SECURITY AND COMMUNICATION NETWORKS
Security Comm. Networks 2013; **6**:115–128
Published online 18 April 2012 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/sec.553

RESEARCH ARTICLE

Feature-based Type Identification of File Fragments

Mehdi Chehel Amirani[1], Mohsen Toorani[2]*, and Sara Mihandoost[1]

[1] Department of Electrical Engineering, Urmia University, Urmia, Iran
[2] Department of Informatics, University of Bergen, Bergen, Norway

ABSTRACT

Digital information is packed into files when it is going to be stored on storage media. Each computer file is associated with a type. Type detection of computer data is a building block in different applications of computer forensics and security. Traditional methods were based on file extensions and metadata. The content-based method is a newer approach with the lowest probability of being spoofed and is the only way for type detection of data packets and file fragments. In this paper, a content-based method that deploys principle component analysis and neural networks for an automatic feature extraction is proposed. The extracted features are then applied to a classifier for the type detection. Our experiments show that the proposed method works very well for type detection of computer files when considering the whole content of a file. Its accuracy and speed is also significant for the case of file fragments, where data is captured from random starting points within files, but the accuracy differs according to the lengths of file fragments. Copyright © 2012 John Wiley & Sons, Ltd.

KEYWORDS
computer security; content-based type detection; computer files; principle component analysis (PCA); support vector machine (SVM); MLP classifier

***Correspondence**
Mohsen Toorani, Department of Informatics, University of Bergen, P.O. Box 7803, 5020 Bergen, Norway.
E-mail: mohsen.toorani@ii.uib.no; ResearcherID: A-9528-2009

1. INTRODUCTION

Computers deal with a huge number of file formats that transmit between networks. Without correct file type detection, the security in computers and networks will not be achievable. File type detection is a building block in proper functionality of operating systems, firewalls, intrusion detection systems, antiviruses, filters, and steganalysis. It is also an important topic in computer forensics and disk carving applications where not only type identification of a computer file as a whole but also type detection of file fragments is important. Computer files are indeed stored at random places in storage media, and some parts of deleted files may be rewritten by other data.

Although there are many applications dealing with type detection of computer files, there are very limited methods for file type detection. Methods of file type detection can be categorized into extension-based, magic bytes-based, and content-based [1,2]. Extension-based and magic bytes-based methods are traditional methods that are extensively used in Windows and UNIX operating systems, respectively. The content-based approach is still an active area for research. It is the only way for type detection of spoofed files whose contents do not match with their claimed types.

Each computer byte consists of 8 bits, so it can take 256 different values. The byte frequency distribution (BFD) of a file can be simply calculated by reading the contents of a file and counting the number of occurrences for each byte value. It is believed that different files of the same type have the same characteristics that can be used for file type detection. In the content-based file type detection, several sample files of the same type are taken, and a fileprint, something similar to a fingerprint, is generated from the sample files. Whenever unknown data are examined, their fileprints are generated with the same procedure and will be compared with the collection of previously generated fileprints. For generating a fileprint, some major features of each file type should be selected or extracted. There are some methods that can be used for the feature extraction. The original principle is to use the BFD of file contents and manipulate its statistical features. Such statistical measurements together form a model of the chosen file type, sometimes called a centroid. The centroids are then compared with an unknown sample file or data fragment, and distances between the sample and centroids are calculated. If the shortest distance is lower than a predefined threshold, the examined file is categorized as being of the same type as the corresponding centroid [2].

The literature in the field of content-based type detection can be divided into two: considering the whole or truncated contents of files including file headers [2–8] and considering file fragments or data packets irrespective of file headers [9–15]. The second approach has an obvious advantage over the first approach as it can be used not only for type detection of computer files but also for type detection of data packets and file fragments, and it has many applications in network security and computer forensics. Computer files are divided into packets before transmission through computer networks, and they arrive randomly at the destination. Then, the first approach does not work for applications dealing with controlling data links. In computer forensics and disk carving applications, different portions of computer files, which are actually file fragments, are extracted from storage media.

In this paper, a content-based type detection method that deploys principle component analysis (PCA) and unsupervised neural networks for an automatic feature extraction is proposed. The extracted features are then applied to a classifier for the type detection. In [2], it is shown that the proposed method works well for type detection of computer files when considering the complete file and using a multilayer perceptron (MLP) classifier. In this paper, earlier results are confirmed with further experiments. It is also shown that the proposed scheme can be used for type detection of file fragments taken from random places within files. The accuracy and speed can also be improved by using a support vector machine (SVM) classifier. The rest of this paper is organized as follows. Section 2 briefly describes the fundamentals of PCA and unsupervised neural networks that will be used for the feature extraction in the proposed method. The proposed method is described in Section 3, and simulation results are presented in Section 4. Finally, Section 5 gives the conclusion.

2. PRELIMINARIES TO FEATURE EXTRACTION AND CLASSIFIERS

Feature extraction refers to the optimal linear or nonlinear combination of statistical features. It creates a smaller set from the original features. It reduces the number of features in a dataset while retaining most of the intrinsic information contents of the dataset. The simplest way to employ is feature selection that chooses an effective subset of the original features.

Feature extraction is an important task for machine learning applications [16]. In this study, PCA + MLP are used for this purpose. The experimental results will show that this combination has better performance than that obtained by other techniques. In this section, the concepts of PCA and unsupervised neural networks that are used for the feature extraction are briefly described. Then, MLP and SVM classifiers are introduced.

2.1. Principal component analysis

The PCA is a well-known feature extraction technique for dimension reduction. It is also known as the Karhunen–Loeve transform [17]. It is an orthogonal transformation of the coordinates in which the data are described. A small number of principle features are usually sufficient to describe the data. Let $\mathbf{X} = \{\mathbf{x}_n \in R^d | n = 1, \ldots, N\}$ represent a d-dimensional dataset. The PCA tries to find a lower dimensional subspace to describe the original dataset whilst preserving the information as much as possible. Then, a new k-dimensional dataset $\mathbf{Z} = \{\mathbf{z}_n \in R^k | n = 1, \ldots, N\}$ will be generated in which k is smaller than d. The orthogonal basis of the feature space is defined as eigenvectors of the total class scattering matrix,

$$\mathbf{S}_t = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^T \tag{1}$$

where $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N}\mathbf{x}_n$ is the mean vector. The eigenvectors $\underline{\Psi}_n$ and eigenvalues $\lambda_n, \{\lambda_n | n = 1, 2, \ldots, N\}$, of the scattering matrix are calculated as:

$$S_t.\underline{\Psi}_n = \lambda_n.\underline{\Psi}_n \tag{2}$$

By ordering the eigenvalues ($\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k \geq \cdots \geq \lambda_N$), k largest eigenvalues are selected. The transform matrix (A) of the k most significant eigenvectors is:

$$A = (\Psi_1 \Psi_2 \cdots \Psi_k)^{\mathrm{T}} \tag{3}$$

Such projection results in a vector containing k coefficients $a_1, \ldots, a_k$. Hereafter, the vectors are represented by a linear combination of eigenvectors having weights $a_1, \ldots, a_k$. The dimensionality reduction error of the PCA can be evaluated as [18]:

$$E_k = \frac{1}{2}\sum_{i=k+1}^{d}\lambda_i \tag{4}$$

Expression (4) indicates that the introduced error E_k can be eliminated by choosing an appropriate dimensionality reduction.

