IntelligenceLab 7.5

.NET Quick Start

www.openwire.org
www.mitov.com

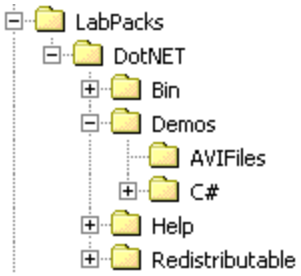
Index

Installation

IntelligenceLab comes with an installation program. Just start the installation by double-clicking on the Setup.exe file and follow the installation instructions.

Where is IntelligenceLab?

After the installation IntelligenceLab is located under a single root directory. The default location is C:\Program Files\LabPacks or C:\Program Files (x86)\LabPacks on 64 bit systems. During the installation the user has the option to select alternative directory. Here is how the directory structure should look like after the installation:

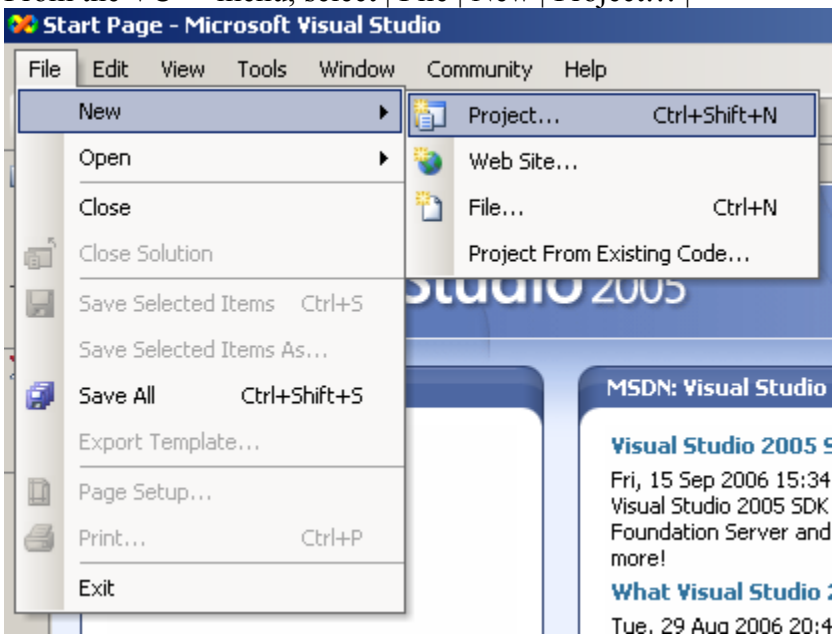


Under the “Demos” directory are located the demo files. The help files and the documentation are located under the “Help” directory. The component . NET 2.0/3.5/4.0 assemblies and the redistributable DLL files are located under the “Bin” directory.
It is a great idea to start by opening and compiling the demo files. The demo projects ware developed with Visual C# 2005.

