STUDY AND TEST A MACHINE LEARNING/NEURAL NETWORK LIBRARY IN C++

Aurelio Marin Aranzana

POLITECNICO
MILANO 1863

Supervisor:
Prof. Luca Formaggia

INDEX

- Introduction
- mlpack structure
- OpenNN structure
- Testing mlpack
- Comparison with Scikit
- Testing OpenNN
- Comparison with Tensorflow

Introduction

- Study mlpack and OpenNN libraries

mlpack

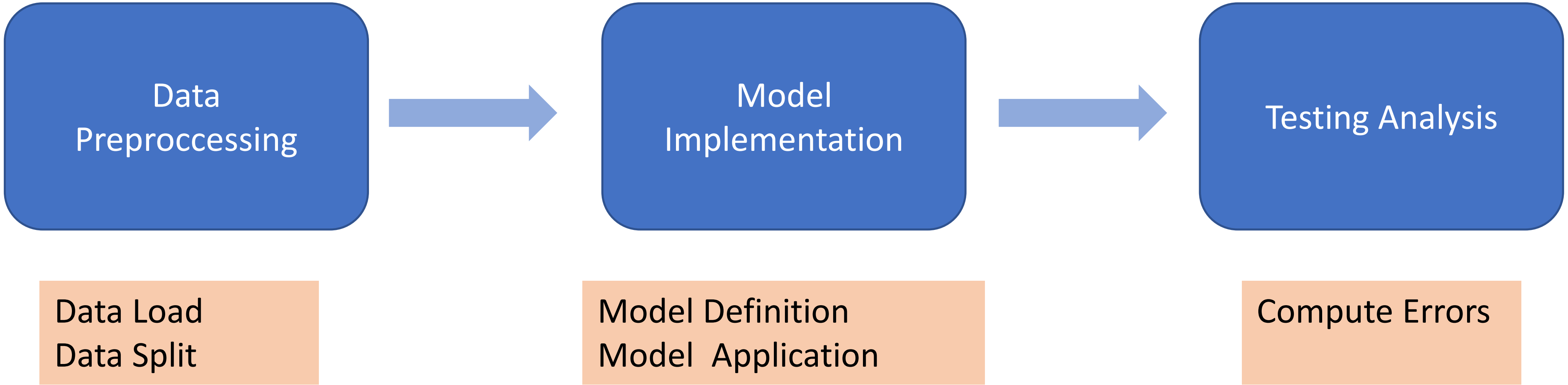
opennn neural networks

- Compared with Python options: Scikit and Tensorflow

scikit learn

TensorFlow

Machine Learning Workflow



Data Preproccessing

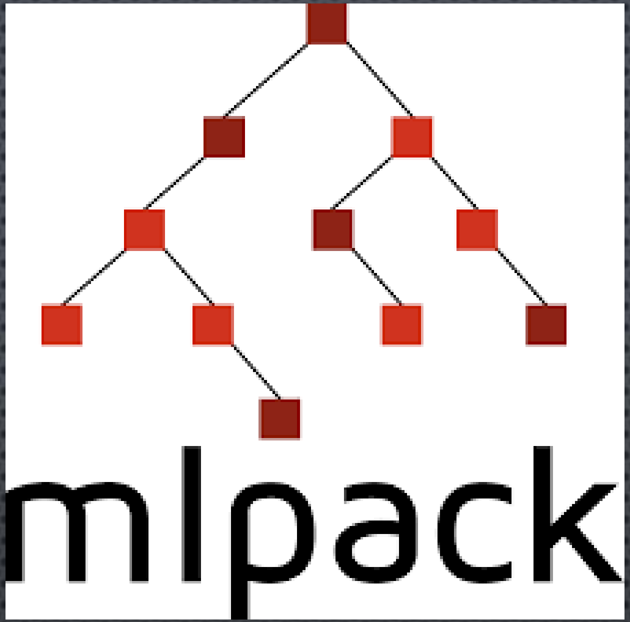
Model Implementation

Testing Analysis

Data Load
Data Split

Model Definition
Model Application

Compute Errors


mlpack

MACHINE
LEARNING

Data Preprocessing

Data stored in Armadillo library Matrix and Vector

Provided data class-> Load and save matrices and models
-> Split function

Data Format ->Column-major matrix
->No header data included
->Converts label to one-hot encoding previously

Model Implementation

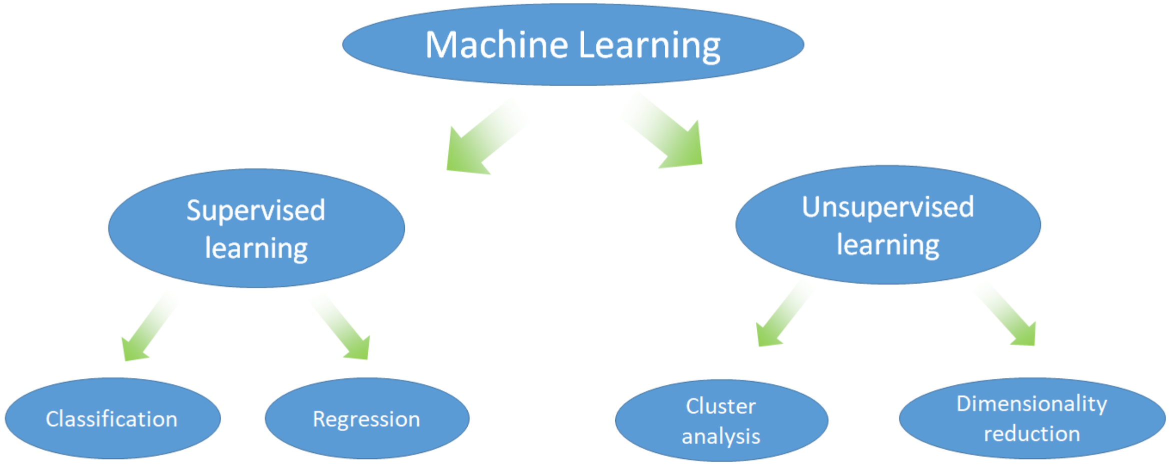
Each algorithm has associated a class

->Instant definition

->Methods to change default parameters

->Train()

->Apply()



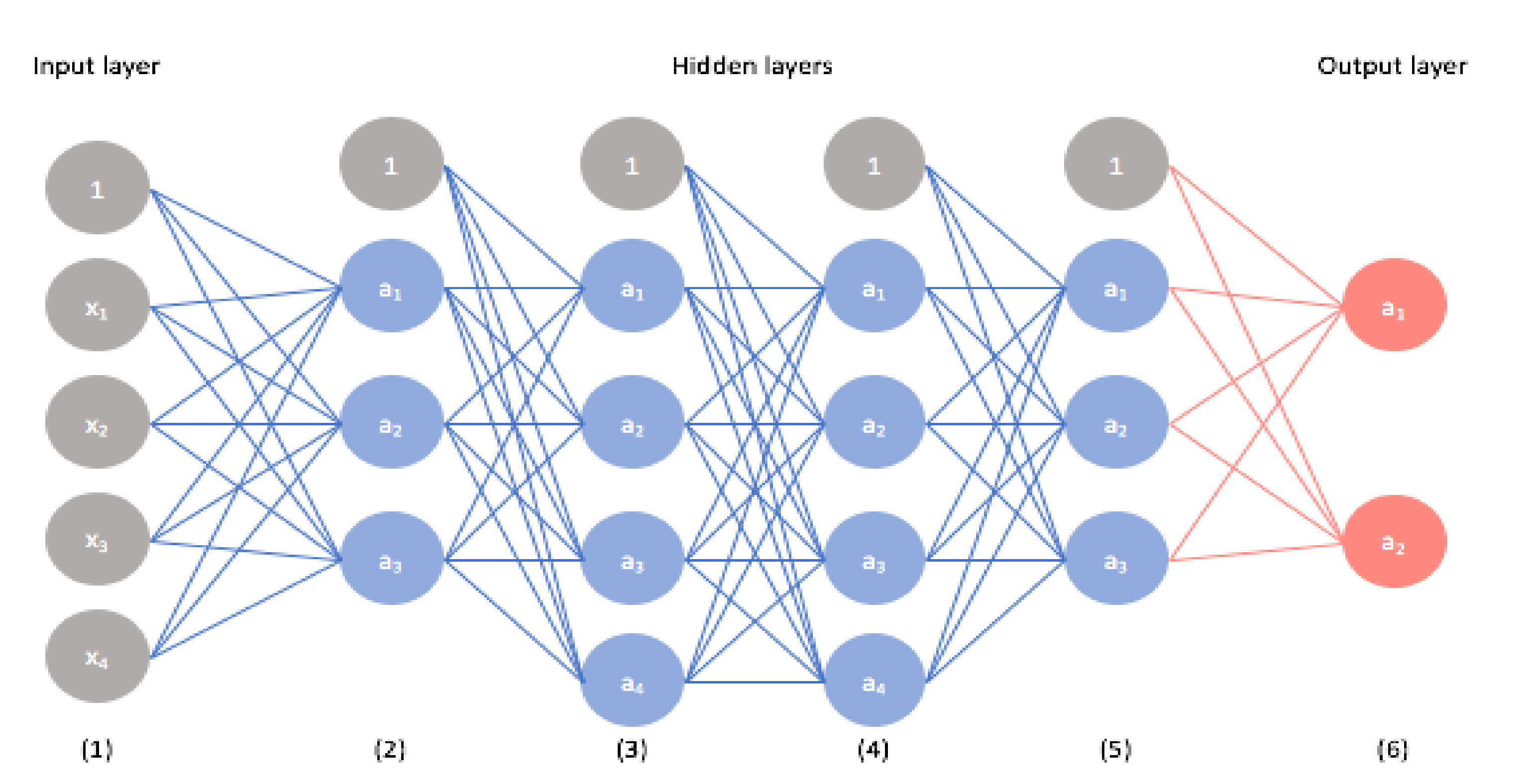
Test Analysis

Each algorithm their function -> ComputeError()

->ComputeAccuraccy()

From data class->ConfusionMatrix()

opennn
neural networks



Data Preprocessing

Data stored in dataset class

- Admits headers
- Specify column uses ->Target,Input or Unused
- Extract scale descriptives-> Mean, Sd, Max and Min
- Split randomly or sequentially->Training,validation and test set

