Dane wymagane przez prowadzącego: <https://archive.ics.uci.edu/ml/datasets/Yeast>

Myślałam dodatkowo o: <https://archive.ics.uci.edu/ml/datasets/Computer+Hardware>

http://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Diagnostic%29

PD2

Zadanie 1:

a) Stwórz funkcję przyjmującą następujące parametry: "dane", "Ynazwa", "XnazwyList", "Nrdzeni", "metoda".

b) Funkcja operując na zbiorze "dane" powinna tworzyć model regresji liniowej dla Ynazwa w odniesieniu do zmiennych XnazwyList.

W najprostszej postaci dla danych z "Zadanie 0" są to modele: ("Y~x1", "Y~x2", "Y~x3" etc.).

To jakiej postaci model powinien być zbudowany, definiowane jest przez parametr "XnazwyList", przyjmujący obiekt typu lista.

Lista ma tyle elementów, ile modeli będzie zbudowanych. Każdy element listy jest wektorem nazw zmiennych "x".

Przykład: "list(x1,c(x1,x5,x7))" buduje dwa modele 1) "Y~x1" oraz 2) "Y~x1+x5+x7".

c) Funkcja powinna budować każdą kombinację modeli równolegle.

d) W zależności od przekazanego argumentu do "metoda", funkcja wykorzystywać powinna albo równoleglą wersję "lapply",

albo równoleglą wersję pętli "for".

e) Każda równoległa pętla powinna zwracać informacje o nazwach zmiennej/zmiennych (pierwsza kolumna tekstowa)

i oszacowaniach parametrów (druga kolumna numeryczna).

f) Funkja powinna zwracać wyniki w formie listy.

g) Nazwa funkcji to "ModelParallel".

PD3

Zadanie 1:

a) Stwórz funkcję "ModelOcena" przyjmującą nastęujące parametry: "y\_tar" (rzeczywista), "y\_hat" (prognoza).

b) Funkcja w pierwszym kroku powinna rozpoznawać czy mamy do czynienia z problemem regresji czy klasyfikacji.

y\_tar: regresja -> numeric, klasyfikacja -> factor.

c) W zależności od problemu funkcja szacować powinna różnego rodzaju błędy:

Regresja: MAE, MSE, MAPE.

Klasyfikacja: AUC (szacowanie metodą trapezów), macierz klasyfikacji (punkt odcięcia wyznaczany jest poprzez index Youdena),

Czułość, Specyficzność, Jakość.

d) Dla czytelności kodu, wszystkie miary powinny być liczone w oparciu o zewnętrzne funkcje (dowolne nazwy),

których definicje znajdować się powinny przed definicją funkcji "ModelOcena".

e) Funkja powinna zwracać wyniki w formie:

Regresja: nazwany wektor (MAE, MSE, MAPE) o trzech elementach.

Klasyfikacja: nazwana lista (Mat, J, Miary) o trzech elementach:

Mat = macierz klasyfikacji, w wierszach znajdują się wartości "y\_tar" a w kolumnach wartości "y\_hat",

nazwy wierszy i kolumn muszą być zgodne z dostępnymi etykietami klas.

J = wartość indexu Youdena,

Miary = nazwany wektor o elementach AUC, Czułość, Specyficzność, Jakość.

f) Funkcja będzie testowana tylko dla klasyfikacji binarnej i regresji.

Zadanie 2:

a) Stwórz funkcję "CrossValidTune" przyjmującą nastęujące parametry: "dane", "kFold", "parTune", "seed".

W skrócie: funkcja powinna krosswalidacyjnie tunować parametry danego algorytu.

b) Funkcja powinna w pierwszym kroku stworzyć listę przechowującą informację, które obserwacje posłużą jako zbiór treningowy,

a które wejdą do zbioru walidacyjnego. Ilość elementów listy zdefiniowana jest przez parametr "kFold" (liczba podzbiorów walidacyjnych).

Każdy element listy jest wektorem o tej samej długości, równej nrow("dane").

