Comparative Classifiers for Software Quality Assessment

Sunida Ratanothayanon, Chouvanee Srivisal, Sirirut Vanichayobonand, and Ladda Preechaveerakul

***Abstract*—Software defect predictive model can efficiently help improve software quality and lessen testing effort. A large number of predictive models are proposed in a software engineering literature, but this paper presents the proposed method in software defect prediction with the comparative results based on two classifiers, i.e., backpropagation neural network and radial basis functions with Gaussian kernels as classifiers. Comparative results on NASA dataset are demonstrated and analyzed on the basis of mean square error and percent of accuracy. Experimental results show that the neural network performs better prediction than the RBF in almost subsets of data from 5.76% to 6.75%.**

***Index Terms*—Software quality, software defect prediction, Software classifiers, fault-prone software modules**

I. Introduction

Development and testing of a large software system consumed resources and time. Software development managers often experience problems of allocating sufficient time and resources for software testing and quality assurance activities. Recently, much work has been researched in a software engineering literature about fault-prone classifiers for software quality assessment. Many well-known classification techniques in machine learning, data mining, statistics such as neural network, Bayesian network, radial basis function (RBF), clustering, probabilistic relational model, decision trees, and naive bayes are used to evaluate software modules. Studies are reported in the literature [2, 3]. In addition, parsimonious classifier of software quality assessment using Gaussian kernel radial basis functions is studied in [4]. A model reported in [5] has been developed by using Bayesian network to assess and predict software quality. Unsupervised learning technique such as clustering is used in prediction of faults in software systems [6, 7]. Moreover, other data mining techniques such as feature subset selection, classifying feature description and associate rule are respectively applied to classify software engineering dataset in [8, 9, 10].

The common point of such studies is that their works focus on the insight and the analysis of their own techniques. They pay very little attention to the comparative results and no recommendation of the software defect classifiers is suggested. Moreover, the heavy metrics of the software engineering dataset is used without performance testing on attribute reduction. Therefore, our work aims to present comparative results of the backpropagation neural network and the RBF. An experimental comparison of the proposed methods is provided in this paper. Different experiments in this research are conducted by using smaller subsets of NASA software engineering dataset [1]. The quality of our models is evaluated based on classification accuracy and mean square error (MSE). Models obtained from each presented technique are analyzed and then selected. The comparative results within the same size of networks and the difference of average percent of accuracy between them are also presented. Then, the recommendation is suggested to software developers.

The rest of paper is organized as follows. Section II discusses about the classification techniques. In Section III, the proposed method is described. The dataset, model implementation, the software predictive results and comparison are discussed in Section IV. Finally, conclusions are reported in Section V.

Manuscript received June 3, 2012; revised July 3, 2012.

The authors are with the Department of Computer Science, Faculty of Science, Prince of Songkla University Hatyai, Songkhla, Thailand 90112 (e-mail:sunida.r@psu.ac.th,chouvanee.s@psu.ac.th,sirirut.v@psu.ac.th,ladda.p@psu.ac.th)

II. Model Description

A. Backpropagation Neural Network

To obtain predictive models, the algorithm of the backpropagation neural network with multihidden layer [12] is applied to software engineering dataset. In the neural network architecture with a perceptron, inputs are activated with the sum of the product of the inputs $\boldsymbol{x}$ and the weights $\boldsymbol{w}$. Its output becomes the input of each hidden units. In each hidden unit, the inputs are computed to obtain a predictive output (O) by a sigmoid function, a non linear function defined by the equation (1).

$$\sigma(t) = \frac{1}{1+e^{-t}} \quad (1)$$

In the training process, using given inputs training data x^k with corresponding output value y^k , an error can be computed by the equation (2).

$$\delta_k \leftarrow y^k \text{-} \sigma(w.x^k) \quad (2)$$

Then, the neural network weight can be updated in the equation (3).

