Project Report

Huu-Phuc Vo

April 19, 2016

1 Overview

In this project, we aim to transform the sequential *SVMLight* to parallel version using Encore programming language. The report firstly introduces the background of the project such as object-oriented parallel programming language Encore, ATLAS Higgs Boson dataset [1], and *SVMLight* library. Then the approach of how to achieve the goal and the implementation will be shown. Ultimately, sequential and parallel versions of *SVMLight* can be evaluated by different ATLAS Higgs Boson datasets. Afterward we can come up with conclusions about the correctness, data scalability, and performance between two versions.

2 Background

Firstly, the dataset of ATLAS Higgs Boson Machine Learning Challenge 2014 [1] will be used for training and testing the library. The goal of the Higgs training data is to distinguish a signal process which produces Higgs bosons from a background process which does not [5]. Secondly, among several classification and regression approaches [7], Support Vector Machine (SVM) [9] is one of the representatives since its popularity. With the given training dataset which is marked as belonging to one or two categories, a model will be built by SVM to predict whether a new input belongs to one category or the other. Thirdly, we have been developing an object-oriented parallel programming language called Encore [2] that supports developers to achieve parallelism-by-default using active and passive objects. Active objects are instances of active classes, and can possess their own single logical thread of control with asynchronous method calls. Whereas the passive objects are constructed from passive classes, and do not have a thread of control. In addition, method calls on passive objects will be executed synchronously. Passive objects can be asynchronously passed between active objects as data.

3 Approach

Our approach is to transform an SVM library from sequential version to the parallel one by taking advantage of Encore programming language. *SVMLight* [8] has been chosen to transform from sequential to parallel library since it is among the top 3 popular SVM libraries that are widely used. In order to construct the parallel *SVMLight* library in Encore language, both training and testing ATLAS Higgs Boson datasets should be represented as passive objects. At the same time, sequential methods of the *SVMLight* library can be correspondingly implemented as parallel methods of active objects in Encore. An interface of *SVMLight* is implemented and packed in a dynamic library in C so that the Encore programming language can load and invoke the functions from the library.

4 Contributions

The aim of the project is to make a parallel version of *SVMLight* using object-oriented parallel programming language Encore. There are three main phases to construct parallel *Encore-SVM* version, *converting*, *learning* and *classifying*. Among three phases, the parallelism can be applied to *classifying* and *converting* phases as all the events in the testing converting datasets are independent. The *learning* phase, however, is difficult to parallelize since all the events in the training dataset are dependently used to construct the learned model.

Contributions The following contributions are achieved in the project:

i. Propose an approach that can convert the csv data format to svm data format.

ii. Implement an *Encore-SVM* version with partial parallel functions (*converting* and *classifying*).

iii. Classify signal to background processes from the given Higgs dataset using *Encore-SVM* version.

iv. Validate the performance and accuracy of original *SVMLight* and *Encore-SVM* versions.

v. The implementation can be found at SVM_Encore repository on Github.

There are 33 different features in Higgs Boson training dataset including label feature which is `s` for signal or `b` for background value. When converting the features from *csv* format, all values are converted exactly the same, but the value of label is changed to `+1` with `s` and `-1` with `b`. The *svmlight* format is defined in Listing 1. Each features/values pairs are separated by a space character and ordered by increasing feature number.

Listing 1: SVMLight data format

```
<line>    .=. <target> <feature>:<value> ... <feature>:<value> # <info>
<target>  .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value>   .=. <float>
<info>    .=. <string>
```

The given Higgs Boson datasets are firstly converted from *csv* format to *svmlight* format. At the moment, the FileIO functions such as reading and writing files are missing in Encore, so they are implemented and packed in the IO Encore library. The IO library can run in 2 different modes, parallel and sequential. By default, the sequential mode will be activated, the parallel mode that will automatically split the input file into n-chunks, however, can be specified by using `-chunk` flag. Each feature of `convert` is specified by given values of `-chunk` flag that is described in Listing 2.

