ECONTECHMOD. AN INTERNATIONAL QUARTERLY JOURNAL – 2017. Vol. 6. No. 2. 15–20

Building computer vision systems using machine learning algorithms

N. Boyko, N. Sokil

Lviv Polytechnic National University, Lviv, Ukraine; e-mail: nataliya.i.boyko@lpnu.ua

Received March 1.2017: accepted June 2.2017

Abstract. In this paper theoretic aspects of machine learning system in the field of computer vision is considered. There are presented methods of behavior analysis. There are offered tasks and problems associated with building systems using machine learning algorithm. The paper provides signs of problems that can be solved by using machine learning algorithms There is demonstrated step by step construction of computer vision system. The paper provides the algorithm of solving the problem of binary (two classes) classification for demonstration the machine learning algorithm possibilities in image recognition field, which can recognize the gender of the person on the photo. Aspects related to the search of data processing are also considered. There is analyzed the search of optimal parameters for algorithms. An interpretation of results in machine learning algorithm is provided. Binarization methods in machine learning algorithm are offered. There is analyzed the technology for improving the accuracy of machine learning algorithm. There are proposed ways to improve computer vision system in neural systems. Also there are analyzed large software modules that work using machine learning systems. The article provides prospects of powerful information technologies, which are necessary for the proper data selection in learning and configuration of feature extraction algorithm to create a computer vision system.

Key words: algorithm, information system, neural network, machine learning, client-server architecture, script, artificial system, machine learning algorithm.

INTRODUCTION

Behavioral analysis techniques require significant computing resources. To solve this kind of problem statistical methods are used. However, that requires large signature base, which should be constantly updated. To get rid of these disadvantages the machine learning method is applied [1, 5, 20].

Machine learning is the subfield of computer science that gives computers the ability to learn without being explicitly programmed. Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data driven predictions or decisions, through building a model from sample inputs [6, 8, 15].

This method is a generalized name of artificial generation of knowledge from experience. Artificial system learns from examples and with the end of the learning phase it can make decisions on its own. It compares the incoming information flow with known data in statistical algorithms and detect certain regularities in the learning data [3]. All this is done with the help of computer vision – theory and technology of creating machines that can perform detection, tracking and classification of objects [9].

The main purpose of this work is to study the methods and problems of creating computer vision systems that affected the relevance of the topic and review the key points in practice.

RELEVANCE AND CLASSIFICATION OF MACHINE LEARNING ALGORITHMS

Nowadays artificial intelligence systems are extremely promising subfield of computer science. Machine learning algorithms have proved to be one of the most effective in developing such systems. Their usage area is exceptionally wide. These systems are being used for pattern recognition in different automated systems, for example unmanned transport, faces recognition in video surveillance and social networks, complex system for processing other graphical data, optimization problems of client-server architecture and much more. In total such systems present a whole new level of automation, both in everyday life and in the area of complex scientific problems [7, 10, 18].

Typical problems solved with machine learning:

- **classification** (using supervised learning) – formal problem, defining a set of objects (situations), divided into separate *classes*;
- **clustering** (unsupervised learning) – a set of inputs is to be divided into groups, that are, unlike in classification, are not known beforehand;
- **regression analysis** – a statistical process for estimating the relationships among variables;
- data dimensionality reduction;
- others.

In this work machine learning is being researched as a basis for computer vision systems along with the task and problems that may arise during the developing of such systems.

PRACTICAL RESEARCH

To demonstrate the power of machine learning in the area of image recognition in this research we are going to develop a small system that will be able to determine a person's gender based on a photo. This means that we are dealing with a binary (two-class) classification problem.

As a development instrument we will be using Python programming language, which provides an ability to work with different algorithms on a rather high level and is in fact one of the best choices for programming machine learning systems [2, 12, 14].

Before starting to deal with machine learning it is crucial to collect enough data to be used as an input for a classifier. There were 1000 pictures collected totally, half of which depict women and the other half – men.

