Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie

Elektroniczne systemy diagnostyki medycznej i terapii

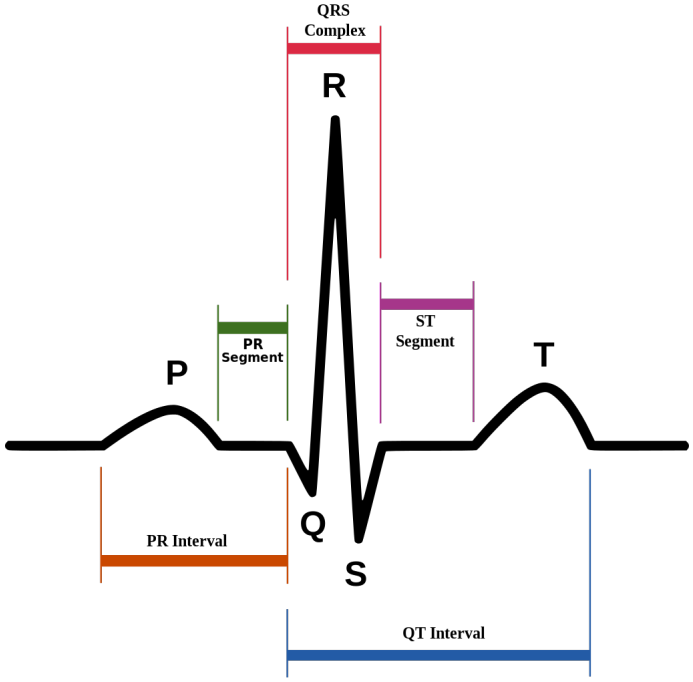
Klasyfikacja pulsu - Naiwny Bayes

Autorzy:
Piotr Janus
Kamil Piszczek

1 Wstęp

Naiwny klasyfikator Bayesa jest prostym klasyfikatorem probabilistycznym opartym na twierdzeniu Bayesa. Jest nazywany naiwnym ze względu na przyjęte założenie, mówiące, że poszczególne cechy są wzajemnie niezależne. Pomimo tak dużego uproszczenia, klasyfikator wypada niespodziewanie dobrze w wielu rzeczywistych problemach. Jego dużą zaletą jest dobra skalowalność, operuje on jedynie na jawnych wzorach w przeciwieństwie do innych metod wykorzystujących podejście iteracyjne.

Jednym z zastosowań klasyfikatora jest diagnozowanie wad i dysfunkcji serca na podstawie sygnału EKG, a dokładniej występującego w nim zespołu QRS. Jest to zespół opisujący pobudzenie mięśni serca. Uproszczony przebieg EKG z zespołem QRS został umieszczony na rysunku 1.



Rysunek 1: Uproszczony zespół QRS – źródło [1]

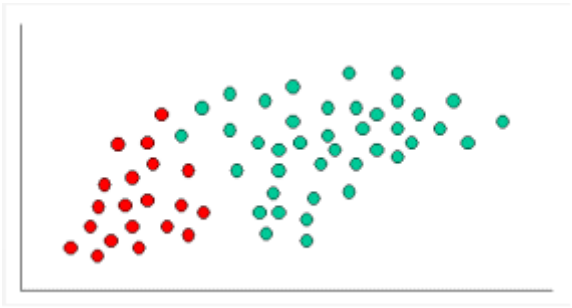
W celu dokonania klasyfikacje konieczne jest zdefiniowanie wskaźników opisujących QRS. Na podstawie rysunku 1 możemy wyróżnić następujący cechy:

- Wartość szczytowa załamka R i moment jej wystąpienie
- Odstęp pomiędzy wcześniejszym a obecnie analizowanym załamkiem R
- Odstęp pomiędzy aktualnie analizowanym i kolejnym załamkiem R
- Początek/koniec oraz początkowa/końcowa wartość załamka P
- Wartość szczytowa załamka P i moment jej wystąpienia
- Początek/koniec i wartość początkowa/końcowa całego zespołu QRS
- Wartość szczytowa załamka T i moment jego wystąpienia
- Koniec i wartość końcowa załamka T