Model Implementation

NeuralNetwork class

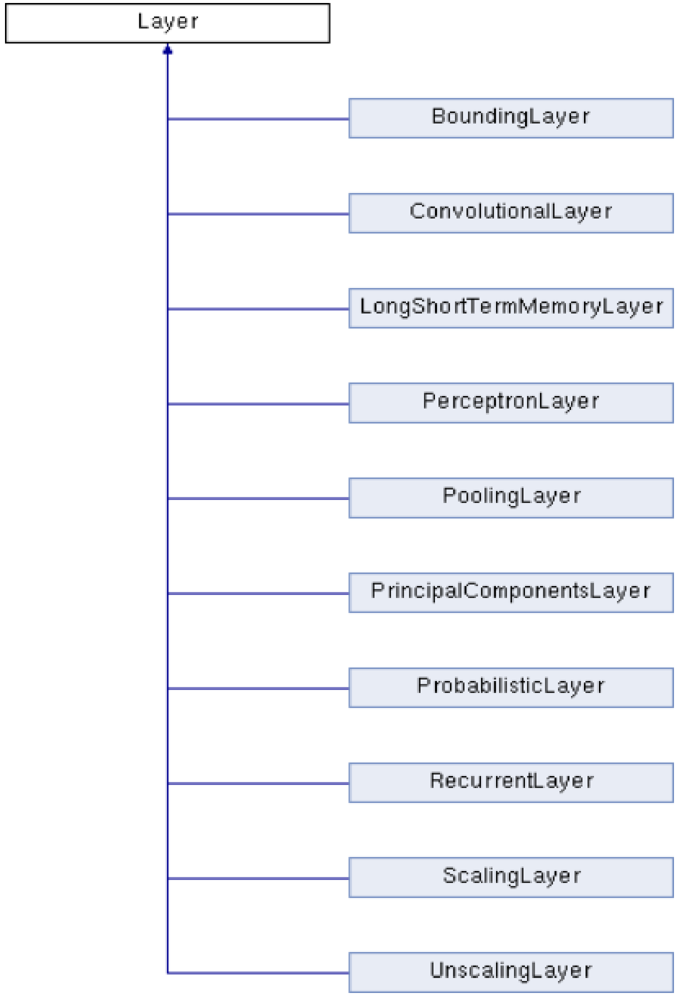
Construction by sequential addition of layers

Predefined models

->Approximation: Perceptron+Linear Activation

->Classification: Perceptron +Softmax Activation

->Forecast: LongShortTermMemory



Model Implementation

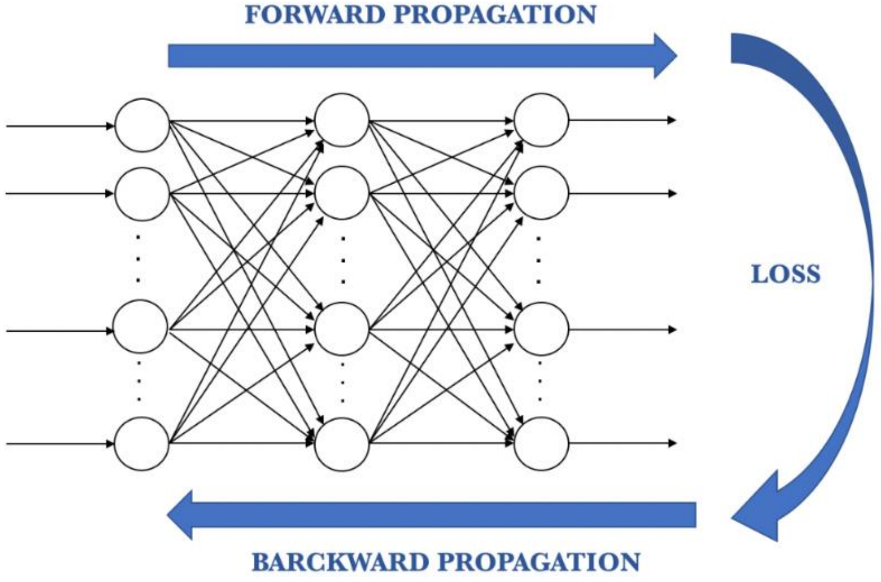
TrainingStrategy class

- LossMethod Class
- OptimizationMethod Class

perform_training()

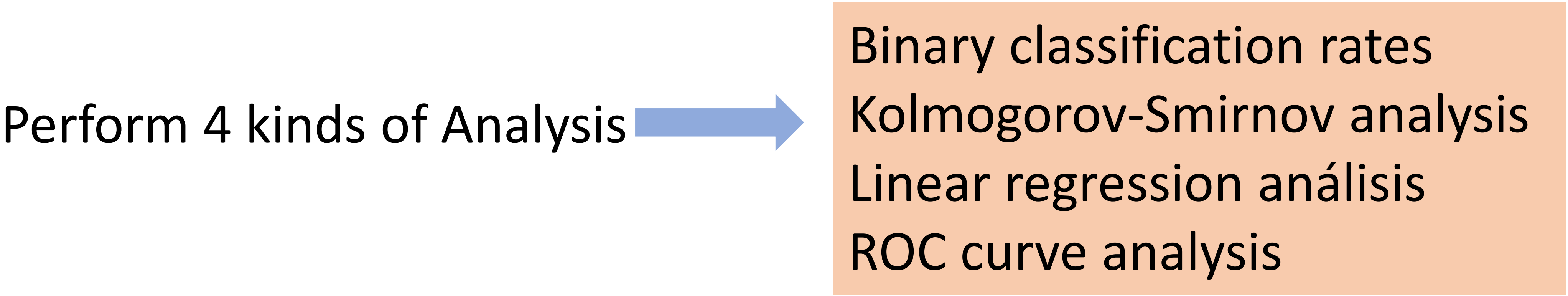
ModelSelection class

- OrderSelectionMethod->Vary neurons
- InputsSelectionMethod->Vary input subset



Test Analysis

TestingAnalysis class



Perform 4 kinds of Analysis

Binary classification rates
Kolmogorov-Smirnov analysis
Linear regression análisis
ROC curve analysis

Problem: Defined Results.save() function, but not implemented

Testing

mlpack

Regression

Linear Regression + Least-Angle Regression

Simple regression problem

Swedish Auto Insurance

63 observations

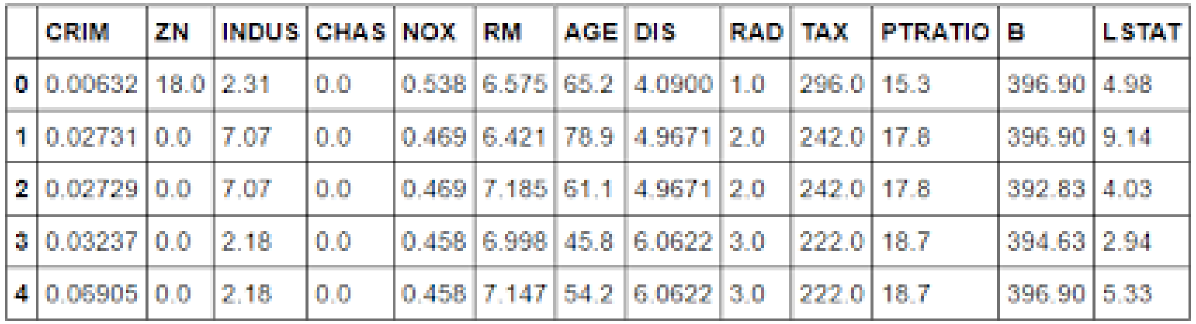
- # Claims
- Total payments

Multivariable regression

Boston House Price

506 observations

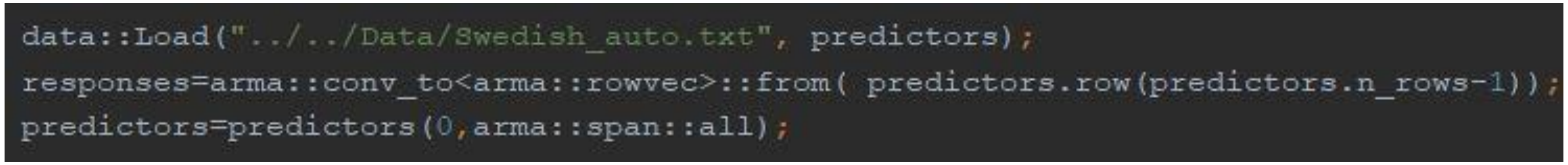
Goal: Predict House Price



	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33

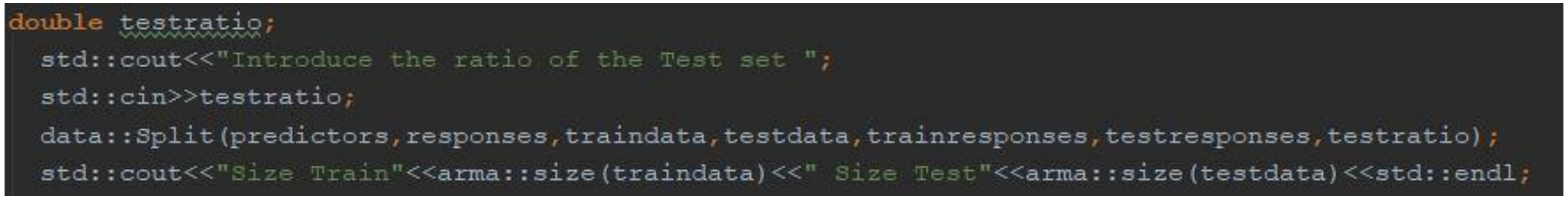
Regression

- Loading



```
data::Load("../../Data/Swedish_auto.txt", predictors);
responses=arma::conv_to<arma::rowvec>::from( predictors.row(predictors.n_rows-1));
predictors=predictors(0,arma::span::all);
```

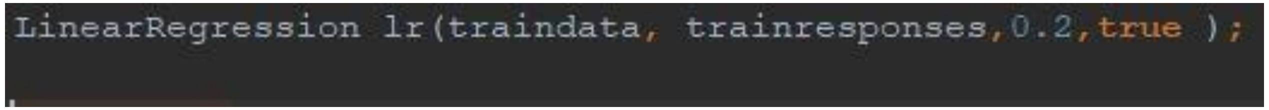
- Splitting



```
double testratio;
  std::cout<<"Introduce the ratio of the Test set ";
  std::cin>>testratio;
  data::Split(predictors,responses,traindata,testdata,trainresponses,testresponses,testratio);
  std::cout<<"Size Train"<<arma::size(traindata)<<" Size Test"<<arma::size(testdata)<<std::endl;
```

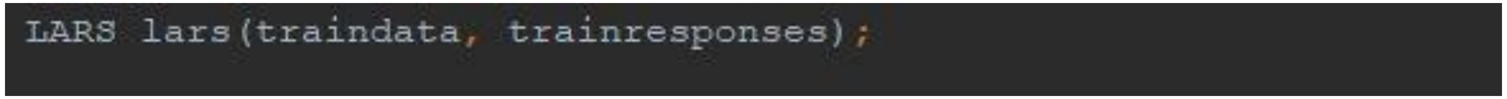
- Algorithm definition

Linear Regressor



```
LinearRegression lr(traindata, trainresponses,0.2,true );
```

Least-Angle Regressor



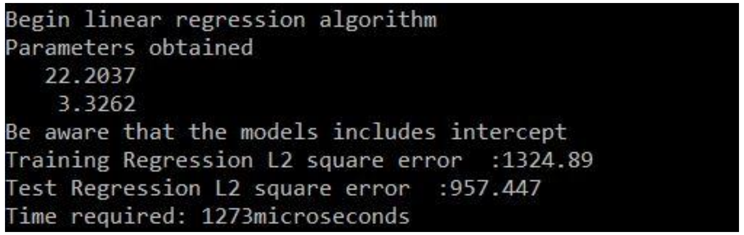
```
LARS lars(traindata, trainresponses);
```