2.2. Unsupervised neural networks

In recent years, neural computing has emerged as a practical technology with successful applications in many fields. Much work has been done to use neural networks for the feature generation and feature selection. A neural network is a massively parallel distributed processor that resembles the brain in two aspects [19]:

(1) Knowledge is acquired by the network through a learning process.
(2) Interconnection strengths known as synaptic weights are used to store the knowledge.

In fact, learning is a process by which the free parameters of a neural network, i.e. synaptic weights and bias levels, are adapted through a continuing process of stimulation by the environment in which the network is embedded. The type of learning is determined by the manner in which the parameter changes take place. This form of learning assumes the availability of a labeled set of training data, made up of N input–output examples:

$$O = \{(x_i, d_i)\}_{i=1}^{N} \tag{5}$$

where x_i is an input vector of the ith example, d_i is the desired response of the ith example, and N represents the sample size. Given the training sample O, the requirement is to compute the parameters of the neural network so that the actual output of the neural network, y_i, is statistically close enough to d_i with respect to x_i for all values of i. Weights and biases are iteratively updated until $E(n)$, the mean square error, is minimized.

$$E(n) = \frac{1}{N}\sum_{i=1}^{N}(d_i - y_i)^2 \tag{6}$$

In this way, outputs approximate the inputs as closely as possible. In order to improve the performance of the system, the output of PCA is fed to an MLP neural network. It has been used as a nonlinear feature extraction system. In this case, a five-layer MLP is formed in which the desired outputs are the same as the inputs. Figure 1 illustrates the architecture of a feed-forward auto-associative neural network with five layers. Outputs at the third layer, which are compressed data, are called the nonlinear principal components and may be considered as a nonlinear generalization of the principal components. During the training process, one attempts to match the outputs with the class number of the labeled inputs. When the training process is complete, the outputs of the hidden layers are utilized as the compressed data or new reduced features.

2.3. Multilayer perceptron classifiers

Multilayer perceptron classifiers are a well-known classifier type that will be used in our experiments. The intricacy of the neural network is characterized by the number of hidden layers. There is no general rule for selecting an appropriate number of hidden layers. An MLP with a small number of neurons may not be adequately powerful to model a complex function. Furthermore, a neural network with too many neurons may lead to overfitting of the training sets and lose its generalization ability, which is one of the main desirable characteristics of an MLP [20]. The most common approach for determining the optimal number of hidden layers is by trial and error. This approach will be considered in our experiments later in Section 4.2.

2.4. Support vector machine classifiers

The basic SVM classifier is similar to MLP classifier, as both of them are linear classifiers. However, the SVM classifier has some advantages over the MLP classifier. With the use of standard optimization software, a unique global optimum for its parameters can be discovered [21]. Moreover, SVM utilizes a structural risk minimization principle, whereas in neural networks, empirical risk minimization is utilized for minimizing the training error.

The SVM classifier prefers one special solution. This classifier separates classes with the maximum margin, between the two nearest data points that belong to two separate classes. Figure 2 depicts a linear SVM for separating two classes, which are represented by circles and rectangles. Figure 2 contains three hyperplanes $w^{\mathrm{T}}x_i + b = 0$, $w^{\mathrm{T}}x_i + b = 1$, and $w^{T}x_i + b = -1$, which have been shown by a solid line and dash lines, in which b is the bias and $\vec{w}$ is the weight vector. Suppose (x_i, y_i) represents the training set in which $x_i \in R^d$, $y_i \in \{1, -1\}$, and $i = 1, \ldots, l$. The training set can be separated by several hyperplanes. If a hyperplane

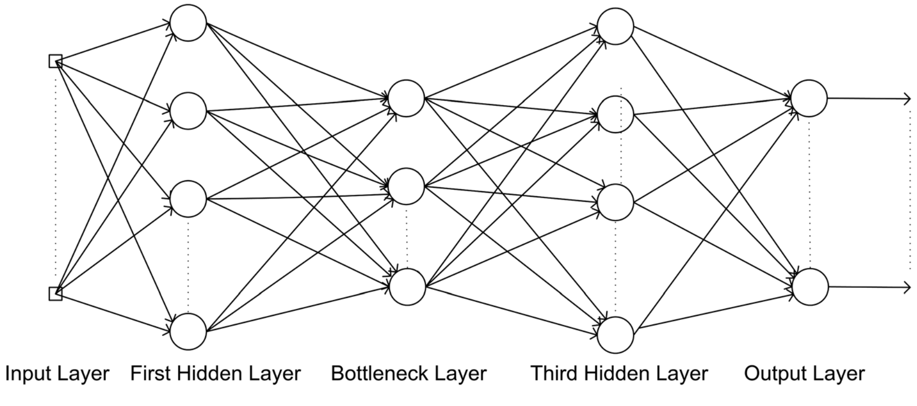

Figure 1. A feed-forward auto-associative neural network.