Każdy z wektorów zawiera albo liczbę 1 (obserwacja jest w zbiorze treningowym) albo 2 (obserwacja jest w zbiorze walidacyjnym).

Przykład: list( c(1,2,1,1), c(2,1,1,2) ) - oznacza, że mamy doczynienia z 2-krotną walidacją na zbiorze z 4 obserwacjami,

gdzie dla pierwszej iteracji tylko jeden element jest w podzbiorze walidacyjnym.

Losowanie rozpoczyna się od ustawienia ziarna na wartość "seed".

c) W kolejnym kroku funkcja powinna stworzyć ramkę danych, w której przechowywane będą wyniki oraz kombinacje parametrów.

Liczba wierszy i kolumn zależy od zagadnienia (klasyfikacja, regresja) oraz od liczby tunowanych parametrów "parTune" i "kFold":

Przykład: "parTune" = data.frame( a = c(1,2), b = c(1,1) ) - oznacza, że algorytm ma 2 parametry do tunowania,

Dla "kFold" = 2 oraz "parTune", kombinacja parametrów to data.frame( "kFold" = c(1,2,1,2), a = c(1,2), b = c(1,1) ).

Kolejne kolumny tabeli wynikowej powinny stanowić miary uzyskane dzięki funkcji "ModelOcena".

Regresja: MAEt, MSEt, MAPEt, MAEw, MSEw, MAPEw - ozanczają miary dla zbioru treningowego i walidacyjnego.

Finalnie tabele jest rozmiaru 4x9.

Klasyfikacja: AUCT, CzułośćT, SpecyficznośćT, JakośćT, AUCW, SpecyficznośćW, MAPEW, JakośćW - j.w.

Finalnie tabele jest rozmiaru 4x11.

d) W ostatnim kroku funkcja powinna budować w pętli model predykcyjny dla danej kombincaji parametrów i uzupełniać tabelę wynikową.

Z racji tego, że nie stworzyliśmy na razie żadnego algorytmu ta część powinna działać następująco:

Każda pętla tworzy dwa podzbiory zdefiniowane przez wektor znajdujący się w liście z pkt b) dla danej kombinacji.

Do kolumn z miarami jakości wstawiane są wartości równe 0.

e) Funkcja zwraca tabelę wynikową.

PD4

Zadanie 1:

a) Stwórz funkcję "KNNtrain" przyjmującą nastęujące parametry: "X", "y\_tar", "k", "XminNew", "XmaxNew".

b) Funkcja w pierwszym kroku powinna sprawdzać czy analiza jest możliwa do wykonania tj:

czy "X" oraz "y\_tar" nie mają braków danych, czy "k" jest większe od 0, czy "X" jest macierzą lub ramką danych.

c) W drugim kroku funkcja dokonuje normalizacji zmiennych z "X" do przedziału "XminNew", "XmaxNew". Jest to możliwe dla zmiennych na skali ilorazowej.

Nowa tabela powinna posiadać 3 atrubuty informujące o (każdy jest wektorem):

- wartościach minimalnych attr(\*,"minOrg") dla każdej zmiennej,

- wartościach maksymalnych attr(\*,"maxOrg") dla każdej zmiennej,

- nowych wartościch minimalnych i maksymalnych attr(\*,"minmaxNew").

d) W kolejnym kroku w obiekcie typu lista o trzech elementach "X","y","k", funkcja umieszcza odpowiednie obiekty.

e) Funkcja powinna zwracać listę z pkt d).

f) Funkcja będzie testowana tylko dla klasyfikacji binarnej i regresji.

\*\*\*) Dla ambitnych: Funkcja może przyjmować dodatkowo parametr "metoda = brute". Oznacza to, że domyślnie nie następuje faza treningowa,

lecz w przypadku przekazania argumentu "kdtree", funkcja przeorganizuję tabelę "X" w pomocniczą strukturę k-d tree.

Taki zabieg będzie wpływał na sposób wykonania Zadania 2. Ta część może być napisana w C/C++.

