Basic information

Name: Kirill Mishchenko
Email: ki.mishchenko@gmail.com
Profiles: GitHub LinkedIn
Maillist submissions: 1, 2, 3

School: Ural Federal University
Field: Computer science
Date study was started: 2014, September
Expected graduation date: 2018, June

Programming experience

- C++ (3.5/5.0)
  - Bachelor degree work (2013, November - 2014, April). The work was directed to do initial investigation of the problem of creating intelligent file agents that are capable to assist users in managing their files. To see code samples go to this module (as it is a more or less separate one). Also see more details in publications 1 and 2, but don't treat them as too much serious research works.
  - Internship in Yandex search, snippets group, summer 2014.
- Python (3.5/5.0). Using it in my everyday life.
- C# (3.0/5.0). Internship in SKB Kontur, billing development team, summer 2012.
- Scala (2.0/5.0). Passing the course Functional Programming Principles in Scala in 2017. Mentioning it since metaprogramming in C++ is usually considered as functional programming.

Open source experience

I'm pretty new to open source development. Nevertheless, I have had an oportunity to do some initial steps to get involved into mlpack development. Designing solution for cross-validation and hyper-parameter tuning project, I have identified what parts of API in mlpack are out of conventions, have created issues for that (#929, #935, #948) and have provided patches (#940, #957, #962), which have been merged.

Machine learning experience

I have taken the online course Machine Learning by Stanford, as well as some other courses related to ML (Artificial Intelligence by Berkeley, Deep Learning by Google and others), from which I have been studying algorithms for a bunch of machine learning tasks like regression, classification, clustering, taking optimal actions, predicting hidden states and others, as well as strategies to apply machine learning algorithms ("debugging" machine learning algorithms, hyper-parameter tuning). I apply machine learning in my current research work – finding/generating a humorous response given a textual input.

Cross-validation and hyper-parameter tuning project proposal

I plan to implement the following functionality.

- Cross-validation
  - Measurements
    - Accuracy
    - Mean squared error
    - Precision
    - Recall
    - F1
  - K-fold cross-validation
  - Simple cross-validation (splitting data once with validation set size specified by a user; it can be more appropriate when we have a lot of training data or when training is a time consuming process)
- Hyper-parameter tuning
  - Grid search based tuning

Designing cross-validation and hyper-parameter tuning modules, I was assuming that all classification and regression models have training constructors (it is indeed true now, see the related issue and patch), all regression models have a `Predict` method, while all classification models have a `Classify` method (it is also true now, see the related issue and patch), and that all classification and regression models have training constructors (as well as corresponding `Train` methods) in at least one of the forms:

```
MLAlgorithm(const MatType& data, const PredictionsType& predictions,
    ...other arguments...);
```

or

```
MLAlgorithm(const MatType& data, const PredictionsType& predictions,
    const WeightsType& weights, ...other arguments...);
```

The last is not completely true by the moment of writing the proposal, but the issue is already received attention (#929) and is appreciated as worth to be solved.

In the following sections I want to present usage examples of prospective cross-validation and hyper-parameter tuning modules, as well as provide links where more implementation details are given.

Cross-validation

Suppose we have some data to train and validate on.

```
arma::mat data /* = ... */;
arma::Row<size_t> labels /* = ... */;
size_t numClasses = 5;
```

To run k-fold cross-validation (by default k = 10) for softmax regression with accuracy as a measurement we will need to write the following piece of code.

```
KFoldCV<SoftmaxRegression<>, Accuracy> softmaxCV(data, labels);
double lambda = 0.1;
double softmaxAccuracy = softmaxCV.Evaluate(numClasses, lambda);
```

The `Evaluate` method will rely on the following `SoftmaxRegression` constructor:

```
SoftmaxRegression(const arma::mat& data,
                  const arma::Row<size_t>& labels,
                  const size_t numClasses,
                  const double lambda = 0.0001,
                  const bool fitIntercept = false);
```

which has the parameters `numClasses` and `lambda` after two conventional arguments (`data` and `labels`). We can skip passing `fitIntercept` (as well as `lambda`), since there is a default value.