Listing 2: Encore convert function

```
./svm_encore convert -chunk k -ei train.csv -eo train.svm -vv
            k = 1 : convert from csv to svmlight.
            k > 1 : convert and split to n files.
            k < 1 : merge n files to 1 file.
    -ei : input file.
    -eo : output file.
    -vv : verbosity mode.
```

The categories of kernel function can be decided by passing kernel options such as linear (by default), polynomial, radial basis function expression, sigmoid tanh, or user defined kernel. Additional information of kernel can be added to the string `<info>`. The kernel can be customized by other implementation of the kernel function in `custom_kernel()` that currently returns `1.0` by default.

In order to make parallel version of the *SVMLight* in Encore, a C-*API* for both *learning* and *classifying* is necessary for creating an `.so` dynamic library with all methods that can be invoked from Encore. The two main functions of *SVMLight*, *svm_learn* and *svm_classify*, are used to train the model from given input data, and produce the result by taking the trained model and testing data as inputs, respectively. The *Encore-SVM* can also take all 28-`[options]` arguments for learning phase exactly as the original version.

5 Implementation

This section explains the implementation of convert, svm_learn, and svm_classify features. In Encore, active classes instantiate active objects that can possess their single logical thread of control with asynchronous method calls. When executing, the main program parses the given arguments and stores in passive object *Params*. Then the corresponding commands such as *svm_encore svm_learn*, *svm_encore svm_classify*, or *svm_encore convert* can be executed.

To obtain the parallelism, the following active and passive classes are being implemented.

Active classes The instances of those active classes are active objects as their method calls can be run asynchronously.

i. *Converter.enc*: The converter takes the parameters from *Params.enc*, and has three different functions

- `convert()`: convert file from `csv` to `svm` format without splitting file.
- `split()`: convert to `svm` format and split input file into n-chunks.
- `join()`: merge n-chunks to 1 file only.

ii. *Core.enc*: This is the bridge that connects with the *.so* library. It takes the method name, the arguments, and sends the method call to the *.so* library.

- `load()`: loads the dynamic shared library with the given path.
- `call()`: sends the method call to shared library by specifying method names, and number of arguments. If the number of argument is `0`, the method call has no return value.

iii. *Main.enc* The main object takes the given arguments to initialize the *Param* object. Then the shared library is able to be loaded, and the *svm_encore svm_learn*, *svm_encore svm_classify*, or *svm_encore convert* function can be extended depending on the given options.

Passive class The passive object *Params* is used to send to and receive from other active objects as data.

i. *Params.enc* The passive object is stored all 28 parameters of *svm_encore svm_learn*, 3 parameters of *svm_encore svm_classify*, and 14 parameters of *svm_encore*. Those parameters can be used by all active objects aforementioned.

6 Validation

Firstly, in order to validate the correctness and the accuracy of *Encore-SVM*, 3 examples of *SVMLight* library will be used to evaluate and compare the results between original and parallel versions. Figure 1 and 2 illustrates the training and testing datasets with 1000 positive and 1000 negative examples in the training phase, and with 600 test examples, respectively. The Figure 1a and 2a show the converting, training and classifying results of *Encore-SVM*. While Figure 1b and 2b display the training, and classifying results of SVMlight without converting as the datasets are already in `svm` format.

