```
Time taken to build model: 0.38 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9686
Mean absolute error                      5.871
Root mean squared error                  7.445
Relative absolute error                 24.0889 %
Root relative squared error             26.6017 %
Total Number of Instances              900
```

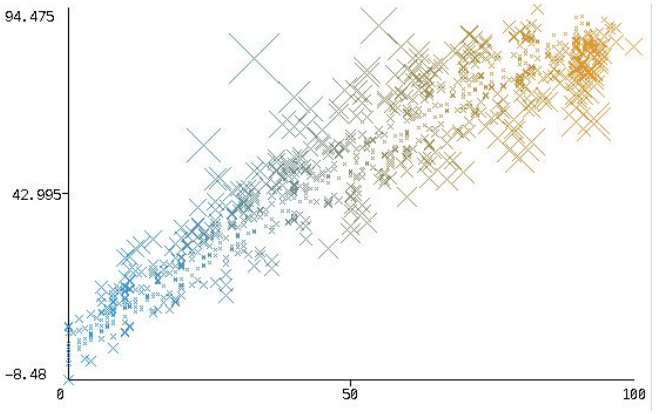


- **M5P:**

```
Time taken to build model: 0.1 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9498
Mean absolute error                      6.7324
Root mean squared error                  8.7858
Relative absolute error                 27.6234 %
Root relative squared error             31.3925 %
Total Number of Instances              900
```

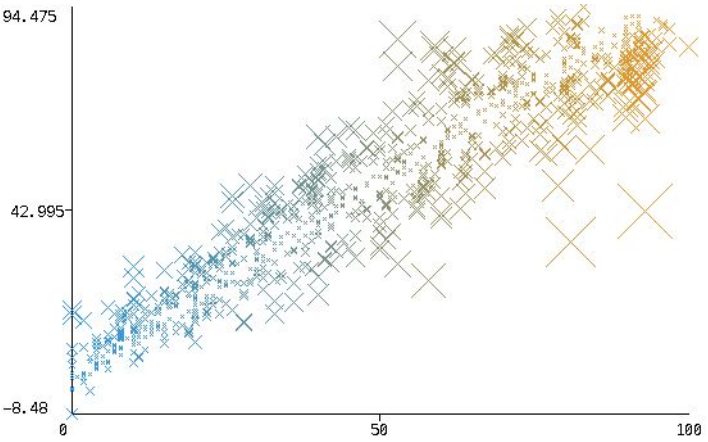


- **M5Rules:**

```
Time taken to build model: 0.53 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9486
Mean absolute error                      6.7916
Root mean squared error                  8.87
Relative absolute error                 27.8665 %
Root relative squared error             31.6933 %
Total Number of Instances              900
```

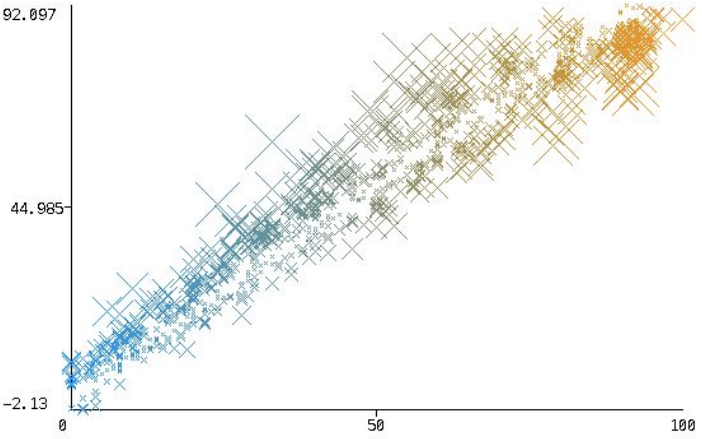


- **Vote:**

```
Time taken to build model: 2.84 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.9718
Mean absolute error                      5.433
Root mean squared error                  6.8486
Relative absolute error                 22.2917 %
Root relative squared error             24.4707 %
Total Number of Instances              900
```



Klasyfikator	**Współczynnik korelacji**	**Czas potrzebny na zbudowanie modelu [s]**
Vote	0.9718	2.84
RandomForest	0.9686	0.38
M5P	0.9498	0.1
M5Rules	0.9486	0.53
MultilayerPerceptron	0.9483	1.8

