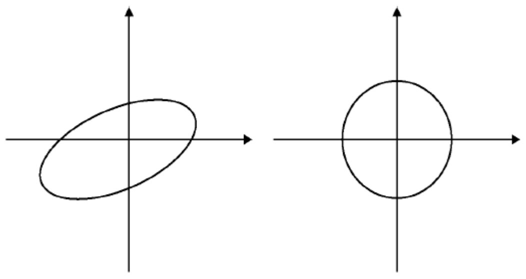
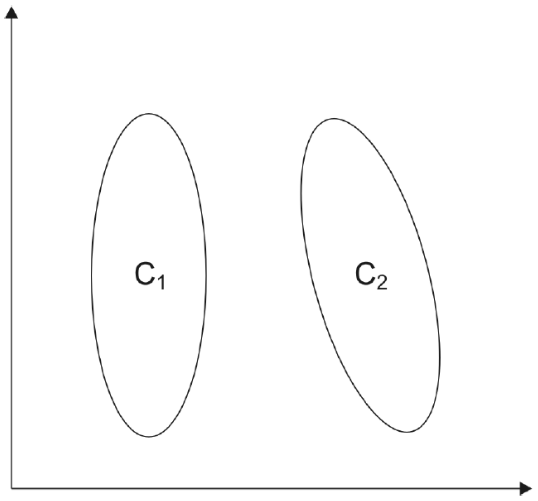
EPART – problems to think through before the first test

1. k-NN classification can be realized in two ways:
   a. Selecting for the classified sample its *k* nearest neighbors **for each class** and computing than neighborhood radius *ri* for each of the classes. As the result of classification we select class *i*, for which its radius *ri* is smallest.
   b. Selecting for the classified sample its *k* nearest neighbors in the training set (without taking class labels into account in this step). As the result of classification we select class *i* which is most numerous in the set of k nearest neighbors of the classified sample.

   Please compare these two classification methods. In what situations can we expect differences in the classification result for a particular set of classified samples?
2. On the first laboratory exercises we implemented quite brutal method to select nearest neighbor.
   What fastest (in the sens of our work– thinking, implementing, etc.) modification can we apply to this code to select k nearest neighbors?
   Having more time (and will) how can we make our solution operating faster?
3. On the left diagram we see contour plot of the probability density of a class (a set of points). Na lewym wykresie widać wykres konturowy warunkowej funkcji gęstości prawdopodobieństwa pewnej klasy (zbioru punktów).

   What can you tell about the mean and covariance matrix of this set of points?
   What operations should we perform on this set of points to get probability density depicted on the right contour plot?
4. We have two classes with normal probability distributions – contour plot is depicted below. On the diagram decision boundary is shown for the equal a priori probabilities and equal costs of the classification decisions.

   Let's assume that coefficients in the covariance matrix are following: $\begin{bmatrix} a & c \\ c & b \end{bmatrix}$ (coefficient *a* is computed for the feature represented on the horizontal axis and *b* for feature represented on the vertical axis).
   Which of the following statements are true for class C_2:

   ☐ a = b ☐ a > b ☐ a < b   ☐ c > 0 ☐ c < 0 ☐ c = 0

   Sketch the decision boundary for *a priori* probabilities $P(C_1)==P(C_2)$ and equal loss values of errors.
   Sketch the decision boundary for *a priori* probabilities $P(C_1) > P(C_2)$ and equal loss values of errors.

5. Linear classifiers differentiate well two classes. In the case when the number of classes is greater than two we use linear classifiers' ensambles. In one vs. one (OVO) strategy individual classifier differentiates between two original classes. In one vs. all (OVA) stategy individual classifier differentiates between one original class and the second class build from the samples of all other classes. Please compare thoroughly these two strategies.
6. In support vector machine polynomial kernel $K(x,y)=(x\bullet y+1)^3$ is used.
   What is the dimensionality of the space, in which linear classification is performed?
   What is the exact form of the transformation from the feature space to the classification space (assuming x and y are two-dimensional)?
7. I have several hundreds of features, each of which used independently in classification gives recognition quality from mean to weak (i.e. not much better than guessing or one, constant answer). To select proper number of features I want to check how correlated are the good and the wrong one-dimensional answers. For each one-dimensional classifier I prepare zero-one vector (error vector) in which zero represents classification error and one correct classification. For each pair of the features I compute number of differences in their respective error vectors. I order the pairs of features with respect to the differences in error vectors in non-increasing order and in this order select the features to be used in final classifier.
   Please indicate the advantages and disadvantages of such a solution.
   How can you improve this solution?
8. We have a training set contaning feature vectors of a few dozens of classes. The set is "sparse", i.e. each class is represented by a dozen (or even a few) samples. 1-NN classifier using the entire training set does not meet our requirements. Please suggest two attempts (other methods of classification) that you undertake you to prepare classifier that meets the requirements. Please provide a brief justification of both the methods and the order of their use (the order is important here).
9. Clustering results are represented in the form of vector containing for each sample in the clustered set a cluster index (this is the cluster to which the sample belongs according to this clustering). Propose effective method of computing Rand index for this representation of clustering results (to make things more concrete let's assume that the number of clusters is less than 50 and samples in the clustered set is around 50000).
10. In naive implementation of k-means algorithm the procedure is as follows:

    ```
    For each sample x from the set:
            Compute distances of x to the means of all clusters
            Find minimum distance and assign point to this cluster (if x was earlier assigned to other cluster mark
            cluster change)
    The above iteration over samples set is repeated until no point changed in the iteration cluster assignment.
    ```

    What possibilities to speed up this procedure do you see?
11. Under the spell of the Kruskal algorithm, I decided to use it to construct a minimum spanning tree for a set of N = 100,000 points. Since my hardware does not allow me to keep all the distances between pairs of points, I decided to define the potential edges of the graph in batches of k * N edges (starting of course with the K * N shortest edges, and subsequent batches to compute only when necessary; prior to checking, it seems to me that k = 5 would be appropriate).
    Please suggest the implementation of the functions determining k * N edges, so that could be used in the Kruskal algorithm.
    What will be the computational complexity of such a method for determining the potential edge of the graph?
    How does this compare to "normal" Kruskal algorithm?