Politechnika Wrocławska

Uczenie maszynowe

Zastosowane programu MATLAB w praktyce inżynierskiej

Wojciech Myszka

14 stycznia 2025

HR EXCELLENCE IN RESEARCH

Uczenie maszynowe

Politechnika Wrocławska

Dane

1. Zazwyczaj mamy z zestawem danych (wielowymiarowych)
   1.1 pewne składowe traktujemy jako predyktory (zmienne niezależne)
   1.2 inne (zmienne „zależne") definiują pewną cechę (na przykład „dobry"—„zły"; klasa „A"—klasa „B")
2. Uczenie składa się z kilku etapów
   - uczenie
   - ocena uczenia (testowanie)
   - użycie klasyfikatora

   klasyfikatora używać będziemy na danych, dla których cecha nie jest zdefiniowana — to model ma zaklasyfikować dostarczony zestaw „wartości niezależnych".

Politechnika Wrocławska

Aplikacja

1. Matlab dostarcza aplikację pozwalającą na (dosyć) wygodne przeprowadzenie procesu uczenia i testowania.
2. Aplikacja nazywa się **Classification Learner App**
3. Najpierw trzeba posiadane dane podzielić (najlepiej losowo) na dwie kategorie:
   - dane uczące
   - dane testujące
4. Najprostsza sytuacja podziału na dwie klasy może być zrealizowana za pomocą funkcji `c = cvpartition(Y,'Holdout',p)`
   - `Y` to kolumna zawierająca cechę; dane trzeba tak podzielić na klasy, żeby w obu rozkład cechy był zbliżony
   - `p` to frakcja ($0 < p < 1$) danych pozostawionych do testowania.
5. Sam podział zbioru danych na uczące/testowe dokonuje się za pomocą funkcji `training()`

(Bardzo prosty) przykład I

1. Mamy dane (o wartościach całkowitych) — dla uproszczenia
2. Wszystkie wartości mniejsze od 3 należą do klasy “A”
3. Pozostałe — do klasy “B”

```
N = 100;
```

liczba generowanych wartości

```
wartosc = randi(11,N,1);
klasa(wartosc>2.5)  = "B";
klasa(wartosc<=2.5) = "A";
```

korzystamy z funkcji table, żeby stworzyć jednolitą tabelę

```
dane = table(wartosc, klasa');
```

pierwsza kolumna tabeli nazywa się `wartosc`, a druga `klasa`.

