Moduł HeartClass

Michał Ciszewski, Łukasz Dudek, Krystian Mucha

Akademia Górniczo - Hutnicza w Krakowie

Na przedmiot: Elektroniczne Systemy Diagnostyki Medycznej i Terapii

Spis treści

Streszczenie

Niniejszy artykuł dotyczy części programu do analizy sygnału elektrokardiograficznego, aby wykrywać w nim wszelkie niezgodności z normami. Ten moduł, nazwany "HeartClass", służy do analizy załamków QRS i grupowania ich według odpowiednio zdefiniowanego podobieństwa.

Słowa kluczowe: elektrokardiografia, klasyfikacja załamków QRS, algorytm k-średnich, algorytm G-średnich, metoda wektorów nośnych.

I. Wstęp

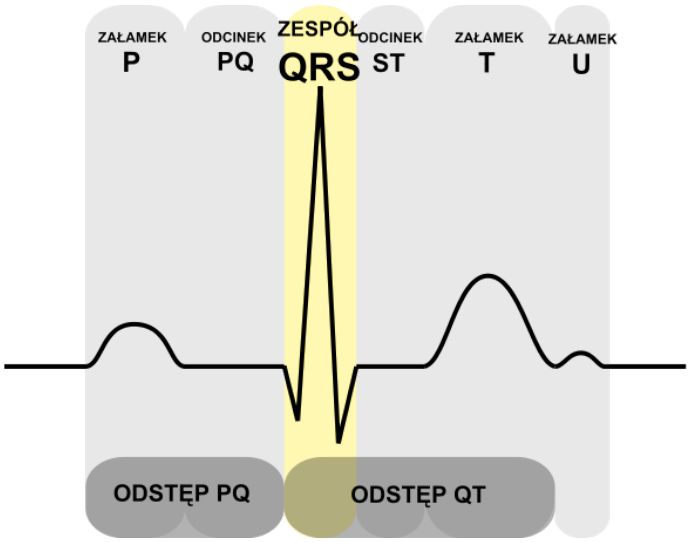
Elektrokardiogram jest jednym z najefektywniejszych narzędzi diagnostycznych do wykrywania chorób serca. EKG dostarcza prawie wszystkich informacji o aktywności elektrycznej serca.

Typowy sygnał EKG składa się z załamka P, zespołu QRS oraz załamków T i U. Spośród wszystkich tych elementów sygału elektrokardiograficznego najbardziej charakterystycznym i zarazem najbardziej znaczącym jest zespół QRS. Na podstawie jego kształtu można zdiagnozować różne dysfunkcje serca, dlatego jego automatyczna klasyfikacja jest ważnym zagadnieniem.

Zespół QRS opisuje pobudzenie mięśni serca i składa się z jednego lub kilku załamków określanych jako Q, R i S.

1. Załamek R – każdy załamek dodatni w obrębie zespołu QRS.
2. Załamek Q – pierwszy ujemny załamek widoczny przed załamkiem R.
3. Załamek S – pierwszy ujemny załamek widoczny po załamku R.

Przykładowy (wyidealizowany) zespół QRS widoczny jest na rys. 1, przedstawiającym schematyczny fragment zapisu elektrokardiograficznego.



Rysunek 1: *Wyidealizowany schemat zapisu EKG z zaznaczonym zespołem QRS. Źródło [1]*

Klasyfikacja zespołu QRS ma na celu wyodrębnienie grup zespołów podobnych (w zadanym zakresie tolerancji). Odmienny kształt zespołu jest konsekwencją odmiennie przebiegającego pobudzenia. Klasyfikacja polega na stwierdzeniu przynależności klasyfikowanego zespołu do jednej z istniejących klas albo tworzenie nowych klas, jeżeli przynależności nie stwierdzono [2].

Celem opisywanego modułu jest wyliczenie liczby klas zespołów QRS, określenie reprezentantów każdej z nich oraz oznaczenie klas zespołów QRS na wykresie EKG. Wyodrębnienie klas QRS występujących w sygnale EKG pozwala na określenie prawidłowości rytmu pracy serca. Z reguły nieregularności mają charakter przejściowy, dlatego ich poprawne wyznaczenie wymaga przeprowadzenia 24-godzinnego badania pracy serca, czyli testu Holtera [3].

Rozwiązanie przyjęte w niniejszej pracy opiera się o ekstrakcję cech z sygnału EKG, klasteryzacji otrzymanych danych przy pomocy algorytmu G-średnich oraz klasyfikacji z wykorzystaniem maszyny wektorów nośnych. Dobór tych metod jest zgodny z wyborem poprzedniego zespołu projektowego, jako że celem niniejszej pracy jest porównanie działania jednego algorytmu realizowanego w różnych językach programowania. Założono, że dane wejściowe dostarczone przez poprzednie moduły są poprawne.

W literaturze można spotkać się z różnymi podejściami do klasyfikacji zespołów QRS. W [4] autorzy zaproponowali algorytm polegający na wykorzystaniu liniowej analizy dyskryminacyjnej (LDA) w celu zredukowania wymiaru przestrzeni cech. Do klasyfikacji zastosowano maszynę wektorów nośnych (SVM). Ponadto w pracy tej podjęto próbę klasyfiacji przy pomocy MLP (ang. Multilayer Perceptrons) oraz klasyfikatora FIS (ang. Fuzzy Inference System). Najlepsze rezultaty autorzy otrzymali wykorzystując SVM. Podobne podejście zastosowano w [5], gdzie dodatkowo w celu ograniczenia zakłóceń oraz ekstrakcji cech wykorzystano transformację falkową. W [6] autorzy stworzyli adaptacyjny algorytm, działający w czasie rzeczywistym, klasyfikujący zespoły QRS. Zaproponowane rozwiązanie bazuje na modelu funkcji Hermite'a. W innej pracy autorzy wyekstrahowali cztery konkretne cechy z sygnału EKG, aby następnie wykorzystać odległość Mahalanobisa jako kryterium klasyfikacji [7].

