Lib15x: A Template Approach towards Building Generic Machine Learning Library

Outline

Yingrui (Ray) Chang
California Institute of Technology

About Lib15x

Yingrui (Ray) Chang
California Institute of Technology

- implements most supervised learning algorithms covered in Caltech machine learning classes CS155/156.
- written in C++, use template for better performance and easing implementation.
- provides **preprocessing tools** for data scaling, feature extraction.
- interfaces to higher level **aggregation method** such as bagging, random forest, boosting, etc.
- use external matrix library "Eigen" for data storage and basic linear algebra operations.
- threaded using openmp on higher level aggregation methods. (currently in development)

Eigen Library Fundamentals

MatrixXd, VectorXd

Yingrui (Ray) Chang
California Institute of Technology

declaring Matrix/Vector:

```
Eigen::MatrixXd mat{numberOfRows, numberOfCols}; mat.fill(value);
Eigen::VectorXd vec{numberOfElements}; vec.fill(value);
```

supported Matrix/Vector operations:

```
Eigen::MatrixXd mat = mat_0 + mat_1; //same for -, *
Eigen::VectorXd vec = vec_0 + vec_1; //same for -
Eigen::VectorXd vec = mat_0 * vec_0;
double product = vec_0.dot(vec_1);
...
```

accessing elements:

```
mat(rowId, colId) = value; vec(id) = value;
Eigen::VectorXd vec = mat.row(rowId);
Eigen::VectorXd vec = mat.col(colId);
mat.row(rowId) = vec;
mat.col(colId) = vec;
...
```

linear algebra operations:

transpose, inverse, determinant, LU factorization, eigen value decomposition, QR decomposition, sparse LA ...

Eigen Library Fundamentals

Eigen::MatrixXd Memory Layout

Yingrui (Ray) Chang
California Institute of Technology

Eigen::MatrixXd are stored in contiguous memory:
(different from std::vector<std::vector<double> >).

row-major matrix:

```
Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor> A{3,4};
```

A_{00}	A_{01}	A_{02}	A_{03}	A_{10}	A_{11}	A_{12}	A_{13}	A_{20}	A_{21}	A_{22}	A_{23}

column-major matrix:

```
Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic> A{3,4};
```

A_{00}	A_{10}	A_{20}	A_{01}	A_{11}	A_{21}	A_{02}	A_{12}	A_{22}	A_{03}	A_{13}	A_{23}

by default Eigen::MatrixXd is typedefed as column-major.

Question: How are Eigen::MatrixXd::row(rowId) and Eigen::MatrixXd::col(colId) implemented? (becoming important later.)

Eigen Library Fundamentals

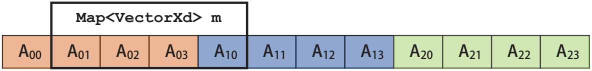
Eigen::Map

Yingrui (Ray) Chang
California Institute of Technology

- `Eigen::Map` : A matrix or vector expression mapping an existing array of data.
- usage: constructing a "Matrix/Vector" on exsiting data.

```
Eigen::MatrixXd A{3,4}; A.fill(1.);
Eigen::Map<Eigen::VectorXd> m(&A(0,1), 3);
A.fill(10.);
std::cout<<m.transpose()<<std::endl;
```

output: 10 10 10 10



- `Eigen::Map` can be implicitly converted to `Eigen::MatrixXd` or `Eigen::VectorXd`

```
Eigen::Map<VectorXd> m(&A(0,1), 3);
Eigen::VectorXd vec = m;
```

but usually involves copying data to create a new vector.

