C++ Final Project

Multiple Regression using Machine Learning Algorithms

Christian Both

260750691

McGill University

3450 University Street

Montreal QC, H3A OE8, Canada

Fall 2016

Table of contents

Table of figures

1 Introduction and project description

As final project for the c++ course, a multiple regression analysis is performed on the "Ames Housing dataset" from Dean De Cock, established in 2011[1]. It is a data set which contains 79 features each for a set of 2930 properties. The actual sales prices of each property have been made available for half of the data set. They are included in the file called *train.csv*.

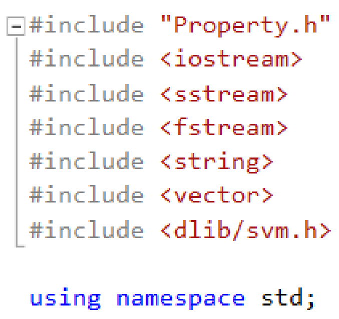
The goal of this project is to predict sales prices written as an executable code in c++ for the second half of property data which is provided as *test.csv*. In order to do so, two regression techniques are implemented by the use of two machine learning algorithms, namely Support Vector Machine (SVM)[2] and Relevance Vector Machine (RVM)[3] which are provided by the dlib c++ machine learning library[4].

Please note that detailed explanation on feature selection and calibration of regression-specific parameters as well as results and future work will be discussed within the presentation given for this project. The purpose of this report is mainly the documentation and implementation of code in c++.

2 Import and conversion of housing dataset

As seen in Figure 1, a class "Property" is created for storing the data set in memory whereas the header file "Property.h" is included in the source code "Regression_project.cpp". The class provides 81 member variables which are exclusively of type integer. This is to store the 79 features, the unique id and the sale price of each property. The complete data sets (train.csv/ test.csv) are stored as a vector consisting of elements of type "Property" (vector<Property> trainSet, vector<Property> testSet). The header file of the used library for regression is added by including <dlib/svm.h>.

[1] https://ww2.amstat.org/publications/jse/v19n3/decock.pdf
[2] http://dlib.net/svr_ex.cpp.html
[3] http://dlib.net/rvm_regression_ex.cpp.html
[4] http://dlib.net/ml.html



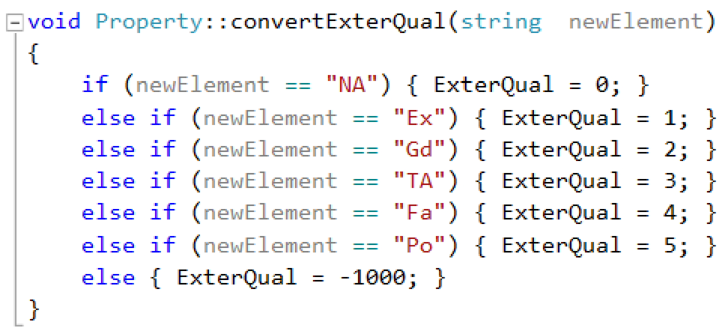
```cpp
#include "Property.h"
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <vector>
#include <dlib/svm.h>

using namespace std;
```

Figure 1 Includes in source code “Regression_project.cpp”

The housing dataset consists of four different types of variables (nominal, ordinal, discrete and continuous) which are converted to an integer type by conversion functions each, which are member functions of the class “Property”. The conversion methods are implemented as follows:

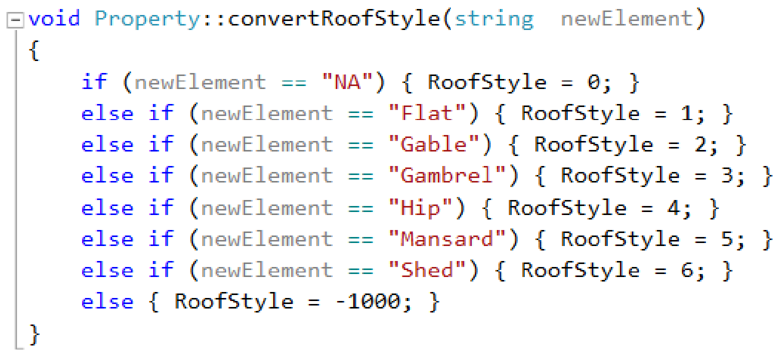
- Discrete/ continuous features are converted to integer variables by use of the “Stoi Function”
- Ordinal values are converted to integer variables by member function in increasing/ decreasing order
- Nominal values are converted to integer variables by member function in random order