II. Koncepcja proponowanego rozwiązania

Algorytm klasyfikacji załamków QRS został podzielony na trzy części. Najpierw dane wejściowe zostają znormalizowane i skwantyzowane, następnie przeprowadzana jest procedura ekstrakcji cech. W drugiej części następuje klasteryzacja wektorów cech zespołów QRS. Polega to na grupowaniu tych danych w klasy, które mają najwięcej wspólnego - leżą najbliżej siebie w przestrzeni o wymiarze równym liczbie porównywanych cech (stosowana jest tutaj metryka Euklidesowa). Warto zaznaczyć, iż każdy współczynnik reprezentuje inną wielkość i z tego powodu wartość tolerancji jest dobierana dla każdego z nich indywidualnie. Do klasteryzacji wykorzystywany jest algorytm G-średnich (ang. "G-means"). W ostatnim kroku następuje klasyfikacja, czyli przyporządkowanie każdego zespołu QRS do jednej z klas. W tym celu wykorzystywana jest metoda wektorów nośnych (ang. Śupport Vector Machine", w skrócie SVM).

Dobór tych metod - w tym również wybór klasyfikacji na podstawie wektorów cech, a nie sygnałów - został dokonany przez poprzedni zespół projektowy, a zadaniem autorów niniejszego raportu było przeprowadzenie porównania funkcjonowania tych metod w trzech innych językach programowania niż oryginalny język implementacji.

I. Ekstrakcja cech

Zadaniem tej części zastosowanego algorytmu jest wyliczenie pewnych istotnych wskaźników charakteryzujących zespół QRS na podstawie znormalizowanych danych wejściowych. Bazując na implementacji poprzedniego zespołu projektowego, stosowane są współczynniki wymienione poniżej [3].

1. Początek i koniec całego zespołu QRS,
2. Wartość szczytowa załamka R,
3. Interwał między poprzednim a rozważanym załamkiem R,
4. Interwał między rozważanym a kolejnym załamkiem R,
5. Wartość szczytowa oraz koniec załamka T,
6. Początek, wartość szczytowa i koniec załamka P.

Wszystkie te współczynniki powinny być zapisywane po synchronizacji wszystkich wykrytych zespołów, na przykład względem pozycji załamka R [2].

II. Klasteryzacja

Jak już zostało wspomniane, do grupowania danych w klastry (klasy) użyto algorytmu G-średnich. Jest rozszerzeniem popularnej metody k-średnich, która polega na dobraniu k klas w zbiorze danych tak, aby każdy punkt należał do klasy, do której środka ciężkości ma najbliżej [8].

Przyjęto, że dane, które należy pogrupować to d-wymiarowe wektory należące do zbioru X o liczności n. S to zbiór klas, a więc $S_j = \{x_i \in X | klasa(x_i) = j\}$ dla $i = 1, 2, ..., n$. Zbiór środków ciężkości klas oznaczony został literą C i zdefiniowany jako: $C = \{c_j = \frac{\sum_{x \in S_j} x}{|S_j|}\}$ dla $j = 1, 2, ..., k$. Cel algorytmu k-średnich to minimalizacja wyrażenia przedstawionego wzorem 1 .

$$\sum_{j=1}^{k} \sum_{x \in S_j} \|x - c_j\| \tag{1}$$

Poważnym problemem tego algorytmu jest fakt, iż liczba klas musi być znana bądź przyjęta z góry, co oznacza posiadanie pewnej wcześniejszej wiedzy na temat klasteryzowanego zbioru danych [9, 10]. W przypadku braku takich informacji, należy zastosować uogólnienie algorytmu k-średnich, które pozwoli dobrać optymalne k względem pewnego wskaźnika jakości. Algorytm, który został zastosowany w opisywanym module dobiera k tak, aby w każdej klasie rozkład punktów był możliwie bliski rozkładu normalnego. Stąd też wzięła się litera "G"w nazwie - od rozkładu Gaussa [9].

Metoda G-średnich zaczyna od niewielkiej liczby klas, by później odpowiednio zwiększać k - nie jest przewidziana procedura zmniejszania tego parametru. W pierwszym kroku zwykle przyjmowane jest $k = 1$, z czego wynika, że C jest zbiorem jednoelementowym, zawierającym środek ciężkości całego zbioru X [9]. W każdym kroku algorytm sprawdza, czy dana klasa ma rozkład normalny, a jeśli nie, to dodaje jej dodatkowy środek. Między każdym takim dodawaniem środków jest używana procedura k-średnich, aby poprawić jakość rozwiązania.

Sprawdzenie normalności rozkładu wewnątrz klasy odbywa się za pomocą testu Andersona - Darlinga. Pozwala on rozstrzygnąć, czy znormalizowane dane są rozłożone zgodnie z pewnym rozkładem prawdopodobieństwa. Elementy zbioru danych oznaczono przez $y_i, i = 1, 2, ..., n$. Wartość statystyki Andersona - Darlinga oblicza się na podstawie wzoru 2.