IV. Czwarty eksperyment.

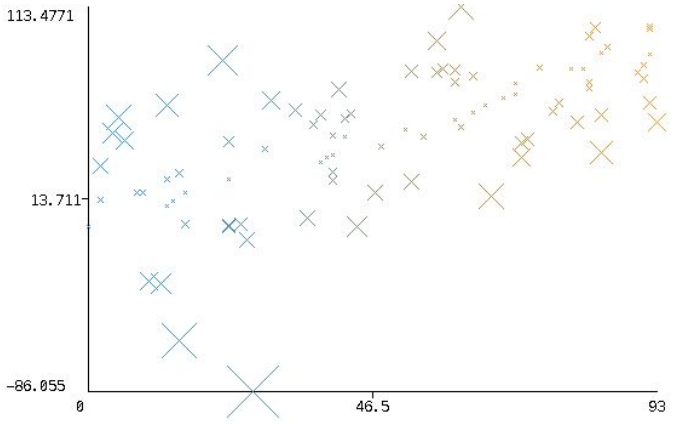
Eksperyment przeprowadzono na 90 danych treningowych. W tym eksperymencie użyto 10 folds cross-validation.

- **MultilayerPerceptron:**

```
Time taken to build model: 0.18 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.6554
Mean absolute error                     20.4807
Root mean squared error                 27.6873
Relative absolute error                 82.7325 %
Root relative squared error             98.0356 %
Total Number of Instances               90
```

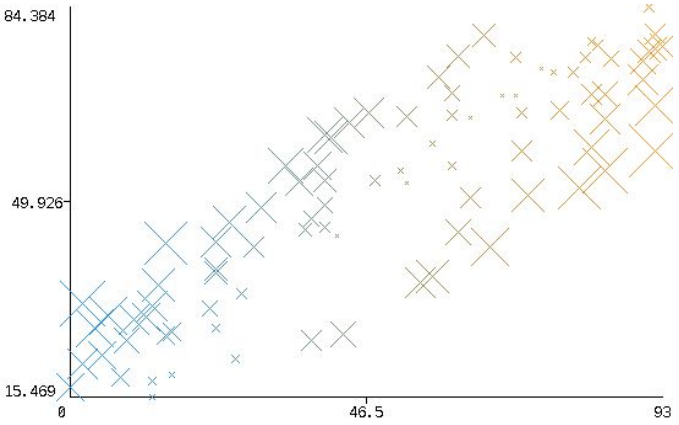


- **RandomForest:**

```
Time taken to build model: 0.06 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8612
Mean absolute error                     13.2287
Root mean squared error                 15.3511
Relative absolute error                 53.4377 %
Root relative squared error             54.3553 %
Total Number of Instances               90
```

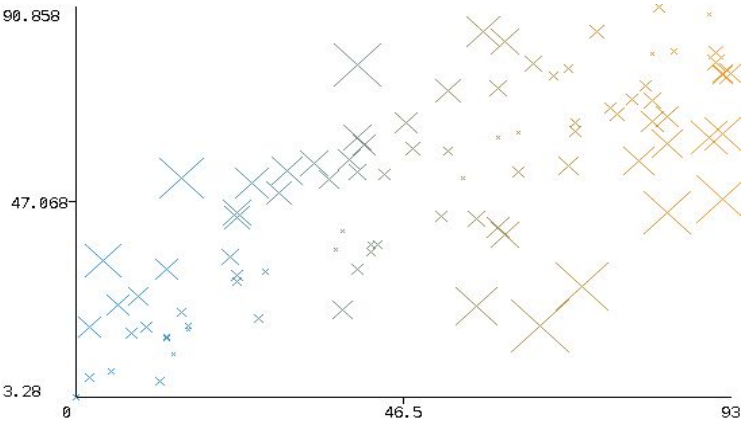


- **M5P:**

```
Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.7729
Mean absolute error                     14.2706
Root mean squared error                 17.8278
Relative absolute error                 57.6467 %
Root relative squared error             63.1249 %
Total Number of Instances               90
```

