Classification of Volatile Organic Compounds with Incremental SVMs and RBF Networks

Zeki Erdem[1,3], Robi Polikar[2], Nejat Yumuşak [3], and Fikret Gürgen[4]

[1] TUBITAK Marmara Research Center, Information Technologies Institute,
41470 Gebze - Kocaeli, Turkey
zeki.erdem@bte.mam.gov.tr
[2] Rowan University, Electrical and Computer Engineering Department,
210 Mullica Hill Rd., Glassboro, NJ 08028, USA
polikar@rowan.edu
[3] Sakarya University, Computer Engineering Department,
Esentepe, 54187 Sakarya, Turkey
nyumusak@sakarya.edu.tr
[4] Bogazici University, Computer Engineering Department,
Bebek, 80815 Istanbul, Turkey
gurgen@boun.edu.tr

Abstract. Support Vector Machines (SVMs) have been applied to solve the classification of volatile organic compounds (VOC) data in some recent studies. SVMs provide good generalization performance in detection and classification of VOC data. However, in many applications involving VOC data, it is not unusual for additional data, which may include new classes, to become available over time, which then requires an SVM classifier that is capable of incremental learning that does not suffer from loss of previously acquired knowledge. In our previous work, we have proposed the incremental SVM approach based on Learn^{++}.MT. In this contribution, the ability of SVMLearn^{++}.MT to incrementally classify VOC data is evaluated and compared against a similarly constructed Learn^{++}.MT algorithm that uses radial basis function neural network as base classifiers.

1 Introduction

Gas sensing systems for detection and recognition of VOCs are of significant importance for many industries and organizations. Examples include food industries for testing the quality of food products, military and humanitarian organizations for locating buried land mines, petrochemical and valve manufacturing companies for detecting and identifying hazardous gases, and airport security and customs inspection agencies for detecting illegal drugs and plastic bombs. Consequently, gas sensing systems for detection and recognition of VOCs, an important class of chemicals that can readily evaporate, have gained considerable attention, due to VOCs are encountered in many real-world applications. The VOCs classification problem is often made harder due to the irreversible behavior of the sensor array overtime such as parameter drift or just noisy data [1]. Furthermore, one of the main challenges in using gas sensing systems is to be able to increase the number of odorants that can be identified over

P. Yolum et al.(Eds.): ISCIS 2005, LNCS 3733, pp. 322 – 331, 2005.

time with additional data. On the other hand, the training dataset that was originally used to train the system may not be available by the time new training datasets become available.

Support Vector Machines (SVMs) have been used to recognition of VOC data in some studies [1-4]. SVMs provide good generalization performance in the context of odor detection and classification, despite the fact that it does not incorporate problem-domain knowledge [1]. As with any type of classifier, the performance and accuracy of SVMs rely on the availability of a representative set of training dataset. However, acquisition of such a representative VOC dataset is expensive and time consuming as mentioned above. Consequently, such data often become available in small and separate batches at different times. In such cases, a typical approach is combining new data with all previous data, and training a new classifier from scratch. In other words, such scenarios require a classifier to be trained and incrementally updated, where the classifier needs to learn the novel information provided by the new data without forgetting the knowledge previously acquired from the data seen earlier. Since SVMs need to be reinitialized and retrained with the combined old and new data to learn the additional information, they are not capable of incremental learning. This causes all previously acquired knowledge to be lost, a phenomenon known as *catastrophic forgetting* [5]. Therefore, SVMs require incrementally training in recognition of VOC data.

In our previous work [6], integrating the SVM classifiers into an ensemble framework using Learn^{++}.MT, we have shown that the SVM classifiers can in fact be equipped with the incremental learning capability. Learn^{++}.MT was developed in response to reduce the effect of out-voting problem, called classifier proliferation, in the ensemble of classifiers for the incremental learning [7]. In this paper, considering that the problem is caused by the nature of gas sensing system, we investigate the ability of the incremental SVM (SVMLearn^{++}.MT) to classify of VOC data, while avoiding the catastrophic forgetting problem and also reducing the effect of out-voting problem. Its performance have been compared the performance of radial basis function network used as the base classifier of the Learn^{++}.MT.

