1

Can machines learn?

This book is about machine learning and F#. In this chapter, we will learn basic concepts of machine learning, Secondly, we will execute and see few sample machine learning projects to get the high level overview of the process along with details of inner workings. In order to understand Machine learning algorithms or to develop your own algorithm, it is essential to have good knowledge of math topics, especially - Linier algebra, Probability/Statistics, Calculus and Optimization. The book covers only necessary math background whenever necessary.

This chapter covers introducing machine learning, need for machine learning, two major classification of machine learning algorithms and finally we would run few sample applications and see their output to get the idea machine learning algorithms in general.

Let’s start with an introduction of the subject.

# 1.1 Introducing Machine learning

**Machine Learning** (ML), involves the development of algorithms that discover knowledge from specific data sets. ML is a heart of Artificial intelligence. It’s moving from research community to the world of business and ML is partly an (black) art and a science. The ultimate goal of machine learning is to make computers that should be able to program themselves by the interaction with their environment

Today, Machine Learning has a wide range of applications, including natural language processing, search engine function, Social media analysis, medical diagnosis, credit card fraud detection, stock market analysis etc.

ML is generally categorized as a Computer Science subject and it has become increasingly aligned with theory and techniques from statistics.

In the recent year machine learning has entered into many business and consumer applications, this transformation from research labs to the “real world” has increased investment, interest and effort to the field. That is, the ML field has been growing rapidly. Algorithm replication, new algorithm development and applying such algorithms to real-world problems are crucial factor to the progress of the field.

Few use cases where ML used are - Search engines learn how to give the best search results, E-commerce software gives us best suggestions to buy, Email spam filter software learns to filter our email messages, Bank transactions are secured by software that learns how to detect frauds and Social networking software suggesting friends, posts. Digital cameras learn to detect faces, personal assistance applications (Ex: Siri) on smart-phones learn to recognize voice commands. Machine learning is also used in scientific applications such as medicine, astronomy etc.

# 1.2 Need for Machine Learning

**Automation**: Traditional software process starts with interviewing the experts and creating an algorithm that automates their process, in machine learning Process, we collect input-output examples from the experts then develop a learning function to map from the input to the output. In other words A typical programming process consist of providing Input and Program to the computer, then the computer runs the program to generate desired result. ML is differing from the traditional programing in the following way. It has to accept input and output in order to produce "Program" as result. That is making the computers to program themselves (automating the automation with help of data). Because writing program for every need is very hard, ML helps to produce programs in a cost effective way.

**Scalability**: From another important perspective, ML replaces rule based systems due to the following reasons. A Rule-based implementation is difﬁcult for the programmer, may miss many edge-cases, becomes difficult to maintain (due to amount of “If-Then-Else” clauses), often doesn’t work too well for many problems (OCR, annotation etc.). Hence, ML is an alternative route to build complicated systems.

**Data Size**: The rapid growth of digital world is producing ever increasing data which has large amount of hidden patterns.

**Reuse**: Already there are tens of thousands of machine learning algorithms and hundreds new algorithms are developed every year. Hence, most of the time, we may have to choose right algorithm for the problem that we have at hand or we may have to extend existing algorithm for the given problem. Sometimes, we may have to develop algorithms from scratch.

**Flexibility**: Ability to develop Machine learning algorithms which are independent of the data domain.

**Program from data**: The beauty of machine learning is to create systems that gets better and better by feeding data to, rather than changing code.

# 1.3 Types of Machine learning algorithms

Broadly speaking, there are two main paradigms exist in machine learning such as **supervised** and **unsupervised learning**.

In **supervised learning** (which is most widely used), objects in a given dataset are classified using a set of features or attributes. The result of the classification process is a set of rules that prescribe assignments of "objects to classes" based on values of features. The goal in supervised learning is to design a system that should accurately predict the class membership of new objects based on the available features. This type of learning also called as predictive learning.

