Meta machine learning on the top of OpenML

Projekt zrealizowany w ramach przedmiotu Hurtownie danych

Autorzy:

Imię	Nazwisko	Numer indeksu
Maciej	Hybiński	129618
Piotr	Konowalski	109727
Marcin	Zabłocki	109690

Przedmiot:

Nazwa	Kierunek	Stopień	Semestr	Tryb studiów	Data
Hurtownie danych	Informatyka	2	1	Stacjonarne	13.06.2016

Prowadzący: dr hab. inż. Robert Wrembel, prof. nadzw.
Opiekun projektu: Besim Bilalli

POLITECHNIKA POZNAŃSKA

1. Wstęp

1.1. Ogólna idea projektu

Uczenie maszynowe jest obecnie często stosowaną techniką w projektach informatycznych, w których przetwarzane są zbiory danych. Można je wykorzystać zarówno do automatycznego klasyfikowania (np. w diagnozie chorób), budowie sztucznej inteligencji (np. do sterowania autonomicznymi pojazdami), czy też do przewidywania trendów. Różnorodność algorytmów, które mogą być wykorzystane w tej dziedzinie, powoduje, że dobór odpowiedniego algorytmu, który w konkretnym zastosowaniu da najlepszy wynik jest bardzo trudne i wymaga specjalistycznej wiedzy. Istotny jest nie tylko rodzaj danych wejściowych, ale też cel, który ma zostać osiągnięty. Często, by wybrać odpowiedni algorytm, wykonywane są eksperymenty np. na próbkach danych, jednak wykonywanie takich testów manualnie, może być bardzo czasochłonne. W związku z tym, pojawia się pomysł, aby wykorzystać uczenie maszynowe do rozwiązania tego problemu. Taki proces, można nazwać meta-uczeniem maszynowym, ponieważ wykorzystuje on dane na temat danych jako dane wejściowe.

1.2. Wymagania funkcjonalne

Problemem, który ma rozwiązać aplikacja stworzona w ramach projektu jest dobór algorytmu uczenia maszynowego dla zbiorów danych wprowadzonych przez użytkownika. Danymi wejściowymi będą:

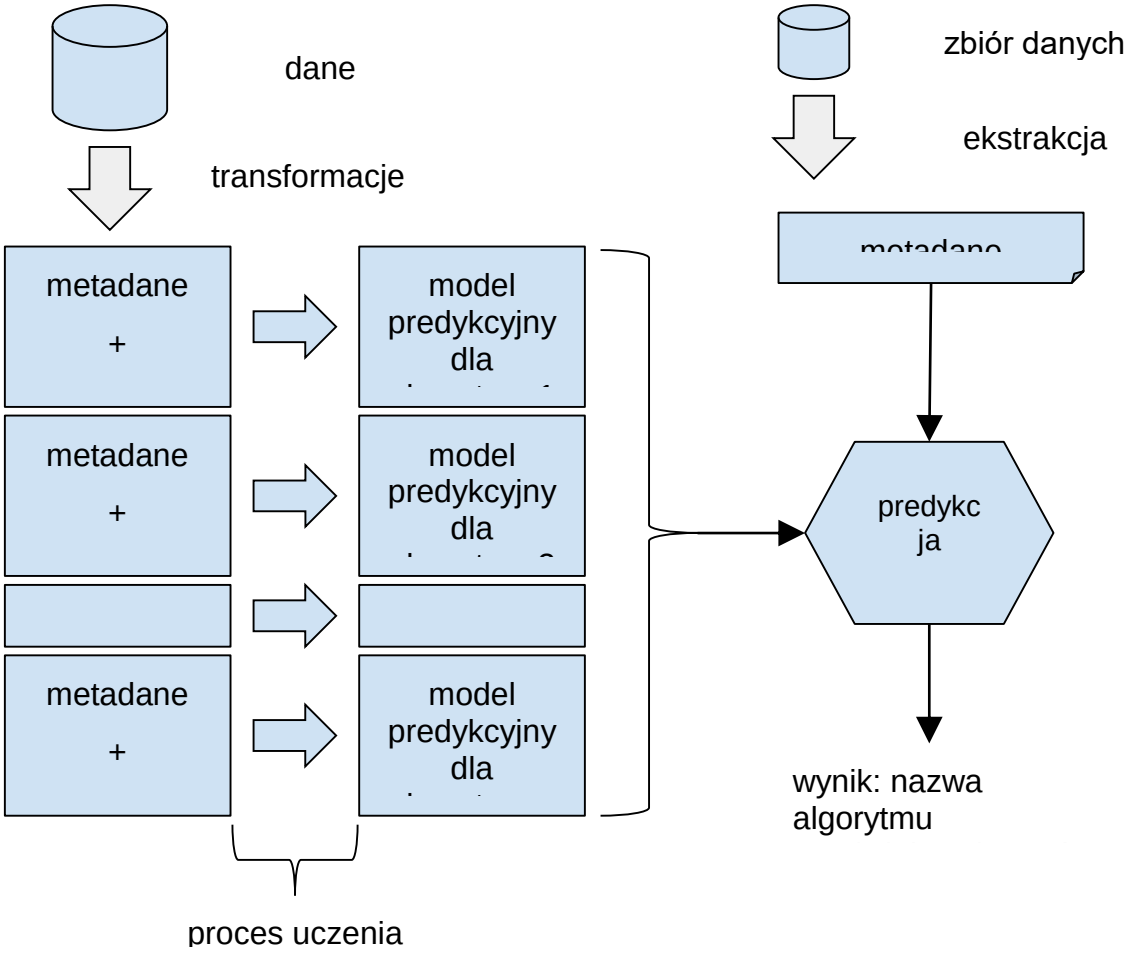
- dane na temat uruchomień algorytmów uczenia maszynowego na wielu zbiorach danych oraz atrybuty charakteryzujące zbiory danych (metadane),
- zbiór danych użytkownika.