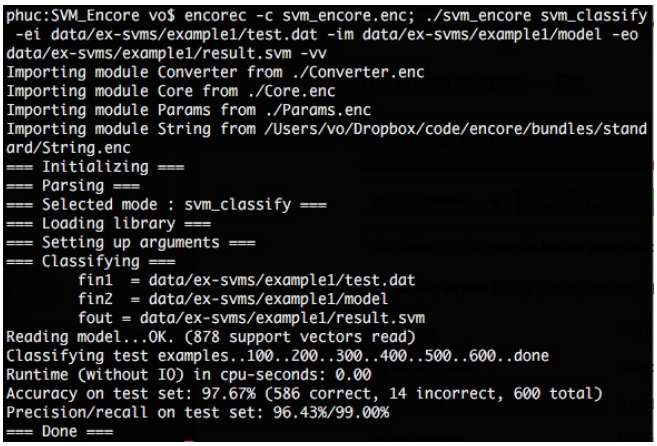
```
phuc:SVM_Encore vo$ encorec -c svm_encore.enc; ./svm_encore svm_learn -ei data/ex-svms/
example1/train.dat -eo data/ex-svms/example1/model -vv
Importing module Converter from ./Converter.enc
Importing module Core from ./Core.enc
Importing module Params from ./Params.enc
Importing module String from /Users/vo/Dropbox/code/encore/bundles/standard/String.enc
=== Initializing ===
=== Parsing ===
=== Selected mode : svm_learn ===
=== Loading library ===
=== Setting up arguments ===
=== Training ===
        fin  = data/ex-svms/example1/train.dat
        fout = data/ex-svms/example1/model
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..900..1000..1100.
.1200..1300..1400..1500..1600..1700..1800..1900..2000..OK. (2000 examples read)
Setting default regularization parameter C=1.0000
Optimizing.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................d
one. (425 iterations)
Optimization finished (5 misclassified, maxdiff=0.00085).
Runtime in cpu-seconds: 0.17
Number of SV: 878 (including 117 at upper bound)
L1 loss: loss=35.67674
Norm of weight vector: |w|=19.55576
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=383.42791
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=5.85% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>95.40% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>93.07% (rho=1.00,depth=0)
Number of kernel evaluations: 45954
Writing model file...done
=== Done ===
```

(a) Encore training result

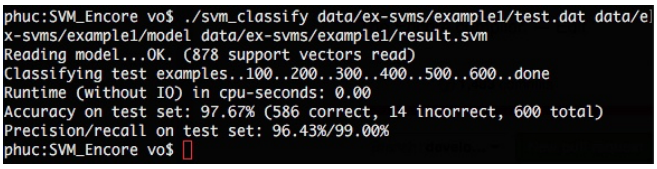
```
phuc:SVM_Encore vo$ ./svm_learn data/ex-svms/example1/train.dat data/ex-
svms/example1/model
Scanning examples...done
Reading examples into memory...100..200..300..400..500..600..700..800..9
00..1000..1100..1200..1300..1400..1500..1600..1700..1800..1900..2000..OK
. (2000 examples read)
Setting default regularization parameter C=1.0000
Optimizing.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
.....................................................................
..done. (425 iterations)
Optimization finished (5 misclassified, maxdiff=0.00085).
Runtime in cpu-seconds: 0.08
Number of SV: 878 (including 117 at upper bound)
L1 loss: loss=35.67674
Norm of weight vector: |w|=19.55576
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=383.42791
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=5.85% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>95.40% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>93.07% (rho=1.00,depth=0)
Number of kernel evaluations: 45954
Writing model file...done
```

(b) SVMLight training result

Figure 1: Training and classifying results of Encore and *SVMLight*



(a) Encore classifying result



(b) SVMLight classifying result

Figure 2: Example 1: Training and classifying results of Encore and *SVMLight*

Secondly, the Higgs Boson training and classifying datasets will be used for training and classifying. Due to the limitation of *SVMLight*, it takes approximately 12 hours for training the Higgs Bosson datasets with each implementations. The classifying results with the model and testing dataset can be found at Higgs Boson large dataset. Figure 3 and 4 show the testing results of *Encore-SVM* and *SVMLight*.

