University of Belgrade - Faculty of Electrical Engineering

Department of Signals and Systems

13EO54PO – Pattern recognition

Student: Teodora Spasojević

Index count: 2019/0427

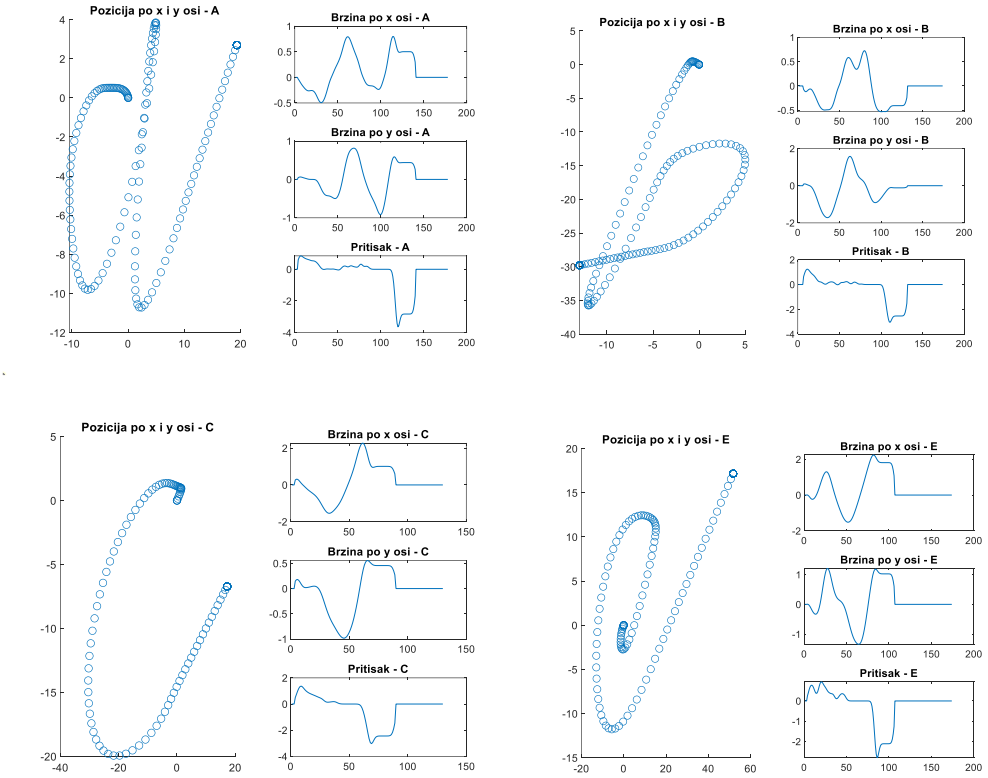
February 2022.

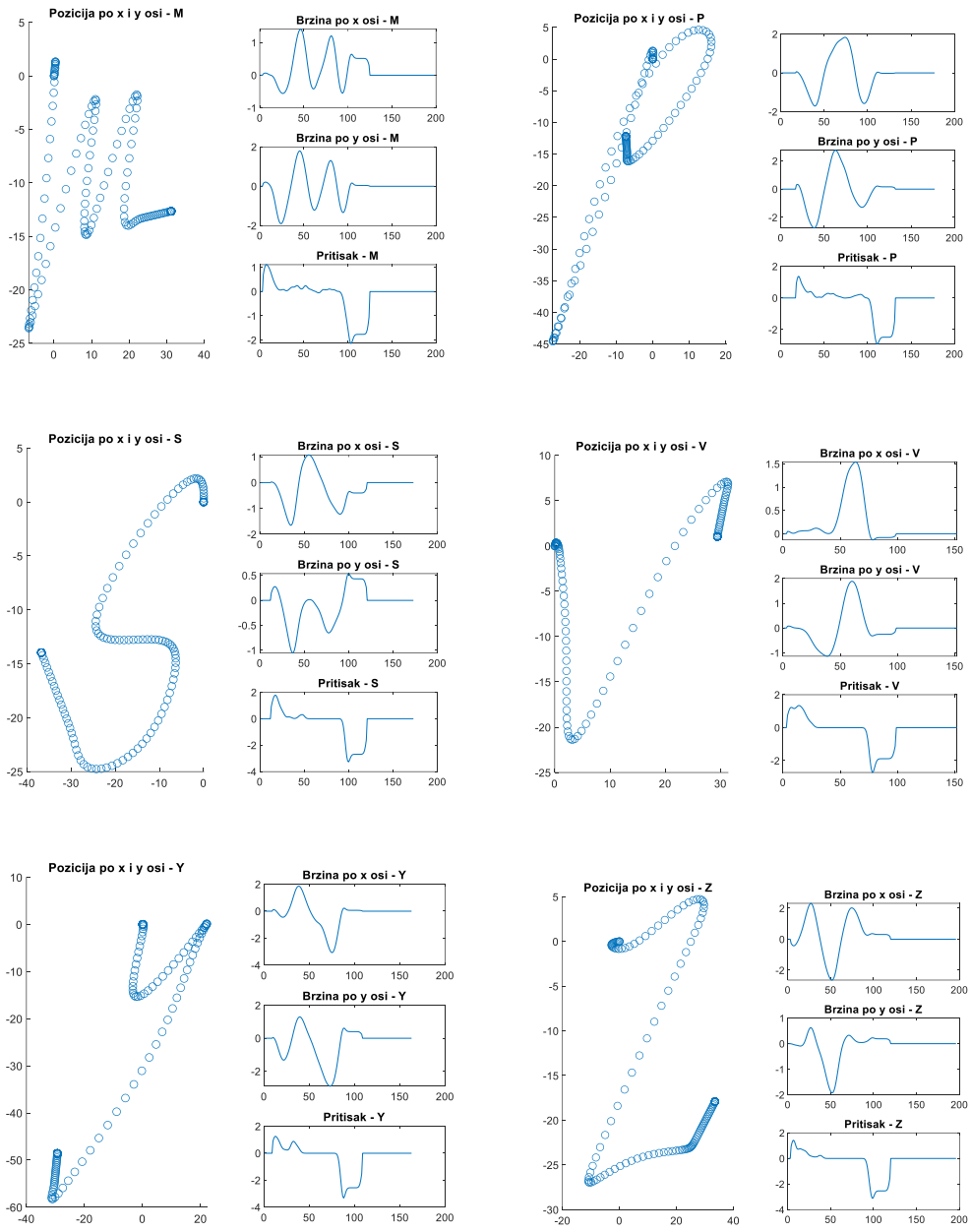
Task 1 - Letter Classification

The task requirement is to classify letters written on a graphics board and, using a classifier based on hypothesis testing, and a parametric classifier. The data we have about the letters is placed in the table PO_slova.mat and it is information about the speed of writing on the x and y axes, as well as the pressure achieved on the graphics board when writing a specific word. The table contains data on 20 letters, of which it is necessary to use 10 and accurately classify them.

Firstly, it is necessary to load the data and display it. To achieve this, a function data_extraction is written that loads individual data on speeds and pressures when writing letters on the graphics board and calculates the position of letters selection on the board as a cumulative sum of speed when writing, on the corresponding axis.