Lib15x Basic Data Types

Yingrui (Ray) Chang
California Institute of Technology

- data is stored in `Eigen::MatrixXd` with each sample corresponds to one row.
- labels type has corresponding enum associated to identify classification or regression for type safety.

```
enum class ProblemType {Classification, Regression};

struct Labels {
  ProblemType _labelType;
  Eigen::VectorXd _labelData;
}
```

- each model defines its own problem type and loss function as static data members:

```
class ClassifierModel {
public:
  static const ProblemType ModelType = ProblemType::Classification;
  static double LossFunction(const Labels&, const Labels&);
  ... ...
}
```

Lib15x User Interfaces(1)

Support Vector Machine

Yingrui (Ray) Chang
California Institute of Technology

import Lib15x namespace:

```
#include <Lib15x.hpp>
using namespace Lib15x;
```

defining learning model:

```
using Kernel = Kernels::RBF;
using LearningModel = Models::LibSVM<Kernel>;
```

model declaration:

```
LearningModel learningModel{numberOfFeatures, numberOfClasses, Kernel{gamma}};
```

training and predicting:

```
learningModel.train(trainData, trainLabels);
Labels predictedLabels = learningModel.predict(trainData);
```

compute in-sample accuracy:

```
constexpr double (*LossFunction)(const Labels&, const Labels&) =
  LearningModel::LossFunction;
double accuracy = 1.0 - LossFunction(predictedLabels, trainLabels);
```

full example at *Lib15x/src/example/libsvm_example.cc*

Lib15x User Interfaces (2)

Cross Validation, Multi-class Classification

Yingrui (Ray) Chang
California Institute of Technology

define multiclass with a binary class model:

```
using Kernel = Kernels::RBF;
using BinaryModel = Models::LibSVM<Kernel>;
using MulticlassModel = Models::MulticlassClassifier<BinaryModel>;
```

declare multiclass model:

```
long numberOfClasses = 2;
BinaryModel binaryModel{numberOfFeatures, numberOfClasses, C, Kernel{gamma}};
MulticlassModel multiclassModel{numberOfFeatures, numberOfClasses, binaryModel};
```

compute cross validation score:

```
CrossValidation crossValidation{trainData, trainLabels, true};
VectorXd losses = crossValidation.computeValidationLosses(&multiclassModel);
```

full example at *Lib15x/src/example/cross_validation_example.cc*

Outline

Yingrui (Ray) Chang
California Institute of Technology

Inheritence & Virtual Functions

Support Vector Machine, Kernels

Yingrui (Ray) Chang
California Institute of Technology

kernel interface:

```
class Kernel {
public:
  virtual double operator() (const VectorXd&, const VectorXd&) const = 0;
};
```

specific kernel implementations:

```
class KernelRBF : public Kernel {
public:
  explicit RBF(double gamma) : _gamma(gamma) {}
  double operator() (const VectorXd& x, const VectorXd& y) const {
    return exp(-_gamma*(x-y).squaredNorm());
  }
private:
  double _gamma;
};

class KernelDot : public Kernel {
public:
  double operator()(const VectorXd& x, const VectorXd& y) const {
    return x.dot(y);
  }
};
```

Inheritence & Virtual Functions

Support Vector Machine, Kernels

Yingrui (Ray) Chang
California Institute of Technology

support vector machine:

```
class SupportVectorClassifier {
public:
  //...
  void train(const MatrixXd& trainData, const Labels& labels) {
    //at some point ...
    G(idI, idJ) = (*kernel)(trainData.row(idI), trainData.row(idJ));
  }
private:
  //...
  Kernel* kernel;
};
```

Inheritence & Virtual Functions

Support Vector Machine, Kernels

Yingrui (Ray) Chang
California Institute of Technology

support vector machine:

```
class SupportVectorClassifier {
public:
  //...
  void train(const MatrixXd& trainData, const Labels& labels) {
    //at some point ...
    G(idI, idJ) = (*kernel)(trainData.row(idI), trainData.row(idJ));
  }
private:
  //...
  Kernel* kernel;
};
```

implementation difficulties caused by pointer data member:

- **choice of pointer** (`std::unique_ptr`, `std::shared_ptr` or raw $*$);
- **more implementation work:**
  copy constructor, copy assignment operator, destructor,
  move constructor; move assignment operator (after C++11),
  clone()...
- **exception safty** ...