Wyjściem będzie nazwa algorytmu uczenia maszynowego, który osiągnie najwyższą przewidywaną jakość.

Poza główną funkcjonalnością, aplikacja ma umożliwiać operacje na danych wejściowych, w celu eksportowania i filtrowania metadanych oraz w celu ich wykorzystania poza aplikacją. Ponadto, aplikacja ma również pozwolić na uruchamianie algorytmów uczenia maszynowego, w celu weryfikacji poprawności przewidywań.

1.3. Koncepcja projektu

Poniższy schemat przedstawia wysokopoziomową koncepcję projektu. Z głównego zbioru danych zostają wydzielone podzbiory zawierające metadane oraz wyniki poszczególnych algorytmów (dla każdego z algorytmów osobno). Na ich podstawie budowane są modele predykcji. Z drugiej strony, z danych użytkownika są wyciągane metadane, które będą stanowiły przypadek testowy. Każdy z modeli zostaje uruchomiony, w rezultacie czego otrzymujemy przewidywaną ocenę jakości każdego z algorytmów na danych użytkownika. Ten, który otrzymał najwyższą ocenę, zostaje wskazany jako najlepszy.



2. Źródło danych

2.1. OpenML

Aby uczenie maszynowe dało oczekiwane rezultaty, wymagane było pozyskanie odpowiednich zbiorów danych wejściowych. Idealnym źródłem okazała się baza danych projektu OpenML. Projekt ten jest otwartą inicjatywą zrzeszającą ludzi związanych z uczeniem maszynowym. W swojej bazie danych, OpenML składuje dane liczbowe na temat działania algorytmów na różnych zbiorach danych (dostępnych na stronie) - dane te generują użytkownicy przez uruchamianie tych algorytmów na swoich komputerach, przy pomocy specjalistycznego oprogramowania, m.in Weka. Cechami charakteryzującymi uruchomienie algorytmu są między innymi (oryginalne nazwy angielskie): predictive accuracy, precision, recall. W dalszej części będą one ogólnie nazywane miarami.

Każdy z dostępnych w bazie OpenML zbiorów danych jest również opisany zestawem atrybutów opisujących sam zbiór - są to właśnie metadane. Należą do nich między innymi: number of instances, number of classes, class entropy, number of numerical attributes i inne.

Baza danych jest udostępniona w wersji pełnej i częściowej pod adresem:

http://www.openml.org/downloads/. Plik zawiera skrypt języka SQL tworzący odpowiednią strukturę tabel, jak i wstawiający do nich dane. Baza danych OpenML działa na DBMS MySQL, więc ten właśnie system został wykorzystany w projekcie.

2.2. Struktura bazy danych

O ile zawartość bazy OpenML okazała się bardzo wartościowa w kontekście projektu, o tyle praca z nią wymagała początkowo długotrwałego procesu reverse engineering, ze względu na brak pełnej dokumentacji struktury relacji, a ta zawierała 51 tabel. Plik SQL, który tworzył bazę nie zawierał poleceń tworzenia ograniczeń integralnościowych w postaci kluczy obcych, ponadto nazwy niektórych kolumn, które były kolumnami połączeniowymi miały całkowicie inne nazwy w różnych tabelach. Na podstawie analizy zawartości tabel jak i sposobu prezentacji danych bezpośrednio na stronie OpenML.org udało się jednak odkryć strukturę relacji, tak aby móc prawidłowo budować zapytania o konkretne dane.

2.3. Użycie bazy danych

Aby móc elastycznie wykonywać zapytania agregujące na bazie danych OpenML, zostały przygotowane widoki, łączące tabele opisujące zbiory danych z tymi opisującymi dane na temat uruchomień algorytmów. Jednym z problemów na tym etapie okazał się sposób reprezentacji danych tabelach. Meta atrybuty opisujące zbiory danych, były przechowywane w bazie w formie wierszowej, tak jak poniższy przykład:

Data	Quality	Value
1	NumberOfMissingValues	2
2	ClassCount	6
3	ClassEntropy	-1.0
…	…	…

Wymaganą reprezentacją, nadającą się do przekazania do algorytmu uczenia maszynowego była natomiast następująca forma:

Data	NumberOfMissingValues	ClassCount	ClassEntropy
1	2	6	NULL
…	…	…	

Dla osiągnięcia takiego efektu, została wygenerowana najpierw lista unikalnych nazw meta atrybutów (quality), a następnie przekształcona ona została na widok zawierający podzapytania w klauzuli select, tworząc w ten sposób perspektywę o rządanej formie reprezentacji. Fragment kodu tworzącego widok przedstawia się następująco:

```
(SELECT
    `ds`.`did` AS `did`,
    (SELECT
            `idq`.`value`
        FROM
            `data_quality` `idq`
        WHERE
            ((`idq`.`data` = `ds`.`did`)
                AND (`idq`.`quality` = 'ClassCount'))
        LIMIT 0 , 1) AS `ClassCount`,
    (SELECT
            `idq`.`value`
        FROM
            `data_quality` `idq`
        WHERE
            ((`idq`.`data` = `ds`.`did`)
                AND (`idq`.`quality` = 'ClassEntropy'))
        LIMIT 0 , 1) AS `ClassEntropy`,
```

2.4. Użycie bazy danych

Zapytanie to odwzorowuje operację pivot wykonaną na tabeli z meta atrybutami - transformuje strukturę wierszową na kolumnową. Widok w końcowej wersji posiadał 101 kolumn z meta atrybutami i 4 kolumny dodatkowe (m.in z identyfikatorem zbioru danych).

