Pattern recognition
Homework Report

Lev Kolezhuk

May 2017

Problem 1.1

Pattern recognition has a really wide range of applications. My first idea that actually led me and one of my friends to conducting some research on the topic of pattern recognition was to distinguish the news articles in the feed by checking weather they contain an overall positive or negative message. Immediately there appeared the problem of extracting features from the data. And this, to my opinion, is the most complex part of developing a pattern recognition algorithm in order to fit one or other task.

Thus, for example in order to determine possible case of retinopathy in a patients eye, we need to extract only the features that actually describe the damage caused, e.g by diabetes. To achieve this we have to segment the image in a way that we obtain parts only corresponding to the retinopathy but we remove other details such as vessels during the segmentation.Besides that we have to characterize the obtained details in a more compact way, such as size, density, location, etc. It is always hard to obtain a set of features which is ideal for the required classification.

Some approaches use automated feature extraction and selection, such as some deep learning methodologies.

Problem 1.2

In order to determine the unknown coefficients k_1 and k_2, we need to take in account that the given distributions are normal distributions, so knowing that

$$f(x, \mu, \sigma) = \frac{1}{\sqrt{(2\pi\sigma^2)}} \exp - \frac{(x-\mu)^2}{2\sigma^2}$$

k_1 and k_2 are equal to $\sqrt{\frac{20}{2}}$ and $\sqrt{\frac{12}{2}}$ accordingly. In order to build the two probability density functions with parameters given in the task, we create a linear space, that will cover the most significant part of the plot from $-max(\mu_1, \mu_2) - 4max(\sigma_1, \sigma_2)$ to $max(\mu_1, \mu_2) + 4max(\sigma_1, \sigma_2)$.

Firstly, lets consider the cases of different relation of prior probabilities

$$\frac{P_2}{P_1} = 1$$

$$\frac{P_2}{P_1} = 0.5$$

$$\frac{P_2}{P_1} = 4$$

The first case means that the two classes have equal weights for the decision boundary and the a posteriori probabilities for the two classes will be equal to the initial probability density functions. The decision boundary of the classifier lies exactly in the intersection of these two pdfs. In other two cases an additional weight is added to the pdfs, so that the final a posteriori probability

$$f_{apost1} = P1 * f_{pdf1}$$
$$f_{apost2} = P2 * f_{pdf2}$$

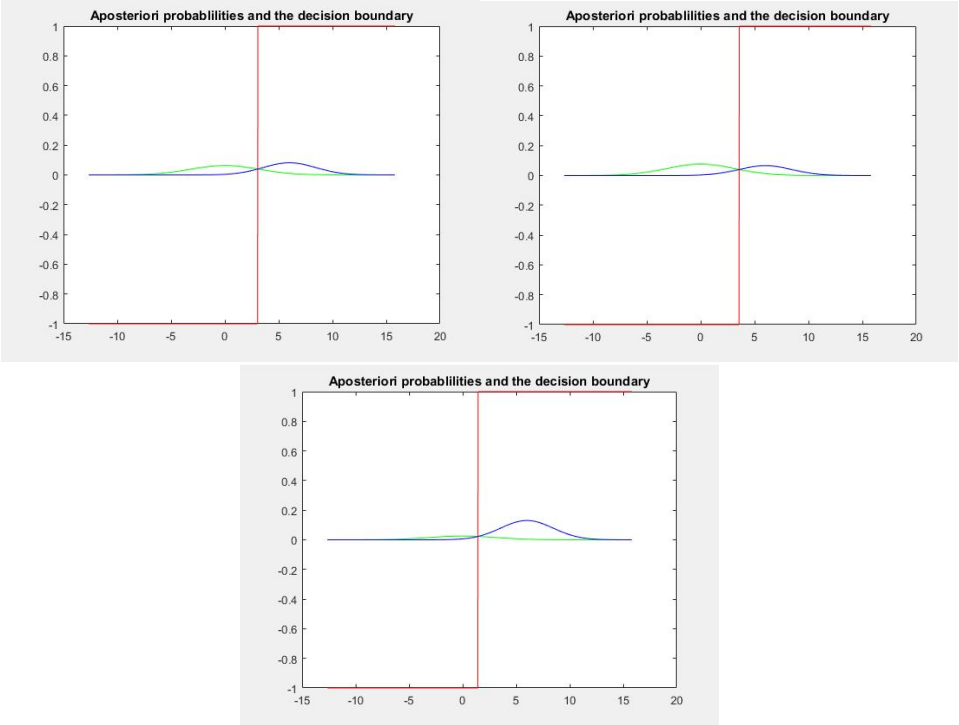

Figure 1: A posteriori probabilities for different prior probabilities

The decision boundary was created with the discriminant function for minimizing the probability of error:

$$g(i) = log(f_{pdf1}(i)) + log(P_1) - log(f_{pdf2}(i)) - log(P_2)$$

This shown decision boundary is the one that minimizes the probability of error as it takes into consideration the prior class probabilities that show how probable is to pick an element of class i from the initial data.

Problem 1.3

Considering a medical diagnosis problem where a biochemical test is used for screening patients. The test returns a result close to 0 for healthy patients and close to 1 for sick patients, according to the following likelihood functions: $p(x|\omega_1) = N(0.0, 0.3)$ and $p(x|\omega_2) = N(1.0, 0.1)$ where $N(\mu, \sigma)$ is the Gaussian univariate density. When we are talking about minimizing the conditional risk in a decision, we have to consider the decision cost matrix, that shows us how costly one or other decision is. In our case we assume that, on average, 1 out of 10,000 patients is sick, and the following costs:

1. $\lambda_{12} = 2000$ euros for not productive test cost in the case of wrongly diagnosed disease;

2. $\lambda_{21} = 1000000$ euros for reimbursement to the patient in the case of not diagnosed disease;

3. $\lambda_{11} = \lambda_{22} = 0$ in the case of correct diagnosis and define the decision rule that minimizes the conditional risk.