$$w_j \leftarrow w_j + \alpha\delta_k x_j^k \sigma'(w.x^k) \quad (3)$$

A derivative of σ is σ'=$\sigma(1-\sigma)$and α is a learning rate. In multilayer case, errors are approximated by backpropagating the final output error. The algorithm of the backpropagation neural network [12] is described in the following procedure in Fig. 1.

```
Input: Given input training data $x^k$
Output: Output from each output unit $O^k$
1.   Initialize the weight to small numbers until satisfied do
2.   For each training example, do
     2.1  Input the training example to the networks and compute actual outputs. (forward pass)
     2.2  Compute weight change (backward pass)
          For each output unit k
              $\delta_k \leftarrow O_k(1-O_k)(d_k-O_k)$   Where $d_k$ is a target output and $O_k$ is an output of each
     output unit $k$.
          For each hidden unit $h$
              $\delta_h \leftarrow O_h(1-O_h)\sum_{k\epsilon outputs} w_{h,k}\delta_k$  Where $O_h$is an output of each hidden unit $h$.
     2.3  Update each network weight $w_{ij}$
          $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$     Where $\Delta w_{i,j} = \alpha\delta_j x_{i,j}$
3.   The algorithm is stopped when the value of the Mean Square Error (MSE) is acceptable.
```

Fig. 1. The backpropagation neural network algorithm [12].

B. Radial Basis Function (RBF)

The SG RBF algorithm, which is a recent RBF algorithm proposed by the author, Shin and Goel [4, 11] is applied to the datasets. RBF architecture is similar to the neural network, but using the Gaussian function as the nonlinearity for the hidden layer. The RBF network consists of three layers, (i.e., input, hidden and output.)

n ✕ *d* input vector is nonlinearly transformed by the basis function to the hidden layer.
$X = (x_1, x_2, \ldots\ldots, x_n)^T \epsilon R^{n\times d}$ where *n* is an input data size, and *d* is a dimensional input data vector.

In practice, the Gaussian function gives a good approximation and is easy to control parameter σ, which is a width of the basis function. The RBF model for the Gaussian case can be described as equation (4)

$X = (x_1, x_2, \ldots\ldots, x_n)^T \epsilon R^{n\times d}$ where *n* is an input data size, and *d* is a dimensional input data vector.

In practice, the Gaussian function gives a good approximation and is easy to control parameter σ, which is a width of the basis function. The RBF model for the Gaussian case can be described as equation (4).

$$\varphi_i(x) = exp\left(-\frac{\|x-\mu_i\|^2}{2\sigma_i^2}\right) \qquad (4)$$

where $x \epsilon R^d$ is the input vector and $\mu_j \epsilon R^d$ is the i^{th} basis function center. Then, the transformed output $\varphi_i(x)$ is linearly weighted to produce the final output. A mapping of $f: R^d \to R$ is in equation (5).

$$f(x) = \sum_{j=1}^{m} w_j\varphi_j(\mathrm{x}) = \sum_{j=1}^{m} w_j\, exp\left[\frac{-\|x-\mu_j\|^2}{2\sigma_j^2}\right] \qquad (5)$$

From equation (5), *m* is the number of basis function. w_j's are weight and ||·|| is the Euclidean distance. A parameter set for the RBF model is defined by $P = (m, \sigma, \mu, w)$. In the SG RBF algorithm, a global parameter σ that users control the value is given. The algorithm selects the number of basis function for a given σ. The center of the basis function μ_j's are determined for a pair of selected m and σ. Next, the weight parameters w are determined by the pseudo-inverse method.

III. Proposed Method

Three core steps of our proposed method are illustrated in Fig 2. They are 1) Data gathering and preprocess, 2) Model building and evaluation, and 3) Model comparison.

In data gathering and preprocess step, dataset is gathered by selecting from the collection of public software engineering data repository. Then, the data are preprocessed by removing redundant data. Outliers are also removed from the dataset by observing from thplot of the attribute values, and data are normalized within the range between 0 and 1. The labels of output are classified as 1 (fault-prone) and 0 (non fault-prone). Null or zero value attributes are eliminated from the dataset. In model building and evaluation step, the backpropagation neural network and RBF are applied to build the models. Two new subsets of data are created under the assumption that subset of attributes contain only important feature will give better result. Only class relevant attributes or only method relevant attributes are selected for class-level dataset or method-level dataset respectively. The experimental results are compared in the last step.