In **unsupervised** (also called **descriptive**) learning, no predefined class labels are available for the objects, unlike supervised learning, we are not told what the desired output is for each input. In this case, the goal is to explore the data and discover similarities between objects. Similarities are used to define groups of objects, referred to as clusters. Clustering aims at dividing objects into groups (clusters) using measures of similarity, such as Euclidean distance. In other words, unsupervised learning is intended to unveil natural groupings in the data, or discovering “interesting structure” in the data.

The below table describes difference between clustering and classification.

|  |  |  |
| --- | --- | --- |
| # | **Clustering** | **Classification** |
| 1 | Unsupervised learning (class label has to be learned) | Supervised learning (class label is known) |
| 2 | Goal is to identify similar groups of objects | Goal is to predict classes from the object properties/attribute values |
| 3 | Groups (clusters, new classes) are discovered | Classifiers are learnt from sets of classified examples |
| 4 | Dataset consists of attributes | Datasets consist of attributes and a class labels |
| 5 | clusters are discovered based on distances. | Pre-defined classes |

Thus, the two paradigms may informally be contrasted as follows: in supervised learning, the data come with class labels, and we learn how to associate labeled data with classes; In unsupervised learning, all the data are unlabeled, and the learning procedure consists of both defining the labels and associating objects with them. There is a third type of machine learning, known as reinforcement learning, which is not commonly used, this learning works based on punishment signals.

Note that, the data could be anything: numbers, text, images, video, categories, time series, date etc. In order to experiment with ML algorithms, we need two groups of data sets.

**Training set**: We should get this depends on the problem, it is usually build by drawing many records. The source of the records may include internal database/warehouse, or external source such as web content, social media, Government data, etc.

**Testing set**: We may also build other collections, i.e. a testing set to test predictions on data we didn’t train with.

The ability to make accurate predictions are depends on whether our training data represents future records adequately. We have to choose better features and increasing the amount of training data often make a bigger difference in prediction quality than improving our learning algorithm. Over increasing of training data may introduce noise in our dataset. So the feature selection and amount data required are depends on problem and expected quality output, so it is a black art. Our prediction quality is a result of the features we choose that are co-related with the output value that we try to predict. If the training data is too small, it can’t capture expected output.

In the remaining chapter, we would run few sample ML application, in the forthcoming chapters we would see many use cases and examples in detail of the two major types of learning algorithms.

# 1.4 See it in action:

In this section, we are going to run the first category of few machine learning samples, that is classification algorithms.

Handwritten recognition by Multi-class Kernel Support Vector Machines.

Data classification using Kernel Support Vector Machines.

Data classification using Support Vector Machine.

Data classification using Multi-class Support Vector Machines.

Data classification with Decision Trees.

Data classification with the Naive Bayes classifier.

## 1.4.1 Support Vector Machines

The first four examples are developed using SVMs. Lets understand Support vector machines.

Support Vectors Machines (SVMs) has been used across a wide range of applications such as recognizing handwritten digits, face identification, text categorization, annotation etc. Support vector machines (SVMs) are a set of related supervised learning methods used for classification and regression. In simple words, given a set of training examples, each marked as belonging to one of two categories, a SVM training algorithm builds a model that predicts whether a new example falls into one category or the other. Intuitively, SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall on.

The Support Vector Machine is by nature a binary classifier. One of the ways to extend the original SVM algorithm to multiple classes is to build a one-against-one scheme where multiple SVMs specialize to recognize each of the available classes. By using a competition scheme, the original multi-class classification problem is then reduced to smaller binary problems. The goal of multiclass classification is trying to put an object into one of a number of classes. For instance, if we have trained the model with historic news articles which categorized by news type, then we could predict whether a new article is about entertainment, sports, politics, religion, etc.