2 Gas Sensing System

Due to their ability to mimic the human olfactory system, although in a very limited sense, gas sensing systems are often referred to as *Electronic Nose (E-nose) Systems* (Figure 1). An electronic nose is an instrument, which comprises an array of electronic chemical sensors with partial specificity and an appropriate pattern recognition system, capable of recognizing simple or complex odors [8]. The sensor array is a collection of sensors exposed to the same sample and producing individual responses as well as an entire response pattern. Piezoelectric acoustic wave sensors, which comprise a versatile class of chemical sensors, are used for the detection of VOCs data used in this study [9]. For sensing applications, a sensitive polymer film is cast on the surface of the Quartz Crystal Microbalance (QCM). This layer can bind a VOC of interest, altering the resonant frequency of the device, in proportion to the added mass.

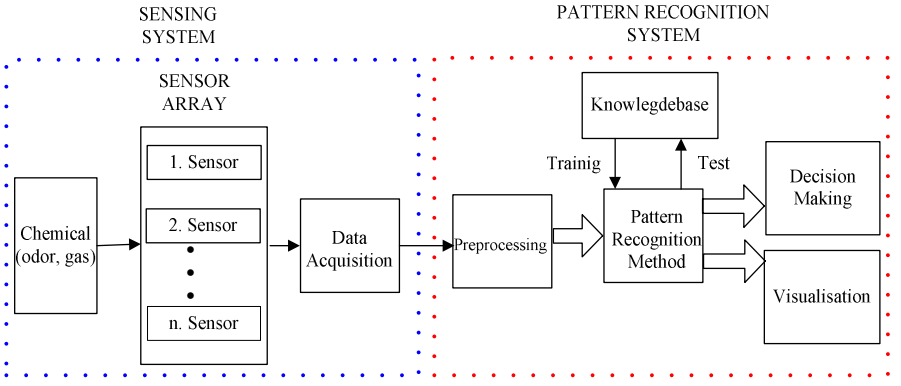

Fig. 1. An Electronic Nose System

Addition or subtraction of gas molecules from the surface or bulk of an acoustic wave sensor results in a change in its resonant frequency. The frequency change Δf, caused by a deposited mass Δm can be described as following:

$$\Delta f = -2.3\times10^{6} \cdot f^{2} \cdot \frac{\Delta m}{A} \tag{1}$$

where f is the fundamental resonant frequency of the bare crystal, and A is the active surface area. The sensor typically consists of an array of several crystals, each coated with a different polymer. This design is aimed at improving identification, hampered by the limited selectivity of individual films. Employing more than one crystal, and coating each with a different partially selective polymer, different responses can be obtained for different gases. The combined response of these crystals can then be used as a signature pattern of the VOC detected.

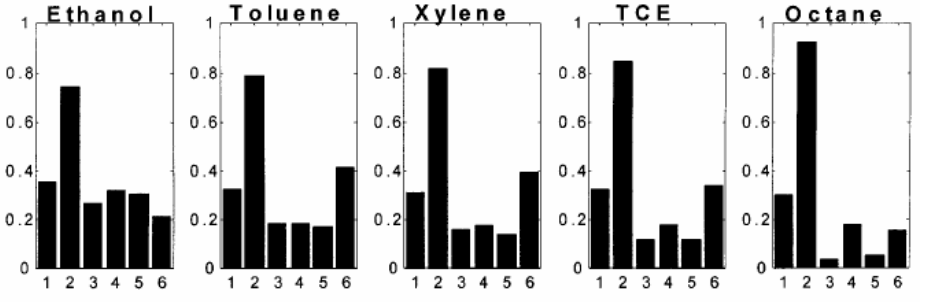

Fig. 2. Sample responses of the six-QCM sensor array to VOCs data

The gas sensing dataset used in this study consisted of responses of six QCMs to five VOCs, including ethanol (ET), xylene (XL), octane (OC), toluene (TL), and trichloroethelene (TCE). Figure 2 illustrates sample patterns for each VOC from six QCMs coated with different polymers, where the vertical axis represents normalized

frequency change. Note that the patterns from toluene, xylene, and trichloroethelene look considerably similar; hence, they are difficult to distinguish from each other.