As an example, conversion functions for one ordinal and one nominal feature are provided in Figure 2 and Figure 3. It can be noticed that exceptions are caught by an *else* statement in every conversion function by setting the respective variables to -1000 in order to detect typing errors in both, the conversion functions and the data set. By writing the converted data set as the csv file *converted training data.csv*, the correct import of the data set can be verified.



```cpp
void Property::convertExterQual(string  newElement)
{
    if (newElement == "NA") { ExterQual = 0; }
    else if (newElement == "Ex") { ExterQual = 1; }
    else if (newElement == "Gd") { ExterQual = 2; }
    else if (newElement == "TA") { ExterQual = 3; }
    else if (newElement == "Fa") { ExterQual = 4; }
    else if (newElement == "Po") { ExterQual = 5; }
    else { ExterQual = -1000; }
}
```

Figure 2 Conversion of feature “External Quality”

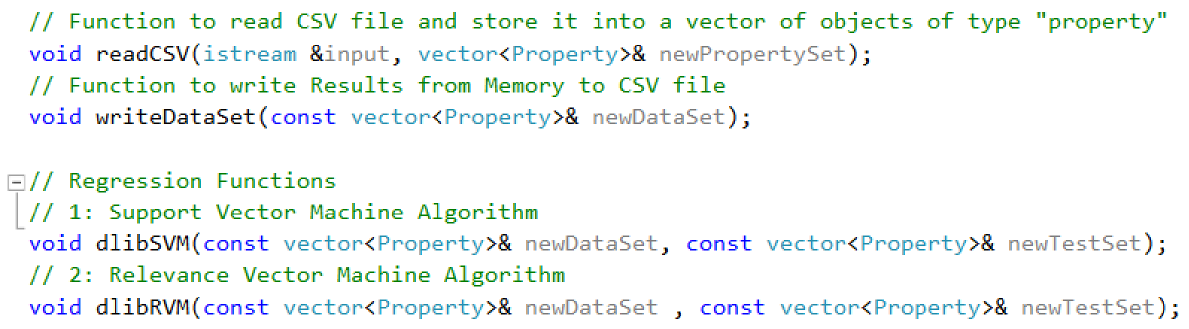


```cpp
void Property::convertRoofStyle(string  newElement)
{
    if (newElement == "NA") { RoofStyle = 0; }
    else if (newElement == "Flat") { RoofStyle = 1; }
    else if (newElement == "Gable") { RoofStyle = 2; }
    else if (newElement == "Gambrel") { RoofStyle = 3; }
    else if (newElement == "Hip") { RoofStyle = 4; }
    else if (newElement == "Mansard") { RoofStyle = 5; }
    else if (newElement == "Shed") { RoofStyle = 6; }
    else { RoofStyle = -1000; }
}
```

Figure 3 Conversion of feature “Roof Style”

3 Functions used in the code

Figure 4 shows the functions including their arguments which are used in the source code. They are used to read the .csv files, writing the converted data set in a new .csv file and performing two types of regression algorithms, namely Support Vector Machine and Relevance Vector Machine. The initiation of functions is discussed hereafter.