For some cases such as classifying the patient as healthy or sick, the costs of taking the decisions can be very different, just like in our case. In order to create a classifier, which is considering decision costs, we may also add it as a certain type of weight to the existing classifier for minimizing average probability of error:

$$g(i) = log(\frac{f_{pdf1}(i)}{f_{pdf2}(i)}) + log(\frac{\lambda_{21} - \lambda_{11}}{\lambda_{12} - \lambda_{22}}) + log(\frac{P1}{P2});$$

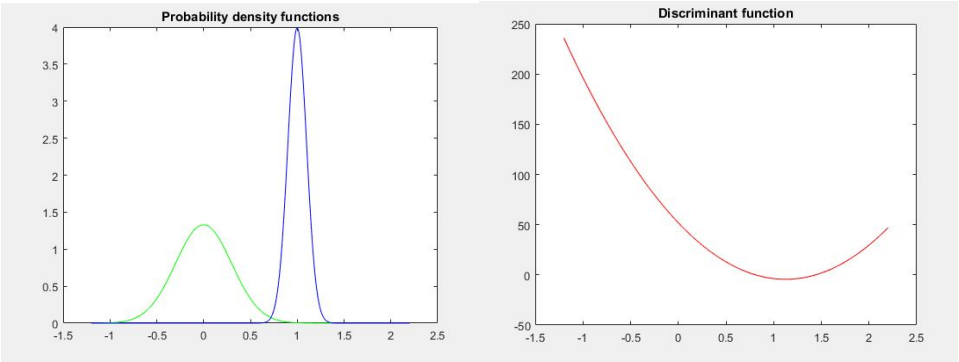

Figure 2: Probability density functions(left) and the discriminant function with considering decision-cost matrix(right)

The decision rule is $g = 0$, so we can see that the class with the extremely

high cost and a quite small prior probability has a very small decision region(only the elements that correspond to the area on the curve which is below zero). This is probably good news for the diagnosed patients.

Problem 1.4.a

I am considering the hw1data.m data-set where every vector has 10 features and a label which provides us information as for to which class the vector really belongs to. We have to split the data-set into training and test sets by randomly selecting 25% of the examples from each class for the test set. In order to manage the random selection, matlab function called *randperm* is used. The proportion of elements belonging to different classes is kept the same in the training subset in order not to change the estimated prior probabilities of selecting an element of a certain class.

In order to implement a linear classifier a discriminant function is used:

$$g(i) = -\frac{1}{2}(x(i) - \mu)\Sigma^{-1}(x(i) - \mu)^{-1} + log(P);$$

Here Σ is the co-variance matrix of the elements in the training set, μ is the mean value in the positive or negative classes of the training set, P is the prior probability of one of the classes. The prior probability can be estimated from the number of elements of each class in the training set and in our case it was 0.5, which means that the classes have equal probabilities of being randomly selected from the training set.

In order to implement the quadratic classifier another discriminant function is used:

$$g(i) = -\frac{1}{2}(x(i)-\mu_m)\Sigma_m^{-1}(x(i)-\mu_m)^{-1} - \frac{1}{2}log(det(\Sigma_m)) - \frac{d}{2}log(2\pi) + log(P)$$

where Σ_m and μ_m are the co-variance matrix and the mean value of class m (positive or negative).

Evaluation of the performance is made by considering the average accuracy of the classifiers through 100 launches of the algorithm on the same data. During these launches the given data-set is split into training and test set in a random way, so we can make sure that a current setting is not influencing the accuracy of the algorithm. The accuracy is calculated with a preset FPR of 0.1. The average accuracies of the Linear classifier and the Quadratic classifier were:

$$A_{lin} = 0.5488$$

$$A_{quad} = 0.6135$$

Problem 1.4.b

Now we proceed with building a knn classifier. This classifier uses the training set to determine how close is the selected vector to elements of the positive and negative classes. The distances were chosen to be Euclidean distances. In order to create the best classifier, we need to select the k amount of points, which provides the best TPR for the selected FPR of 0.1. In order to do this we create two loops - the outer loop changes the value of k in the classifier, the other one - provides a certain amount of iterations (was chosen as 100) to run the algorithm as well as the splitting into training and test set, to ensure, that the TPR that we obtain is a stable average value and it doesn't depend too much on the splitting of the data set.

After running the algorithm mentioned above, first of all, I could clearly see that the maximum variance of the TPR throughout all iterations is quite small (less than 1%), which, to my opinion, is normal, taking to account different data set splitting.

The other observation is that the best algorithm performance (maximum TPR for 0.1 FPR) is when k is around 30-40. It means that after this values we may consider the algorithm to be overfitting, under that value - underfitting. In order to implement the best setting for the classifier, we can choose.

Problem 1.5

We assume that the given data set is an extension to the one provided in the homework, so we can use the existing data set to train the classifier and that the obtained optimal values for k of knn that we have determined in the previous task. It means that we use 100% of our given data set as training and the input matrix A is fully used as a test set.

There is also a setting for maximum allowed FPR (by default 0.1) in the results of the algorithm. This sets a working point on the ROC curve. This value can be changed in the first lines of the test.m function.

Code and implementation

The code for all tasks is attached in the archive together with the code. It is separated by tasks. The files $main[taskname].m$ correspond to each of the

5 tasks in the sheet.

References