CSE 6740 Lecture 7

How Do I Do ’Industrial-Strength’ Machine Learning?
(FASTlib and MLPACK)

Alexander Gray
agray@cc.gatech.edu

Georgia Institute of Technology
Today

1. How the Project Works
2. Learning FASTlib/MLPACK

My way of making your project *real*. You will develop good machine learning code that other people can use.
How the Project Works
Motivations for Project Structure

- **Learning by doing.** Implementing the math as code is the only way to really understand things.

- **Creating code for the field.** There are existing collections of machine learning code, but there is a need for a comprehensive collection of efficient code.
Parts

Part 1.

- Choose a partner.
- Choose the two ML methods you’ll implement. These must be approved by the instructor/TA BEFORE submitting.
- Submit two+ pages of description for each method, including the math and relevant pseudocode.
- Say what functions you’ll write and who will write them.
- Say how you’ll test the methods using reference datasets.
Part 2.

- Turn in the code.
- Describe the amount of unit testing.
- Describe the amount of adherence to the style guidelines.
- Describe the amount of documentation.
- Show correctness/performance on the reference datasets.
Part 3.

- Turn in a version of the code updated according to the TA’s review.

- Also turn in experimental results showing either an interesting/novel application of the method(s) to some data, OR a thorough comparison of the methods to other methods in MLPACK for the same task.
Comparisons

As part of the code testing requirement, the methods will be compared statistically and computationally on a selection of provided datasets:

**MSE:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Synth Data 1</th>
<th>Astro Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear regression</td>
<td>6.3</td>
<td>8.4</td>
</tr>
<tr>
<td>Nadaraya-Watson regression</td>
<td>1.1</td>
<td>2.3</td>
</tr>
</tbody>
</table>

**CPU seconds:**

<table>
<thead>
<tr>
<th>Method</th>
<th>Synth Data 1</th>
<th>Astro Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear regression</td>
<td>33</td>
<td>57</td>
</tr>
<tr>
<td>Nadaraya-Watson regression</td>
<td>109</td>
<td>483</td>
</tr>
</tbody>
</table>
Do a Good Job

- The difficulty (or non-difficulty) of your project will be taken into account in your project grade.
- If you do a good job, this is an item for your c.v. You can say your work is part of the MLPACK distribution.
- The group with the best project will get the 2009 Best Project award and will claim the title of “King/Queen of Machine Learning” for 2009.
Learning FASTlib/MLPACK
FASTlib and MLPACK

MLPACK is what the collection will be called. Its first public release will be at the end of the semester. Its codes are built using the FASTlib C++ library, which features:

- Support for standardized argument passing and management of experiment output
- Linear algebra methods (through LAPACK and Trilinos)
- Optimization and other numerical methods (through Numerical Recipes)
- Data structures for fast discrete algorithms
- Use of templates for elegance/comprehendibility
- Memory and CPU efficiency
- Standards for code reviews, unit testing, and consistent style
Learning FASTlib

- Tutorial code example
- About 15-20 ML methods to look at
- In the manual: FAQ and “cookbook”
- If all else fails: email the TA
How to Install FASTlib

- Install CMake (available online)
- Download FASTlib
- run:
  - cd fastlib_dir/bin
  - cmake ..
  - make
  - make install
How to Compile with FASTlib

- Call `g++` with additional flags:
  - `-L/fastlib_dir/bin -lfastlib`
  - `-I/fastlib_dir`

- In your program: `#include <fastlib/fastlib.h>`
Reading from File

(Assuming `using fastlib;`)

After setting `data_file`:

```cpp
Matrix data_mat;
data::Load(data_file, &data_mat);
```

Later on, we can use:

```cpp
data::Save(out_file, results_mat);
```

Caveat: Matrices transpose on load/save; column major!

Alternately: Can read `.arff` with class `Dataset`
Ex: Distance matrix

Initialize:

```cpp
int n = data_mat.n_cols();

Matrix dist_mat;
dist_mat.Init(n, n);
```
Ex: Distance matrix

for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        dist_mat.set(i, j, sqrt(dist_sq));
    }
}

Ex: Distance matrix

for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        dist_mat.set(i, j, sqrt(dist_sq));
    }
}

Note matrix/vector access.
Ex: Distance matrix

```c++
for (int j = 0; j < n; ++j) {
    Vector j_vec;
    data_mat.MakeColumnVector(j, &j_vec);

    for (int i = 0; i < n; ++i) {
        Vector i_vec;
        data_mat.MakeColumnVector(i, &i_vec);

        double dist_sq =
            la::DistSqEuclidean(i_vec, j_vec);
        dist_mat.set(i, j, sqrt(dist_sq));
    }
}
```

Note application of BLAS.
GaussianKernel gaussian;
    gaussian.Init(fx_param_double(NULL, "h", 1.0));

    for (int j = 0; j < n; ++j) {
        Vector j_vec;
        data_mat.MakeColumnVector(j, &j_vec);

        for (int i = 0; i < n; ++i) {
            Vector i_vec;
            data_mat.MakeColumnVector(i, &i_vec);

            double dist_sq =
                la::DistSqEuclidean(i_vec, j_vec);
            kernel_mat.set(i, j, gaussian.EvalUnnormOnSq(dist_sq));
        }
    }

    int dim = data_mat.n_rows();
    la::Scale(gaussian.CalcNormConstant(dim), kernel_mat);
Matrix averaging_mat;
averaging_mat.Init(n, n);
averaging_mat.SetAll(1.0 / n);

Matrix avg_by_kernel, avg_kernel_avg;
Matrix centered_mat;

la::MulInit(averaging_mat, kernel_mat, &avg_by_kernel);
la::SubInit(avg_by_kernel, kernel_mat, &centered_mat);
la::TransposeSquare(&avg_by_kernel);
la::SubFrom(avg_by_kernel, &centered_mat);
la::MulInit(averaging_mat, avg_by_kernel, &avg_kernel_avg);
la::AddTo(avg_kernel_avg, &centered_mat);

Vector eigenvalues;
Matrix eigenvectors;
la::EigenvectorsInit(centered_mat, &eigenvalues, &eigenvectors);
Ex: Kernel PCA

Matrix averaging_mat;
averaging_mat.Init(n, n);
averaging_mat.SetAll(1.0 / n);

Matrix avg_by_kernel, avg_kernel_avg;
Matrix centered_mat;

la::MulInit(averaging_mat, kernel_mat, &avg_by_kernel);
la::SubInit(avg_by_kernel, kernel_mat, &centered_mat);
la::TransposeSquare(&avg_by_kernel);
la::SubFrom(avg_by_kernel, &centered_mat);
la::MulInit(averaging_mat, avg_by_kernel, &avg_kernel_avg);
la::AddTo(avg_kernel_avg, &centered_mat);

Vector eigenvalues;
Matrix eigenvectors;
la::EigenvectorsInit(centered_mat, &eigenvalues, &eigenvectors);