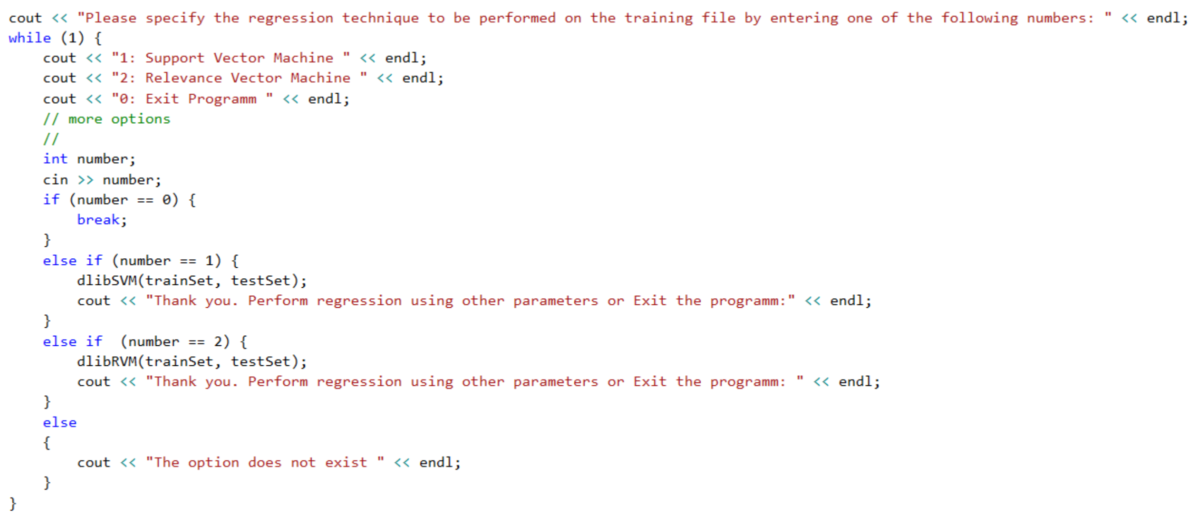


```cpp
// Function to read CSV file and store it into a vector of objects of type "property"
void readCSV(istream &input, vector<Property>& newPropertySet);
// Function to write Results from Memory to CSV file
void writeDataSet(const vector<Property>& newDataSet);

// Regression Functions
// 1: Support Vector Machine Algorithm
void dlibSVM(const vector<Property>& newDataSet, const vector<Property>& newTestSet);
// 2: Relevance Vector Machine Algorithm
void dlibRVM(const vector<Property>& newDataSet , const vector<Property>& newTestSet);
```

Figure 4 Initiation of functions in source code “Regression_Project.cpp”

The function “readCSV” is used to load both data sets from the .csv files into the memory. It takes strings of type istream as an argument from the data files train.csv and test.csv and stores them a vector of type <Property>. The function “writeDataSet” takes this vector type as input. The regression functions take two stored data sets as arguments, whereas the training data (newDataSet) is used for training and validation of the regression model. Furthermore, the trained regression function is applied on the testing data (newTestSet), which is also given as attribute for both regression functions.

The code is structured in the way that after reading the training and testing data, regression functions are executable multiple times, which is shown in Figure 5. This makes it possible for the user to perform regression several times by choosing between two types of regression algorithms, the number of samples to be used for regression and several algorithm-specific parameters. However, some attributes of the regression functions are fixed namely the kernel type used and the features used to perform the training. They have been selected and optimized beforehand by a process of feature selection, which is discussed in the presentation given for this project.



```
cout << "Please specify the regression technique to be performed on the training file by entering one of the following numbers: " << endl;
while (1) {
    cout << "1: Support Vector Machine " << endl;
    cout << "2: Relevance Vector Machine " << endl;
    cout << "0: Exit Programm " << endl;
    // more options
    //
    int number;
    cin >> number;
    if (number == 0) {
        break;
    }
    else if (number == 1) {
        dlibSVM(trainSet, testSet);
        cout << "Thank you. Perform regression using other parameters or Exit the programm:" << endl;
    }
    else if  (number == 2) {
        dlibRVM(trainSet, testSet);
        cout << "Thank you. Perform regression using other parameters or Exit the programm: " << endl;
    }
    else
    {
        cout << "The option does not exist " << endl;
    }
}
```

Figure 5 Second part of main function in source code

3.1 Implementation of regression

Next to the stored data sets, the regression functions use different parameters in order to create the regression model for the prediction of sale prices. Figure 6 shows the setup which data types are used for input of samples and targets of the training set. The sample type is set to a matrix which has the size of the number of features used for regression. The vectors of both, samples and targets are filled by a *for* loop where data from the stored training data in the memory is used as input. Furthermore, the user can specify how many samples of the stored data shall be used for regression.