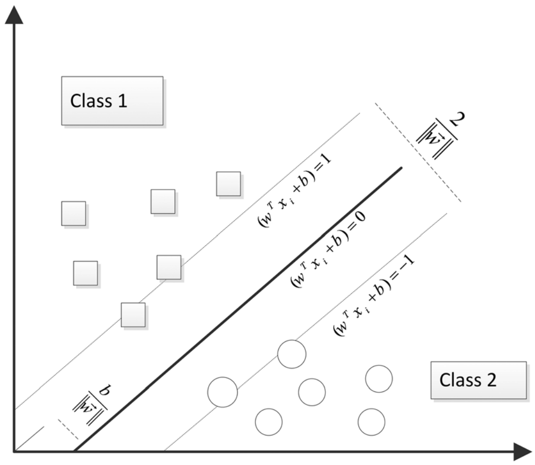

Figure 2. The linear support vector classifier for separating two classes.

maximizes the margin, then the following inequality should be valid for all input data:

$$y_i\left(w^{\mathrm{T}} x_i + b\right) \geq 1 \quad \text{for all } x_i \quad i = 1, 2, \ldots, l \tag{7}$$

The margin of the hyperplane equals to $\frac{2}{\|\vec{w}\|}$, where $\|\vec{w}\|^2 = w \cdot w$. Thus, the margin can be minimized by minimizing $\|\vec{w}\|^2$. It can be demonstrated that this special solution has the highest generalization ability [21].

Indeed, the SVM contracts a hyperplane or set of hyperplanes in a high or infinite dimension space [22]. There are four basic kernel functions: linear, polynomial, radial basis function, and sigmoid. For example, in the linear SVM, points x of the feature space are mapped into the hyperplane spaces, and the kernel function $K\left(\vec{x_i}, \vec{x_j}\right)$ is defined as:

$$K\left(\vec{x_i}, \vec{x_j}\right) = \vec{x_i}\,\vec{x_j} \tag{8}$$

The performance of classification can be improved with a suitable choice of hyperplane. A nonlinear margin can be utilized without much additional computational effort. In this study, we used the Gaussian radial basis function as the nonlinear kernel of the SVM classifier. The corresponding kernel function is

$$k\left(\vec{x_i}, \vec{x_j}\right) = \exp\left(\frac{-\left\|\vec{x_i} - \vec{x_j}\right\|^2}{2\sigma^2}\right) \tag{9}$$

in which σ is the parameter of the kernel function.

3. THE PROPOSED METHOD

Our proposed method is a content-based approach for type detection of computer files and file fragments. It uses feature extraction techniques that were described in Section 2. The basic idea is to consider the BFD of data as basic features. Figure 3 depicts the normalized BFD for different file types. Each computer byte can take 256 different values, varying between 0 and 255. The feature space then consists of 256 basic features that can be used for the type detection. However, multiplicity of features will decrease the speed and accuracy of our detection. Then, feature extraction is required for dimensionality reduction in the feature space. In selecting the most appropriate feature extraction technique, we considered the following goals for our proposed algorithm:

- Accuracy: The algorithm should provide a reasonable accuracy in correct type detection.
- Speed: After required trainings and pre-computations, the algorithm should have an acceptable speed for type detection of unknown data.
- Flexibility: The trade-off between accuracy and speed of the algorithm should be customizable.
- Automatic feature extraction: As there are typically many file types that should be added to the database, the algorithm, after the initial settings, should be able to extract suitable features without any need for direct manipulations.
- Independence from metadata and file headers: It enables our method to work also for type detection of file fragments.

In order to fulfill all the aforementioned requirements, our proposed algorithm includes a hierarchical feature extraction that is followed by a classifier for the type detection. The hierarchical feature extraction is composed of the PCA and an auto-associative neural network. Principals of our feature extraction approach were explained in Section 2. The PCA is a fast procedure, but it is poor when compared with the nonlinear mapping methods. On the other hand, the use of an auto-associative neural network with nonlinear transfer functions has a good performance. However, the computational costs and the risk of being trapped in local minima reduce its efficiency. To resolve such problems and to have a customizable trade-off between accuracy and speed, we use a hierarchical feature extraction method that deploys PCA and an auto-associative neural network. The PCA extracts N_1features out of 256 basic features taken from the BFD. The extracted features are then fed to an auto-associative neural network that extracts N_2 features. These N_2 features will be regarded as the fileprint of the input data. Values of N_1 and N_2 will be determined experimentally in Section 4.

Our hierarchical feature extraction provides speed, accuracy, and flexibility. The speed is preserved by the PCA, which decreases the number of features from 256 to N_1. The accuracy is increased by the auto-associative neural networks, which finally outputs N_2 features. The flexibility and trade-off between accuracy and speed is controlled and customized by selected values for N_1 and N_2. The proposed method provides an automatic feature extraction, as after initial settings and determining values of N_1 and N_2, it does

