Poznań, 7 kwietnia 2014

Informatyka w medycynie

Klasyfikacja publikacji naukowych

***Punkt kontrolny I**: Proponowana koncepcja rozwiązania problemu*

Jakub Matjanowski 106050, Jan Staniewicz 106612
Poniedziałek 8:00

1 Wstęp

W ramach projektu zaliczeniowego z przedmiotu *Informatyka w medycynie* postanowiliśmy rozwiązać problem zadany podczas konkursu: JRS 2012 Data Mining Competition. Zadanie polegało na zbudowaniu klasyfikatora przyporządkowującemu artykuły naukowe klasom decyzyjnym. Klasyfikator ten będzie uczony zbiorem artykułów z przypisanymi już klasami (zbiór uczący), a następnie nauczony klasyfikator będzie przydzielał atrybuty decyzyjne nowym artykułom o nieznanych atrybutach decyzyjnych (zbiór testowy). Każdy artykuł może być przydzielony do wielu klasy decyzyjnych. Głównym celem jest uzyskanie jak największej skuteczności na zbiorze testowym.

Zarówno zbiór uczący jak i testowy mają 23912 atrybuty warunkowe typu liczbowego – ilorazowego oraz 83 atrybuty decyzyjne typu symbolicznego - nominalnego (decyzja – tak/nie – należy lub nie do danej klasy decyzyjnej).

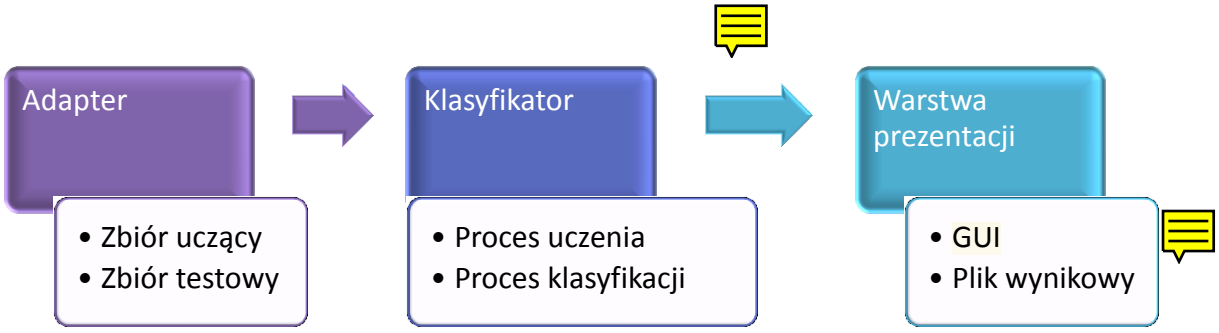
Atrybuty warunkowe są wartościami liczbowymi od 0 do 1000 wskazującymi na stopień powiązania danego artykułu z daną kategorią tematyczną (0-brak powiązania, 1000 – idealne powiązanie).

2 Proponowana architektura systemu

Zważywszy na fakt, że zarówno dane wejściowe jak i sposób reprezentacji danych wynikowych, nie koniecznie będzie zawsze taki sam postanowiliśmy system podzielić na 3 główne części:

1) adapter danych wejściowych
2) klasyfikator
3) warstwa prezentacji danych wyjściowych

Jak widać realną funkcję logiczną będzie pełniła część druga, tj. klasyfikator. Przyjmie on w określonym formacie zbiór danych uczących, a następnie zbiór danych testowych, przetworzy je, a w dalszej kolejności, przekaże do części trzeciej dane wynikowe (również w określonym wcześniej formacie).



2.1 Adapter

Moduł adaptera będzie odpowiadał za wczytanie danych wejściowych i przedstawienie ich w jednolitym, znanym klasyfikatorowi, formacie danych wejściowych. Ponadto rozróżni on zbiór uczący od zbioru testowego i zapewni dostarczenie ich klasyfikatorowi w odpowiednim momencie cyklu przetwarzania.