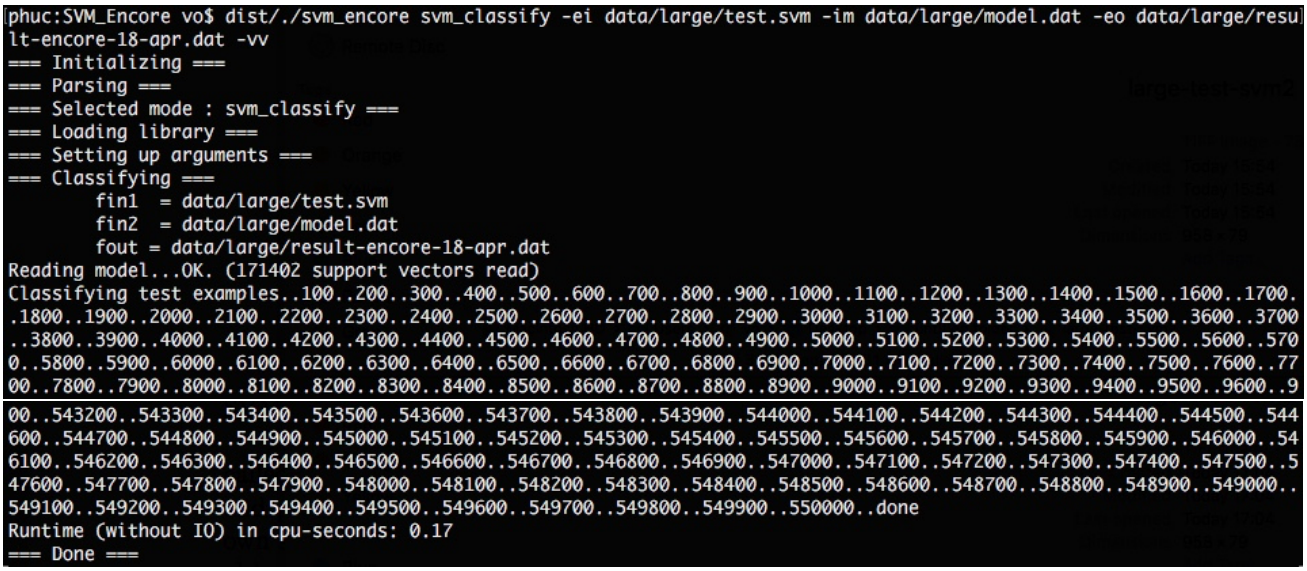


Figure 3: Higgs Boson: Classifying results of Encore

Thirdly, the performance between two versions is not much different when training and classifying with the small datasets as shown in Example 1 and 2. However, the performance of *Encore-SVM* in classifying phase is better than original version when testing *55,000* events in the large Higgs Boson dataset, and *170,000* support vectors in the model. The execution time of *Encore-SVM* is `0.17 seconds`, of *SVMLight* is `0.22 seconds`.

7 Conclusion

Encore-SVM [4] is a parallel implementation of *SVMLight* that is implemented using object-oriented parallel programming language Encore. It takes advantage of Encore language such as active objects that can send and receive messages asynchronously. The parallelism is partially obtained in the converting phase, and classifying phase. The *Encore-SVM* implementation is centered around the Higgs Boson datasets that contain large training and testing data. The accuracy and performance are firstly

```
phuc:SVM_Encore vo$ dist/./svm_classify data/large/test.svm data/large/model.dat data/large/result-svml-18-apr.dat
Reading model...OK. (171402 support vectors read)
Classifying test examples..100..200..300..400..500..600..700..800..900..1000..1100..1200..1300..1400..1500..1600..1700.
.1800..1900..2000..2100..2200..2300..2400..2500..2600..2700..2800..2900..3000..3100..3200..3300..3400..3500..3600..3700
..3800..3900..4000..4100..4200..4300..4400..4500..4600..4700..4800..4900..5000..5100..5200..5300..5400..5500..5600..570
0..5800..5900..6000..6100..6200..6300..6400..6500..6600..6700..6800..6900..7000..7100..7200..7300..7400..7500..7600..77
00..7800..7900..8000..8100..8200..8300..8400..8500..8600..8700..8800..8900..9000..9100..9200..9300..9400..9500..9600..9
0..541700..541800..541900..542000..542100..542200..542300..542400..542500..542600..542700..542800..542900..543000..5431
00..543200..543300..543400..543500..543600..543700..543800..543900..544000..544100..544200..544300..544400..544500..544
600..544700..544800..544900..545000..545100..545200..545300..545400..545500..545600..545700..545800..545900..546000..54
6100..546200..546300..546400..546500..546600..546700..546800..546900..547000..547100..547200..547300..547400..547500..5
47600..547700..547800..547900..548000..548100..548200..548300..548400..548500..548600..548700..548800..548900..549000..
549100..549200..549300..549400..549500..549600..549700..549800..549900..550000..done
Runtime (without IO) in cpu-seconds: 0.22
```