In the case of weighted learning we will need to pass weights during construction and to specify a template bool parameter for weighted learning explicitly.

```
arma::rowvec weights /* = ... */;
KFoldCV<DecisionStump<>, Accuracy, true> dStumpCV(data, labels, weights);
```

Then we call `Evaluate` in a similar way.

```
size_t bucketSize = 5;
double dStumpAccuracy = dStumpCV.Evaluate(numClasses, bucketSize);
```

Simple cross-validation has the same interface as for k-fold cross-validation, except it takes as an argument a proportion (from 0 to 1) of the data used as a validation set in its constructor. To validate `LinearRegression` with 20% of training data we will need to write the following code.

```
arma::vec responces /* = ... */;
float validationPortion = 0.2F;
SimpleCV<LinearRegression, MeanSquaredError>
    lRegressionSV(data, responces, validationPortion);
double lRegressionLambda = 0.05;
double lRegressionMSE = lRegressionSV.Evaluate(lRegressionLambda);
```

Note that types of data and predictions are not specified explicitly while constructing cross-validation objects in the examples. The default value for the type of data is `arma::mat`, what will cover most using cases. The type of predictions (like `arma::Row<size_t>` for classification algorithms and `arma::mat`, `arma::vec` or `arma::rowvec` for regression algorithms) is supposed to be deduced automatically from the type of an algorithm (like `SoftmaxRegression<>`

or `LinearRegression`). It's going to be done by exploring what Train signatures the algorithm class has. I have already implemented a prototype for detecting what Train forms a class has, you can see it in Appendix A. See also more details about k-fold cross-validation and simple cross-validation interfaces in Appendix B.

Hyper-parameter tuning

The usage of the prospective hyper-parameter tuning class is analogous to the validation classes. If we want to tune hyper-parameters for softmax regression by optimizing accuracy and applying 5-fold cross validation, we can construct a grid search based optimizer in the following way.

```
GridSearchOptimizer<SoftmaxRegression<>, Accuracy, KFoldCV>
    softmaxOptimizer(data, labels, 5);
```

Before optimization we need to provide a collection of values for each additional constructor argument of `SoftmaxRegression<>`.

```
std::array<size_t, 1> numClasses = {5}
arma::vec lambdas = arma::logspace(-3, 1); // {0.001, 0.01, 0.1, 1}
```

After that, we need to pass the collections to the method `Optimize`, which returns a tuple of optimal (from what we specified) parameters.

```
std::tuple<size_t, double> bestSoftmaxParams =
    softmax_optimizer.Optimize(numClasses, lambdas);
```

We also can request the value of accuracy for the found parameters, as well as a corresponding trained model.

```
double bestSoftmaxAccuracy = softmaxOptimizer.BestMeasurement();
SoftmaxRegression<>& bestSoftmaxModel = softmaxOptimizer.BestModel();
```

In the case of weighted learning we will need to pass weights into constructor and to specify a template bool parameter for weighted learning explicitly.

```
arma::rowvec weights /* = ... */;
GridSearchOptimizer<DecisionStump<>, Accuracy, KFoldCV, true>
    dStumpOptimizer(data, labels, weights);
```

A more complex example is the following.

```
GridSearchOptimizer<HoeffdingTree<>, Accuracy, SimpleCV>
    hoeffdingTreeOptimizer(data, labels, 0.2);

// Setting a set of values for each parameter
std::array<data::DatasetInfo, 1> datasetInfo /* = {...} */;
std::array<bool, 1> batchTraining = {false};
arma::vec successProbabilities = arma::regspace(0.9, 0.01, 0.99);
std::array<size_t, 3> maxSamplesSet = {0, 3};
std::array<size_t, 3> checkIntervals = {80, 100, 120};
```

```
std::array<size_t, 3> minSamplesSet = {50, 100, 150};

// Making variables for best parameters
data::datasetInfo _;
size_t __;
bool ___;
double successProbability;
size_t maxSamples;
size_t checkInterval;
size_t minSamples;

// Finding best parameters
auto bestParameters =
    hoeffdingTreeOptimizer.Optimize(datasetInfo, numClasses, batchTraining,
        successProbabilities, maxSamplesSet, checkIntervals, minSamplesSet);

// Unpacking best parameters
std::tie(_, __, ___, successProbability, maxSamples, checkInterval,
    minSamples) = bestParameters;
```

Note that we don't specify a return tuple type when calling the method `Optimize`. In C++14 it can be implemented by simply declaring to deduce the return type from a method body. In C++11 we need to explicitly specify how the return type depends on method argument types. A prototype solution can be found in Appendix C.

The information about whether we need to maximize or minimize a given measurement is stored in its class. See details in Appendix D. See also more details about grid search based parameter tuning interface in Appendix E.