After we have loaded all the data, we select one copy of each selected letter and display the information we have about it:





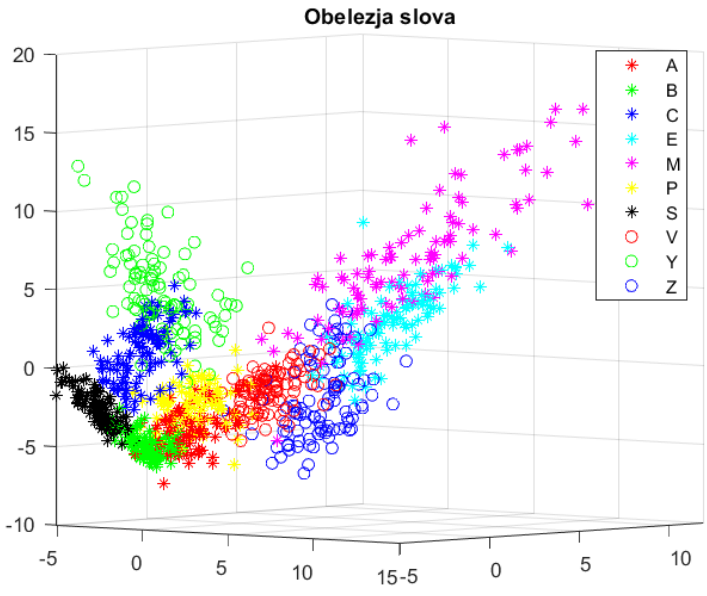
Looking at the obtained graphics, we can see the connection of the obtained images for the position of the letters, with writing speeds on the x and y axes. Each sudden (min/max) change in writing on a particular axis represents a change in the direction of writing on that axis, while the parts where the speeds are constant represent the strokes in which We don't change the direction of writing on a given axis.

After successfully loading the letters that we need to classify, it is necessary to choose the features that we will use to recognize different classes of letters. For this, a special function, feature_extraction, is written, which finds 15 features from the information we have about each letter. The features we have singled out are min and max positions of shapes on x and y axes, ranges of these positions, min and max pressure, pressure range, mean speed value on the x and y axes, as well as the number of letters selected on the left, right upper and lower half of the image.

As it is very ungrateful to work with such a number of features, both because of the mathematical complexity, and because it is much easier for us to work with forms with 2 or 3 features, because in this way we can visually display the shapes and more intuitively design the classifier, the next step is to use one of the methods of dimension reduction.

The method used for dimension reduction is the LDA method. It provides a solution for finding a transformational matrix, which will map our 15-character shape into a new form with 3 features, without reducing the knowledge that we have summed up in all the features. This is achieved by choosing the highest eigen values of matrix S as elements of the transformation matrix. We know that the error of our classifier is proportional to the sum of all the eigen values obtained, and by simply finding the minimum eigen values and corresponding vectors, we can move onto the three-dimensional forms with minimal loss of information. Matrix S, for which we calculate eigen values and vectors, is a matrix formed to contain information about the distance of class forms and the dispersion of shapes from a particular class. The matrix S contains information about the relationship between the matrix of inter-class scattering (Sb) and the matrix of intra-class scattering (Sw). The matrix Sb observes the distance of classes by observing the distance of the estimated mean values of classes and is in the numerator because we want to maximize this distance and make classification as easy as possible. On the other hand, the Sw matrix observes class scattering by looking at the covariate matrices of all of the classes and is found in the denominator because we want to minimize these values, to make as little overlap as possible in the selections, among the different classes. By setting a criterion function that depends on S (e.g. trace(S)), we manage to come to the above-mentioned conclusion about the choice of a transformation matrix in this method, and move into the three-dimensional system in which we have new features, which manage to preserve the separability of classes.

After applying dimension reduction, we can visualize the shapes obtained for our letters and assess whether the features are sufficiently well chosen, so that there is separability between classes.



We see that there is a clear separability among the obtained forms of certain classes, while in some classes there is a greater overlap of forms. Once we have formed the feature vectors, we can proceed to the design of classifiers based on hypothesis testing.

We design the classifier by first estimating the mean values and covariate matrices of our classes. After that, we go through all forms again and observe what is the probability density function of this form, relative to the estimated parameter of all classes. After finding all the functions of the probability density, we decide to classify the form into the class that we get the highest probability that the class of our form is. As we know for all forms which class they actually originate from, after we get the result of the classifier, we can record the error or hit of the classifier depending on the result, in the confusion matrix. Because of the knowledge we have about which class belongs to which form, this type of classification training is called supervised learning. Also, a classifier in which we classify by setting a hypothesis, such as that a form belongs to the class for which it possesses the highest value of the function, is called classifier based on hypothesis testing. The probability density is the classifier of the hypothesis test. There will be more information about these types of classifications in Task 2.

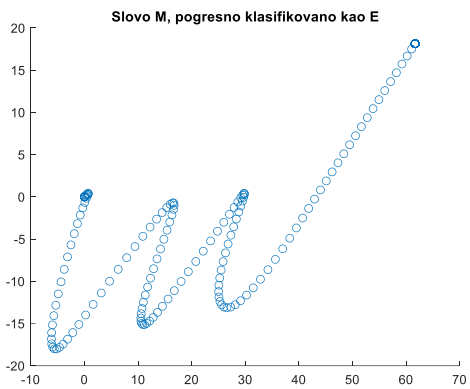
As a result of the algorithm, we also get a confusion matrix that carries information about the number of (in)exactly classified forms. Rows of the confusion matrix represent labels of the estimated classes, and columns are labels of the exact classes.

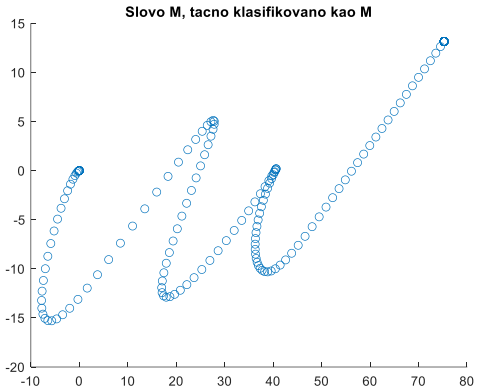
The confusion matrix:

	A	**B**	**C**	**E**	**M**	**P**	**W**	**V**	**Y**	**Z**
A	**36**	0	0	0	0	11	0	0	0	0
B	51	**99**	0	0	0	29	0	0	0	0
C	0	1	**84**	0	0	0	1	0	12	0
E	0	0	0	**94**	11	0	0	0	0	0
M	0	0	0	6	**82**	0	0	1	0	2
P	13	0	0	0	0	**58**	0	0	3	0
W	0	0	0	0	0	0	**99**	0	0	0
V	0	0	0	0	1	0	0	**96**	0	5
Y	0	0	16	0	0	2	0	0	**85**	0
Z	0	0	0	0	6	0	0	3	0	**93**

Total classification error: **0.174%**

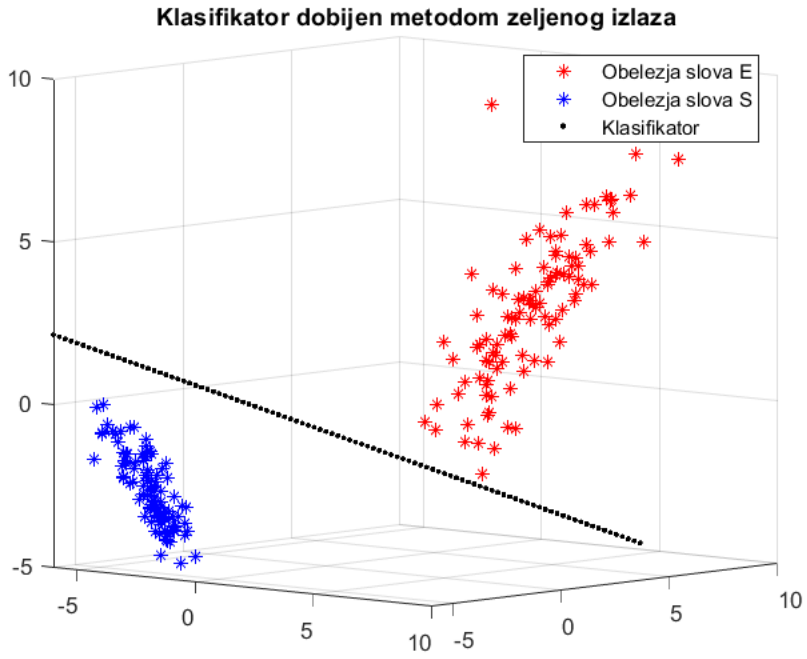
We see that the error of our classifier is very small, but that it exists. We can look at the example of good and bad classification of the letter M.