2 Algorytm

2.1 Założenia

Idee metody Naiwnego Bayesa można łatwo wyjaśnić na prostym przykładzie, precyzyjny opis matematyczny został zamieszczony w rozdziale 3. Na rysunku 2 przedstawiony został zbiór punktów, podzielony na dwie klasy (czerwone i zielone). Zadaniem klasyfikatora jest przydzielenie nowego obiektu do jednej z tych klas. W tym przypadku klasyfikacja będzie dokonana na podstawie położenia i elementów znajdujących się w sąsiedztwie nowo dodanego obiektu. Podany zbiór punktów, pełni w tym przypadku rolę zbioru uczącego.

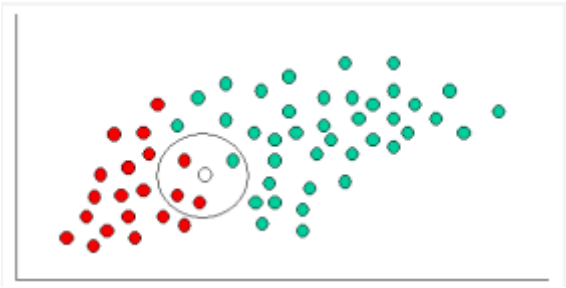


Rysunek 2: Prosty przykład – zbiór punktów (źródło [3])

2.2 Klasyfikacja

W omawianym przykładzie możemy zauważyć, że obiektów zielonych jest dwa razy więcej niż czerwonych. W związku z tym możemy założyć "z góry", że nowy obiekt ma dwa razy większe prawdopodobieństwo bycia zielonym niż czerwonym. Obliczone w ten sposób prawdopodobieństwo nazywane jest prawdopodobieństwem *a priori*.

Wszystkich obiektów jest 60 w czym 40 zielonych i 20 czerwonych. Prawdopodobieństwo *a priori* wylicza się jako iloraz liczby obiektów danego koloru do liczby wszystkich obiektów. Następnie przystępujemy do kolejnego etapu klasyfikacji, przyjmijmy pewne sąsiedztwo nowego punktu (rysunek 3). Możemy założyć, że im więcej obiektów danego koloru w otoczeniu nowego obiektu, tym bardziej prawdopodobne, że jest on tego koloru. Wyznaczone w ten sposób prawdopodobieństwo nazywane jest szansą, oblicza się je jako stosunek liczby obiektów danego koloru w sąsiedztwie, do całkowitej liczby obiektów tego koloru.



Rysunek 3: Prosty przykład – sąsiedztwo (źródło [3])

Mając dane prawdopodobieństwo *a priori* oraz szansę, możemy przystąpić do ostatniego etapu klasyfikacji. Końcowe prawdopodobieństwo czy nowy obiekt należy do danej klasy (jest

danego koloru) obliczane jest jako iloczyn dwóch wyznaczonych wcześniej prawdopodobieństw. Obiekt zostaje oczywiście przypisany do klasy o większym prawdopodobieństwie.

2.3 Wykorzystanie w detekcji pulsu

Omówiony w poprzednich punktach przykład, był bardzo uproszczony i miał na celu jedynie pokazać idee klasyfikatora. Klasyfikacja zespołu QRS jest problemem wielowymiarowym (dokładnie 18 wymiarowym, gdyż taka jest długość wektora cech). W takim przypadku nieco inaczej oblicza "szansa". Osobno wyznaczane jest prawdopodobieństwo przynależności do danej klasy na podstawie każdego elementu z wektora cech. Ostatecznie pod uwagę brany jest iloczyn wszystkich prawdopodobieństw.

Kolejnym zagadnieniem jest sposób obliczania prawdopodobieństwa na podstawie cechy. W przypadku niniejszego projektu wykorzystywany jest w tym celu rozkład normalny (rozdział 3.3). Dla każdej klasy, poszczególne cechy mają przypisaną wartość średnią i odchylenie standardowe. Są one obliczane na podstawie zbioru uczącego w trakcie procesu uczenia.