Regression- Results

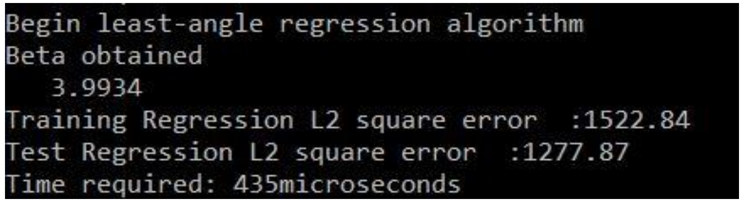
Linear Regression

Least-Angle Regression

Swedish Auto Insurance

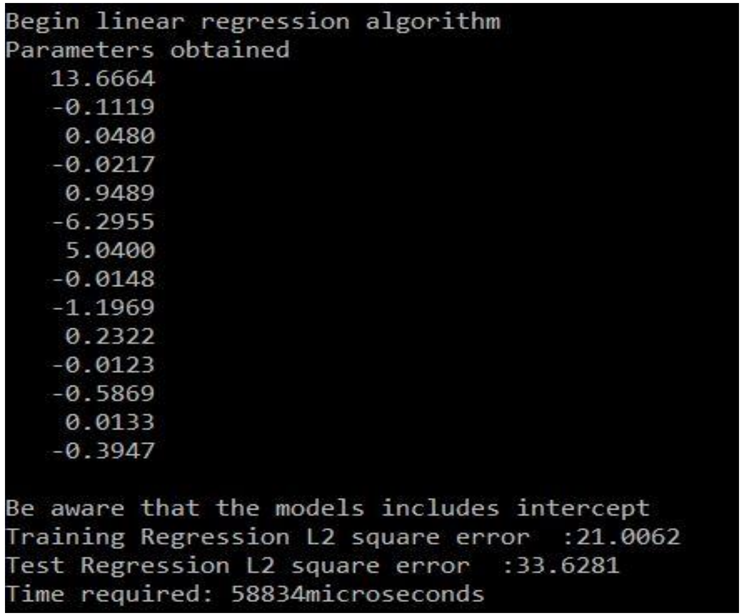


```
Begin linear regression algorithm
Parameters obtained
   22.2037
    3.3262
Be aware that the models includes intercept
Training Regression L2 square error  :1324.89
Test Regression L2 square error  :957.447
Time required: 1273microseconds
```

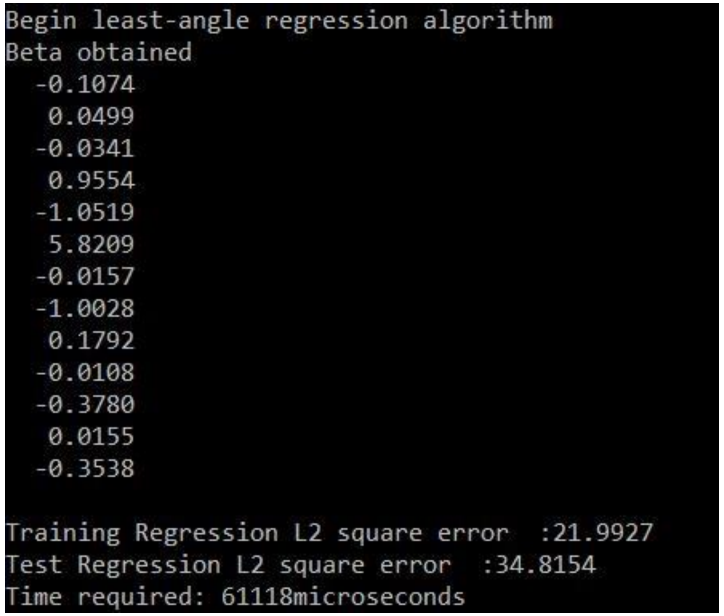


```
Begin least-angle regression algorithm
Beta obtained
   3.9934
Training Regression L2 square error  :1522.84
Test Regression L2 square error  :1277.87
Time required: 435microseconds
```

Boston House Price



```
Begin linear regression algorithm
Parameters obtained
   13.6664
   -0.1119
    0.0480
   -0.0217
    0.9489
   -6.2955
    5.0400
   -0.0148
   -1.1969
    0.2322
   -0.0123
   -0.5869
    0.0133
   -0.3947

Be aware that the models includes intercept
Training Regression L2 square error  :21.0062
Test Regression L2 square error  :33.6281
Time required: 58834microseconds
```



```
Begin least-angle regression algorithm
Beta obtained
  -0.1074
   0.0499
  -0.0341
   0.9554
  -1.0519
   5.8209
  -0.0157
  -1.0028
   0.1792
  -0.0108
  -0.3780
   0.0155
  -0.3538

Training Regression L2 square error  :21.9927
Test Regression L2 square error  :34.8154
Time required: 61118microseconds
```

Clustering + Transformation

Iris dataset

Iris Versicolor

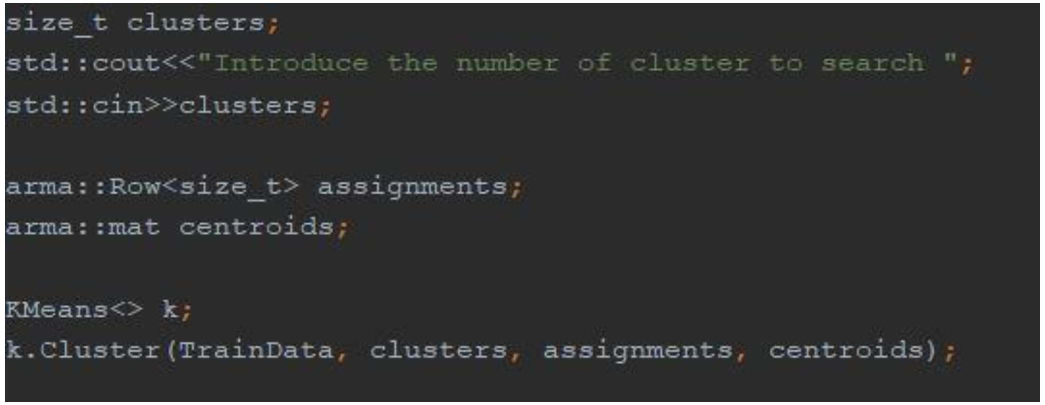
Iris Setosa

Iris Virginica

150 observations

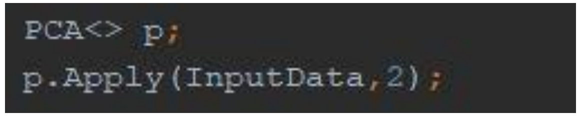
- Sepal length in cm
- Sepal width in cm
- Petal length in cm
- Petal width in cm

K-Means



```
size_t clusters;
std::cout<<"Introduce the number of cluster to search ";
std::cin>>clusters;

arma::Row<size_t> assignments;
arma::mat centroids;

KMeans<> k;
k.Cluster(TrainData, clusters, assignments, centroids);
```

PCA



```
PCA<> p;
p.Apply(InputData,2);
```

Classification

Logistic Regression

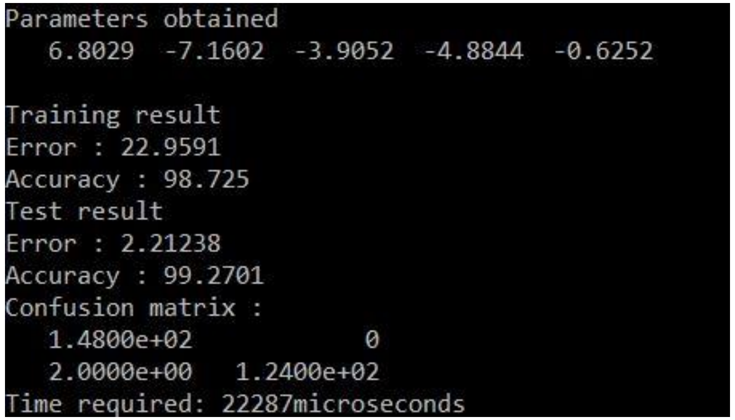
Banknote Authentication

1372 observations

Wavelet Transformed image

- Variance
- Skewness
- Kurtosis
- Entropy

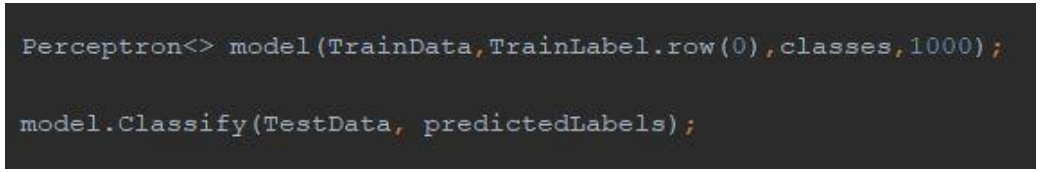
Result



```
Parameters obtained
   6.8029   -7.1602   -3.9052   -4.8844   -0.6252

Training result
Error : 22.9591
Accuracy : 98.725
Test result
Error : 2.21238
Accuracy : 99.2701
Confusion matrix :
   1.4800e+02            0
   2.0000e+00   1.2400e+02
Time required: 22287microseconds
```

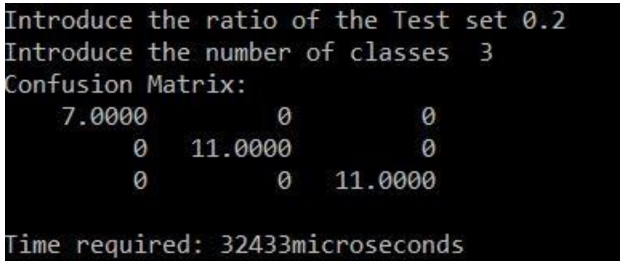
Perceptron

Iris dataset



```
Perceptron<> model(TrainData,TrainLabel.row(0),classes,1000);

model.Classify(TestData, predictedLabels);
```

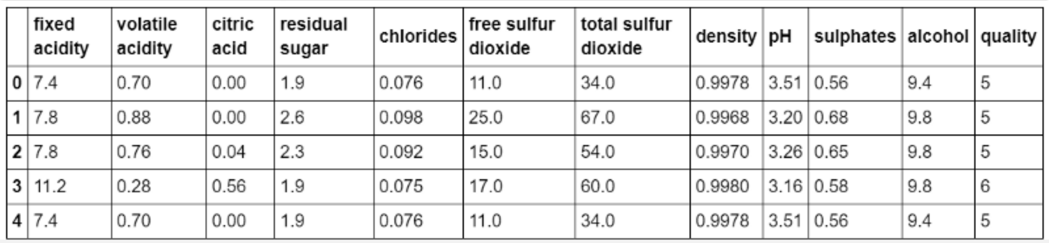
Result



```
Introduce the ratio of the Test set 0.2
Introduce the number of classes  3
Confusion Matrix:
    7.0000         0         0
         0   11.0000         0
         0         0   11.0000

Time required: 32433microseconds
```