Performance Issues Caused by Virtual Function

Yingrui (Ray) Chang
California Institute of Technology

- an extra v-pointer (large data type: typically 8 bytes);
- tracing pointers (cache unfriendly);
- harmful to data alignment.
- impossible for function inlining.

Performance Issues Caused by Virtual Function

Yingrui (Ray) Chang
California Institute of Technology

- an extra v-pointer (large data type: typically 8 bytes);
- tracing pointers (cache unfriendly);
- harmful to data alignment.
- impossible for function inlining.

benchmark test designed by *crashwork*:

```
class TestVec {
public:
  float GetX() { return x; }
  float SetX(float to) { return x=to; }
  // GetY(), GetZ()45 SetY(), SetZ(), GetW(), SetW()
private:
  float x,y,z,w;
};
```

test process:

- populate three arrays each with 1024 `TestVec`
- ran a loop that calls `Get*()` and `Set*()` to add each member to one another 1000 times.
- record average time used in each loop when `Get*()` and `Set*()` are *virtual*, *direct* and *inline*.

source code available at :
http://assemblyrequired.crashworks.org/code-for-testing-virtual-function-speed/ (for Microsoft VS compiler)
https://github.com/yingryic/performance_study/ (for GCC compiler)

Performance Issues Caused by Virtual Function

Yingrui (Ray) Chang
California Institute of Technology

testing results:

test on my own machine:

compiler: g++-4.9
cpu: Intel Core i7, 2.20GHz
L1 cache size: 64KB

average running time for a single loop:

- virtual: 75ms
- direct: 68ms
- inline: 2ms

report by *crashwork*

compiler: Microsoft Visual Studio
cpu: 3GHz
L1 cache size: big enough to fit all the data

- virtual: 160ms
- direct: 68ms
- inline: 8ms

Performance Issues Caused by Virtual Function

Yingrui (Ray) Chang
California Institute of Technology

testing results:

test on my own machine:

compiler: g++-4.9
cpu: Intel Core i7, 2.20GHz
L1 cache size: 64KB

average running time for a single loop:

- virtual: 75ms
- direct: 68ms
- inline: 2ms

report by *crashwork*

compiler: Microsoft Visual Studio
cpu: 3GHz
L1 cache size: big enough to fit all the data

- virtual: 160ms
- direct: 68ms
- inline: 8ms

take-home message:

- virtual function call cannot be faster than direct function call (usually slower because of the vpointer tracing mechanism).
- consider function inlining whenever possible.

Template Alternative

Support Vector Machine

Yingrui (Ray) Chang
California Institute of Technology

kernels are not derived from any class:

```
class KernelDot {
public:
  //...
  double operator() (const VectorXd& x, const VectorXd& y) const {
    return x.dot(y);
  }
};
```

Template Alternative

Support Vector Machine

Yingrui (Ray) Chang
California Institute of Technology

kernels are not derived from any class:

```
class KernelDot {
public:
  //...
  double operator() (const VectorXd& x, const VectorXd& y) const {
    return x.dot(y);
  }
};
```

svm class contains a kernel object instead of pointer:

```
template<class Kernel>
class SupportVectorClassifier {
public:
  //...
  void train(const MatrixXd& trainData, const Labels& labels) {
    //...
    G(idI, idJ) = kernel(trainData.row(idI), trainData.row(idJ));
  }
private:
  //...
  Kernel kernel;
};
```

Template vs Virtual Function

Yingrui (Ray) Chang
California Institute of Technology

advantage of using template:

- no extra pointer and virtual function calls.
- can rely on the default copy/move constructor/assignment operator, destructor, etc.
- possible for function inlining (compiler can see the implementation).