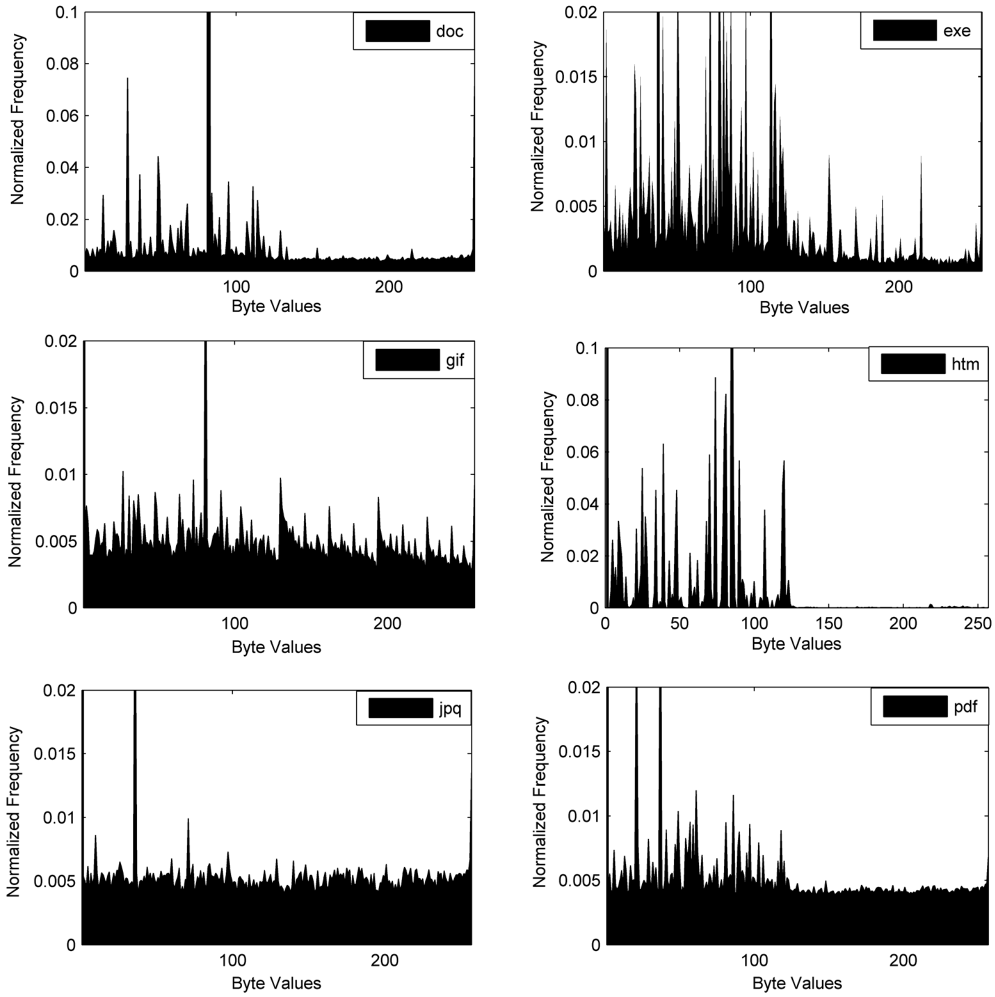

Figure 3. Byte frequency distribution of different file types.

not require any direct manipulations. It is also independent of metadata and file headers because the sample data can be file fragments selected from random starting points within files. This attribute will be experimentally verified in Section 4 by the obtained accuracy.

The proposed scheme includes two phases: the training phase, which is depicted in Figure 4, and the detection phase, which is depicted in Figure 5. During the training phase, the neural networks are trained, and the system is initialized using sample data with known types. The output will be fileprints from different file types. In the detection phase, trained neural networks and fileprints will be used for the type detection.

The training phase starts with normalization of the dataset that is obtained from the BFD of the sample data. As depicted in Figure 4, the output is then applied to the PCA. As explained in Section 2.1, the PCA projection matrix should be calculated. First, the covariance matrix of the training dataset is calculated. Then, the corresponding eigenvectors of the N_1^{th} largest eigenvalues are selected to construct the projection matrix. The value of N_1 is specified according to the allowable error, which is determined by expression (4). This is an obvious customization on the trade-off between accuracy and speed, which exhibits the flexibility attribute of our proposed method. The output features from the PCA are then fed to an auto-associative neural network whose architecture was explained in Section 2.2. The number of neurons at the first and fifth layers is equal to N_1. The number of neurons at the third layer, which is referred to as the bottleneck layer, is equal to N_2, where $N_2 < N_1$. These N_2 features will be used for the type detection at the classification step. The neural network is trained using the back-propagation algorithm so that the desired outputs are the same as inputs. In the training phase of neural networks, the problem of becoming trapped in local minima can be controlled by selecting small values for the step size or learning rate.

In the detection phase, which is depicted in Figure 5, the hierarchical feature extraction system again consists of the PCA and an auto-associative neural network. When an unknown file or file fragment is to be examined for its type detection, its BFD is firstly calculated and normalized.

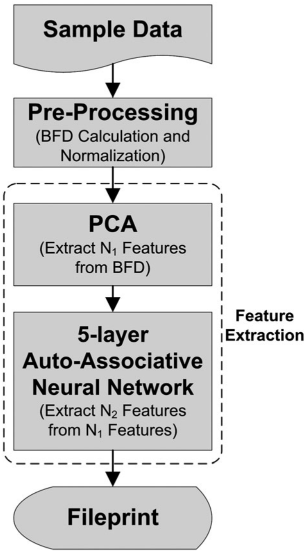
Sample Data
Pre-Processing
(BFD Calculation and Normalization)
PCA
(Extract N₁ Features from BFD)
5-layer Auto-Associative Neural Network
(Extract N₂ Features from N₁ Features)
Feature Extraction
Fileprint

Figure 4. The proposed method (in training phase).

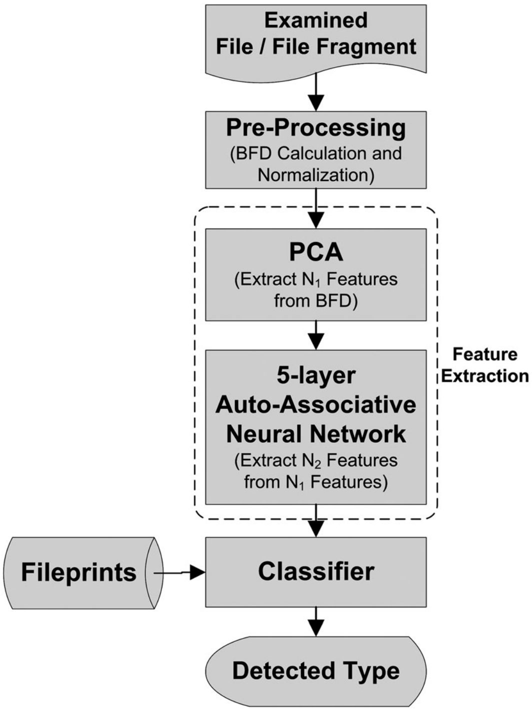
Examined File / File Fragment
Pre-Processing
(BFD Calculation and Normalization)
PCA
(Extract N₁ Features from BFD)
5-layer Auto-Associative Neural Network
(Extract N₂ Features from N₁ Features)
Feature Extraction
Fileprints
Classifier
Detected Type

Figure 5. The proposed method (in detection phase).

The hierarchical feature extraction is then used to extract N_2 features. These features work as the fileprint of the examined data and will be applied to the classifier for the type detection. The classifier compares the fileprint of the examined data with fileprints of data types and declares the type of examined data. For the classifier, any classifier can be used. This is because the fileprints are independent of the classifier. In our experiments, we tried a three-layer MLP classifier and an SVM classifier whose principals were explained in Section 2.3. As our experiments will show in Section 4, the SVM classifier provides better speed and accuracy. For the case of the MLP classifier, although the feature extraction and classification procedures could be compressed in one MLP, we separated it from the feature extraction block. It enhances the learning rate of networks and makes the turnover surveying easy.

4. EXPERIMENTAL RESULTS

In this section, experimental results for the proposed method are presented. It is also shown here that the proposed method works for type detection of computer files when considering the whole content and for type detection of file fragments that are randomly captured with different lengths from randomly selected starting points within files. In Section 4.1, we discuss how our sample data were collected. In Section 4.2, practical points regarding trainings and how to select N_1 and N_2 are explained. Section 4.3 presents practical results including accuracies of the proposed method for type detection of files and file fragments. Section 4.4 compares the results with related works.