3 Incremental SVM

3.1 Learn^{++}.MT Algorithm

The Learn^{++} algorithm has been introduced as an incremental learning algorithm that is capable of learning additional information [10], even difficult learning conditions. Learn^{++} not only assumes the previous data to be no longer available, but it also allows additional classes to be introduced with new data, while retaining the previously acquired knowledge. It is an ensemble approach, inspired primarily by the AdaBoost algorithm [11]. Learn^{++} also creates an ensemble of classifiers, each trained on a subset of the current training dataset, and later combined through weighted majority voting. Training instances for each classifier are drawn from an iteratively updated distribution. The main difference is that the distribution update rule in AdaBoost is based on the performance of the previous classifier, which focuses the algorithm on *difficult instances*, whereas that of Learn^{++} is based on the performance of the entire ensemble, which focuses this algorithm on instances that carry *novel information*. This distinction gives Learn^{++} the ability to learn new data, even when previously unseen classes are introduced. As new data reach, Learn^{++} creates additional classifiers, until the ensemble learns the new information. Since no classifier is discarded, previously acquired knowledge is retained.

Learn^{++} uses weighted majority voting, where each classifier receives a voting weight based on its training performance. This works well in practice even for incremental learning problems. However, if the incremental learning problem involves introduction of new classes, then the voting scheme proves to be unfair towards the newly introduced class: since none of the previously created classifiers can pick the new class, a relatively large number of new classifiers need to be generated that recognize the new class, so that their total weight can out-vote the first batch of classifiers on instances coming from this new class. This in return populates the ensemble with an unnecessarily large number of classifiers. Learn^{++}.MT, explained below, is specifically designed to address this issue of *classifier proliferation* [7].

The main innovation in Learn^{++}.MT is the way by which the voting weights are determined. Learn^{++}.MT, also obtains a set of voting weights based on the individual performances of the classifier, however, these weights are then adjusted based on the classification of the specific instance at the time of testing, through *dynamic weight voting (DWV)* algorithm [7]. For any given test instance, Learn^{++}.MT compares the class predictions of each classifier and cross-references them with the classes on which they were trained. Essentially, if a subsequent ensemble overwhelmingly chooses a class it has seen before, then the voting weights of those classifiers that have not seen that class are proportionally reduced. The Learn^{++}.MT algorithm is given in Figure 3.

For each dataset ($\mathbb{D}_k$) that becomes available to Learn^{++}.MT, the inputs to the algorithm are (i) a sequence of m training data instances x_i along with their correct labels y_i, (ii) a classification algorithm, and (iii) an integer T_k specifying the maximum num-

ber of classifiers to be generated using that database. If the algorithm is seeing its first database ($k=1$), a data distribution (D_t), from which training instances will be drawn, is initialized to be uniform, making the probability of any instance being selected equal. If $k>1$ then a distribution initialization sequence initializes the data distribution. The algorithm adds T_k classifiers to the ensemble starting at $t=eT_k+1$, where eT_k denotes the current number of classifiers in the ensemble.

Input: For each dataset $\wp_k$ $k=1,2,\ldots,K$

- Sequence of m instances $S=[(x_1,y_1),\ldots,(x_m,y_m)]$ with labels $y_i \in Y_k = \{1,\ldots,c\}$
- Learning algorithm **BaseClassifier**.
- Integer T_k, specifying the number of iterations

Do for $k=1,2,\ldots,K$

If k=1 **Initialize** $w_1 = D_1(i) = 1/m,\ eT_1 = 0$ for all i.