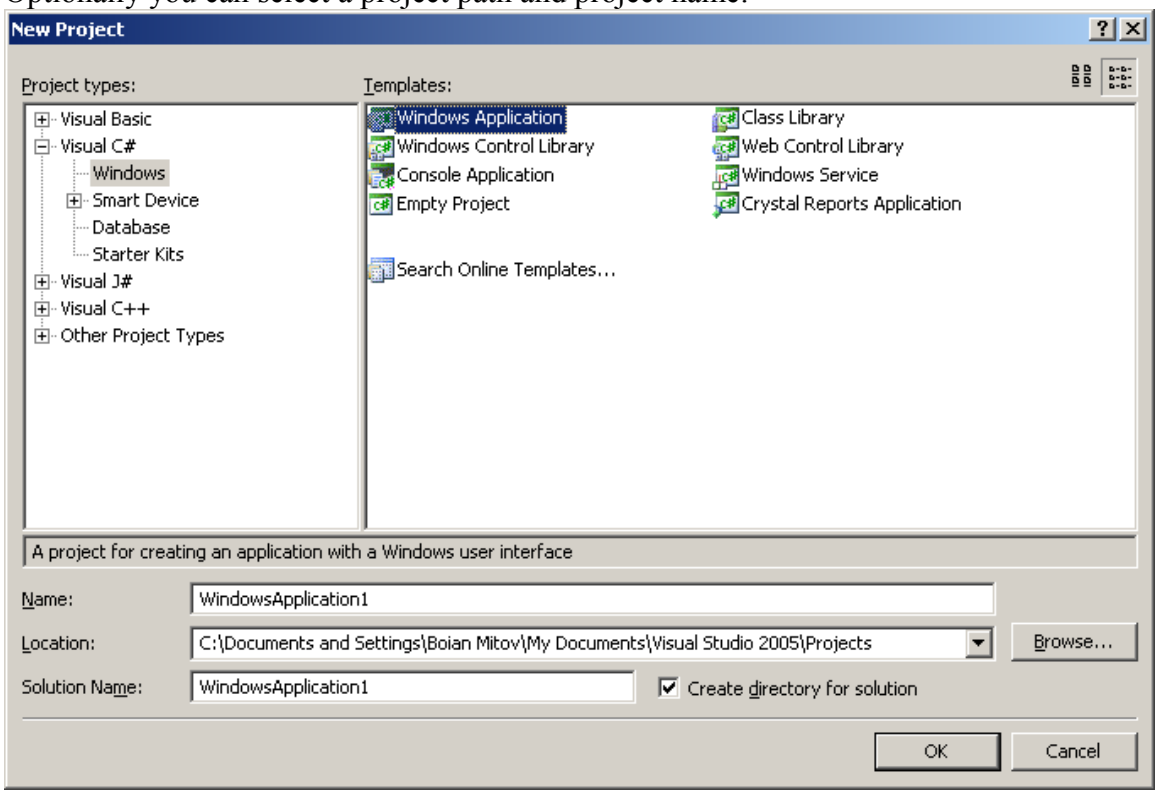
Creating a new IntelligenceLab project in Visual C#

All of the examples in this manual start with creating a C# Windows .NET based project. The following chapters will assume that you have created the project and will teach you how to add specific IntelligenceLab functionality.

Start by creating a new project.
From the VC++ menu, select | File | New | Project… |



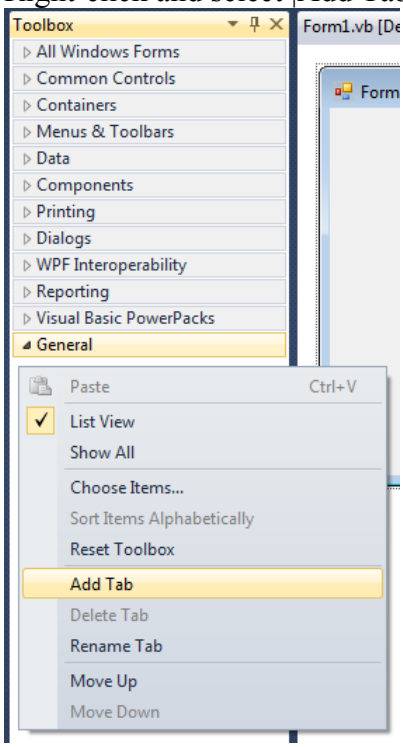
In the "New Project" dialog select | Visual C# | Windows Application |
Optionally you can select a project path and project name:



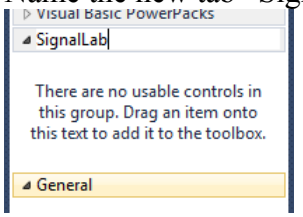
Click OK.