Wydzielenie modułu adaptera jest o tyle ważne, że pozwala na potencjalną zmianę formatu danych wejściowych. Jest to naszym zdaniem elastyczne rozwiązanie, które być może przyniesie sporo korzyści przy ewentualnym rozwoju systemu o nowe typy danych wejściowych.

2.2 KLASYFIKATOR

Klasyfikator jest rdzeniem systemu. Moduł ten odpowiada zarówno za analizę danych uczący i budowanie pewnego modelu wnioskowania, jak i za wykorzystanie zebranych danych w fazie podejmowania decyzji dla zbioru testowego.

Wydzielenie tego modułu również jest ważne z perspektywy rozwoju aplikacji. Być może dla innych rodzajów problemów, stworzony przez nas klasyfikator będzie nieefektywny. Łatwe będzie wtedy zastosowanie bliźniaczego modułu o innych właściwościach logicznych.

2.3 WARSTWA PREZENTACJI

Warstwa prezentacji będzie odpowiadała za przedstawienie użytkownikowi systemu wyniku przetwarzania. W wersji podstawowej może być to generowanie pliku wynikowego lub proste przedstawienie wyników tabelarycznie w graficznym interfejsie użytkownika. W przypadku rozwijania aplikacji, warstwę prezentacji można rozszerzyć o generowanie pliku HTML lub nawet interaktywnych wykresów porównawczych.

3 Technologia

Całość aplikacji zostanie wykonana w .NET 4.5 w języku C#. Wybraliśmy tę technologię głównie z powodu dostępności bibliotek pomagających w obsłudze zadanych nam plików wejściowych, przede wszystkim dużej ilości dostępnych implementacji uczenia maszynowego, które możemy wykorzystać.

Co więcej przewidujemy, że może być konieczne użycie dodatkowej bazy danych do przechowywania danych wejściowych. Jeśli w fazie implementacji okaże się to koniecznie, użyjemy bazy danych SQL Server Express, aby w wygodny sposób uzyskać dostęp do danych.

3.1 DOSTĘP DO DANYCH

Pierwszą operacją w procesie przetwarzania będzie zawsze wczytanie danych wejściowych. Do tego celu użyjemy biblioteki ArffSharp[1] stworzonej na licencji MIT. Ułatwi ona zdecydowanie dostęp do danych w formacie ARFF i pozwoli w prosty sposób przekształcić dane do struktury wymaganej przez moduł klasyfikatora.

W ramach dostępu do danych warto też uwzględnić hipotetyczną konieczność zapisu danych do bazy danych. Jeśli w fazie implementacji uznamy, że nie jest rozsądne trzymanie danych wejściowej w pamięci operacyjnej, użyjemy lekkiej bazy danych SQL Server Express (ewentualnie jeszcze lżejszej, plikowej bazy SQLite). Odciąży to trochę pamięć operacyjną, ale stworzy problem konieczności analizowania danych porcjami. Do komunikacji z bazami użyjemy mechanizmu dostępu do baz danych ADO.NET, gdzie prawdopodobnie wykorzystamy Entity Framework, czyli narzędzie ORM firmy Microsoft.

3.2 KLASYFIKACJA

Oczywistym jest, że zastosowanie gotowej już implementacji jednej z metod uczenia maszynowego nie da zadowalających efektów. Z drugiej strony, napisanie od zera własnego klasyfikatora jest również złym pomysłem, bo zabierze to zbyt wiele czasu, a do tego bardzo trudno będzie osiągnąć dobrą jakość klasyfikacji. Postanowiliśmy więc użyć framework'u AForge.NET[2]. Jest to zbiór bibliotek, które implementują wiele rodzajów klasyfikatorów. Naszym celem jest wybranie kilku, które sprawdzą się najlepiej dla naszego problemu, zoptymalizować ich parametry i połączyć ich siły, aby wspólnie, zbudowały zadawalający dla naszego problemu, klasyfikator.