The last requirement of the task is to choose two letters that are linearly separable and choose one of the parametric classification methods for their classification. The letters chosen are E and S, and the parametric classification method is the wanted output method. It finds the parameters of the linear classifier, V I v0, and taking into account the given matrix, he wants the output. By choosing values in the matrix for the desired output, we influence which class, and which shapes we want to classify more accurately.

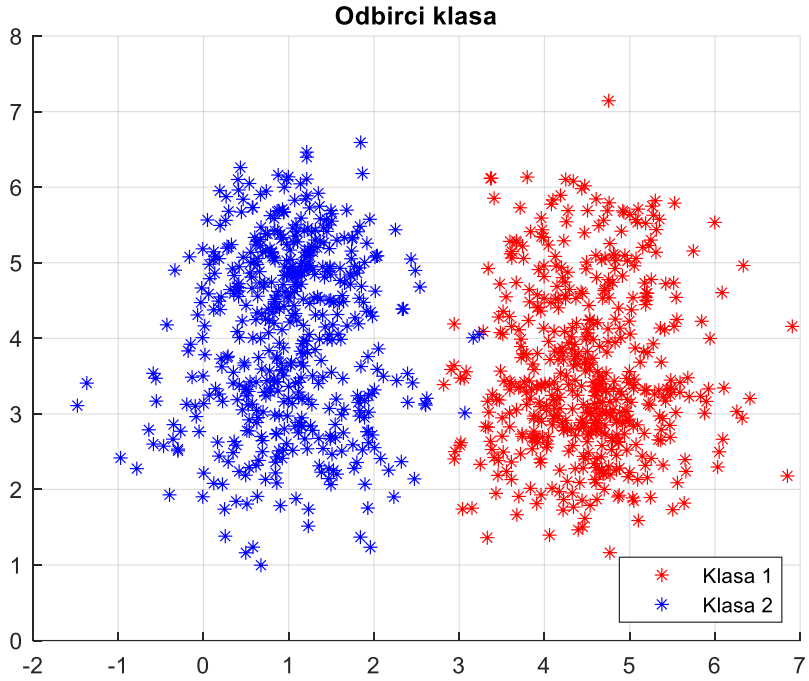
As a result of this method, we get the following classifier:



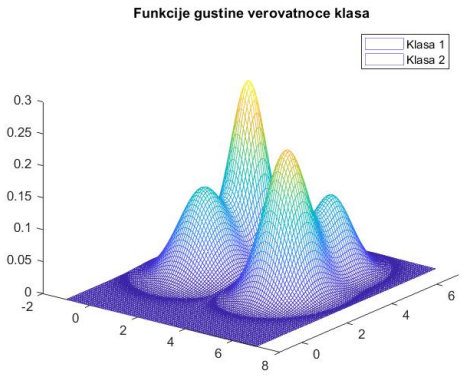
Task 2 – Hypothesis Testing

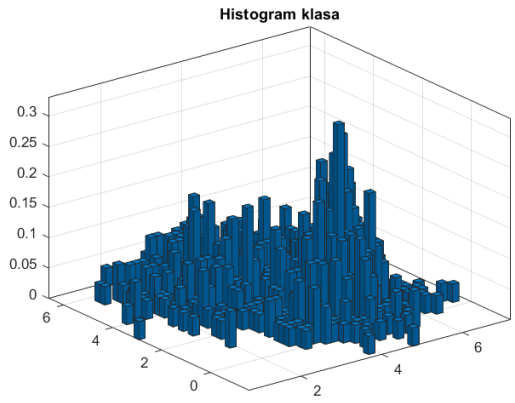
A special type of classifiers are classifiers formed based on hypothesis testing. In them, the classification is made by setting a hypothesis, which is then tested in the test, and based on the results of the set tests, we decide in which class to classify certain forms. The problem with these classifiers is that we need to know the probability density function of a shape from different classes, to be able to set the test. Therefore, often the probability density function is evaluated using a nonparametric method (such as histogram, using kernel functions, etc.), and then the estimated probability density function is used in setting up hypothesis tests. This type of classification is also called the non-parametric method of classification, because of the use of a nonparametric method for estimation of the probability density function.

The class samples that we will classify in this, and subsequent tasks, are generated by defining the desire of the mean matrix and the covariation matrix of classes, and then we get the samples using the mvnrnd function. This function generates uniformly distributed samples, and then by applying the transformation matrix $\boldsymbol{\Phi\Lambda^{1/2}}$, where Φ is the matrix of its eigen vectors and Λ is the matrix of its eigen values, obtained by them from the desired mean and covariate matrix. Using this transformation matrix, uniform any distributions are transformed into samples of classes with desired parameters. Using this method and defining the desired class parameters, we get the following sample classes:



In order to proceed to the design of classifiers, as already stated, it is necessary to calculate the probability density functions of classes. For the probability density functions of classes, we get the left graph:





We have said that in addition to the computation of the exact probability density function, we can also estimate it using a nonparametric method, such as histogram, which is shown on the right graph.

The first classifier we have designed is **the Bayesian classifier**. The test set by the Bayesian classifier compares the values of the conditional probabilities whether the form is from the first or second class, based on the probability density functions that we know for classes:

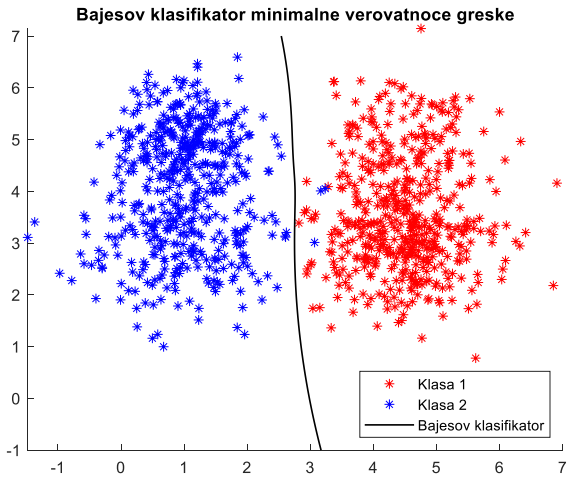
$q1(x) \lessgtr q2(x)$, $qi(x) = Pifi(x)/f(x)$, and classifies the shapes into a class with a larger *qi(x).*

By developing this inequality and computing its logarithm, we get the final form of the test:

$$h(x) \lessgtr T, \quad h(x) = -\ln\left(\frac{f1(x)}{f2(x)}\right), \quad T = \ln\left(\frac{P1}{P2}\right)$$

where *h(x)* is preceded by the discriminatory function and T is the threshold of the classifier.

As a classifier for our classes, we obtain the following discriminatory function:



The advantage of this classifier is that it minimizes the overall probability of error of the classifier. However, it is often more important to us that the classifier makes a minor mistake

in classifying the choices of one class than another. That is why we form a minimum price classifier.