Installing the components on the Toolbox

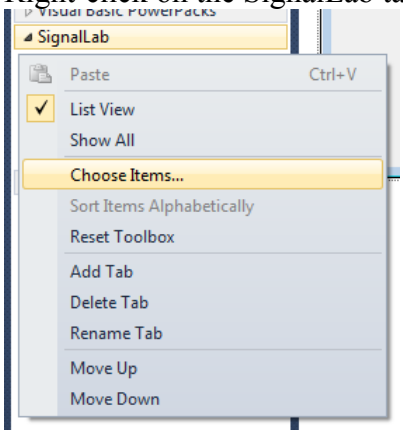
Before using the components in your project, you will have to install them on the component Toolbox.
The install in version 3.1 and up will automatically install the components on the toolbar, however if it fails, or if you have selected not to do so during the installation, here is a way to install the components manually:
We assume that you have already created a project, and the toolbox with the .NET components has appeared.
Open the component toolbox and expand the General section.
Right-click and select |Add Tab| from the menu:



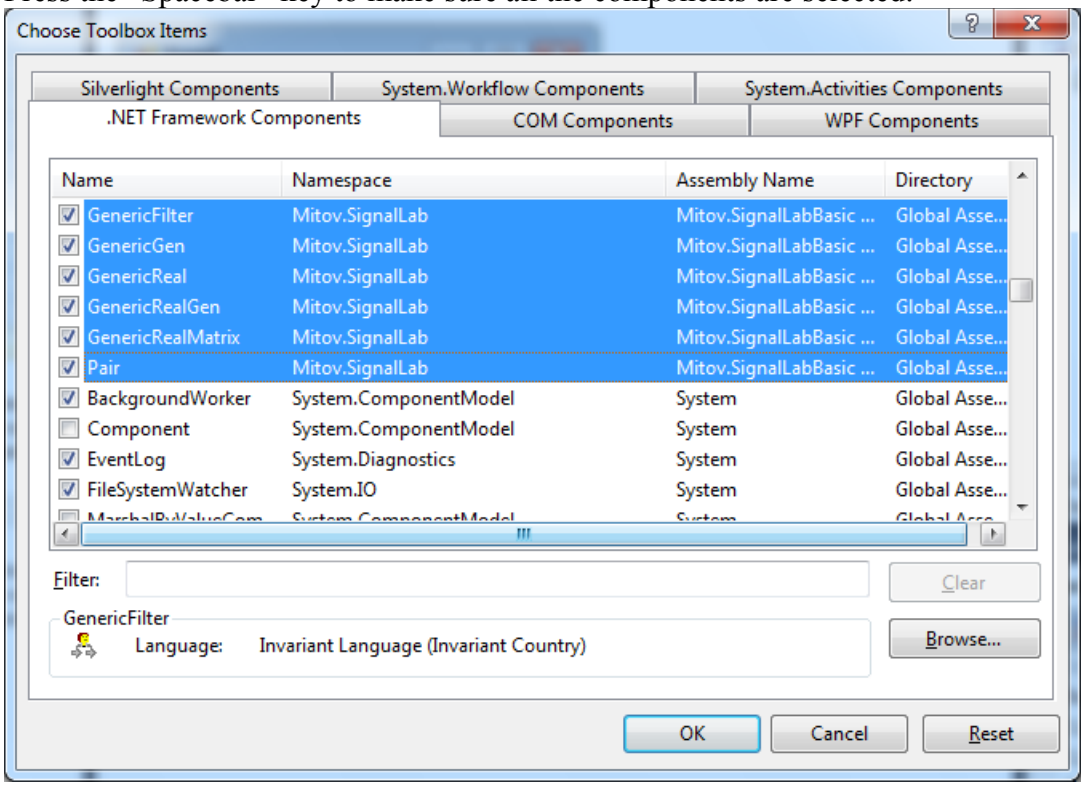
Name the new tab “SignalLab”:



Right-click on the SignalLab tab and select |Choose Items…| from the menu:

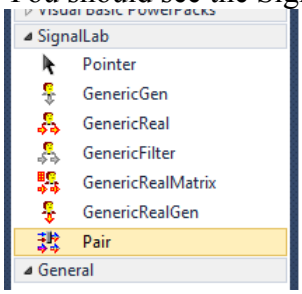


In the "Choose Toolbox Items" dialog select the components that belong to the Mitov.SignalLabBasic.DLL (You can order the components by "Assembly Name"). Press the "Spacebar" key to make sure all the components are selected:



Click OK.