2.4 Zbiór testowy i uczący

Metoda Naiwnego Bayesa należy do grupy algorytmów nadzorowanego uczenia maszynowego (z nauczycielem). Dane, które podlegają klasyfikacji za pomocą tej metody należy podzielić na dwa zbiory: uczący oraz testowy. Pierwszy z nich wykorzystywany jest w procesie uczenia klasyfikatora. Drugi weryfikuje, czy nauczony klasyfikator ma zdolność generalizacji, a także sprawdza efektywną skuteczność klasyfikacji. Warto zaznaczyć, że algorytmy nadzorowanego uczenia maszynowego mogą stracić zdolność do generalizacji w wyniku zjawiska overfittingu. Następuje ono wtedy, gdy algorytm nadmiernie dostosowuję się do zbioru uczącego - tzn. błąd zbioru uczącego maleje, a błąd zbioru testowego zaczyna rosnąć.

Istnieje wiele sposobów podziału danych na zbiór uczący i testowy. Generalnie metody te nazywane są sprawdzianem krzyżowym [2]. Poniżej opisane zostały najbardziej znane metody.

Pierwszy ze sposobów to metoda Prostej Walidacji. Jest ona obecnie używana w niniejszym projekcie. Metoda ta polega na losowym podziale danych na dwa rozłączne zbiory: uczący i testowy. W tym wariancie zbiór testowy stanowi co najwyżej 1/3 całkowitej próbki danych. Dodatkowo, zastosowana implementacja gwarantuje, że proporcje ilościowe klas w obu zbiorach będą takie same jak w początkowym zbiorze (przed podziałem).

Drugi ze sposobów to K-krotna Walidacja. W tym wariancie dane dzielone są na K różnych podzbiorów. Następnie kolejno każdy z nich bierze się jako zbiór testowy, a pozostałe podzbiory stanowią zbiór uczący. Cały proces powtarzany jest K razy. Otrzymane rezultaty łączone są w jeden za pomocą wybranego sposobu. Najczęściej są one po prostu uśredniane.

3 Opis matematyczny

3.1 Twierdzenie Bayesa

Twierdzenie w teorii prawdopodobieństwa określające zależność między prawdopodobieństwem warunkowym wystąpienia zdarzeń $A|B$ i $B|A$. Przyjmijmy zbiór zdarzeń X, w którym zdarzenia $B_i \in X$, $P(B_i) > 0$ $(i = 1, 2, \ldots, n)$ tworzą układ zupełny (iloczyn każdych dwóch zdarzeń jest zdarzeniem niemożliwym, natomiast suma wszystkich zdarzeń jest zdarzeniem pewnym). Wówczas dla dowolnego $A \in X$ zachodzi następująca zależność:

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{P(A)} \tag{1}$$

Wykorzystując dodatkowo wzór na prawdopodobieństwo całkowite, powyższa zależność może zostać przekształcona do następującej postaci:

$$P(B_i|A) = \frac{P(A|B_i)P(B_i)}{\sum_{k=1}^{n} P(A|B_k)P(B_k)} \tag{2}$$

3.2 Model probabilistyczny

Zdefiniujmy k-elementowy zbiór klas $C = \{C_1, C_2, \ldots, C_k\}$ oraz dane do klasyfikacji opisane jako wektor $x = \{x_1, x_2, \ldots, x_n\}$ zawierający n niezależnych cech. Prawdopodobieństwo przynależności do danej klasy może zostać zapisane z wykorzystaniem twierdzenia Bayesa (rozdział 3.1):

$$P(C_i|x) = \frac{P(x|C_i)P(C_i)}{P(x)} \tag{3}$$

Prawdopodobieństwo $P(x)$ występujące w mianowniku wzoru (3) nie zależy od C i jest stałe, licznik może natomiast zostać przekształcony poprzez wykorzystanie definicji prawdopodobieństwa warunkowego:

$$P(x_1, \ldots, x_n|C_i)P(C_i) = P(x_1, \ldots, x_n, C_i) \tag{4}$$

$$P(x, C_i) = P(x_1|x_2, \ldots, x_n, C_i)P(x_2|x_3, \ldots, x_n, C_i) \ldots P(x_{n-1}|x_n, C_i)P(x_n|C_i)P(C_i) \tag{5}$$

