**import** scipy.io **as** sio  
**import** sklearn.model\_selection **as** sms  
**import** sklearn.naive\_bayes **as** snb  
**import** sklearn.neighbors **as** sn  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.colors **as** mc  
**import** math  
  
*#1*banana = sio.loadmat(**"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 = sms.train\_test\_split (train\_data, train\_labels.ravel(), test\_size=0.70)  
dummy, test, dummy, test\_targets = sms.train\_test\_split (test\_data, test\_labels.ravel(), test\_size=0.70)  
*#1  
  
  
#2*gaussiannb = snb.GaussianNB()  
tmp = gaussiannb.fit(train, train\_targets)  
Z = tmp.predict(test)  
*#2  
  
  
#3*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()  
*#3  
  
  
#4*print(round(tmp.score(test, test\_targets) \* 100, 2))  
*#4  
  
  
#5*tmp = sn.NearestCentroid()  
*#5  
  
  
#6*tmp.fit(train, train\_targets)  
Z = tmp.predict(test)  
*#6  
  
  
#7*plt.close()  
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()  
*#7  
  
  
#8*print(**"Sprawnosc klasyfikatora: "**, tmp.score(test, test\_targets))  
*#8  
  
  
#9*bestScore = 0  
bestK = 0  
**for** k **in** range(1, 10):  
 clf = sn.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)  
*#9  
  
  
#10*cmap\_light = mc.ListedColormap([**'#FFAAAA'**, **'#AAFFAA'**, **'#AAAAFF'**])  
cmap\_bold = mc.ListedColormap([**'#FF0000'**, **'#00FF00'**, **'#0000FF'**])  
Z = sn.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()  
*#10  
  
  
#11*clf = sn.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)))  
*#11*