Else **Go to** Step 5 to evaluate the current ensemble on new data set

$\wp_k$, update weights, and recall current number of classifiers $eT_k = \sum_{j=1}^{k-1} T_j$

Do for $t= eT_k +1,\ eT_k +2,\ldots,\ eT_k + T_k$:

1. Set $D_t = w_t \Big/ \sum_{i=1}^{m} w_t(i)$ so that D_t is a distribution.
2. Call **BaseClassifier** providing it with a subset of $\wp_k$ randomly chosen using D_t.
3. Obtain a hypothesis h_t : X → Y, and calculate the error h_t : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$

   If $\varepsilon_t > ½$, discard h_t and go to step 2.

   Otherwise, compute normalized error as $\beta_t = \varepsilon_t / (1-\varepsilon_t)$.

4. $CTr_t = Y_k$, save labels of classes used in training h_t.
5. Call **DWV** to obtain the composite hypothesis H_t.
6. Compute the error of the composite hypothesis $E_t = \sum_{i:H_t(x_i)\neq y_i} D_t(i)$
7. Set $B_t=E_t/(1-E_t)$, and update the instance weights:

$$w_{t+1}(i) = w_t \times \begin{cases} B_t, & if\ H_t(x_i) = y_i \\ 1, & otherwise \end{cases}$$

Call **DWV** to obtain the final hypothesis, H_{final}.

Fig. 3. The Learn^{++}.MT Algorithm

For each iteration t, the instance weights, w_t, from the previous iteration are first normalized to create a weight distribution D_t. A classifier, h_t, is generated from a subset of $\wp_k$ that is drawn from D_t. The error, ε_t, of h_t is then calculated; if $\varepsilon_t > ½$, the algorithm deems the current classifier, h_t, to be too weak, discards it, and returns and redraws a training dataset, otherwise, calculates the normalized classification error β_t. The class labels of the training instances used to generate this classifier are then stored. The *DWV* algorithm is called to obtain the composite classifier, H_t, of the ensemble. H_t represents the ensemble decision of the first t hypotheses generated thus far. The error of the composite classifier, E_t is then computed and normalized. The instance weights w_t are finally updated according to the performance of H_t such that the weights of instances correctly

classified by H_t are reduced and those that are misclassified are effectively increased. This ensures that the ensemble focus on those regions of the feature space that are not yet learned, performing the incremental learning [7].

3.2 SVM Classifiers and Its Ensemble

Support vector machines (SVMs) have been successfully employed in a number of real world problems [12, 13]. They directly implement the principle of structural risk minimization [12] and work by mapping the training points into a high dimensional feature space, where a separating hyperplane ($\boldsymbol{w}$, b) is found by maximizing the distance from the closest data points (boundary-optimization). Given a set of training samples $\boldsymbol{S}=\{(\boldsymbol{x}_i,y_i) \mid i=1,\ldots,m\}$, where $\boldsymbol{x}_i \in \boldsymbol{R}^n$ are input patterns, $y_i \in \{+1, -1\}$ are class labels for a 2-class problem, SVMs attempt to find a classifier $h(\boldsymbol{x})$, which minimizes the expected misclassification rate. A linear classifier $h(\boldsymbol{x})$ is a hyperplane, and can be represented as $h(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T\boldsymbol{x}+b)$. The optimal SVM classifier can then be found by solving a convex quadratic optimization problem:

$$\max_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \quad \text{subject to } y_i\left(\langle w, x_i\rangle + b\right) \geq 1-\xi_i \text{ and } \xi_i \geq 0 \tag{2}$$

where b is the bias, $\boldsymbol{w}$ is weight vector, and C is the regularization parameter, used to balance the classifier's complexity and classification accuracy on the training set S. Simply replacing the involved vector inner-product with a non-linear kernel function converts linear SVM into a more flexible non-linear classifier, which is the essence of the famous *kernel trick*. In this case, the quadratic problem is generally solved through its dual formulation:

$$L(w,b,\alpha) = \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\left(\sum_{i=1}^{m} y_i y_j \alpha_i \alpha_j K(x_i,x_j)\right) \text{ subject to } C \geq \alpha_i \geq 0 \text{ and } \sum_{i=1}^{m}\alpha_i y_i = 0 \tag{3}$$

where α_i are the coefficients that are maximized by Lagrangian. For training samples $\boldsymbol{x}_i$, for which the functional margin is one (and hence lie closest to the hyperplane), $\alpha_i > 0$. Only these instances are involved in the weight vector, and hence are called the *support vectors* [13]. The non-linear SVM classification function (optimum separating hyperplane) is then formulated in terms of these kernels as:

$$h(x) = sign\left(\sum_{i=1}^{m}\alpha_i y_i K(x_i,x_j) - b\right). \tag{4}$$

The final composite SVM classifier is obtained using the *DWV* algorithm for Learn^{++}.MT algorithm (Figure 4), as follows. Inputs of *DWV* are (i) the current training data and corresponding correct labels, (ii) classifiers h_t (iii) β_t, normalized error for each h_t, and (iv) a vector containing the classes on which h_t has been trained. The SVMs classifier weights, $W_t = \log(1/\beta_t)$, are first initialized according to where each single SVM classifier first receives a standard weight that is inversely proportional to its normalized error β_t so that those classifiers that performed well on their training data are given higher voting weights. A normalization factor is then created as the sum of the weights of all classifiers trained with class c=1,2,…,C. For each instance, a per-class confidence factor $0<P_c<1$ is generated. P_c is the sum of weights of all the

classifiers that choose class c divided by the sum of the weights of all classifiers trained with class c. Then, for each class, the weights are adjusted for classifiers that have not been trained with that class, that is, the weights are lowered proportional to the ensemble's preliminary decision on that class. The final composite SVM classifier is then calculated as the maximum sum of the weights that chose a particular class:

$$H_{final}(x_i) = \arg\max_{c} \sum_{t:h_t(x_i)=c} W_t \ . \tag{5}$$

Inputs:
- Sequence of i=1,…,n training instances or test instance x_i
- Classifiers h_t.
- Hypothesis error values, β_t.
- Classes, CTr_t used in training h_t.

For t=1,2,…,T where T is the total number classifiers.

1. Initialize classifier weights $W_t = \log(1/\beta_t)$

2. Create normalization factor, Z, for each class $Z_c = \sum_{t:c\in CTr_t} W_t$, for c=1,2,…,C classes

3. Obtain preliminary decision $P_c = \left.\sum_{t:h_t(x_i)=c} W_t \right/ Z_c$, for c=1,2,…,C classes

4. Update voting weights $W_{t:c\notin CTr_t} = W_{t:c\notin CTr_t}(1-P_c)$ for c=1,2,…,C

5. Compute final hypothesis $H_{final}(x_i) = \arg\max_{c} \sum_{t:h_t(x_i)=c} W_t$

Fig. 4. The Dynamic Weight Voting Algorithm

4 Experimental Results

A single hidden layer classical Radial Basis function (RBF) network and SVM with RBF kernel were used as the base classifier in our experiments. SVM and RBF network with Learn^{++}.MT (SVMLearn^{++}.MT and RBFLearn^{++}.MT) have been tested on VOC dataset.

RBF kernel : $$K(x_i, x_j) = \exp\left(-\left\|x_i - x_j\right\|^2 / 2\sigma^2\right) \tag{6}$$

SVM classifier parameters are the regularization constant C, the spread σ of RBF kernel. RBF classifier parameters are the mean squared error goal and the spread of radial basis functions. The choice of classifier parameters is a form of model selection. Although the machine learning community has extensively considered model selection with SVMs, optimal model parameters are generally domain-specific [14]. Therefore, we used the cross-validation technique with 5-folds to jointly select the SVM and also RBF network parameters.