The minimum price classifier sets a test that compares the cost of the decision that the shape belongs to one of the classes.

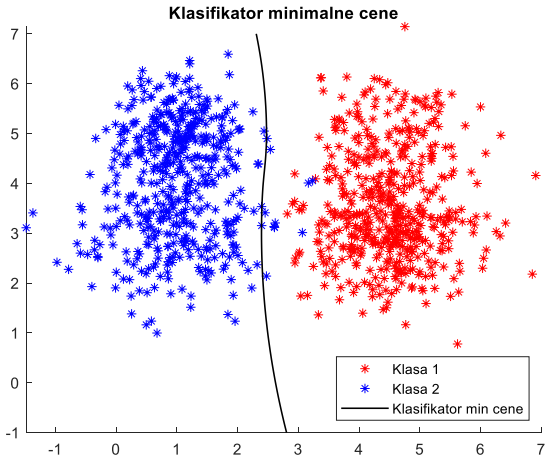
$r1(x) \lessgtr r2(x),\ ri(x) = ci1q1(x) + ci2q2(x),$ and classifies shapes into a class with a lower price decision.

By developing this inequality and computing its logarithm, we get the final form of the test:

$$h(x) \lessgtr T, \quad h(x) = -\ln\left(\frac{f1(x)}{f2(x)}\right), \quad T = \ln\left(\frac{P1(c21-c11)}{P2(c12-c22)}\right)$$

where *h(x)* is preceded by the discriminatory function and T is the threshold of the classifier.

As a classifier for our classes, we obtain the following discriminatory function:



This classifier manages to classify shapes, so that greater importance is attached to the good classification of forms from class 1. However, the disadvantage of this classifier is that we do not know exactly how to choose the price coefficients of classification of the shape in a certain class. That's why we're forming the Neyman-Pearson test.
Also, the Bayesian test and the minimum price test can be generalized to the test of multiple hypotheses and then used to classify more than two classes.

The **Neyman-Pearson classifier** sets a criterion function in which it wants to minimize the probability of error of the first type (which corresponds to the failed detection which is more important) and sets the probability of error of the second type (false alarm) at a fixed value.

$$r = \varepsilon 1 + \mu(\varepsilon 2 - \varepsilon 0)$$

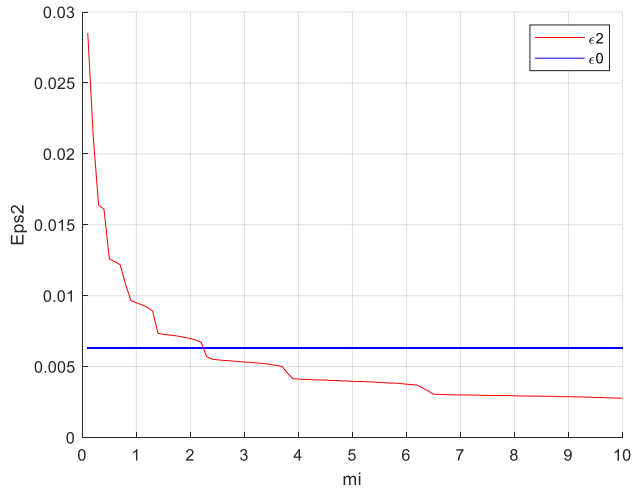
The result of minimization of this criterion is the following classifier:

$$h(x) \lessgtr T, \quad h(x) = -\ln\left(\frac{f1(x)}{f2(x)}\right), \quad T = -\ln(\mu)$$

where μ is found from the following equation:

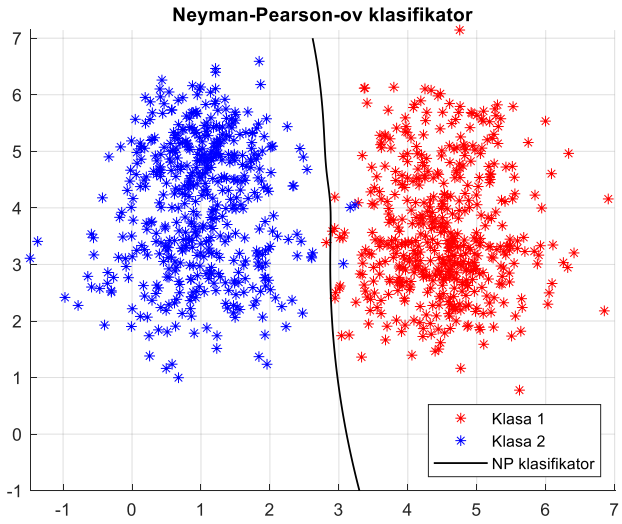
$$\varepsilon 0 = \int_{-\infty}^{-\ln(\mu)} f(h|w2)dh$$

The question is how to choose a value on which to fix probability of error of the second type. One option is to fix it on the value obtained from a previous test (such as Bayes), which we did in the task.



On the graph, we showed the dependence of the probability of the error of the first type from the choice of threshold μ and selected the value that we get for the probability of error of the second type, obtained from the Bayesian classifier .

As a classifier for our classes, we obtain the following discriminatory function:



In order to design a Neyman-Pearson classifier, it was necessary to calculate the error of the classifier (we calculated the error of the Bayesian classifier). We calculated this error in two ways:

1. Using a confusion matrix.

2. Using a theoretical approach that involves integrating surfaces below the probability density functions of classes, but in areas belonging to the opposite class from that for which we observe the probability density function.

As a confusion matrix, and the error calculated through it we get:

	X1	X2
X1_est	500	3
X2_est	0	497

Probability mistake of the first type - from a confused matrix: **0.006**

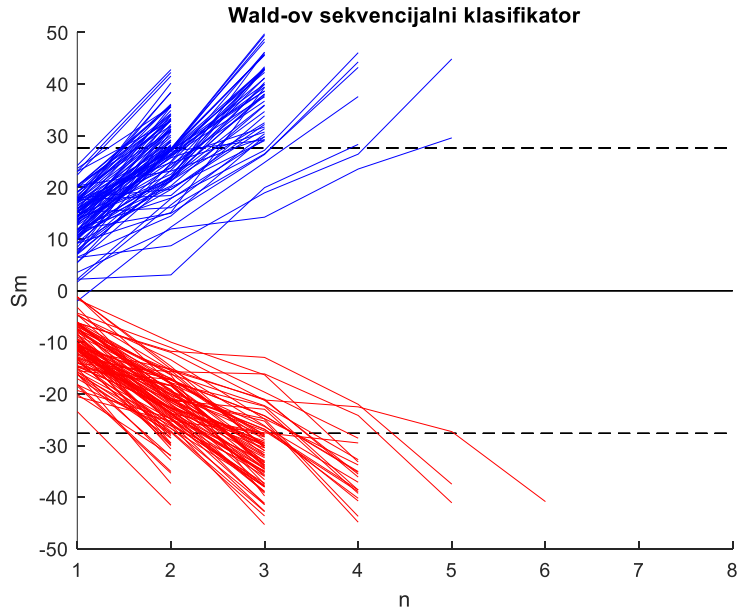
Probability mistake of the second type - from the confused matrix: **0**

For the error calculated by the theoretical approach, we get:

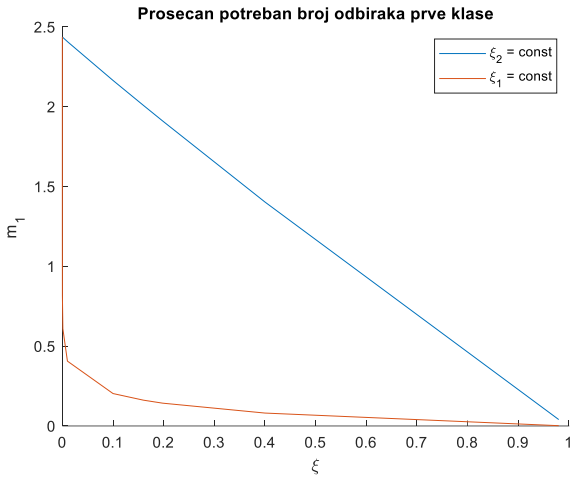
Probability mistake of the first type - theoretical approach: **0.0054435**
Probability mistake of the second type - theoretical approach: **0.0062965**

The last classifier we're designing is **Wald's sequential classifier**. Sequential hypothesis testing is based on the fact that with each new sample of a class, the classifier gathers knowledge by summing up discriminatory functions for individual measuring vectors, and after a defined number of selections, it makes a decision to which class the selections belong to. Unlike the general sequential test, Wald's sequential test defines the thresholds that the collected knowledge from the samples (which we keep in Sm) must cross, for a decision to be made. These thresholds are chosen so that Wald's classifier always decides the optimal number of samples (m).

In the following graphic we can see the iterations of Wald's sequential test shown:



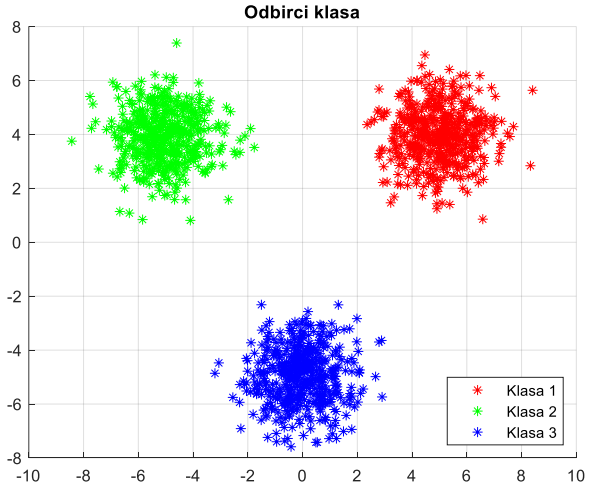
Depending on whether we fix the probability of an error of the first or second type, we get the following graphs of changing the required number of choices m:



Task 3 – Parametric classification methods

Since often the probability density functions of the classes are not available for classification of forms, the solution is found in new classifier design methods. One of these solutions are parametric classification methods, which do not require knowledge of the functions of the probability densities of classes but assume the shape of the classifier based on the distribution of class samples and evaluate the parameters of the assumed classifier based on the part of the available samples we call **the training set**. Later, we evaluate the performance of our classifier on the part of the samples that did not participate in the training, and we call this set of selections **a testing set**.

The first requirement is to generate linearly separable classes and design two classifiers that can classify these classes. The classes we have generated are distributed as follows:



The first classifier we designed is a part-by-part linear classifier. We use it in situations where we have inter-linearly separable classes, but their number is greater than 2, and one classification line is not enough. We design it by designing, in the first case, an optimal linear classifier between every two classes. The optimal linear classifier is of the form $V^T X + v0 \lessgtr 0$, and the parameters of the classifier, *V* and *v0* are calculated on the basis of equations obtained by optimizing the probability of the error of the classifier:

$$s = \frac{\frac{\partial \varepsilon}{\partial \sigma_1^2}}{\frac{\partial \varepsilon}{\partial \sigma_1^2} + \frac{\partial \varepsilon}{\partial \sigma_2^2}} = \frac{-\eta_1 / \sigma_1^2}{-\eta_1 / \sigma_1^2 + \eta_2 / \sigma_2^2}$$

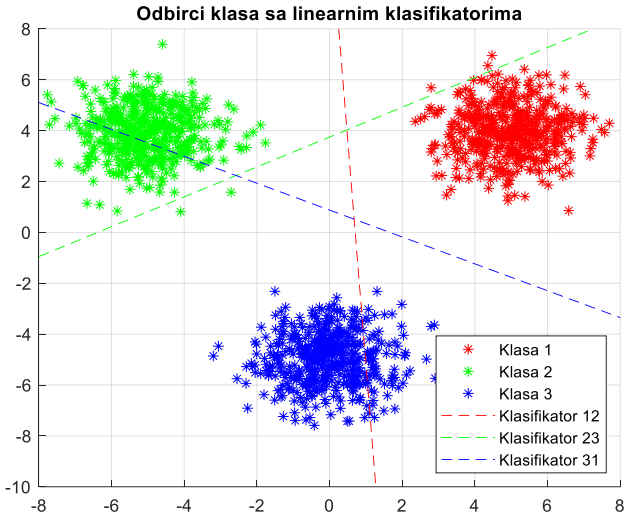
$$V = \left[s\Sigma_1 + (1-s)\Sigma_2 \right]^{-1} (M_2 - M_1)$$

$$v_0 = -\frac{s\sigma_1^2 V^T M_2 + (1-s)\sigma_2^2 V^T M_1}{s\sigma_1^2 + (1-s)\sigma_2^2}$$

Since this system of equations is transcendent, it is necessary to apply some iterative algorithm to design classifiers. A hold-out or third iterative method is selected in the task. In it, we divide the available data set into two sets, training, and test. In the beginning, in the training set we evaluate the initial mean matrices and the covariation matrices. Then, instead of calculating the value of s by the formula, we change s between the values of 0 and 1 with a defined step. Now that we have all the necessary parameters, we calculate the vector *V* via the second equation. On vector *V* we then project the samples from the training and test set. The projections of the training set are sorted ascending, and the threshold of the classifier -*vo* is calculated as an arithmetic mean between every two adjacent projections. For each selected -*vo*, we go through the projections of the selection test set on the vector *V* and calculate the classifier error for the given -*v0*. We choose the threshold -*v0* as the one for which we get the lowest number of misclassified samples. We find the optimal threshold for each value of s. The final parameters of the classifier are calculated for the value s for which we obtain the lowest error of the classifier.

We see that both the training and the test set participate in the training of classifiers. For this reason, the data is assigned to another set called **a validation set**. We used the training and validation set to train the classifier, and the test set to assess the error of the classifier and find the confusion matrix.

The discriminatory lines obtained using the hold-out method are:



The confusion matrix that we obtained, using the test set is:

	K1	K2	K3
K1_est	100	0	0
K2_est	0	100	0
K3_est	0	1	99

We see that due to the separation of the validation and test sets, we get a situation where the classifier makes a classification error.

In the next step, the discriminatory lines of the part-by-part linear classifier are designed using the method of desired output. In the desired output method, we form new vectors Z, so that we can set an unambiguous test, when classifying class forms:

$h(X) > 0$, $h(X) = W^T Z$they, $W = [v0\ V]^T$

$Z = [-1\ -X1 - X2]^T$ if X∈w1,

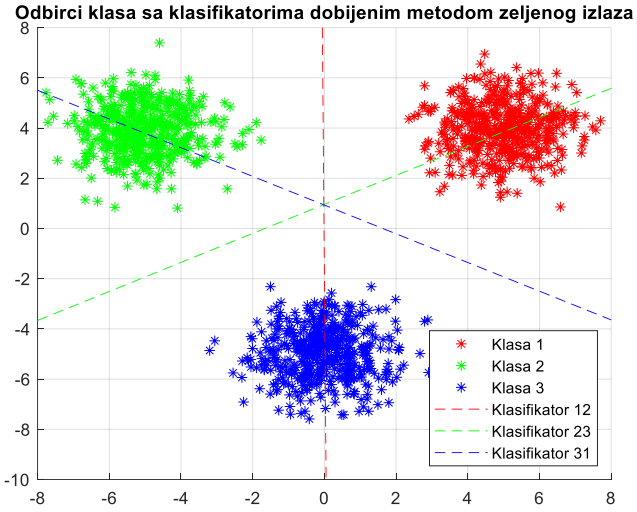
$Z = [1\ X1\ X2]^T$ if X∈w2

The design of the classifier involves finding the matrix *W*. By minimizing the criterion function of this method, we get that *W* is calculated as:

$W = (UU^T)^{-1}U\Gamma$, $\Gamma = [\gamma(Z1)\ \gamma(Z2)...\ \gamma(ZN)]^T$ and $U = [Z1\ Z2...\ ZN]$

where Γ is matrix of the desired outputs. By adjusting the elements of this matrix, we can influence a better classification of one class compared to another, etc.