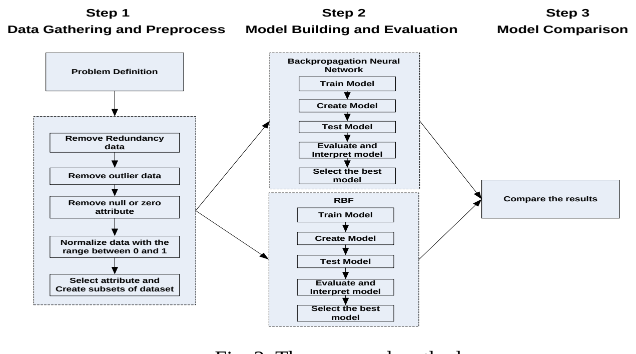

Step 1
Data Gathering and Preprocess
Step 2
Model Building and Evaluation
Step 3
Model Comparison
Problem Definition
Remove Redundancy data
Remove outlier data
Remove null or zero attribute
Normalize data with the range between 0 and 1
Select attribute and Create subsets of dataset
Backpropagation Neural Network
Train Model
Create Model
Test Model
Evaluate and Interpret model
Select the best model
RBF
Train Model
Create Model
Test Model
Evaluate and Interpret model
Select the best model
Compare the results

Fig. 2. The proposed method.

IV. Implementation and Analysis

A. Dataset

The dataset KC1 (defective or not) is taken from the NASA software database as reported in the PROMISE data repository [1]. This dataset can be categorized into two data subsets, which are class-level and method level data. Attributes of the dataset are presented in Table I. A set of static measures are used as a predictor variables. The numbers of instances are 145 records. The first 31 attributes are input, and the last attribute is output classified into a discrete value as 0 and 1. If the class or the method of the program module contains one or more defects, the output is 1 and 0 otherwise. There are 60 records in class 1 and 85 record in class 0. In modeling process, the dataset is partitioned to three data subsets: class-level, method-level and all attributes.

B. RBF Model

The dataset used is randomly selected 50% for training, 25% for validation and 25% for testing. The RBF model is selected based on low value of MSE on testing data, but it should have moderate number of basis function to avoid overfitting problem. From the experiments, 95% and 99% of the confidential level are assigned for testing and comparing the results. The models with a small number of the Gaussian function that have high value of σ and large training error are considered to be underfitting models because they do not learn enough. However, the complex models with the large number of the Gaussian function with small value of σ and large testing error are overfitting models because it is unable to provide good generalization on unseen data. The results of the RBF classifiers on three data subset: class-level, method-level and all attributes are presented in Table IIa)-IIc) and their MSE plots are shown in Fig. 3a)-3c), respectively.

TABLE I: ATTRIBUTES OF THE INPUT DATASET

Features at Class Level
1. PERCENT_PUB_DATA: The percentage of data that is public and protected data in a class.
2. ACCESS_TO_PUB_DATA: The amount of times that a class's public and protected data is accessed.
3.COUPLING_BETWEEN_OBJECTS: The number of distinct non-inheritance-related classes on which a class depends.
4. DEPTH: The level for a class.
5. LACK_OF_COHESION_OF_METHODS: This metric indicates low or high percentage of cohesion.
6. NUM_OF_CHILDREN: The number of classes derived from a specified class.
7. DEP_ON_CHILD: Whether a class is dependent on a descendant.
8. FAN_IN: This is a count of calls by higher modules.
9.RESPONSE_FOR_CLASS: A count of methods implemented within a class plus the number of methods accessible toan object class due to inheritance.
10. WEIGHTED_METHODS_PER_CLASS: A count of methods implemented within a class

Features Transformed to Class Level (Originally at Method Level)
11. sumLOC_BLANK: Lines with only white space or no text content.
12. sumBRANCH_COUNT: This metric is the number of branches for each module.
13. sumLOC_CODE_AND_COMMENT: Lines that contain both code and comment.
14. sumLOC_COMMENTS:The number of lines in a module including all blank lines, comment lines, and source lines.
15. sumCYCLOMATIC_COMPLEXITY: It is a measure of the complexity of a modules decision structure. It is the number of linearly independent paths.
16. sumDESIGN_COMPLEXITY: Design complexity is a measure of a module's decision structure as it relates to calls to other modules.
17. sumESSENTIAL_COMPLEXITY: Essential complexity is a measure of the degree to which a module contains unstructured constructs.
18. sumLOC_EXECUTABLE: Source lines of code that contain only code and white space.
19. sumHALSTEAD_CONTENT: Complexity of a given algorithm independent of the language used to express the algorithm.
20. sumHALSTEAD_DIFFICULTY: Level of difficulty in the program.
21.sumHALSTEAD_EFFORT: Estimated mental effort required to develop the program.
22. sumHALSTEAD_ERROR_EST: Estimated number of errors in the program.
23. sumHALSTEAD_LENGTH: This is a Halstead metric that includes the total number of operator occurrences and total number of operand occurrences.
24. sumHALSTEAD_LEVEL: Level at which the program can be understood.
25. sumHALSTEAD_PROG_TIME: Estimated amount of time to implement the algorithm.
26. sumHALSTEAD_VOLUME: This is a Halstead metric that contains the minimum number of bits required for coding the program.
27. sumNUM_OPERANDS: Variables and identifiers Constants Function names when used during calls.
28. sumNUM_OPERATORS: Number of operators.
29. sumNUM_UNIQUE_OPERANDS: Variables and identifiers Constants (numeric literal/string) Function names when used during calls
30. sumNUM_UNIQUE_OPERATORS: Number of unique operators.
31. sumLOC_TOTAL: Total Lines of Code.

