LEMGA: Learning Models and Generic Algorithms

Introduction with Examples

(Updated to v0.1β)

Ling Li

Learning Systems Group, Caltech

January 22, 2003

What is Lemga?

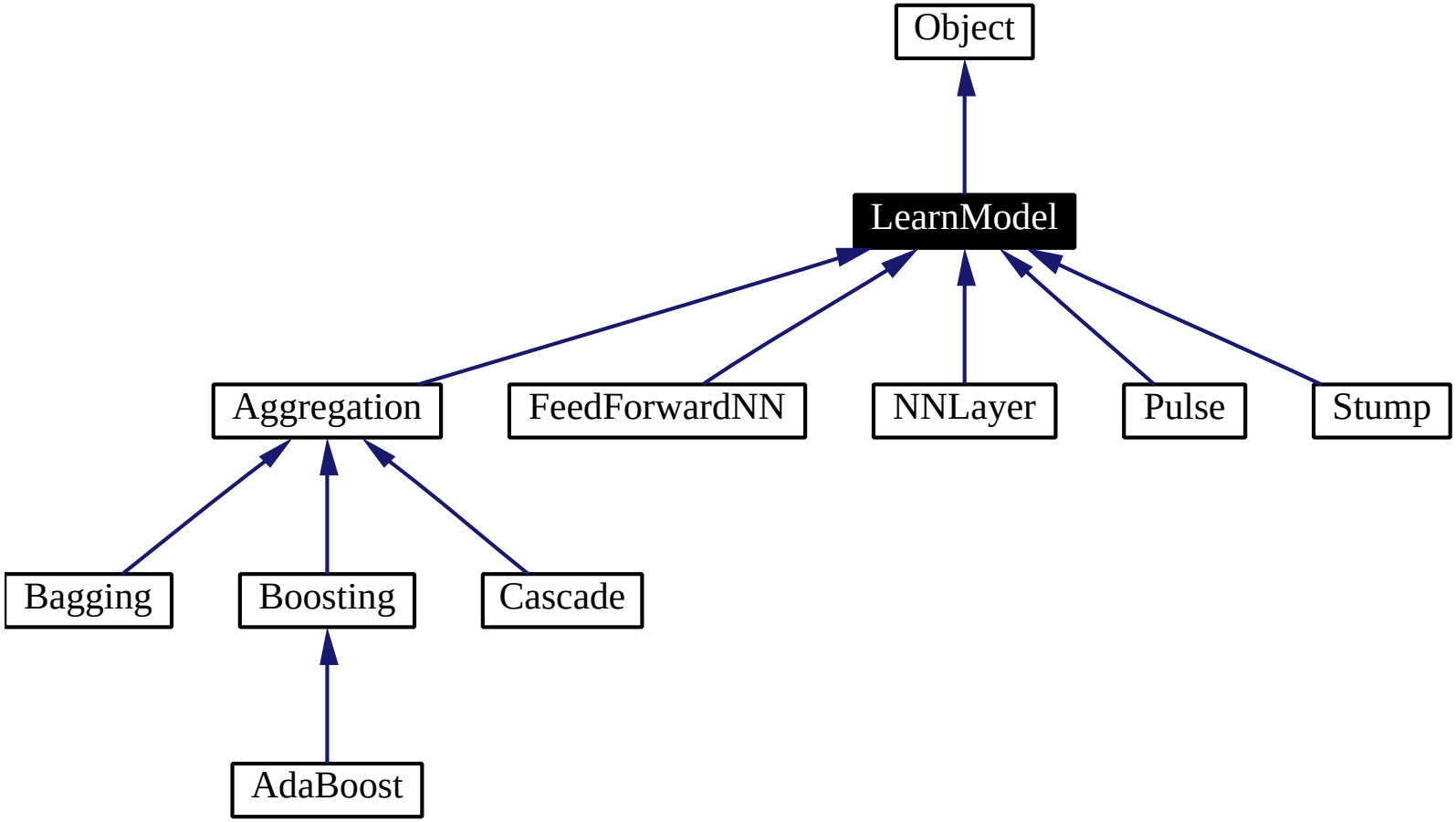
Lemga is a C++ library consisting of

- Learning models/frames: neural networks, decision stumps, bagging, boosting, ...
- Generic optimization algorithms: gradient descent, conjugate-gradient, ...
- Auxiliary parts: data set, ...