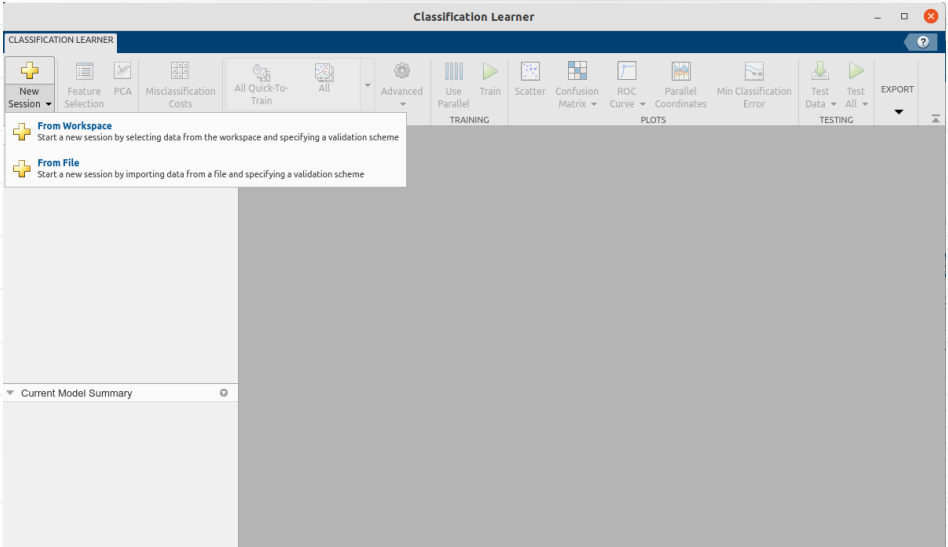
Politechnika Wrocławska

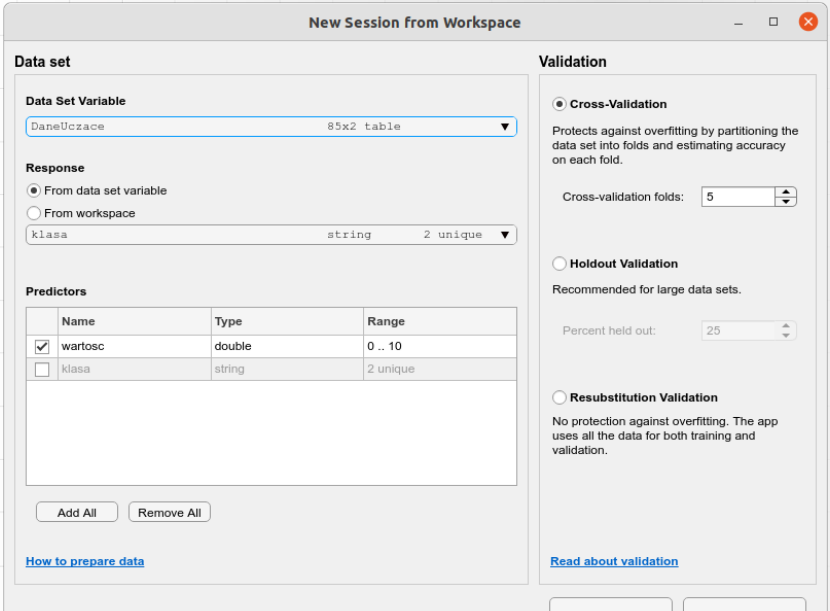
(Bardzo prosty) przykład II

```
podzial = cvpartition(dane.klasa, 'Holdout', 0.15)
indeksy = training(podzial);
DaneUczace = dane(indeksy);
DaneTestowe = dane(~indeksy);
```

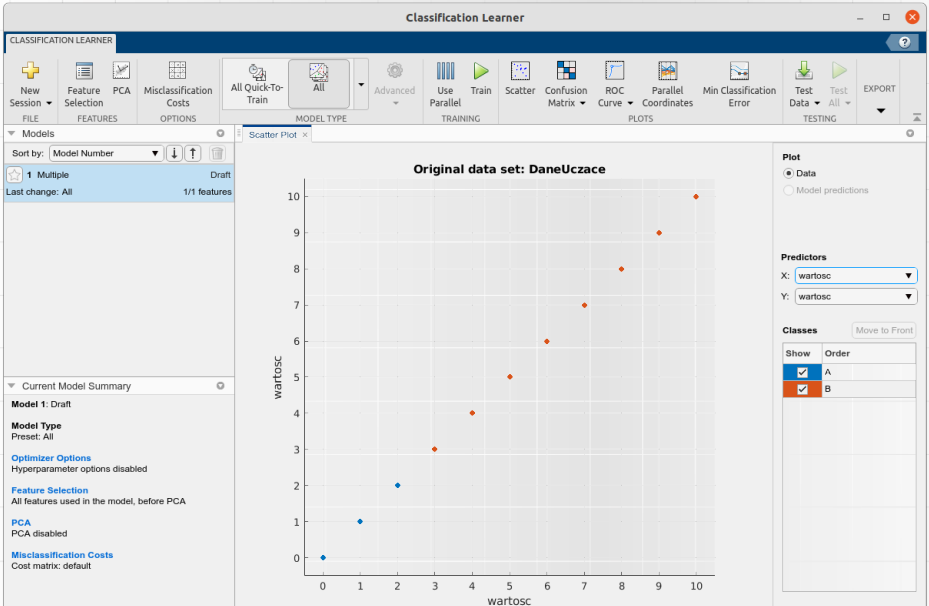
Uruchamiamy aplikację

Wybieramy import danych z przestrzeni roboczej




New Session from Workspace
Data set
Data Set Variable
DaneUczace 85x2 table
Response
From data set variable
From workspace
klasa string 2 unique
Predictors
Name
Type
Range
wartosc
double
0 .. 10
klasa
string
2 unique
Add All
Remove All
How to prepare data
Validation
Cross-Validation
Protects against overfitting by partitioning the data set into folds and estimating accuracy on each fold.
Cross-validation folds:
5
Holdout Validation
Recommended for large data sets.
Percent held out:
25
Resubstitution Validation
No protection against overfitting. The app uses all the data for both training and validation.
Read about validation

