Time Complexity Analysis of Support Vector Machines (SVM) in LibSVM

Abdiansah Abdiansah
Intelligent System Laboratory
Comp. Science Department
Sriwijaya University

Retantyo Wardoyo
Intelligent System Laboratory
Comp. Science & Electronic Dept.
Gadjah Mada University

ABSTRACT
Support Vector Machines (SVM) is one of machine learning methods that can be used to perform classification task. Many researchers using SVM library to accelerate their research development. Using such a library will save their time and avoid to write codes from scratch. LibSVM is one of SVM library that has been widely used by researchers to solve their problems. The library also integrated to WEKA, one of popular Data Mining tools. This article contain results of our work related to complexity analysis of Support Vector Machines. Our work has focus on SVM algorithm and its implementation in LibSVM. We also using two popular programming languages i.e C++ and Java with three different dataset to test our analysis and experiment. The results of our research has proved that the complexity of SVM (LibSVM) is O(n3) and the time complexity shown that C++ faster than Java, both in training and testing, beside that the data growth will be affect and increase the time of computation.

General Terms
Machines Learning, Support Vector Machines, LibSVM, Complexity

Keywords
SVM, LibSVM, C++, Java, WEKA, Data Mining

1. INTRODUCTION
Support Vector Machines (SVM) proposed by Vapnik in early 1990s is used for computational tool for supervised learning which has advantages than other methods in various types of application [18]. In the classification task, SVM leading than other methods because SVM provides a global solution for data classification. SVM gives a unique global hyperplane to separate data points for different classes. SVM used Structural Risk Minimization (SRM) principle to reduce risk during training phase. SVM generally used for classification and regression [1][3][6][10][11][16]. At the first time, SVM only be used for binary classification, but now it can be used for multi-class [7][15][9]. SVM is widely used for classification in the areas such as disease detection, text categorization, software defect, intruder detection, time-series forecasting, detection and others.

LibSVM is a programming library to facilitate researchers to perform SVM classification, it is developed by [8]. Many researchers used LibSVM to cut off software development process so that they only focus on data rather than tools. LibSVM also integrated into WEKA as a default SVM module. WEKA is an application as a tool for testing and evaluation, where it contains many algorithms for data mining areas such as: preprocessing, classification, clusterization, association, selection and attributes visualization. It makes LibSVM should be considered as one of SVM library. Many researchers used it both as experiments and real applications.

This paper contains LibSVM analysis using algorithm complexity (all routines in LibSVM) and it has tested using two popular programming languages i.e C++ and Java. Yields of experiment are expected to provide an information related to the complexity of the algorithm in LibSVM and knowing the running-time indicator of training and testing both for C++ and Java. Furthermore, this paper is organized as follows: Section 2 contains methodology are used in this experiment i.e data analysis, brief theory of SVM, LibSVM and the tools has been used. Section 3 contains analysis and experiment i.e finding main routines in LibSVM, compute the algorithms complexity and implementing it using C++ and Java. Section 4 is discussion about results and Section 5 is conclusion.

2. METHODOLOGY
2.1 Data Analysis
Generally, data is formed as datasets with a particular format. The following is data format used in LibSVM:

<label> <index1>:<value1> <index2>:<value2> ...

Where <label> is binary-class (-1, 1) or multi-class. <index> is attribute represent integer numbers from 1 to n. <value> is the value of attribute represent real numbers. There are three datasets used in this experiment i.e:

- LibSVM: heart_scale (binary-class, 270 records, 13 attributes)
- Hsu et al. [5]: train.3 (binary-class, 2.000 records, 22 attributes)
- WEKA: iris.train (multi-class, 150 records, 4 attributes).

WEKA using LibSVM to apply SVM algorithm, therefore we conducted an experiment to measure the accuracy of classification between (WEKA + LibSVM) versus (CPP + LibSVM) and (Java + LibSVM) using iris.train dataset, the result is same. The experiments show that LibSVM give consistent result even with different media. LibSVM also good in compatibility because it is use standar programming library.