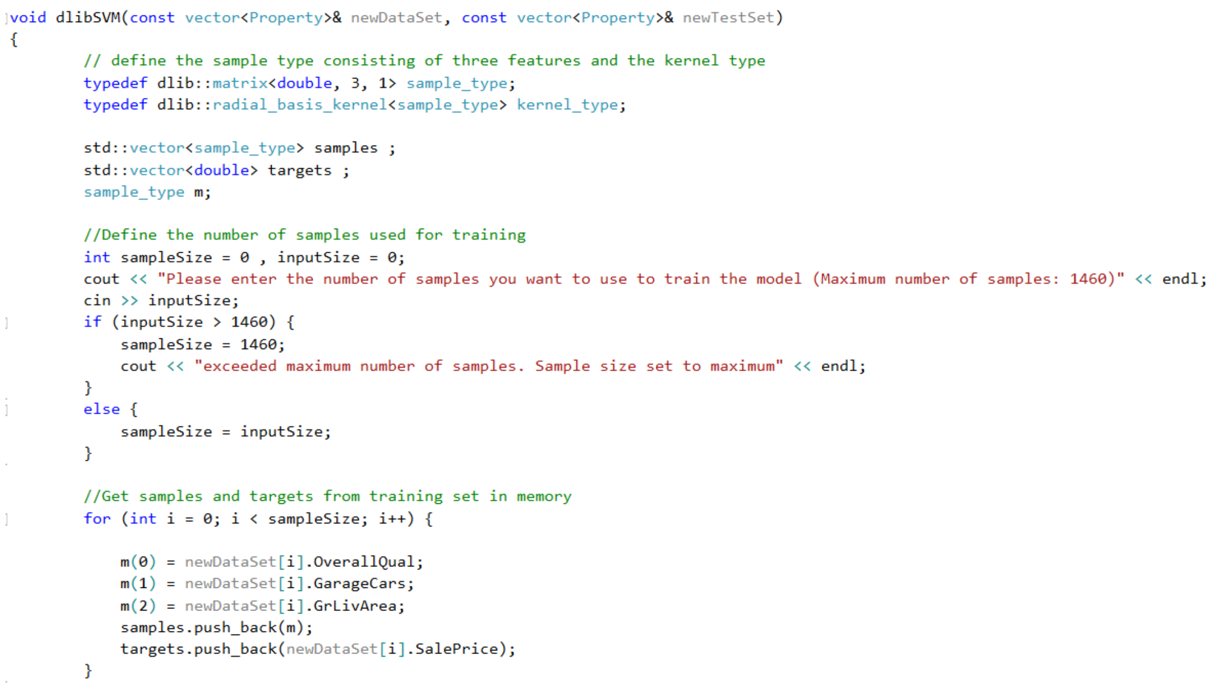

```
void dlibSVM(const vector<Property>& newDataSet, const vector<Property>& newTestSet)
{
        // define the sample type consisting of three features and the kernel type
        typedef dlib::matrix<double, 3, 1> sample_type;
        typedef dlib::radial_basis_kernel<sample_type> kernel_type;

        std::vector<sample_type> samples ;
        std::vector<double> targets ;
        sample_type m;

        //Define the number of samples used for training
        int sampleSize = 0 , inputSize = 0;
        cout << "Please enter the number of samples you want to use to train the model (Maximum number of samples: 1460)" << endl;
        cin >> inputSize;
        if (inputSize > 1460) {
            sampleSize = 1460;
            cout << "exceeded maximum number of samples. Sample size set to maximum" << endl;
        }
        else {
            sampleSize = inputSize;
        }

        //Get samples and targets from training set in memory
        for (int i = 0; i < sampleSize; i++) {

            m(0) = newDataSet[i].OverallQual;
            m(1) = newDataSet[i].GarageCars;
            m(2) = newDataSet[i].GrLivArea;
            samples.push_back(m);
            targets.push_back(newDataSet[i].SalePrice);
        }
```

Figure 6 Input of data in regression function

As a next step, samples and targets are normalized by using a member function which is available for both regression types, which can be seen in Figure 7 Normalization of samples The function "normalizer" subtracts the mean and divides by the standard deviation calculated for every sample.