Naciskamy Start Session



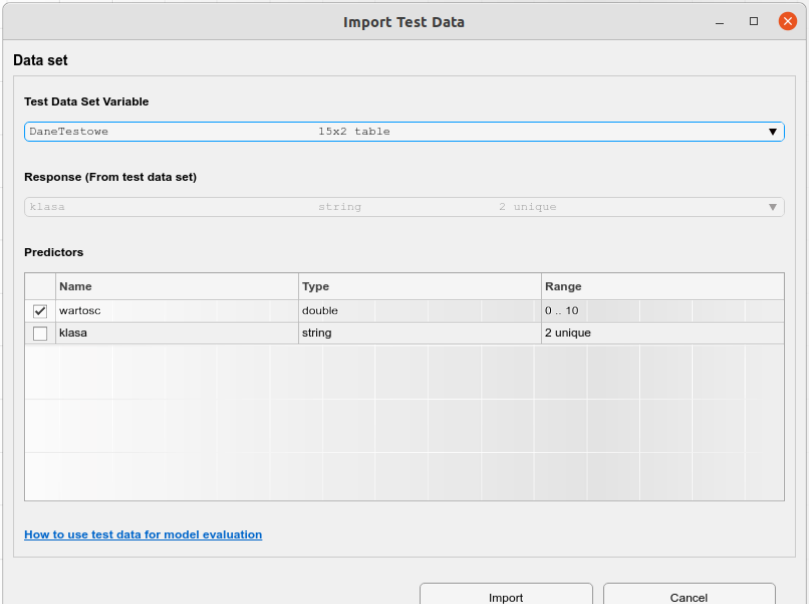
Wybór rodzaju klasyfikatora

- Wybieramy wszystkie
- Naciskamy „Train”
- W okienku (lewy , górny róg) pojawiają się rezultaty uczenia wraz z oszacowaną jakością.
- W tym przypadku więcej niż jeden algorytm klasyfikujący ma jakość 100%; najgorszy niecałe 73%.
- Można ustawić je w kolejności dokładności (*Accuracy (validation)*)

Przechodzimy do danych testowych

Klikamy w **Testdata** i wybieramy zmienną testową, klikamy Import i naciskamy Test; testować można wszystkie, lub wybrane klasyfikatory

Politechnika Wrocławska



Wyniki testu I

- Problem był tak prosty, że rozpoznawanie dla kilku modeli jest dokładne.
- Kolejny krok to eksport wytrenowanego modelu; można to zrobić na kilka sposobów
- Klikamy w ostatnią po prawej stronie zakładkę i dostajemy
  - *Generate function*
  - *Export Model*
    - *Export Model*
    - *Export compact model*
- Compact Model jest najprostszy — w przestrzeni roboczej zapisywany jest „wytrenowany" model (*pretrained*) z którego można korzystać.
- po wybraniu tej opcji system pyta nas o nazwę (ja podałem `prostymodel`) i tworzy strukturę:

Wyniki testu II

```
prostymodel =
  struct with fields:
              predictFcn: [function_handle]
        RequiredVariables: {'wartosc'}
       ClassificationTree: [1×1 ClassificationTree]
                    About: 'This struct is a trained model exported
                           from Classification Learner R2021a.'
              HowToPredict: 'To make predictions on a new table, T,
                           yfit = c.predictFcn(T)
                           replacing 'c' with the name of the
                           variable that is this struct, e.g.
                           'trainedModel'. The table, T, must conta
                           the variables returned by:
                           c.RequiredVariables
```

Wyniki testu III

```
Variable formats (e.g. matrix/vector,
datatype) must match the original traini
Additional variables are ignored.
For more information, see How to predict
an exported model.'
```

- utworzymy dane testujące:

```
t = 0:0.5:10;
T = table(t','VariableNames',{'wartosc'});
[T, prostymodel.predictFcn(T)]
ans =
  21×2 table
    wartosc    Var2
    -------    -----
```

Wyniki testu IV

0	{'A'}
0.5	{'A'}
1	{'A'}
1.5	{'A'}
2	{'A'}
2.5	{'B'}
3	{'B'}
3.5	{'B'}
4	{'B'}
4.5	{'B'}
5	{'B'}
5.5	{'B'}
6	{'B'}
6.5	{'B'}