4.1. Data collecting method

As an extension of our earlier work [2], our experiments concentrate on six data types: “doc,” “pdf,” “exe,” “jpg,” “htm,” and “gif”. These file types are the most common types on the Internet [1]. However, our current dataset is different from the dataset of [2], and it includes a larger set of test files. The sample files used in experiments were collected from the Internet by using a general search on the *Google* website. For instance, “pdf” files were collected from the *Google* search engine by using the search term “pdf”. Such test files can be considered as randomly selected and unbiased samples because they are captured in the wild. For the case of “doc,” “pdf,” and “htm” files where the BFD of sample files may depend on the language, we tried to collect sample files from different languages to minimize the effects of local settings on the randomization procedure. We collected 1200 random files from the six aforementioned types. We used half of the sample files for the training and the remaining half for testing. The average BFD of sample files are depicted in Figure 3. As the proposed method is not size specific, sizes of our sample files were completely random, and we did not have any control on them. Table I shows the variation of file sizes among our sample files. Although the accomplished randomizations can decrease the accuracy of our results, we were not scared of having the worst situation.

For illustrating the effectiveness of the proposed method for type identification of file fragments and data packets, we repeated our experiments from the starting point for file fragments of 1500 and 1000 bytes length that were randomly selected from random places of the aforementioned files. We used half of such random file fragments for the training and half of them for testing the algorithm. The starting positions

Table I. Specifications of data set used in experiments.

Type of sample files	Number of files	Minimum size (bytes)	Maximum size (bytes)	Average size (bytes)
doc	200	3,494	6,906,880	267,830
exe	200	1,884	21,715,672	7,376,930
gif	200	9,817	4,123,106	79,368
htm	200	232	418,819	42,268
jpg	200	3,717	6,674,957	395,386
pdf	200	12,231	9,025,199	734,380

of the considered file fragments were selected using the "rand" command of Visual C++, and the extracted file fragments were a sequence of consecutive bytes beginning from random starting points within files. We have considered file fragments of 1500 bytes length because they can model data packets going through computer networks such as the Internet. We selected the starting points from random positions to relate our model to the reality as much as possible.

4.2. Training and parameter selection

As described in Section 3, the PCA reduces the number of features from 256 to N_1. These features are then reduced to N_2 features by an auto-associative neural network. Thus, N_2 features form fileprints and will be used for the type detection. The dimensionality reduction in the feature space is associated with a decrease in accuracy but improves speed. As mentioned in Section 3, the trade-off between accuracy and speed is customized by the selected values for N_1 and N_2. The value of N_1 can be selected according to the introduced error of the PCA, mentioned in expression (4). Figure 6 shows the introduced error of the PCA for a variety of the selected features. It indicates that for the case of $N_1 = 60$, the introduced error is negligible. We repeated the procedure with different values of N_1 and deduced that $N_1 = 60$ is an optimum choice that can establish a good trade-off between accuracy and speed. This exemplifies the flexibility attribute that was imposed by the designing goals.

For the value of N_2, we reached to $N_2 = 15$ with a trial-and-error approach. Then, we actually used 15 features for generating fileprints and modeling each file type. Because only 15 features are extracted from 256 basic features, the speed of type detection is greatly increased.

For the case of an MLP classifier, we used a supervised three-layer MLP with 25 neurons in its hidden layer. After several trials, it was observed that one hidden layer network achieves the task to high accuracy. The most suitable network configuration that was found has 25 neurons at its hidden layer. The MLP has been implemented using MATLAB 2010a (The MathWorks Inc., Natick, MA, USA). In the hidden layer and the output layer, the MATLAB *sigmoid* function was used. The *sigmoid* function is nonlinear and allows the network to perform complex mappings of input-to-output vector spaces. It is also continuous and differentiable, so it allows the error gradient to be used for updating the weights. The MLP neural network was trained with the back propagation algorithm. For the back propagation algorithm, the learning rate was 0.01 and the MLP was trained in 300 epochs.

The SVM classifier was implemented using the well-know LIBSVM package [23] from MATLAB 2010a. We used the Gaussian radial basis function of expression (9) with the default value of $\sigma = 1$.

To illustrate the effectiveness of the feature extraction, Figures 7 and 8 illustrate two-dimensional and three-dimensional scatter plots, respectively. In Figures 7 and 8, the auto-associated neural network has two and three

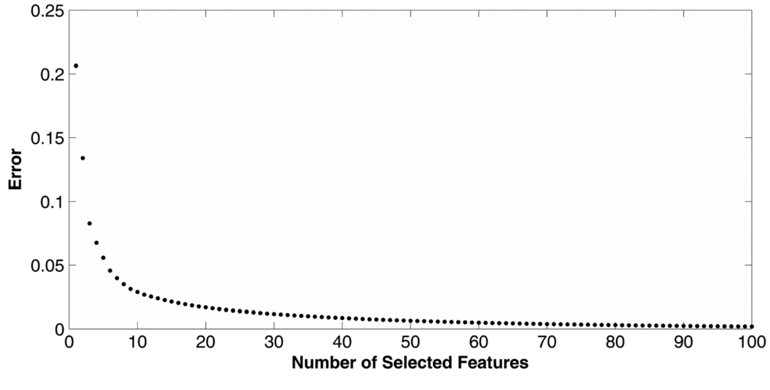

Figure 6. Introduced error of the principal component analysis (PCA).

DOI: 10.1002/sec

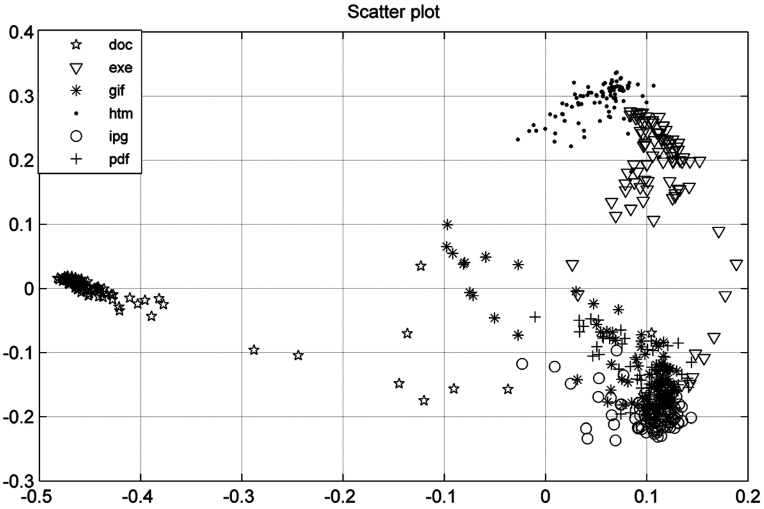

Figure 7. Two-dimensional scatter plot.

