

Statistical Machine Learning with OpenCV

Philipp Wagner*

January 30, 2011

Abstract

This document is an introduction to Machine Learning with OpenCV. Artificial Neural Networks and Support Vector Machines are briefly explained and examples in OpenCV are given.

1 Introduction

Machine Learning is a branch of Artificial Intelligence and concerned with the question how to make machines able to learn from data. The core idea is to enable a machine to make intelligent decisions and predictions based on experiences from the past. Algorithms of Machine Learning require interdisciplinary knowledge and often intersect with topics of statistics, mathematics, physics, pattern recognition and more.

OpenCV (Open Source Computer Vision) is a library for computer vision and comes with a machine learning library for:

- Decision Trees
- Boosting
- Support Vector Machines
- Expectation Maximization
- Neural Networks

This document helps with:

- Setting up a development environment in Windows and Linux
- Getting Started with CMake and OpenCV
- Using the Machine Learning Library of OpenCV

A brief introduction to Support Vector Machines and Neural Networks are given. The code example used in this document is available from <https://github.com/bytefish/opencv/>.

2 Setup

Follow this guide to have a Development Environment for Linux and Windows. You have to download Eclipse and CMake. Windows users can use MinGW as a C/C++ compiler.

2.1 Eclipse with CDT (Windows/Linux)

Eclipse is an Integrated Development Environment (IDE). It is possibly best known as a Java development platform, but can be extended for languages such as C/C++, PHP, Python and more. The easiest way to get Eclipse with C/C++ support is to download *Eclipse for C++ Developers* for your operating system at: <http://www.eclipse.org/downloads/>. The download already includes the CDT Plugin (C/C++ Development Toolkit).

There is no installation. Just unpack and start the eclipse executable.

*<http://www.bytetfish.de>, bytefish-at-gmx-dot-de

2.2 MinGW (Windows only)

This step is Windows only. [MinGW \(Minimalist GNU for Windows\)](#) is a port of the [GNU Compiler Collection \(GCC\)](#) and can be used for development of native [Microsoft Windows](#) applications. Download the automated mingw-get-installer [from sourceforge](#) (called *mingw-get-inst-20101030.exe* at time of writing this). If the path to the download changes, navigate there from the MinGW project page at: <http://www.mingw.org>. Be sure to select "C++ Compiler" in the *Compiler Suite* dialog during setup.

MinGW doesn't add its binaries to the global Windows PATH environment. The MinGW Page says: *Add C:\MinGW\bin; to the PATH environment variable by opening the System control panel, going to the Advanced tab, and clicking the Environment Variables button. If you currently have a Command Prompt window open, it will not recognize the change to the environment variables; you will need to open a new Command Prompt window to get the new PATH.*

2.3 CMake

[CMake](#) is an open-source, cross-platform build system used by many opensource projects including: [OpenCV](#), [MiKTeX](#) or [Blender 3D](#). It works on Linux, Windows, Mac OS X and can generate (among many others) Eclipse CDT project files and that's why it is suggested as a build system for OpenCV projects.

Windows

Download the installer from the [Resources Page](#) at <http://www.cmake.org> (called *cmake-2.8.3-win32-x86.exe* at time of writing). This will install cmake, ccmake and the cmake-gui.

Make sure to select "Add CMake to the system PATH for all users" during setup.

Linux

CMake is often available from a distributions repository or can be downloaded as an installer from the [Resources Page](#) at <http://www.cmake.org>. Installing the generic Linux binaries with the is done by typing:

```
sudo sh cmake-2.8.3-Linux-i386.sh --prefix=/usr/local
```

In Debian or Ubuntu cmake can be installed with apt:

```
sudo apt-get install cmake cmake-gui
```

2.4 OpenCV

[OpenCV](#) is a Computer Vision Library started by [Intel](#) in 1999 and now actively developed by [Willow Garage](#). It includes advanced computer vision algorithms and sets the focus on real-time image processing. In 2009 OpenCV2 got a C++ interface and integrates with Python. If you encounter any problems with this install guide consult the [OpenCV Install Guide](#) aswell.

Windows

[Microsoft Windows](#) users download the Windows installer from the sourceforge repository of OpenCV at <http://sourceforge.net/projects/opencvlibrary/>. (called *OpenCV-2.2.0-win32-vs2010.exe* at time of writing this).

Make sure to select "Add OpenCV to the system PATH for all users" during setup.

Linux

Linux users should inspect their repositories, because most distributions have pre-packaged binaries. In Debian/Ubuntu you simply type:

```
sudo apt-get install opencv opencv-doc
```

Build OpenCV from sources

OpenCV is open source and comes with CMake as build system. Sometimes you need to compile OpenCV yourself, for example if the pre-packaged binaries use SSE2 and your CPU doesn't support SSE2. If so, download the OpenCV sources, unpack them and cd to the directory.

For building OpenCV from source type:

```
cmake .
make
sudo make install
```

Building OpenCV without SSE2

If you are actually working on a CPU without SSE2 you will need to deselect the SSE2 build option before compiling it, otherwise OpenCV won't work as expected. To see which flags your CPU supports (assuming Linux), type `cat /proc/cpuinfo | grep flags` in a terminal.

On my workstation:

```
philipp@banana:~$ cat /proc/cpuinfo | grep flags
flags : fpu vme de pse tsc msr pae mce cx8 sep mtrr pge mca cmov pat pse36 mmx fxsr sse syscall mp mmxext 3dnowext 3dnow up ts fid vid
```

It's an [AMD](#) Athlon XP and does not support SSE2. In order to turn off SSE2 support for OpenCV with the cmake-gui:

- Start `cmake-gui` either from prompt or menu.
- Set the OpenCV source folder as *Source Directory* and *Build Directory*
- Click *configure* and choose: **Unix Makefiles**.
- Deselect `ENABLE_SSE2` support from the list of flags.
- Click *configure* again to accept the setup.
- Click *generate* and proceed with the normal compiling procedure as described above.