Kolejnym problemem na etapie tworzenia widoków okazała się wydajność. Skrypt tworzący bazę danych nie zawierał poleceń tworzących indeksy. Zadowalającą szybkość działania zapytań użytych do tworzenia perspektyw osiągnięto przez dodanie indeksów na wszystkich kolumnach użytych w obrębie zapytania. Dalsza diagnoza problemów z wydajnością nie była wymagana.

Każdy algorytm w bazie danych jest opisany za pomocą trzech kolumn: klasy, wersji i parametrów. Wymaganiem projektu była maksymalna elastyczność w kwestii wyciągania danych z bazy, stąd powstały trzy dodatkowe perspektywy, pozwalające grupować miary algorytmów według każdej z podanych wyżej kolumn. W trakcie grupowania, funkcją agregującą miarę algorytmów była mediana. W uwagi na utrudnioną implementację takiej agregacji wewnątrz zapytania SQL (w dialekcie MySQL), obliczenia mediany i grupowania dokonano po stronie aplikacji. W ramach projektu, ograniczono zakres wyciąganych miar algorytmów do tych, opisujących eksperymenty wykonane za pomocą metody 10-fold cross validation zgodnie z wymaganiami prowadzącego projekt.

Główne perspektywy prezentowały się następująco:

Dzięki takiej strukturze, zapytanie wyszukujące wszystkie metadane, ze wszystkich dostępnych zbiorów danych, wraz z miarą dla wybranego algorytmu zostało bardzo uproszczone i wyglądało następująco:

```
SELECT amd.*, meta.*  FROM algorithm_measures_on_datasets amd
        join dataset_metadata meta on amd.did = meta.did
where amd.measure = 'predictive_accuracy'
    and algorithm_class = 'weka.NaiveBayes'
```

Jego wynikiem była tabela o następującej strukturze:

did	name	ClassCount	ClassEntropy	Dimensionality	NumBinaryAtts	NumNominalAtts
1	anneal	6.0	-1.0	0.043429844097995544	14.0	32.0
3	kr-vs-kp	2.0	0.9985755...	0.011576971214017523	34.0	36.0
4	labor	2.0	0.9348490...	0.2982456140350877	3.0	8.0
5	arrhythmia	16.0	-1.0	0.6194690265486725	73.0	73.0
6	letter	26.0	4.6998107...	8.5E-4	0.0	0.0
7	audiology	24.0	3.4219467...	0.30973451327433627	61.0	69.0

(pozostałe kolumny)

(pozostałe wiersze)

Poza głównymi perspektywami, zostało utworzonych jeszcze kilka mniejszych, które pozwoliły na łatwe wyświetlanie dostępnych atrybutów i algorytmów z poziomu GUI aplikacji - były to widoki wyświetlające unikalne wartości wymaganych kolumn.