2.2 Support Vector Machines
Support Vector Machines (SVM) is one of machine learning algorithms using supervised learning models for pattern recognition. SVM is often used for classification and regression analysis. [17] showed that SVM classification is quite good with accuracy above 80.0%. e.g, given a training set,

$$(X_i, Y_i), i = 1, \ldots, n,$$

where,

$$X_i = (x_i, \ldots, x_{id})$$

is a sample of d-dimension and $y_i \in \{1, -1\}$ is a label given to sample. The SVM task is to find a linear discriminant function $g(x) = w^T X + w_0$ so that,

$$w^T x_i + w_0 \geq +1 \quad \text{for} \quad y_i = +1$$

$$w^T x_i + w_0 \leq -1 \quad \text{for} \quad y_i = -1$$

solutions for these problems must satisfy the following equation:

$$y_i (w^T x_i + w_0) \geq 1 \quad i = 1, \ldots, n \quad (1)$$

optimal linear function can be obtained by minimizing the following quadratic programming problems [13]:

$$min \frac{1}{2} w^T w - \sum_{i=1}^{n} \alpha (y_1 (w^T x_i + w_0) - 1) \quad (2)$$

which will produce the following solutions:

$$w = \sum_{i=1}^{n} \alpha y_i x_i \quad (3)$$

where, $\{\alpha, i = 1, \ldots, n; \alpha \geq 0\}$ is Lagrange multipliers.

to make data separated linearly, feature space mapped into a high dimensional space. The technique used to perform mapping function is called Kernel. The kernel is a function:

$$k : \chi \times \chi \rightarrow \mathbb{R}$$

which took two samples from input space and mapped into a real number that indicates the level of similarity. For all,

$$x_i, x_j \in \chi,$$

then the Kernel function must satisfy:

$$k(x_i, x_j) = \langle \emptyset(x_i), \emptyset(x_j) \rangle \quad (4)$$

where $\emptyset$ is an explicit mapping from input space χ to features of dot product of space H [4]. To applied the Kernel into SVM, generally, the equation (2) were solved by the following equation:

$$max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \quad (5)$$

where $x_i \cdot x_j$ is inner product of the two samples is implicit kernel in the equation similarity measure between x_i and x_j inner product can be replaced with another kernel function so that the equation (5) will be as the following equation:

$$max \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \quad (6)$$

There are four basic types of kernels: linear, polynomial, radial basis function and sigmoid. SVM can be used for multi-class, which is using a strategy of one-against-one [14] which has been tested by [5] and the results are quite good.

2.3 LibSVM

LibSVM is a programming library for SVM algorithm was developed by [8], it is used researcher for classification and regression task. LibSVM also integrated into WEKA that contains a collection of machine learning algorithms for Data Mining. Existing algorithms in Weka can be used directly or invoked using Java library. LibSVM has been used for various areas starting from 2000 to 2010, more than 250,000 have download it and 10,000 emails from users who asked related to the library [2]. LibSVM support three functions i.e 1) SVC (Support Vector Classification - binary-class and multi-class), which can be used for classifications task; 2) SVR (Support Vector Regression), is used for regressions task; and 3) One-Class SVM, which is used for distribution estimation. This paper will only discuss LibSVM for classification because the concept of its libraries are the same for all functions.

In Figure 1, we can see the library organization in LibSVM for training process. svm_train is a main routine used to perform SVM training. svm_train_one which underneath is a routine to select one of three functions LibSVM (SVC, SVR and one-class SVM). Under svm_train_one there are various types of SVM functions it can be used depends on the choice of svm_train_one. These options produced a solving model for the data that has been trained earlier.

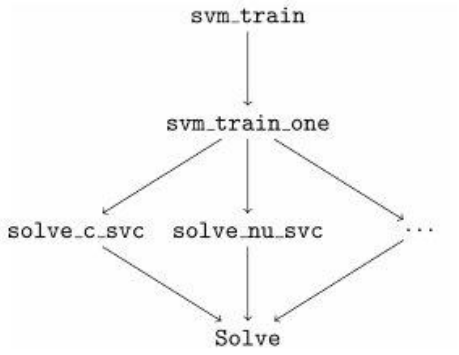

Figure 1. Library organization in LibSVM

Classification Accuracy

After training and testing phase, the accuracy is measured using the following equation:

$$\text{Accuracy} = \frac{\text{\# correctly predicted data}}{\text{\# total testing data}} x\,100\% \quad (7)$$