Classification

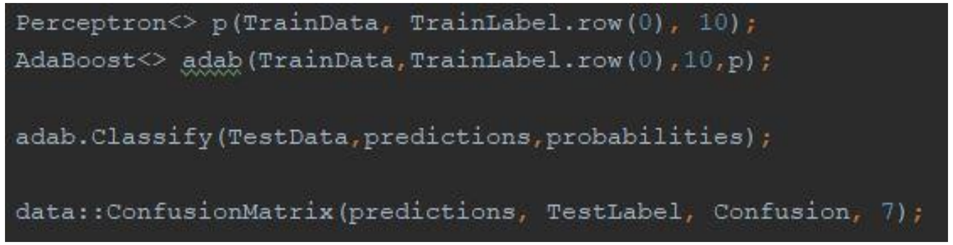
Wine Quality



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

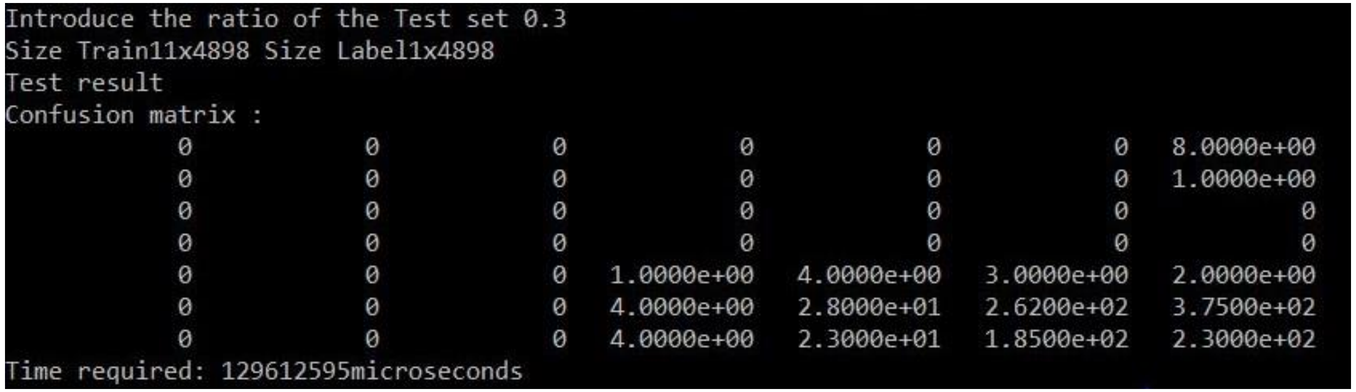
Adaboost

Perceptron/Decision tree



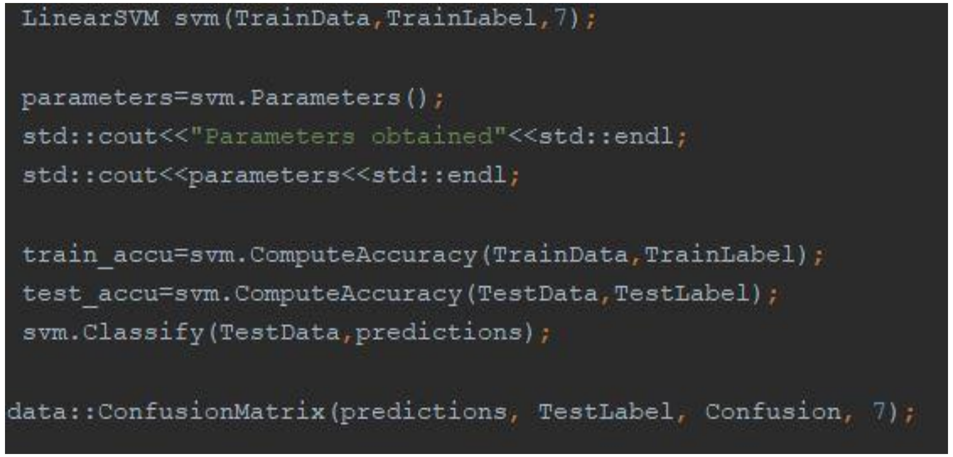
```
Perceptron<> p(TrainData, TrainLabel.row(0), 10);
AdaBoost<> adab(TrainData,TrainLabel.row(0),10,p);

adab.Classify(TestData,predictions,probabilities);

data::ConfusionMatrix(predictions, TestLabel, Confusion, 7);
```

Result



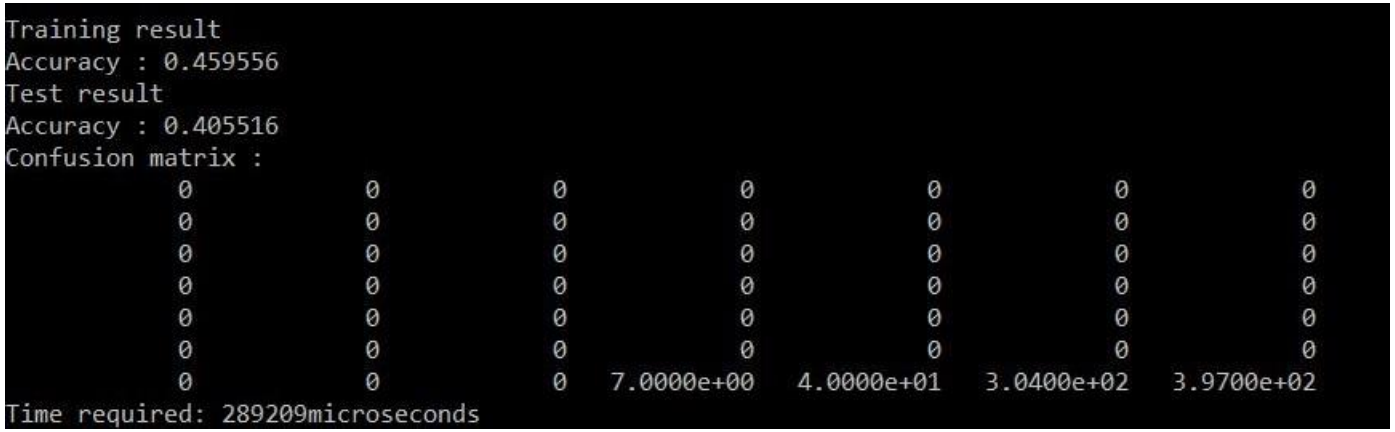
```
Introduce the ratio of the Test set 0.3
Size Train11x4898 Size Label1x4898
Test result
Confusion matrix :
           0           0           0           0           0           0  8.0000e+00
           0           0           0           0           0           0  1.0000e+00
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0  1.0000e+00  4.0000e+00  3.0000e+00  2.0000e+00
           0           0           0  4.0000e+00  2.8000e+01  2.6200e+02  3.7500e+02
           0           0           0  4.0000e+00  2.3000e+01  1.8500e+02  2.3000e+02
Time required: 129612595microseconds
```

SVM



```
LinearSVM svm(TrainData,TrainLabel,7);

parameters=svm.Parameters();
std::cout<<"Parameters obtained"<<std::endl;
std::cout<<parameters<<std::endl;

train_accu=svm.ComputeAccuracy(TrainData,TrainLabel);
test_accu=svm.ComputeAccuracy(TestData,TestLabel);
svm.Classify(TestData,predictions);

data::ConfusionMatrix(predictions, TestLabel, Confusion, 7);
```

Result



```
Training result
Accuracy : 0.459556
Test result
Accuracy : 0.405516
Confusion matrix :
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0           0           0           0           0
           0           0           0  7.0000e+00  4.0000e+01  3.0400e+02  3.9700e+02
Time required: 289209microseconds
```

Comparison

Both are class oriented

Multivariate Regression	Test error	Time elapsed
mlpack	33,63$	58834ms
Scikit-learn	16,57$	578ms

Logisctic Regression	Test accuracy	Time elapsed	Confusion matrix
mlpack	99,27%	22287ms	148 0 2 124
Scikit-learn	99,18%	440ms	146 2 2 124

