# Systemy uczące się - laboratorium

Filip Drapejkowski - nr indeksu: 2034050

# Ćwiczenie 1. Klasyfikator oparty na twierdzeniu Bayesa przy naiwnym założeniu o wzajemnej niezależności atrybutów. ¶

## I: podstawowe, techniczne operacje:

In [1]:

```
import csv
import random
import math
def loadCsv(filename):
    lines = csv.reader(open(filename, "rb"))
    dataset = list(lines)
    for i in range(len(dataset)):
        dataset[i] = [float(x) for x in dataset[i]]
    return dataset
```

Podział na zbiór uczący i testowy:

In [2]:

```
def splitDataset(dataset, splitRatio):
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    copy = list(dataset)
    while len(trainSet) < trainSize:
        index = random.randrange(len(copy))
        trainSet.append(copy.pop(index))
    return [trainSet, copy]
```

Przykładowe wyjście funkcji: {0: [[2, 21, 0]], 1: [[1, 20, 1], [3, 22, 1]]}

In [3]:

```
def separateClasses(dataset):
    separated = {}
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separated):
            separated[vector[-1]] = []
        separated[vector[-1]].append(vector)
    return separated
```

Zbiór danych - legenda

1. Number of times pregnant
2. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
3. Diastolic blood pressure (mm Hg)
4. Triceps skin fold thickness (mm)
5. 2-Hour serum insulin (mu U/ml)
6. Body mass index (weight in kg/(height in m)^2)
7. Diabetes pedigree function
8. Age (years)
9. Class variable (0 or 1) - informacja, czy pacjent w ciągu 5 lat od dokonania pomiarów choruje na cukrzycę

# II: definicje funkcji

$$s = \sqrt{\frac{\sum(x - \bar{x})^2}{n - 1}}$$

In [4]:

```
def stdev(numbers):
    avg = mean(numbers)
    variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)
```

In [5]:

```
def mean(numbers):
    return sum(numbers)/float(len(numbers))

def summarize(dataset):
    summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(*datas
et)]
    del summaries[-1]
    return summaries

def summarizeClasses(dataset):
    separated = separateClasses(dataset)
    summaries = {}
    for classValue, instances in separated.iteritems():
        summaries[classValue] = summarize(instances)
    return summaries
```

## Normal Probability Density Function

$$F(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

In [6]:

```
def calculateProbability(x, mean, stdev):
    exponent = math.exp(-(math.pow(x-mean,2)/(2*math.pow(stdev,2))))
    return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent
```

$$p(C_k|\mathbf{x}) = \frac{p(C_k)\, p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

$$\hat{y} = \underset{k \in \{1,\ldots,K\}}{\operatorname{argmax}}\ p(C_k) \prod_{i=1}^{n} p(x_i|C_k).$$

In [7]:

```
def calculateClassProbabilities(summaries, inputVector):
    probabilities = {}
    for classValue, classSummaries in summaries.iteritems():
        probabilities[classValue] = 1
        for i in range(len(classSummaries)):
            mean, stdev = classSummaries[i]
            x = inputVector[i]
            probabilities[classValue] *= calculateProbability(x, mean, stdev)
    return probabilities
```

In [8]:

```
def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)
    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.iteritems():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

def getPredictions(summaries, testSet):
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])
        predictions.append(result)
    return predictions

def getAccuracy(testSet, predictions):
    correct = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
```

# III: Uczenie i testowanie

In [9]:

```
filename = 'pima-indians-diabetes.data.csv'
splitRatio = 0.67
dataset = loadCsv(filename)
trainingSet, testSet = splitDataset(dataset, splitRatio)
print('Split {0} rows into train={1} and test={2} rows').format(len(dataset), le
n(trainingSet), len(testSet))
# prepare model
summaries = summarizeClasses(trainingSet)
# test model
predictions = getPredictions(summaries, testSet)
accuracy = getAccuracy(testSet, predictions)
print('Accuracy: {0}%').format(accuracy)
```

```
Split 768 rows into train=514 and test=254 rows
Accuracy: 71.6535433071%
```

Przykład:

In [10]:

```
print predict(summaries, [8,183,64,0,0,23.3,0.672,32,1])
```

```
1.0
```

