An Implementation of Hierarchical Multi-Label Classification System
User Manual

by

5331028421 Thanawut Ananpiriyakul

5331039321 Piyapan Poomsilivilai

Supervisor

Dr. Peerapon Vateekul

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

January 2014

Table of Contents

This document will tell you how to use "Hierarchical Multi-Label Classification System" To use this system, there are three main steps; dataset preparation, parameter configuration, and system execution. Before that, you can download the source code from https://sites.google.com/site/hrsvmproject/home.

1 Dataset Preparation

This section will describe the format of dataset input file. Note that file input **should not contain any extra space and/or blank line beyond the given format**.

1.1 Binary and Multi-class Dataset

The input data of is much like LIBSVM format. Each line contains class part and feature part followed the syntax on Figure 1.

According on Figure 1, each line describes one training example. Each line consists of 2 parts: class part and feature part. Class part describes classes to which example belongs. Feature part describes value of each feature which is in format of "*X*:*Y*" separated by space, where *X* is feature identifier and *Y* is feature value. Note that the non-specified feature will be treated as zero.

For example, Figure 2 indicates that there are 2 examples. First example belongs to class 'spam' which feature 1 has value of 0.0024 and feature 5 has value of 0.2141. The non-specified features are 2, 3, and 4 which all have value of 0. Second example belongs to class 'non-spam' which features 1, 2, 3, 4, and 5 have value of 1.234, 0.1221, 0.11, -0.12, and 0.7903 respectively.

```
<class> <feature>:<value> <feature>:<value> …
```

Figure 1. A template of data file for binary classification

```
spam 1:0.0024 5:0.2141
non-spam 1:1.234 2:0.1221 3:0.11 4:-0.12 5:0.7903
```

Figure 2. An example of data file for binary classification

1.2 Multi-Label Dataset

For multi-label dataset, the format is much like in binary dataset except class part which is allowed to have multiple classes. The syntax is depicted in Figure 3. Each line describes one training example. The class part is allowed to have multiple classes separated by comma. The feature part syntax is identical to binary classification dataset.

For example, Figure 4 indicates that this example belongs to class "action", "comedy", and "romantic" which feature 1 has value of 0.0024 and feature 5 has value of 0.2141. The non-specified features are 2, 3, and 4 which all have value of 0.

```
<class>,<class>,… <feature>:<value> <feature>:<value> …
```

Figure 3. A template of data file for multi-label and multiclass classification

```
action,comedy,romantic 1:0.0024 5:0.2141
```

Figure 4. An example of data file for multi-label classification

1.3 Hierarchical Multi-Label Dataset

For hierarchical multi-label classification, two files are needed. First is data file and second is hierarchy description. Data file has the same format as in multi-label classification. Hierarchy description describes hierarchical structure of classes. See a template in Figure 5. Each line contains a pair of parent node and child node separated by white space.

For example, Figure 6 indicates that class "action" and "romantic" are the children of class "movie".

```
<parent> <child>
```

Figure 5. A template of hierarchy description for hierarchical multi-label classification

```
movie action
movie romantic
```

Figure 6. An example of hierarchy files for hierarchical multi-label classification

2 Parameter Configuration

The interface of this system is command-line interface. As a traditional command-line interface software, user can easily configure the system using several options and matching parameters provided by the system. In this system, we provide a lot of useful options to make it more than complete. The list of options is following.

Table 1. A complete list of training options which provided by the system.

Option	Parameter	Description
-t*	kernel_type	Set type of kernel function (default 2) 0 linear 1 polynomial 2 radial basis function 3 sigmoid
-k*	classification_type	Set the classification type (default binary) binary binary classification multiclass multi-class classification multilabel multi-label classification hierarchical <hierarchical file> hierarchical classification Note that in case of hierarchical mode, you need to specify hierarchical file next to this option.
-a	enable_R-SVM	Enable/disable R-SVM algorithm (default 0) 0 disable 1 enable
-f	enable_feature_selection	Enable/disable feature selection algorithm (default 0) 0 disable 1 enable
-l	minimum_example_in_class	Set the threshold which program will remove classes that have number of examples lower than threshold (default 0)
-d*	degree	Set degree in kernel function (default 3)
-g*	gamma	Set gamma in kernel function (default 1/num_features)

Option	Parameter	Description
		num_features is the number of features of dataset.
-r*	coef0	Set coef0 in kernel function (default 0)
-c*	cost	Set the parameter C of C-SVC (default 1)
-m	cache_size	Set cache memory size in MB (default 100)
-e	epsilon	Set tolerance of termination criterion (default 0.001)
-v	n	n-fold cross validation mode The system randomly splits the data into n parts and calculates cross validation accuracy on them.
-x	n	specify random seed in k-fold cross validation mode (default: random)

* option means that the option will automatically set in hierarchical classification mode. The system uses one-vs-the-rest strategy [1] and automatically select appropriate parameter on each class under following criteria,

Table 2. Kernel parameter on each class in hierarchical classification

Class criteria	Kernel parameter
#example is less than #feature	Linear kernel
Otherwise	Radial basis (Gaussian) kernel with gamma = 0.1

Table 3. C Parameter on each class in hierarchical classification

Class criteria	C parameter
#positive example is less than 15%	C = 2.0
Otherwise	C = 1.0

3 System Execution

After finishing dataset preparation and parameter, the next process is system execution. You can directly execute the system via command line (Windows platform) or terminal (Linux platform).