Testing

Testing should not be very tricky, but, nevertheless, it's worth to mention how it is planed to be done. To test measurements, we need to check that for given data an appropriate prediction method is called (Classify for classification algorithms, Predict for regression algorithms) and the results were used in the way to calculate a given measurement.

To test validation classes, we need to check that it works for algorithms with different interfaces of constructors and Train methods, and compare results retrieved manually and from cross-validation classes. Roughly the same approach for testing can be applied to the hyper-parameter tuning module.

Timeline

Below is a rough plan when things are supposed to be done. I plan to start working in the community bonding period by solving the issue #929. Since the beginning of June I will be able to work full-time. I can be occupied from May 31 to June 3 by participating in the Dialogue conference, which accepted the work of mine and my colleges. In this case I will do the planed

work week earlier. Also note that I plan to take vacation from August 21 to August 25, during which I am going to participate in Russian Summer School in Information Retrieval (RuSSIR), which takes place in the city I live.

Community bonding period (May 4 - May 29) - normalizing training constructors and Train methods where it hasn't been done yet.

1 week (May 30 - June 5) - implementing and testing accuracy and mean squared error.

2 week (June 6 - June 12) - finishing to test accuracy and mean squared error, starting to implement simple cross-validation.

3 week (June 13 - June 19) - implementing simple cross-validation.

4 week (June 20 - June 26) - testing simple cross-validation.

5 week (June 27 - July 3) - implementing k-fold cross validation.

6 week (July 4 - July 10) - implementing and testing k-fold cross validation.

7 week (July 11 - July 17) - finishing to test k-fold cross validation, starting to implement grid search based hyper-parameter tuning.

8 week (July 18 - July 24) - implementing grid search based hyper-parameter tuning.

9 week (July 25 - July 31) - testing grid search based hyper-parameter tuning.

10 week (August 1 - August 7) - implementing precision, recall and F1.

11 week (August 8 - August 14) - testing precision, recall and F1.

12 week (August 15 - August 20) - implementing extra functionality, fixing bugs, writing better documentation.

13 week (August 21 - August 25) - vacation (attending RuSSIR).

13 week (August 26 - August 29) - implementing extra functionality, fixing bugs, writing better documentation.

If time is left, some extra functionality can be implemented, like the following.

- Some additional measurements of performance for machine learning algorithms (see e.g. the scikit-learn list)
- Some additional cross-validation or hyper-parameter tuning classes (see for inspiration the ones implemented in scikit-learn here and here).

Additional comments

Even though I have discussed a lot of parts of the proposal with the prospective mentor, it is likely that some parts can be improved or edited. We can discuss these details in the community bonding period.

All code samples (including ones in the appendixes) can be found in GitHub.

Appendix A Method form detection

Implementation

The following code is possible implementation of method form detection, which can be viewed as an "advanced" version of the macros `HAS_MEM_FUNC` from <mlpack/core/util/sfinae_utility.hpp>. Consult the comment of @micyril made March 31 under the issue #929 for an overview of the approach. The code is presented for the purpose of showing that method form detection is possible rather than to provide a production-quality solution.

Here the struct `HasTrain` is defined, which is bound to `Train` methods. It is straightforward to implement the solution as a macros that generates a corresponding struct for an arbitrary method name analogously to `HAS_MEM_FUNC`.

```
#include <type_traits>

template<template<typename...> class MethodTemplate,
         typename Class,
         int AdditionalArgsCount>
struct MethodTypeDetecter;

template<template<typename...> class MethodTemplate, typename Class>
struct MethodTypeDetecter<MethodTemplate, Class, 0>
{
  void operator()(MethodTemplate<Class>);
};

template<template<typename...> class MethodTemplate, typename Class>
struct MethodTypeDetecter<MethodTemplate, Class, 1>
{
  template<typename T1>
  void operator()(MethodTemplate<Class, T1>);
};

template<template<typename...> class MethodTemplate, typename Class>
struct MethodTypeDetecter<MethodTemplate, Class, 2>
{
  template<typename T1, typename T2>
  void operator()(MethodTemplate<Class, T1, T2>);
};

template<typename>
struct True
{
  const static bool value = true;
};

template<typename FunReturnDecltype, typename Result = void>
using EnableIfCompiles =
    typename std::enable_if<True<FunReturnDecltype>::value, Result>::type;
```

```
template<template<typename...> class MethodTemplate,
          typename Class,
          int AdditionalArgsCount>
struct HasTrain
{
  using yes = char[1];
  using no = char[2];

  // Making short aliases
  template<template<typename...> class MT, typename C, int N>
  using MTD = MethodTypeDetecter<MT, C, N>;
  template<typename C, typename...Args>
  using MT = MethodTemplate<C, Args...>;
  static const bool N = AdditionalArgsCount;

  template<typename C>
  static EnableIfCompiles<decltype(MTD<MT, C, N>()(&C::Train)), yes&> chk(int);
  template<typename>
  static no& chk(...);

  static bool const value = sizeof(chk<Class>(0)) == sizeof(yes);
};
```