Wykorzystując przyjęte na początku założenie, że cechy $x_1, \ldots, x_n$ są niezależne można wyprowadzić następującą zależność:

$$P(x_j|x_{j+1}, \ldots, x_n, C_i) = P(x_j|C_i) \quad dla \quad j = \{1, 2, \ldots, n-1\} \tag{6}$$

Podstawiając (6) do równania (5) otrzymujemy:

$$P(x_1, \ldots, x_n, C_i) = P(C_i) \prod_{j=1}^{n} P(x_j|C_i) \tag{7}$$

Ostatecznie wzór (3) można zapisać w postaci:

$$P(C_i|x) = \frac{P(C_i) \prod_{j=1}^{n} P(x_j|C_i)}{P(x)} \tag{8}$$

3.3 Rozkłady prawdopodobieństwa

Występujące w równaniu (8) prawdopodobieństwo, że dany obiekt należy do klasy j na podstawie cechy i może zostać wyznaczone na różne sposoby, w zależności od analizowanego problemu. Jak zostało napisane w rozdziale 2.3, w niniejszym projekcie wykorzystywany jest rozkład normalny. Warto pamiętać, że istnieje także możliwość wykorzystania innych funkcji gęstości prawdopodobieństwa.

3.3.1 Rozkład normalny

$$p(x_i|C_j) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(\frac{-(x-\mu_{ij})^2}{2\sigma_{ij}^2}\right), \quad -\infty < x < \infty,\ -\infty < \mu_{ij} < \infty,\ \sigma_{ij} > 0 \tag{9}$$

Gdzie: μ_{ij} – średnia, σ_{ij} – odchylenie standardowe

3.3.2 Rozkład lognormalny

$$p(x_i|C_j) = \frac{1}{x\sigma_{ij}(2\pi)^{1/2}} \exp\left(\frac{-(log(x/m_{ij}))^2}{2\sigma_{ij}^2}\right), \quad 0 < x < \infty,\ m_{ij} > 0,\ \sigma_{ij} > 0 \tag{10}$$

Gdzie: m_{ij} – parametr skali, σ_{ij} – parametr kształtu

3.3.3 Rozkład Gamma

$$p(x_i|C_j) = \frac{(x/b_{ij})^{c_{ij}-1}}{b_{ij}\Gamma(c_{ij})} \exp\left(\frac{-x}{b_{ij}}\right), \quad 0 \leqslant x < \infty,\ b_{ij} > 0,\ c_{ij} > 0 \tag{11}$$

Gdzie: b_{ij} – parametr skali, c_{ij} – parametr kształtu

3.3.4 Rozkład Poissona

$$p(x_i|C_j) = \frac{\lambda_{ij}^x \exp\big(-\lambda_{ij}\big)}{x!}, \quad 0 \leqslant x < \infty,\ \lambda_{ij} > 0,\ x = 0, 1, 2, \ldots \tag{12}$$

Gdzie: λ_{ij} – średnia

4 Implementacja

4.1 Prototyp

Prototyp algorytmu został zaimplementowany w *Pythonie*. Ta implementacja miała na celu dostosowanie opisu matematycznego zamieszczonego w rozdziale 3 do problemu klasyfikacji pulsu i wstępne zweryfikowanie poprawności metody oraz oszacowanie dokładności przed przejściem do właściwej implementacji w *C++*.

Oprócz realizacji samego algorytmu dodano także wiele funkcji upraszczających i pozwalających częściowo zautomatyzować procedurę weryfikacji i testowania. Program daje możliwość wykorzystania jednego zbioru wejściowego, który zostaje automatycznie w losowy sposób rozdzielony na uczący (70%) i testowy (30%). Drugą opcją jest wykorzystanie dwóch osobnych zestawów danych wejściowych, jednego tylko do uczenia i drugiego do testów. Ponad to istnieje także możliwość douczenia modelu z wykorzystaniem kolejnych plików wejściowych. Po załadowaniu danych, program przeprowadza proces generacji modelu prawdopodobieństwa (inaczej proces uczenia), a następnie uruchamia procedurę testową i zwraca dokładność metody, swoistość, czułość oraz opcjonalnie czas wykonania.