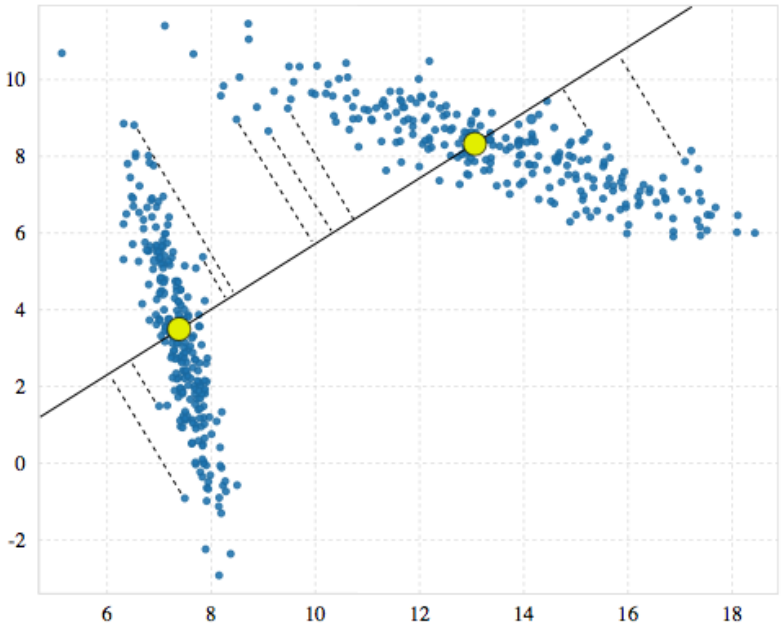
$$A^2 = -n - \frac{1}{n} \sum_{i=1}^{n} (2i - 1)(log(\Phi(y_i)) + log(1 - \Phi(y_{n-i+1}))) \tag{2}$$

Gdy wartość średnia i odchylenie standardowe w testowanym zbiorze danych są obliczane na jego podstawie (a nie znane), wartość statystyki należy poprawić według wzoru 3 [9].

$$A^{2*} = A^2(1 + \frac{4}{n} + \frac{25}{n^2}) \tag{3}$$

Parametrem wejściowym dla tego testu jest poziom ufności α. Jest on zwykle wyrażony w procentach, bądź w ułamku dziesiętnym. Zależy od niego tzw. wartość krytyczna, czyli próg wartości statystyki, ponad którym odrzucana jest hipoteza o rozkładzie normalnym danych. Wzorem autorów algorytmu G-średnich, użyto $\alpha = 0.0001$ [9].

Test Andersona - Darlinga można stosować tylko do jednowymiarowych danych, a więc należy sprowadzić wyjściowy zbiór (o wymiarze d) do przestrzeni liczb rzeczywistych. W tym celu algorytm G-średnich dla każdej klasy używa metody k-średnich z $k = 2$ oraz dwoma środkami $c_j^{1,2} = c_j \pm m$, gdzie m jest wektorem o normie niewielkiej w porównaniu z odległościami między



Rysunek 2: *Przykład rzutowania zbioru danych na wektor łączący znalezione środki ciężkości. Źródło: [10]*

punktami w klasie. Niech otrzymane w wyniku tej operacji środki to c^1 oraz c^2, a wektor u opisuje odległość między nimi: $u = c^1 - c^2$. Cała klasa S_j jest rzutowana prostopadle na wektor u, w wyniku czego otrzymuje się jednowymiarową przestrzeń S'_j, dla której po normalizacji stosuje się test Andersona - Darlinga. W przypadku gdy wartość statystki jest mniejsza niż wartość krytyczna dla danej ufności, nowe środki są odrzucane. W przeciwnym razie zachowuje się je, dzieląc klasę S_j na dwie.

Rysunek 2 przedstawia przykład takiego rzutowania. Widać na nim dwa nowe środki znalezione przez metodę k-średnich dla danej klasy oraz rzutowanie prostopadłe punktów tej klasy na prostą wyznaczaną przez wektor między tymi dwoma środkami. Jak można się spodziewać, rozkład takiego zbioru danych jest bimodalny, a nie normalny, tak więc te dwa widoczne na rysunku środki zostaną przyjęte.

Cały zastosowany algorytm klasteryzacji może być przedstawiony w następujących krokach:

1. Jako parametry wejściowe przyjmij zbiór danych oraz ufność testu Andersona-Darlinga.
2. Wylicz początkowy środek ciężkości: $C = \{\bar{x}\}$.
3. Wykonaj klasteryzację: C = k-średnich(X, C).
4. Dla każdej klasy S_j sprawdź jej rozkład testem Andersona-Darlinga:
   (a) Wylicz dwa środki pochodne c_j^1, c_j^2.
   (b) Wykonaj ponowną klasteryzację: $\{c^1, c^2\} = k - średnich(S_j, \{c_j^1, c_j^2\})$.
   (c) Wyznacz wektor $u = c^1 - c^2$.
   (d) Wyznacz jednowymiarową przestrzeń S'_j.
   (e) Wylicz wartość statystyki Andersona - Darlinga dla S'_j.

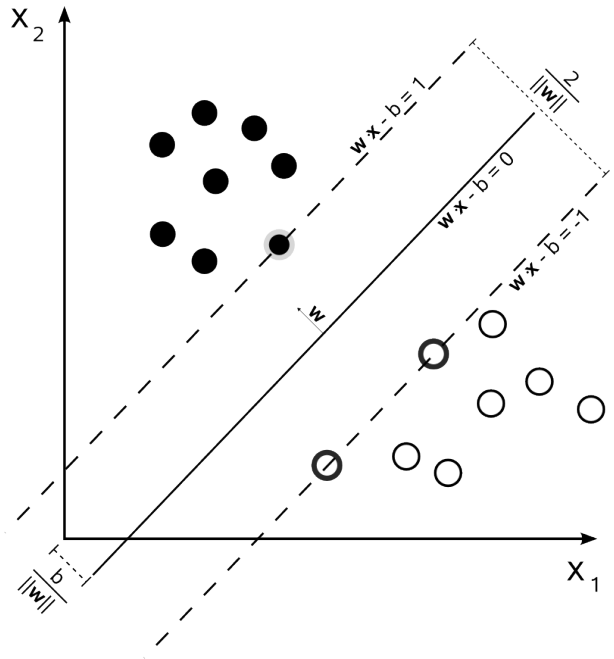
(f) Jeśli jest ona większa od wartości krytycznej, podziel klasę S_j na dwie ze środkami c^1 oraz c^2. Jeśli jest mniejsza, zachowaj poprzedni środek.