Figure 4: Higgs Boson: Classifying results of *SVMLight*

checked by using enclosing examples of *SVMLight*, and then comparing the results between original and parallel versions. The large Higgs Boson training and testing datasets expose the limitation of *SVMLight* and *Encore-SVM*, it took more than 8 hours to train the model with the given training dataset. The implementation of *Encore-SVM* also requires a lot of time since additional features of Encore language are unavailable at the moment and need to be developed before used. In summary, the project shows the possibility, performance and accuracy of applying Encore programming language to machine learning in general, and *SVMLight* in particular. The Encore will be considered to develop more necessary features and case studies. We will also investigate how the Encore language can be extended in the future to further support the developer in writing correct parallel programs.

References

[1] *Atlas Higgs Challenge 2014*. http://opendata.cern.ch/collection/ATLAS-Higgs-Challenge-2014.

[2] Stephan Brandauer et al. "Formal Methods for Multicore Programming: 15th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2015, Bertinoro, Italy, June 15-19, 2015, Advanced Lectures". In: ed. by Marco Bernardo and Broch Einar Johnsen. Cham: Springer International Publishing, 2015. Chap. Parallel Objects for Multicores: A Glimpse at the Parallel Language Encore, pp. 1–56.

[3] *CERN*. https://root.cern.ch.

[4] *Encore-SVM*. https://github.com/PhucVH888/uu-projects/tree/master/SVM_Encore.

[5] *Higgs Datasets 2014*. https://archive.ics.uci.edu/ml/datasets.

[6] *Higgs Training database*. https://archive.ics.uci.edu/ml/machine-learning-databases/00280/.

[7] *Statistical packages*. https://en.wikipedia.org/wiki/List_of_statistical_packages.

[8] *SVMLight*. http://svmlight.joachims.org.

[9] *SVMs tools*. http://www.svms.org/software.html.

8 Appendices

This appendices include all the user manual and test results. The source code of the *Encore-SVM* can be found at SVM_Encore repository on Github.

8.1 User manual

From the SVM_Encore folder, use `make` to build the program. Here is the options for makefile.

```
$ make
  + all          // by default
  + tidy         // clean the object files and other stuff.
  + clean        // indlude tidy and clean all the executable files.
  + help | info  // show help.
  + svm_lib      // build the api and shared library.
  + svm_encore   // build svm_encore.
  + convert      // test convert function with 3 different options.
  + demo_encore  // convert, train, and classify by Encore-SVM
  + demo_svm     // convert, train, and classify by SVMLight
  + classify_large  // classify with large Higgs Boson dataset by
                    // both Encore-SVM and SVMLight
```

```
make for SVM-ENCORE              Huu-Phuc Vo, 2016

Thanks to SVMLight's authors for the open source library.

USAGE: make [clean | help | info | all | tidy

             svm_light_api | svm_encore |

             svm_lib | distribute]

    svm_light_api       builds the learning, classfication modules, and libraries

    svm_encore          builds the svm_encore module

    svm_lib             builds shared object library that can be linked into

    distribute          distributes shared object library,

                        and executable files to suitable folders

    help                prints help for using makefile

    all (default)       builds svm_light_api, svm_encore, and svm_lib,

                        then distributes libraries

    clean               removes .o and target files
```