The command line for doing the same would be (assuming you are in the OpenCV folder):

```
cmake --DENABLE_SSE2=OFF .
```

followed by:

```
make
sudo make install
```

3 Getting Started with OpenCV

Now that Eclipse, MinGW (Windows), CMake and OpenCV are installed it is time to create the first OpenCV project. It is called *hello_opencv*. Files for this example are put in a folder at: `C:\workspace\hello_opencv`. Of course any valid path can be chosen and kept consistent with the example.

1. CMakeLists.txt

CMake searches for a file called `CMakeLists.txt` The OpenCV library is linked against the executable. Save `CMakeLists.txt` from Listing 1 to `C:\workspace\hello_opencv`.

Listing 1: `CMakeLists.txt`

```
cmake_minimum_required(VERSION 2.6)
PROJECT(hello_opencv_proj)
FIND_PACKAGE( OpenCV REQUIRED )
ADD_EXECUTABLE( hello_opencv main.cpp)
TARGET_LINK_LIBRARIES( hello_opencv ${OpenCV_LIBS} )
```

2. main.cpp

Now the OpenCV C++ API is introduced.

```
#include "cv.h"
#include "ml.h"

using namespace std;

int main() {
    cv::Mat plot(240,320,CV_8UC3);
    plot.setTo(cv::Scalar(255,255,255));
    cv::putText(plot,"Hello_OpenCV",Point(1,100), FONT_HERSHEY_SIMPLEX, 1.0,Scalar(0,0,0), 1, 8, false);
    cv::namedWindow("Hello_OpenCV", CV_WINDOW_AUTOSIZE);
    cv::imshow("Hello_OpenCV", plot);
    cv::waitKey();
}
```

Save main.cpp from Listing 3 to C:\workspace\hello_opencv.

3. Generate project and makefiles

We will need to import the project to eclipse, so we generate the CDT specific Makefiles:

- Start the *cmake-gui*
- Browse to C:\workspace\hello_opencv as *Source Folder* and *Build Folder*
- Click *configure* and select "Eclipse CDT4 - Unix Makefiles" from the Dialog
- Click *configure* again to accept the configuration
- Click *generate* to generate the Makefiles and Eclipse related project files

Your folder will now look like this:

```
C:\workspace\hello_opencv>dir
11.12.2010 14:33 <DIR> .
11.12.2010 14:33 <DIR> ..
11.12.2010 14:33 22.202 .cproject
11.12.2010 14:33 3.591 .project
11.12.2010 14:28 29.062 CMakeCache.txt
11.12.2010 14:33 <DIR> CMakeFiles
11.12.2010 14:18 190 CMakeLists.txt
11.12.2010 14:28 1.470 cmake_install.cmake
11.12.2010 14:14 556 main.cpp
11.12.2010 14:33 5.179 Makefile
```

If you feel more familiar with a terminal type:

```
C:\> cd C:\workspace\hello_opencv
C:\> cmake -G"Eclipse_CDT4_-_Unix_Makefiles" .
```

4. Import project and makefiles into Eclipse

Start Eclipse and from the menu choose:

1. File -> Import...
2. General -> Existing Projects into Workspace
3. Select C:\workspace\hello_opencv as root directory
4. Click Finish

5. Build and run the project

To build the project:

- Project -> Build All

To run the project:

- Run -> Run

And a window similar to Figure 3 pops up.

Figure 1: OpenCV: Hello World



4 Random Numbers in OpenCV

Each thread in OpenCV has access to a default random number generator `cv::theRNG()`. For a single pass the seed for the random number generator doesn't have to be set, but if many random numbers have to be generated (for example in a loop) the seed has to be set.

This can be done by assigning the local time to the random number generator:

```
cv::theRNG() = cv::RNG(time(0))
```

4.1 Uniform Distribution

Uniform Distributions can be generated in OpenCV with the help of `cv::randu`. The signature of `cv::randu` is:

```
void randu(Mat& mtx, const Scalar& low, const Scalar& high);
```

Where

- **mtx** is the Matrix to be filled with uniform distributed random numbers
- **low** is the inclusive lower boundary of generated random numbers
- **high** is the exclusive upper boundary of generated random numbers

4.2 Normal Distribution

Normal Distribution can be generated in OpenCv with the help of `cv::randn`.

```
void randn(Mat& mtx, const Scalar& mean, const Scalar& stddev);
```

Where

- **mtx** is the Matrix to be filled with normal distributed random numbers
- **mean** the mean value of the generated random numbers
- **stddev** the standard deviation of the generated random numbers

4.3 Bivariate Gaussian Distribution

Multivariate normal distributions can be generated with the C API by using `cvRandMVNormal` from the Machine Learning library. However this function seems to have no C++ equivalent yet, so in order to generate random numbers for the bivariate case one can also use the function given in [2](#).