Figure 7 shows an overview of training process. First, user must provide path to training data, preferred options in Table 1 and classifier output path. If you want to run the system in hierarchical mode, path to hierarchy description must be provided. User is required to use svm-train.exe (svm-train in Linux) and execute a command in the format shown in Figure 8.

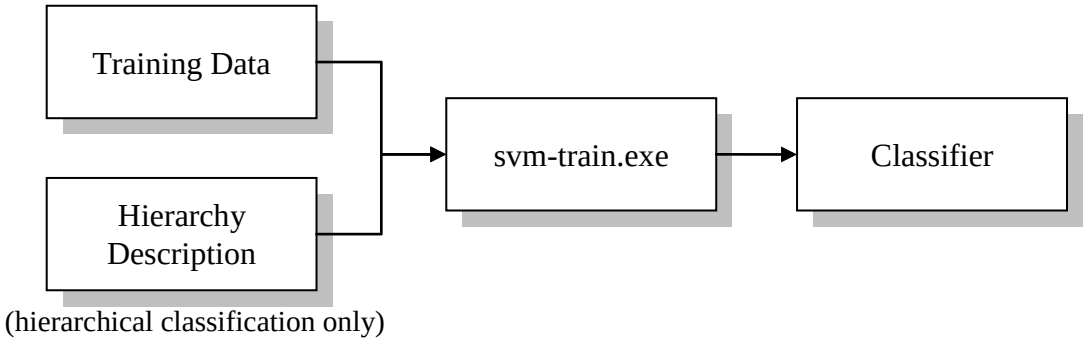

Figure 7. An overview of training process

```
svm-train <options> <training data> <model output folder>
```

Figure 8. A template of training command

For example, Figure 9 indicates that svm-train will run in binary mode. The classifier kernel is radial basis with C=1 and gamma=0.1. Training data path is "C:\training.dat". The classifier folder will be produced at "C:\model".

Here is another example; Figure 10 indicates that svm-train will run in hierarchical mode. Hierarchy description path is "C:\my_hierarchy.h". R-SVM is enabled. The classes that contain examples lower that 50 will be removed. Training data path is "C:\training.dat". The classifier folder will be produced at "C:\model".

```
svm-train -k binary -a 1 -t 2 -c 1 -g 0.1
"C:\training.dat" "C:\model"
```

Figure 9. An example of training command

```
svm-train -k hierarchical "C:\my_hierarchy.h" -a 1 -l 50
"C:\training.dat" "C:\model"
```

Figure 10. An example of training command

After finished training process, the system will produce classifier as a folder. Classifier folder contains classifier for each class in training data.

For predicting process, user must provide testing data path to be predicted, classifier path, and prediction output path. The data input must have the same format as in Figure 1. If user does not know the true classes to which each example belongs. User can specify arbitrary classes for each example and ignore the accuracy result that is produced in predicting process.

Figure 11 shows an overview of testing process. User is required to use svm-predict.exe (svm-predict in Linux) and execute a command in the format shown in Figure 12.

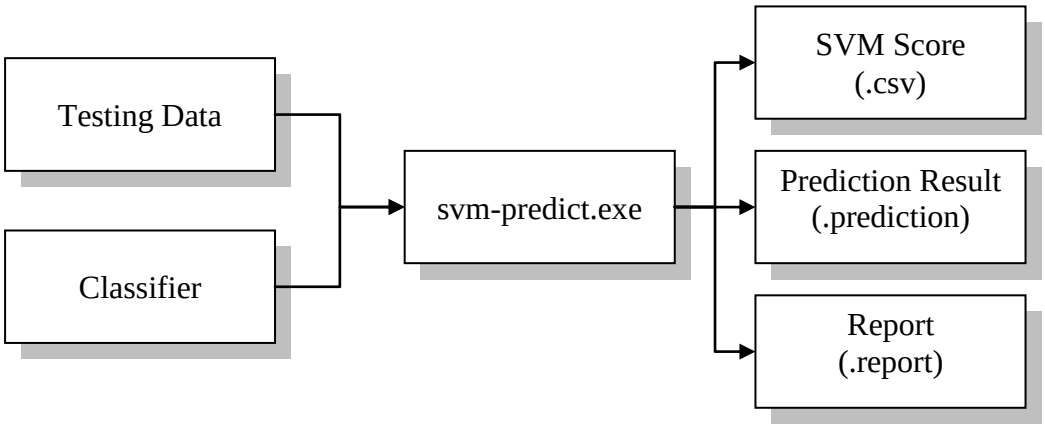

Figure 11. An overview of testing process

```
svm-predict <testing data> <classifier> <prediction output>
```