Template vs Virtual Function

Support Vector Machine

Yingrui (Ray) Chang
California Institute of Technology

advantage of using template:

- no extra pointer and virtual function calls.
- can rely on the default copy/move constructor/assignment operator, destructor, etc.
- possible for function inlining (compiler can see the implementation).

advice from *Bjarne Stroustrup*:

- Prefer a template over derived classes when run-time efficiency is at a premium.
- Prefer derived classes over a template if adding new variants without recompilation is important.
- Prefer a template over derived classes when no common base can be defined.
- Prefer a template over derived classes when built-in types and structures with compatibility constraints are important.

The C++ Programming Language, 3rd Edition, chapter 13.8

Outline

Yingrui (Ray) Chang
California Institute of Technology

Avoid Data Copying with Individual Sample

EIGEN::MAP

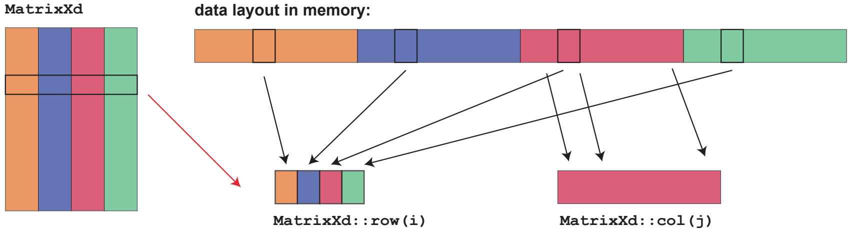
Yingrui (Ray) Chang
CALIFORNIA INSTITUTE OF TECHNOLOGY

data forwarding problem:

```
Kernel::operator(const VectorXd&, const VectorXd&) const;
SupportVectorClassifier::train(const MatrixXd& trainData, const Labels& trainLabels) {
  //...
  G(idI, idJ)=kernel(trainData.row(sampleIdI), trainData.row(sampleIdJ));
}
```

question: what does `Eigen::MatrixXd::row(rowId)` **return? possibilily avoiding copying?**

answer: cannot avoid copying since `Eigen::MatrixXd` **is colomn major by default.**



but we should do better!

Avoid Data Copying with Individual Sample

EIGEN::MAP

Yingrui (Ray) Chang
California Institute of Technology

use row major matrix by default:

```
using MatrixXd = Eigen::Matrix<double, Eigen::Dynamic, Eigen::Dynamic, Eigen::RowMajor>;
```

`MatrixXd`

data layout in memory:

possible to create “reference” to each data sample:

```
Map<VectorXd> vec_1(&trainData(1,0), numberOfCols)
```

forwarding `Eigen::Map` instead of vector to avoid data copying:

```
Kernel::operator()(const Eigen::Map<const VectorXd>&,
                   const Eigen::Map<const VectorXd>&) const;
```

```
SVM::train(const Eigen::MatrixXd& trainData, const Labels& trainLabels) {
  //...
  Eigen::Map<const VectorXd> dataI(&trainData(idI,0), numberOfFeatures);
  Eigen::Map<const VectorXd> dataJ(&trainData(idJ,0), numberOfFeatures);
  G(idI, idJ)=kernel(dataI, dataJ);
}
```

Avoid Data Copying

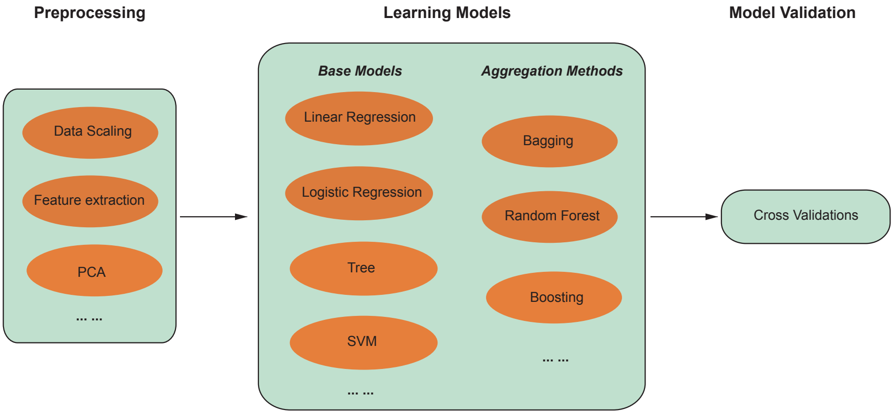
Interface Design

Yingrui (Ray) Chang
California Institute of Technology

training interface:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

problem: typical machine learning pipline:



Avoid Data Copying

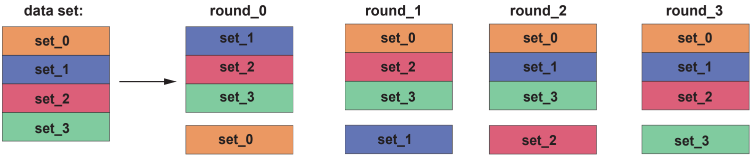
INTERFACE DESIGN

Yingrui (Ray) Chang
CALIFORNIA INSTITUTE OF TECHNOLOGY

training interface:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

cross validation in action:



Avoid Data Copying

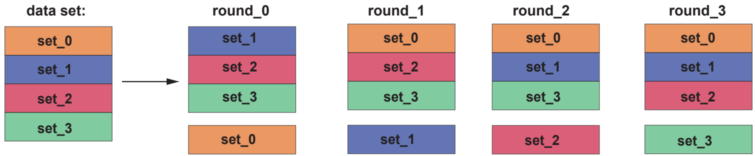
INTERFACE DESIGN

Yingrui (Ray) Chang
CALIFORNIA INSTITUTE OF TECHNOLOGY

training interface:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

cross validation in action:



implementation:

```
template<class LearningModel>
VectorXd CrossValidation::computeValidationLosses(LearningModel* learningModel) {
  //...
  for (long currentRoundId =0; ...) {
    //...
    MatrixXd trainDataOfThisRound{...};
    Labels trainLabesOfThisRound{...};
    //populate training and testing data
    learningModel->train(trainDataOfThisRound, trainLabelsOfThisRound);
    //do the same for testing ...
  }
};
```

Index Based Training

Yingrui (Ray) Chang
California Institute of Technology

inner training interface:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices) {
  //train the model with the data identified by the trainIndices
};
```

Index Based Training

Yingrui (Ray) Chang
California Institute of Technology

inner training interface:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices) {
  //train the model with the data identified by the trainIndices
};
```

used by higher level classes:

- cross validation:

```
template<class LearningModel>
VectorXd CrossValidation::computeValidationLosses(LearningModel* learningModel) {
  for (long currentRoundId =0; ...) {
    //populate training data indices
    learningModel->train(_data, _labels, trainIndces);
  }
};
```

- bagging classifier:

```
BaggingClassifier::train(const MatrixXd& trainData, const Labels& trainLabels,
                         const vector<long>& trainIndices) {
  for (long modelId =0; ...) {
    //sample training data indices into trainIndices with replacement
    baseModels[modelId}.train(_data, _labels, trainIndcesForThisModel);
  }
};
```

- Multi-class model, Random Forest, ...

Outline

Yingrui (Ray) Chang
California Institute of Technology

Interfaces Summary

Yingrui (Ray) Chang
California Institute of Technology

user interfaces:

```
void train(const MatrixXd& trainData, const Labels& trainLabels) {
  //check user input
  //populate trainIndices
  train(trainData, trainLabels, trainIndices);
}

Labels predict(const MatrixXd& testData) {
  //check user input
  //populate testIndices
  return predict(trainData, trainLabels, trainIndices);
}
```

Interfaces Summary

Yingrui (Ray) Chang
California Institute of Technology

user interfaces:

```
void train(const MatrixXd& trainData, const Labels& trainLabels) {
  //check user input
  //populate trainIndices
  train(trainData, trainLabels, trainIndices);
}

Labels predict(const MatrixXd& testData) {
  //check user input
  //populate testIndices
  return predict(trainData, trainLabels, trainIndices);
}
```

interfaces for library developer:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices) {
  //do training
}