Figure 5: Makefile options

8.2 Test results

Several test results are shown in the following figures.

```
phuc:SVM_Encore vo$ dist/./svm_encore -eh
Please read the instruction:

-------------------------------------------------------------------------------
Usage:
dist/./svm_encore  [ -lib: path's lib | -ea : all help with [options] | -eh : help | -ed : convert-only ]

        | convert [-chunk n] -ei csv_file    -eo svml_file

            -chunk 1    convert from csv to svm format only

            -chunk -1   join from n file into one

            -chunk n    (n>1) split file into n files

        | svm_learn    [options] -ei in_file     -eo model_file

        | svm_classify [options] -ei in_file     -eo out_file    -im model_file

-------------------------------------------------------------------------------
```

Figure 6: Encore-SVM help

```
phuc:SVM_Encore vo$ encorec -c svm_encore.enc; ./svm_encore svm_learn -ei data/ex-svms/example2/train_induction.dat -eo
 data/ex-svms/example2/model_induc -vv
Importing module Converter from ./Converter.enc
Importing module Core from ./Core.enc
Importing module Params from ./Params.enc
Importing module String from /Users/vo/Dropbox/code/encore/bundles/standard/String.enc
=== Initializing ===
=== Parsing ===
=== Selected mode : svm_learn ===
=== Loading library ===
=== Setting up arguments ===
=== Training ===
        fin  = data/ex-svms/example2/train_induction.dat
        fout = data/ex-svms/example2/model_induc
Scanning examples...done
Reading examples into memory...OK. (10 examples read)
Setting default regularization parameter C=1.0000
Optimizing.done. (2 iterations)
Optimization finished (0 misclassified, maxdiff=0.00000).
Runtime in cpu-seconds: 0.00
Number of SV: 10 (including 2 at upper bound)
L1 loss: loss=0.03469
Norm of weight vector: |w|=3.13458
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=10.82562
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=20.00% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>80.00% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>80.00% (rho=1.00,depth=0)
Number of kernel evaluations: 185
Writing model file...done
=== Done ===
```

(a) Inductive Encore training result

```
phuc:SVM_Encore vo$ ./svm_learn data/ex-svms/example2/train_induction.dat data/ex-svms/example2/model_svm_induc
Scanning examples...done
Reading examples into memory...OK. (10 examples read)
Setting default regularization parameter C=1.0000
Optimizing.done. (2 iterations)
Optimization finished (0 misclassified, maxdiff=0.00000).
Runtime in cpu-seconds: 0.00
Number of SV: 10 (including 2 at upper bound)
L1 loss: loss=0.03469
Norm of weight vector: |w|=3.13458
Norm of longest example vector: |x|=1.00000
Estimated VCdim of classifier: VCdim<=10.82562
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=20.00% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>80.00% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>80.00% (rho=1.00,depth=0)
Number of kernel evaluations: 185
Writing model file...done
```