Features

- Uniform interface for learning models: `initialize()`, `set_train_data()`, `train()`, `>>`, `<<`, `create()`, ...
- Flexible training methods: different optimization methods can be applied to a same object
- Easily extendable
- Fast (C++ is fast)

Learning Models



Neural Network

- Construct a network of $4 \times 10 \times 1$

```
lemga::NNLayer l1(4, 10), l2(10, 1);
l1.set_weight_range(-0.2, 0.2);
l2.set_weight_range(-0.5, 0.5);
lemga::FeedForwardNN nn;
nn.add_top(l1);
nn.add_top(l2);
```

- Get the training data

```
std::ifstream fd("train.dat");
lemga::pDataSet trd = lemga::load_data(fd, 100, 4, 1);
```

- Train the network

```
nn.initialize();     // have run set_seed(0);
nn.set_train_data(trd);
nn.set_train_method(nn.CONJUGATE_GRADIENT);
nn.set_parameter(0.1, 1e-4, 1000);
nn.train();
```

- Save the network

```
std::ofstream fw("nnet.lm");
fw << nn;
```

- Load the network

```
std::ifstream fr("nnet.lm");
lemga::FeedForwardNN nn2;
fr >> nn2;
```

- Load the test data

```
fd.open("test.dat");
lemga::pDataSet ted = lemga::load_data(fd, 200, 4, 1);
```

- Calculate test error

```
double err = 0;
for (size_t i = 0; i < ted->size(); ++i)
    err += nn2.r_error(nn2(ted->x(i)), ted->y(i));
err /= ted->size();
```

Bagging and AdaBoost

Given a base learner, bagging or AdaBoost will generate a list of hypotheses.

- (cont. from last example) Construct

```
lemga::Bagging bag;
bag.set_base_model(nn);
bag.set_max_models(20);
```

- Set the training data and train

```
bag.initialize();     // have run set_seed(0);
bag.set_train_data(trd);
bag.train();
```

We can also save the bagged hypothesis to a file, load it, and test it.

Using AdaBoost is similar.

We can even do AdaBoost on Bagging, or vice verse.

Generic Optimization Algorithms

The optimization algorithms can be applied to any object, as long as it provides the following interfaces:

- `cost()`
- `weight()`, `set_weight()`
- `gradient()`
- `stop_opt()`

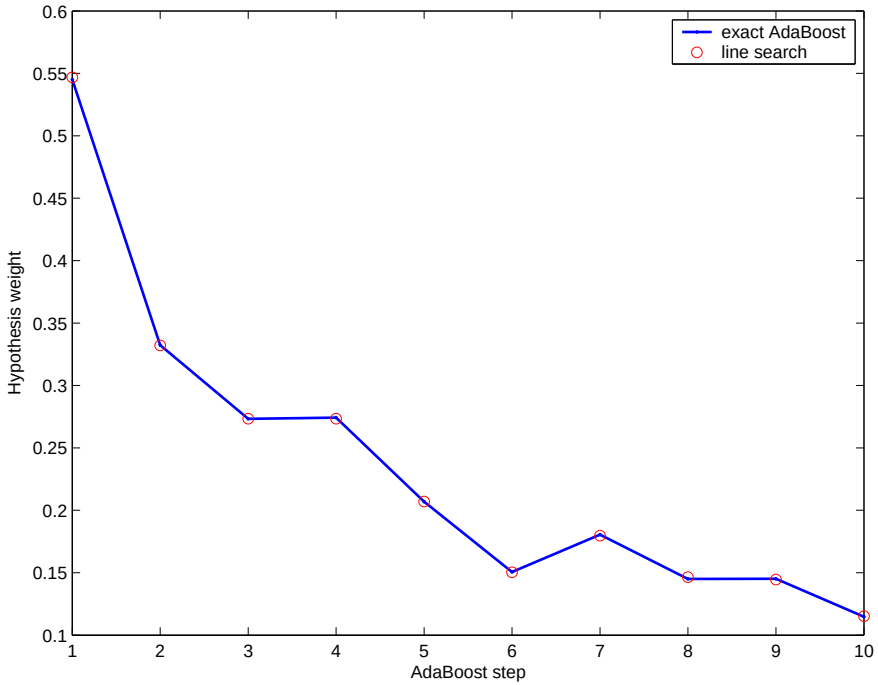
Current methods include gradient descent (and that with adaptive learning rate, momentum), line search, and conjugate gradient.