Listing 2: bivariate_gaussian dtribution

```
void bivariate_gaussian(float mu_x, float sigma_x, float mu_y, float sigma_y, float rho, cv::Mat &x, cv::Mat &y) {  
    assert(x.rows == y.rows && x.rows > 0);  
  
    int n = x.rows;  
  
    randn(x, mu_x, sigma_x);  
  
    cv::Mat r(n, 1, CV_32FC1);  
    cv::theRNG() = cv::RNG(time(0));
```

```

randn(r, 0, 1);
for(int i = 0; i < n; i++) {
    y.at<float>(i, 0) = sqrt(sigma.y*sigma.y - (rho * sigma.y)*(rho * sigma.y)) * r.at<float>(i, 0) + mu.y;
    y.at<float>(i, 0) = y.at<float>(i, 0) + rho * sigma.y/sigma.x * (x.at<float>(i, 0) - mu.x);
}

```

5 Preparing the Training and Test Dataset for OpenCV ML

In the C++ Machine Learning API of OpenCV training and test data is given as a `cv::Mat` matrix. The constructor of `cv::Mat` is defined as:

```

Mat::Mat(int rows, int cols, int type);

```

Where

- rows is the number of samples (for all classes!)
- columns is the number of dimensions
- type is the image type

In the machine learning library of OpenCV each row or column in the training data is a n-dimensional sample. The default ordering is row sampling and class labels are given in a matrix with equal length (one column only, of course).

```

cv::Mat trainingData(numTrainingPoints, 2, CV_32FC1);
cv::Mat testData(numTestPoints, 2, CV_32FC1);

cv::randu(trainingData, 0, 1);
cv::randu(testData, 0, 1);

cv::Mat trainingClasses = labelData(trainingData, equation);
cv::Mat testClasses = labelData(testData, equation);

```

Since only binary classification problems are considered the function `f` returns the classes `-1` and `1` for a given two-dimensional data point:

```

// function to learn
int f(float x, float y, int equation) {
    switch(equation) {
        case 0:
            return y > sin(x*10) ? -1 : 1;
            break;
        case 1:
            return y > cos(x * 10) ? -1 : 1;
            break;
        case 2:
            return y > 2*x ? -1 : 1;
            break;
        case 3:
            return y > tan(x*10) ? -1 : 1;
            break;
        default:
            return y > cos(x*10) ? -1 : 1;
    }
}

```

And to label data one can use the function `labelData`:

```

cv::Mat labelData(cv::Mat points, int equation) {
    cv::Mat labels(points.rows, 1, CV_32FC1);
    for(int i = 0; i < points.rows; i++) {
        float x = points.at<float>(i, 0);
        float y = points.at<float>(i, 1);
        labels.at<float>(i, 0) = f(x, y, equation);
    }
    return labels;
}

```

6 Support Vector Machines (SVM)

Support Vector Machines were first introduced by Vapnik and Chervonenkis in their paper "*Theory of Pattern Recognition*" [VC74]. The core idea is to find the optimal hyperplane to separate a dataset, while there are theoretically infinite hyperplanes to separate the dataset. A hyperplane is chosen, so that the distance to the nearest datapoint of both classes is maximized (Figure 2). The points spanning the hyperplane are the *Support Vectors*, hence the name *Support Vector Machines*. [CV95]

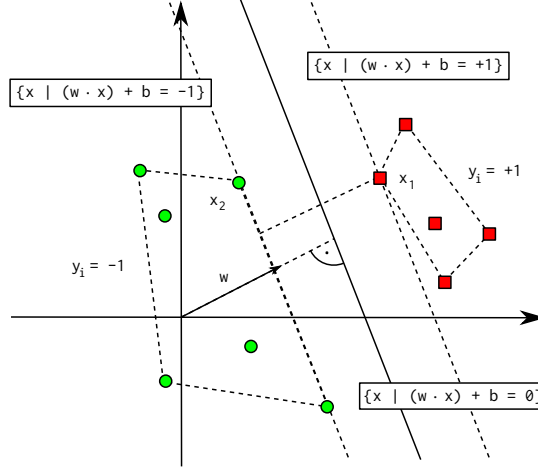


Figure 2: Maximum Margin Classifier.

6.1 Definition

Given a Set of Datapoints \mathcal{D} :

$$\mathcal{D} = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^n$$

where

- x_i is a point in p-dimensional vector
- y_i is the corresponding class label

We search for $\omega \in \mathbb{R}^n$ and bias b , forming the Hyperplane H:

$$\omega^T x + b = 0$$

that separates both classes so that:

$$\begin{aligned} \omega^T x + b &= 1, \text{ if } y = 1 \\ \omega^T x + b &= -1, \text{ if } y = -1 \end{aligned}$$

The optimization problem that needs to be solved is:

$$\min \frac{1}{2} \omega^T \omega$$

subject to:

$$\begin{aligned} \omega^T x + b &\geq 1, y = 1 \\ \omega^T x + b &\leq -1, y = -1 \end{aligned}$$

Such quadratic optimization problems can be solved with standard solvers, such as [GNU Octave](#) or [Matlab](#).

6.1.1 Non-linear SVM

The kernel trick is used for classifying non-linear datasets. It works by transforming data points into a higher dimensional feature space with a *kernel function*, where the dataset can be separated again (see Figure 3).

Commonly used kernel functions are *RBF kernels*:

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{\sigma^2}\right)$$

or *polynomial kernels*:

$$k(x, x') = (x \cdot x')^d$$

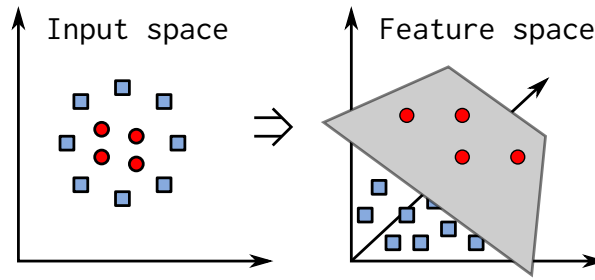


Figure 3: Kernel Trick

6.2 SVM in OpenCV

Parameters for a SVM have to be defined in the structure *CvSVMParams*.