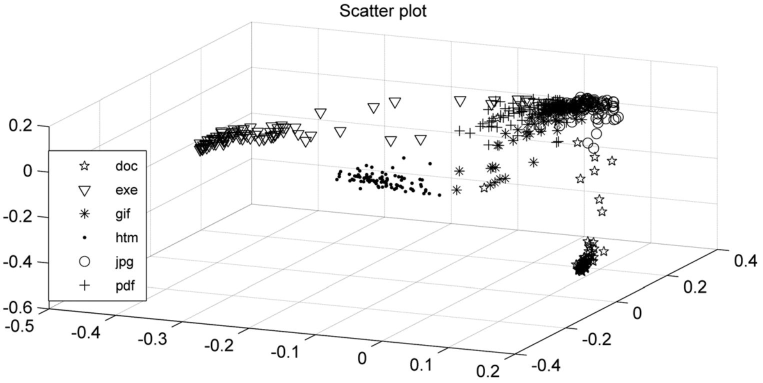

Figure 8. Three-dimensional scatter plot.

neurons at its bottleneck layer, so they only used two and three selected features, respectively. As Figures 7 and 8 show, the accuracy of classification increases by increasing the number of extracted features from two to three. As we extracted 15 features for our experiments, the results are more accurate than those of Figures 7 and 8.

4.3. Practical results

In our earlier work [2], it is shown that the proposed method works very well for type detection of computer files when considering their whole contents. In [2], we used 720 files of six types with the same file types and data collecting method of current experiments. We used 90 files from each type for the training and the remaining 30 files for texting. We used an MLP classifier and had an average correct detection rate of 98.33%. The result was excellent compared with related works [2].

In current experiments, we collected 1200 files of six types and extracted two file fragments of 1500 and 1000 bytes lengths with random starting points from each file. The method of data collection and fragment creation has been explained in Section 4.1. We used half of the files/file fragments for the training and the remaining half for the testing. For the classifier, we tried both MLP and SVM classifiers. Tables II–IV show the confusion matrices for the mentioned test files and file fragments. Average accuracies for different scenarios and different classifiers are summarized in Table V. As Table V shows, the algorithm provides better results with the SVM classifier. With the SVM classifier, the correct classification rate is 99.16%, 85.5%, and 82% for the whole content, file fragments of 1500 bytes length and file fragments of 1000 bytes length, respectively.

Simulations have been carried out using MATLAB 2010a on a 2.62-GHz Dell laptop (Dell Inc., Round Rock, TX, USA) with 4-GB RAM and Windows 7 operating system

Table II. The confusion matrix for 600 test files by considering whole contents.

	Actual type											
Detected	MLP classifier						SVM classifier					
type	doc	exe	gif	htm	jpg	pdf	doc	exe	gif	htm	jpg	pdf
doc	**100**	0	0	0	0	1	**100**	0	0	0	0	0
exe	0	**97**	0	0	0	0	0	**98**	0	0	0	0
gif	0	3	**99**	0	1	2	0	1	**100**	0	1	2
htm	0	0	0	**100**	0	0	0	0	0	**100**	0	0
jpg	0	0	1	0	**99**	0	0	1	0	0	**99**	0
pdf	0	0	0	0	0	**97**	0	0	0	0	0	**98**

MLP, multilayer perceptron; SVM, support vector machine.

Table III. The confusion matrix for 600 random file fragments of 1500 bytes length.

	Actual type											
Detected	MLP classifier						SVM classifier					
type	doc	exe	gif	htm	jpg	pdf	doc	exe	gif	htm	jpg	pdf
doc	**88**	6	1	1	7	1	**89**	5	1	1	7	1
exe	2	**83**	3	1	2	0	2	**85**	2	1	2	1
gif	3	4	**78**	0	5	4	3	2	**80**	0	4	2
htm	1	0	0	**95**	1	2	1	1	0	**95**	1	1
jpg	5	5	11	1	**74**	8	4	5	10	1	**75**	6
pdf	1	2	7	2	11	**85**	1	2	7	2	11	**89**

MLP, multilayer perceptron; SVM, support vector machine.

Table IV. The confusion matrix for 600 random file fragments of 1000 bytes length.

	Actual type											
Detected	MLP classifier						SVM classifier					
type	doc	exe	gif	htm	jpg	pdf	doc	exe	gif	htm	jpg	pdf
doc	**83**	8	2	2	7	2	**85**	7	1	2	7	1
exe	3	**80**	3	1	1	0	3	**81**	3	0	1	1
gif	4	3	**72**	1	8	4	3	3	**76**	1	6	4
htm	1	0	0	**90**	1	1	0	0	1	**91**	1	1
jpg	7	5	13	1	**71**	9	7	5	10	1	**73**	9
pdf	2	4	10	5	12	**84**	2	4	10	5	12	**86**

MLP, multilayer perceptron; SVM, support vector machine.

Table V. Summarized type detection accuracies and running times for 100 examined data of each type.

Status of data	Classifier	doc (%)	exe (%)	gif (%)	htm (%)	jpg (%)	pdf (%)	CCR (%)	Running time (s)
Whole file contents	MLP	100	97	99	100	99	97	98.67	0.047
Whole file contents	SVM	100	98	100	100	99	98	**99.16**	**0.032**
File fragments of 1500 bytes length	MLP	88	83	78	95	74	85	83.83	0.013
File fragments of 1500 bytes length	SVM	89	85	80	95	75	89	**85.50**	**0.009**
File fragments of 1000 bytes length	MLP	83	80	72	90	71	84	80	0.007
File fragments of 1000 bytes length	SVM	85	81	76	91	73	86	**82**	**0.006**

MLP, multilayer perceptron; SVM, support vector machine; CCR, correct classification rate.

Table VI. Comparison between the proposed method and related works.