Figure 12. A template of prediction command

The example in Figure 13 indicates that svm-predict will predict file “data.dat” by using classifier in “model/train.model”. The system will provide three outputs: SVM-score, prediction result, and report in “data.csv”, “data.prediction”, and “data.report” respectively.

- SVM-score file contains SVM-score of every class in CSV format.
- Prediction result file contains class labels that system predicted for each example. The *i*-th line in this file is the prediction class of *i*-th example in data file. If there are more than one predicted class (e.g. multi-label classification), predicted class will be separated by comma.
- Report file contains performance measurement of each class, and summary of all measurement. Note that, this file may vary in different classification type.

```
svm-predict data.dat model/train.model data.prediction
```

Figure 13. An example of prediction command

4 Running Example

This section provides you a running example on iris multiclass dataset.

Figure 14 shows some part of iris dataset which contains four features and three classes: Iris-setosa, Iris-versicolor, and Iris-virginica.

```
Iris-setosa 1:5.1 2:3.5 3:1.4 4:0.2
Iris-setosa 1:4.9 2:3 3:1.4 4:0.2
Iris-versicolor 1:7 2:3.2 3:4.7 4:1.4
Iris-versicolor 1:6.4 2:3.2 3:4.5 4:1.5
Iris-virginica 1:6.3 2:2.5 3:5 4:1.9
Iris-virginica 1:6.5 2:3 3:5.2 4:2
```

Figure 14. The iris multiclass dataset

Figure 15 shows the command to start the system for iris dataset. Author chose Gaussian SVM-kernel with C=1 and gamma=0.1. After finished training process, the model folder named “iris_model” will created in path “./model”.

Figure 16 shows the prediction process of iris model. Author chose to predict “iris.svm” which is the same dataset in training process.

```
svm-train -k multiclass -a 1 -t 2 -c 1 -g 0.1
./iris.svm ./model/iris_model
```

Figure 15. An example of training command for iris dataset

```
svm-predict ./iris.svm ./model/iris_model ./iris.prediction
```

Figure 16. An example of predicting command for iris dataset

After finished predicting process, the system will output SVM-score file named “iris.prediction.csv” as in Figure 17, report file named "iris.prediction.report” as in Figure 18, and prediction result file named “iris.prediction” as in Figure 19.

Iris-setosa	Iris-versicolor	Iris-virginica
1.28623	-1.541	-1.53522
1.18457	-1.40986	-1.46509
1.29837	-1.66068	-1.31334
1.17262	-1.43732	-1.39529
1.31238	-1.61457	-1.48389
1.08801	-1.13735	-1.8286
1.27339	-1.6304	-1.33823
1.22963	-1.4199	-1.57736
1.15747	-1.52236	-1.22393
1.18353	-1.36995	-1.52319
1.21696	-1.38534	-1.68991

Figure 17. Part of iris SVM-score file

```
Model File : ./model/iris.model/iris.model
#Examples : 150

Overall classes
Accuracy        : 0.982222
Micro Precision : 0.973333
Macro Precision : 0.973333
Micro Recall    : 0.973333
Macro Recall    : 0.973333
Micro F1Score   : 0.973333
Macro F1Score   : 0.973333
Test Time       : 0 sec.

Measures of each classes
Class Iris-setosa
Accuracy  : 1
Precision : 1
Recall    : 1
F1Score   : 1

Class Iris-versicolor
Accuracy  : 0.973333
Precision : 0.96
Recall    : 0.96
F1Score   : 0.96

Class Iris-virginica
Accuracy  : 0.973333
Precision : 0.96
Recall    : 0.96
F1Score   : 0.96
```

Figure 18. Report file of iris dataset

```
Iris-versicolor
Iris-versicolor
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Iris-virginica
Iris-versicolor
Iris-virginica
Iris-virginica
```

Figure 19. Part of iris prediction result file

5 References

1. Peerapon Vateekul, Miroslav Kubat, and Kanoksri Sarinnapakorn, “Top-Down Optimized SVMs for Hierarchical Multi-Label Classification: a Case Study in Gene Function Prediction,” Intelligent Data Analysis, 2013 (accepted on March 1, 2013).