Struktura k-d tree jest możliwa do stworzenia dla zmiennych na skali ilorazowej.

Zadanie 2:

a) Stwórz funkcję "KNNpred" przyjmującą nastęujące parametry: "KNNmodel", "X".

b) Funkcja w pierwszym kroku powinna sprawdzać czy predykcja jest możliwa do wykonania tj:

czy "X" nie ma braków danych, czy wszystkie potrzebne zmienne/kolumny istnieją zarówno w "KNNmodel$X" jak i w "X".

c) Następnie funkcja powinna normalizaowć dane z "X" uwzględniając informacje zawarte w atrybutach obiektu "KNNmodel$X".

d) W zależności od typu zmiennych w obiektach "KNNmodel$X" i "X" odległość pomiędzy obserwacjami powinna być liczona odległością:

Euklidesa, Hamminga lub Gowera.

e) Funkcja powinna następnie rozpoznawać czy mamy do czynienia z problemem regresji czy klasyfikacji,

a następnie dokonywać odpowiedniej agregacji wyników (średnia lub głosowanie większościowe).

f) Funkcja powinna zwracać:

- dla regresji: wektor z wartościami przewidywanymi.

- dla klasyfikacji: nazwaną ramkę danych o rozmiarze "n x k+1", np. dla wersji binarnej o etykietach "P", "N",

tabela wygląda następująco: data.frame( P = c(0.3,0.6), N = c(0.7,0.4), Klasa = c("N","P") ),

tj. pierwsze dwie kolumny zawierają prawdopodobieństwo przynależności danej obserwacji do danej klasy (nazwy kolumn są ważne),

ostatnia kolumna wskazuję klasę - jest to factor o poziomach takich jak originalna zmienna "y\_tar" z Zadania 1.

g) Funkcja będzie testowana dla problemu regresji i klasyfikacji binarnej na zbiorze danych zawierającym:

- tylko zmienne na skali ilorazowej.

- tylko zmienne na skali porządkowej.

- tylko zmienne na skali nominalnej.

- zmienne na skali mieszanej.

\*\*\*) Dla ambitnych: W zależności od sposoby wykonania Zadania 1, funkcja wyszukuje najbliższych sąsiadów na podstawie struktury k-d tree.

PD8

Zadanie 1:

a) Stwórz funkcję "StopIfNot" przyjmującą nastęujące parametry: "Y", "X", "data", "type", "depth", "minobs", "overfit", "cf".

b) Funkcja powinna sprawdzać czy nauka modelu jest możliwa do wykonania, tj:

- czy "data" jest ramką danych,

- czy wszystkie wymienione zmienne ("Y", "X") istnieją w "data",

- czy zmienna "Y" oraz zmienne "X" w tabeli "data" nie ma braków danych,

- czy "depth" oraz "minobs" są większe od 0,

- czy "type" przyjmuje watrtość "Gini", "Entropy", "SS",

- czy "overfit" przyjmuje watrtość "none" lub "prune",

- czy "cf" jest w przedziale (0,0.5],

- czy możliwe kombinacje parametrów mają sens, np. "type = SS" kiedy "Y" jest faktorem.

c) W przypadku niespełniania któregoś z warunków, funkcja powinna wyświetlić w konsoli, czego dotyczy problem.

d) Funkcja zwraca "TRUE", jeżeli nauka jest możliwa, w przeciwnym wypadku "FALSE".

Zadanie 2:

a) Stwórz funkcję "AssignInitialMeasures" przyjmującą nastęujące parametry: "tree", "Y", "data", "type", "depth".

b) Funkcja powinna na podstawie parametrów wejściowych przypisywać do obiektu "tree" (czyli korzenia) wartości początkowe:

- "depth" = 0.

- w zależności od "type" wartość miary Gini, Entropy, SS dla calej populacji (bo to korzeń).