SVM	Test accuaracy	Time elapsed
mlpack	40,55%	58834ms
Scikit-learn	45,11%	2138ms

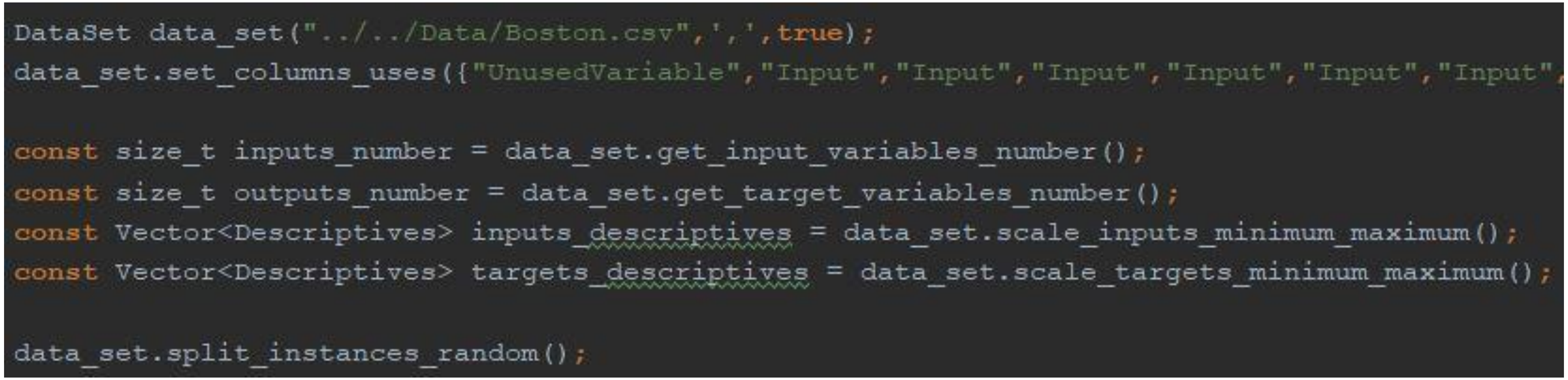
Testing

opennn
neural networks

Multi-variate Regression

Boston House Price

- Data Preprocessing

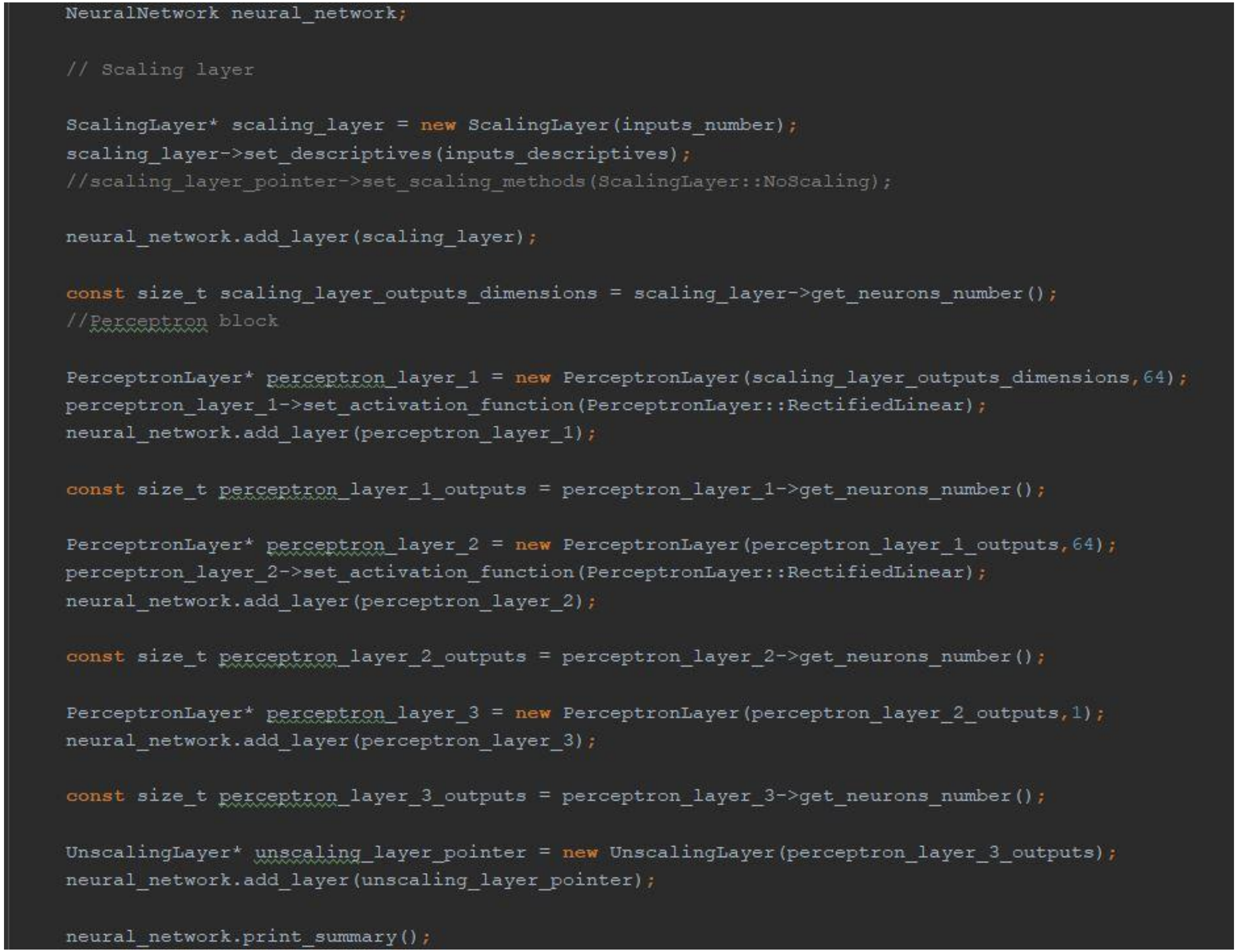


```
DataSet data_set("../../Data/Boston.csv",',',true);
data_set.set_columns_uses({"UnusedVariable","Input","Input","Input","Input","Input","Input",

const size_t inputs_number = data_set.get_input_variables_number();
const size_t outputs_number = data_set.get_target_variables_number();
const Vector<Descriptives> inputs_descriptives = data_set.scale_inputs_minimum_maximum();
const Vector<Descriptives> targets_descriptives = data_set.scale_targets_minimum_maximum();

data_set.split_instances_random();
```

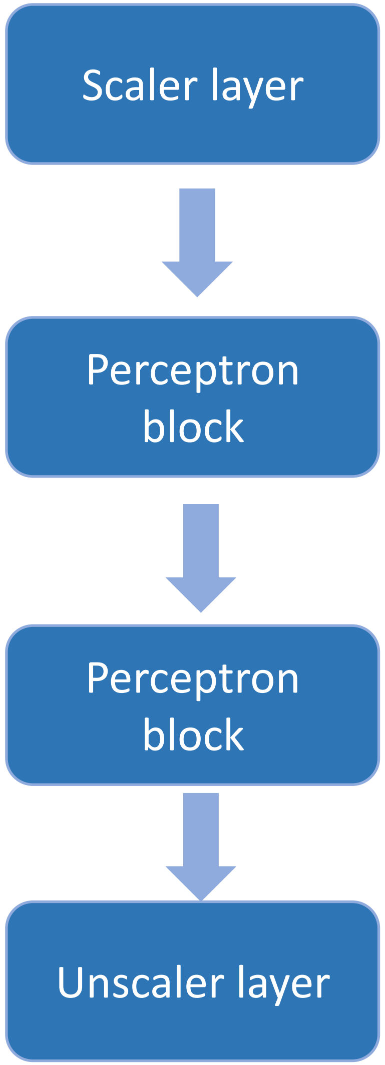
Multi-variate Regression

Boston House Price

- Model



```
NeuralNetwork neural_network;

// Scaling layer

ScalingLayer* scaling_layer = new ScalingLayer(inputs_number);
scaling_layer->set_descriptives(inputs_descriptives);
//scaling_layer_pointer->set_scaling_methods(ScalingLayer::NoScaling);

neural_network.add_layer(scaling_layer);

const size_t scaling_layer_outputs_dimensions = scaling_layer->get_neurons_number();
//Perceptron block

PerceptronLayer* perceptron_layer_1 = new PerceptronLayer(scaling_layer_outputs_dimensions,64);
perceptron_layer_1->set_activation_function(PerceptronLayer::RectifiedLinear);
neural_network.add_layer(perceptron_layer_1);

const size_t perceptron_layer_1_outputs = perceptron_layer_1->get_neurons_number();

PerceptronLayer* perceptron_layer_2 = new PerceptronLayer(perceptron_layer_1_outputs,64);
perceptron_layer_2->set_activation_function(PerceptronLayer::RectifiedLinear);
neural_network.add_layer(perceptron_layer_2);

const size_t perceptron_layer_2_outputs = perceptron_layer_2->get_neurons_number();

PerceptronLayer* perceptron_layer_3 = new PerceptronLayer(perceptron_layer_2_outputs,1);
neural_network.add_layer(perceptron_layer_3);

const size_t perceptron_layer_3_outputs = perceptron_layer_3->get_neurons_number();

UnscalingLayer* unscaling_layer_pointer = new UnscalingLayer(perceptron_layer_3_outputs);
neural_network.add_layer(unscaling_layer_pointer);

neural_network.print_summary();
```



Scaler layer

Perceptron block

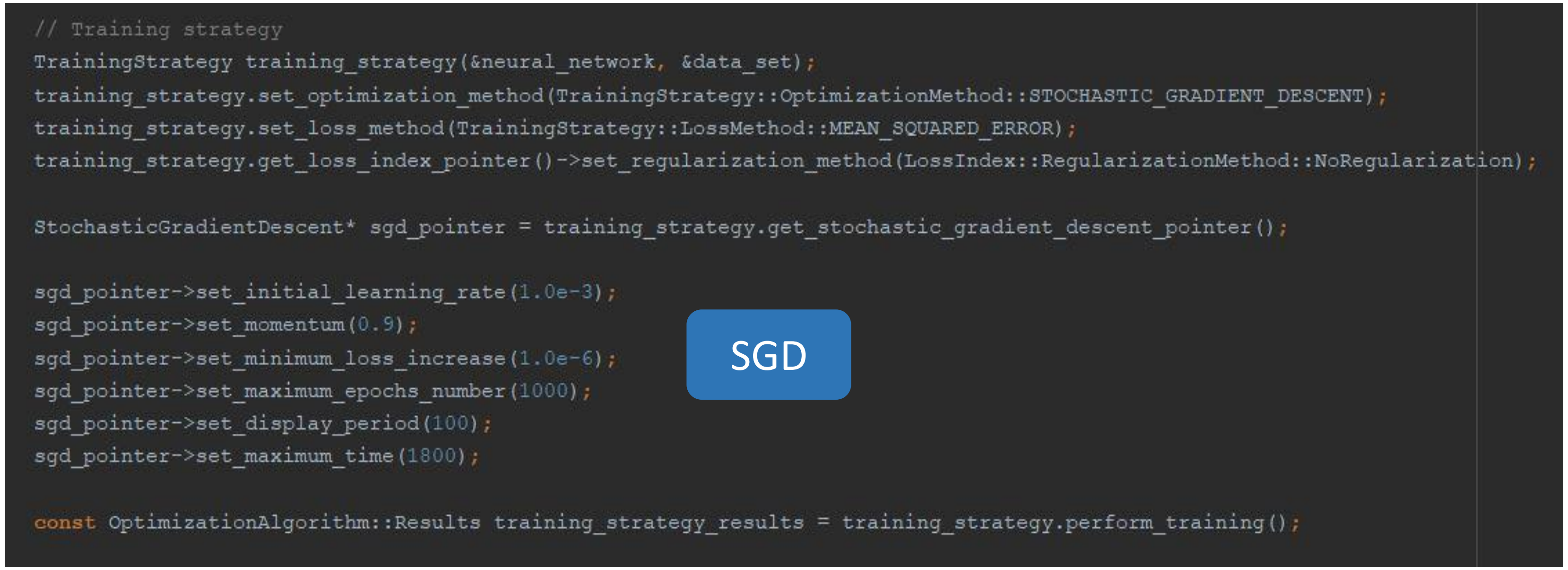
Perceptron block

Unscaler layer

Multi-variate Regression

Boston House Price

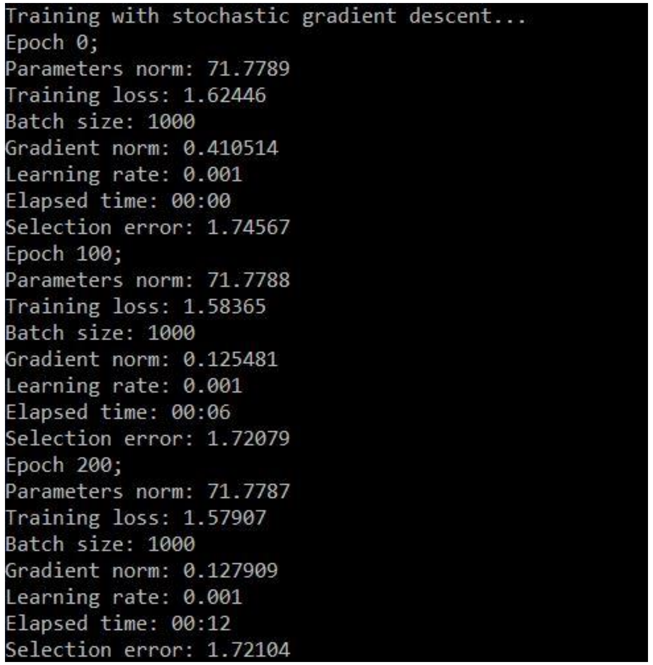
- Training Strategy



```
// Training strategy
TrainingStrategy training_strategy(&neural_network, &data_set);
training_strategy.set_optimization_method(TrainingStrategy::OptimizationMethod::STOCHASTIC_GRADIENT_DESCENT);
training_strategy.set_loss_method(TrainingStrategy::LossMethod::MEAN_SQUARED_ERROR);
training_strategy.get_loss_index_pointer()->set_regularization_method(LossIndex::RegularizationMethod::NoRegularization);

StochasticGradientDescent* sgd_pointer = training_strategy.get_stochastic_gradient_descent_pointer();

sgd_pointer->set_initial_learning_rate(1.0e-3);
sgd_pointer->set_momentum(0.9);
sgd_pointer->set_minimum_loss_increase(1.0e-6);
sgd_pointer->set_maximum_epochs_number(1000);
sgd_pointer->set_display_period(100);
sgd_pointer->set_maximum_time(1800);

const OptimizationAlgorithm::Results training_strategy_results = training_strategy.perform_training();
```