Wyniki testu V

7	{'B'}
7.5	{'B'}
8	{'B'}
8.5	{'B'}
9	{'B'}
9.5	{'B'}
10	{'B'}

Dane „zaburzone” I

- Teraz wszystkie wartości < 2 to klasa `"A"`, natomiast > 3 to klasa `"B"`
- „dwójki” i „trójki” będą losowane, z jednakowym prawdopodobieństwem i zaliczane albo do `"A"` albo do `"B"`.
- Powtarzamy procedurę
- Ponieważ dane nie są jednoznaczne nie ugaje się uzyskać 100% dokładności
  - Regresja logistyczna, Naive Bayes, Linear SVM, Quadratic SVM oraz Fine KNN uzyskały dokładność (na danych uczących) 89,4%; najgorzej wypadło Coarse KNN (74,1%)
- Dane testowe wygenerowały się tak nieszczęśliwie, że nie było wśród nich „trójek” i tylko jedna „dwójka”; w związku z tym udało się uzyskać 100% dokładność dla wielu klasyfikatorów:
  - Fine KNN, Ensemble,...

Dane „zaburzone” II

- Natomiast rezultaty otrzymane z tego klasyfikatora są lekko zadziwiające:

wartosc	klasa	Var3
2	"B"	{'B'}
2	"A"	{'B'}
2	"B"	{'B'}
2	"B"	{'B'}
2	"B"	{'B'}
2	"A"	{'B'}
2	"A"	{'B'}
2	"B"	{'B'}
3	"A"	{'A'}
3	"B"	{'A'}

Dane „zaburzone” III

```
3       "A"     {'A'}
3       "A"     {'A'}
3       "B"     {'A'}
3       "A"     {'A'}
3       "A"     {'A'}
3       "B"     {'A'}
3       "B"     {'A'}
```

wszystkie „dwójki” klasyfikowane są jako `"B"`, a trójki jako `"A"`.

Uwagi

Warto prześledzić dokumentację — jest tam sporo przykładów...

Politechnika Wrocławska

Aplikacja I

1. Przygotowana aplikacja oprócz eksportu *wytrenowanego modelu* pozwala na eksport kodu, który może służyć do:
   - zrozumienia procesu uczenia,
   - przygotowywanie własnego kodu.
2. Wybieramy sekcję **Export** i klikamy w **Generate Function**.
3. Tworzony jest kod (ze sporą ilością komentarzy).
4. Aby kod „działał" należy uruchomić go, dostarczając **oryginalnych** danych; można też dostarczyć nowego zestawu danych.

   ```
   [trainedModel,validationAccuracy] = trainClassifier(dane)
   ```

   wynikiem będzie struktura danych `trainedModel` oraz informacja na temat jego jakości `validationAccuracy`.

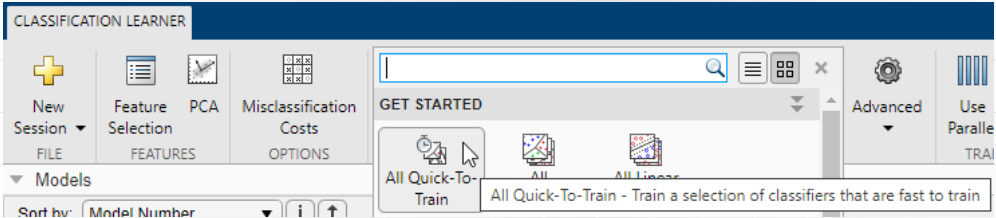
Aplikacja II

5. Funkcja `trainedModel.HowToPredict` zwraca informację jak należy z utworzonego klasyfikatora korzystać.
6. Polecenie `yfit = trainedModel.predictFcn(T)` dokona klasyfikacji.

   Trzeba jednak pamiętać o tym, że zmienna/tablica `T` musi mieć identyczną strukturę jak dane użyte do uczenia; nawiązując do przykładu:

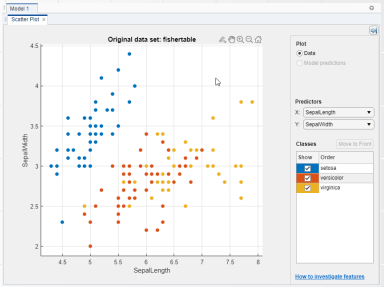
   - tablica z danymi (do uczenia) zawierała dwie kolumny o nazwach `wartosc` i `klasa`,
   - dane do klasyfikacji musi to być **tablica** nie wektor, a kolumna **musi** nazywać się `wartosc`.