Short Example of LibSVM

LibSVM's technical tutorial can be read at README file and a paper written by [8]. This example is taken from [2], they used default LibSVM dataset (heart_scale) that contain 270 records is divided into 170 records for training (heart_scale.tr) and 100 records for testing (heart_scale.te). There are two executable files i.e svm_train to conduct training and svm_predict to classify. In Figure 2 and 3 shows the results of the execution of each applications.

```
$ ./svm_train heart_scale.tr
***
optimization finished, #iter = 87
nu = 0.471645
obj = -67, 299458, rho = 0.203495
nSV = 88, nBSV = 72
Total nSV = 88
Figure 2. Result of svm_train
```

Figure 2. Result of svm_train

svm_train automatically created heart_scale.tr.model file. The file will used as input by svm_predict.

```
$./svm_predict heart_scale.tr heart_scale.tr.model output
Accuracy = 83% (83/100) (classification)
Figure 3. Result of svm_predict
```

Figure 3. Result of svm_predict

LibSVM Code Organization

All of SVM's (LibSVM) algorithm for training and testing implemented by svm file (svm.cpp/svm.java). Both of svm_train and svm_predict that has discussed earlier is an example of user interface application. These application will call methods in svm file to perform classification task. Therefore, this paper will discuss the complexity of codes in the svm file.

2.4 Experimental Tools

In this experiment we used two of LibSVM's libraries i.e C++ and Java. Both libraries were tested using a computer with the following specifications: Intel (R) Core (TM) i5-3230M - 2.60 GHz (4 CPU), RAM - 4 GiB, LINUX operating system with Linux Mint KDE distro (Ubuntu Core - 14:04 LTS - 32 bit), NetBeans 8.0.2, 1.8.0 JDK for Java IDE and Code :: Block 13:12, GNU GCC compiler for C++.

3. ANALYSIS AND EXPERIMENTS

This section will discuss about analysis and experiments are conducted on LibSVM. Analysis was done by tracking codes of LibSVM's libraries, which are implemented in three files: svm_train, svm_predict and svm. Next is finding main routines and calculated its complexity using Big-O notation. Lastly is run the LibSVM application using C ++ and Java to see the results of running time.

3.1 Finding LibSVM's Routines

The experiments is conducted using default parameter and the type of problem is classification. Based on search of results, there are routines that are ignored, because it is not suitable for default parameter. This paper only examines some routines are used in the file svm_train, svm_predict and svm (as method not file). There are ten methods are analyzed and has computed its complexity. In general, relationship of the third of files can be seen in Figure 4.

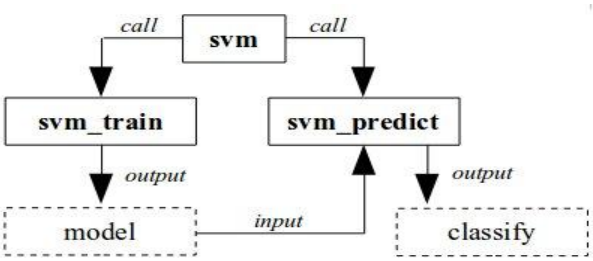

Figure 4. Relationship the third of files

In Figure 4, svm_train and svm_predict will call svm which contains routines of SVM algorithm. svm_train produce a model that will be the input for svm_predict. In svm_train there are several routines are analyzed and has computed i.e:

- parse_command_line
- read_problem, svm_train (call)
- svm_check_parameter (call)
- svm_save_model (call)

In svm_predict i.e:

- svm_load_model (call)
- svm_predict (call)
- Predict
- svm_check_probability_model (call)
- svm_predict_ probability (call)