Contributors	Approach	Header dependent	Size categorization	Method	# File/ data types	# Total samples	Accuracy (%)	Running time (s)
McDaniel and Heydari [7,8]	File	Yes	No	Byte frequency analysis Byte frequency cross correlation File header/trailer analysis	30 30 30	120 120 120	27.5 45.83 95.83	0.010 * 1.19 * 0.015 *
Li *et al.* [6]	File	Yes	Yes	K-means algorithm under Manhattan distance for generating fileprints; Mahalanobis distance for type detection; sample files truncated and multicentroid used for improving accuracy	8 (5 classes)	800	82 (one-centroid), 89.5 (multicentroid), 93.8 (exempler files)	NA
Dunham *et al.* [4]	File	Yes	No	Neural networks to classify types of files encrypted with the same key. Selected features: byte frequency, byte frequency of autocorrelation, and 32 bytes of header	10	760	91.3	NA
Karresand and Shahmehri [12]	File fragment	No	No	Oscar method (distance measurement between a sample vector and a centroid, based on mean and standard deviation of byte values and BFD)Results are biased for type detection of "jpeg"	49	53	97.9 (jpg)	NA
Karresand and Shahmehri [13]	File fragment	No	No	Oscar method + rate of change between consecutive byte valuesResults are biased for type detection of "jpeg"	51	57	87.3–92.1 (jpg),46–84 (zip), 12.6 (exe)	1.20–2.50
Zhang *et al.* [15]	File fragment	No	Yes	BFD and Manhattan distance for detecting executable data packets	2	100	92.5	NA
Moody and Erbacher [14]	File fragment	No	No	statistical modeling technique (mean, standard deviation, and kurtosis)	8	200	74.2	NA
Calhoun and Coles [11]	File fragment	No	No	Fisher's linear discriminant and longest common subsequences applied to nine statistical measurements and their combinations	2	100	68.3-88.3 (Considering bytes 129–1024), 60.3-86 (Considering bytes 513–1024)	NA

(*Continues*)

Table VI. (Continued)

Contributors	Approach	Header dependent	Size categorization	Method	# File/ data types	# Total samples	Accuracy (%)	Running time (s)
Amirani *et al.* [2]	File	No	No	Feature extraction (PCA + neural networks) + MLP classifier	6	720	98.33	NA
Cao *et al.* [3]	File	No	No	Gram frequency distribution + vector space model	4	1000	90.34 (2-gram analysis + 256 grams as file type signature)	NA
Ahmed *et al.* [5]	File	No	No	Cosine similarity + divide-and-conquer approach; multigroup with clustering + MLP classifier	10	2000	90.19	NA
Ahmed *et al.* [9]	File and file fragment	No	No	Feature selection + content sampling + KNN classifier	10	NA	90.5 (using 40% of features), 88.45 (using 20% of features)	NA
Ahmed *et al.* [10]	File and file fragment	No	No	Feature selection + content sampling + KNN classifier	10	5000	90.5 (using 40% of features), 88.45 (using 20% of features)	0.077 (mp3, whole contents),[†] 0.007 (exe, whole contents)[†]
The proposed method	File and file fragment	No	No	Feature extraction (PCA + neural networks) + SVM classifier	6	1200	99.16 (whole contents) 85.5 (1500 bytes fragments) 82 (1000 bytes fragments)	0.032 [‡] 0.009 [‡] 0.006 [‡]

*Measured on 800-MHz Pentium III laptop with 512-MB RAM.

[†] Measured on 2.7-GHz PC with 2-GB RAM.

[‡] Measured on 2.62-GHz laptop with 4-GB RAM by MATLAB codes.

BFD, byte frequency distribution; PCA, principal component analysis; MLP, multilayer perceptron; KNN, K-nearest neighbor; SVM, support vector machine; NA, Not Available.

DOI: 10.1002/sec

(Microsoft Corp., Redmond, WA, USA). The average running-time for type detection of computer files and file fragments using the proposed method for different classifiers is summarized in Table V. The best results are for the SVM classifier where the average running-time is 0.032, 0.009, and 0.006 s for type detection of a computer file with its whole content, a file fragment of 1500 bytes length, and a file fragment of 1000 bytes length, respectively. The result can be greatly improved by improved programming techniques and fast programming languages such as C++.

4.4. Comparison with related works

The advantage of the proposed method over other literature for type detection of computer files when considering their whole content is shown in [2]. The results can be improved using an SVM classifier as shown in Table V. For the case of file fragments, an average correct detection rate of 85.5% and 82% is achieved for random file fragments of 1500 and 1000 bytes lengths, respectively. The achieved result is excellent when compared with the results of other literature in the context of type detection of computer files and file fragments. Table VI compares the proposed method with related works from different aspects. It shows that the proposed method outperforms other works in terms of accuracy. The results of the approach by Ahmed *et al.* [10], which was inspired by our earlier work [2], is compared with our results in Table VII. Table VII shows that the proposed method outperforms the results byAhmed *et al.* [10] in terms of accuracy. For the case of running time, the speed of the proposed method outperforms those of [7,8], as shown in Table VI. Other literature did not have any evaluation on the speed and running time of their proposed methods.

In comparison between the results, one should note that we deployed a simultaneous comparison between six different types in our detection phase, whereas the results of some literatures, such as [15] and [11], are based on selecting between only two file types. It is also noteworthy that the starting points of our file fragments were random. This is different from what followed in [11] in taking file fragments from special places within files. This shows that the proposed method not only works well for type detection of computer files but it also works at an average rate of above 82% for random file fragments of 1000 bytes length that can be deemed as data packets of a specific type going through computer networks. This implies the effectiveness of the proposed approach when the results are compared with results of other methods. The average detection rate of 85.5% of the proposed method that is obtained for the case of random file fragments of 1500 bytes length is greater than the average rate of 82% in [6] and 27.5% in [8] when considering the whole content of files.

Although our proposed method uses some features extracted from the BFD of data content, it is not under any direct influence of file headers or positions of data fragments within files, so it works for the case of file fragments. It cannot be spoofed by changing the file header

Table VII. Comparison between accuracies of the proposed method (6% of features extracted) and those of Ahmed *et al.* [9] (40% of features selected).

File Type	Best results of Ahmed *et al.* [9] with different classifiers		Ahmed *et al.* [9] with KNN classifier (as their optimum choice)	Ahmed *et al.* [9] with SVM classifier	The proposed method with SVM classifier
	Classifier	Accuracy (%)	Accuracy (%)	Accuracy (%)	Accuracy (%)
doc	NN	88	87	75.5	100
exe	NN	99	96	87	98
gif	K-means	100	91	90.5	100
htm	KNN	73.5	73.5	66	100
jpg	SVM	92.5	90	92.5	99
pdf	KNN	97	97	95	98

KNN, K-nearest neighbor; NN, neural network; SVM, support vector machine.