Parameters

Listing 3: Example CvSVMParams

```
CvSVMParams param = CvSVMParams();

param.svm_type = CvSVM::C_SVC;
param.kernel_type = CvSVM::LINEAR;

param.degree = 0; // for poly
param.gamma = 20; // for poly/rbf/sigmoid
param.coef0 = 0; // for poly/sigmoid

param.C = 7; // for CV_SVM_C_SVC, CV_SVM_EPS_SVR and CV_SVM_NU_SVR
param.nu = 0.0; // for CV_SVM_NU_SVC, CV_SVM_ONE_CLASS, and CV_SVM_NU_SVR
param.p = 0.0; // for CV_SVM_EPS_SVR

param.class_weights = NULL; // for CV_SVM_C_SVC
param.term_crit.type = CV_TERMCRIT_ITER | CV_TERMCRIT_EPS;
param.term_crit.max_iter = 1000;
param.term_crit.epsilon = 1e-6;
```

Where the parameters are (taken from the OpenCV 1.0 documentation¹):

- **svm_type**
 - **CvSVM::C_SVC** n-class classification ($n \geq 2$), allows imperfect separation of classes with penalty multiplier C for outliers.
 - **CvSVM::NU_SVC** n-class classification with possible imperfect separation. Parameter ν (in the range $0 \dots 1$, the larger the value, the smoother the decision boundary) is used instead of C .
 - **CvSVM::ONE_CLASS** one-class SVM. All the training data are from the same class, SVM builds a boundary that separates the class from the rest of the feature space.
 - **CvSVM::EPS_SVR** regression. The distance between feature vectors from the training set and the fitting hyper-plane must be less than p . For outliers the penalty multiplier C is used.
 - **CvSVM::NU_SVR** regression; ν is used instead of p .
- **kernel_type**
 - **CvSVM::LINEAR** no mapping is done, linear discrimination (or regression) is done in the original feature space. It is the fastest option. $d(x, y) = x \cdot y == (x, y)$.
 - **CvSVM::POLY** polynomial kernel: $d(x, y) = (\text{gamma} * (x \cdot y) + \text{coef0})^{\text{degree}}$.
 - **CvSVM::RBF** radial-basis-function kernel; a good choice in most cases: $d(x, y) = \exp(-\text{gamma} * |x - y|^2)$.
 - **CvSVM::SIGMOID** sigmoid function is used as a kernel: $d(x, y) = \tanh(\text{gamma} * (x \cdot y) + \text{coef0})$.
- **C, nu, p** Parameters in the generalized SVM optimization problem.
- **class_weights** Optional weights, assigned to particular classes. They are multiplied by C and thus affect the misclassification penalty for different classes. The larger weight, the larger penalty on misclassification of data from the corresponding class.

¹http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_ml.htm

- **term_criteria** Termination procedure for iterative SVM training procedure (which solves a partial case of constrained quadratic optimization problem)
 - **type** is either `CV_TERMCRIT_ITER` or `CV_TERMCRIT_ITER`
 - **max_iter** is the maximum number of iterations in training.
 - **epsilon** is the error to stop training.

Training

Training can either be done by passing the vector with the training data and vector with the corresponding class labels to the constructor or the train method.

```
CvSVM(const CvMat* _train_data,
      const CvMat* _responses,
      const CvMat* _var_idx=0,
      const CvMat* _sample_idx=0,
      CvSVMParams _params=CvSVMParams());
```

where

- **_train_data** is a Matrix with the n-dimensional feature vectors
- **_responses** is a vector with the class for the corresponding feature vector
- **_var_idx** identifies features of interest (can be left empty for this example, in code: `cv::Mat()`)
- **_sample_idx** identifies samples of interest (can be left empty for this example, in code: `cv::Mat()`)
- **_params** Parameter for the SVM from Listing 3

This applies to the train method aswell:

```
virtual bool train(const CvMat* _train_data,
                  const CvMat* _responses,
                  const CvMat* _var_idx=0,
                  const CvMat* _sample_idx=0,
                  CvSVMParams _params=CvSVMParams());
```

The *train* methods of the SVM has some limitations:

- Only `CV_ROW_SAMPLE` is supported
- Missing measurements are not supported

The *train_auto* method finds the best parameters with a Gridsearch and a k-fold cross validation. This method is only available for OpenCV Versions ≥ 2.0 .

Prediction

Self explaining code.

```
for(int i = 0; i < testData.rows; i++) {
    cv::Mat sample = testData.row(i);
    float result = svm.predict(sample);
}
```

Support Vectors

The support vectors of a SVM can be obtained using the `get_support_vector` function of the API:

```
int svec_count = svm.get_support_vector_count();
for(int vecNum = 0; vecNum < svec_count; vecNum++) {
    const float* vec = svm.get_support_vector(vecNum);
}
```

A complete example for Support Vector Machines in OpenCV is given in the Appendix.

7 Multi Layer Perceptron

An Artificial Neural Network is a biological inspired computational model. *Inputs* multiplied by *weights* result in an *activation* and form the *output* of a network.

Research in Artificial Neural Networks (ANN) began in 1943, when McCulloch and Pitts gave a definition of a formal neuron in their paper "*A Logical Calculus of the Ideas Immanent in Nervous Activity*" [MP43]. In 1958 Rosenblatt invented the perceptron, which is a simple feedforward neural network. The downfall of the perceptron algorithm is that it only converges on lineary seperable

datasets and is not able to solve non-linear problems such as the XOR problem. This was proven by Minsky and Papert in their monograph *"Perceptrons"*, but they showed that a two-layer feedforward architecture can overcome this limitation. It was until 1986 when Rumelhart, Hinton and Williams presented a learning rule for Artificial Neural Networks with hidden units in their paper *"Learning Internal Representations by Error Propagation"*. The original discovery of backpropagation is actually credited to Werbos who described the algorithm in his 1974 Ph.D. thesis at Harvard University, see [Wer94] for the roots of backpropagation.