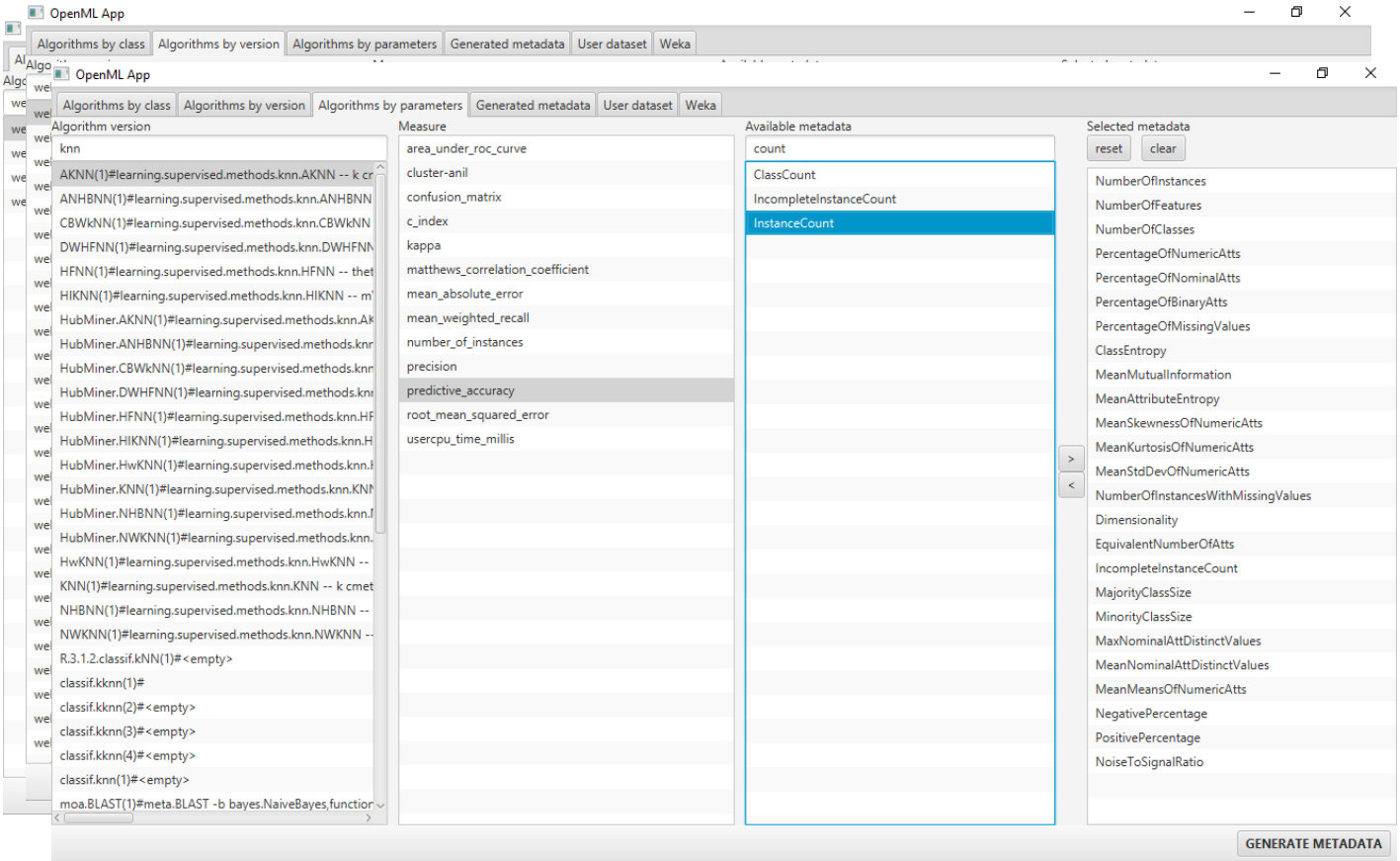
3. Moduły aplikacji

Zgodnie z założeniem aplikacja ma być przeznaczona zarówno dla ekspertów jak i użytkowników podstawowych. Rola eksperta sprowadza się do przygotowywania zbiorów metadanych, które następnie mogą zostać wykorzystane przez użytkownika w celu przeprowadzenia predykcji optymalnego algorytmu dla własnego zbioru danych.

3.1. Dla ekspreta

Część ekspercka jest najbardziej rozbudowana i to właśnie tam znajduje się większość modułów integracji z zewnętrznymi systemami, w tym w szczególności z bazą OpenML. Dla eksperta przeznaczone są 4 zakładki aplikacji:

- Algorithms by class
- Algorithms by version
- Algorithms by parameters
- Generated metadata



Pierwsze trzy zakładki służą do selekcji danych, będącymi później danymi uczącymi modele. Na każdej z tych zakładek ekspert może wybrać rodzaj algorytmu (algorithm class/version), miarę (measure) oraz metadane, które chce wykorzystać do utworzenia modelu. Różnica jednak polega na szczegółowości wyboru algorytmu.

Pierwsza z zakładek (Algorithms by class) pozwala na najbardziej ogólny wybór algorytmu. Dla danej klasy algorytmu (np. weka.NaiveBayes) dostępna jest tylko jedna wersja.

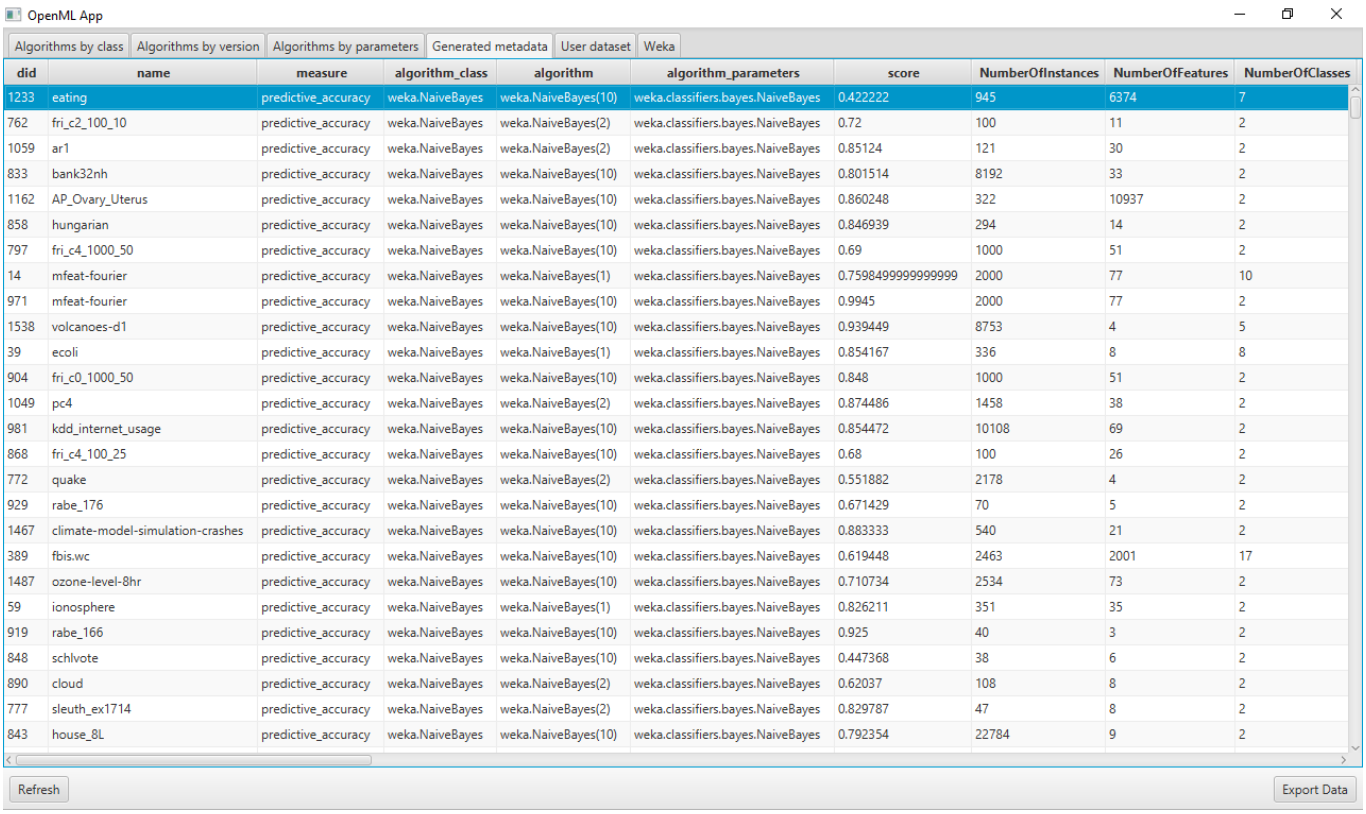
Druga zakładka (Algorithm by version) oferuje bardziej szczegółowy wybór. Dla danej klasy algorytmu (np. weka.NaiveBayes) dostępnych jest najczęściej kilka jego różnych wersji np. weka.NaiveBayes(1), weka.NaiveBayes(2), itd.

