Read CaboCha

#2 Dependency Training With SVM

Yuta Hayashibe

2012 March 23th 15:00~(JST)

Where are the main functions?

$ find . -maxdepth 1 |xargs grep "int main" -l

- ./cabocha-learn.cpp
  - cabocha_learn(argc, argv);
- ./cabocha-model-index.cpp
  - cabocha_model_index(argc, argv);
- ./normalizer.cpp
  - CaboCha::Normalizer::compile("normalizer.rule", "normalizer_rule.h");
- ~~./svm.cpp~~
- ./cabocha-system-eval.cpp
  - cabocha_system_eval(argc, argv);
- ./cabocha.cpp
  - cabocha_do (argc, argv);

CABOCHA_LEARN

learner.cpp

```
int cabocha_learn(int argc, char **argv) {
    static const CaboCha::Option long_options[] ={(…)}
    CaboCha::Param param;
    param.open(argc, argv, long_options);
    …
    const std::string stype = param.get<std::string>("parser-type");
    const ParserType type = parser_type(stype.c_str());
    CHECK_DIE(type != -1) <<  "unknown parser type: " << stype;
    if (type == TRAIN_DEP) {
        CHECK_DIE(CaboCha::DependencyTrainingWithSVM(…))
        CHECK_DIE(CaboCha::SVM::compile(…))
    }
    else if (type == TRAIN_CHUNK || type == TRAIN_NE) {
        CHECK_DIE(CaboCha::ChunkingTrainingWithCRFPP(…))
    }
    return 0;
}
```

Get parameters from *argv*

Switch by *parser-type*

CaboCha::Option (Struct)

- Manage options of programs like *boost::program_options*
- CaboCha::Option Struct

const char *	name
char	short_name
const char *	default_value
const char *	arg_description
const char *	description

- static const CaboCha::Option long_options[] = {

{"parser-type", 'e', "dep", "STRING“, "choose from ne/chunk/dep (default dep)" },

…

{"cost", 'c', "0.001", "FLOAT",

"set FLOAT for cost parameter (default 0.001)" },

… }

CHECK_DIE in common.h

```
#define WHAT what_.stream_

#define CHECK_FALSE(condition) ¥
 if (condition) {} else return ¥
   wlog(&what_) & what_.stream_ <<           ¥
     __FILE__ << "(" << __LINE__ << ") [" << #condition << "] "

#define CHECK_TREE_FALSE(condition) ¥
 if (condition) {} else return ¥
  wlog(tree->allocator()->mutable_what()) & ¥
    tree->allocator()->mutable_what()->stream_ << ¥
     __FILE__ << "(" << __LINE__ << ") [" << #condition << "] "

#define CHECK_DIE(condition) ¥
(condition) ? 0 : die() & std::cerr << __FILE__ << ¥
"(" << __LINE__ << ") [" << #condition << "] "

#endif
```

__LINE__ and __FILE__ are macros defined in the preprocessor

Switch by *condition*

If negative, print the filename and the line number of the source code.

DEPENDENCY TRAINING WITH SVM

Input format

It is the same of outpus of “cabocha -f1 -O4 -n0”

For instance “東京に明日行く”

```
* 0 2D
東京　名詞,固有名詞,地域,一般,*,*,東京,トウキョウ,トーキョー,,
に　助詞,格助詞,一般,*,*,*,に,ニ,ニ,,
* 1 2D
明日　名詞,副詞可能,*,*,*,*,明日,アシタ,アシタ,,
* 2 -1D
行く　動詞,自立,*,*,五段・カ行促音便,基本形,行く,イク,イク,いく/行く,
EOS
```

DependencyTrainingWithSVM(…) at dep_learner.cpp

- const char *train_file
- const char *model_file
  - str_train_file : ".str"
  - id_train_file : ".id"
  - svm_learn_model_file : ".model"
- CharsetType charset,
- PossetType posset,
- int degree, (1 ≤ degree ≤ 3)
- double cost, (0.0 < cost)
- double cache_size (40.0 ≤ cache_size)

Read training data

```
std::ofstream ofs(WPATH(str_train_file.c_str()));
tree.allocator()->set_stream(&ofs);
scoped_ptr<Analyzer> analyzer(new DependencyParser);
scoped_ptr<Analyzer> selector(new Selector);
…
size_t line = 0;
std::string str;
while (ifs) {
        CHECK_DIE(read_sentence(&ifs, &str, INPUT_CHUNK));
        CHECK_DIE(tree.read(str.c_str(), str.size(),
                        INPUT_CHUNK)) << "cannot parse sentence";
        CHECK_DIE(selector->parse(&tree)) << selector->what();
        CHECK_DIE(analyzer->parse(&tree)) << analyzer->what();
        if (tree.empty()) continue;
        if (++line % 100 == 0)
          std::cout << line << ".. " << std::flush;
}
std::cout << "¥nDone! ";
```

Output “model_file.str” when allocation

Examples in "model_file.str"

-1 A:の DIST:6- F_F0:を F_F1:助詞 F_F2:格助詞 F_F3:一般 F_H0:キロ F_H1:名詞 F_H2:接尾 F_H3:助数詞 G_CASE:から G_CASE:の G_PUNC:、a:形 容詞・アウオ段 f_F0:将来 f_F1:名詞 f_F2:副詞可能 f_H0:将来 f_H1:名詞 f_H2:副詞可能 f_PUNC:、