The VOC dataset consisted of 384 six dimensional signals, 220 of which were used for training, and 164 of which were used for testing (TEST). Training dataset was divided into three training subsets (DS1, DS2, DS3). DS1 had instances from ET, OC, and TL, DS2 added instances mainly from TCE and very few from the previous three,

Table 1. VOC data distribution

Class	ETHANOL (C1)	TCE (C2)	OCTANE (C3)	XYLENE (C4)	TOLUENE (C5)
DS1	20	0	20	0	40
DS2	10	25	10	0	10
DS3	10	15	10	40	10
Test	24	24	24	40	52

Table 2. SVMLearn^{++}.MT with RBF kernel ($\sigma = 3$, $C = 100$) results

	C1	C2	C3	C4	C5	Gen.	Std
DS1	94%	-	91%	-	100%	59%	1.42%
DS2	98%	97%	83%	-	93%	70%	1.24%
DS3	95%	95%	90%	100%	71%	88%	1.63%

Table 3. RBFLearn^{++}.MT ($\sigma = 0.45$, $goal = 0.5$) results

	C1	C2	C3	C4	C5	Gen.	Std.
DS1	93%	-	82%	-	98%	57%	1.18%
DS2	96%	96%	77%	-	91%	68%	1.44%
DS3	90%	92%	80%	99%	69%	85%	3.83%

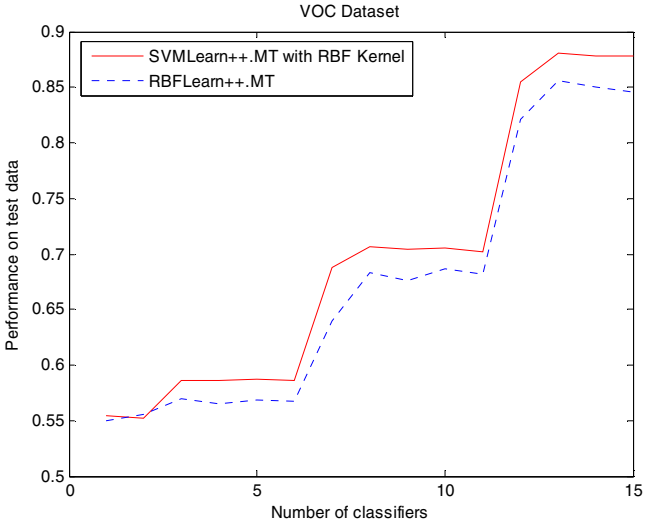

Fig. 5. Performance Results

and DS3 added instances from XL and very few from the previous four. TEST set included instances from all classes. Only DS_k was used during the k^{th} training session. Table 1 presents the distribution of the datasets where subsequent datasets are biased toward the new class. Such a distribution results in challenge; since the algorithm will no longer have the opportunity to see adequate number of instances from previously introduced classes in the subsequent training sessions.

SVMLearn^{++}.MT and RBFLearn^{++}.MT were incrementally trained with three subsequent training datasets. They were permitted to generate as many classifiers as necessary to obtain their maximum performance. The numbers of classifiers generated were 6, 5, and 4 to achieve their best performance in three training sessions, respectively. Results from tests are shown in Tables 2 and 3 based on averages of 30 trials.

As expected, the performances of the classifiers on their own training data were very high. We note that the performance on the TEST dataset improves as incremental learning progresses and the system learns new classes. This is also expected, since TEST set had instances from all five classes, and instances from all classes were not introduced to classifiers until the last session. The performance improvement on the TEST data as new datasets are introduced demonstrates the incremental learning capability of the algorithm.

Performance results from tests are shown in Figure 5 based on mean of 30 trials. Generalization performance of SVMLearn^{++}.MT and RBFLearn^{++}.MT on the test dataset progressively improved from 59-57% to 88-85%, respectively, as new data was introduced, demonstrating its incremental learning capability even when instances of new classes are introduced in subsequent training sessions.