Zadanie 3:

a) Stwórz funkcję "AssignInfo" przyjmującą nastęujące parametry: "tree", "Y", "X", "data", "type", "depth", "minobs", "overfit", "cf".

b) Funkcja powinna na podstawie parametrów wejściowych przypisywać do obiektu "tree" (jako attrybuty obiektu) wartości owych parametrów.

Zadanie 4:

a) Stwórz funkcję "Tree" przyjmującą nastęujące parametry: "Y", "X", "data", "type", "depth", "minobs", "overfit", "cf".

b) Jest to rozwinięcie funkcji ze slajdu nr 19. Funckja powinna po kolei wywoływać pozostałe funkcje:

- "StopIfNot", jeżeli zwracana wartość to "FALSE" to kończymy działanie całej funkcji (zwracamy obiekt niewidzialny),

- tworzenie obiektu "tree",

- "AssignInitialMeasures",

- "BuildTree",

- "PruneTree", na tę chwilę ta funkcja jest pusta PruneTree<-function(){},

- "AssignInfo".

c) Funkcja powwina zwracać obiekt "tree".

Zadanie 5:

1. Dokonaj integracji opracowanej funkcji "FindBestSplit" z funkcjami "Tree" oraz "BuildTree".

PD9

Zadanie 1:

a) Stwórz funkcję "FindBestSplit" przyjmującą nastęujące parametry: "Y", "X", "data", "parentVal", "type", "minobs".

b) Funkcja powinna zwracać tabelę z wynikami najlepszego możliwego podziału, zawierjącą:

- "infGain" - zysk informacyjny dla podziału,

- "lVal" - miarę niejednorodności dla lewego węzła,

- "rVal" - miarę niejednorodności dla prawego węzła,

- "point" - punkt (lub zbiór punktów dla zmiennych kategorycznych) podzału,

- "Ln" - liczbę obserwacji w lewym węźle,

- "Rn" - liczbę obserwacji w prawym węźle.

c) Funkcja powinna akceptować zmienne ciagłe, porządkowe oraz nominalne. Dwa ostatnie typy reprezentpwane są jako factor.

Zadanie 2:

a) Dokonaj integracji opracowanej funkcji "FindBestSplit" z funkcjami "Tree" oraz "BuildTree".

Zadanie 3:

a) Stwórz funkcję "PredictTree" przyjmującą nastęujące parametry: "tree", "data".

b) Funkcja w pierwszym kroku powinna sprawdzać czy przewidywanie zmiennej celu dla nowego zbioru danych jest możliwe do wykonania,

tj. czy wszystkie zmienne, które budują strukturę drzewa istnieją w nowym zbiorze danych

oraz czy wszystkie kategorie dla zmiennych porządkowych i nominalnych istnieją w nowym zbiorze danych.

c) Funkcja powinna rekurencyjnie przechodzić po strukturze drzewa i wykonywać testy w każdym węźle dla danego atrybutu i punktu podziału.

Przechodząc do finalnego liścia funkcja powinna odczytywać wartość prognozowaną.

d) Funkcja powinna zwracać:

- dla regresji: wektor z wartościami przewidywanymi.

- dla klasyfikacji: nazwaną ramkę danych o rozmiarze "n x k+1", np. dla wersji binarnej o etykietach "P", "N",

tabela wygląda następująco: data.frame( P = c(0.3,0.6), N = c(0.7,0.4), Klasa = c("N","P") ),

tj. pierwsze dwie kolumny zawierają prawdopodobieństwo przynależności danej obserwacji do danej klasy (nazwy kolumn są ważne),

PD11

Zadanie 1:

a) Opracuj uogólnienie funkcji "trainNN" przyjmującej nastęujące parametry: "Yname", "Xnames", "data", "h", "lr", "iter", "seed".

Znaczenie parametrów: Yname - nazwa zmiennej celu z parametru data.