+1 DIST:1 F_F0:する F_F1:動詞 F_F2:自立 F_F5:基本形 F_F6:サ変・スル F_H0:する F_H1:動詞 F_H2:自立 F_H5:基本形 F_H6:サ変・スル a:の f_F0:を f_F1:助詞 f_F2:格助詞 f_F3:一般 f_H0:キロ f_H1:名詞 f_H2:接尾 f_H3:助数詞

+1 A:を DIST:6- F_F0:する F_F1:動詞 F_F2:自立 F_F5:基本形 F_F6:サ変・スル F_H0:する F_H1:動詞 F_H2:自立 F_H5:基本形 F_H6:サ変・スル G_CASE:から G_CASE:の G_CASE:を G_PUNC:、a:形容詞・アウオ段 f_F0:将来 f_F1:名詞 f_F2:副詞可能 f_H0:将来 f_H1:名詞 f_H2:副詞可能 f_PUNC:、

Make *dic* from "model_file.str"

```
std::map<std::string, int> dic;
...
std::ifstream ifs(WPATH(str_train_file.c_str()));
CHECK_DIE(ifs) << "no such file or directory: " << str_train_file;
scoped_fixed_array<char, BUF_SIZE * 32> buf;
scoped_fixed_array<char *, BUF_SIZE> column;
while (ifs.getline(buf.get(), buf.size())) {
    const size_t size = tokenize(buf.get(), " ",
                                 column.get(), buf.size());
    CHECK_DIE(size >= 2);
    for (size_t i = 1; i < size; ++i) {
      dic[std::string(column[i])]++;
    }
}
```

Make *id_table* from *dic*

```
std::map<std::string, int> dic;
...
std::vector<std::pair<int, std::string> > freq;
for (std::map<std::string, int>::const_iterator it = dic.begin();
    it != dic.end(); ++it) {
  freq.push_back(std::make_pair(it->second, it->first));
}

std::sort(freq.begin(), freq.end());
int id = 0;
for (size_t i = 0; i < freq.size(); ++i) {
  dic[freq[i].second] = id;
  ++id;
}
```

copy

Sort by frequencies of features

give ids to features

Make a training file of SVM

```
std::vector<int *> x;
std::vector<double> y;
FreeList<int> feature_freelist(8192);
…
std::set<std::string> dup;
while (ifs.getline(buf.get(), buf.size())) {
    if (dup.find(buf.get()) != dup.end())
      continue;
    …
}
```

abbr.

Learn a SVM model

```
scoped_ptr<SVMModel> model;
model.reset(SVMSolver::learn(y.size(),
                             &y[0],
                             &x[0],
                             cost,
                             degree,
                             cache_size));
CHECK_DIE(model.get());
```

double *

Int **

Learning here

```
class SVMModel {
      public:
       double bias() const { return bias_; }
       void set_bias(double bias) { bias_  = bias; }
       size_t degree() const { return degree_; }
       void set_degree(size_t degree) { degree_ = degree; }
       size_t size() const { return alpha_.size(); }
       double alpha(size_t i) const { return alpha_[i]; }
       const int *x(size_t i) const { return x_[i]; }

       void add(double alpha, const int *x) {
         alpha_.push_back(alpha);
         x_.push_back(x);
       }

       double classify(const int *x) const;

       SVMModel(): bias_(0.0), degree_(0) {}

      private:
       double bias_;
       size_t degree_;
       std::vector<double> alpha_;
       std::vector<const int*> x_;
     };
```

model.txt of the SVM Model

0 A:あえて
1 A:あまり
2 A:あまりに
3491 A:あらゆる
3 A:ある
…
10013 f_PUNC:、
9804 f_PUNC:。

Feature id numbers

degree and bias

2
0.010408
-0.000547102 5965 6395 6527 8416 9787 9829 987
-0.00900905 4254 4836 6684 9886 9919 10011 100
-8.45161e-05 105 541 5344 9836 9855 9930 9979
…
-1 8475 9375 9960 9961 10011 10012 10014 10015 10016 10017 10021

alpha(i) and **new** ids

Collect numbers of "weighted" features

```
std::set<int> active_feature;
for (size_t i = 0; i < model->size(); ++i) {
  for (const int *fp = model->x(i); *fp >= 0; ++fp) {
    active_feature.insert(*fp);
  }
}

std::map<int, int> old2new;
int id = 0;
for (std::map<std::string, int>::const_iterator it = dic.begin();
    it != dic.end(); ++it) {
 if (active_feature.find(it->second) != active_feature.end()) {
   if (old2new.find(it->second) == old2new.end()) {
     old2new[it->second] = id;
     ++id;
   }
 }
}
```

Give them new IDs

Write a model file

```
std::ofstream ofs(WPATH(model_file));
for (std::map<std::string, int>::const_iterator it = dic.begin();
    it != dic.end(); ++it) {
  if (active_feature.find(it->second) != active_feature.end()) {
    ofs << old2new[it->second] << ' ' << it->first << std::endl;
  }
}
ofs << std::endl;
ofs << model->degree() << std::endl;
ofs << model->bias() << std::endl;
for (size_t i = 0; i < model->size(); ++i) {
  ofs << model->alpha(i);
  std::vector<int> new_x;
  for (const int *fp = model->x(i); *fp >= 0; ++fp) {
    new_x.push_back(old2new[*fp]);
  }
  std::sort(new_x.begin(), new_x.end());
  for (std::vector<int>::const_iterator it = new_x.begin();
      it != new_x.end(); ++it) {
    ofs << ' ' << *it;
  }
  ofs << std::endl;
}
```

Remove temporal files

```
Unlink(str_train_file.c_str());
Unlink(id_train_file.c_str());
Unlink(svm_learn_model_file.c_str());
```