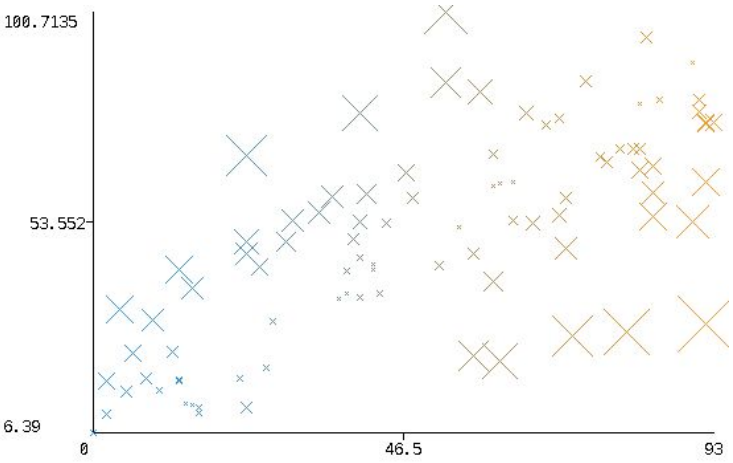


- **M5Rules:**

```
Time taken to build model: 0.02 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.7137
Mean absolute error                     15.3114
Root mean squared error                 19.8941
Relative absolute error                 61.8509 %
Root relative squared error             70.4416 %
Total Number of Instances               90
```

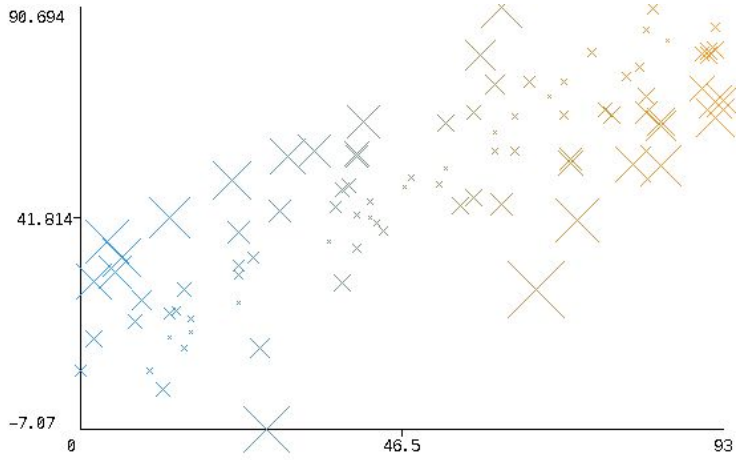


- **Vote**

```
Time taken to build model: 0.29 seconds

=== Cross-validation ===
=== Summary ===

Correlation coefficient                  0.8355
Mean absolute error                     12.1424
Root mean squared error                 15.4308
Relative absolute error                 49.0497 %
Root relative squared error             54.6376 %
Total Number of Instances               90
```



Klasyfikator	**Współczynnik korelacji**	**Czas potrzebny na zbudowanie modelu [s]**
RandomForest	0.8612	0.06
Vote	0.8355	0.29
M5P	0.7729	0.02
M5Rules	0.7137	0.02
MultilayerPerceptron	0.6554	0.18

6.3 Wnioski

Do naszego projektu wybraliśmy te klasyfikatory, które najlepiej dopasowują się do naszych danych, czyli mają najlepszy współczynnik korelacji. Są to: RandomFroest, M5P, MultilayerPerceptron oraz Vote, który składa się z wymienionych wcześniej klasyfikatorów.

Po przetestowaniu klasyfikatorów na różnych zbiorach danych zobaczyliśmy, że im więcej jest danych treningowych, tym lepszy osiągamy współczynnik korelacji. Dlatego lepiej stosować duże zbiory danych do trenowania. Jednak wiąże się to ze zwiększonym czasem na budowanie modeli. W naszym projekcie nie jest to problemem, ponieważ zastosowaliśmy system agentowy i uczenie każdego klasyfikatora jest wykonywane w osobnym agencie.