Parametry Aplikacji I



1. New Session
2. Feature Selection (bardzo ważne w przypadku danych wielowymiarowych)

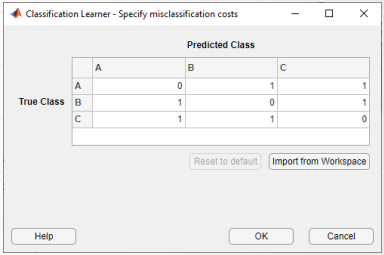
Parametry Aplikacji II



Widać wyraźnie, że zmienne `SepalLength` and `SepalWidth` (określające wymiary płatka) pozwalają na rozróżnienie dwu gatunków; warto sprawdzić, czy inne parametry pozwolą na dokonanie innych rozróżnień

Parametry Aplikacji III

3. PCA (*principal component analysis*) pozwala na uruchomienie analizy, która pozwala automatycznie wybrać parametry, odpowiedzialne za 95% wariancji (zbyt wysoka wartość parametru grozi nadmiernym dopasowaniem, zbyt mała — usuwa użyteczne zmienne).
4. Missclasificcation Costs pozwala zadeklarować „koszty" związane ze złymi decyzjami.



Parametry Aplikacji IV

5. **Use parallel**
   - trzeba mieć zainstalowany Parallel Toolbox;
   - w przypadku prostych zadań nie opłaca się uruchamiać – „wystartowanie" obliczeń równoległych kosztuje dużo...;
   - W przypadku dużych zadań (i silnego sprzętu) może przyspieszyć obliczenia.

Politechnika Wrocławska

Klasyfikatory I

1. Drzewa decyzyjne
2. Analiza dyskryminacyjna (zbliżona do analizy wariancji)
3. Regresja logistyczna (taki rodzaj regresji, w której zmienna zależna przyjmuje dwie wartości)
4. Naive Bayes (Twierdzenie Bayesa; zakładamy niezależność predyktorów)
5. Support Vector Machnines (maszyna wektorów nośnych) — generuje hiperpowierzchnę rozdzielającą cechy w wielowymiarowej przestrzeni cech
6. Nearest Neighbour (k Nearest Neughbours) — na podstawie serii uczącej wylicza się „średnią" odległość klasyfikowanego obiektu od k obiektów serii uczącej)
7. Kernel Approximation Classifier: metoda nieliniowa, najlepsze zastosowanie przy dużej liczbie danych
8. Ensemble Classifier (klasyfikatory zespołowe)

Deep Learning Toolbox

Uwagi

1. Chyba nie warto instalować
2. Warto skorzystać on-line (Classify Image Using Pretrained Network)
3. Jest sporo przykładów, które można otwierać jako *live script*-y i bawić się korzystając z realnych danych na swoim dysku sieciowym.

Politechnika Wrocławska

Możliwości I

Bardzo duże:

1. Rozpoznawanie obrazów
2. Rozpoznawanie szeregów czasowych
3. Wykrywanie obiektów
4. Segmentacja obiektów

Politechnika Wrocławska

Możliwości II

Pretrained networks I

Network	Dept	Size	Parameters (Millions)	Image Input Size
squeezenet	18	5.2 MB	1.24	227-by-227
googlenet	22	27 MB	7.0	224-by-224
inceptionv3	48	89 MB	23.9	299-by-299
densenet201	201	77 MB	20.0	224-by-224
mobilenetv2	53	13 MB	3.5	224-by-224
resnet18	18	44 MB	11.7	224-by-224

Pretrained networks II

`resnet50`	50	96 MB	25.6	224-by-224
`resnet101`	101	167 MB	44.6	224-by-224
`xception`	71	85 MB	22.9	299-by-299
`inceptionresnetv2`	164	209 MB	55.9	299-by-299
`shufflenet`	50	5.4 MB	1.4	224-by-224
`nasnetmobile`	*	20 MB	5.3	224-by-224
`nasnetlarge`	*	332 MB	88.9	331-by-331

Pretrained networks III

darknet19	19	78 MB	20.8	256-by-256
darknet53	53	155 MB	41.6	256-by-256
efficientnetb0	82	20 MB	5.3	224-by-224
alexnet	8	227 MB	61.0	227-by-227
vgg16	16	515 MB	138	224-by-224
vgg19	19	535 MB	144	224-by-224

Politechnika Wrocławska

Deep Learning HDL Toolbox

- Funkcjonalności Deep Learning na komputerach wyposażonych w specjalistyczne układy obliczeniowe