PD-R-Py 2019/2020

Praca domowa nr 2 (max. = 30 p.)

W ramach niniejszego projektu zaimplementujesz i przetestujesz algorytm k-najbliższych sąsiadów (k-nn).

Maksymalna ocena: 30 p. Termin oddania pracy: 17.05.2020, godz. 23:59.

Do przesłania na adres prowadzącego laboratoria `M.Bartoszuk@mini.pw.edu.pl` lub `A.Cena@mini.pw.edu.pl` ze swojego konta pocztowego `*@*pw.edu.pl`:

- jedno archiwum ZIP (czyli nie: RAR, 7Z itp.) o nazwie typu `NNick_Nazwisko_Imie_NrAlbumu_pd2.zip`.

W archiwum powinien koniecznie znajdować się jeden katalog, `Nick_Nazwisko_Imie_NrAlbumu_pd2`, w którym umieszczone są następujące pliki:

- plik `knn.R` zawierający implementację procedury znajdowania k-najbliższych sąsiadów w postaci funkcji `knn()` oraz opisanych niżej funkcji agregujących, a także dodatkowych funkcji pomocniczych;
- plik `testy.Rmd` – testy poprawności implementacji funkcji `knn()` na przynajmniej trzech zbiorach danych z $\mathbb{R}^2$ (z ilustracjami m.in. w postaci wykresów) jako raport wygenerowany przy użyciu pakietu `knitr`; w szczególności należy sprawdzić, czy 1-nn w przypadku, gdy próba ucząca i testowa są tożsame, pozwala dokładnie odtworzyć wektor prawdziwych etykiet;
- plik `testy.pdf` – skompilowana wersja powyższego;
- plik `raport.Rmd` – raport z analizy danych benchmarkowych stworzony przy użyciu pakietu `knitr`;
- plik `raport.pdf` – skompilowana wersja powyższego;
- pliki `.csv` zawierające wyniki benchmarków – to na ich podstawie utworzysz raport;
- pliki `.R` generujące wyniki benchmarków i inne skrypty pomocnicze.

Punktacja za poszczególne etapy to:

- implementacja - [12p]
- testy - [3p]
- raport - [15p]

Nazwy plików nie powinny zawierać polskich liter diakrytyzowanych (przekształć $ą \rightarrow a$ itp.).

Uwaga: temat wiadomości to `[PDRPy] Praca domowa nr 2`.

`Nick` to wymyślony przez Ciebie identyfikator, który pojawi się w arkuszu ocen i zapewni Ci odpowiednią anonimowość. Uzyj koniecznie nicku, który wybrałaś/eś przy okazji pracy domowej nr 1.

1 Zadanie klasyfikacji

Niech dana będzie macierz postaci

$$\mathbf{X} = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1d} \\ x_{21} & x_{22} & \dots & x_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n1} & x_{n2} & \dots & x_{nd} \end{bmatrix} \in \mathbb{R}^{n \times d},$$

gdzie i-ty wiersz macierzy $\mathbf{X}$ (obserwacja), $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})$ opisuje cechy i-tego obiektu, $i = 1, \dots, n$ (nazywane też atrybutami lub po prostu zmiennymi okreslają te cechy). Dodatkowo mamy dany wektor:

$$\mathbf{y}^T = (y_1, y_2, \dots, y_n)$$

taki, że $y_i \in \{s_1, s_2, \ldots, s_l\}$ określa przynależność każdej obserawacji do jednej z l klas, $l >= 2$, $i = 1, \ldots, n$.

Macierz $\mathbf{X}$ i wektor $\mathbf{y}$, a dokładniej pary $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$ będziemy nazywać **zbiorem uczącym**.

Zadanie klasyfikacji[1] polega na wyznaczeniu reguły decyzyjnej w oparciu o zbiór uczący, która pozwoli na wyznaczenie klasy dla nowych obserwacji.