5. Powtarzaj od kroku 3 dopóki żadne nowe środki nie zostaną dodane.

III. Klasyfikacja

Aby sklasyfikować powstałe w poprzednim kroku klastry wykorzystano klasyfikator SVM (Support Vector Machine). Problem klasyfikacji wymaga podziału zbioru danych wejściowych na zbiór uczący oraz zbiór testowy. Każdy z elementów zbioru uczącego zawiera wartość oczekiwaną (np. etykieta klasy) oraz jakąś ilość atrybutów (np. wektor cech). Celem SVM jest stworzenie modelu (bazujcego na danych uczących), który przewiduje nieznaną wartość oczekiwaną (etykietę) dla podanego na wejściu elementu zbioru testowego.

W najprostszej postaci klasyfikator SVM służy do wyznaczenia hiperpłaszczyzny rozdzielającej dwa liniowo separowalne zbiory. Hiperpłaszczyzna ta wyznaczana jest z maksymalnym marginesem, tzn. tak, aby suma jej odległości od najbliższych próbek z obu klas była jak największa (patrz rys. 3).



Rysunek 3: *Dwuwymiarowy przypadek hiperpłaszczyzny rozdzielającej dwie klasy z zaznaczonym marginesem. Źródło [11]*

Mając dany zbiór uczący, będący zbirem par składających się z wektora cech oraz etykiety (x_i, y_i), $i = 1, ..., l$, gdzie $x_i \in \mathbb{R}^n$ i $y_i \in \{1, -1\}$, maszyna wektorów nośnych (SVM) wymaga rozwiązania następującego problemu optymalizacji [12]:

$$\min_{w,b,\xi} \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \tag{4}$$

z ograniczeniem:

$$\begin{aligned} &y_i \left(w^T \phi \left(x_i \right) + b \right) \geqslant 1 - \xi_i, \\ &\xi_i > 0 \end{aligned} \tag{5}$$

W wielu przypadkach nie można zagwarantować liniowej separowalności zbiorów. W takich sytuacjach stosuje się tzw. Kernel Trick. Polega to na zwiększeniu wymiaru przestrzeni danych wejściowych, aby w nowej przestrzeni istniała własność liniowej separowalności zbiorów.

W zdefiniowanym wzorami (4) oraz (5) problemie optymalizacji wektor uczacy x_i przekształcany jest do przestrzeni o większym wymiarze dzięki funkcji ϕ. Parametr $C > 0$ jest karą za niespełnienie warunków zadania. Ponadto, funkcja $K(x_i, x_j) = \phi(x_i)^T \phi(x_j)$ jest nazywana funkcją jądra (ang. kernel function). Poniżej wypisano cztery przykładowe funkcje jądra, jakie można spotkać w literaturze poświęconej SVM.

1. Liniowa: $K\left(x_i, y_j\right) = x_i^T x_j$
2. Wielomianowa: $K\left(x_i, y_j\right) = \left(\gamma x_i^T x^j + r\right)^d, \gamma > 0$
3. Radialna funkcja bazwa (RBF): $K\left(x_i, y_j\right) = \exp\left(-\gamma \|x_i - x_j\|^2\right), \gamma > 0$
4. Sigmoida: $K\left(x_i, y_j\right) = \text{tgh}\left(\gamma x_i^T x_j + r\right)$

W opisywanym module wykorzystana została funkcja RBF (ang. Radial Basis Function).

Aby klasyfikator mógł działać wcześniej należy go wytrenować. Polega to na podaniu mu ciągu wektorów uczących. Opisywany klasyfikator został wytrenowany za pomocą bazy danych MIT-BIH Arrhythmia Database [13]. Gotowy model klasyfikatora wczytywany jest z pliku, w którym zapisane są różne parametry oraz zestaw wektorów nośnych, na których opiera się działanie metody SVM.

III. Rezultaty i wnioski

I. Funkcjonowanie algorytmu G-średnich

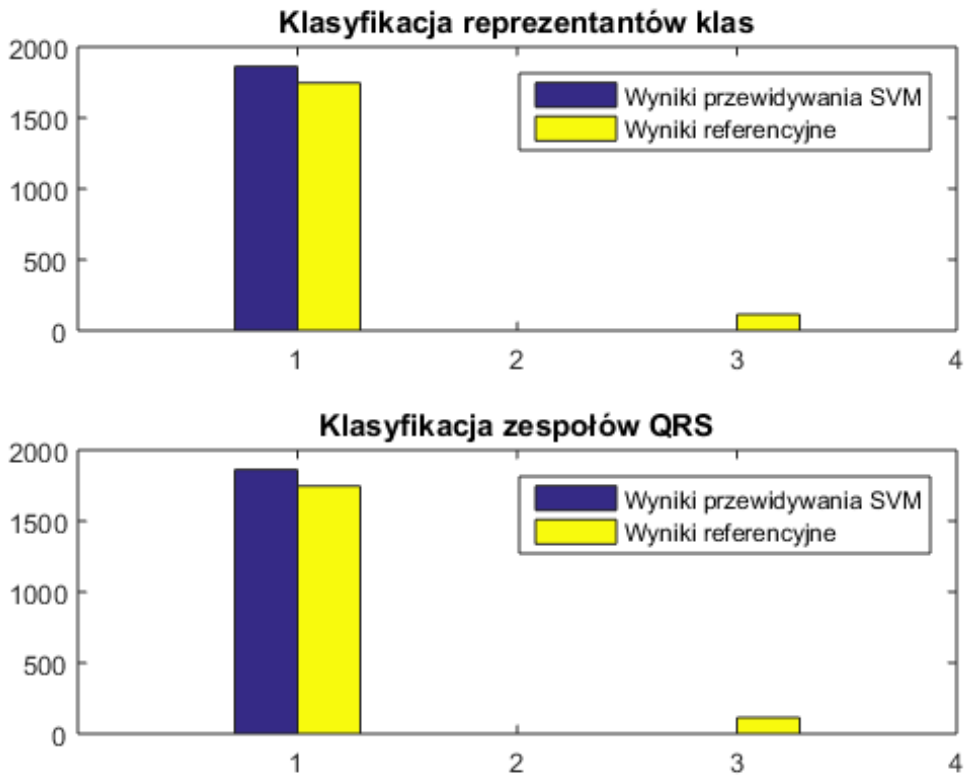
Użyto dwóch wersji algorytmu G-średnich. Pierwsza została znaleziona w internecie - jest to implementacja Laboratori d'Aplicacions Bioacustiques z Politechniki Katalońskiej w Barcelonie. Druga to własna implementacja oparta na oryginalnej koncepcji algorytmu, która jest zawarta w [9].

