Sprawozdanie nr 1

Systemy Sztucznej Inteligencji

1. Wczytać zbiór danych iris. Podać liczbę próbek w tym zbiorze oraz ilość atrybutów opisujących każdą z nich.

```
iris = []
 with open("E:/ssi/iris.data") as file:
 for line in file:
 iris.append(line.rstrip().split(","))
print("Probe amount: ", len(iris))
 print("Attribute count in each probe: ", len(iris[1]))
```

2. Odczytać wartości atrybutów dla próbek o numerach 10 i 75. Obliczyć ich odległość euklidesową.

```
print("Iris 10th probe: ", iris[9])
print("Iris 75th probe: ", iris[74])
dist = math.sqrt
( pow((float(iris[9][0]) - float(iris[74][0])), 2)
 + math.sqrt(pow(float(iris[9][1]) - float(iris[74][1]), 2)))
 print("Euclidean distance: ", dist)
```

3. Podać wartości minimalne, maksymalne, średnie i odchylenia standardowe dla każdego z atrybutów

```
def printStatistics(x):
print("min: ", min(x))
print("max: ", max(x))
print("mean: ", numpy.mean(numpy.asarray(x, dtype=float)))
print("std: ", numpy.std(numpy.asarray(x, dtype=float)))
irisArray = numpy.array(iris)
irisArrayTrans = irisArray.T print("slength:")
printStatistics(irisArrayTrans[0])
print("swidth:")
printStatistics(irisArrayTrans[1])
print("plength:")
printStatistics(irisArrayTrans[2])
print("pwidth:")
printStatistics(irisArrayTrans[3])
```

4. Dokonać wizualizacji zbioru iris w przestrzeni złożonej z dwóch pierwszych atrybutów

```
plt.scatter(numpy.asarray(irisArrayTrans[0],dtype=float),
numpy.asarray(irisArrayTrans[1],dtype=float) )
plt.show()
```

5. Dokonać wizualizacji zbioru iris w przestrzeni złożonej z atrybutów 1 oraz 3 przy czym elementy każdej z klas zaznaczyć innym kolorem

```
numpy.random.seed(19680801)
colors = numpy.random.rand(len(iris))
plt.scatter(numpy.asarray(irisArrayTrans[0], dtype=float),
numpy.asarray(irisArrayTrans[2], dtype=float), c=colors)
plt.show()
```

6. Podać średnią wartość zmierzonych atrybutów dla próbek z klasy *setosa* oraz *versicolor*.

```
def means(x):
print([numpy.mean(numpy.asarray(x[0], dtype=float)),
numpy.mean(numpy.asarray(x[1], dtype=float)),
numpy.mean(numpy.asarray(x[2], dtype=float)),
numpy.mean(numpy.asarray(x[3], dtype=float))])
setosaArray = []
versicolorArray = []
for i in range(len(irisArray)):
if (irisArray[i][4] == "Iris-setosa"): setosaArray.append(irisArray[i])
if (irisArray[i][4] == "Iris-versicolor"):
versicolorArray.append(irisArray[i])
setosaArrayTrans = numpy.array(setosaArray).T
versicolorArrayTrans = numpy.array(versicolorArray).T
print("Setosa")
means(setosaArrayTrans)
print("Versicolor")
means(versicolorArrayTrans)
```

7. Dane poddać normalizacji i po dokonaniu tej operacji obliczyć ponownie wartości minimalne, maksymalne, średnie i odchylenia standardowe dla każdego z atrybutów.

```
irisArray2 = numpy.delete(irisArray, numpy.s_[4], axis=1)
standardizedData = StandardScaler().fit_transform(irisArray2)
standardizedDataTrans = numpy.array(standardizedData).T
#Using printStatistics(x) from 3.
print("slength:")
printStatistics(standardizedDataTrans[0])
print("swidth:")
printStatistics(standardizedDataTrans[1])
print("plength:")
printStatistics(standardizedDataTrans[2])
print("pwidth:")
printStatistics(standardizedDataTrans[3])
```

8. Wygenerować losowo zbiór 10 danych w przestrzeni dwuwymiarowej. Pierwszy atrybut z rozkładu N(−2,1)N(−2,1), drugi z rozkładu jednostajnego na przedziale [0,10][0,10]. Zbiór danych zwizualizować za pomocą wykresu.

```
s0 = np.array(numpy.split(np.random.normal(-2, 1, 10), 1))
s1 = np.array(np.split(np.random.uniform(0, 10, 10), 1))
array = np.concatenate((s0.T, s1.T), axis=1)
plt.scatter(numpy.asarray(array[0], dtype=float),
numpy.asarray(array[1], dtype=float))
plt.show()
```

9. Podać macierz odległości euklidesowych, mahalanobisa oraz Minkowskiego L1L1 dla wszystkich par elementów tego zbioru

```
euclDist=metrics.pairwise.pairwise_distances(array, metric='euclidean')
print(euclDist)
mahDist=metrics.pairwise.pairwise_distances(array, metric='mahalanobis')
print(mahDist)
minkDist=metrics.pairwise.pairwise_distances(array, metric='minkowski')
print(minkDist)
```

10. Dokonać skalowania liniowego wygenerowanego zbioru na przedział [0,1][0,1] i ponownie obliczyć odległości dla wszystkich par obiektów.

```
scaling = preprocessing.MinMaxScaler((0,1))
scal = scaling.fit_transform(array)
euclDist=metrics.pairwise.pairwise_distances(scal, metric='euclidean')
print(euclDist)
mahDist=metrics.pairwise.pairwise_distances(scal, metric='mahalanobis')
print(mahDist)
minkDist=metrics.pairwise.pairwise_distances(scal, metric='minkowski')
print(minkDist)
```

11. Zaproponować postać funkcji klasyfikujących dla problemu klasyfikacyjnego dla dwóch klas i dwuwymiarowej przestrzeni cech przy założeniu, że elementy klasy 1 znajdują się w drugiej, a elementy klasy 2 w czwartej ćwiartce układu współrzędnych. Podać wzór określający powierzchnię decyzyjną tego klasyfikatora.

```
def f1(x):
return -x[0] + x[1]
def f2(x):
return x[0] - x[1]
# Powierzchnia decyzyjna: x[0]=-x[1]
# Classificator
def classify(x):
if f1(x) > f2(x):
 return 1
 else:
 return 2
```

12. Wygenerować przykładowy zbiór danych testowych (po 10 próbek na klasę) dla problemu z poprzedniego zadania. Dokonać testowania zaproponowanego klasyfikatora.

```
x1 = np.array(np.split(np.random.uniform(1, 10, 10), 1))
x2 = np.array(np.split(np.random.uniform(-10, -1, 10), 1))
data = np.array(np.concatenate((np.concatenate((x1.T, x2.T), axis=1),
np.concatenate((x2.T, x1.T), axis=1)), axis=0))
# randomize order in array
np.random.shuffle(data)
print(data)
labels = np.array([classify(data[i]) for i in range(len(data))])
print('Classifications for data:')
print(labels)
```