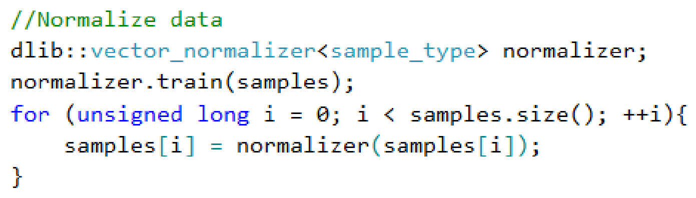
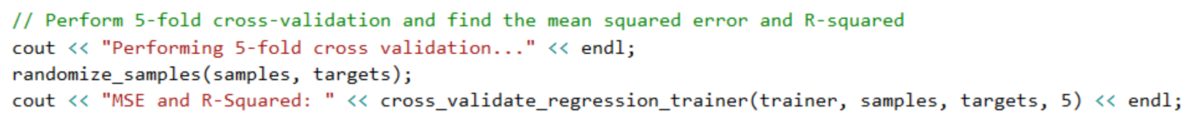

```
//Normalize data
dlib::vector_normalizer<sample_type> normalizer;
normalizer.train(samples);
for (unsigned long i = 0; i < samples.size(); ++i){
    samples[i] = normalizer(samples[i]);
}
```

Figure 7 Normalization of samples

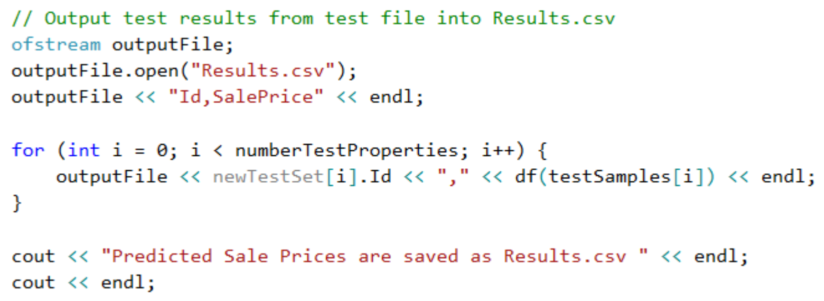
It is considered an indispensable step to perform a validation of the regression algorithm that evaluates the performance of the trained regression function independently on data which has not been used for training. The dlib library provides a function which uses k-fold cross validation as a built-in function. The performance measures in place for evaluation are mean squared error (MSE) between predicted sale prices and their respective target values and coefficient of determination (R^2). The split of training data is set to five and samples are randomized before the execution of the function, which is shown in Figure 8.



```
// Perform 5-fold cross-validation and find the mean squared error and R-squared
cout << "Performing 5-fold cross validation..." << endl;
randomize_samples(samples, targets);
cout << "MSE and R-Squared: " << cross_validate_regression_trainer(trainer, samples, targets, 5) << endl;
```

Figure 8 Validation of regression function

Finally, both regression functions include the prediction of sale prices on the test set, which is shown in Figure 9. The predicted sale prices are written in a file which is called “Results.csv”. This file contains the predictions of sale prices including its unique id of each property including a header line. The file is created every time when one of the regression functions is called.



```
// Output test results from test file into Results.csv
ofstream outputFile;
outputFile.open("Results.csv");
outputFile << "Id,SalePrice" << endl;

for (int i = 0; i < numberTestProperties; i++) {
    outputFile << newTestSet[i].Id << "," << df(testSamples[i]) << endl;
}

cout << "Predicted Sale Prices are saved as Results.csv " << endl;
cout << endl;
```

Figure 9 Output of results

3.2 Implementation of Dlib

Dlib should be implemented very easily by following these steps[5]:

1. Download latest library package on http://dlib.net/ and extract
2. Add folder containing the extracted dlib files to *include path*
3. Add dlib/all/source.cpp to project

4 Conclusion and future work

Multiple regression could be performed by using two linear regression methods to predict sale prices of properties on the basis of a data set providing 79 features of housing. The code written in C++ enables the user to load training and testing data in memory, execute the regression methods multiple times and store predicted results in a separate .csv file. Furthermore, the code includes k-fold cross validation to evaluate regression performance.

[5] http://dlib.net/compile.html

Best results were achieved using Support Vector Machine leading to a prediction accuracy of $R^2 = 0.83$ by using the ten features showing the highest correlation to sales prices. One challenge faced was the beneficial use of nominal features for linear regression. Possible future work can include an implementation of other regression methods such as Artificial Neural Networks (ANN) or decision trees suitable to solve a regression problem.