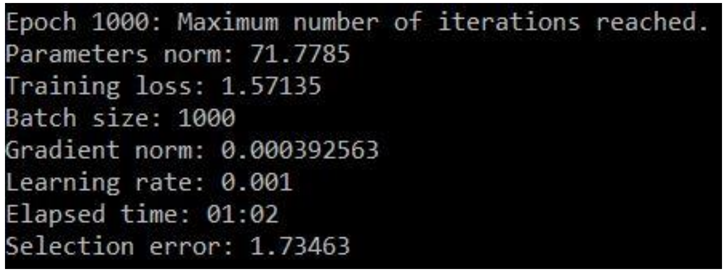
SGD

Training



```
Training with stochastic gradient descent...
Epoch 0;
Parameters norm: 71.7789
Training loss: 1.62446
Batch size: 1000
Gradient norm: 0.410514
Learning rate: 0.001
Elapsed time: 00:00
Selection error: 1.74567
Epoch 100;
Parameters norm: 71.7788
Training loss: 1.58365
Batch size: 1000
Gradient norm: 0.125481
Learning rate: 0.001
Elapsed time: 00:06
Selection error: 1.72079
Epoch 200;
Parameters norm: 71.7787
Training loss: 1.57907
Batch size: 1000
Gradient norm: 0.127909
Learning rate: 0.001
Elapsed time: 00:12
Selection error: 1.72104
```

Result

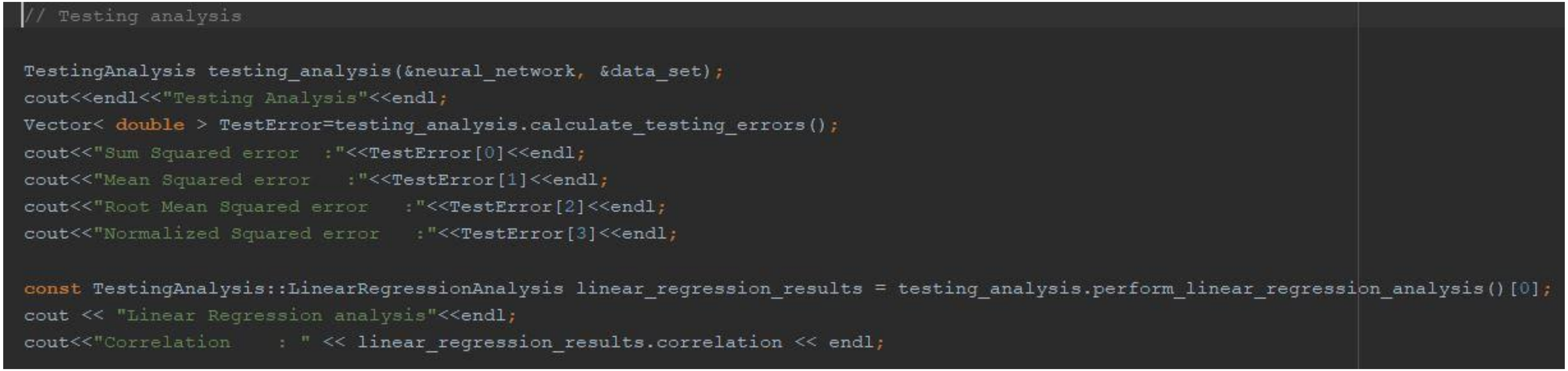


```
Epoch 1000: Maximum number of iterations reached.
Parameters norm: 71.7785
Training loss: 1.57135
Batch size: 1000
Gradient norm: 0.000392563
Learning rate: 0.001
Elapsed time: 01:02
Selection error: 1.73463
```

Multi-variate Regression

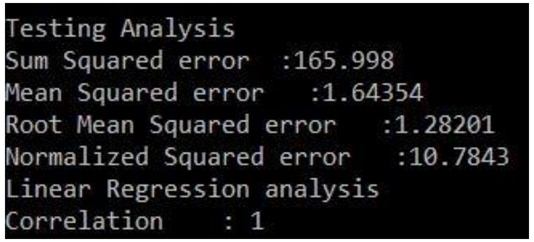
Boston House Price

- Testing Analysis



```
// Testing analysis

TestingAnalysis testing_analysis(&neural_network, &data_set);
cout<<endl<<"Testing Analysis"<<endl;
Vector< double > TestError=testing_analysis.calculate_testing_errors();
cout<<"Sum Squared error  :"<<TestError[0]<<endl;
cout<<"Mean Squared error    :"<<TestError[1]<<endl;
cout<<"Root Mean Squared error    :"<<TestError[2]<<endl;
cout<<"Normalized Squared error    :"<<TestError[3]<<endl;

const TestingAnalysis::LinearRegressionAnalysis linear_regression_results = testing_analysis.perform_linear_regression_analysis()[0];
cout << "Linear Regression analysis"<<endl;
cout<<"Correlation     : " << linear_regression_results.correlation << endl;
```

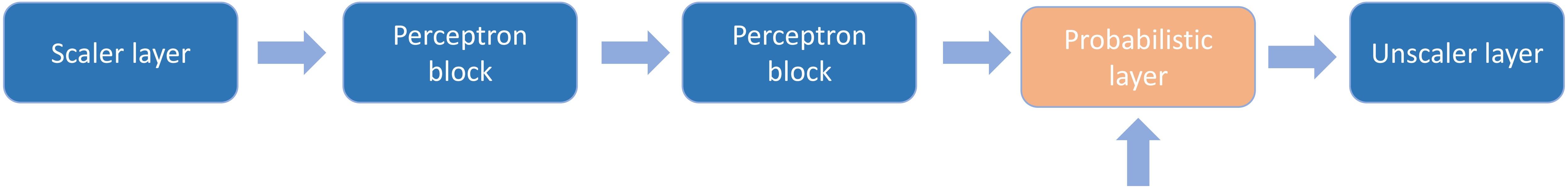
Result

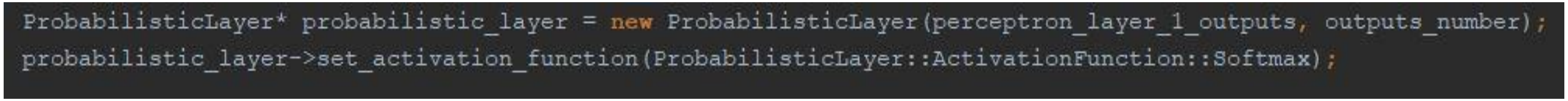


```
Testing Analysis
Sum Squared error  :165.998
Mean Squared error    :1.64354
Root Mean Squared error    :1.28201
Normalized Squared error    :10.7843
Linear Regression analysis
Correlation     : 1
```

Multi-Classification

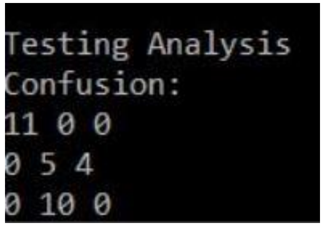
Iris dataset





```
ProbabilisticLayer* probabilistic_layer = new ProbabilisticLayer(perceptron_layer_1_outputs, outputs_number);
probabilistic_layer->set_activation_function(ProbabilisticLayer::ActivationFunction::Softmax);
```

Result

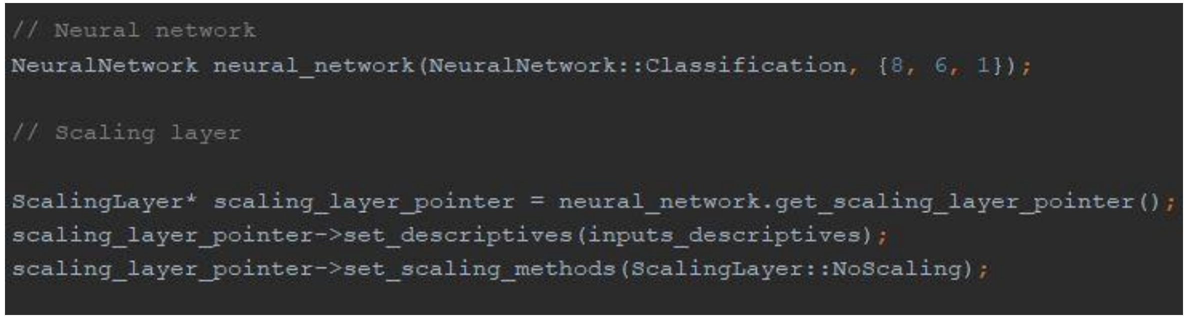


```
Testing Analysis
Confusion:
11 0 0
0 5 4
0 10 0
```

Binary Classification

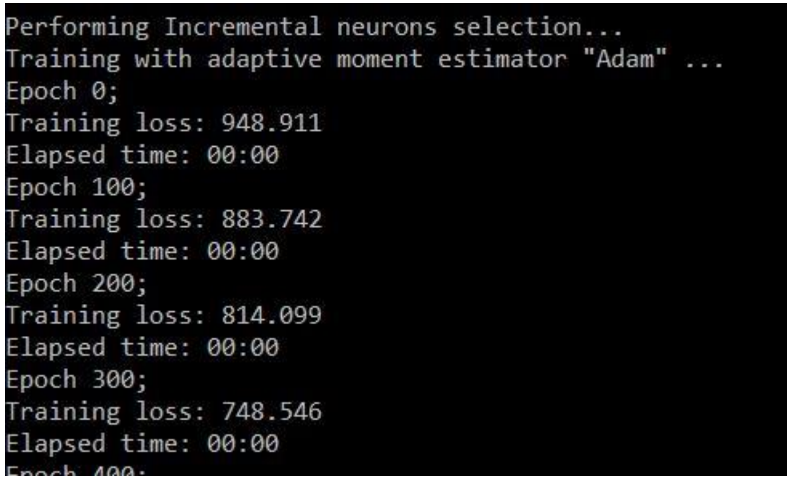
Banknote Authentication

- Pre-Defined Model



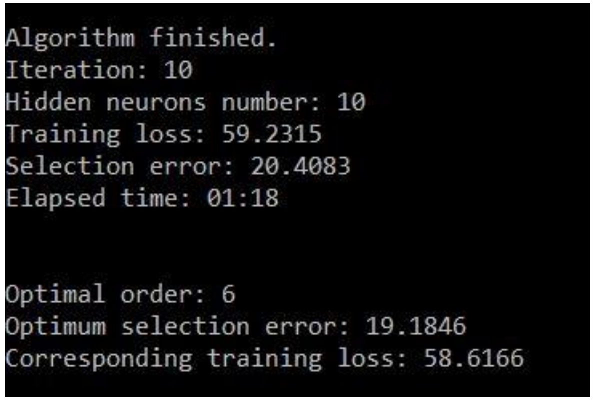
```
// Neural network
NeuralNetwork neural_network(NeuralNetwork::Classification, {8, 6, 1});

// Scaling layer

ScalingLayer* scaling_layer_pointer = neural_network.get_scaling_layer_pointer();
scaling_layer_pointer->set_descriptives(inputs_descriptives);
scaling_layer_pointer->set_scaling_methods(ScalingLayer::NoScaling);
```