A detailed introduction to Pattern Recognition with Neural Networks is given by [Bis95].

7.1 Backpropagation

1. Initialize weights with random values
2. Present the input vector to the network
3. Evaluate the output of the network after a forward propagation of the signal
4. Calculate $\delta_j = (y_j - d_j)$ where d_j is the target output of neuron j and y_j is the actual output $y_j = g(\sum_i w_{ij}x_i) = (1 + e^{-\sum_i w_{ij}x_i})^{-1}$, (when the activation function is of a sigmoid type).
5. For all other neurons (from the first to the last layer) calculate $\delta_j = \sum_k w_{jk}g'(x)\delta_k$, where δ_k is δ_j of the succeeding layer and $g'(x) = y_k(1 - y_k)$
6. Update weights with $w_{ij}(t+1) = w_{ij}(t) - \eta y_i y_j (1 - y_j) \delta_j$, where η is the learning rate.
7. Termination Criteria. Goto Step 2 for a fixed number of iterations or an error.

The network error is defined as:

$$E = \frac{1}{2} \sum_{j=1}^m (d_j - y_j)^2$$

7.2 MLP in OpenCV

A Multilayer Perceptron in OpenCV is an instance of CvANN_MLP.

CvANN_MLP mlp;

Parameters

The performance of a Multilayer perceptron depends on its parameters. The parameters I use are given in Listing 7.2.

```
CvTermCriteria criteria;
criteria.max_iter = 100;
criteria.epsilon = 0.00001f;
criteria.type = CV_TERMCRIT_ITER | CV_TERMCRIT_EPS;

CvANN_MLP_TrainParams params;
params.train_method = CvANN_MLP_TrainParams::BACKPROP;
params.bp_dw_scale = 0.05f;
params.bp_moment_scale = 0.05f;
params.term_crit = criteria;
```

Where the parameters are (taken from the OpenCV 1.0 documentation²):

- **term_crit** The termination criteria for the training algorithm. It identifies how many iterations is done by the algorithm (for sequential backpropagation algorithm the number is multiplied by the size of the training set) and how much the weights could change between the iterations to make the algorithm continue.
- **train_method** The training algorithm to use; can be one of CvANN_MLP_TrainParams::BACKPROP (sequential backpropagation algorithm) or CvANN_MLP_TrainParams::RPROP (RPROP algorithm, default value).
- **bp_dw_scale** (Backpropagation only): The coefficient to multiply the computed weight gradient by. The recommended value is about 0.1.
- **bp_moment_scale** (Backpropagation only): The coefficient to multiply the difference between weights on the 2 previous iterations. This parameter provides some inertia to smooth the random fluctuations of the weights. It can vary from 0 (the feature is disabled) to 1 and beyond. The value 0.1 or so is good enough.

²http://www.cognotics.com/opencv/docs/1.0/ref/opencvref_ml.htm

- `rp_dw0` (RPROP only): Initial magnitude of the weight delta. The default value is 0.1.
- `rp_dw_plus` (RPROP only): The increase factor for the weight delta. It must be > 1 , default value is 1.2 that should work well in most cases, according to the algorithm's author.
- `rp_dw_minus` (RPROP only): The decrease factor for the weight delta. It must be < 1 , default value is 0.5 that should work well in most cases, according to the algorithm's author.
- `rp_dw_min` (RPROP only): The minimum value of the weight delta. It must be > 0 , the default value is `FLT_EPSILON`.
- `rp_dw_max` (RPROP only): The maximum value of the weight delta. It must be > 1 , the default value is 50.

Layers

The purpose of a neural network is to generalize, which is the ability to approximate outputs for inputs not available in the training set. [Sar02] While small networks may not be able to approximate a function, large networks tend to overfit and not find any relationship in data.³ It has been shown that, given enough data, a multi layer perceptron with one hidden layer can approximate any continuous function to any degree of accuracy. [HSW92]

The number of neurons per layer is stored in a row-ordered `cv::Mat`.

```
cv::Mat layers = cv::Mat(4, 1, CV_32SC1);

layers.row(0) = cv::Scalar(2);
layers.row(1) = cv::Scalar(10);
layers.row(2) = cv::Scalar(15);
layers.row(3) = cv::Scalar(1);

mlp.create(layers);
```

Training

The API for training a multilayer perceptron takes the training data, training classes and the structure for the parameters.

```
mlp.train(trainingData, trainingClasses, cv::Mat(), cv::Mat(), params);
```

Prediction

The API for the prediction is slightly different from the SVM API. Activations of the output layer are stored in a `cv::Mat` response, simply because one can design neural networks with multiple neurons in the output layer.

Since the problem used in this example is a binary classification problem, it is sufficient to have only one neuron in the output layer. It is therefore the only activation to check.

```
mlp.predict(sample, response);
float result = response.at<float>(0,0);
```

8 Evaluation

In this section the following algorithms will be used for classification:

- Support Vector Machine
- Multi Layer Perceptron
- k-Nearest-Neighbor
- Normal Bayes
- Decision Tree

To evaluate a predictor it is possible to calculate its accuracy. For two classes it is given as:

$$Accuracy = \frac{\text{true positive}}{\text{true positive} + \text{false positive}}$$

³The model describes random error or noise instead of the relationship of the data.