Tabela 1 przedstawia wyniki działania algorytmu G-średnich w postaci liczby znalezionych klastrów dla każdej paczki danych. Co prosto zauważyć, kolumny dla języków Python i Julia są takie same (przyczyna jest przedstawiona w podrozdziale I Dodatku A). Wyniki otrzymane przy użyciu gotowej implementacji w języku Matlab znacząco się różnią. Jest to spowodowane innym sposobem obliczania wartości krytycznej testu Andersona - Darlinga.

W języku Matlab jest ona bezpośrednio zależna od ilości wektorów cech, które algorytm ma przetworzyć. Konkretna zależność to: $|X| * 0.0005$, gdzie $|X|$ to liczność zbioru danych. Z kolei dla implementacji w języku Python został zastosowany polecany przez twórców algorytmu poziom czułości $\alpha = 0.0001$ i odpowiadająca mu wartość krytyczna $v_{kryt} = 1.8692$. Korzystając z formuły użytej w języku Matlab otrzymuje się znacznie mniejsze dopuszczalne wartości, co zwykle skutkuje zdecydowanie większą liczbą klastrów. Nie udało się natomiast uzasadnić występowania tylko jednego klastra w niektórych przypadkach.

Tablica 1: *Liczby klas wyliczonych przez G-means*

Nr paczki z MIT-BIH	Matlab	Python	Julia
100	16	16	16
101	14	16	16
102	14	16	16
103	27	16	16
104	29	16	16
105	1	20	20
106	35	16	16
107	31	16	16
108	32	15	15
109	1	21	21
111	22	17	17
112	15	19	19
113	13	16	16
114	1	15	15
115	15	16	16
116	1	16	16
117	23	16	16
119	33	16	16
121	27	16	16
122	29	21	21
123	1	16	16
124	30	15	15
200	32	20	20
201	56	13	13
202	29	21	21
203	1	26	26
205	1	21	21
208	36	24	24
209	21	31	31
210	1	20	20
212	18	25	25
213	28	32	32
214	33	18	18
215	18	32	32
217	65	17	17
219	36	15	15
220	22	16	16
221	33	19	19
222	1	22	22
223	1	18	18
230	15	20	20
231	39	12	12
232	28	17	17
233	42	24	24
234	27	29	29



Rysunek 4: *Histogramy klasyfikacji zespołów dla paczki danych o numerze 101*

II. Funkcjonowanie algorytmu SVM

W przypadku implementacji w Matlabie wykorzystano bibliotekę libsvm [12]. Istnieje również wersja tej biblioteki przeznaczona do wykorzystania w Pythonie, jednak po licznych, nieudanych próbach wykorzystania jej zdecydowano się zaimplementować własną maszynę wektorów nośnych w oparciu o pliki źródłowe biblioteki libsvm. Analogicznie postąpiono w przypadku implementacji w Julii. Efektem tego jest krótszy czas wykonywania się modułu napisanego w Matlabia (patrz Tab.2), a wynika to między innymi z braku doświadczenia związanego z optymalizacją kodu w wykorzystanych językach programowania.

III. Wnioski

Krótki czas wykonywania programu napisanego w Matlabie wynika głównie z wykorzystania biblioteki libsvm, która jest napisana w C. Dla pozostałych języków zaimplementowano maszynę wektorów nośnych w oparciu o pliki źródłowe wspomnianej biblioteki.

Wykorzystanie metody SVM z szesnastoelementowym wektorem cech wydaje się złym rozwiązaniem, gdyż niemal zawsze wektor klasyfikowany jest do jednej klasy - patrz Rys.4. Poza zbyt licznym wektorem cech wpływ na to ma również fakt, że około 90% zespołów QRS z bazy MIT-BIH reprezentują pobudzenia nadkomorowe. Rozwiązaniem problemu klasyfikacji może okazać się odpowiednie zmniejszenie liczności wykorzystywanego wektora cech.

Przeprowadzono również testy, w których SVM korzystał z różnych modeli (wygenerowanych z różnych paczek danych). Zaobserwowano, że normalizacja wektorów uczących znacznie przyspiesza proces uczenia, czego potwierdzeniem jest Rys.3. Wątpliwa natomiast jest skuteczność tej metody - niezależnie od tego, czy klasyfikacji poddawane były całe klastry czy tylko ich reprezentanci, wyniki nie różniły się specjalnie od siebie, a więc przytłaczająca większość wektorów cech była przypisywana do jednej klasy. W bazach PhysioBank są dostępne adnotacje lekarzy co do przynależności wybranych kompleksów QRS do rzeczywistych klas pobudzeń. W celu przetestowania metody SVM (lub dowolnego innego algorytmu klasyfikacji), należałoby potraktować obecne tam dane jako punkt odniesienia. Niestety, taka próba nie została podjęta w ramach tego projektu.

Podstawową różnicą względem projektu zeszłorocznego, jest obecność w tu opisywanym metody G-średnich używanej do klasteryzacji. Zostało to szerzej opisane w podrozdziale III Dodatku A. W związku z tym ciężko jest porównywać funkcjonowanie obu modułów. Zaskakujący jest fakt, iż mimo używania takiego samego modelu SVM, implementacje tegoroczne klasyfikują zdecydowaną większość wektorów cech do jednej klasy.