Let's collect the paths to images:

visor.py:

```
M_PATH = 'D:\\gen_visor\\images\\men\\'
W_PATH = 'D:\\gen_visor\\images\\women\\'
TEST_PATH =
'D:\\gen_visor\\images\\unlabeled\\'
```

Now our goal is to mark all the images (assign them to different classes). All the images that represent males will be marked as 0, females – 1.

visor.py:

```
males = [M_PATH + f for f in
os.listdir(M_PATH)]
m_labels = [0 for m in males]

females = [W_PATH + f for f in
os.listdir(W_PATH)]
f_labels = [1 for f in females]
```

Same procedure has to be executed on our test data. Later we would presume this data is sorted, so we are going to make sure of that with a simple function called get_pic_index(file).

visor.py:

```
import src.util as util
test = sorted([TEST_PATH + f for f in
os.listdir(TEST_PATH)],
key=lambda p: util.get_pic_index(p))
images = males + females
labels = m_labels + f_labels
test_labels = [1 if util.get_pic_index(x) < 113
else 0 for x in test]
```

util.py:

```
import ntpath
import os
def get_pic_index(path):
   return
int(ntpath.basename(os.path.splitext(path)[0]))
```

Now we import the *numpy* library and transform our data into an array provided by this library. Numpy significantly extends Python's features of processing collections and is a de-facto standart library when dealing with large data sets.

visor.py:

```
import numpy as np
    labels = np.array(labels)
```

Now the images have to be transformed to ensure effective classification. All the code that deals with this task resides in another file called transform.py. The convert function is a façade for working with this file.

transform.py:

```
def convert(img_array):
   result = []
   for image in img_array:
      img = img_to_matrix(image)
      result.append(img)
   return np.array(result)
```

First of all we need to convert an image file into a matrix of its pixels. To achieve this we are using *mahotas* – a library developed for image processing.

transform.py:

```
import mahotas as mh
def img_to_matrix(filename):
image = mh.imread(filename)
```

Now we are going to use an interesting technique. Since it is safe to guess that a person's face is situated in the center of an image, we should make the algorithm's work easier and make sure that is the area where it will be looking for data. To achieve this we will blur the edges of the picture placing its center in focus. To blur the image we are applying Gaussian filter to each of the image's channels (red, green and blue).

Gaussian blur (also known as **Gaussian smoothing**) is the result of blurring an image by a Gaussian function. It is a widely used effect in graphics software, typically to reduce image noise and reduce detail [1, 13, 18].

transform.py:

```
def center_img(image):
    r, g, b = image.transpose(2, 0, 1)
    r12 = mh.gaussian_filter(r, 12.)
    g12 = mh.gaussian_filter(g, 12.)
    b12 = mh.gaussian_filter(b, 12.)
    im12 = mh.as_rgb(r12, g12, b12)
    h, w = r.shape
    Y, X = np.mgrid[:h, :w]
    Y = Y - h/2
    Y = Y / Y.max()
    X = X - w/2
    X = X / X.max()
    C = np.exp(-2.*(X**2 + Y**2))
    C = C - C.min()
    C = C / C.ptp()
    C = C[:, :, None]
    return mh.stretch(image*C + (1 - C)*im12)
```

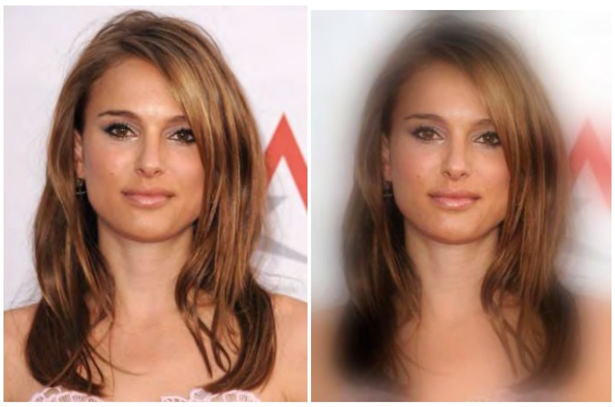

Fig. 1. Process of centering

The information about colors is unnecessary in this case, so it is safe to remove it and use black-and-white images. Additional we can blur the whole image. From a point of view of a human that is a strange idea, because it is definitely harder for our eyes to see the content of a blurred image. Nevertheless blurring should remove a lot of noises and it would improve the results of image binarization.