You should see the SignalLabBasic components on your toolbox:



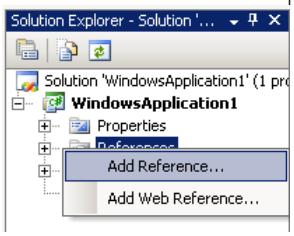
Close and restart the Visual Studio IDE, then reopen the project.

Continue repeating the same steps and install the following assemblies:
On the "IntelligenceLab" tab and install IntelligenceLab.dll.

Now you can start using the components in your .NET development.

Adding the necessary assembly references to your application

Visual studio will automatically add the assemblies being referenced when adding components to the project. If this mechanism fails, you can manually add the necessary assemblies as shown here:
In the "Solution Explorer" select the "References" node and right-click on it.
From the menu select |Add Reference…|



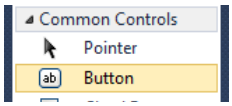
Navigate to the Select the Mitov.AudioLabBasic.dll from the LabPacks\Bin\2.0 subdirectory and add the necessary assemblies.
Here is the list of necessary assemblies:

- For Mitov.BasicLab.DLL – None.
- For Mitov.SignalLabBasic.DLL:
    a. Mitov.BasicLab.DLL
- For Mitov.IntelligenceLabBasic.DLL:
    a. Mitov.BasicLab.DLL
- For Mitov.IntelligenceLab.DLL:
    a. Mitov.IntelligenceLabBasic.DLL:
    b. Mitov.SignalLabBasic.DLL
    c. Mitov.BasicLab.DLL

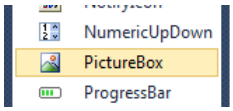
Creating classifier application

Create and setup a new project as described in the "Creating a new IntelligenceLab project in Visual C#" chapter.

In the Toolbox select and drop on the form Button component from the "Common Controls" tab:



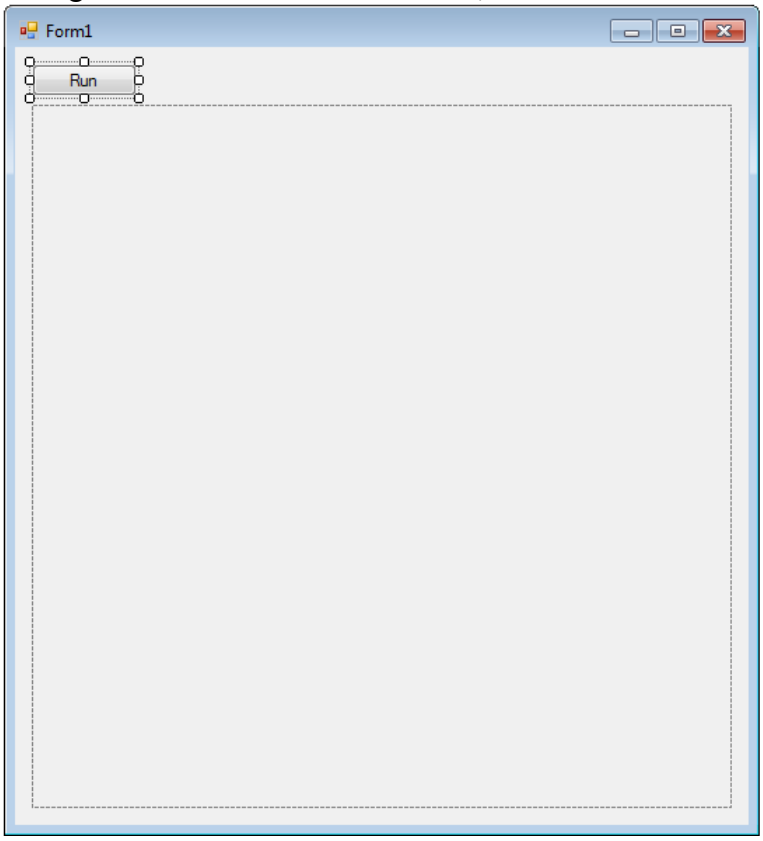
In the Toolbox select and drop on the form PictureBox component from the "Common Controls" tab:



In the Form1 add the highlighted lines:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private int m_i;
        private int m_j;
        private System.Drawing.Graphics m_Graphics;

        private System.Drawing.Brush m_RedBrush = new
System.Drawing.SolidBrush(Color.FromArgb(127, 0, 0));
        private System.Drawing.Brush m_GreenBrush = new
System.Drawing.SolidBrush(Color.FromArgb(0, 127, 0));
        private System.Drawing.Brush m_BlueBrush = new
System.Drawing.SolidBrush(Color.FromArgb(0, 0, 127));

        private Random m_RandomGen = new Random(123);

        public Form1()
        {
            InitializeComponent();
        }
    }
}
```

Change the Text of the button to Run, and double-click on the button:

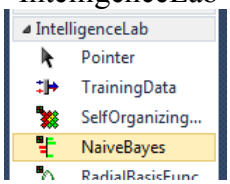


In the button1_Click event add the highlighted lines:

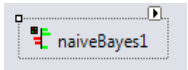
```
        private void button1_Click(object sender, EventArgs e)
        {
            Mitov.SignalLab.RealMatrixBuffer ATrainingData = new
Mitov.SignalLab.RealMatrixBuffer(150, 2);
            Mitov.SignalLab.RealBuffer AResposes = new
Mitov.SignalLab.RealBuffer(150);

            Bitmap image = new Bitmap(500, 500);
            m_Graphics = System.Drawing.Graphics.FromImage(image);

            for( int i = 0; i < 50; i ++)
            {
                AResposes[i] = 1;
                ATrainingData[i][0] = m_RandomGen.NextDouble() * 250;
                ATrainingData[i][1] = m_RandomGen.NextDouble() * 200;
            }

            for (int i = 50; i < 100; i++)
            {
                AResposes[i] = 2;
                ATrainingData[i][0] = 300 + m_RandomGen.NextDouble()
* 199;
                ATrainingData[i][1] = 100 + m_RandomGen.NextDouble()
* 200;
            }
```

```
            for (int i = 100; i < 150; i++)
            {
                AResposes[i] = 3;
                ATrainingData[i][0] = m_RandomGen.NextDouble() * 300;
                ATrainingData[i][1] = 300 + m_RandomGen.NextDouble()
* 199;
            }

            naiveBayes1.Train(ATrainingData, AResposes, false);

            Mitov.SignalLab.RealBuffer ATestData = new
Mitov.SignalLab.RealBuffer(2);
            for (m_i = 0; m_i < 500; m_i++)
                for (m_j = 0; m_j < 500; m_j++)
                {
                    ATestData[1] = m_i;
                    ATestData[0] = m_j;
                    naiveBayes1.Predict(ATestData);
                }

            // display the original training samples
            for (int i = 0; i < 150 / 3; i++)
            {
                m_Graphics.FillEllipse(System.Drawing.Brushes.Red,
(float)ATrainingData[i][0], (float)ATrainingData[i][1], 5, 5);
                m_Graphics.FillEllipse(System.Drawing.Brushes.Lime,
(float)ATrainingData[i + 50][0], (float)ATrainingData[i + 50][1], 5,
5);
                m_Graphics.FillEllipse(System.Drawing.Brushes.Blue,
(float)ATrainingData[i + 100][0], (float)ATrainingData[i + 100][1],
5, 5);
            }

            pictureBox1.Image = image;