TABLE II: RESULTS OF THE RBF CLASSIFIERS

Model	sigma	Basis Functions (m)	%Correct Classification			Mean Square Error (MSE)		
			Train	Validation	Test	Train	Validation	Test
A	1	7	79.3427	57.1427	72.3810	0.1987	0.2301	0.2192
B	0.9	8	78.8730	59.0473	73.3333	0.1933	0.2323	0.1972
C	0.8	10	78.8733	60.9523	72.3810	0.1823	0.2282	0.2141
D	0.7	12	75.1177	59.0477	72.3810	0.1697	0.2339	0.1998
E	0.6	15	75.1173	61.9047	75.2380	0.1623	0.2321	0.2191
F	0.5	20	76.9953	57.1427	73.3333	0.1537	0.2511	0.1985
G	0.4	29	79.8123	60.9523	73.3333	0.1253	0.2636	0.2142
H	0.3	41	82.6290	56.1907	69.5240	0.0849	0.2973	0.2023
I	0.2	56	84.5070	54.2857	74.2857	0.0476	0.3540	0.2015
J	0.1	65	88.2630	52.3813	74.2857	0.0132	0.4887	0.2162

a) Results of the RBF classifiers on Class-level

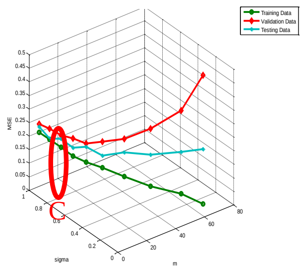
Model	sigma	Basis Functions (m)	%Correct Classification			Mean Square Error (MSE)		
			Train	Validation	Test	Train	Validation	Test
A	1	2	71.2960	54.6293	71.4287	0.1880	0.2536	0.1712
B	0.9	2	71.7590	56.4813	72.3810	0.1880	0.2527	0.1699
C	0.8	2	71.2960	57.4073	77.1430	0.1858	0.2506	0.1675
D	0.7	3	71.2960	56.4813	71.4287	0.1804	0.2537	0.1758
E	0.6	3	71.2960	53.7037	72.3810	0.1820	0.2526	0.1645
F	0.5	3	74.5370	57.4077	69.5240	0.1685	0.2512	0.1836
G	0.4	4	75.9257	53.7037	68.5713	0.1618	0.2651	0.2042
H	0.3	5	77.7780	53.7037	71.4287	0.1474	0.2693	0.1818
I	0.2	9	81.0187	52.7780	73.3333	0.1205	0.3023	0.1760
J	0.1	18	85.1853	50.9260	71.4287	0.1065	0.3867	0.1863

b) Results of the RBF classifiers on Method-level

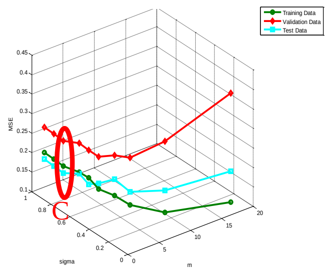
Model	sigma	Basis functions (m)	%Correct Classification			Mean Square Error (MSE)		
			Train	Validati on	Test	Train	Validati on	Test
A	1	8	65.9917	55.6643	68.5717	0.1951	0.2324	0.1939
B	0.9	10	70.2777	55.6643	67.6193	0.1890	0.2452	0.2073
C	0.8	12	72.6587	61.4923	73.3333	0.1740	0.2415	0.1971
D	0.7	15	75.5023	56.6990	68.5713	0.1646	0.2436	0.1955
E	0.6	21	80.2647	55.8280	68.5717	0.1436	0.2547	0.2102
F	0.5	27	81.6797	55.6643	68.5713	0.1219	0.2552	0.1951
G	0.4	33	85.9657	60.6210	70.4763	0.0982	0.2464	0.2041
H	0.3	39	87.8703	70.3703	72.3810	0.0822	0.2263	0.2037
I	0.2	48	90.7010	68.3553	68.5717	0.0649	0.2405	0.2324
J	0.1	57	95.7937	57.7887	68.5717	0.0310	0.3566	0.2533