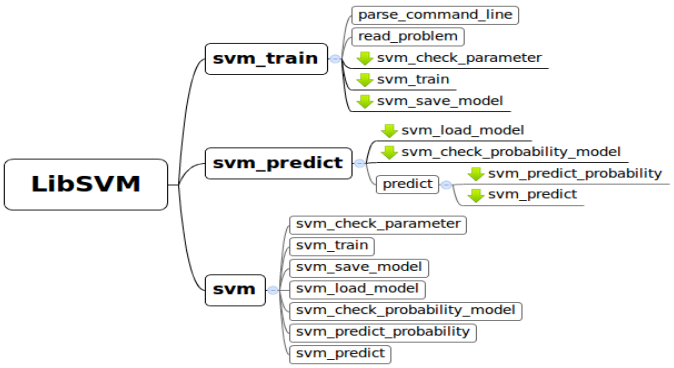

Figure 5. Hierarchy of methods in LibSVM

svm routines called by svm_train and svm_training and both marked with the symbol 'call in parentheses' and 'down arrow' (see Figure 5). In Figure 5, it can be seen the hierarchy of routines, e.g the main code of svm_train can be seen in Figure 6.

```
int main(int argc, char **argv) {
  char input_file_name[1024];
  char model_file_name[1024];
  const char *error_msg;

  parse_command_line(argc, argv, input_file_name,
  model_file_name);
```

```
read_problem(input_file_name);
error_msg = svm_check_parameter(&prob,&param);

if(error_msg) {
  fprintf(stderr,"ERROR: %s\n",error_msg);
  exit(1);
}
if(cross_validation)
  do_cross_validation();
else {
  model = svm_train(&prob,&param);
  if(svm_save_model(model_file_name,model)) {
    fprintf(stderr, "can't save model to file %s\n",
    model_file_name);
    exit(1);
  }
svm_free_and_destroy_model(&model);
}
svm_destroy_param(&param);
free(prob.y); free(prob.x);
free(x_space); free(line);
return 0;
}
```

Figure 6. svm_train main code

In Figure 6 can be seen that the routines were analyzed (in bold) and are not analyzed (crossed out). do_cross_validation routine is one example of a routine is not analyzed, because cross_validation variable contains the 0 (false) which is default value. Figure 7 shows the default parameters in the file svm_train.

```
// default values
param.svm_type = C_SVC;
param.kernel_type = RBF;
param.degree = 3;
param.gamma = 0;     // 1/num_features
param.coef0 = 0;
param.nu = 0.5;
param.cache_size = 100;
param.C = 1;
param.eps = 1e-3;
param.p = 0.1;
param.shrinking = 1;
param.probability = 0;
param.nr_weight = 0;
param.weight_label = NULL;
param.weight = NULL;
cross_validation = 0;
```

Figure 7. Default parameters of LibSVM

3.2 Algorithm Complexity

The complexity of an algorithm generally calculated using Big-O notation. Complexity can be divided into two kinds of complexity i.e: 1) time complexity, deal with how long the algorithm is executed, and 2) space complexity, deal with how much memory is used by it's algorithm. In this paper we only discussed time complexity. An algorithm will process amounts of data, where N is a symbol of amounts of data. If an algorithm does not depend on N then the algorithm has constant complexity or symbolized by O(1) (Big-O one). On the contrary, if the algorithm is dependent on N, the complexity depends on line code in algorithm and it is can be O(n), O(n2), O(log n) and others.

To explain the calculation of Big-O that has used in this article, we will give an example to compute complexity of svm_check_parameter. The code snippets can be seen in Figure 8.

```
for(i=0;i<nr_class;i++) {
  int n1 = count[i];
  for(int j=i+1;j<nr_class;j++) {
    int n2 = count[j];
    if(param->nu*(n1+n2)/2 > min(n1,n2)) {
      free(label);
      free(count);
      return "specified nu is infeasible";
    }
  }
}
```

```
for(i=0;i<nr_class;i++) {
  ...
  for(int j=i+1;j<nr_class;j++) {
    ...
  }
}
```

Figure 8. Code snippets of svm_check_parameter

Figure 8 is divided in two parts i.e top and bottom. The upper part contains a complete code snippet and the lower part contains an incomplete piece of code that simply take the code for nesting. The code will be considered as constant or the complexity is O(1) if it's not loop. Contrary, the complexity affected by lower bound, upper bound and total number of loop iteration (N). Next is to assign nr_class with 5 then find the total of interation on the existing code in Figure 8 (bottom), Tabel 1 shown the simulation of iterations.