            pictureBox1.Invalidate();
        }
```

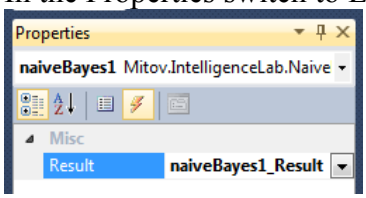
In the Toolbox select and drop on the form NaiveBayes component from the "IntelligenceLab" tab:



Select the naiveBayes1:



In the Properties switch to Events, and add a new event for Result:

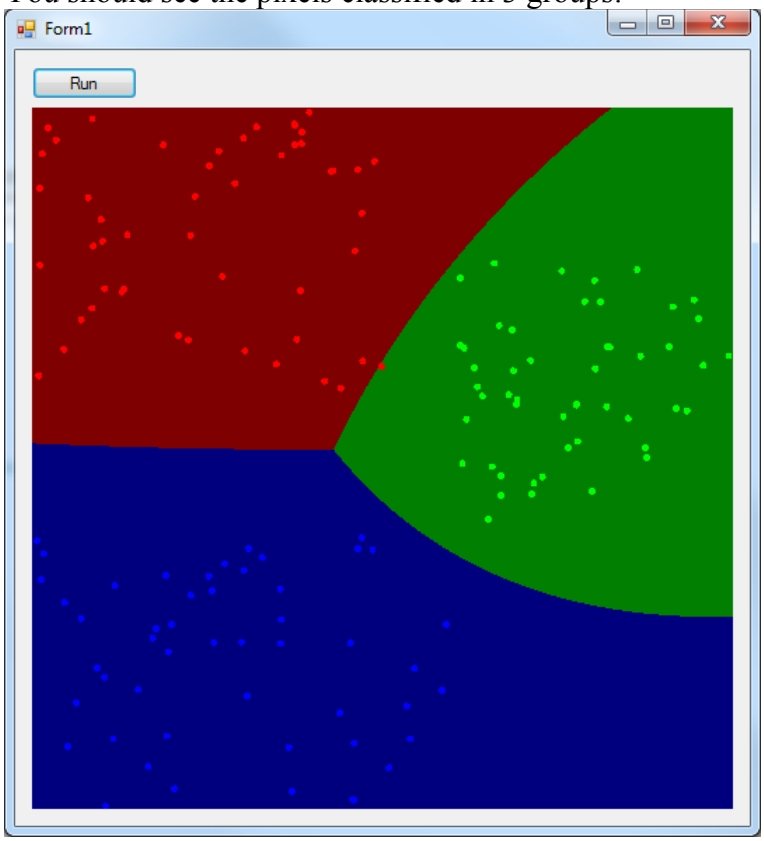


In the naiveBayes1_Resul event add the highlighted lines:

```
        private void naiveBayes1_Result(object Sender,
Mitov.IntelligenceLab.NaiveBayesResultEventArgs Args)
        {
            switch ((int)(Args.Result.ResultClass + 0.5))
            {
                case 1: m_Graphics.FillRectangle(m_RedBrush, m_j,
m_i, 1, 1); break;
                case 2: m_Graphics.FillRectangle(m_GreenBrush, m_j,
m_i, 1, 1); break;
                case 3: m_Graphics.FillRectangle(m_BlueBrush, m_j,
m_i, 1, 1); break;
            }
        }
```

Compile and run the application.

Click on the “Run” button.
You should see the pixels classified in 3 groups:



You have just learned how to use IntelligenceLab classifier.

Distributing your application

Once you have finished the development of your application you most likely will need to distribute it to other systems. Version 5.0.2 and higher of the library will move all the necessary DLL files in the Release directory of your project. You will only need to distribute the files in the directory. To use this feature, make sure that the “Copy Local” property is set for all the assemblies from the library. Please check with the Visual Studio help for your version of Video Studio on how to configure assemblies as private assemblies.

Deploying your 32 bit application with the IPP DLLs

The compiled applications can be deployed to the target system by simply copying the executable. The application will work, however the performance can be improved by also copying the Intel IPP DLLs provided with the library.
The DLLs are under the [install path]\LabPacks\IppDLL\Win32 directory([install path] is the location where the library was installed).
In 32 bit Windows to deploy IPP, copy the files to the [Windows]\System32 directory on the target system.
In 64 bit Windows to deploy IPP, copy the files to the [Windows]\SysWOW64 directory on the target system.
[Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS
This will improve the performance of your application on the target system.

Deploying your 64 bit application

The current version of the library requires when deploying 64 bit applications, the Intel IPP DLLs to be deployed as well.

The DLLs are under the [install path]\LabPacks\IppDLL\Win64 directory([install path] is the location where the library was installed).

To deploy IPP, copy the files to the [Windows]\System32 directory on the target system.
[Windows] is the Windows directory - usually C:\WINNT or C:\WINDOWS
This will improve the performance of your application on the target system.