W obu przypadkach ze względu na pewien stopień generalizacji wyboru i konieczność grupowania wyników dla kilku różnych rodzajów parametryzacji uruchomienia algorytmu, dla wybranej miary (np. predictive_accuracy) obliczana, a następnie zwracana jest mediana. Ta cecha może oczywiście negatywnie wpływać na jakość otrzymanego rezultatu predykcji, dlatego też, dla najbardziej zaawansowanych użytkowników przeznaczona jest trzecia zakładka, która pozwala na wybór nie tylko wersji algorytmu, ale również konfiguracji parametrów.

Wybór określonych wartości w przypadku kolumn Algorithm oraz Measure dokonujemy poprzez zaznaczenie interesującej nas wartości. Dla kolumny Available metadata dostępna jest specjalna kontrolka, pozwalająca wybrać wiele wartości poprzez przerzucenie ich z lewej do prawej kolumny. Wartości z prawej kolumny zostaną wybrane do wygenerowowanego zbioru danych.

Kolumny Algorithm oraz Available metadata ze względu na bardzo rozległą dziedzinę dostępnych wartości umożliwiają wyszukiwanie poprzez wpisanie całości lub fragmentu interesującej nas nazwy w pole tekstowe znajdujące się nad daną kolumną.

W każdej ze wspomnianych zakładek, w prawym dolnym rogu znajduje się przycisk Generate metadata, który pobiera z bazy danych wybrane wcześniej wartości i przenosi je do zakładki Generated metadata.



did	name	measure	algorithm_class	algorithm	algorithm_parameters	score	NumberOfInstances	NumberOfFeatures	NumberOfClasses
1233	eating	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.422222	945	6374	7
762	fri_c2_100_10	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.72	100	11	2
1059	ar1	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.85124	121	30	2
833	bank32nh	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.801514	8192	33	2
1162	AP_Ovary_Uterus	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.860248	322	10937	2
858	hungarian	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.846939	294	14	2
797	fri_c4_1000_50	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.69	1000	51	2
14	mfeat-fourier	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(1)	weka.classifiers.bayes.NaiveBayes	0.7598499999999999	2000	77	10
971	mfeat-fourier	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.9945	2000	77	2
1538	volcanoes-d1	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.939449	8753	4	5
39	ecoli	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(1)	weka.classifiers.bayes.NaiveBayes	0.854167	336	8	8
904	fri_c0_1000_50	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.848	1000	51	2
1049	pc4	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.874486	1458	38	2
981	kdd_internet_usage	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.854472	10108	69	2
868	fri_c4_100_25	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.68	100	26	2
772	quake	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.551882	2178	4	2
929	rabe_176	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.671429	70	5	2
1467	climate-model-simulation-crashes	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.883333	540	21	2
389	fbis.wc	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.619448	2463	2001	17
1487	ozone-level-8hr	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.710734	2534	73	2
59	ionosphere	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(1)	weka.classifiers.bayes.NaiveBayes	0.826211	351	35	2
919	rabe_166	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.925	40	3	2
848	schlvote	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.447368	38	6	2
890	cloud	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.62037	108	8	2
777	sleuth_ex1714	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(2)	weka.classifiers.bayes.NaiveBayes	0.829787	47	8	2
843	house_8L	predictive_accuracy	weka.NaiveBayes	weka.NaiveBayes(10)	weka.classifiers.bayes.NaiveBayes	0.792354	22784	9	2

Tam wyselekcjonowane dane trafiają do interaktywnej tabelki, dzięki której użytkownik może je z łatwością przeglądać, automatycznie sortować oraz zmieniać kolejność kolumn.