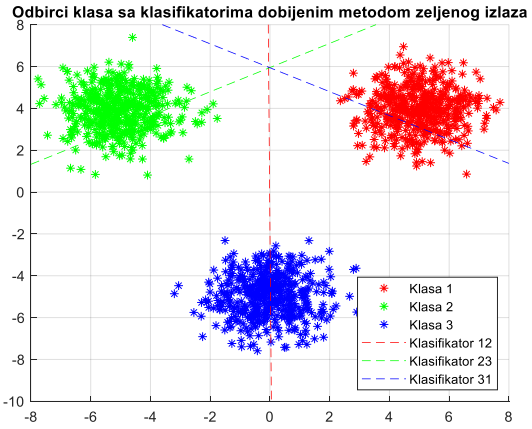
The discriminatory lines that we get using the desired output method are:



The confusion matrix we got is:

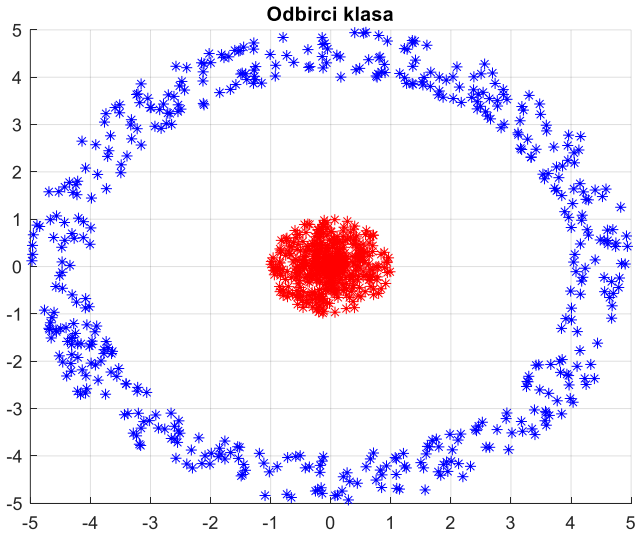
	K1	K2	K3
K1_est	500	0	0
K2_est	0	500	0
K3_est	0	0	500

If we want to see the impact of the value of the matrix of the desired outputs, we can make a requirement that class 3 is to be classified much better than other classes. Then, we get the following graphic:



We see in it that discriminatory curves have significantly distanced themselves from third-class samples.

The next task requirement is to generate two nonlinear separable classes. The classes we generated are as follows:

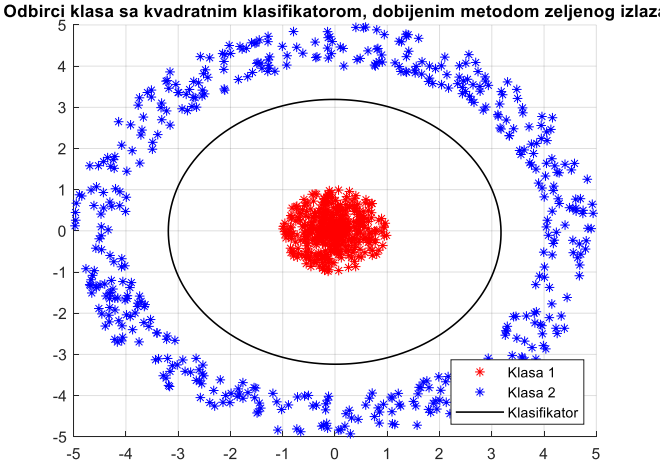


In order to successfully classify nonlinear separable classes, we need to switch from using a linear classifier to using a quadratic classifier. We designed the quadratic classifier by using the desired output method again, where we now modify the matrix Z to include combinations of features:

$Z = [1\ X1\ \ X2\ X1^2\ X2^2\ 2X1X2]^T$, if X∈w1,

$Z = [-1 - X1\ \ -X2\ \ -X1^2 -\ X2^2\ \ -2X1X2]^T$, if X∈w2

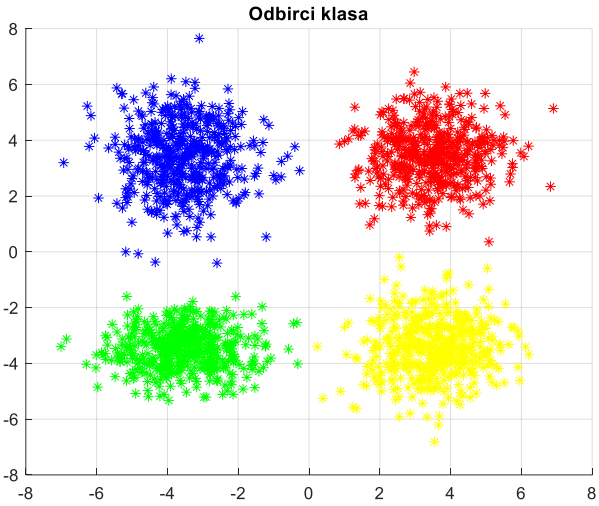
The discriminatory function we get for a quadratic classifier designed on the basis of the desired output is:



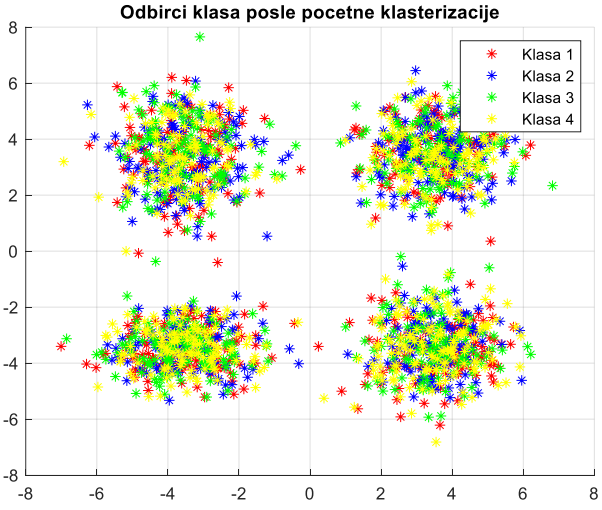
Task 4 - Clustering

Clustering is a classification method in which we do not rely on any prior knowledge related to our samples. Unlike the previous method of classification, where we owned a training set on which we find the parameters of our model, we do not have a training set in clustering. Clustering methods can also be parametric and non-parametric, but only parametric methods were used in this task. The parametric methods used in the task are the c-mean method, the maximum credibility method, and the quadratic decomposition method.