(b) Inductive SVMLight training result

Figure 7: Example: Inductive training results of Encore and *SVMLight*

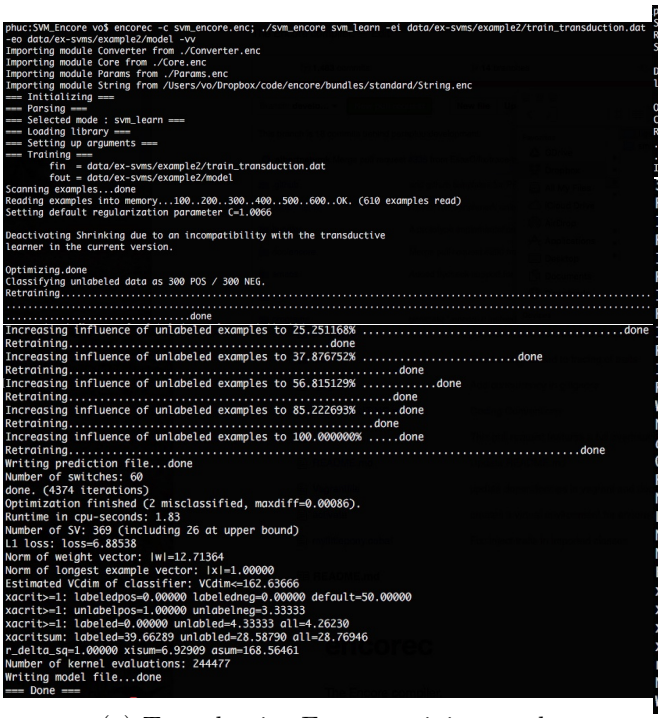
```
phuc:SVM_Encore vo$ encorec -c svm_encore.enc; ./svm_encore svm_classify -ei data/ex-svms/example2/test.dat -im data/ex
-svms/example2/model_induc -eo data/ex-svms/example2/result_encore_ind.svm -vv
Importing module Converter from ./Converter.enc
Importing module Core from ./Core.enc
Importing module Params from ./Params.enc
Importing module String from /Users/vo/Dropbox/code/encore/bundles/standard/String.enc
=== Initializing ===
=== Parsing ===
=== Selected mode : svm_classify ===
=== Loading library ===
=== Setting up arguments ===
=== Classifying ===
        fin1  = data/ex-svms/example2/test.dat
        fin2  = data/ex-svms/example2/model_induc
        fout = data/ex-svms/example2/result_encore_ind.svm
Reading model...OK. (10 support vectors read)
Classifying test examples..100..200..300..400..500..600..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 84.33% (506 correct, 94 incorrect, 600 total)
Precision/recall on test set: 89.62%/77.67%
=== Done ===
```

(a) Inductive Encore classifying result

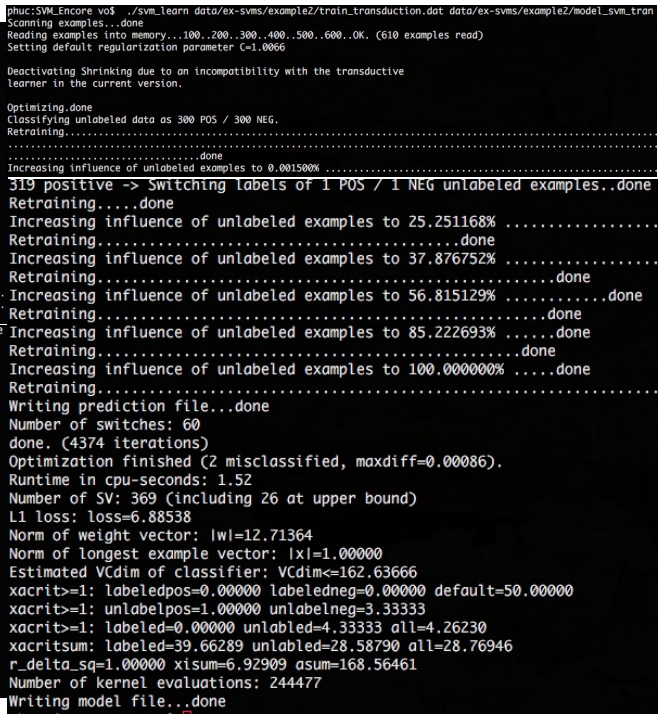
```
phuc:SVM_Encore vo$ ./svm_classify data/ex-svms/example2/test.dat data/ex-svms/example2/model_svm_induc data/ex-svms/ex
ample2/result_svm_ind.svm
Reading model...OK. (10 support vectors read)
Classifying test examples..100..200..300..400..500..600..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 84.33% (506 correct, 94 incorrect, 600 total)
Precision/recall on test set: 89.62%/77.67%
```

(b) Inductive *SVMLight* classifying result

Figure 8: Example: Inductive training and classifying results of Encore and *SVMLight*