transform.py:

```
def img_to_matrix(filename):
    image = mh.imread(filename)
        image = center_img(image)
        image = mh.colors.rgb2gray(image,
    dtype=np.uint8)
        image = mh.gaussian_filter(image, 4)
```

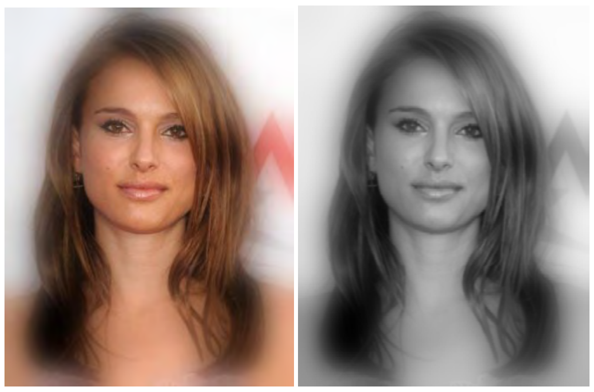

Fig. 2. Colors removed and additional filter applied

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white, though any two colors can be used. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.

BINARIZATION TECHNIQUES IN MACHINE LEARNING ALGORITHM

Otsu's method is used to automatically perform clustering-based image thresholding, or, the reduction of a gray-level image to a binary image. The algorithm assumes that the image contains two classes of pixels following bi-modal histogram (foreground pixels and background pixels), it then calculates the optimum threshold separating the two classes so that their combined spread (intra-class variance) is minimal, or equivalently (because the sum of pairwise squared distances is constant), so that their inter-class variance is maximal [8–12].

Let's compute Otsu's threshold and save the binary image.

transform.py:

```
thresh = mh.thresholding.otsu(image.astype(np.uint16))
    otsu_img = image > thresh
    mh.imsave('remove.jpg',
    255*otsu_img.astype(np.uint8))
```

Now we will open this temporary image with PIL library and resize it to match a defined common size. We are using size STANDARD_SIZE = (170, 300). Naturally, the images that have significantly different aspect ratio will get malformed. That is why in machine learning it is crucial to spend some time and effort on input data pre-processing and preparation. Ideally we should have a set of images where all of them are of the same size or we would have to use a more complex algorithms to convert them to this size without the risk of loosing large amounts of information.

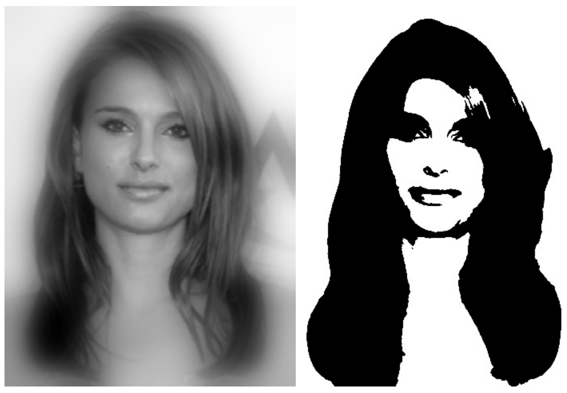

Fig. 3. Image binarization

transform.py:

```
from PIL import Image
image = Image.open('remove.jpg')
image = image.resize(STANDARD_SIZE)
res = np.array(list(image.getdata()))
return res
```

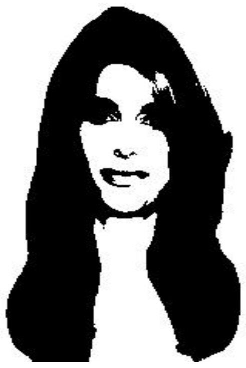

Fig. 4. Final image

Now let's visualize our data reducing it to 2 dimensions.