Może się okazać, że nie wszystkie z 18 cech przedstawionych w rozdziale 1 mają wpływ na klasyfikację z wykorzystaniem metody Naiwnego Bayesa. W związku z tym dodano także możliwość wyłączenia z modelu prawdopodobieństwa niektórych cech opisujących *QRS*. Takie podejście pozwoliło przetestować wiele przypadków. Dokładny opis i zestawienie przeprowadzonych testów zostało przedstawione w rozdziale 5, natomiast szczegółową instrukcję do programu umieszczono w rozdziale 6.

4.2 Implementacja w C++

4.2.1 Opis

Implementacja w *C++* miała na celu zapewnienie wyższej wydajności w stosunku do implementacji prototypu w *Pythonie*. Sam algorytm został dokładnie odwzorowany bez stosowania żadnych uproszczeń, wszystkie dodatkowe funkcje oraz sposób użycia programu pozostały bez zmian.

Jedynym mankamentem jest odczyt danych wejściowych z pliku. *Python* udostępnia zoptymalizowane funkcje do parsowania danych zapisanych w formacie *csv*. W przypadku *C++* konieczne było napisanie od podstaw funkcji odczytującej. Takie rozwiązanie powoduje, że sam odczyt i przetwarzanie danych z pliku trwa dłużej niż w przypadku prototypu. W związku z tym, aby nie zaburzać rzeczywistego czasu wykonywania się algorytmu zdecydowano się pomiar czasu rozpoczynać w momencie zakończenia odczytu danych (oczywiście w obu wersjach implementacji).

4.2.2 Wykorzystane biblioteki

W celu polepszenia wydajności implementacji. poprawienie funkcjonalności i uproszczenia kodu źródłowego wykorzystano opensourcowe biblioteki. Z racji niewielkich rozmiarów zostały one dołączone do folderu projektowego, takie rozwiązanie ułatwia kompilację i przenoszenie projektu.

Przy implementacji algorytmu zastosowana została bilbioteka *Eigen*[4]. Z jej wykorzystaniem można zoptymalizować operacje matematyczne wykonywane na dużych wektorach danych. W omawianym przypadku zbiór danych wejściowych zawierał do 30 tysięcy rekordów, zastosowanie zoptymalizowanej biblioteki pozwoliło znacząco podnieść wydajność programu a także uprościć implementację.

W celu implementacji obsługi dodatkowych funkcji z poziomu wiersza poleceń wykorzystano bibliotekę *Tclap* [5]. Udostępnia ona prosty interfejs do obsługi argumentów z wiersza poleceń. Jej zaletą jest implementacja jedynie z wykorzystaniem plików nagłówkowych, dzięki czemu jest łatwa w użyciu i integracji z różnymi projektami.

5 Testy

5.1 Dane testowe i metodologia

Do testów zaimplementowanej metody wykorzystano dane referencyjne zamieszczone w repozytorium [6]. Wspomniany zbiór zawiera 38 zestawów danych wejściowych o różnych długościach i złożoności (ilości obecnych klas). Użyto danych zamieszczonych w katalogu `data_unique_labels`, zostały one tak przygotowane, że w każdym z nich klasa oznaczająca zdrową osobę ma to samo id, takie podejście zdecydowanie ułatwia procedurę testowania. Dodatkowo każdy ze zbiorów został rozdzielony na uczący (70%) i testowy(30%).

Po przeprowadzeniu szczegółowej analizy z wykorzystaniem dodatkowych skryptów i narzędzi opisanych w rozdziale 6.3 zdecydowano się przeprowadzić testy algorytmu wykorzystującego wszystkie dostępne cech zespołu *QRS*, ponieważ właśnie taka konfiguracja zapewnia najlepsze wyniki.