The performance of Support Vector Machines and especially Neural Networks depend on the parameters chosen. In case of a neural network it is difficult to find the appropriate parameters and architecture. Designing an Artificial Neural Network is often more a rule of thumb and networks should be optimized iteratively starting with one hidden layer and few neurons. Parameters for a Support Vector Machine can be estimated using Cross Validation and Grid Search (both can be used as `train_auto` in OpenCV ≥ 2.0).

Parameters are not optimized in this experiment, remember to optimize the parameters yourself when using one of the algorithms.

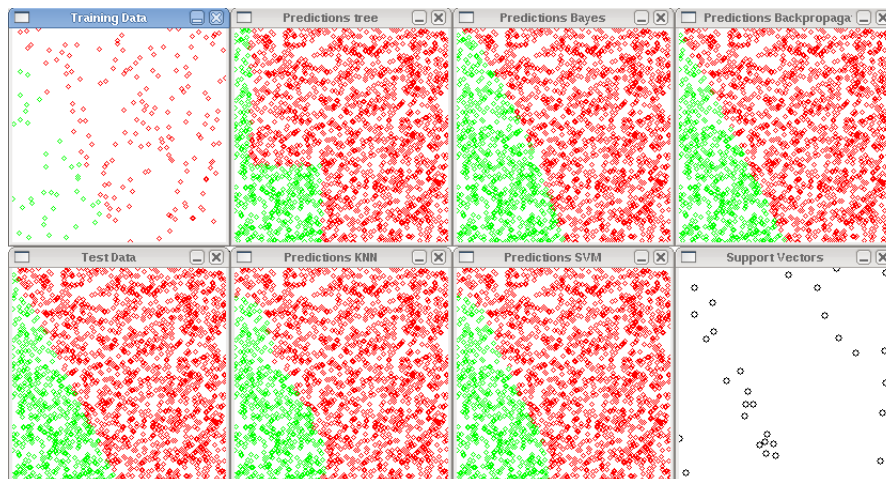
8.1 Experiment Data

In this experiment linear and non-linear functions are learned. 200 points for training and 2000 points for testing are generated.

8.2 $y = 2x$

Predictor	Accuracy
Support Vector Machine	0.99
Multi Layer Perceptron (2, 10, 15, 1)	0.994
k-Nearest-Neighbor ($k = 3$)	0.9825
Normal Bayes	0.9425
Decision Tree	0.923

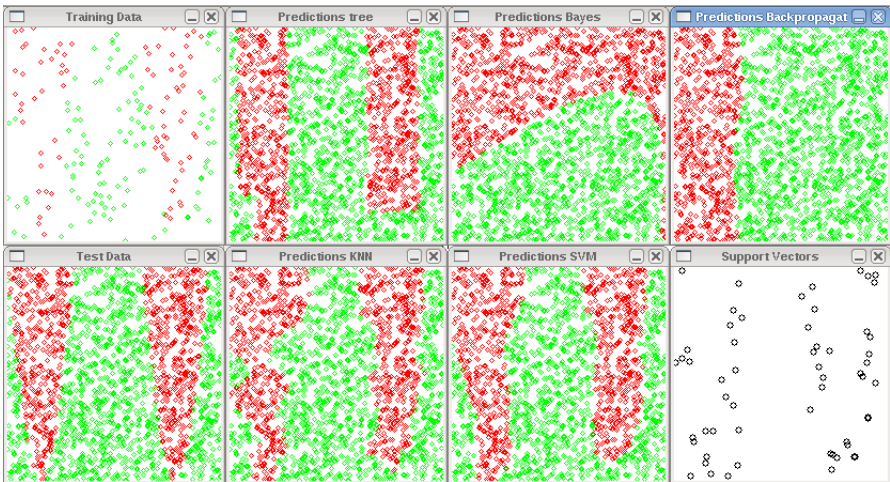
8.2.1 Plot



8.3 $y = \sin(10x)$

Predictor	Accuracy
Support Vector Machine	0.913
Multi Layer Perceptron (2, 10, 15, 1)	0.6855
k-Nearest-Neighbor ($k = 3$)	0.9
Normal Bayes	0.632
Decision Tree	0.886

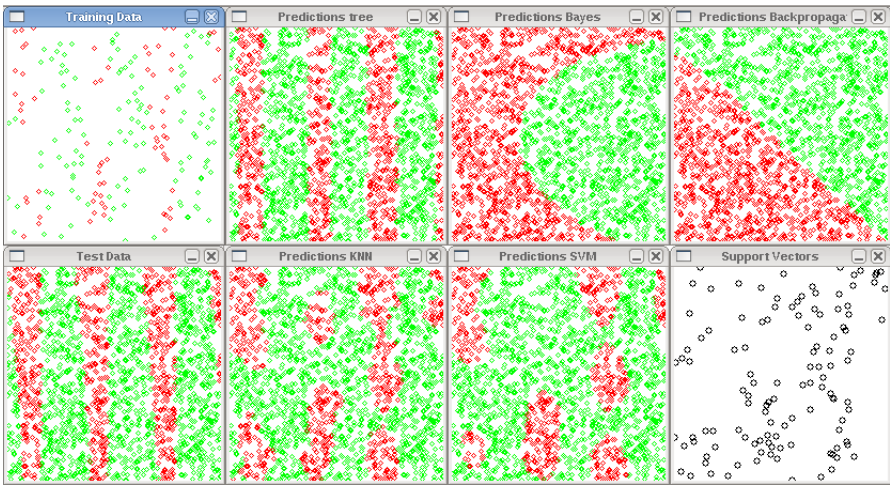
8.3.1 Plot



8.4 $y = \tan(10x)$

Predictor	Accuracy
Support Vector Machine	0.7815
Multi Layer Perceptron (2, 10, 15, 1)	0.5115
k-Nearest-Neighbor (k = 3)	0.8195
Normal Bayes	0.542
Decision Tree	0.9155