util.py

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

    def plot_graph(data_array, labels):
    pca = PCA(n_components=2)
    X = pca.fit_transform(data_array)

    colors = ['red', 'green']
    tags = ['female', 'male']
    plt.figure()
    for color, tag, i in zip(colors, tags, [0, 1]):
     plt.scatter(X[labels == i, 0], X[labels == i, 1],
color=color, label=tag)

             plt.legend()
             plt.show()
```

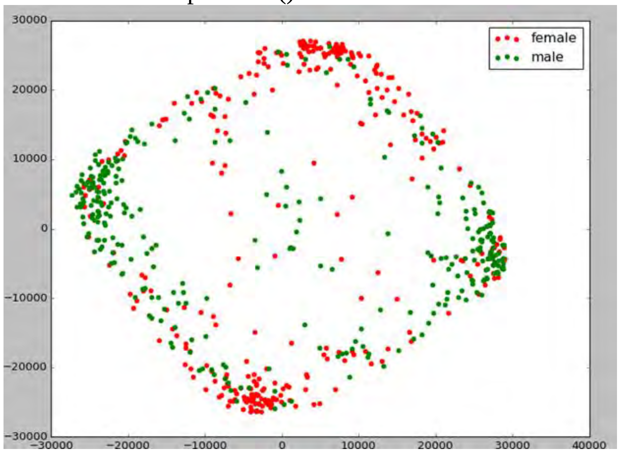

Fig. 5. Visualization (600 samples)

As of this moment we have an array of marked images, which is enough to start training a classifier with any machine learning algorithm. However each image is represented by an array of size (170, 300). This means that the algorithm will have to deal with 51000 features, which may be pretty much excessive and such computations would require lots of resources [4, 16, 17].

Based on this we are facing a problem of dimensionality reduction, just as it was during the visualization step. To achieve this, we will be using Principal Component Analysis – a method of factor analysis in statisticsб which is applied to reduce the dimensionality of data with minimal loss of information.

The principle of the method – for a given set of vectors $x_1, x_2, \dots, x_m \in R^n$ for each $k = 0,1\dots,n-1$ there is a need to find $L_k \subset R^n$ that the sum of the squared distances from x_i to L_k will be minimal:

$$\sum_{i=1}^{m} dist^2(x_i, L_k) \to \min \qquad (1)$$

Let us reduce our data set to 10 dimensions.

File visor.py

```
    pca = PCA(n_components=10)
    data = pca.fit_transform(data)
```

Now the data is ready for processing and we should provide a classifier. For now let us use the most simple algorithm possible – the k-nearest neighbors algorithm. It is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. *k*-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/*d*, where *d* is the distance to the neighbor.

File knn.py

```
from scipy.spatial.distance import euclidean
class MyKnn:
  def fit(self, X_train, y_train):
    self.X_train = X_train
    self.y_train = y_train

  def predict(self, X_test):
    predictions = []
    for row in X_test:
      label = self.closest(row)
      predictions.append(label)
    return predictions

  def closest(self, row):
    best_dist = euclidean(row, self.X_train[0])
    best_index = 0
    for i in range(1, len(self.X_train)):
      dist = euclidean(row, self.X_train[i])
      if dist < best_dist:
        best_dist = dist
        best_index = i
    return self.y_train[best_index]
```

Training the classifier:

File visor.py

```
knn = MyKnn()
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(data, labels)
```

Now we are creating an associative array (a dictionary) with filenames from testing data set as keys and image data as values. Using this map we are getting the result of classification.

File visor.py

```
test_map = dict(zip(test, pca.fit_transform
(tf.convert(test))))
predicted = {}
for k, v in test_map.items():
  r = knn.predict(v.reshape(1, -1))
  predicted[util.get_pic_index(k)] = r

predicted_labels = []
for key in sorted(predicted):
  predicted_labels.append(predicted[key])
```

To assess the precision of our computations we should compare it to the manually classified values.

File visor.py

```
import sklearn.metrics as metrics

print(str(metrics.accuracy_score(test_labels,
predicted_labels) * 100) + '%')
```

Console output:

```
C:\Python35\python.exe D:\gen_visor\src\visor.py
59.52138261273%
```

This precision is pretty high for such a simple algorithm (random classification would give us 50%), but there is still much more to be desired.

How can we possibly improve the results? Our classifier was using only the nearest neighbor to plot its predictions (k = 1). Let's make use of the provided classifier from ***sklearn*** package that is more configurable. We should also test the other algorithms implementations from this library.

The class sklearn.tree.DecisionTreeClassifier is based on the so called *decision tree*, that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Decision trees are commonly used in operations research and operations management. If in practice decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

The class sklearn.svm.SVC is a classifier that implements a *support vector machine*. In machine learning, support vector machines (SVMs, also support vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space.

The resulting algorithm is formally similar, except that every dot product is replaced by a nonlinear kernel function. This allows the algorithm to fit the maximum-margin hyperplane in a transformed feature space. The transformation may be nonlinear and the transformed space high dimensional; although the classifier is a hyperplane in the transformed feature space, it may be nonlinear in the original input space.

It is noteworthy that working in a higher-dimensional feature space increases the generalization error of support vector machines, although given enough samples the algorithm still performs well.

The most common kernels:

- Polynomial: $k(x,x') = (\langle x,x' \rangle + const)^d$;
- Radial basis function:
$$k(x,x') = e^{g\|x-x'\|^2}, g > 0;$$
- Gaussian radial basis function:
$$k(x,x') = e^{\frac{\|x-x'\|^2}{2s^2}};$$
- Sigmoid:
$$k(x,x') = \tanh(k\langle x,x' \rangle + c), k > 0, c < 0.$$

In our case we will be using a radial basis function, which is a set of interpolation methods. A radial basis function (RBF) is a real-valued function whose value depends only on the distance from the origin, so that; or alternatively on the distance from some other point *c*, called a *center*, so that $f(x,c) = f(\|x-c\|)$. Sums of radial basis functions are typically used to approximate given functions.

visor.py

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

knn = KNeighborsClassifier(n_neighbors=5)
dtc = DecisionTreeClassifier(max_depth=4)
svc = SVC(kernel='rbf')
# ...
```

Console output:

```
C:\Python35\python.exe D:\gen_visor\src\visor.py
K nearest neighbors: 61.14035087719%
Decision tree: 62.2807017544 %
Support vector: 90.350877193%
```

Based on the above we can see that the implementation of the support vector machines was the most accurate in its classification.

Full source code can be found here: https://github.com/nestorsokil/gen_visor

CONCLUSIONS

In this research a computer vision system was developed and all the aspects of building such system were examined, including collecting the data, its preparation and transformation, finding the optimal algorithm parameters and results interpretation.

The main purpose was to investigate the methods and problems that occur during the development and to explore the topicality of this issue in the modern world.

There are many ways to improve the developed system. First of all it is the more thorough collection of input data (ideally one should have a significant amount of high resolution images to develop and make use of such system). Obviously, this would require a lot more resources both computational and temporal. It is for this reason that large program modules that are based on machine learning algorithms are executed on powerful computers with high-end graphics processors to be able to process big sets of image data on high speed. These costs are really necessary when the accuracy is important, because the correct data and configuring the algorithm to improve feature extraction is key to success when building computer vision system.