The first requirement of the task is to cluster linearly separable classes. The classes we generated are as follows:



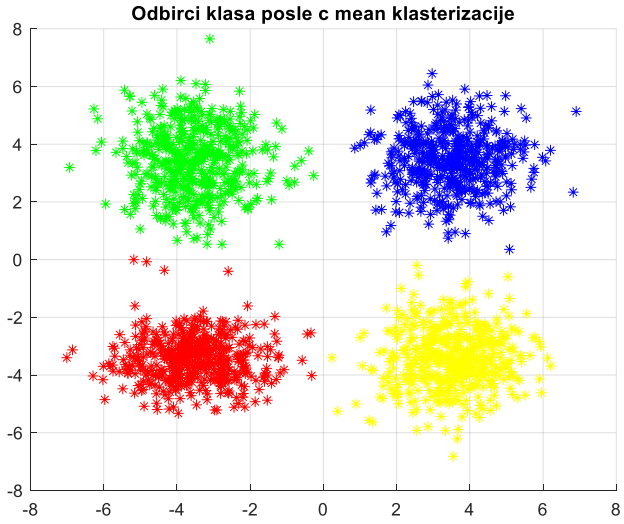
First, we will cluster classes using **the c-mean method**. With each clustering method, the algorithm starts with the execution of the initial clustering, which we performed in this task completely random. The result of the initial clustering is as follows:



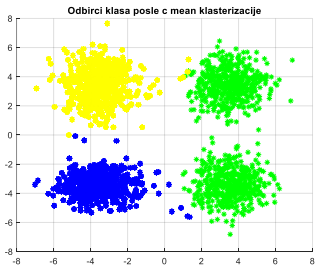
We can see that the samples of the classes are completely mixed. Once we have performed the initial clustering, we go into executing the algorithm.

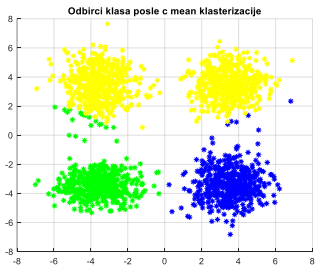
First, it is necessary to estimate the average values of the initially obtained clusters. After we have done this, we enter a loop which we do not exit until an iteration in which no new clustering has been performed occurs. In every iteration of the algorithm, we go through the samples of all the clusters and find their Euclidean distances from the current clusters. After finding the Euclidean distance for sample from each cluster, we recognize a cluster to which the Euclidean distance is minimal. Since our selection is the closest to this cluster, we believe that this sample should be placed into the recognized cluster. We do this by forming auxiliary clusters that we fill through iteration and then put for the final cluster. If the cluster recognized as the closest is not the cluster from which the sample originated, we record that new clustering has occurred.

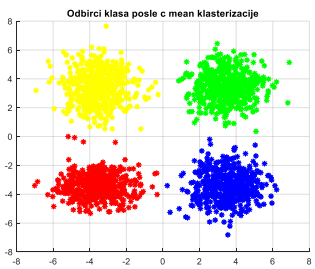
As Euclidean distance is viewed as a criterion for clustering, this method is used to clustering linearly separable classes. The result of clustering is:



We see that the algorithm has managed to cluster the selections into 4 separate classes. However, the c-mean method is heavily dependent on the choice of initial clustering. Therefore, if we change the initial clustering and reapply this method, we get the following graphics:



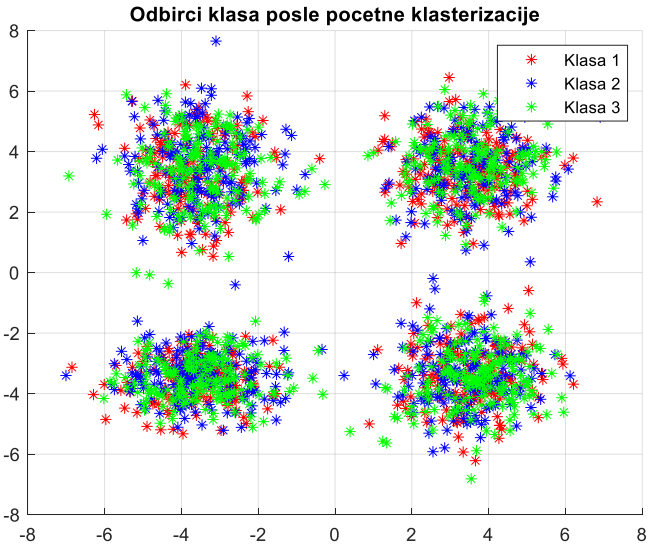




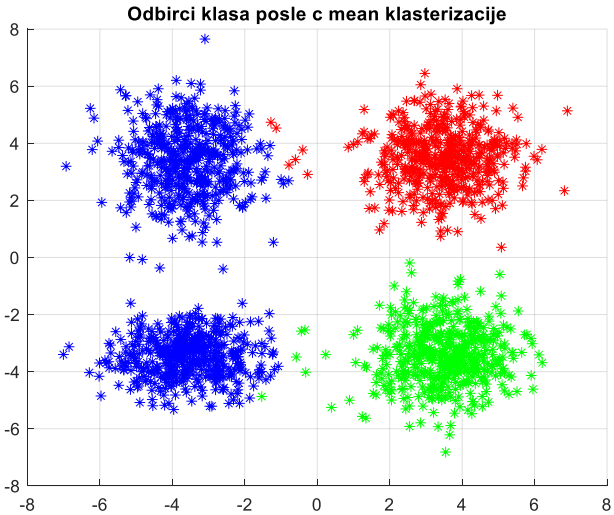
We see that depending on the initial clustering, we get different results. Sometimes we get well-recognized clusters, but in substitute places, and sometimes we don't get all 4 clusters recognized. This is because this algorithm can get stuck in the local minimum and recognize the wrong number of clusters, and we know that it does not have a prior knowledge of the number of clusters.

The mean number of iterations required is **5.1** iterations and it depends on the number of clusters and the initial clustering.

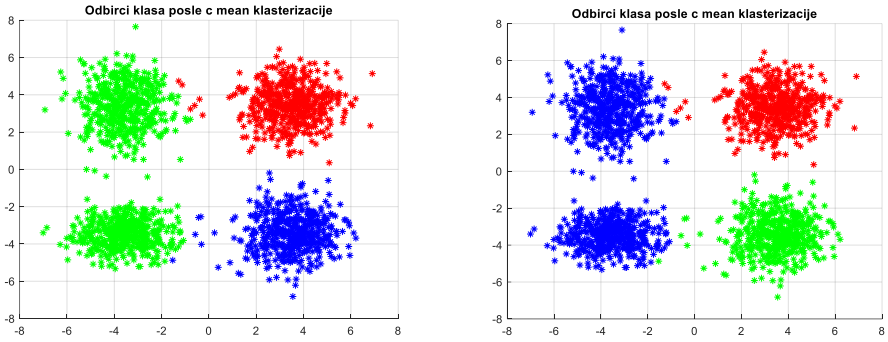
We can also show the impact of the number of clusters on the results of clustering. If we divide the initial 4 clusters into 3 clusters and perform the initial clustering, we get the following graph:



As a result of the clustering, we get:



If we were to change the initial clustering again, some of the graphics we would get are:



The average number of iterations required is **6.6** iterations.

We see that the algorithm, assuming that there are 3 clusters, merged 2 clusters into one and obtained the given results.

