Sprawozdanie nr 2

Systemy Sztucznej Inteligencji

1. Wczytać zbiór danych banana. Dokonać podziału zbioru danych na część uczącą oraz testową w sposób losowy, np. 30%30% dla uczenia, 70%70% dla testowania.

```
banana = sio.loadmat("D:/banana.mat")
train_data = banana["train_data"]
train_labels = banana["train_labels"]
train_labels = np.array(train_labels)
test_data = banana["test_data"]
test_labels = banana["test_labels"]
test_labels = np.array(test_labels)
train, dummy, train_targets, dummy = train_test_split
(train_data, train_labels.ravel(), test_size=0.70)
dummy, test, dummy, test_targets = train_test_split
(test_data, test_labels.ravel(), test_size=0.70)
```

2. Dokonać uczenia klasyfikatora Bayesa na zbiorze uczącym i jego testowania na zbiorze testowym.

```
gaussiannb = GaussianNB()
tmp = gaussiannb.fit(train, train_targets)
Z = tmp.predict(test)
```

3. Zwizualizować na płaszczyźnie 2D wynik klasyfikacji na zbiorze testowym. Narysować obszary decyzyjne.

```
c1 = (Z == 1).nonzero()
c2 = (Z == 2).nonzero()
plt.scatter(test[c1, 0], test[c1, 1], c="g", label="Grupa 1")
plt.scatter(test[c2, 0], test[c2, 1], c="r", label="Grupa 2")
plt.legend()
# # obszary decyzyjne
C = 1.0
h = .02
x_min, x_max = test[:, 0].min() - 1, test[:, 0].max() + 1
y_min, y_max = test[:, 1].min() - 1, test[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
np.arange(y_min, y_max, h))
Z = tmp.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, cmap=plt.cm.Paired)
plt.show()
```

4. Ocenić sprawność klasyfikatora (procent poprawnych klasyfikacji).

```
print(round(tmp.score(test, test_targets) * 100, 2))
```

5. Zaimplementuj klasyfikator minimalno-odległościowy.

```
tmp = NearestCentroid()
```

6. (zbiór danych banana) Dokonać uczenia klasyfikatora minimalno-odległościowego na zbiorze uczącym i jego testowania na zbiorze testowym.

```
tmp.fit(train, train_targets)
Z = tmp.predict(test)
```

7. Zwizualizować na płaszczyźnie 2D wynik klasyfikacji na zbiorze testowym, zaznaczając każdą z klas innym kolorem. Na wykresie umieścić również środki klas uzyskane w procesie uczenia.

```
c1 = (Z == 1).nonzero()
c2 = (Z == 2).nonzero()
plt.scatter(test[c1, 0], test[c1, 1], c="g", label="Klasa 1")
plt.scatter(test[c2, 0], test[c2, 1], c="r", label="Klasa 2")
plt.legend()
plt.scatter(tmp.centroids_[:, 0], tmp.centroids_[:, 1], c="b")
plt.show()
```

8. Ocenić sprawność klasyfikatora (procent poprawnych klasyfikacji).

```
print("Sprawnosc klasyfikatora: ", tmp.score(test, test_targets))
```

9. Wczytać zbiór banana. Przetestować klasyfikator kNN dla kilku wartości parametru k i wybrać tą dla której uzyskana sprawność jest maksymalna – podać ile wynosi.

```
bestScore = 0
bestK = 0
for k in range(1, 10):
clf = neighbors.KNeighborsClassifier(k, weights='uniform',
metric='euclidean') clf.fit(train, train_targets)
tempScore = clf.score(test, test_targets)
if tempScore > bestScore:
bestScore = tempScore
bestK = k
print("Best score: ", bestScore, ", for k: ", bestK)
```

10. Dla parametru kk z poprzedniego zadanie wizualizować wyniki klasyfikacji na płaszczyźnie 2D, zaznaczając każdą z klas innym kolorem.

```
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
Z = neighbors.KNeighborsClassifier(bestK, weights='uniform',
metric='euclidean').fit(train, train_targets).predict(
np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z)
plt.scatter(test[:, 0], test[:, 1], c=test_targets, cmap=cmap_bold)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.show()
```

11. Sprawdzić ile obiektów zbioru testowego zostalo błędnie zaklasyfikowanych.

```
clf = neighbors.KNeighborsClassifier(bestK, weights='uniform',
metric='euclidean')
clf.fit(train, train_targets)
clfScore = clf.score(test, test_targets)
print("Sprawnosc: ",clfScore)
print("zle zakwalifikowanych: ", math.floor(len(test_data) * (1 -
clfScore)))
```