5 Conclusions

In this contribution, we have shown that incremental SVM (SVMLearn^{++}.MT) can incrementally learn new classes, while retaining the previously acquired knowledge and also reducing of the classifier proliferation. In other words, the ability of learning new information provided by subsequent datasets, including new knowledge provided by instances of previously unseen classes, has been presented. SVMLearn^{++}.MT with RBF kernel has also been compared against standard RBF network used as the base classifier of Learn^{++}.MT. The results demonstrate that SVMLearn^{++}.MT produced slightly better generalization performance, but also provided a significantly more stable improvement as seen from the reduced standard deviation.

Acknowledgements

This work is supported in part by the National Science Foundation under Grant No. ECS-0239090, "CAREER: An Ensemble of Classifiers Approach for Incremental Learning." Z.E. would like to thank Mr. Apostolos Topalis and Mr. Michael Muhlbaier graduate students at Rowan University, NJ, for their invaluable suggestions and assistance.

References

1. Distante, C., Ancona, N., Siciliano, P.: Support Vector Machines for Ofactory Signals Recognition. Sensors and Actuators B Vol. 88 (2003), 30-39.
2. Erdem, Z., Gürgen, F., Yumuşak, N.: Electronic Nose Data Classification using Support Vector Machines. Proceedings of IEEE 10th Turkish Signal Processing and Applications Conference (SIU'2002), Vol, 2. (2002) 1174-1179.

3. DeCoste, D., Burl, M. C., Hopkins, A., Lewis, N. S.: Support Vector Machines and Kernel Fisher Discriminations: A Case Study using Electronic Nose Data. Fourth Workshop on Mining Scientific Datasets, Seventh ACM SIGKDD International Conference on Kwowlegde Discovery and Data Mining (KDD-2001), August 26, 2001.
4. Trihaas, J., Bothe, H.H.: An application of Support Vector Machines to E-nose data. In Proceedings of ISOEN '2002 (International Symposium in Olfaction and Electronic Noses), Eds: A. D'Amico and C. Di Natale, (2003) 170-174.
5. French, R.: Catastrophic forgetting in connectionist networks: Causes, Consequences and Solutions. Trends in Cognitive Sciences, Vol. 3. No. 4. (1999)128-135.
6. Erdem, Z., Polikar, R., Gürgen, F., Yumuşak, N.: Reducing the Effect of Out-voting Problem in Ensemble Based Incremental Support Vector Machines. International Conference on Artificial Neural Networks (ICANN 2005) 11-15 September 2005, Warsaw, Poland.
7. Muhlbaier, M., Topalis, A., Polikar, R.: Learn++.MT: A New Approach to Incremental Learning. 5th Int. Workshop on Multiple Classifier Systems (MCS 2004), Springer LNCS Vol. 3077, (2004) 52-61.
8. Gardner, J.W., Bartlett, P.N.: A brief history of electronic noses. Sensors and Actuators B Vol. 18-19 (1994.), 211-220.
9. Polikar, R., Shinar, R., Honavar, V., Udpa, L., Porter, M. D.: Detection and Identification of Odorants using An Electronic Nose. Proc. of IEEE 26th Int. Conf. on Acoustics, Speech and Signal Proc., vol. 5. (2001) 3137-3140.
10. Polikar, R., Udpa, L., Udpa, S., Honavar, V.: Learn^{++}: An incremental learning algorithm for supervised neural networks. IEEE Transactions on Systems, Man, and Cybernetics. Part C: Applications and Reviews, Vol. 31, No. (2001) 497-508.
11. Freund, Y., Schapire, R.: A decision theoretic generalization of on-line learning and an application to boosting. Computer and System Sciences, vol. 57. no. 1. (1997) 119-139.
12. Vapnik, V., Statistical Learning Theory. New York: Wiley, (1998).
13. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000).
14. Duan, K., Keerthi, S.S., Poo, A.N.: Evaluation of simple performance measures for tuning SVM hyperparameters. Neurocomputing, Vol. 51. (2003) 41-59.