The classes generated for the previous request will be reused, and we will use the **maximum credibility method for** clustering. This method is based on the assumption that the probability density functions are the Gaussian function and this assumption is maximized by the parameters Pi, Mi and Σi, where the index i index of one of the clasters. By setting the criterion function that will maximize this assumption and finding its first derivative, we come to the following equations:

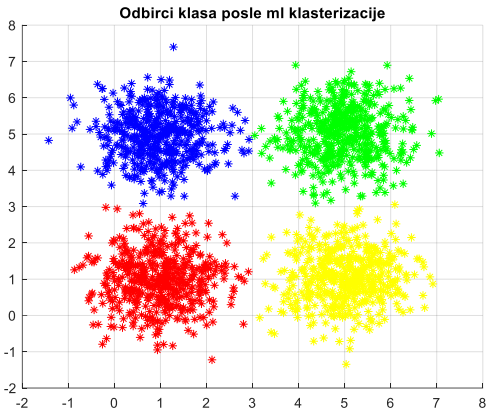
$$Pi = \frac{1}{N}\sum_{j=1}^{N} qi(Xj), \quad Mi = \frac{1}{N}\sum_{j=1}^{N} Xjqi(Xj), \quad \Sigma i = \frac{1}{N}\sum_{j=1}^{N} qi(Xj)(Xj - Mi)(Xj - Mi)^T$$

$$qi(Xj) = \frac{Pifi(Xj)}{f(Xj)}$$

We start the method again with the initial clustering and estimation of the parameters of such obtained clusters. Then we go into the algorithm. In every iteration of the algorithm, we calculate the parameters Pi, Mi and Σi as in the formula. Based on them, we calculate the probability density functions as Gaussian probability density functions, and based on them, the conditional probabilities qi(Xj), as in the formula. We calculate all the parameters for all of the samples from clusters. After finding all the conditional probabilities, we find absolute value of the difference between these probabilities obtained in the current iteration and obtained in the previous iteration. Then we take the biggest difference in probabilities obtained and see if it is below the set threshold. If this biggest difference is below the threshold, we can stop the algorithm. If the difference is greater than the threshold, we enter the next iteration.

The result of this part of the algorithm are vectors with conditional probabilities per cluster, for each sample. In order to perform the clustering itself, we go through all of the samples again, for each of the obtained vectors we find the maximum conditional probability, and based on which cluster it is tied up, we execute new clustering of the sample.

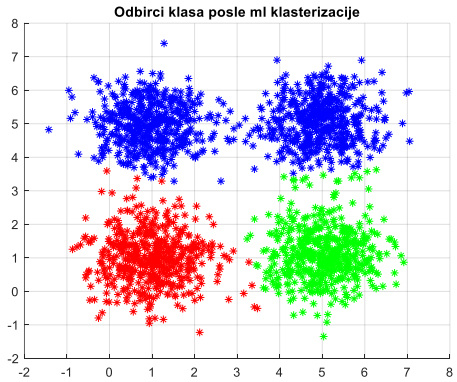
The result of clustering is as follows:

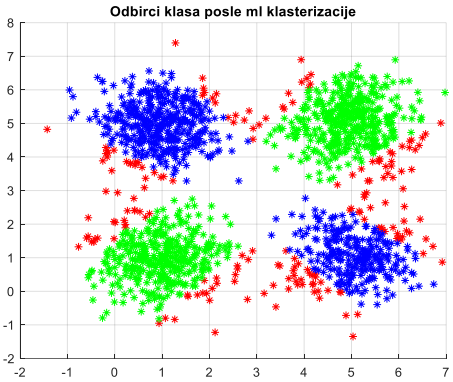


In this case, in order to help the algorithm in clustering data as quickly as possible, we did not perform the initial clustering by accident, but placed part of the selections in clusters that they really belong.

The average number of iterations in this method is **36.9.** We see that the required number of iterations is significantly higher in this method than in the c-mean method.

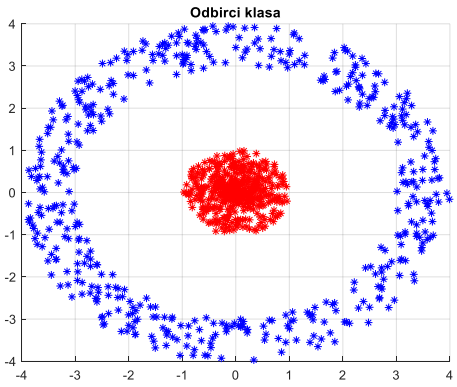
This method, like the previous one, depends on the initial clustering, and can get stuck in the local minimum and give the same errors when clustering. We will show problems when we change the initial clustering when we assume there are 3 clusters.





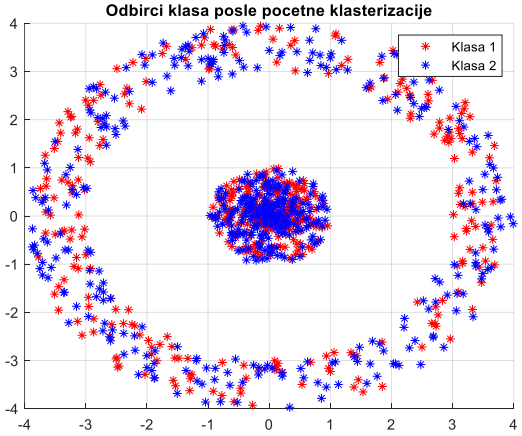
From graphs we also see the similar behavior of the algorithm when we assume the wrong number of classes.

The last requirement of the task is to generate two nonlinear separable classes:



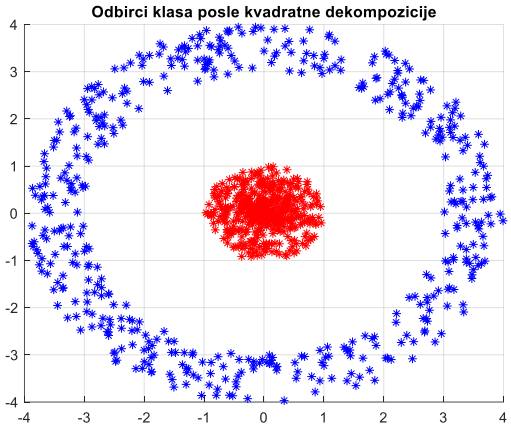
To cluster nonlinear separable clusters, we use **the quadratic decomposition method**.

As with previous algorithms, we first perform the initial clustering:

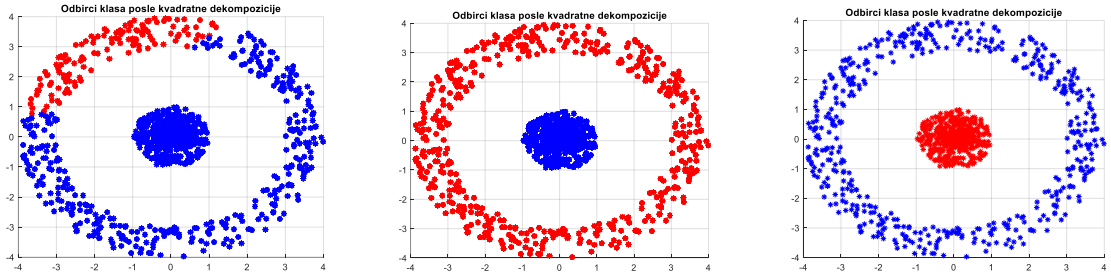


The quadratic decomposition method is executed the same way as the c-mean method, and the only difference is the distance at which it clusters the samples. Unlike the c-mean method, which observes Euclid distance, this method observes the statistical distance of the samples from the clusters and because of this change, it can be used for clustering non-linearly separable classes.

As a result of the clustering, we get:



However, this algorithm, like the previous one, is sensitive to the choice of initial clustering and we can get the same anomalies in clustering, as before:



The mean number of iterations required is **7.8** and we see that it is close to the number we get with the c-mean method.