7. Podsumowanie

System został zaimplementowany zgodnie z naszymi założeniami. Wykorzystanie modelu agentowego ma wiele zalet. Przede wszystkim każdy agent jest uruchamiany na osobnym wątku, co powoduje zwiększenie wydajności systemu. Również WEKA okazała się być bardzo prostą w użyciu biblioteką, dostarczającą narzędzia uczenia maszynowego oraz wygodnie API.

7.1 Dalszy kierunek rozwoju

W przyszłości system można rozszerzyć o nowe pierwiastki, co umożliwi wprowadzanie bardziej różnorodnych stopów. Zwiększyć można również ilość dostępnych w programie klasyfikatorów, a także umożliwić użytkownikowi specyfikowanie parametrów gdy jest to możliwe, takich jak na przykład ilość warstw sieci neuronowej. Uściślić należałoby wzory wykorzystywane między innymi do obliczania wsadu i kosztów, gdyż zostały one przez nas w dużym stopniu uproszczone ze względu na brak wiedzy w danej dziedzinie.

8. Bibliografia

- [1] Data mining software in Java - https://www.cs.waikato.ac.nz/ml/weka/
- [2] Java Agent Development Framework - http://jade.tilab.com
- [3] Projekty studentów z poprzednich lat - http://jagular.iisg.agh.edu.pl/~kozlak/Praktyka2018/Przedsiebiorstwo.zip
- [4] JavaFX Overview -
- https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-overview.htm
- [5] Klasyfikatory - http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html
- [6] Stopy aluminium - https://pl.wikipedia.org/wiki/Stopy_aluminium
- [7] Stopy magnezu - https://pl.wikipedia.org/wiki/Stopy_magnezu
- [8] Stopy miedzi - https://pl.wikipedia.org/wiki/Stopy_miedzi
- [9] Stopy niklu - https://pl.wikipedia.org/wiki/Stopy_niklu
- [10] Stopy ołowiu - https://pl.wikipedia.org/wiki/Stopy_ołowiu
- [11] Temperatury topnienia metali - http://www.alfa-tech.com.pl/temperatury
- [12] Jadwiga Skawa: “Przydatność analizy termicznej do oceny procesu krystalizacji” - http://www.afe.polsl.pl/index.php/pl/2969/przydatnosc-analizy-termicznej-do-oceny-procesu-krystalizacji-siluminow-modyfikowanych.pdf
- [13] Ceny metali - https://www.money.pl/gielda/surowce/
- [14] Wzory błędów - https://stats.stackexchange.com/questions/131267/how-to-interpret-error-measures

9. Manual

Wszystkie pliki źródłowe zawierają się w pakiecie agh.
Pakiet ten zawiera 5 pakietów oraz klasę główną programu Main.
W pakiecie agents znajdują się wszystkie klasy odpowiedzialne za model agentowy.
W pakiecie calculation znajduje się klasa Calculator odpowiedzialna za obliczenie wsadu i kosztu.
W pakiecie classification znajdują się klasy odpowiedzialne za uczenie maszynowe.
W pakiecie controllers znajdują się klasy kontrolerów odpowiedzialne za GUI.
W pakiecie generator znajduje się generator danych.

Pliki .fxml opisujące GUI znajdują się w folderze main/resources/fxml
Wszystkie pliki .java znajdują się w folderze /main/java
Plik z danymi treningowymi znajduje się w folderze głównym.

Program w momencie pojedynczej predykcji tworzy plik Predicition.arff w którym zapisuje wprowadzone parametry z nieznaną jakością oraz plik Predicted.arff uzupełniony o jakość. Podczas klasyfikacji pliku z danymi z nieznaną jakością, program uzupełnia dane o jakość i zapisuje do nowego pliku z dopiskiem “classified”. Dostępne stopy przechowywane są w pliku “Stops.txt”, w przypadku jego braku tworzony jest nowy z domyślnymi stopami.

Program nie wymaga instalacji. Do uruchomienia wymagana jest zainstalowana Java 8. Jest to projekt typu Maven, a więc wszystkie niezbędne do działania programu biblioteki zapisane są w pliku pom.xml i automatycznie pobierane podczas importowania. Program jest dostarczony z plikiem TrainingData.arff, który jest wymagany do działania programu.

Link do repozytorium:
https://github.com/tomsoonn/IndustryOptimizer