# IV: Macierz błędu

| | | klasa rzeczywista | |
|---|---|---|---|
| | | **pozytywna** | **negatywna** |
| **klasa predykowana** | **pozytywna** | prawdziwie pozytywna (TP) | fałszywie pozytywna (FP) |
| | **negatywna** | fałszywie negatywna (FN) | prawdziwie negatywna (TN) |

Miary:

- prawdziwie pozytywna (*true positive* TP)
- prawdziwie negatywna (*true negative* TN)
- fałszywie pozytywna (*false positive* FP), błąd I typu
- fałszywie negatywna (*false negative* FN), błąd II typu
- czułość (*sensitivity*) lub odsetek prawdziwie pozytywnych (*true positive rate* TPR)
  $$TPR = TP/P = TP/(TP + FN)$$
- specyficzność (*specificity* SPC) lub odsetek prawdziwie negatywnych (*True Negative Rate* TNR)
  $$TNR = TN/N = TN/(FP + TN)$$
- precyzja (*precision*)
  $$prezycja = TP/(TP + FP)$$
- dokładność (*accuracy* ACC)
  $$ACC = (TP + TN)/(P + N)$$

F1 - średnia harmoniczna precyzji i czułości(recall) (poniżej dwie metody obliczania tej samej wartości)

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

$$F_\beta = \frac{(1+\beta^2) \cdot \mathrm{true\ positive}}{(1+\beta^2) \cdot \mathrm{true\ positive} + \beta^2 \cdot \mathrm{false\ negative} + \mathrm{false\ positive}}.$$

In [11]:

```
class DictTable(dict):
    # Overridden dict class which takes a dict in the form {'a': 2, 'b': 3},
    # and renders an HTML Table in IPython Notebook.
    def _repr_html_(self):
        html = ["<table width=100%>"]
        for key, value in self.iteritems():
            html.append("<tr>")
            html.append("<td>{0}</td>".format(key))
            html.append("<td>{0}</td>".format(value))
            html.append("</tr>")
        html.append("</table>")
        return ''.join(html)
def getTFCounts(testSet, predictions):
    FP=0
    TP=0
    FN=0
    TN=0
    P = 0
    N = 0
    for i in range(len(testSet)):
        if testSet[i][-1] == predictions[i] == 1:
            TP += 1
            P += 1
        elif testSet[i][-1] == predictions[i] == 0:
            TN += 1
            N += 1
        elif testSet[i][-1] != predictions[i] == 1:
            FP += 1
            P += 1
        elif testSet[i][-1] != predictions[i] == 0:
            FN += 1
            N += 1
    return[P,N,TP,TN,FP,FN]
def getMetrics(P,N,TP,TN,FP,FN):
    sensivity = float(TP) / (TP + FN)
    specificity = float(TN) / N
    precision = float(TP) / (TP + FP)
    f1 = 2 * (precision * sensivity) / (precision + sensivity)
    return(sensivity, specificity, precision, f1)
```

In [12]:

```
counts = getTFCounts(testSet, predictions)
metrics = getMetrics(*counts)
print counts
print metrics
m = [('Czulosc',metrics[0]),('Specyficznosc',metrics[1]),
('Precyzja',metrics[2]), (u'f1',metrics[3])]
m = dict(m)
DictTable(m)
```

```
[101, 153, 55, 127, 46, 26]
(0.6790123456790124, 0.8300653594771242, 0.5445544554455446, 0.6043
956043956045)
```

Out[12]:

| | |
|---|---|
| Specyficznosc | 0.830065359477 |
| Precyzja | 0.544554455446 |
| Czulosc | 0.679012345679 |
| f1 | 0.604395604396 |