Xnames - nazwy potencjalnych zmiennych objaśniających z parametru data.

data - analizowany zbiór danych.

h - wektor wskazujący liczbę warst ukrytych oraz liczbę neuronów ukrytych,

np. c(3,2) definiuje dwie warstwy ukryte, odpowiednio z trzema oraz dwoma neuronami.

lr - szybkość uczenia.

iter - maksymalna liczba iteracji.

seed - punkt początkowy dla PRNG.

b) Zakładami, że wszystkie zmienne objaśniające są ciągłe oraz rozwiązujemy problem klasyfikacji binarnej.

Pomoce:

https://en.wikipedia.org/wiki/Backpropagation

https://machinelearningmastery.com/the-chain-rule-of-calculus-for-univariate-and-multivariate-functions/

https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

https://ml-cheatsheet.readthedocs.io/en/latest/backpropagation.html

PROJEKT:

# 1) Do rozwiązania mają Państwo 3 problemy:

# a) klasyfikacja binarna,

# b) klasyfikacja wieloklasowa,

# c) regresja.

# Na tę chwilę każdy z Państwa otrzymał jeden zbiór danych (albo do klasyfikacji binarnej albo wieloklasowej).

# Z repozytorium https://archive.ics.uci.edu/ml/index.php wybieraja Państwo, wedle upodobania, dodatkowe dwa zbiory tj.

# do regresji i klasyfikacji (tej której nie dotyczy otrzymany już zbiór danych).

Mamy zbiory do klasyfikacji. Dobieramy jeszcze 2 zbiory danych z UCI MachineLearning

# 2) Rozwiązują Państwo 3 powyższe problemy wykorzystując opracowane algorytmy:

# a) Dla klasyfikacji binarnej: k-najbliższych sąsiadów, drzewa decyzyjne, sieci neuronowe.

# b) Dla klasyfikacji wieloklasowej: k-najbliższych sąsiadów, drzewa decyzyjne, sieci neuronowe.

# c) Dla regresji: k-najbliższych sąsiadów, drzewa decyzyjne, sieci neuronowe.

Analiza dla każdego zbioru danych i wykorzystujemy 3 algorytmy. KNN, drzewa decyzyjne, sieci neuronowe

Iteracje powinny się wykonywać równolegle.

# 3) Porównują Państwo wyniki otrzymane dla własnych algorymtów z wynikami otrzymanymi dla algorytmów pochodzących z innych

# pakietów w R, np: caret, rpart, nnet, neuralnet.

Porównanie do benchmark

# 4) Analiza powinna obejmować:

# a) Wpływ hiper-parametrów na jakość opracowanych modeli.

# np. liczba neuronów ukrytych, liczba najbliższych sąsiadów, głębokość drzewa itd.

# b) Porównanie wyników najlepszych modeli, własnych jak i tych wbudowanych.

Jak zmieniają się hiper-parametry przy zmianach zmiennych

Na osi pionowej mamy błąd, np. MSE, MAE, MAPE, Dla klasyfikacji AUC

Na osi poziomej mamy parametr, który został odłożony dla KNN, im mniejsza wartość k tym dopasowanie będzie lepsze. Pokazujemy jak będzie wyglądała jakość na zbiorze uczącym, czyli błąd mniejszy wraz z większą ilością k.

Wyznaczamy błąd dla zbioru uczącego, może być ona np. stała

I interpretacja tego! Zaznaczenie tego, że algorytm wyłapuje globalne wzorce, a nie lokalne.

Wybieramy ten, który jest najlepszy

# 5) Spakowany katalog z projektem powinien zawierać:

# a) Pliki z danymi źródłowymi,

# b) Plik funkcje.R zawierający definicję wszystkich opracowanych funkcji,

# c) Plik Glowny.R przeprowadzający całą analizę. Uruchomienie tego pliku powinno generować różne obiekty i wykresy, które umieszczone będą w pliku z wynikami.

# - plik dla każdego z 3 problemów: wczytuje funkcje, wczytuje dane i przekształca je, przeprowadza cross-walidację własnych oraz wbudowanych algorytmów, wybiera najlepsze wyniki.

# d) Plik o dowolnym rozszerzeniu np. pdf, docx, pptx, zawierający opis i wyniki z przeprowadzonej analizy.