Table 1. Loop Simulation

i	**j**	**Numbers of Iteration**
0	1, 2, 3, 4	4 times
1	1, 2, 3	3 times
2	1, 2	2 times
3	1	1 time
4	0	-
Total of Iteration		10 times

rom these simulations we made equation based on analytic experiment for number of N data, which is formulated as follows:

$$\sum_{i=0}^{n-1} (n-1)-i \quad (8)$$

where N (nr_class) is the numbers of data and i is the counter variable. To search the complexity, equation (8) must be first converted into form of equation using sigma equation, the equation broken down to be:

$$\sum_{i=0}^{n-1} (n-1)-i = \sum_{i=0}^{n-1} (n-1) - \sum_{i=0}^{n-1} i \quad (9)$$

equation (9) can be parsed becomes,

$$(n-1)^2 - \tfrac{1}{2}(n-1)((n-1)-1)$$

$$(n-1)^2 - \tfrac{1}{2}(n-1)(n-2)$$

$$(n^2-2n+1)-(\tfrac{1}{2}(n^2-3n+2))$$

$$\frac{(2n^2-4n+2)-(n^2-3n+2)}{2}$$

$$\frac{n^2-n}{2} \ldots (9)$$

Equation (9) is a formula to find total number of iteration from numbers of data (N). Based on equation (9), the complexity of code snippet in Figure 8 is: O (n2/2) or O (n2). Complexity sought to take a significant variable, so in this case n2 is more significant than the n. Complexity for svm_train, svm_predict and svm can be seen in Figures 9, 10 and 11. Tabel 2 shows total of complexity for all methods in SVM.

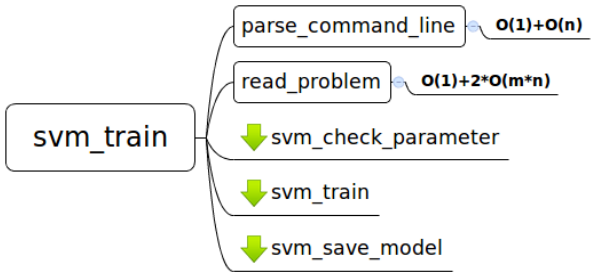

Figure 9. svm_train complexity

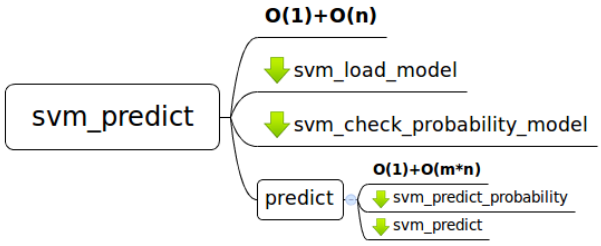

Figure 10. svm_predict complexity

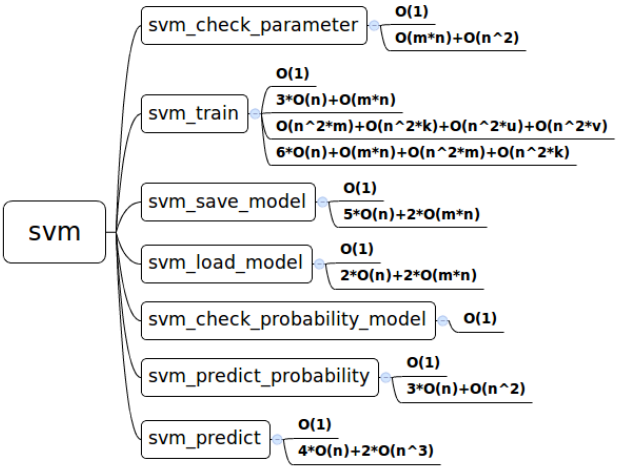

Figure 11. svm complexity

Table 2. Total Of Complexity

Num.	Methods	Complexity (Big-O)
1	parse_command_line	O(1)+O(n)
2	read_problem	O(1)+2*O(m*n)
3	svm_predict (main) dan predict	O(1)+O(n)+O(n*m)
4	svm_check_parameter	O(1)+O(m*n)+O(n^2)
5	svm_train	O(1)+9*O(n)+2*O(m*n)+ 6*O(n^2*m)
6	svm_save_model	O(1)+5*O(n)+2*O(m*n)
7	svm_load_model	O(1)+2*O(n)+2*O(m*n)
8	svm_check_probability_model	O(1)
9	svm_predict_probability	O(1)+3*O(n)+O(n^2)
10	svm_predict	O(1)+4*O(n)+2*O(n^3)

Table 2 shows some methods in LibSVM where svm_predict provide the highest complexity is O(n^3). Actually, svm_train also have the same complexity with svm_predict, but in this case we limit our computation only to main routine without any further search the sub-routine, and then some routine in svm_train considered as O(1). Our analysis shows that SVM algorithm requires three loops in nested loop so that the complexity of SVM is O(n^3). This analysis in accordance to the standards of SVM complexity [12].