c) Results of the RBF classifiers on all attributes

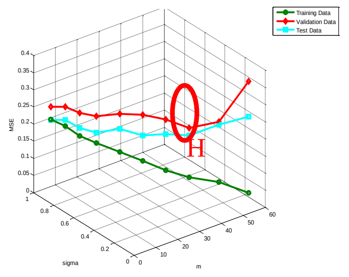
After analyzing the choice of models (A to J) with different parameters (sigma and basis function) along with their MSE plots shown in Fig. 3, the best model is chosen based on low validation and testing error. The best model should be the best compromise point between the training error and the validation error. Therefore, the model C is the best model for two data subsets: class-level and method-level (see Table IIa and 2b). For all attributes, model H is selected because it also has the best compromise point between training and validation error when compared to other models.



a) Class-level



b) Method-level



c) All attributes

Fig. 3. Mean square error (MSE)

A comparison among three data subsets shows that the selected model using the method-level dataset gives better result than others because this model has the lowest testing error, which is 0.1675, and it is less complex than models from other subsets of data.

C. Backpropagation Neural Network Model

The backpropagation neural network is applied, and the datasets are randomly partitioned in to 50% for training and another 50% for testing. Learning rate = 0.1 is assigned, and the starting point for the momentum is at 0.9. The feedforward backpropagation algorithm is repeated up to 2000 epochs to update weight and the obtained results are in Table III.

From Table IIIa)-IIIc), the class-level dataset with the network size 20 has the best result with the lowest MSE on testing data and the highest percent of correct classification, which are 0.165 and 81.69% respectively. For the method-level dataset, the network size 60 gives the best result with the MSE testing 0.17 and 72.7% correct classification. For dataset using all attributes, the network size 30 gives the best result with the MSE testing 0.177 and 80% correct classification. Among three datasets, the class-level has the best modeling result.

TABLE III: RESULTS OF THE NEURAL NETWORK CLASSIFIERS

Network size	**MSE training**	**MSE testing**	**% correct classification**
5	0.1560	0.1730	78.8800
10	0.1570	0.1660	80.2800
20	0.1480	0.1650	81.6900
30	0.1520	0.1840	77.4700
40	0.1470	0.1780	78.8800
50	0.1530	0.1650	78.8800
60	0.1480	0.1880	76.0600

a) Class-level

Network size	**MSE training**	**MSE testing**	**% correct classification**
5	0.1820	0.1960	69.8700
10	0.1790	0.1980	67.1300
20	0.1800	0.2000	68.4900
30	0.1770	0.1840	69.8700
40	0.1860	0.1840	68.4900
50	0.1770	0.1800	69.8700
60	0.1700	0.1700	72.7000
100	0.1700	0.1800	72.0000

b) Method-level

Network size	**MSE training**	**MSE testing**	**% correct classification**
5	0.1290	0.2000	75.4000
10	0.1280	0.1880	74.0000
20	0.1280	0.1960	74.0000
30	0.1220	0.1770	80.0000
40	0.1230	0.1840	76.8200
50	0.1230	0.1890	74.0000
60	0.1190	0.1930	79.7100

c) All attributes

D. Comparative Results of Models

When considering the values of MSE testing and percentage of correct classification, the models from the neural network size 20 is the best (MSE = 0.165 and accuracy = 81.69%). Beside, the models from the neural network have better results than the RBF in all datasets except the method-level. The neural network on the class-level dataset gives the lowest MSE testing and the highest percentage of correct classification. However, our observation is that to obtain almost the same MSE value, the models from the neural network approach are more complicate than the RBF because the models from the neural network approach have more number of hidden units than the RBF, leading to consume more processing time.