- ModelSelection Class



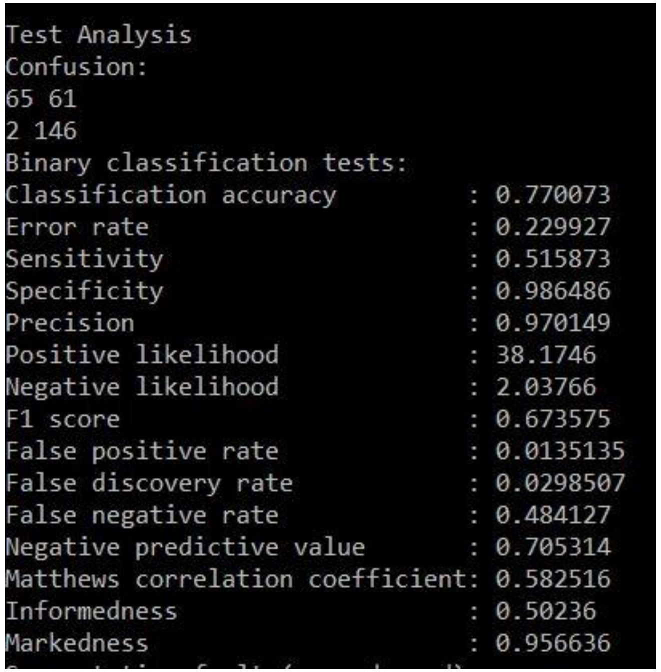
```
Performing Incremental neurons selection...
Training with adaptive moment estimator "Adam" ...
Epoch 0;
Training loss: 948.911
Elapsed time: 00:00
Epoch 100;
Training loss: 883.742
Elapsed time: 00:00
Epoch 200;
Training loss: 814.099
Elapsed time: 00:00
Epoch 300;
Training loss: 748.546
Elapsed time: 00:00
Epoch 400;
```

Selection



```
Algorithm finished.
Iteration: 10
Hidden neurons number: 10
Training loss: 59.2315
Selection error: 20.4083
Elapsed time: 01:18


Optimal order: 6
Optimum selection error: 19.1846
Corresponding training loss: 58.6166
```

Result



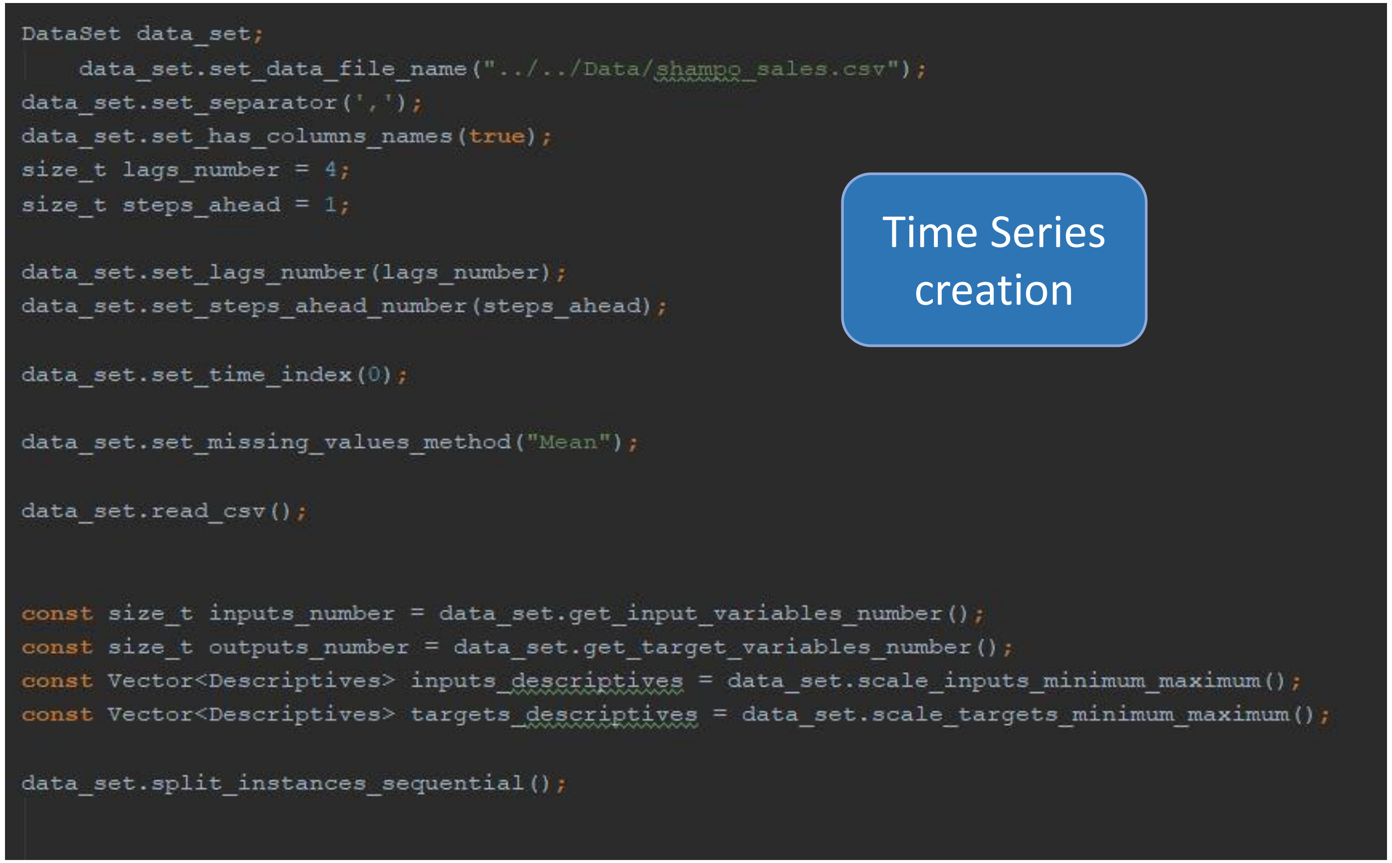
```
Test Analysis
Confusion:
65 61
2 146
Binary classification tests:
Classification accuracy        : 0.770073
Error rate                     : 0.229927
Sensitivity                    : 0.515873
Specificity                    : 0.986486
Precision                      : 0.970149
Positive likelihood            : 38.1746
Negative likelihood            : 2.03766
F1 score                       : 0.673575
False positive rate            : 0.0135135
False discovery rate           : 0.0298507
False negative rate            : 0.484127
Negative predictive value      : 0.705314
Matthews correlation coefficient: 0.582516
Informedness                   : 0.50236
Markedness                     : 0.956636
```

Time Series Forecasting

Shampoo Sales

36 observations

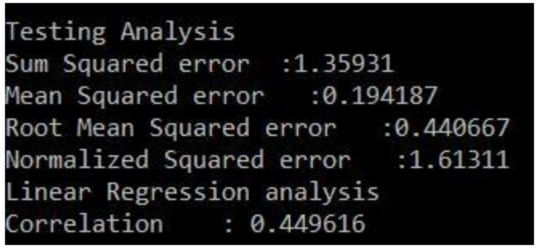
3 years of sales



```
DataSet data_set;
    data_set.set_data_file_name("../../Data/shampo_sales.csv");
data_set.set_separator(',');
data_set.set_has_columns_names(true);
size_t lags_number = 4;
size_t steps_ahead = 1;

data_set.set_lags_number(lags_number);
data_set.set_steps_ahead_number(steps_ahead);

data_set.set_time_index(0);

data_set.set_missing_values_method("Mean");

data_set.read_csv();


const size_t inputs_number = data_set.get_input_variables_number();
const size_t outputs_number = data_set.get_target_variables_number();
const Vector<Descriptives> inputs_descriptives = data_set.scale_inputs_minimum_maximum();
const Vector<Descriptives> targets_descriptives = data_set.scale_targets_minimum_maximum();

data_set.split_instances_sequential();
```