Jeśli osiągnięty rezultat przetwarzania nie spełnia założonych oczekiwań proces można powtórzyć powracając do jednej z trzech zakładek służących do selekcji danych.

W przeciwnym przypadku dane można wyeksportować do pliku *.csv po kliknięciu przycisku Export Data znajdującego się w prawym dolnym rogu.

Oprócz podstawowych funkcjonalności dostępnych poprzez interfejs graficzny aplikacji, dostępne są również pliki konfiguracyjne pozwalające zdefiniować listy atrybutów, które mają być zawsze eksportowane oraz takich, które mają być domyślnie wybrane.

3.2. Dla użytkownika

Aby dokonywać predykcji na danych użytkownika, niezbędnym było wyciągnięcie ze zbioru danych użytkownika metadanych, odpowiadającym tym ze zbiorów uczących. W tym celu dokonano integracji z biblioteką Fantail, która wchodzi w skład kodu źródłowego samej aplikacji webowej OpenML.org i jest dostępna pod adresem: https://github.com/openml/OpenML/tree/master/metafeatures.

Integracja polegała na wydzieleniu tego modułu z kodu źródłowego OpenML, a następnie przeniesieniu go do aplikacji wraz z wymaganymi zależnościami. Tak samo, jak w przypadku wybierania danych do eksportu w zakładach dla eksperta, tak i tutaj dodana została możliwość wyboru tylko niektórych atrybutów. Biblioteka Fantail pozwala na ekstrakcję 54 typów meta atrybutów, jednak nazwy niektórych z nich nie były spójne z nazwami atrybutów z bazy danych OpenML, np. NumberOfInstances wewnątrz Fantail, w bazie był nazwany InstanceCount. Aby rozwiązać ten problem, zaprojektowany został system aliasów, konfigurowalny za pomocą pliku XML. Moduł Fantail rozszerzono w ten sposób, aby przed zwróceniem metadanych, przetransformował ich nazwy zgodnie ze skonfigurowanymi aliasami. Dzięki temu zabiegowi, metadane wyciągnięte ze zbioru użytkownika mogły zostać z łatwością użyte jako dane wejściowe do modeli predykcji, zbudowanych w aplikacji. Struktura pliku XML przedstawia się następująco:

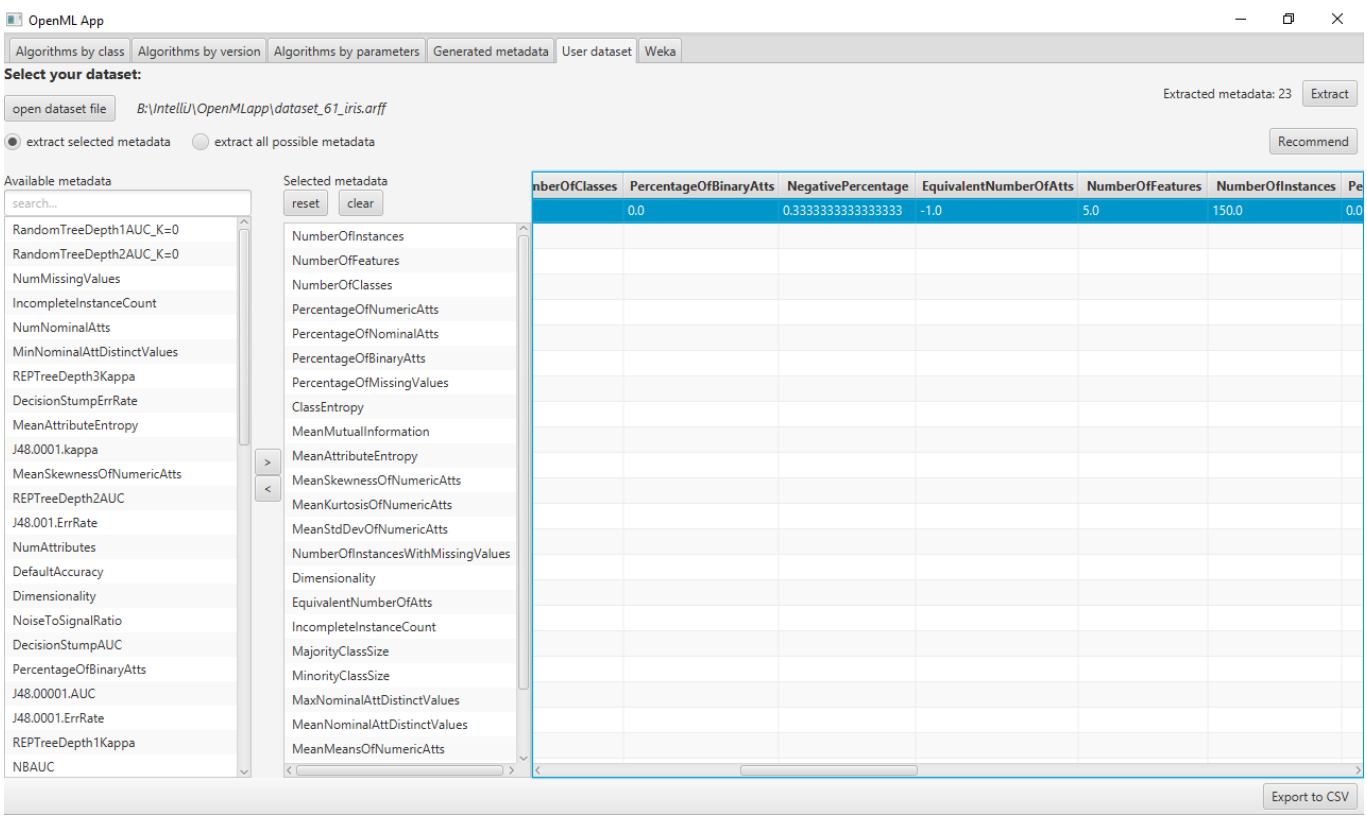
```
<?xml version="1.0" encoding="utf-8" ?>
<metattributes>
    <aliases>
        <for>NumberOfInstances</for>
        <alias>InstanceCount</alias>
    </aliases>
    <aliases>
        <for>NumberOfFeatures</for>
        <alias>NumAttributes</alias>
    </aliases>
</metattributes>
```