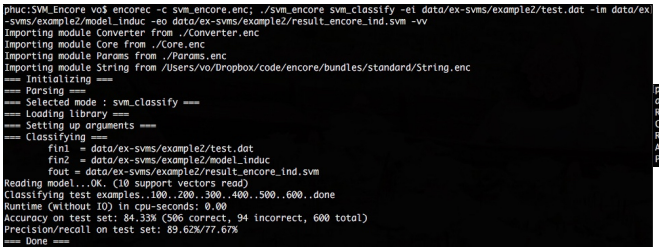


(a) Transductive Encore training result

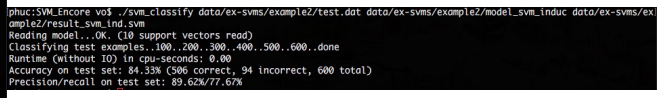


(b) Transductive *SVMLight* training result

Figure 9: Example: Transductive training results of Encore and *SVMLight*

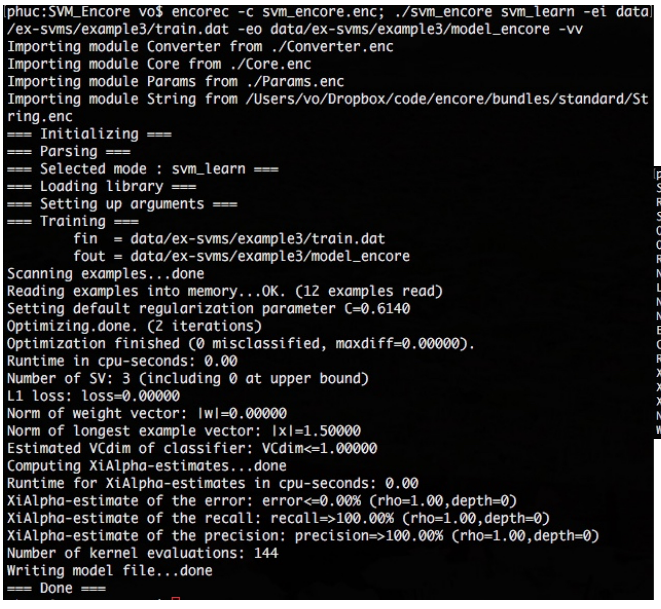


(a) Transductive Encore classifying result

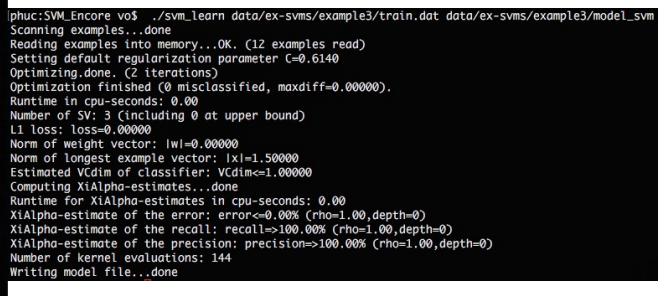


(b) Transductive *SVMLight* classifying result

Figure 10: Example: Transductive Training and classifying results of Encore and *SVMLight*

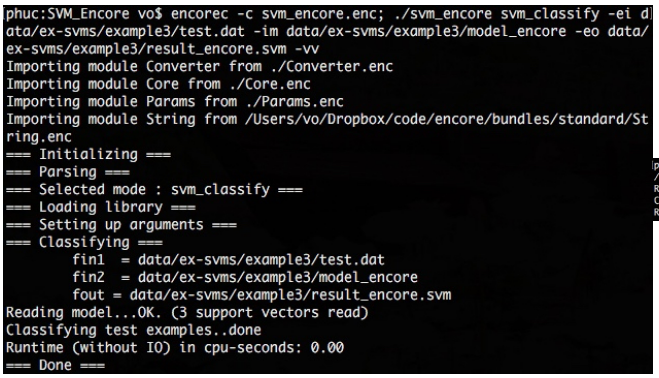


(a) Encore training result

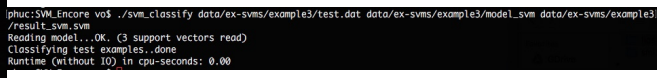


(b) SVMLight training result

Figure 11: Training and classifying results of Encore and *SVMLight*



(a) Encore classifying result



(b) SVMLight classifying result

Figure 12: Example: Training and classifying results of Encore and *SVMLight*