2 Metoda k-najbliższych sąsiadów

2.1 Wyznaczenie najbliższych sąsiadów

Napisz funkcję `knn()` (koniecznie taka nazwa), która przyjmuje jako argumenty kolejno:

1. macierz rzeczywistą $\mathbf{X}$ typu $n \times d$, reprezentującą n punktów w $\mathbb{R}^d$ (zbiór treningowy);
2. n-elementowy wektor liczb naturalnych, $\mathbf{y}$, gdzie y_j reprezentuje liczbowy kod etykiety odpowiadający obserwacji `X[j,]`;
3. macierz rzeczywistą $\mathbf{Z}$ typu $m \times d$, reprezentującą m punktów w $\mathbb{R}^d$ (zbiór testowy);
4. liczbę całkowitą $1 \leq k \leq n$, oznaczającą liczbę najbliższych sąsiadów biorących udział w poszukiwaniu etykiety odpowiadającej punktom ze zbioru testowego;
5. wartość rzeczywistą $p \geq 1$, domyślnie równą 2, określającą, która metryka Minkowskiego L_p będzie używana do poszukiwania najbliższych sąsiadów (możliwe jest, by $p = \infty$).

Funkcja ma zwracać macierz liczb naturalnych typu $m \times k$, $\mathbf{W}$, gdzie `W[i, j]` reprezentuje etykietę j-tego najbliższego sąsiada `Z[i,]` spośród punktów w $\mathbf{X}$ (względem przyjętej metryki).

> *Uwaga: Funkcji tej nie musisz implementować w Rcpp. Być może pewne realizowane tu podzadania warto zlecić funkcjom pomocniczym, które już w Rcpp opłaca się zaimplementować (dla chętnych).*

3 Funkcje agregujące etykiety

W klasycznej metodzie k-nn, zwłaszcza w przypadku danych jakościowych, „odgadywaną" dla i-tej obserwacji `Z[i,]` etykietę wyznacza się przy użyciu mody (dominanty). W pliku `knn.R` zaimplementuj więc funkcję `moda()`, która na wejściu przyjmuje macierz liczb naturalnych typu $m \times k$ i zwraca m-elementowy wektor $\mathbf{r}$ taki, że r_i jest modą z `W[i,]`. Jeśli moda nie jest określona jednoznacznie, należy wybrać losową spośród najczęściej występujących wartości (aby estymator nie był obciążony).

W zadaniu klasyfikacji dla danych porządkowych (zwanym także regresją porządkową, od ang. *ordinal regression*) – a takie będziemy tutaj rozważać – zakłada się, że na zbiorze etykiet została określona relacja porządku liniowego (por. argument `ordered` w funkcji `factor()` – bez straty ogólności zakładamy dalej, że zmienna $\mathbf{y}$ jest określona na zbiorze $\{1, \ldots, l\}$ dla pewnego l, na którym jest określony naturalny porządek $\leq$). Zadanie to jest więc w pewnym sensie „łatwiejsze" niż ogólne zadanie klasyfikacji, w którym zakłada się, że na zbiorze etykiet jest określona jedynie relacja równoważności.

W takiej przestrzeni możemy więc rozważać inne niż moda funkcje agregujące. Zaimplementuj w pliku `knn.R` następujące z nich.

- `srednia_a()` – wyznacza najbliższą wartość naturalną do średniej arytmetycznej z etykiet w każdym wierszu; jeśli taka wartość nie jest określona jednoznacznie, należy zwrócić wartość losową, np. dla $(1, 1, 2, 2, 1, 2)$ należy losowo wybierać liczby spośród zbioru $\{1, 2\}$;
- `mediana()` – zwraca najbliższą wartość naturalną do mediany (jw.);

[1] Zob. np. [Koronacki J., Ćwik J., *Statystyczne systemy uczące się*, EXIT, 2008, rozdz. 9] lub [Hastie T., Tibshirani R., Friedman J., *The Elements of Statistical Learning*, Springer, 2017] – http://web.stanford.edu/~hastie/ElemStatLearn/

- `minkara1.5()` – dla każdego i zwraca wartość u, która minimalizuje funkcję straty $\sum_{j=1}^{k} |\texttt{W[i,j]} - u|^{1.5}$;
- `minkara3.0()` – dla każdego i zwraca wartość u, która minimalizuje funkcję straty $\sum_{j=1}^{k} |\texttt{W[i,j]} - u|^{3.0}$;
- jakaś (co najmniej jedna) inna funkcja agregująca własnej produkcji, np. ważona średnia arytmetyczna z malejącymi wagami (najbliższy sąsiad dostaje większą wagę niż dalszy).

Każda funkcja powinna przyjmować na wejściu macierz, a zwracać wektor etykiet (tak samo, jak funkcja `moda()`). Ciekawostka: `srednia_a()` to `minkara2.0()`, a `mediana()` – `minkara1.0()` w naszym ustawieniu.