Labels predict(const MatrixXd& trainData, const Labels& trainLabels,
               const vector<long>& trainIndices) {
  //do predicting
}
```

Implementation Difficulties

Yingrui (Ray) Chang
California Institute of Technology

user interfaces:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

```
Labels predict(const MatrixXd& testData);
```

interfaces for library developer:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices);
```

```
Labels predict(const MatrixXd& trainData, const Labels& trainLabels,
               const vector<long>& trainIndices);
```

Implementation Difficulties

Yingrui (Ray) Chang
California Institute of Technology

user interfaces:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

```
Labels predict(const MatrixXd& testData);
```

interfaces for library developer:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices);
```

```
Labels predict(const MatrixXd& trainData, const Labels& trainLabels,
               const vector<long>& trainIndices);
```

template awkward:

same user interface implementations for every model, but template requires every model implementing all the interfaces.

Implementation Difficulties

Yingrui (Ray) Chang
California Institute of Technology

user interfaces:

```
void train(const MatrixXd& trainData, const Labels& trainLabels);
```

```
Labels predict(const MatrixXd& testData);
```

interfaces for library developer:

```
void train(const MatrixXd& trainData, const Labels& trainLabels,
           const vector<long>& trainIndices);
```

```
Labels predict(const MatrixXd& trainData, const Labels& trainLabels,
               const vector<long>& trainIndices);
```

template awkward:

same user interface implementations for every model, but template requires every model implementing all the interfaces.

solution:

need to explore class hierarchy to ease implementation but without using virtual functions.

Curiously Recurring Template Pattern (CRTP)

Yingrui (Ray) Chang
California Institute of Technology

base model:

```
template<class DerivedModel>
class BaseModel {
public:
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //do all the checking
    //populate indices for all training data
    static_cast<DerivedModel*>(this)->train(trainData, trainLabels, trainIndices);
  }
};
```

Curiously Recurring Template Pattern (CRTP)

Yingrui (Ray) Chang
California Institute of Technology

base model:

```
template<class DerivedModel>
class BaseModel {
public:
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //do all the checking
    //populate indices for all training data
    static_cast<DerivedModel*>(this)->train(trainData, trainLabels, trainIndices);
  }
};
```

derived model:

```
class SVM : public BaseModel<SVM> {
public:
  using BaseModel<SVM> train;
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //do the actual training
  }
};
```

```
template<typename ImpurityRule>
class TreeClassifier : public BaseModel<TreeClassifer<Impurityrule> > {...};
```

Curiously Recurring Template Pattern (CRTP)

Yingrui (Ray) Chang
California Institute of Technology

base model:

```
template<class DerivedModel>
class BaseModel {
public:
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //do all the checking
    //populate indices for all training data
    static_cast<DerivedModel*>(this)->train(trainData, trainLabels, trainIndices);
  }
};
```

derived model:

```
class SVM : public BaseModel<SVM> {
public:
  using BaseModel<SVM> train;
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //do the actual training
  }
};
```

```
template<typename ImpurityRule>
class TreeClassifier : public BaseModel<TreeClassifer<Impurityrule> > {...};
```

user code:

```
SVM model{...}; model.train(trainData, trainLabels);
TreeClassifier<Gini> model{...}; model.train(trainData, trainLabels);
```

Example: Lib15x BaseClassifier

Yingrui (Ray) Chang
California Institute of Technology

```
template<class DerivedClassifier>
class _BaseClassifier {
public:
  static const ProblemType ModelType = ProblemType::Classification;
  static constexpr const char* ModelName=DerivedClassifier::ModelName;
  static double LossFunction(const Labels& predictedLabels, const Labels& testLabels) {...};
```