[1] https://bitbucket.org/iano/arffsharp

2 http://www.aforgenet.com/framework/

3.3 PREZENTACJA

Do prezentacji wyników wykorzystamy głownie technologię WinForms. W aplikacji okienkowej zaprezentujemy proste statystyki dotyczące wyników klasyfikacji. Oprócz tego wyniki wyeksportujemy w formacie tekstowym do pliku.

4 Logika i algorytm

Ponieważ ilość danych które należy przetrzymać w pamięci jest bardzo duża (4B * 25000 atrybutów * 10 000 wierszy = 1GB danych) zamierzamy przepisać plik wejściowy do bazy danych dla łatwiejszego dostępu do danych. Następnie wybierając z niej artykuły należące do kolejnych klas będziemy wyszukiwać istotnych atrybutów dla danej klasy. Jako istotne będziemy traktować te atrybuty, których odchylenie standardowe jest odpowiednio małe(próg, od którego będziemy akceptować atrybut zostanie dobrany doświadczalnie). Oczywiście w ten sposób możemy otrzymać różne atrybuty dla różnych klas. Próg postaramy się dobrać tak, aby dla każdej klasy otrzymać maksymalnie kilkaset kluczowych atrybutów. Dzięki temu klasyfikacja artykułów testowych będzie wielokrotnie szybsza.

Przykład:

Artykuły x(0.1,0.5,0.9),y(0.6, 0.52, 0.4),z(0.8, 0.47, 0.2) należą do klasy A. Możemy zauważyć, że powodem przypisania powyższych artykułów do klasy A nie są parametry ni 3, ponieważ mimo ich zupełnie różnych wartości wciąż są klasyfikowane wspólnie. Co innego atrybut drugi, którego rozbieżność jest niewielka. Zatem sprawdzając przynależność kolejnych artykułów do klasy A będziemy brali pod uwagę jedynie 2 parametr.

Dla otrzymanych atrybutów wyliczymy ich średnią arytmetyczną dla każdej klasy. Zostaną one podane na wejście klasyfikatorów.

Każdy artykuł zostanie poddany analizie przez kilka różnych klasyfikatorów. Aby przyporządkować artykuł do danej klasy konieczne będzie „zaliczenie" testu na większości z nich. W przypadkach granicznych sprawdzimy czy lepsze wyniki da odrzucanie artykułów czy zaklasyfikowanie ich do danej klasy.

5 Przewidywane problemy i zagrożenia

Głównym problemem może okazać się to, że nasz pomysł na logikę aplikacji okaże się w praktyce nieefektywny. Musimy więc liczyć się z tym, że w fazie implementacji należało będzie ponownie zaprojektować pomysł na klasyfikację. Modularna architektura systemu ułatwi jednak te zmiany. Jedyne co będziemy musieli zmienić to właśnie ten moduł. Jeśli kilka naszych autorskich pomysłów okaże się nieefektywnych, wtedy będziemy zmuszeni, aby pomoc w rozwiązaniu problemu zaczerpnąć z projektu Weka. Użyjemy wtedy portu[3] biblioteki Weka do framework'u .NET, aby w dużo łatwiejszy sposób zbudować potrzebny nam klasyfikator.

6 Możliwe rozszerzenia

Rozszerzenia nad którymi myśleliśmy obejmują każdy z modułów systemu. Rozszerzenie adaptera obejmowałoby oczywiście możliwość importu danych z innych formatów: XML, JSON, itp. Warstwę prezentacji moglibyśmy uzupełnić kiedyś o prezentacje wykresów i szczegółowych statystyk – zarówno w interfejsie użytkownika jak i w generowanych dokumentach (np. HTML). Ciekawym rozszerzeniem modułu klasyfikatora byłoby umożliwienie wyboru jednego z wielu klasyfikatorów, a także dostrajanie parametrów klasyfikatorów w GUI programu. Po zastosowaniu takich rozszerzeń aplikacja nie byłaby wtedy narzędziem do rozwiązywania konkretnego problemu, ale uniwersalnym programem dokonującym klasyfikacji według sparametryzowanych klasyfikatorów użytkownika.

[3] http://sourceforge.net/projects/wekadotnet/