Testing

The solution can be tested in the following way.

```
#include <iostream>

#include <armadillo>

class A
{
public:
  void Train(const arma::mat&, const arma::Row<size_t>&, int);
  void Train(const arma::vec&, size_t, int);
};

class B
{
public:
  void Train(const arma::mat&, const arma::rowvec&);
};

template<typename Class, typename...T>
using TrainForm1 =
    void(Class::*)(const arma::mat&, const arma::Row<size_t>&, T...);

template<typename Class, typename...T>
using TrainForm2 = void(Class::*)(const arma::mat&, const arma::rowvec&, T...);
```

```
int main()
{
  std::cout << HasTrain<TrainForm1, A, 0>::value << std::endl; //prints 0
  std::cout << HasTrain<TrainForm1, A, 1>::value << std::endl; //prints 1
  std::cout << HasTrain<TrainForm1, B, 0>::value << std::endl; //prints 0
  std::cout << HasTrain<TrainForm1, B, 1>::value << std::endl; //prints 0
  std::cout << std::endl;
  std::cout << HasTrain<TrainForm2, A, 0>::value << std::endl; //prints 0
  std::cout << HasTrain<TrainForm2, A, 1>::value << std::endl; //prints 0
  std::cout << HasTrain<TrainForm2, B, 0>::value << std::endl; //prints 1
  std::cout << HasTrain<TrainForm2, B, 1>::value << std::endl; //prints 0
}
```

Appendix B Cross-validation interfaces

K-fold cross-validation

```
#include <armadillo>

/**
 * The class KFoldCV implements k-fold cross validation for regression and
 * classification algorithms.
 *
 * @tparam MLAlgorithm A regression or classification algorithm.
 * @tparam Measurement A measurement to assess the quality of the trained model.
 * @tparam MatType A matrix type of data.
 * @tparam PredictionsType A type of predictions of the algorithm. Usually it is
 *     arma::Row<size_t> for classification algorithms and arma::vec/arma::mat
 *     for regression algorithms.
 * @tparam WeightedLearning An indicator whether weighted learning should be
 *     performed.
 */
template<typename MLAlgorithm,
         typename Measurement,
         bool WeightedLearning = false,
         typename MatType = arma::mat>
class KFoldCV
{
public:
  /* After deducing the type of predictions making an alias for that */
  // using PredictionsType = ...using metaprogramming...;

  /**
   * A constructor that prepares data for performing k-fold cross validation.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data. Should be a vector (a
   *     column or a row) or a column-major matrix.
   * @param k An amount of folds for k-fold cross validation.
   */
  KFoldCV(const MatType& xs, const PredictionsType& ys, const size_t k = 10);

  /**
   * A constructor that prepares data for performing k-fold cross validation
   * with weighted learning.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data. Should be a vector (a
   *     column or a row) or a column-major matrix.
   * @param weights Weights for points from the data.
   * @param k An amount of folds for k-fold cross validation.
   */
  KFoldCV(const MatType& xs,
          const PredictionsType& ys,
```

```
          const arma::rowvec weights,
          const size_t k = 10);

  /**
   * Perform k-fold cross validation and return the mean value of measurements
   * for all splits of the data. The method uses different constructors of the
   * class MLAlgorithm depending on the flag WeightedLearning:
   * MLAlgorithm(trainingXs, trainingYs, mlAlgorithmArgs...) when
   * WeightedLearning is false, and MLAlgorithm(trainingXs, trainingYs,
   * trainingWeights, mlAlgorithmArgs...) otherwise.
   *
   * @param mlAlgorithmArgs A pack of other constructor arguments for
   *     MLAlgorithm in addition to data and predictions (and weights in the
   *     case of weighted learning).
   */
  template<typename...MLAlgorithmArgs>
  double Evaluate(const MLAlgorithmArgs& ...mlAlgorithmArgs);

  //! Access a model trained on one of the splits.
  const MLAlgorithm& GetModel() const;

  //! Modify the model trained on one of the splits.
  MLAlgorithm& GetModel();
};
```