When considering the same size of network, the models from the neural network have better results than the RBF in all datasets except the method-level. The neural network has lower MSE testing and higher percentage of correct classification. Considering *m* = 10 in Table IIa) and the network size 10 shown in Table IIIa), the MSE testing of the model from the neural network is only 0.166 while one from the RBF is 0.214.

Overall, the neural network performs better than the RBF. To confirm the fact that the neural network has better percent of accuracy in most subsets of data, average percent of accuracy of three datasets are computed and the comparative results between two approaches are presented in the graph in Fig. 4.

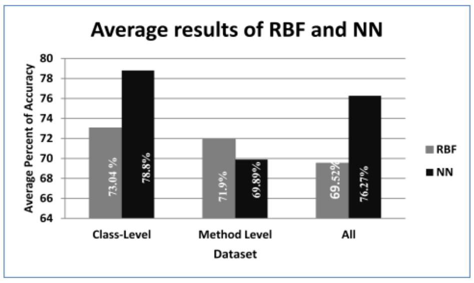


Fig. 4. Average percent of accuracy results of the RBF and the neural network (NN).

From Fig. 4, the neural network achieves 5.76% of accuracy higher than the RBF in the class-level. However, in the method-level, the RBF is slightly better than the neural network, which is 2.01% more accurate. In all attributes dataset, the *neural* network performs better than the RBF 6.75%.

V. Concluding Remarks

The software defect predictive models using the neural network and the RBF techniques are developed in our proposed method. Overall, the selected models from both approaches *offer* impressive result. The comparative results show that the performances are better by using subset of data with some features. Based on our results compared with the same size of network, the backpropagation neural network performs better classification than the RBF with lower MSE and higher percent of accuracy in most subsets of data. In the future work, the research will be focus on feature subset selection *on* software engineering dataset to seek for the best and simplest model as possible. It is therefore; conclude that the backpropagation *neural* network is the best classifier in our experiment. This approach is recommended to help to identify software fault-prone classes or modules that are required immediate attention from the software developers.

References

[1] KC1/Software defect prediction (2012, June 3). Retrieved from promise.site.uottawa.ca/SERepository/datasets.

[2] Z. Jianhong, S. P Sandhu, and S. Rani, "A Neural Network Based Approach for Modeling of Severity of Defects in Function Based Software Systems," *International Conference on Electronics and Information Engineering (ICEIE 2010)*, vol. 2, pp. 568-575.

[3] E. R. M. Bezerra, L. I. A. Oliverira, and R. L S. Meira, "A Constructive RBF Neural Network for Estimating the Probability of Defects in Software Modules," *Proceedings of International Joint Conference on Neural Networks 2007.*

[4] A. Goel, M. Shin, S. Ratanothayanon, and A. P. Raymond, "Parsimonious Classifiers For Software Quality Assessment," *High Assurance Systems Engineering Symposium, HASE '07. 10th IEEE.2007*, pp. 411-412.

[5] S. *A* Wagner, "Bayesian network approach to assess and predict software quality using activity-based quality models," *ACM* 2009.

[6] S. P. Sandhu, R. Goel, S. A. Brar, J. Kaur, and S. Anand, "A model for early prediction of faults in software systems," IEEE 2010, vol4, pp. 281-285.

[7] X. Tan, X. Peng, S. Pan, and W. Zhao, "Aessing Software Quality by Program Clustering and Defect Prediction," *IEEE* 2011, pp. 244-248.

[8] S. S. Kamaruddin, Yahaya, Jamaiah. A. Deraman, and R. Ahmad, "Feature Subset Selection Method for Dynamic Software Quality Assessment," *IEEE* 2011, pp. 304-306.

[9] L. Zhang and Z. Shang, "Classifying feature description for software defect prediction," *IEEE* 2011, pp.138-143.

[10] Q. Song, M. Shepperd, M. Cartwright, and M. Carolyn, "Software defect association mining and defect correction effort prediction," *IEEE transactions on Software Engineering*, 2006, vol. 32, no. 2, pp. 69-82.

[11] A. Goel and M. Shin, "Radial basis functions: An algebraic approach (with data mining applications)," *Tutorial at the 15th European Conference on Machine Learning, (ECML 2004)*, Pisa, Italy, 2004.

[12] T. Mitchell, Machine Learning: Neural Network (2012, May 10). Retrieved from [Online]. Available: http://www.cs.cmu.edu /project/theo-20/www/mlbook/ch4.pdf