3.3 LibSVM Implementation

LibSVM implemented using two programming languages are C ++ and Java. The reason for choosing these languages are subjective based on general knowledge and opinion that both languages are quite widely used among researchers. This implementation merely shows LibSVM performance against tested datasets to see how much time it took to perform the classification (running time). The time is recorded from the beginning execution until the end using TIME aplication (default application on Linux), this technique referring to [5].Tables 3 and 4 is results of implementation using C++ and Java with three sample dataset. Experiments were conducted to look for running-time of training and testing. In

this experiment, we used training data for data test. Tables 5 and 6 give results of running-time for train.3 dataset which is divided into five subsets i.e: 400 records, 800 records, 1200 records, 1600 records and 2000 records.

Table 3. Running Time In C++

Datasets	Training Time (sec.)	Testing Time (Sec.)
heart_scale	0.018	0.013
train.3	0.752	0.598
iris.train	0.007	0.005

Table 4. Running Time In Java

Datasets	Training Time (sec.)	Testing Time (Sec.)
heart_scale	0.145	0.122
train.3	1.557	1.185
iris.train	0.112	0.102

Table 5. Running Time Of Train.3 Using C++

Datasets	Training Time (sec.)	Testing Time (Sec.)
train.3 (400)	0.047	0.041
train.3 (800)	0.145	0.117
train.3 (1200)	0.392	0.301
train.3 (1600)	0.535	0.435
train.3 (2000)	0.752	0.598

Table 6. Running Time Of Train.3 Using Java

Datasets	Training Time (sec.)	Testing Time (Sec.)
train.3 (400)	0.255	0.182
train.3 (800)	0.428	0.339
train.3 (1200)	0.657	0.485
train.3 (1600)	0.928	0.689
train.3 (2000)	1.557	1.185

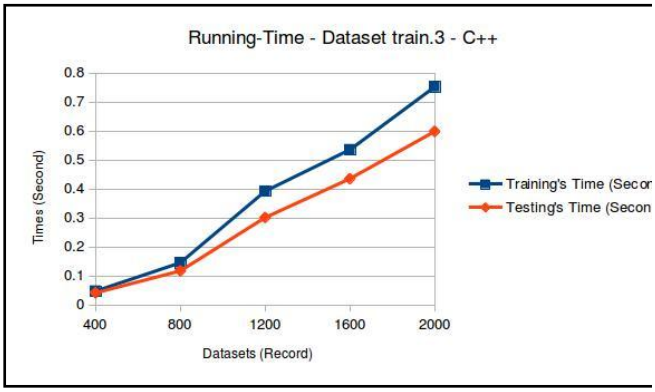

Figure 12. Running-time graphics using C++

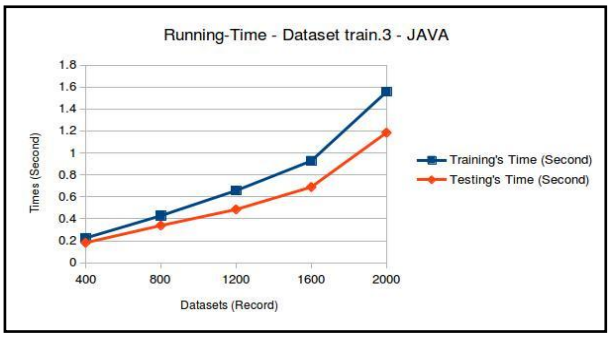

Figure 13. Running-time graphics using Java

Figures 12 and 13 shows running-time graph for C++ and Java with train.3 datasets that are divided into five subsets. Based on the chart it can be concluded that the running-time of C++ is faster than Java, both for training and testing. In the graph also can be seen that the data growth will affects the running-time, if there are more data then it need more time.

4. DISCUSSION

The experiments for LibSVM complexity have been done and the results has been obtained, but the calculation is not involve all the existing routine. Experiments in LibSVM restricted to default parameters such mentioned previously. There are several LibSVM functions were not counted because by default the functions is not executed. Furthermore, the implementation of LibSVM done by re-compile the original library. Experiments showed that LibSVM portability is very good so that it is not difficult to re-implemented. Problems arise when we have different results if we run more than once, but the difference is not significance. In order to obtain an average time we run more than once and use three digits behind comma to get high-precision.