Simple cross-validation

```
#include <armadillo>

/**
 * The class SimpleCV splits data into training and validation sets, runs
 * training on the training set and evaluates performance on the validation set.
 *
 * @tparam MLAlgorithm A regression or classification algorithm.
 * @tparam Measurement A measurement to assess the quality of the trained model.
 * @tparam MatType A matrix type of data.
 * @tparam PredictionsType A type of predictions of the algorithm. Usually it is
 *     arma::Row<size_t> for classification algorithms and arma::vec/arma::mat
 *     for regression algorithms.
 * @tparam WeightedLearning An indicator whether weighted learning should be
 *     performed.
 */
template<typename MLAlgorithm,
         typename Measurement,
         bool WeightedLearning = false,
         typename MatType = arma::mat>
class SimpleCV
{
public:
  /* After deducing the type of predictions making an alias for that */
  // using PredictionsType = ...using metaprogramming...;
```

```
  /**
   * A constructor that splits data into training and validation set.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data. Should be a vector (a
   *     column or a row) or a column-major matrix.
   * @param validationSize A proportion (from 0 to 1) of the data used as a
   *     validation set.
   */
  SimpleCV(const MatType& xs,
             const PredictionsType& ys,
             const float validationSize);

  /**
   * A constructor that splits data into training and validation set in the case
   * of weighted learning.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data. Should be a vector (a
   *     column or a row) or a column-major matrix.
   * @param weights Weights for points from the data.
   * @param validationSize A proportion (from 0 to 1) of the data used as a
   *     validation set.
   */
  SimpleCV(const MatType& xs,
             const PredictionsType& ys,
             const arma::rowvec weights,
             const float validationSize);

  /**
   * Train on the training set and assess performance on the validation set by
   * using the class Measurement. The method uses different constructors of the
   * class MLAlgorithm depending on the flag WeightedLearning:
   * MLAlgorithm(trainingXs, trainingYs, mlAlgorithmArgs...) when
   * WeightedLearning is false, and MLAlgorithm(trainingXs, trainingYs,
   * trainingWeights, mlAlgorithmArgs...) otherwise.
   *
   * @param mlAlgorithmArgs A pack of other constructor arguments for
   *     MLAlgorithm in addition to data and predictions (and weights in the
   *     case of weighted learning).
   */
  template<typename...MLAlgorithmArgs>
  double Evaluate(const MLAlgorithmArgs& ...mlAlgorithmArgs);

  //! Access the trained model.
  const MLAlgorithm& GetModel() const;

  //! Modify the trained model.
  MLAlgorithm& GetModel();
};
```

Appendix C Tuple type deduction

Implementation

```
#include <tuple>

/* Declaring a template and defining it when CTypes is empty */
template<typename...CTypes>
struct CollectionTypes
{
  template<typename...VTypes>
  struct ValueTypes
  {
    using Tuple = std::tuple<VTypes...>;
  };
};

/* Defining a template when CTypes is not empty */
template<typename Head, typename...Tail>
struct CollectionTypes<Head, Tail...>
{
  template<typename...VTypes>
  struct ValueTypes
  {
    using Tuple = typename CollectionTypes<Tail...>::template
        ValueTypes<VTypes..., typename Head::value_type>::Tuple;
  };

  using TupleOfValues = typename ValueTypes<>::Tuple;
};

template<typename...Collections>
using TupleOfValues = typename CollectionTypes<Collections...>::TupleOfValues;
```

Testing

```
#include <iostream>
#include <array>
#include <armadillo>

int main()
{
  using int_array = std::array<int, 10>;

  using t1 = TupleOfValues<int_array>;
  using t2 = TupleOfValues<int_array, arma::vec>;
  using t3 = TupleOfValues<int_array, arma::vec, int_array>;

  std::cout << std::is_same<t1, std::tuple<int>>::value
            << std::endl; // prints 1
  std::cout << std::is_same<t2, std::tuple<int, double>>::value
            << std::endl; // prints 1
```

```
  std::cout << std::is_same<t3, std::tuple<int, double, int>>::value
            << std::endl; // prints 1
}
```