4 Ocena jakości klasyfikatorów

Do celów testowych wykorzystujemy zbiory danych pobrane ze strony http://gagolewski.com/resources/data/ordinal-regression/. Każdą ramkę danych należy „rozbić" na dwa obiekty: pierwsza kolumna to zmienna odpowiedzi (wektor etykiet), pozostałe kolumny należy zrzutować na macierz liczbową.

Aby oszacować jakość działania klasyfikatora, skorzystamy z 5-krotnej kroswalidacji. Dany zbiór benchmarkowy $(\mathbf{X}, \mathbf{y})$ dzielimy losowo na 5 prawie-równych części: $\mathbf{X}^{(1)}, \ldots, \mathbf{X}^{(5)}$ oraz $\mathbf{y}^{(1)}, \ldots, \mathbf{y}^{(5)}$. Dla każdego $i = 1, \ldots, 5$, $(\mathbf{X} \setminus \mathbf{X}^{(i)}, \mathbf{y} \setminus \mathbf{y}^{(i)})$ traktujemy jako próbę uczącą, a $(\mathbf{X}^{(i)}, \mathbf{y}^{(i)})$ – testową, tzn. obliczamy $\mathbf{w}^{(i)} =$ `klasyfikator`$(\mathbf{X} \setminus \mathbf{X}^{(i)}, \mathbf{y} \setminus \mathbf{y}^{(i)}, \mathbf{X}^{(i)},$ *dodatkowe_argumenty*$)$. Następnie obliczamy następujące miary błędów:

- proporcja błędnej klasyfikacji:

$$\mathrm{ERR}^{(i)} = \frac{1}{|\mathbf{w}^{(i)}|} \sum_{j=1}^{|\mathbf{w}^{(i)}|} \mathbf{1}(w_j^{(i)} \neq y_j^{(i)}),$$

  gdzie $\mathbf{1}(a \neq b) = 1$ jeśli $a \neq b$ oraz 0 w przeciwnym przypadku;
- błąd bezwzględny:

$$\mathrm{MAD}^{(i)} = \frac{1}{|\mathbf{w}^{(i)}|} \sum_{j=1}^{|\mathbf{w}^{(i)}|} |w_j^{(i)} - y_j^{(i)}|;$$

- błąd średniokwadratowy:

$$\mathrm{MSE}^{(i)} = \frac{1}{|\mathbf{w}^{(i)}|} \sum_{j=1}^{|\mathbf{w}^{(i)}|} |w_j^{(i)} - y_j^{(i)}|^2.$$

Następnie jako miarę błędu danej metody (na konkretnym zbiorze danych) przyjmujemy średnią arytmetyczną z powyższych 5 pomiarów, np. $\mathrm{MSE} = (\mathrm{MSE}^{(1)} + \cdots + \mathrm{MSE}^{(5)})/5$.

5 Metody do przetestowania

Należy zbadać następujące algorytmy klasyfikacji:

- k-najbliższych sąsiadów z metryką L_1, L_2 oraz L_∞ dla wszystkich kombinacji różnych k, np. $1, 3, 5, 7, 9, 11, 13, 15, 17, 19$, oraz funkcji agregujących;
- algorytm lasów losowych (`randomForest::randomForest()`);
- regresję logistyczną dla danych porządkowych (*ordinal logistic regression* lub *proportional odds model*, `MASS::polr()`);
- co najmniej jedną inną metodę klasyfikacji dostępną w R.

Wyniki przedstaw w estetycznym i czytelnym raporcie z **badań** (tabele, wykresy, dyskusja, opis wyników). Nie zapomnij dokonać podsumowania szeroko pojętej *jakości* działania algorytmów na wszystkich zbiorach benchmarkowych, przeprowadź dyskusję itd.

Dodatkowo (aby uzyskać maksymalną liczbę punktów):

- zbadaj zachowanie się metody k-najbliższych sąsiadów z metryką L_1, L_2 oraz L_∞ przy założeniu, że cały zbiór benchmarkowy (przed podziałem na 5 części) został wystandaryzowany, tj. od każdej kolumny w $\mathbf{X}$ odejmujemy jej średnią arytmetyczną a następnie dzielimy przez odchylenie standardowe;
- inne pomysły na eksperymenty mile widziane.

Uwaga: brana będzie pod uwagę jakość kodu. Na przykład kod należy zamknąć w dobrze udokumentowane, wyspecjalizowane funkcje tak, by uniknąć powtórzeń itp.