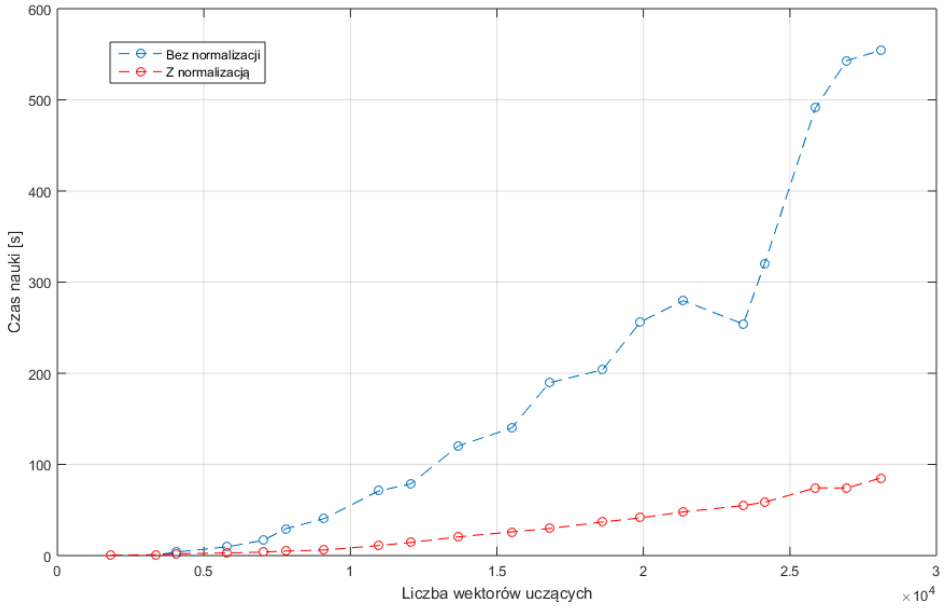
IV. Podsumowanie

W ramach projektu zaimplementowano moduł HeartClass w trzech różnych językach programowania: Matlabie, Pythonie (3.4.x) oraz Julii (0.4.x). Moduł ten ma celu klasyfikowanie zespołów QRS sygnału EKG. W pracy udało się zaimplementować moduł odpowiedzialny za klasteryzację oraz klasyfikację. Z wykonanych testów wynika, że moduł najszybciej wykonuje się w Matlabie. Powodem tego jest głównie fakt, iż w tym module wykorzystano gotową bibliotekę, a nie, jak w przypadku pozostałych dwóch, gdzie zaimplementowano własną maszynę SVM. Dużym zaskoczeniem okazała się implementacja w Julii. Moduł średnio wykonuje się prawie 10 razy szybciej niż w Pythonie i niewiele wolniej niż w Matlabie.

Ciężko dokonać jakiegokolwiek sensownego porównania z bazową implementacją w języku C++. Jest to, oczywiście, spowodowane jej wybrakowaniem, szczególnie w kontekście implementacji algorytmu G-średnich. Co jednak trzeba jej oddać, algorytm SVM funkcjonuje tam

Tablica 2: *Czasy wykonywania programu w poszczególnych językach dla różnych paczek danych. Wyniki wyrażone w sekundach*

Nr paczki z MIT-BIH	C++	Matlab	Python	Julia
100	2.7842	1.9523	12.8154	5.9019
101	2.4618	1.4195	10.4921	1.8999
102	2.7384	1.2609	12.2422	2.2158
103	3.2374	2.3951	11.7082	2.1800
104	2.6769	1.6457	11.8960	2.1335
105	3.4991	1.3186	14.7023	3.1231
106	2.3171	1.8217	10.3660	2.1149
107	2.2940	1.5331	9.9881	2.0690
108	1.9152	1.7391	8.5037	2.1076
109	3.2367	1.2506	13.9100	3.4216
111	2.6589	1.8369	11.5751	2.8741
112	3.2395	2.2931	14.7076	3.3450
113	2.3822	1.2857	10.2796	1.9384
114	1.2847	0.5056	5.5476	1.5342
115	2.7577	1.2325	11.0596	2.2074
116	3.6987	1.242	13.3443	2.5876
117	2.1506	1.506	8.4621	1.6437
119	8.4046	1.5771	10.9920	2.0713
121	2.4702	2.0902	10.2799	1.9764
122	4.0723	2.0928	14.2113	3.1179
123	2.0768	0.7942	8.3970	1.7214
124	1.9720	1.6778	8.1376	1.7144
200	2.5885	1.8849	11.5171	2.8278
201	2.3562	1.8963	9.2493	1.8646
202	2.7007	2.132	13.7891	3.7784
203	3.1012	1.2565	14.5795	3.1471
205	3.3001	1.3475	15.0669	3.2603
208	4.4140	2.4411	15.5731	3.5292
209	3.9942	2.139	17.3970	4.0028
210	3.2321	1.1721	26.9134	6.4331
212	3.5982	2.9475	26.8902	3.3901
213	4.1917	2.3988	31.4794	4.1461
214	2.6582	1.8288	18.5719	2.4209
215	4.5147	1.6584	30.4394	3.9549
217	2.5368	2.0103	18.7012	2.5877
219	2.8246	1.8043	19.7484	3.2817
220	2.7018	1.2670	18.7101	2.6660
221	3.0609	1.7631	13.7086	3.0062
222	3.1459	1.2655	14.3837	3.2238
223	3.0877	1.2356	13.7317	3.0060
230	2.7823	1.8286	14.1915	2.8759
231	2.0888	1.4458	8.6989	3.3912
232	2.2694	1.3707	10.3214	2.355
233	3.4706	2.5759	15.5837	3.4776
234	3.3963	3.6223	16.4922	3.5836
Czas średni	**3.029401**	**1.5836**	**20.6123**	**2.3554**



Rysunek 5: *Wpływ normalizacji na czas nauki SVM*

zdecydowanie lepiej. Jest to niejakim zaskoczeniem, ponieważ wszystkie 3 implementacje korzystają z tej samej biblioteki i tego samego modelu.