Element for wskazuje na nazwę używaną przez bibliotekę Fantail, natomiast element alias wskazuje na nazwę zgodną z nazewnictwem OpenML.

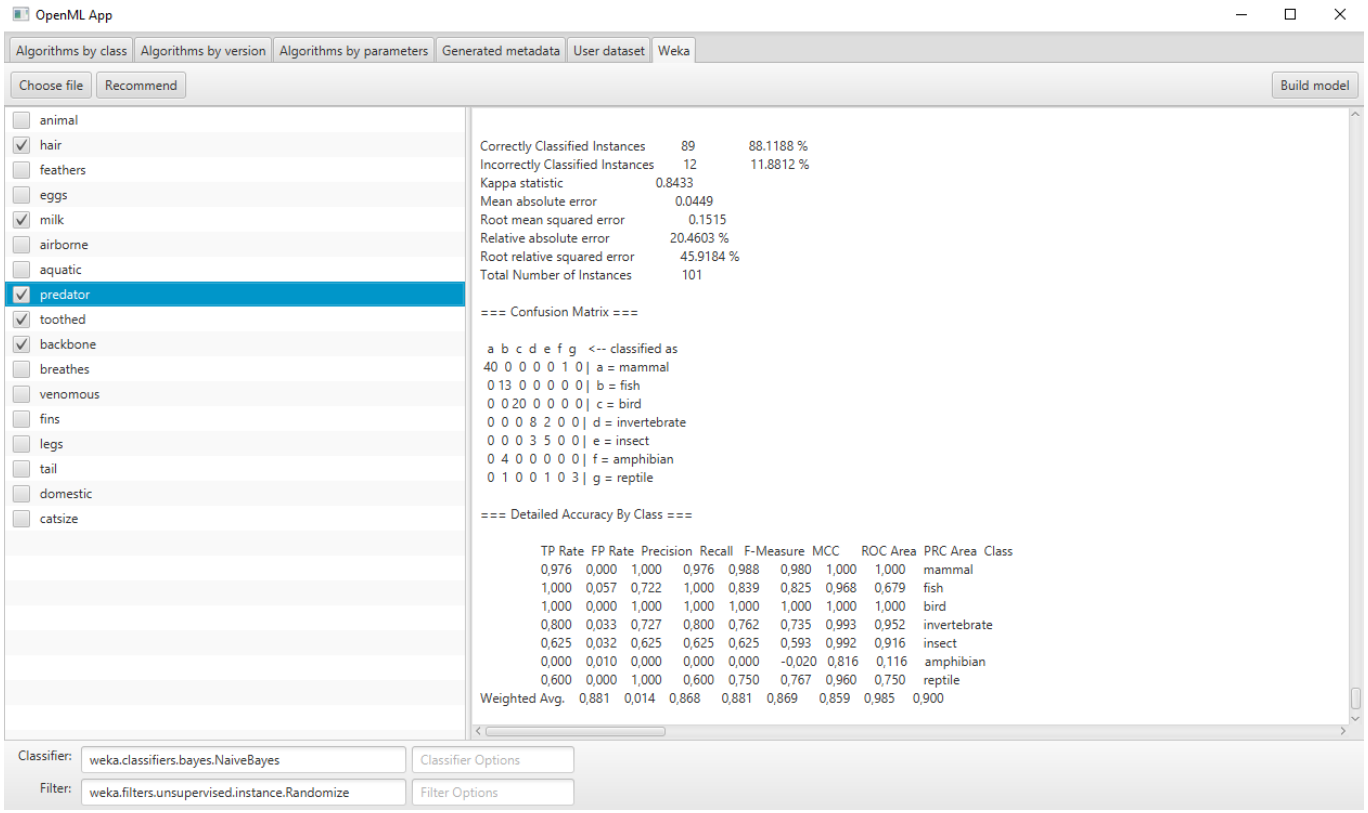
By móc zarekomendować użytkownikowi algorytm, konieczne jest wykonanie kilku kroków. Po za posiadaniem zbioru danych, na którym chce operować, konieczne jest posiadanie zbiorów treningowych tzn. plików .csv wygenerowanych z bazy z użyciem tej aplikacji. By predykcja była wiarygodna, muszę one posiadać takie same meta-atrybuty, a powinny się różnić wersja

algorytmu, dla którego eksportujemy dane z bazy. Wspomniane pliku powinny zostać zapisane do wspólnego folderu. Po przejściu do zakładki User dataset konieczne jest wskazanie pliku, dla którego chcemy dokonać predykcji. Następnie użytkownik wskazuje ścieżkę do folderu, w którym znajdują się pliku, które posłużą za zbiory treningowe. Konieczne jest również wyciągniecie meta-danych dla wybranego pliku, które posłużą za zbiór testowy.