Appendix D Measurements interface

Accuracy

```
#include <armadillo>

/**
 * The class Accuracy implements the classical measurement of performance for
 * classification algorithms that is equal to a proportion of correctly labeled
 * test items among all ones for given test items.
 */
class Accuracy
{
public:
  /**
   * Run classification and calculate accuracy.
   *
   * @param model A test classification model.
   * @data Column-major data containing test items.
   * @labels Ground truth (correct) labels for the test items.
   */
  template<class MLAlgorithm, class DataType>
  static double Evaluate(MLAlgorithm& model, const DataType& data,
      const arma::Row<size_t>& labels);

  /**
   * Information for hyper-parameter tuning code. It indicates that we want
   * to maximize the measurement.
   */
  static const bool NeedsMinimization = false;
};
```

Mean squared error

```
/**
 * The class MeanSquaredError implements the measurement of performance for
 * regression algorithms that is equal to the mean squared error between
 * predicted values and ground truth (correct) values for given test items.
 */
class MeanSquaredError
{
public:
  /**
   * Run prediction and calculate the mean squared error.
   *
   * @param model A test classification model.
   * @data Column-major data containing test items.
   * @responces Ground truth (correct) target values for the test items, should
   *     be either a vector or a column-major matrix.
   */
  template<typename MLAlgorithm, typename DataType, typename ResponcesType>
  static double Evaluate(MLAlgorithm& model, const DataType& data,
```

```
      const ResponcesType& responses);

  /**
   * Information for hyper-parameter tuning code. It indicates that we want
   * to minimize the measurement.
   */
  static const bool NeedsMinimization = true;
};
```

Appendix E Hyper-parameter tuning interface

```
#include <armadillo>

/**
 * The class GridSearchOptimizer exhaustively searchs over specified values for
 * hyper parameters of a given machine learning algorithm in order to find
 * hyper parameters that lead to the best performance defined by the class
 * Measurement.
 *
 * @tparam MLAlgorithm A regression or classification algorithm.
 * @tparam Measurement A measurement to assess the quality of the trained model.
 * @tparam CV A type of cross-validation used to assess a set of hyper
 *     parameters.
 * @tparam MatType A matrix type of data.
 * @tparam PredictionsType A type of predictions of the algorithm. Usually it is
 *     arma::Row<size_t> for classification algorithms and arma::vec/arma::mat
 *     for regression algorithms.
 * @tparam WeightedLearning An indicator whether weighted learning should be
 *     performed.
 */
template<typename MLAlgorithm,
         typename Measurement,
         template<typename, typename, bool, typename> class CV,
         bool WeightedLearning = false,
         typename MatType = arma::mat>
class GridSearchOptimizer
{
public:
  /* After deducing the type of predictions making an alias for that */
  // using PredictionsType = ...using metaprogramming...;

  /**
   * A constructor that prepares cross-validation for usage.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data.
   * @param cvArgs A pack of other constructor arguments for CV in addition to
   *     data and predictions.
   */
  template<typename...CVArgs>
  GridSearchOptimizer(const MatType& xs,
                      const PredictionsType& ys,
                      const CVArgs& ...cvArgs);

  /**
   * A constructor that prepares cross-validation for usage in the case of
   *     weighted learning.
   *
   * @param xs Column-major data to train and validate on.
   * @param ys Predictions for points from the data.
```

```
   * @param weights Weights for points from the data.
   * @param cvArgs A pack of other constructor arguments for CV in addition to
   *     data and predictions.
   */
  template<typename...CVArgs>
  GridSearchOptimizer(const MatType& xs,
                      const PredictionsType& ys,
                      const arma::rowvec weights,
                      const CVArgs& ...cvArgs);

  /**
   * Exhaustively search over specified values for hyper parameters to find ones
   * that lead to the best performance. The method returns an std::tuple of the
   * best parameters.
   *
   * @param parameterCollections A pack of iterable collections, one per
   *     additional constructor agrument in MLAlgorithm. If MLAlgorithm has a
   *     constructor MLAlgorithm(xs, ys, hyperParameter1, ..., hyperParamterN)
   *     or MLAlgorithm(xs, ys, weights, hyperParameter1, ..., hyperParamterN),
   *     then you should call Optimize(valuesForHyperParameter1, ...,
   *     valuesForHyperParameterN).
   */
  template<typename...Collections>
  TupleOfValues<Collections...> Optimize(
      const Collections& ...parameterCollections);

  //! Access the best model.
  const MLAlgorithm& BestModel() const;

  //! Modify the best model.
  MLAlgorithm& BestModel();

  //! Access the performance measurement of the best model.
  double BestMeasurement() const;
};
```