8.4.1 Plot



A main.cpp

Listing 4: main.cpp

```
#include <iostream>
#include <math.h>
#include <string>
#include "cv.h"
#include "ml.h"
#include "highgui.h"

using namespace cv;
using namespace std;

#define numTrainingPoints 200
#define numTestPoints 2000
#define size 200
#define eq 0

#define plotSupportVectors
```

```

// accuracy
float evaluate(cv::Mat& predicted, cv::Mat& actual) {
    assert(predicted.rows == actual.rows);
    int t = 0;
    int f = 0;
    for(int i = 0; i < actual.rows; i++) {
        float p = predicted.at<float>(i,0);
        float a = actual.at<float>(i,0);
        if((p >= 0.0 && a >= 0.0) || (p <= 0.0 && a <= 0.0)) {
            t++;
        } else {
            f++;
        }
    }
    return (t * 1.0) / (t + f);
}

// plot data and class
void plot_binary(cv::Mat& data, cv::Mat& classes, string name) {
    cv::Mat plot(size, size, CV_8UC3);
    plot.setTo(CV_RGB(255,255,255));
    for(int i = 0; i < data.rows; i++) {
        float x = data.at<float>(i,0) * size;
        float y = data.at<float>(i,1) * size;

        if(classes.at<float>(i, 0) > 0) {
            cv::circle(plot, Point(x,y), 2, CV_RGB(255,0,0),1);
        } else {
            cv::circle(plot, Point(x,y), 2, CV_RGB(0,255,0),1);
        }
    }
    cv::imshow(name, plot);
}

// function to learn
int f(float x, float y, int equation) {
    switch(equation) {
        case 0:
            return y > sin(x*10) ? -1 : 1;
            break;
        case 1:
            return y > cos(x * 10) ? -1 : 1;
            break;
        case 2:
            return y > 2*x ? -1 : 1;
            break;
        case 3:
            return y > tan(x*10) ? -1 : 1;
            break;
        default:
            return y > cos(x*10) ? -1 : 1;
    }
}

// label data with equation
cv::Mat labelData(cv::Mat points, int equation) {
    cv::Mat labels(points.rows, 1, CV_32FC1);
    for(int i = 0; i < points.rows; i++) {
        float x = points.at<float>(i,0);
        float y = points.at<float>(i,1);
        labels.at<float>(i, 0) = f(x, y, equation);
    }
    return labels;
}

void svm(cv::Mat& trainingData, cv::Mat& trainingClasses, cv::Mat& testData, cv::Mat& testClasses) {
    CvSVMParams param = CvSVMParams();

    param.svm_type = CvSVM::C_SVC;
    param.kernel_type = CvSVM::RBF; //CvSVM::RBF, CvSVM::LINEAR ...
    param.degree = 0; // for poly
    param.gamma = 20; // for poly/rbf/sigmoid
    param.coef0 = 0; // for poly/sigmoid

    param.C = 7; // for CV_SVM_C_SVC, CV_SVM_EPS_SVR and CV_SVM_NU_SVR
    param.nu = 0.0; // for CV_SVM_NU_SVC, CV_SVM_ONE_CLASS, and CV_SVM_NU_SVR
    param.p = 0.0; // for CV_SVM_EPS_SVR

    param.class_weights = NULL; // for CV_SVM_C_SVC
    param.term_crit_type = CV_TERMCRIT_ITER + CV_TERMCRIT_EPS;
    param.term_crit_max_iter = 1000;
    param.term_crit_epsilon = 1e-6;

    // SVM training (use train auto for OpenCV>=2.0)
    CvSVM svm(trainingData, trainingClasses, cv::Mat(), cv::Mat(), param);

    cv::Mat predicted(testClasses.rows, 1, CV_32F);

    for(int i = 0; i < testData.rows; i++) {
        cv::Mat sample = testData.row(i);

        float x = sample.at<float>(0,0);
        float y = sample.at<float>(0,1);

        predicted.at<float>(i, 0) = svm.predict(sample);
    }

    cout << "Accuracy_{SVM} _=" << evaluate(predicted, testClasses) << endl;
    plot_binary(testData, predicted, "Predictions_SVM");

    // plot support vectors
#ifdef plotSupportVectors
    cv::Mat plot_sv(size, size, CV_8UC3);
    plot_sv.setTo(CV_RGB(255,255,255));

    int svec_count = svm.get_support_vector_count();
    for(int vecNum = 0; vecNum < svec_count; vecNum++) {
        const float* vec = svm.get_support_vector(vecNum);
        cv::circle(plot_sv, Point(vec[0]*size, vec[1]*size), 3, CV_RGB(0, 0, 0));
    }
    cv::imshow("Support_Vectors", plot_sv);
#endif
}

```

```

}

void mlp(cv::Mat& trainingData, cv::Mat& trainingClasses, cv::Mat& testData, cv::Mat& testClasses) {

    cv::Mat layers = cv::Mat(4, 1, CV_32SC1);

    layers.row(0) = cv::Scalar(2);
    layers.row(1) = cv::Scalar(10);
    layers.row(2) = cv::Scalar(15);
    layers.row(3) = cv::Scalar(1);

    CvANN_MLP mlp;
    CvANN_MLP_TrainParams params;
    CvTermCriteria criteria;
    criteria.max_iter = 100;
    criteria.epsilon = 0.00001f;
    criteria.type = CV_TERMCRIT_ITER | CV_TERMCRIT_EPS;
    params.train_method = CvANN_MLP_TrainParams::BACKPROP;
    params.bp_dw_scale = 0.05f;
    params.bp_moment_scale = 0.05f;
    params.term_crit = criteria;

    mlp.create(layers);

    // train
    mlp.train(trainingData, trainingClasses, cv::Mat(), cv::Mat(), params);

    cv::Mat response(1, 1, CV_32FC1);
    cv::Mat predicted(testClasses.rows, 1, CV_32F);
    for(int i = 0; i < testData.rows; i++) {
        cv::Mat sample = testData.row(i);
        mlp.predict(sample, response);
        predicted.at<float>(i,0) = response.at<float>(0,0);
    }

    cout << "Accuracy-{MLP} := " << evaluate(predicted, testClasses) << endl;
    plot_binary(testData, predicted, "Predictions_Backpropagation");
}

void knn(cv::Mat& trainingData, cv::Mat& trainingClasses, cv::Mat& testData, cv::Mat& testClasses, int K) {

    CvKNearest knn(trainingData, trainingClasses, cv::Mat(), false, K);
    cv::Mat predicted(testClasses.rows, 1, CV_32F);
    for(int i = 0; i < testData.rows; i++) {
        const cv::Mat sample = testData.row(i);
        predicted.at<float>(i,0) = knn.find_nearest(sample, K);
    }

    cout << "Accuracy-{KNN} := " << evaluate(predicted, testClasses) << endl;
    plot_binary(testData, predicted, "Predictions_KNN");
}

void bayes(cv::Mat& trainingData, cv::Mat& trainingClasses, cv::Mat& testData, cv::Mat& testClasses) {

    CvNormalBayesClassifier bayes(trainingData, trainingClasses);
    cv::Mat predicted(testClasses.rows, 1, CV_32F);
    for (int i = 0; i < testData.rows; i++) {
        const cv::Mat sample = testData.row(i);
        predicted.at<float>(i, 0) = bayes.predict(sample);
    }

    cout << "Accuracy-{BAYES} := " << evaluate(predicted, testClasses) << endl;
    plot_binary(testData, predicted, "Predictions_Bayes");
}

void decisiontree(cv::Mat& trainingData, cv::Mat& trainingClasses, cv::Mat& testData, cv::Mat& testClasses) {

    CvDTree dtree;
    cv::Mat var_type(3, 1, CV_8U);

    // define attributes as numerical
    var_type.at<unsigned int>(0,0) = CV_VAR_NUMERICAL;
    var_type.at<unsigned int>(0,1) = CV_VAR_NUMERICAL;
    // define output node as numerical
    var_type.at<unsigned int>(0,2) = CV_VAR_NUMERICAL;

    dtree.train(trainingData, CV_ROW_SAMPLE, trainingClasses, cv::Mat(), cv::Mat(), var_type, cv::Mat(), CvDTreeParams());
    cv::Mat predicted(testClasses.rows, 1, CV_32F);
    for (int i = 0; i < testData.rows; i++) {
        const cv::Mat sample = testData.row(i);
        CvDTreeNode* prediction = dtree.predict(sample);
        predicted.at<float>(i, 0) = prediction->value;
    }

    cout << "Accuracy-{TREE} := " << evaluate(predicted, testClasses) << endl;
    plot_binary(testData, predicted, "Predictions_tree");
}

int main() {

    cv::Mat trainingData(numTrainingPoints, 2, CV_32FC1);
    cv::Mat testData(numTestPoints, 2, CV_32FC1);

    cv::randu(trainingData, 0.1);
    cv::randu(testData, 0.1);

    cv::Mat trainingClasses = labelData(trainingData, eq);
    cv::Mat testClasses = labelData(testData, eq);

    plot_binary(trainingData, trainingClasses, "Training_Data");
    plot_binary(testData, testClasses, "Test_Data");

    svm(trainingData, trainingClasses, testData, testClasses);
    mlp(trainingData, trainingClasses, testData, testClasses);
    knn(trainingData, trainingClasses, testData, testClasses, 3);
    bayes(trainingData, trainingClasses, testData, testClasses);
}

```



```

decisiontree(trainingData, trainingClasses, testData, testClasses);
cv::waitKey();
return 0;
}

```

References

- [Bis95] BISHOP, C. M.: *Neural Networks for Pattern Recognition*. Oxford : Oxford University Press, 1995
- [CV95] CORTES, Corinna ; VAPNIK, Vladimir: Support-Vector Networks. In: *Machine Learning* Bd. 20, 1995, S. 273–297
- [HSW92] HORNIK, K. ; STINCHCOMBE, M. ; WHITE, H.: Multilayer Feedforward Networks Are Universal Approximators. In: WHITE, Halber (Hrsg.): *Artificial Neural Networks: Approximation and Learning Theory*. Oxford, UK : Blackwell, 1992, S. 12–28
- [MP43] MCCULLOCH, W. ; PITTS, W.: A logical calculus of the ideas immanent in nervous activity. In: *Bulletin of Mathematical Biophysics* 5 (1943), S. 115–133
- [Sar02] SARLE, Warren S.: *comp.ai.neural-nets FAQ, Part 3 of 7: Generalization*. <http://www.faqs.org/faqs/ai-faq/neural-nets/part3/>, 2002
- [VC74] VAPNIK, V. N. ; CHERVONENKIS, A. Y.: *Theory of Pattern Recognition [in Russian]*. USSR : Nauka, 1974
- [Wer94] WERBOS, P.J.: *The Roots of Backpropagation: From ordered derivatives to Neural Networks and Political Forecasting*. New York : John Wiley and Sons, 1994