DOI: 10.1002/sec

and works even if the file header has been corrupted. It is invulnerable to file scrambling, that is, it cannot be spoofed by mixing and changing the positions of data fragments. It does not care about file extensions, so extension spoofing cannot take place.

It is also noteworthy that the average correct detection rate of 99.16% is obtained from the worst situation where we considered the whole contents of files and did not do any truncation that could make our results header-dependent and increase the accuracy of type detection. We did not also categorize the downloaded sample files according to their sizes. File sizes were varying between 232 bytes and 21.2 MB as demonstrated in Table I, and we did not care about the problem of sample bias that some literatures (such as [6] and [15]) tried to prevent by file size categorization of the examined files. The actual results may be better than the results presented here.

Although we concentrated on six file types, the proposed method can be tested for further file types. In this case, if the confusion among the examined files is too much, an interclass clustering in the confused files can be performed. In other words, such files can be assumed as two or more kinds, and if the classifier detects each of them, the corresponded file type is declared. This is equivalent to the multicentroid approach. When dealing with the whole content of files, truncation can improve the accuracy and speed, but it can increase the dependency of the method on metadata and file headers.

5. CONCLUSION

In this paper, a content-based file type detection method that can be used for type detection of computer files, file fragments, and data packets is proposed. The proposed method includes a training phase and a detection phase. In the training phase, 15 major features are automatically extracted from BFD of the sample data by using PCA and an unsupervised neural network. In the detection phase, a classifier is used for the type detection. In our experiments, we examined both SVM and MLP classifiers, but the SVM classifiers provided better speed and accuracy. With the SVM classifiers, we obtained an average correct detection rate of 99.16%, 85.5%, and 82% when considering the whole file contents, random file fragments of 1500 bytes lengths, and random file fragments of 1000 bytes lengths, respectively. The results are significant in comparison with those of other literature.

ACKNOWLEDGEMENTS

We would like to thank Prof. Matthew G. Parker for reviewing the manuscript and his helpful comments. We thank Dr. Ali Asghar Beheshti Shirazi for his comments on earlier stages of this work. We are grateful to anonymous reviewers for their valuable comments and suggestions.

REFERENCES

1. Hickok DJ, Lesniak DR, Rowe MC. File type detection technology. In *38th Midwest Instruction and Computing Symposium (MICS'05)*, Eau Claire, Wisconsin, 2005.
2. Chehel Amirani M, Toorani M, Beheshti Shirazi AA. A new approach to content-based file type detection. In *13th IEEE Symposium on Computers and Communications (ISCC'08)*, Marrakech, Morocco, 2008; 1103–1108.
3. Cao D, Luo J, Yin M, Yang H. Feature selection based file type identification algorithm. In *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems (ICIS)*, Xiamen, China, 2010; 58–62.
4. Dunham JG, Sun MT, Tseng JCR. Classifying file type of stream ciphers in depth using neural networks. In *3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA'05)*, Cairo, Egypt, 2005.
5. Ahmed I, Lhee K, Shin H, Hong MP. Content-based file-type identification using cosine similarity and a divide-and-conquer approach. *IETE Technical Review* 2010; **27**: 465–477.
6. Li WJ, Wang K, Stolfo SJ, Herzog B. Fileprints: identifying file types by n-gram analysis. In *6th Annual IEEE Information Assurance Workshop (IAW'05)*, West Point, NY, USA, 2005; 64–71.
7. McDaniel M. Automatic file type detection algorithm. Master's Thesis, James Madison University, 2001.
8. McDaniel M, Heydari MH. Content based file type detection algorithms. In *36th IEEE Annual Hawaii International Conference on System Science (HICSS'03)*, Hawaii, USA, 2003.
9. Ahmed I, Lhee K, Shin H, Hong MP. Fast file-type identification. In *2010 ACM Symposium on Applied Computing (SAC'10)*, Sierre, Switzerland, 2010; 1601–1602.
10. Ahmed I, Lhee KS, Shin HJ, Hong MP. Fast content-based file type identification. In *7th Annual IFIP WG 11.9 International Conference on Digital Forensics, Advances in Digital Forensics VII*, Orlando, Florida, 2011; 65–75.
11. Calhoun WC, Coles D. Predicting the types of file fragments. *Digital Investigation* 2008; **5**: 14–20.
12. Karresand M, Shahmehri N. Oscar–file type identification of binary data in disk clusters and RAM pages. In *IFIP International Information Security Conference: Security and Privacy in Dynamic Environments (SEC'06)*, Karlstad, Sweden, 2006; 413–424.
13. Karresand M, Shahmehri N. File type identification of data fragments by their binary structure. In *IEEE Workshop on Information Assurance*, West Point, NY, 2006; 140–147.

14. Moody SJ, Erbacher RF. SÁDI – Statistical Analysis for Data type Identification. In *3rd IEEE International Workshop on Systematic Approaches to Digital Forensic Engineering*, Oakland, USA, 2008; 41–54.
15. Zhang L, White GB. An approach to detect executable content for anomaly based network intrusion detection. In *IEEE Parallel and Distributed Processing Symposium*, Long Beach, CA, USA, 2007; 1–8.
16. Cheng Q, Zhou H, Cheng J. The Fisher–Markov Selector: fast selecting maximally separable feature subset for multi-class classification with applications to high-dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 2011; **33**: 1217–1233.
17. Panahi N, Shayesteh MG, Mihandoost S, Zali Varghahan B. Recognition of different datasets using PCA, LDA, and various classifiers. In *5th International Conference on Application of Information and Communication Technologies (AICT)*, Baku, Azerbaijan, 2011; 1–5.
18. Bishop CM. *Neural Networks for Pattern Recognition*. Oxford University Press: USA, 1995.
19. Haykin S. *Neural Networks: A Comprehensive Foundation* (2nd edn). Prentice Hall: Upper Saddle River, NJ, 1999.
20. Derya Übeyli E. Statistics over features: EEG signals analysis. *Computers in Biology and Medicine* 2009; **39**: 733–741.
21. Subasi A, Ismail Gursoy M. EEG signal classification using PCA, ICA, LDA and support vector machines. *Expert Systems with Applications* 2010; **37**: 8659–8666.
22. Ye C, Coimbra MT, Vijaya Kumar B. Arrhythmia detection and classification using morphological and dynamic features of ECG signals. In *2010 Annual IEEE International Conference on Engineering in Medicine and Biology Society (EMBC)*, Buenos Aires, 2010; 1918–1921.
23. Chang CC, Lin CJ. LIBSVM: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)* 2011; **2**: 27.