# V: Walidacja krzyżowa

In [13]:

```
import numpy as np
from sklearn.cross_validation import KFold
n_folds=10
kf = KFold(len(dataset), n_folds=n_folds)
results=[]
for train_index, test_index in kf:
    dataset = np.asarray(dataset)
    trainingSet, testSet = dataset[train_index], dataset[test_index]
    print('Podział {0} wierszy na ciąg_uczący o {1} wierszach i ciąg_testowy o
 {2} wierszach').format(len(dataset), len(trainingSet), len(testSet))
    summaries = summarizeClasses(trainingSet)
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    results.append(accuracy)
    print('Dokładność: {0}%').format(accuracy)
print results
print('Średnia dokładność z walidacji krzyżowej o {0} złożeniach: {1}').format(n
_folds,np.mean(results))
```

```
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 70.1298701299%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 79.2207792208%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 71.4285714286%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 66.2337662338%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 74.025974026%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 75.3246753247%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 75.3246753247%
Podział 768 wierszy na ciąg_uczący o 691 wierszach i ciąg_testowy o
 77 wierszach
Dokładność: 80.5194805195%
Podział 768 wierszy na ciąg_uczący o 692 wierszach i ciąg_testowy o
 76 wierszach
Dokładność: 75.0%
Podział 768 wierszy na ciąg_uczący o 692 wierszach i ciąg_testowy o
 76 wierszach
Dokładność: 76.3157894737%
[70.12987012987013, 79.22077922077922, 71.42857142857143, 66.233766
23376623, 74.02597402597402, 75.32467532467533, 75.32467532467533,
 80.51948051948052, 75.0, 76.31578947368422]
Średnia dokładność z walidacji krzyżowej o 10 złożeniach: 74.352358
1681
```

In [14]:

```
import numpy as np
from sklearn.cross_validation import KFold
n_folds=3
kf = KFold(len(dataset), n_folds=n_folds)
results=[]
for train_index, test_index in kf:
    dataset = np.asarray(dataset)
    trainingSet, testSet = dataset[train_index], dataset[test_index]
    print('Podział {0} wierszy na ciąg_uczący o {1} wierszach i ciąg_testowy o
 {2} wierszach').format(len(dataset), len(trainingSet), len(testSet))
    summaries = summarizeClasses(trainingSet)
    predictions = getPredictions(summaries, testSet)
    accuracy = getAccuracy(testSet, predictions)
    results.append(accuracy)
    print('Dokładność: {0}%').format(accuracy)
print results
print('Średnia dokładność z walidacji krzyżowej o {0} złożeniach: {1}').format(n
_folds,np.mean(results))
```

```
Podział 768 wierszy na ciąg_uczący o 512 wierszach i ciąg_testowy o
 256 wierszach
Dokładność: 73.828125%
Podział 768 wierszy na ciąg_uczący o 512 wierszach i ciąg_testowy o
 256 wierszach
Dokładność: 68.359375%
Podział 768 wierszy na ciąg_uczący o 512 wierszach i ciąg_testowy o
 256 wierszach
Dokładność: 76.953125%
[73.828125, 68.359375, 76.953125]
Średnia dokładność z walidacji krzyżowej o 3 złożeniach: 73.046875
```

Przetestowawszy ilości złożeń takie jak: 2,3,5,10,50,100,200 stwierdzam, że dokładność oscylowała pomiędzy wartościami 73 a 75 bez monotonicznej zależności.

Procedura walidacji krzyżowej ma na celu głównie przeciwdziałanie przeuczaniu (overfitting) podczas selekcji modelu (unikanie błędu 3ciego rodzaju). Można spodziewać się, że w przypadku wydzielenia 3ciego ciągu do testów, ukrytego na czas selekcji modelu użycie wielu złożeń powinno dawać lepsze wyniki.

Zwyczajową ilością złożeń jest 10.

# VI: Wnioski

Naiwny klasyfikator bayesa jest łatwą w implementacji, szybką i dość skuteczną metodą klasyfikacji danych liniowo niezależnych.

Niestety osiągane wyniki (dokładność do 75% zależnie od złożeń) nie są idealnie satysfakcjonujące. Wg autora wynika to z tego, że między danymi zachodzą zależności, których naiwny klasyfikator bayesa nie jest w stanie zauważyć.

Być może redukcja wymiarów (np metodą PCA) umożliwyłaby uzyskanie lepszych wyników. Założenie pochodzenia danych z rozkładu normalnego powinno zostać dodatkowo zweryfikowane testami normalności (np testem Kołmogorowa - Smirnowa lub tesstem Shapiro - Wilka).

Co ciekawe, walidacja krzyżowa nie spowodowała znacznego spadku dokładności klasyfikacji (wyniki pomiędzy 75% a 73%), co wskazuje na poprawną zdolność generalizacji.

Wg autora naiwny klasyfikator bayesowski, z uwagi na prostotę w implementacji i możliwość radzenia sobie z małą ilością przykładów może być bardzo skutecznie stosowany jako forma wspomagająca w innych metodach uczenia maszynowego. Np wydaje się być on dość dobry do szybkiego sortowania które wartości hiperparamterów sieci neuronowej bądź metody random forest warto przetestować najpierw.

In [ ]: