Matlab podstawy + testowanie dokładności modeli inteligencji obliczeniowej

Podstawy matlaba cz.II

Funkcje

Dotychczas kod zapisany w matlabie stanowił skrypt który pozwalał na określenie kolejności wykonywania operacji. Drugą o wiele bardziej użyteczną formą jest zapis fragmentów programu w postaci funkcji.
Główną korzyścią z wykorzystania funkcji w matlabie jest możliwość wywoływania fragmentów kodu z różnymi zestawami parametrów. Innymi słowy pojęcie funkcji w matlabie jest tożsame z pojęciem funkcji w innych językach programowania. W matlabie funkcje definiowane są poprzez wpisanie na początku m-pliku słowa kluczowego ***function*** po którym w nawiasie kwadratowym występuje lista argumentów zwracanych, przez funkcję po czym pojawia się symbol =, następnie nazwa funkcji i w nawiasach typu () lista argumentów wejściowych, przykład:

```
function  [a,b] = policz(c,d,e,f)
a = c+d+e+f;
b = c.*d.*e.*f;
```

Najprostszą metodą tworzenia funkcji jest na wstępie stworzenie skryptu, a następnie wpisanie na początku pliku słowa kluczowego *function* jak powyżej. Tworząc jednak funkcję niezbędne jest spełnienie kilku warunków:
- nazwa pliku w którym zapisana jest funkcja musi być zgodna z nazwą funkcji. Jest tak gdyż Matlab wykorzystuje nazwę pliku do zidentyfikowania nazwy funkcji. Jeśli warunek ten nie będzie spełniony może to doprowadzić powstawania różnego rodzaju błędów.
- nazwy plików nie mogą zawierać symboli specjalnych typu -,+,^, itp. Możliwe jest natomiast wykorzystanie symbolu podkreślenia '_'

Funkcje w matlabie posiadają kilka unikatowych cech
- można je wywoływać z dowolną liczbą parametrów, innymi słowy jeśli funkcja jest poprawnie napisana to powyższą funkcję teoretycznie można wywołać jako [x,y] = policz(z)
- możliwy jest odczyty dowolnej liczby parametrów zwracanych przez funkcję np. x = policz(a,b,c,d). Wówczas wartość ***b*** zwracana przez funkcję jest ignorowana.
- funkcje w odróżnieniu od skryptów wykonują się znacznie szybciej,

Tablice komórkowe

Tablice komórkowe są specjalną postacią tablicy pozwalającą na przechowywanie elementów dowolnego typu. Tradycyjne tablice w matlabie przechowuje zmienne tego samego typu, powoduje to pewne komplikacje np. jeśli chcielibyśmy przechowywać stringi różnej długości np. polecenie:
Z = ['Ala'; 'ma'; 'kota']
Zgłosi błąd o nie spójności wymiarów tablicy.

Dlatego w takich sytuacjach niezbędne jest skorzystanie z tablic komórkowych. Tablice komórkowe tworzy się poprzez zastąpienie symbolu [ ] symbolem { }. Tablice tego typu posiadają kilka istotnych wad np. nie można na nich wykonywać operacji arytmetycznych typu +,-,*,/ jak miało to miejsce w przypadku tradycyjnych tabeli.
Przykładem wykorzystanie tablic komórkowych może być
Z = {Ala'; 'ma'; 'kota'}
Wówczas interpreter matlaba nie zgłosi błędu i utworze *cellarray* o wymiarach size(Z) => 3x1. Chcąc odczytać daną komórkę należy zapisać jako c = Z{2}, wówczas *c* przyjmie wartość 'ma'.
W podobny sposób można też zapisać do tablicy komórkowej macierze o różnym rozmiarze np.:
C{1} = [1 2 3 4 5];
C{2} = [6;7;6];
C{3} = [1 2 3;4 5 6]
Z tablic komórkowych istnieje też bezpośredni dostęp do poszczególnych elementów tablic składowych np. zapis:
C{3}(2,3) zwróci liczbę 6 gdyż w tablicy o indeksie 3 odczytywana jest komórka o wsp. (2,3)
Podobnie
C{2}(2) zwróci liczbę 7

Inteligencja obliczeniowa

Inteligencja obliczeniowa - Jest to dziedzina nauki zajmująca się rozwiązywaniem za pomocą obliczeń problemów, które nie są efektywnie algorytmizowalne.

Cechy inteligentnego systemu:

- zdolność do przyswajania nowej wiedzy;
- samoadaptacja;
- akceptacja danych niepełnych i nie w pełni spójnych logicznie;
- kreatywność.

W skład metod inteligencji obliczeniowej wchodzą:

[1]

Co oznacza „nie są efektywnie algorytmizowane”?
Sformułowanie to oznacza iż nie da się w prosty sposób napisać algorytmu postępowania – ciągu instrukcji które doprowadzą do rozwiązania danego problemu.

Przykładem problemów rozwiązywanych przez inteligencję obliczeniową są:

- Strategie stosowane w grach komputerowych np. szachy
- Problem rozpoznawania obrazów np. identyfikacja osób na podstawie obrazu z kamery
- Autonomiczne sterowanie robotem
- Itp.

Ze względu na swoją naturę problemy rozwiązywane przez metody inteligencji obliczeniowej możemy podzielić na

- Uczenie nadzorowane – system podczas uczenia ma dostęp do informacji o wynikach które powinien uzyskać – np. nauka rozpoznawania płci na podstawie wzrostu i rozmiaru stopy, gdy zbiorem uczącym są studenci danej grupy, a system ma działać dla wszystkich studentów Politechniki
  W uczeniu nadzorowanym wyróżniamy:
  - Problemy klasyfikacyjne – gdy informacja którą system się uczy jest typu symbolicznego np. kobieta, mężczyzna, dziecko
  - Problemy regresyjne – gdy informacja którą uczy się system jest liczbą porządkową (rzeczywistą) np. wartości indeksów giełdowych

- Uczenie nienadzorowane – grupowanie danych/analiza skupień gdy podczas uczenia systemu nie mamy informacji o tym co system ma się nauczyć.

Z uwagi na bogactwo różnych algorytmów wchodzących w skład problemów inteligencji obliczeniowej niezmiernie istotna jest metoda testowania umożliwiająca dobór możliwie najlepszego algorytmu do postawionego zadania.
Wyróżniamy tutaj test typu:

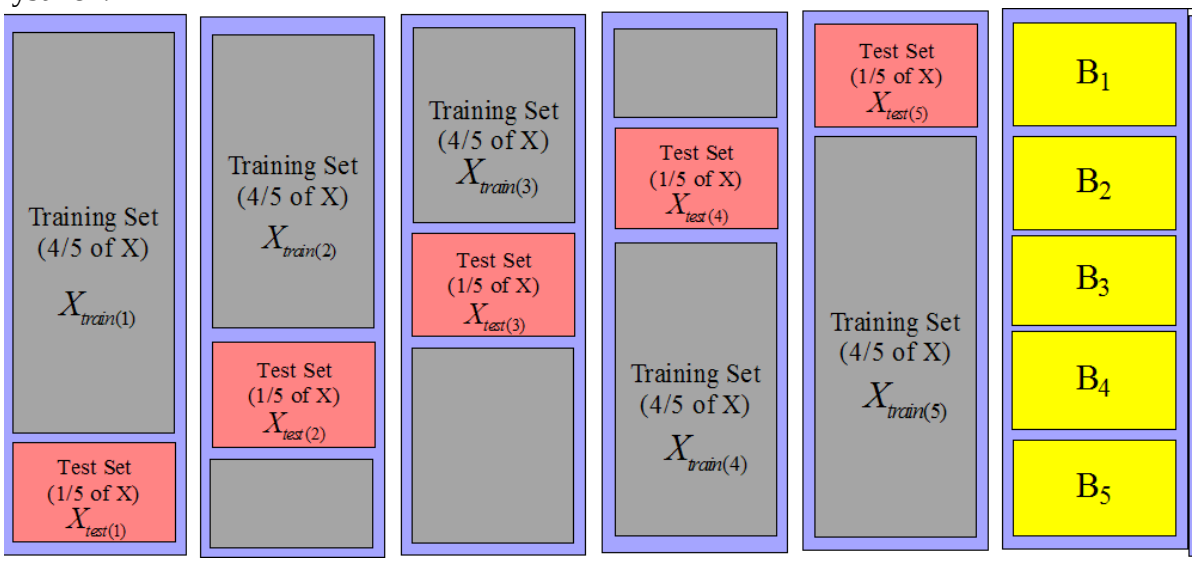
- Trening/test
- Test krzyżowy (corss validation)
- Test jeden pozostaw

[1] Źródło: http://dydaktyka.polsl.pl/KWMIMKM/mio.aspx

- Bootstrap

Test typu trening test polega na podziale zbioru danych treningowych na dwie niezależne części, z których jedna używana jest do nauczenia systemu, a druga, gdy system jest nauczony do weryfikacji jak dobrze system się nauczył

Test krzyżowy polega na wielokrotnym podziale zbioru uczącego na trening test (n-krotnie) i na tej podstawie można policzyć średnią dokładność z *n* prób. Dzięki temu mamy dużo dokładniejsze oszacowanie jak dobrze nauczył się nasz system. Test krzyżowy ma dodatkowo taką właściwość iż zbiór danych dzielony jest na *n* równych części, wówczas system uczony jest na *n*-1 częściach, a testowany na nie użytej do treningu części. Przedstawia to poniższy rysunek:



Zadanie w matlabie:

1) Zbuduj funkcję w matlabie która dokona podziału zbioru danych na część treningową i testową, tak iż wywołując funkcję będzie można podać ile procent zbioru danych przeznaczone będzie na część treningową (pozostała część będzie stanowiła część testową)
   Wywołanie: [trening,test]=podziel(zbiór,procent)
   Gdzie
   - trening – struktura o dwóch polach X i Y
     a) trening.X – zbiór danych używany do treningu
     b) trening.Y – zbiór etykiet używany do treningu
   - test – struktura o dwóch polach X i Y
     a) test.X – zbiór danych używany do testowania
     b) test.Y– zbiór etykiet używany do testowania poprawności klasyfikatora
   - zbiór – całkowity dostępny zbiór danych
   - procent – liczba z zakresu 0-1 określająca ile proc. zbioru danych ma stanowić zbiór treningowy
2) Napisz prosty klasyfikator wg. algorytmu poniżej
   Algorytm powinien być zapisany jako dwie funkcje p = ucz_klasyfikator(dane)
   Gdzie:
   - Dane – zbiór danych treningowych składający się z pól
     - X to zbiór danych treningowych
     - Y to zbiór etykiet dla poszczególnych wektorów zbioru treningowego

- p – zbiór parametrów nauczonego klasyfikatora – w tym przypadku najczęściej występująca etykieta

Oraz z funkcji daneW = testuj_klasyfikator(dane,p)

Gdzie:

- Dane – zbiór danych testowych – struktura o polach:
  - X – zbiór danych testowych
  - Y – zbiór etykiet (Y może być puste Y=[])
- p – Zbiór wzorców uzyskanych w wyniku uczenia
- DaneW – zbiór danych wynikowych – struktura o polach:
  - X – zbiór danych testowych (daneW.X = dane.X)
  - Y – zbiór etykiet przewidziane wartości etykiet

Uczenie:

a) Znajdź najczęściej występującą klasę i przypisz wszystkie wektory do danej klasy
b) Wykorzystaj polecienie sum(dane.Y == ?)

Testowanie:

a) Przypisz każdemu wektorowi etykietę najczęściej występującej klasy daneW.X = dane.X; dane.Y(1:size(daneW.X,1)) = C;

Przydatne funkcje:

M = mean(X) – liczy osobno dla każdej kolumny wartość średnia.

Id = find(warunek) – znajduje indeksy wktorów które spełniają określony warunek

U = unique(X) – funkcja zwraca listę unikatowych wartości występujących w X (przydatne do zbadania liczby klas poprzez c = unique(y)

3) Zbuduj funkcję obliczającą dokładność klasyfikatora jako:
   Acc = dokladnosc(dane,daneW)
   Funkcja powinna policzyć dokładność jako średnią liczbę przypadków dla których dane. Y==daneW.Y
   Gdzie:
   - dane.Y – rzeczywiste (prawdziwe) wyjścia z systemu
   - daneW.Y – wyjścia przewidziane przez klasyfikator

4) Przetestuj zbudowany algorytm wykorzystując funkcję *podziel* tak, iż zbiór danych treningowych (trening) powinien zostać wykorzystany do uczenia klasyfikatora z punktu 2, natomiast zbiór testowy (test) powinien zostać wykorzystany do sprawdzenia dokładności klasyfikatora. Dokładność uzyskanych wyników sprawdź korzystając z funkcji dokładność z punktu 3

5) Zbuduj funkcję realizującą test krzyżowy, funkcja powinna mieć postać:
   zbiory=testrzyzowy(zbior,n)
   gdzie:
   - zbiory – tablica komórkowa o rozmiarze {n 2}zawierająca w każdym i'tym wierszu osobno zbiory{i,1} – zbiór treningowy zbiory{i,2} – zbiór testowy
   - zbiór – zbiór danych użyty do uczenia
   - n – liczba podziałów na część treningową i testową.