REFERENCES

1. **Boyko N. 2016** Basic concepts of dynamic recurrent neural networks development / N. Boyko, P. Pobereyko // ECONTECHMOD : an international quarterly journal on economics of technology and modelling processes, Lublin: Polish Academy of Sciences, Vol. 5, No. 2, pp. 63–68.
2. **Coelho L. 2013** Building Machine Learning Systems with Python / Luis Pedro Coelho, Willi Richert, Birmingham – Mumbai: Published by Packt Publishing Ltd., 290 p.
3. **Bishop C. M. 2006** Pattern recognition and machine learning / Christopher M. Bishop, Springer Science+Business Media, LLC, 78 p.
4. .**Lytvyn V. 2012** Method of automation building and evaluation of data knowledge quality. / V. V. Lytvyn, M. J. Hopyak, A. B. Demchuk // Automation system. Harkiv : XNYRE, No. 161, pp. 62–69 (in Ukrainian).
5. **Demchuk A. B. 2011** The method of ontological agent building on subject area A. B. Demchuk, V. V. Lytvyn, M. N. Voychyshen // Informational systems and networks. Lviv: Lviv Polytechnic Publishing House, No. 715, pp. 215–225 (in Ukrainian).
6. **Palagin A. V. 2006** The architecture of ontological computer systems / A. V. Palagin// Cybernetics and system analysis. – Moscow: Cybernetics and system analysis, No. 2, pp. 111–125 (in Russian).
7. **Nivikov P. S. 1973** Basis in logic, 2 edition/ P. S. Novikov, Moscow : Nauka, 400 p. (In Russian).
8. **Gilbert D. 1947** The basis of theoretical logic / D.Gilbert, V. Akkeman. Moskva: GIIL, 302 p. (in Russian).
9. **Elkan C. 2003** Using the triangle inequality to accelebrate k-means / C. Elkan // In Proceedings of the Twelfth International Conference on Machine Learning, pp. 147–153.
10. **Demchuk A. B. 2014** Videocontent for the blind: the method tyflokomentuvannya / A. B. Demchuk // Radioelektronika, informatyka, upravlinnya, No. 1 (30), pp. 146–149 (in Ukrainian)
11. **Matov O. Ia. 2009** Modern technologies of information resources integration / O. Ia. Matov // Registration, storage and processing of data, Vol. 11, No. 1, pp. 33–42.
12. **Khramova I. O. 2009** The use of service-oriented architectures in the integration of information resources / I. O. Khramova // Registration, storage and processing of data, Vol. 11, No. 2, pp. 70–76.
13. **Matov O. Ia. 2009** Mathematical models of conflict losses performance of the mediators ontology for General use in GRID environment / O. Ia. Matov // Registration, storage and processing of data, Vol. 11, No. 3, pp. 18–25.
14. **Matov O. Ia. 2007** The problem of horizontal integration of information resources in a multi-tiered organizational structures with dynamic configuration / O. Ia. Matov // Registration, storage and processing of data, Vol. 9, No. 3, pp. 88–97.
15. **Matov O. Ia. 2006** Dynamic integration of information resources of the unified information infrastructure of the electricity market / O. Ia. Matov // The functioning and development of electricity and gas markets: collection of scientific works Institute of modelling in energy im. H. Ie. Pukhova, pp. 93–98.
16. **Matov O. Ia. 2006** Model performance the operating nodes of the information infrastructure of corporate information systems in the field of electricity / O. Ia. Matov // Information technology in power engineering: collection of scientific works Institute of modelling in energy im. H. Ie. Pukhova, pp. 95–105.
17. **Matov O. Ia. 2006** The organization of ontologies in common use in the integrated information infrastructure preparation of data for decision-making / O. Ia. Matov // The functioning and development of electricity and gas markets: collection of scientific works Institute of modelling in energy im. H. Ie. Pukhova, pp. 99–103.
18. **Matov O. Ia. 2005** The problem of the use of GRID technologies as the basis of integration of information and analytical resources to support processes of electronic control / O. Ia. Matov // Proceedings of the Academy of engineering Sciences of Ukraine, No. 2 (25), pp. 82–89.
19. **Boyko N. 2016** A look trough methods of intellectual data analysis and their applying in informational systems / Nataliya Boyko // Computer sciences and information technologies CSIT 2016: Proc. of XI International scientific practical conference CSIT 2016: proceedings, Lviv: Lviv Polytechnic Publishing House, pp. 183–185.
20. **Boyko N.I. 2016** Data processing technologies in dynamic systems / N. I. Boyko // Modern problems of applied mathematics and informatics., Lviv: Lviv National University named by Ivan Franko, pp. 37–40 (in Ukrainian).