AdaBoost can be viewed as gradient descent in functional space.

Example: AnyBoost

```
struct _boost_gd {
    Boosting* b;
    _boost_gd (Boosting* pb) : b(pb) {}

    REAL cost () const { return b->cost(); }

    Boosting::BoostWgt weight () const {
        return Boosting::BoostWgt(b->lm, b->lm_wgt);
    }

    void set_weight (const Boosting::BoostWgt& bw) {
        b->lm = bw.models(); b->lm_wgt = bw.weights();
        b->n_in_agg = b->lm.size();
        assert(b->lm.size() == b->lm_wgt.size());
    }

    Boosting::BoostWgt gradient () const {
        return Boosting::BoostWgt
            (b->train_with_smpwgt(b->sample_weight()), -1);
    }

    bool stop_opt (size_t step, REAL) const {
        return step >= b->max_n_model;
    }
};
```

```
Use stump, with AdaBoost.M1 [linear]
random seed = 12, pd size = 616, va size = 64
*.......          cost = 0.867893, step = 0.546875
*.......          cost = 0.821924, step = 0.332031
*.......          cost = 0.792337, step = 0.273438
*.......          cost = 0.763387, step = 0.273438
-*......          cost = 0.747384, step = 0.207031
-*.......         cost = 0.738958, step = 0.150391
-*.......         cost = 0.727258, step = 0.179688
-*.......         cost = 0.719554, step = 0.146484
-*.......         cost = 0.712046, step = 0.144531
--*......         cost = 0.707418, step = 0.115234
```



Different Cost Functions

The `Boosting` class provides a general way (AnyBoost) to do boosting. Changing `_cost()` and `_cost_deriv()` will result in a different boosting method.

Example: `AdaBoost` (code simplified)

- `_cost()` returns the cost c for an individual example

```
REAL AdaBoost::_cost (Output F, Output y) {
    return exp(- F[0] * y[0]);
}
```

- `_cost_deriv()` returns $\frac{\partial c(F(x),y)}{\partial F(x)}$ for an individual example

```
REAL AdaBoost::_cost_deriv (Output F, Output y) {
    return - exp(- F[0] * y[0]) * y[0];
}
```

A Practical Project: αBoost

The gradient descent in functional space tends to decrease the cost function. After we get a list of hypotheses, we can still tune the hypothesis weights (α's) to decrease the cost.

(code simplified)

```
struct _alpha_descent {
    AlphaBoost* ab;
    size_t max_runs;

    REAL cost () { return ab->cost(); }

    MODEL_WEIGHTS weight () { return ab->lm_wgt; }
    void set_weight (MODEL_WEIGHTS& wgt) { ab->lm_wgt = wgt; }

    MODEL_WEIGHTS gradient () { /* omitted */ }

    bool stop_opt (size_t step, REAL) { return step >= max_runs; }
};

void AlphaBoost::optimize_alpha (size_t max_runs) {
    _alpha_descent d;
    d.max_runs = max_runs; d.ab = this;
    lemga::iterative_optimize(lemga::_conjugate_gradient
        <_alpha_descent,MODEL_WEIGHTS,REAL,REAL>(&d, 0.01));
}
```

Suggestions

- Read the code before using it.
- Be careful about assumptions. (Some classes assume binary-class problems.)
- Don't hesitate to modify the code.
- Look for other resources. Lemga was basically for my own use. It is not as robust and complete as some other software packages I found from the Internet. Find a more suitable one for your project.

Need Help?

- Online manual: `http://www.work.caltech.edu/ling/lemga/`
- Email or talk to me: `ling@caltech.edu`
- Don't hesitate to send bug reports. We can make Lemga better.