There are two scenarios of experiment that aimed to see how the results of running-time. The first scenario uses three datasets are: heart scale, train.3 and iris.train, where the results of the experiment showed that the C++ is faster than Java because it C++ is native. Testing's time is smaller than training's time and a large data will increasing computation time. In second scenario, experiments focus on the data train.3 were divided into five subsets, it is aims to look at the effect of running-time where the data is growing up to be bigger.

5. CONCLUSION

Support Vector Machines is one of machines learning using supervised learning as knowledge training. In the classification task, SVM is more favored than the other methods because SVM provides a global solution for data classification. To facilitate researchers using SVM algorithm, Lin et al. develop LibSVM that has been widely used by researchers and has been integrated into WEKA. This paper contains analysis of LibSVM by doing such experiments: compute the complexity of algorithm and implementing using two programming language C ++ and Java. Experimental has obtained results that the running-time using C ++ is faster than Java because C++ is native. The results also showed that the running-time for training and testing with dataset train.3 is rise quadratic. Broadly speaking, the experimental results may indicate that the running-time of testing is smaller than training.

6. REFERENCES

[1] Agarwal. S (2011). Weighted support vector regression approach for remote healthcare monitoring. In 2011 International Conference on Recent Trends in Information Technology (ICRTIT), IEEE, pp. 969–974.Che, JinXing. (2013). Support vector regression based on optimal training subset and adaptive particle swarm optimization algorithm, Appl. Soft Comput. 13 (8), pp. 3473–3481.

[2] Chang, C. C., & Lin, C. J. (2011). LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27.

[3] Gunn S, R . (1998). Support vector machines for classification and regression, ISIS technical report 14.

[4] Hofmann, T., Schölkopf, B., & Smola, A. J. (2008). Kernel methods in machine learning. The annals of statistics, 1171-1220.

[5] Hsu, C. W., & Lin, C. J. (2002). A comparison of methods for multiclass support vector machines. Neural Networks, IEEE Transactions on, 13(2), 415-425.

[6] Huang, Weimin, Leping Shen. (2008). Weighted support vector regression algorithm based on data description. In ISECS International Colloquium on Computing, Communication, Control, and Management CCCM'08, vol. 1, IEEE, pp. 250–254.

[7] Lee, Y, Lin, Y, G. Wahba. (2001). Multicategory support vector machines. Comput. Sci. Stat. 33, pp. 498–512.

[8] Lin, C. J., Hsu, C. W., & Chang, C. C. (2003 – Last updated: April 15, 2010). A practical guide to support vector classification. National Taiwan U., www. csie. ntu. edu. tw/cjlin/papers/guide/guide. Pdf.

[9] Nemmour, H, Chibani, Y. (2006). Multi-class SVMs based on fuzzy integral mixture for handwritten digit recognition. Geometric modeling and imaging—new trends, pp. 145–149.

[10] Suykens, A.K. Johan, Brabanter Jos De, Lukas Lukas, Vandewalle Joos. (2002). Weighted least squares support vector machines: robustness and sparse approximation. Neurocomputing 48 (1) 85–105.

[11] Tomar, Divya, Arya Ruchi, Agarwal Sonali. (2011). Prediction of profitability of industries using weighted SVR. Int. J. Comput. Sci. Eng. 3 (5) pp. 1938–1945.

[12] Tsang, I. W., Kwok, J. T., & Cheung, P. M. (2005). Core vector machines: Fast SVM training on very large data sets. In Journal of Machine Learning Research (pp. 363-392).

[13] Vapnik, V. (2000). The nature of statistical learning theory. Springer Science & Business Media.

[14] Webb, A. R. (2002). Statistical pattern recognition, 2nd Edition. John Wiley & Sons.

[15] Weston, J, Watkins, C. (1998). Multi-class support vector machines. CSD-TR-98-04 royal holloway, University of London, Egham, UK.

[16] Xue, Zhenxia, Liu Wanli. (2012). A fuzzy rough support vector regression machine. In 2012 9th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), Dover, pp. 840–844.

[17] Zhang, D., & Lee, W. S. (2003). Question classification using support vector machines. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval (pp. 26-32). ACM.

[18] Zhu, G., Huang, D., Zhang, P., & Ban, W. (2015). ε-Proximal support vector machine for binary classification and its application in vehicle recognition. Neurocomputing, 161, 260-266.