Wyniki działania modułu odbiegają od zakładanych - w obliczu braku punktu odniesienia ciężko jest jednak określić, jak bardzo. Przygotowane implementacje konkretnych metod różnią się w pewnych szczegółach i żadna na pewno nie jest idealna. Jak zostało wspomniane wyżej, w podrozdziale III rozdziału III, potrzebna by była kompleksowa analiza wybranych metod względem dobrego wzorca. Mogłyby być nim adnotacje lekarzy dostępne dla niektórych sygnałów w bazach PhysioBank.

V. Bibliografia

[1] QRS Complex, `https://en.wikipedia.org/wiki/QRS_complex` . Stan na: 05.12.2015 r.

[2] Augustyniak, P. *Przetwarzanie sygnałów elektrodiagnostycznych*. Uczelniane Wydawnictwo Naukowo - Dydaktyczne, AGH, Kraków, 2001 r.

[3] Studenci Automatyki i Robotyki. *ESDMiT - Raport końcowy*. 04.02.2015 r.

[4] Song, M. H., i inni. *Support Vector Machine Based Arrhythmia Classification Using Reduced Features*. International Journal of Control, Automation, and Systems, vol. 3, nr 4, strony 571-579, grudzień 2005.

[5] Thakare, A. S. *QRS Complex Detection and Arrhythmia Classification*. Uniwersytet Pune, Indie.

[6] Laguna, P., Jane, R., Caminal, P. *Adaptive Feature Extraction for QRS Classification and Ectopic Beat Detection*. Computers in Cardiology, IEEE, 1991.

[7] Moraes, J.C.T.B., Seixas, M.O., Vilani, F.N., Costa, E.V. *A Real Time QRS Complex Classification Method using Mahalanobis Distance*. Escola Politécnica da Universidade de São Paulo, São Paulo, Brazylia.

[8] MacQueen, J. *Some methods for classification and analysis of multivariate observations*. W materiałach: "Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics", strony 281-297, University of California Press, Berkeley, Kalifornia, USA, 1967.

[9] Hamerly, G., Elkan, Ch. *Learning the k in k-means*. W: Neural Information Processing Systems, MIT Press, 2003.

[10] Użytkownik 'ashenfad'. *Divining the 'K' in K-means Clustering*. `http://blog.bigml.com/2015/02/24/divining-the-k-in-k-means-clustering/`. Dodane: 24.02.2015 r. Stan na: 05.12.2015 r.

[11] Support vector machine, `https://en.wikipedia.org/wiki/Support_vector_machine` . Stan na: 05.12.2015 r.

[12] Chih-Wei, H., Chih-Chung C., Chih-Jen L. A Practical Guide to Support Vector Classification Department of Computer Science, National Taiwan University, Tajpej, Tajwan, 2003 `https://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf`. Stan na 05.12.2015 r.

[13] MIT-BIH Arrhythmia Database. `https://www.physionet.org/physiobank/database/mitdb/` Stan na: 05.12.2015 r.

[14] Dr. Adorio, E. P. *Implementacja testu Andersona-Darlinga w języku Python.* Wersja 0.0.1 . UPDEPP, UP Clarkfield, Stany Zjednoczone, 26.10.2009 r.

Dodatek A: Opis opracowanych programów

Pierwszym faktem, na który należy zwrócić uwagę na początku tego dodatku jest iż praca wykonana w ramach tego projektu miała na celu porównanie funkcjonowania algorytmów w różnych językach programowania. W związku z tym, że traktowano jako odniesienie ubiegłoroczny projekt napisany w języku C++, wejściem wszystkich utworzonych modułów są dane tekstowe pobrane z tamtego projektu. Dołączone na nośniku CD pliki z różnicami zawierają szczegóły zmian wprowadzonych w plikach C++ w celu dodania zapisywania danych w różnych momentach działania algorytmu.

Dane wynikowe są prezentowane użytkownikowi w postaci obiektów w konkretnych językach. Dokładny opis sposobu zwracania danych znajduje się poniżej, w podrozdziale II tego dodatku.

I. Kluczowe aspekty implementacji

W języku Matlab wykorzystano istniejącą implementację algorytmu G-średnich przygotowaną przez Laboratori d'Aplicacions Bioacustiques Politechniki Katalońskiej w Barcelonie. Ten program został poddany prostym testom, aby sprawdzić, jak się zachowuje w różnych przypadkach. Zakończyły się one pozytywnym wynikiem, stąd też decyzja o jego użyciu.

Podjęto próbę samodzielnego napisania tego algorytmu w języku Python w oparciu o opis koncepcyjny, dostępny w oryginalnej pracy twórców algorytmu - [9], a nie na wyżej wymienionym programie w języku Matlab. Próba zakończyła się sukcesem - powstała implementacja dobrze zachowująca się w przypadku różnych typów danych. W związku z tym, w języku Julia użyto modułu „PyCall", aby bezpośrednio odwoływać się do programu w języku Python. Ta implementacja została poniżej szczegółowo omówiona.

Program realizujący algorytm G-średnich został podzielony na dwa pliki. W pierwszym z nich znajduje się implementacja testu statystycznego Andersona - Darlinga przygotowana przez dr Ernesto Adorio [14]. W drugim jest umieszczona klasa `GMeans`, której metoda `cluster_data` używa właśnie omawianego algorytmu. Został on podzielony na części, dokładnie według opisu w [9].