Po wykonaniu powyższych kroków użytkownik może skorzystać z przycisku rekomendacji. Reakcja na to jest uruchomienie budowania modeli dla wszystkich plików .csv zawartych we wskazanym folderze. Są one budowane z użyciem algorytmu M5Prime, który pozwala dokonać predykcji atrybutów numerycznych. Po zbudowaniu modelu dla każdego zbioru danych, dokonywana jest ewaluacja z wykorzystaniem zbioru testowego, w rezultacie czego otrzymywany jest wynik określający dokładność predykcji. Po sprawdzeniu wszystkich wyników, użytkownik jest informowany o tym, jaki algorytm zadziałać najbardziej efektywnie na wskazanym zbiorze danych.



Użytkownik może sprawdzić, czy faktycznie predykcja jest prawidłowa wykorzystując zakładkę Weka. Wskazując zbiór danych, dla konkretnych atrybutów, można sprawdzić, jak konkretny algorytm działa dla podanego pliku. Dodano możliwość parametryzacji i filtrowania, co daje użytkownikowi pełną swobodę w testowaniu działania wszystkich algorytmów dostępnych w Wece.



4. Funkcjonalność dodatkowa

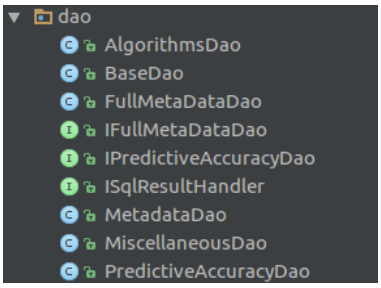
4.1. Przenoszenie

Projekt jest odpowiednio dostosowany, by w łatwy i bezproblemowy sposób móc przenieść projekt na inną platformę. Najprostszym sposobem edycji jego zawartości jest wczytanie go poprzez Intellij IDEA. Można wykorzystać również inne zintegrowane środowisko programistyczne, ale w takim wypadku konieczna jest instalacja oprogramowania Apache Maven. Zostało ono wykorzystane, gdyż pozwala automatycznie pobierać niezbędne biblioteki bez interakcji użytkownika. Zawarte w pliku pom.xml zależności zostają zaimportowane i dołączone do projektu podczas jego importowania.

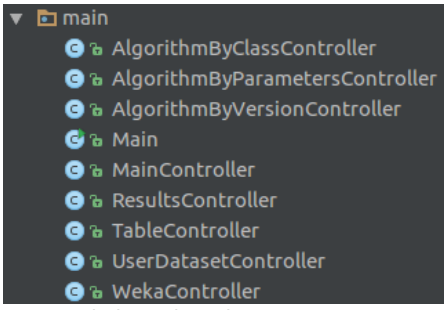
By móc korzystasz z projektu, konieczne jest posiadania bazy danych OpenML. Odpowiedni skrypt tworzący ją można znaleźć na stronie organizacji. Dostarczone do projektu skrypty tworzące widoku i zakładające indeksy na odpowiednich tabelach umożliwiają proste dostosowanie bazy do wymagań aplikacji. W docelowym folderze z projektem zawarte są także pliki, które pozwalają na modyfikację listy atrybutów i meta-atrybutów, które mają zostać domyślnie wybrane przy eksporcie danych oraz na określenie połączenia z bazą danych. Taka praktyka pozwala użytkownikowi zaoszczędzić czas, gdyż proces adaptacji aplikacji na nowej platformie wygania jedyne dokonywania modyfikacji zawartości odpowiednich plików tekstowych i nie wymagana jest na tym etapie żadna interakcja z kodem źródłowym.

4.2. Rozszerzalność

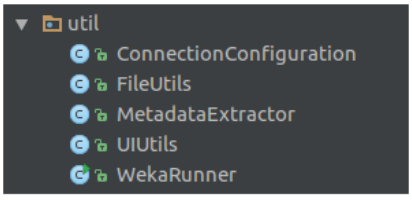
Aplikacja została stworzona w oparciu o bibliotekę JavaFX, która pozwala w prosty sposób tworzyć graficzne interfejsy użytkownika. Tworząc projekt zastosowany został wzorzec Model-View-Controller. Wykorzystana została niezależność modeli od widoków, co dało możliwość tworzenia wielu widoków reprezentujących dane na różne sposoby. Zawartość projektu została podzielona na odpowiednie sekcje (zaprezentowane poniżej), pozwalające łatwo zorientować się w strukturze aplikacji i pozwala to na zorientowania się w sposobie działania programu przy pierwszym kontakcie z kodem źródłowym.



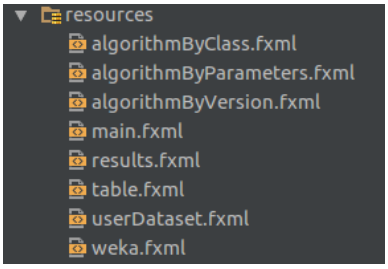
Dostęp do bazy danych



Dostęp do bazy danych



Intuicyjne korzystanie z API



Widoki JavaFX