```
  long getNumberOfFeatures() const {...};
  long getNumberOfClasses() const {...};
  VerboseFlag& whetherVerbose();
protected:
  long _numberOfFeatures;
  long _numberOfClasses;
  VerboseFlag _verbose = VerboseFlag::Quiet;
};
```

Every model should implement:

- **train(...);**
- **predictOne(...);**
- **_clearModel( );**

Lib15x/src/include/internal/_BaseClassifier.hpp

Example: Lib15x BaseClassifier

Yingrui (Ray) Chang
California Institute of Technology

```
template<class DerivedClassifier>
class _BaseClassifier {
public:
  static const ProblemType ModelType = ProblemType::Classification;
  static constexpr const char* ModelName=DerivedClassifier::ModelName;
  static double LossFunction(const Labels& predictedLabels, const Labels& testLabels) {...};
  void train(const MatrixXd& trainData, const Labels& trainLabels) {
    //...
    static_cast<DerivedModel*>(this)->train(trainData, trainLabels, trainIndices);
  }
  Labels predict(const MatrixXd& testData) const {...};
  Labels predict(const MatrixXd& testData, const vector<long>& testIndices) const {
    //...
    for (auto testDataId : testIndices){
      Map<const VectorXd> instance(&testData(testDataId, 0), _numberOfFeatures);
      predictedLabels._labelData(testDataId) =
        static_cast<const DerivedClassifier*>(this)->predictOne(instance);
    }
    return predictedLabels;
  }
  void clear() {
    static_cast<DerivedClassifier*>(this)->_clearModel();
  }
  long getNumberOfFeatures() const {...};
  long getNumberOfClasses() const {...};
  VerboseFlag& whetherVerbose();
protected:
  long _numberOfFeatures;
  long _numberOfClasses;
  VerboseFlag _verbose = VerboseFlag::Quiet;
};
```

Every model should implement:

- **train(...);**
- **predictOne(...);**
- **_clearModel( );**

Lib15x/src/include/internal/_BaseClassifier.hpp

Example: TreeClassifier

Yingrui (Ray) Chang
California Institute of Technology

```
template<double (*ImpurityRule)(const vector<long>&) = gini>
class TreeClassifier : public _BaseClassifier<TreeClassifier<ImpurityRule> > {
public:
  using BaseClassifier = _BaseClassifier<TreeClassifier<ImpurityRule> >;
  using BaseClassifier::train;
  static constexpr const char* ModelName = "TreeClassifier";
  static constexpr double (*LossFunction)(const Labels&, const Labels&) = BaseClassifier::LossFunction;







private:
  long _minSamplesInALeaf;
  long _minSamplesInANode;
  long _maxDepth;
  long _maxNumberOfLeafNodes;
  _ClassificationTree _tree;
};
```

Lib15x/src/include/models/TreeClassifier.hpp

Example: TreeClassifier

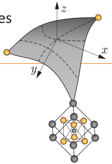
Yingrui (Ray) Chang
California Institute of Technology

```
template<double (*ImpurityRule)(const vector<long>&) = gini>
class TreeClassifier : public _BaseClassifier<TreeClassifier<ImpurityRule> > {
public:
  using BaseClassifier = _BaseClassifier<TreeClassifier<ImpurityRule> >;
  using BaseClassifier::train;
  static constexpr const char* ModelName = "TreeClassifier";
  static constexpr double (*LossFunction)(const Labels&, const Labels&) = BaseClassifier::LossFunction;

  void train(const MatrixXd& trainData, const Labels& trainLabels, const vector<long>& trainIndices) {
    //...
  }
  double predictOne(const Map<const VectorXd>& instance) const {
    //...
  }
  void _clearModel() {
    //...
  }

private:
  long _minSamplesInALeaf;
  long _minSamplesInANode;
  long _maxDepth;
  long _maxNumberOfLeafNodes;
  _ClassificationTree _tree;
};
```

Lib15x/src/include/models/TreeClassifier.hpp



Thank you for your interest!

Yingrui (Ray) Chang

yingryic@gmail.com

http://www.its.caltech.edu/~ycchang

References::

http://www.its.caltech.edu/~ycchang/lib15x.html

http://www.github.com/lib15x/