II. Instrukcja uruchomienia programów

Poniżej zamieszczone są krótkie instrukcje uruchomienia plików projektowych w każdym z języków programowania.

1. **Julia**
   (a) Należy posiadać zainstalowany interpreter języka Julia (oprogramowanie było pisane w wersji 0.4.x) oraz następujące paczki:
      i. Gadfly
      ii. Logging
      iii. PyCall
      iv. StatsBase
   (b) Projekt można uruchomić na dwa sposoby. Pierwszy to uruchomienie skryptu `main.jl`. To uruchomi tylko algorytmy.
   (c) Drugi sposób wygeneruje dla każdej paczki danych histogram w formacie SVG. Należy w tym celu uruchomić skrypt `HistSvgGenerator.jl`.
2. **Matlab**

(a) Programy były uruchamiane przy pomocy Matlaba R2014b oraz R2015a.

(b) Jedyną zewnętrzną biblioteką, jaka jest wykorzystywana to LIBSVM. Należy ją skompilować, aby można było wykorzystywać jej artefakty w Matlabie.

(c) Na dołączonym do niniejszego raportu nośniku CD znajdują się skompilowane obiekty w środowisku Windows.

(d) W celu użycia LIBSVM na innym systemie operacyjnym, należy pobrać jej kod źródłowy z oficjalnej strony - `https://www.csie.ntu.edu.tw/~cjlin/libsvm/` - i postępować według instrukcji w pliku README. Odsyła ona do drugiego pliku README wewnątrz katalogu `matlab`, gdzie znajduje się opis kilku prostych kroków, które należy wykonać, aby skompilować tą bibliotekę.

(e) Włączanie programu projektowego odbywa się przez uruchomienie skryptu `main.m`.

(f) Dostępne są również programy testujące algorytmy G-średnich oraz SVM - ich nazwy to odpowiednio: `gmeans_testbench.m` oraz `SVMtest.m` znajdujący się w katalogu SVMtestbench.

3. **Python**

(a) Programy były uruchamiane przy pomocy interpreterów języka Python w wersjach 2.7.x oraz 3.4.x.

(b) Należy posiadać zainstalowany interpreter (najlepiej w wersji 3.x) oraz następujące biblioteki:

i. numpy
ii. scipy
iii. matplotlib
iv. scikit-learn (znane również jako sklearn)
v. biblioteki wbudowane (math, logging, datetime, os)

(c) Aby włączyć program, należy uruchomić skrypt `HeartBeatClassifier.py`.

(d) Dodatkowo, skrypty `gmeans.py` oraz `anderson_darling_test.py` mają zaimplementowane testowanie metod w nich zapisanych. Wykonanie procedury testowej odbywa się przez uruchomienie odpowiedniego skryptu.

III. Różnice względem projektu w C++

Podstawową różnicą względem ubiegłorocznego projektu w języku C++ jest obecność algorytmu G-średnich we wszystkich trzech tegorocznych implementacjach. Nieznane są przyczyny takiego stanu rzeczy. Poprzednia implementacja wykorzystywała do klasteryzacji tylko algorytm k-średnich. Jest on z oczywistych względów nieakceptowalnym rozwiązaniem, gdy nie znamy struktury danych wejściowych, co ma miejsce w przypadku badania grup QRS.

Poza tym, procedura normalizacji cech w ubiegłorocznym projekcie obejmuje wszystkie cechy, nawet te, które są indeksami początków, środków lub końców kolejnych załamków. Wprawdzie literatura nic na ten temat nie mówi, jednak podjęto decyzję, aby nie normalizować tych indeksów - sensowność takiego działania jest wątpliwa.

Wszystkie zmiany, jakie zostały wprowadzone do ubiegłorocznego projektu (ze względu na chęć wypisywania odpowiednich danych) są podsumowane w dwóch plikach tekstowych, `diff_HeartClass_cpp.txt` oraz `diff_HeartClass_h.txt`, które odpowiednio zawierają zmiany w pliku `HeartClass.cpp` oraz `HeartClass.h` ubiegłorocznego projektu. Zostały one umieszczone w katalogu `Raport` w repozytorium projektowym oraz na nośniku CD dołączonym do niniejszego raportu.

Dodatek B: Spis zawartości dołączonego nośnika CD

Nośnik CD, traktowany jako suplement do projektu, zawiera następującą strukturę katalogów, będącą bezpośrednim zapisem repozytorium projektowego:

1. Julia - w środku znajduje się kompletna implementacja w języku Julia. W osobnym folderze zostały umieszczone pliki związane z algorytmem SVM.
2. Literatura - zawiera część plików wymienionych w Bibliografii niniejszego raportu.
3. Matlab - ten katalog mieści 3 kategorie plików:
   (a) skrypty wykorzystywane przez ten projekt,
   (b) skrypty pomocnicze, używane do testowania lub tworzenia wykresów,
   (c) bibliotekę LIBSVM w postaci skompilowanych obiektów w środowisku Windows.
4. Python - podobnie jak w katalogu Julia, znajdują się tutaj pliki związane z implementacją w Pythonie. Osobne foldery zawierają implementację algorytmów G-średnich i SVM. Wszystkie katalogi posiadają również pliki `__init__.py` potrzebne, aby Python rozpoznawał je jako lokalizacje swoich plików źródłowych.
5. Raport - w środku znajdują się pliki źródłowe niniejszego raportu w języku LaTeX, wszystkie użyte tu grafiki oraz sam raport w formacie PDF. Znajdują się tam również pliki z różnicami, które zostały wprowadzone w projekcie ubiegłorocznym.
6. ReferencyjneDane - ten katalog zawiera wszystkie paczki danych, które zostały zapisane w czasie działania projektu ubiegłorocznego w języku C++.
7. SVM_models - znajdują się tam modele stosowane przez algorytm SVM, zapisane w formacie tekstowym.
8. Pliki w głównym katalogu: README.md oraz .gitignore.