Time Series creation

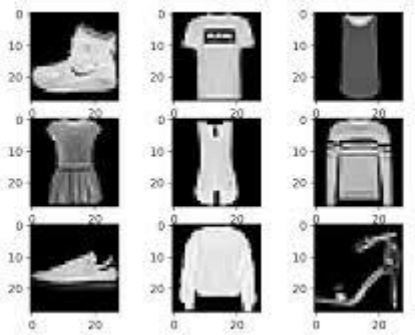
Result

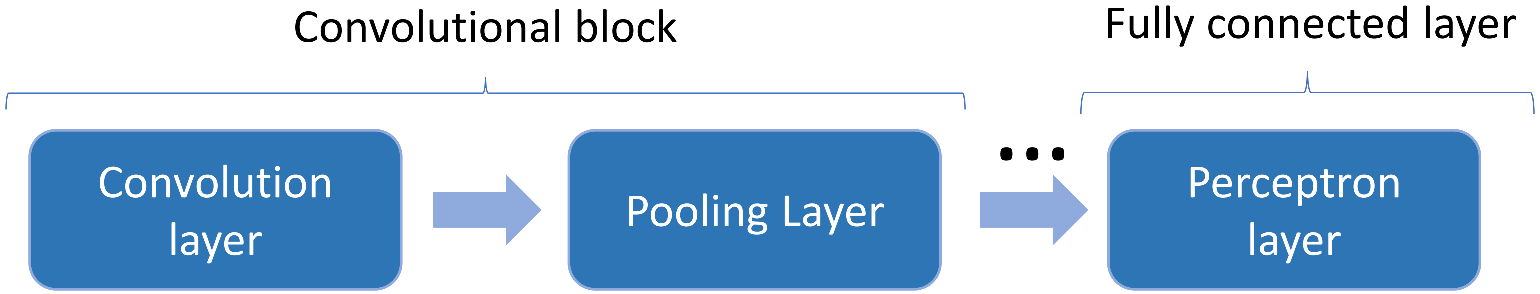


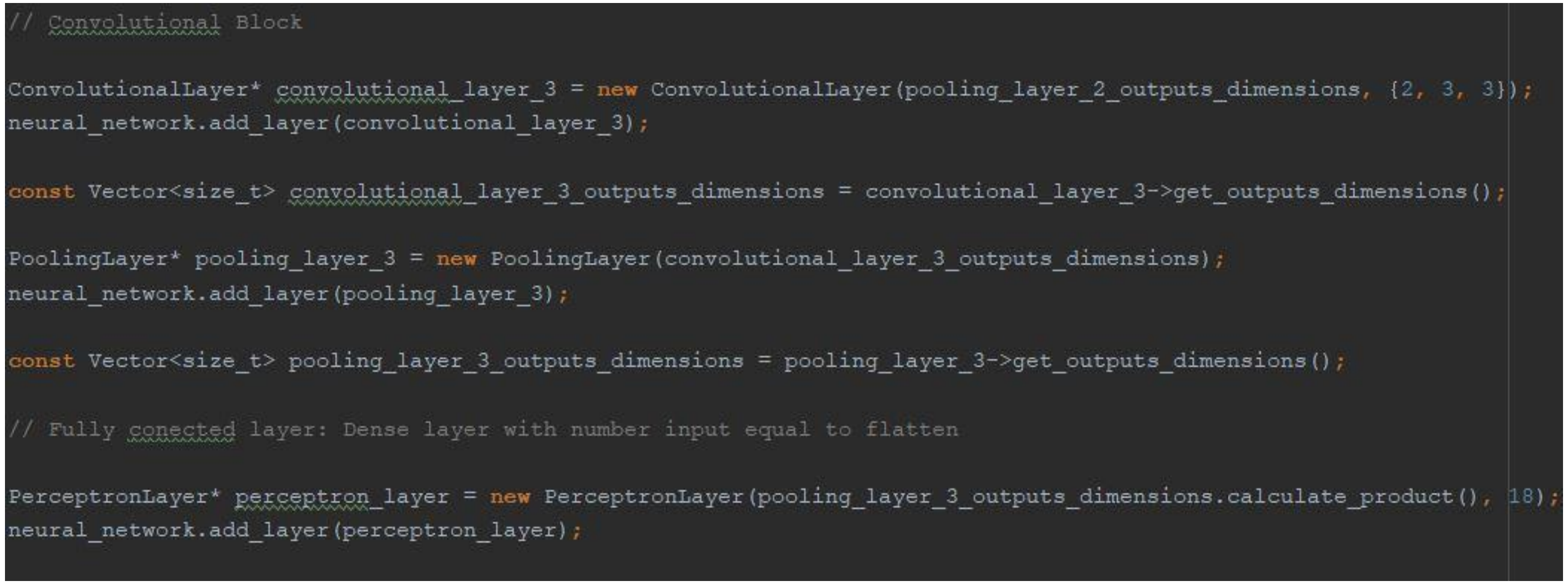
```
Testing Analysis
Sum Squared error   :1.35931
Mean Squared error    :0.194187
Root Mean Squared error    :0.440667
Normalized Squared error    :1.61311
Linear Regression analysis
Correlation     : 0.449616
```

Image Classification

Fashion M-NIST

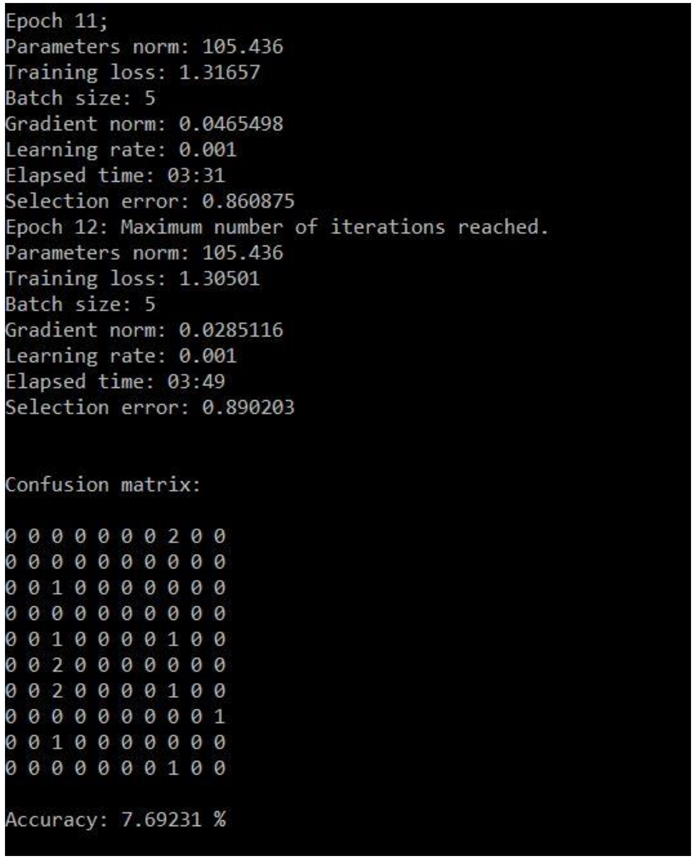






```
// Convolutional Block

ConvolutionalLayer* convolutional_layer_3 = new ConvolutionalLayer(pooling_layer_2_outputs_dimensions, {2, 3, 3});
neural_network.add_layer(convolutional_layer_3);

const Vector<size_t> convolutional_layer_3_outputs_dimensions = convolutional_layer_3->get_outputs_dimensions();

PoolingLayer* pooling_layer_3 = new PoolingLayer(convolutional_layer_3_outputs_dimensions);
neural_network.add_layer(pooling_layer_3);

const Vector<size_t> pooling_layer_3_outputs_dimensions = pooling_layer_3->get_outputs_dimensions();

// Fully conected layer: Dense layer with number input equal to flatten

PerceptronLayer* perceptron_layer = new PerceptronLayer(pooling_layer_3_outputs_dimensions.calculate_product(), 18);
neural_network.add_layer(perceptron_layer);
```

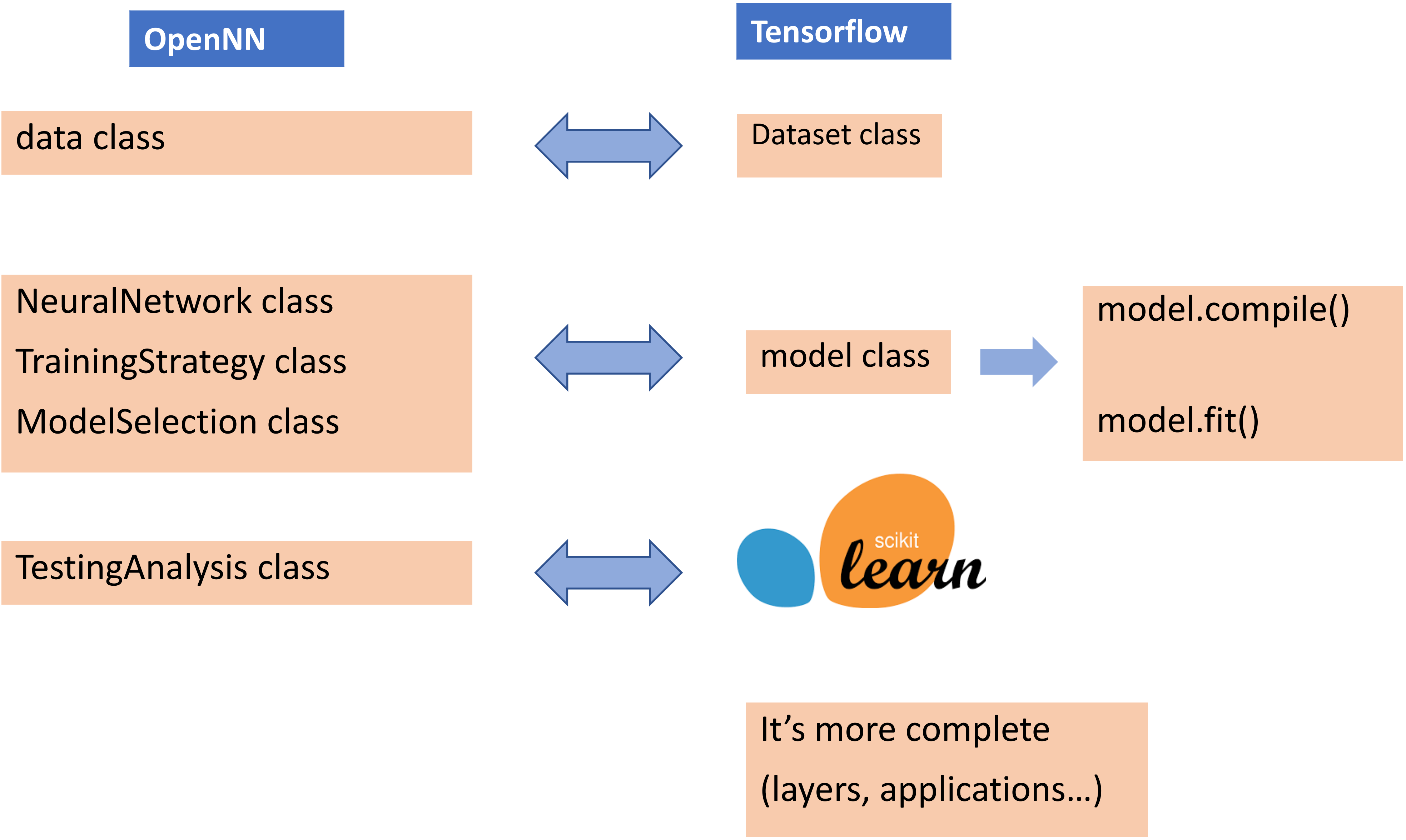
Result



```
Epoch 11;
Parameters norm: 105.436
Training loss: 1.31657
Batch size: 5
Gradient norm: 0.0465498
Learning rate: 0.001
Elapsed time: 03:31
Selection error: 0.860875
Epoch 12: Maximum number of iterations reached.
Parameters norm: 105.436
Training loss: 1.30501
Batch size: 5
Gradient norm: 0.0285116
Learning rate: 0.001
Elapsed time: 03:49
Selection error: 0.890203


Confusion matrix:

0 0 0 0 0 0 0 2 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0
0 0 2 0 0 0 0 0 0 0
0 0 2 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 1
0 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0

Accuracy: 7.69231 %
```

Problem: Data loading -> Time Consuming

Comparison



OpenNN

Tensorflow

data class

Dataset class

NeuralNetwork class

TrainingStrategy class

ModelSelection class

model class

model.compile()

model.fit()

TestingAnalysis class

scikit learn

It's more complete

(layers, applications…)

Test

Multivariate Regression	Test error/R2 score	Time elapsed
OpenNN	1,64$/1	62s
Tensorflow	11,85$/0,87	14,6s

LSTM	R2 score	Time elapsed
OpenNN	44,96%	75s
Tensorflow	<0	71s

Image Classification	Test accuaracy/data used	Reading time elapsed
OpenNN	7,69%/125	Several minutes
Tensorflow	63,11%/60000	3,47s

Bibliography

- https://machinelearningmastery.com/standard-machine-learning-datasets/
- https://www.analyticsvidhya.com/blog/2018/03/comprehensive-collection-deep-learning-datasets/
- https://machinelearningmastery.com/time-series-datasets-for-machine-learning/
- M. Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.
- L. Artificial Intelligence Techniques. “opennn neural networks”.
- S. Bhardwaj, R. R. Curtin, M. Edel, Y. Mentekidis, and C. Sanderson. *ensmallen:a flexibleC++library for efficient function optimization*.
- R. R. Curtin, M. Edel, M. Lozhnikov, Y. Mentekidis, S. Ghaisas, and S.Zhang. “mlpack 3: a fast, flexible machine learning library”. In: *Journal of Open Source Software* 3 (26 2018), p. 726.
- F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830
- C. Sanderson. “Armadillo C++ Linear Algebra Library”.
- B. Schäling. The boost C++ libraries. XML Press, 2014
- H. "Xiao, K. Rasul, and R.Vollgraf. “"Fashion-MNIST: aNovel Image Dataset for Benchmarking Machine Learning Algorithms"”.