W celu sprawdzenia dokładności przygotowanej metody wykorzystano następujące wskaźniki wskaźniki:

1. *skuteczność* definiowana jako procent poprawnie rozpoznanych klas

2. *czułość* – stosunek wyników prawidłowo rozpoznanych jako pozytywnych (prawdziwie dodatnich) do sumy wyników prawdziwie dodatnich i błędnie rozpoznanych jak fałszywe (fałszywie ujemnych), w naszym przypadku przez wynik pozytywnie dodani rozumiane jest poprawne rozpoznanie osoby zdrowej, natomiast negatywnie ujemny to zakwalifikowanie osobny zdrowej do grypy chorych

3. *swoistość* – stosunek wyników prawidłowo rozpoznanych jako negatywne (pozytywnie ujemnych) do sumy wyników pozytywnie ujemnych i błędnie rozpoznanych jako pozytywne (fałszywie dodatnich), w naszym przypadku przez wynik pozytywnie ujemny rozumiane jest poprawne rozpoznanie osoby chorej, natomiast negatywnie dodatni to zakwalifikowanie osobny chorej do grypy zdrowych

Zarówno *czułość* i *swoistość* są wskaźnikami binarnymi, ponieważ w analizowanych zbiorach klasa 1 zawsze oznacza osobę zdrową przyjęto założenie, że klasyfikacja do jakiejkolwiek innej klasy oznacza osobę chorą.

5.2 Wyniki

Przed przystąpieniem do analizy dokładności algorytmu przeprowadzono porównanie wydajności obu implementacji. Czas wykonania obu wersji dla poszczególnych zbiorów testowych zamieszczono w tabeli 1. Zgodnie z oczekiwaniami implementacja w *C++* w każdym przypadku jest szybsza od prototypu. W zależności od testowanego zbioru czas wykonania jest nawet do trzech razy szybszy.

Drugim etapem była analiza dokładności algorytmu, szczegółowe wyniki dla poszczególnych zbiorów zostały zamieszczone w tabeli 2. W przypadku niektórych testów w kolumnach *swoistość* i *czułość* można zauważyć wartości `NaN`, oznacza to że podczas obliczania danego wskaźnika wystąpiło dzielenie przez zero. Taka sytuacja może wystąpić w przypadku gdy w danych zbiorze nie ma w ogóle osób chorych, wtedy nie odnotowujemy żadnego przypadku pozytywnie ujemnego ani fałszywie dodatniego przez co podczas obliczania swoistość dzielimy przez 0. Analogicznie sytuacja może pojawić się podczas obliczania czułości gdy zbiór zawiera jedynie zdrowe osoby.

Plik	Python [s]	C++ [s]
100	0.032	0.015
101	0.017	0.011
102	0.024	0.012
103	0.015	0.006
104	0.013	0.007
105	0.056	0.030
106	0.032	0.013
108	0.003	0.002
109	0.019	0.011
111	0.009	0.004
112	0.042	0.016
113	0.023	0.012
118	0.004	0.002
119	0.023	0.011
121	0.015	0.006
122	0.030	0.011
124	0.030	0.017
200	0.009	0.006
201	0.003	0.002
202	0.022	0.010
203	0.033	0.015
205	0.059	0.026
208	0.050	0.020
209	0.012	0.005
210	0.054	0.037
212	0.022	0.008
213	0.113	0.053
214	0.021	0.007
215	0.019	0.009
217	0.011	0.005
219	0.055	0.024
221	0.013	0.006
222	0.057	0.032
223	0.005	0.004
228	0.033	0.019
231	0.046	0.030
233	0.053	0.023
234	0.070	0.033
Średnia	0.030	0.015

Tablica 1: Porównanie czasu wykonania implementacji Bayes dla Pythona i C++

Plik	Skuteczność [%]	Czułość [%]	Specyficzność [%]
100	100.00	100.00	100.00
101	100.00	100.00	NaN
102	95.05	NaN	95.05
103	100.00	100.00	NaN
104	84.38	18.75	97.50
105	99.59	99.79	50.00
106	98.09	98.42	96.72
108	96.88	100.00	0.00
109	98.83	NaN	98.83
111	100.00	NaN	100.00
112	100.00	100.00	NaN
113	99.56	100.00	0.00
118	87.10	NaN	87.10
119	100.00	100.00	100.00
121	100.00	100.00	NaN
122	100.00	100.00	NaN
124	99.53	NaN	99.53
200	79.17	80.88	50.00
201	51.72	100.00	0.00
202	96.23	96.23	NaN
203	93.66	93.66	NaN
205	99.76	100.00	0.00
208	86.52	86.64	85.00
209	94.92	95.41	88.89
210	99.64	100.00	0.00
212	97.92	100.00	95.56
213	88.53	91.90	51.61
214	90.91	NaN	90.91
215	99.45	99.44	100.00
217	81.82	97.78	47.62
219	89.02	89.24	0.00
221	100.00	100.00	100.00
222	75.67	75.50	76.67
223	96.88	NaN	96.88
228	98.35	98.86	95.00
231	98.69	94.00	99.40
233	96.60	96.80	85.71
234	100.00	100.00	NaN
Średnia	94.06	93.98	69.60

Tablica 2: Wyniki klasyfikacji pulsu za pomocą Naiwnego Bayesa

6 Dodatek A: Instrukcja uruchomienia programów

Repozytorium z projektem dostępne jest pod linkiem: Klasyfikacja pulsu - Naiwny Bayes.

6.1 Program w C++

6.1.1 Kompilacja

Program uruchamiany jest przez wiersz poleceń, kod źródłowy znajduje się w katalogu `Program`. Wszystkie dodatkowe biblioteki umieszczone są w folderze projektowym, nie ma konieczności dołączania zewnętrznych bibliotek. Do kompilacji wykorzystywany jest plik `Makefile`, generujący plik wykonywalny `bayes.exe`. Program po każdym uruchomieniu generuje tymczasowy plik `bayes.dump` z ostatnim zapamiętanym modelem prawdopodobieństwa.

6.1.2 Dane wejściowe

Do uruchomienia programu określenie dwóch plików, jednego zawierającego zbiór zespołów *QRS* i drugiego zawierającego przyporządkowane im klasy. Ścieżki mogą zostać podane jako argumenty w wierszu poleceń, w tym celu należy wykorzystać flagę *–d* w następujący sposób: *–d <ścieżka do qrs_data> –d <ścieżka do class_id>*. W przypadku niezdefiniowania ścieżek zostanie wykorzystane domyślne wejście.

6.1.3 Lista cech

Wykorzystując flagę *–f* można zdefiniować maskę binarną aktywującą poszczególne cechy. Lista cech zawartych w danych testowych jest zgodna z z analizą przedstawioną w rozdziale 1 i przedstawia się następująco:
r_peak, r_peak_value, rr_pre_interval, rr_post_interval, p_onset, p_onset_val, p_peak, p_peak_val, p_end, p_end_val, qrs_onset, qrs_onset_val, qrs_end, qrs_end_val, t_peak, t_peak_val, t_end, t_end_val.
Maska podawana jest w postaci heksadecymalnej z przedrostkiem *0x*. Najmłodszy bit odpowiada pierwszej cesze z listy, przykładowo maska: *–f* $0x0040c$ powoduje, że algorytm bierze pod uwagę jedynie: *rr_pre_interval, rr_post_interval, qrs_onset.*

6.1.4 Pozostałe funkcje

Pozostałe funkcje mogą zostać aktywowane poprzez ustawienie odpowiednich flag:

- -r – resetuje efekty uczenia zapisane w pliku
- -v – rozszerzone logowanie, wyświetla dodatkowo efekty uczenia w postaci wartości średniej i odchylenia standardowego każdej cechy oraz liczbę próbek wykorzystanych do nauczenia danej klasy.
- -l – wykonywany jest jedynie proces uczenia/douczenia z wykorzystaniem podanego zbioru wejściowego
- -t – wykonywane są jedynie testy z wykorzystaniem podanego zbioru wejściowego, model probabilistyczny musi zostać wygenerowany wcześniej z użyciem flagi *–l* i jest odczytywany z pliku
- -m – mierzy i wyświetla czas wykonania programu

Jedyna zabroniona kombinacja to użycie jednocześnie flag *–l* i *–t*. W celu wykorzystania tylko jednego zbioru i automatycznego rozdzielenia danych na zbiór na testowy oraz uczący należy pominąć obie flagi. Dokładny opis korzystania programu został zamieszczony w rozdziale 6

6.1.5 Przykład użycia

Kolejność podawania flag przy uruchomieniu programu jest dowolna, przykładowe wywołanie programu:
`bayes --d ../ReferencyjneDane/100/train_data.txt --d ../ReferencyjneD ane/101/train_label.txt --r --l --f` $0x3fffe$
Instrukcja ta powoduje usunięcie modelu prawdopodobieństwa zapisanego w pliku (jeżeli istnieje) i wygenerowanie nowego (bez przeprowadzenia testów z powodu flagi *–l*). Zgodnie ze zdefiniowaną maską, algorytm uwzględni wszystkie cechy oprócz pierwszej (*r_peak*).

6.2 Prototyp

Model programowy algorytmu został napisany przy pomocy Pythona. Do skonfigurowania środowiska uruchomieniowego dla projektu służą poniższe instrukcje:

1. Kod programu jest kompatybilny z interpreterem języka Python w wersji 2.7.x i znajduje się w folderze `Model`.

2. Aby włączyć program, należy uruchomić skrypt `main.py` z odpowiednimi argumentami, składnia jest identyczna jak w przypadku programu w *C++*.

3. Wynik działania programu jest przekierowany na standardowe wyjście oraz zapisany do pliku `bayes_logger.txt`.

Program po każdym uruchomieniu generuje tymczasowy plik `bayes.dump` z ostatnim zapamiętanym modelem prawdopodobieństwa. Przykład użycia:
`bayes --d ../ReferencyjneDane/101/test_data.txt --d ../ReferencyjneD ane/101/test_label.txt --t`
Instrukcja ta powoduje przeprowadzenie testów z wykorzystaniem podanych plików wejściowych, model prawdopodobieństwa zostanie odczytany z tymczasowego pliku wygenerowanego w trakcie poprzedniego wywołania programu.

6.3 Dodatkowe skrypty

Dodatkowo w folderze `scripts` znajdują się skrypty, które wspomagały testowanie implementacji algorytmu oraz jego optymalizację. Zostały one również napisane w Pythonie:

1. skrypty `test_distributed.py` oraz `time_compare.py` przeprowadzały naukę i klasyfikację na poszczególnych folderach z danymi. Rezultaty były zapisywane do pliku wynikowego w postaci tabeli w języku `LaTeX`. Pierwszy skrypt wykonywał testy jakościowe, a drugi mierzył czas działania programów.

2. skrypt `random_search.py` testował losowe wektory cech spośród 18 bazowych cech. Został użyty do poszukiwania optymalnych cech dla klasyfikatora

Literatura

[1] Wikipedia, hasło: *QRS complex*, https://en.wikipedia.org/wiki/QRS_complex (ostatni dostęp 11.01.2017)

[2] Wikipedia, hasło: *Sprawdzian krzyżowy*, https://pl.wikipedia.org/wiki/Sprawdzian_krzy%C5%BCowy (ostatni dostęp 11.01.2017)

[3] Internetowy Podręcznik Statystyki, rozdział: Naiwny klasyfikator Bayesa, http://www.statsoft.pl/textbook/stathome.html (ostatni dostęp 11.01.2017)

[4] Eigen, http://eigen.tuxfamily.org/index.php?title=Main_Page (ostatni dostęp 11.01.2017)

[5] Tclap, http://tclap.sourceforge.net/ (ostatni dostęp 11.01.2017)

[6] Dane referencyjne, https://github.com/wgml/ecg-classification/tree/master/data (ostatni dostęp 12.01.2017)