Consider another example dataset (Fig 1.3) described patients by 2 genes, gene X and gene Y. the patients are grouped under normal and cancer patients. First of all the objects are represented geometrically (by vectors). Assume that we have converted our problem space into mathematical representation. That is, objects are represented as some points on surface. Few points are representing Cancer patients and few are Normal patients. The objective of the solution/algorithm is finding a line/curve in the space to separate the two groups of points. The line/curve is a called decision surface that separates two classes of points. The linear decision surface (“hyperplane”) that can separate patient classes and has the largest distance (i.e., largest “gap” or “margin”) between border-line patients (i.e., “support vectors”). A hyperplane is a linear decision surface that splits the space into two parts, It is obvious that a hyperplane is a binary classifier.

Fig 1.1

Normal Patients

Cancer Patients

Gene X

Gene Y

Gap

Insertimage\_9348OS\_01\_01.png

The solid straight line in the center is the decision surface. If the hyperplane maximizes the margin then it would have better generalization. Better the generalization means, it can predict new items more accurately. If such decision surface does not exist, then we have to map the data into a much higher dimensional space where the separating decision surface is found. Note that a hyperplane in is a line and a hyperplane in is a plane. The higher dimension space is constructed via mathematical projection called “kernel trick”. By using optimization theory, we could efficiently compute the hyperplane that separates two classes with the largest “gap”.

From our hands on example, first we have loaded dataset, and we have done the training. During the training, SVMs takes numerical inputs, and SVM algorithm finds the support vectors using the training inputs. From the support vectors, it finds the dividing line. Remember that support vectors are points which are closer to the lines (Fig 1.3 contains three points which are closer to the lines). Once the training is over, it can be used to automatically classify new items, Classifying new item is easy and faster, the new point should be plotted on the surface and decide which side the new object belongs to from the center line. Hence there is no need to consult the training dataset while deciding on new objects.

These sample applications are part of Accord.NET framework, which comprised by the set of libraries and sample applications. We are going to execute these sample applications on Windows and see how they work.

All the sample applications are written in C# which utilises two frameworks. These are listed below:

AForge.NET: An open-source framework for image processing applications such as video capturing, computer vision, general image processing, etc.

Accord.NET: The Accord.NET is an open source framework, and it extends the AForge.NET Framework. This framework provides machine learning, mathematics, statistics, computer vision, computer audition, and several scientific computing related methods and techniques to .NET. It’s a complete scientific computing environment.

## 1.4.2 General setup and requirements:

The Accord.NET Framework requires AForge.NET to be already installed, each release of the Accord.NET Framework is always built against one particular version AForge.NET, for this tutorial we are going to use Accord.NET 2.12 and AForge.NET 2.2.5. These two frameworks should be compatible in order to run the samples. Accord.NET v2.12 is compiled against AForge.NET Framework 2.2.5.

Download the AForge.NET Framework 2.2.5 from https://code.google.com/p/aforge/downloads/list

(Select the installer AForge.NET Framework-2.2.5.exe and download it)

After the executable installer has finished downloading, double-click it to start the setup process. Leave the default settings in the installer wizard.

Download the Accord.NET Framework from

https://github.com/accord-net/framework/releases

(Select Accord.NET Framework v2.12 version and download Accord.NET.Framework-2.12.0.exe installer). There is also a compressed file which would be of interest for Linux users willing to run the framework on Mono.

After the executable installer has finished downloading, please double-click it to start the setup process. Leave the default settings in the installer wizard

By default, all files will be installed under C:\Program Files\Accord.NET

After the installation, Source code, Documentation and several sample applications will be available by default at

Source code will be at Accord.NET\Framework\Sources

Documentation will be at Accord.NET\Framework\Docs

Sample applications will be at Accord.NET\Framework\Samples

The goal of this section is to show how to run the samples.  
  
1.4.3 Handwritten recognition by Multi-class Kernel Support Vector Machines

The first sample application explains the recognition of handwritten digits using Multi-class Support Vector Machines (SVM), it is a type of classification algorithm.

Steps to run the application:

1. Open the application.

(From the installed location. C:\Program Files\Accord.NET\Framework\Samples\MachineLearning\Handwriting (SVMs)\bin\x86\Release\Handwriting (SVMs).exe

1. To load the dataset:

Select File | Open, This would load sample dataset.

Note: The dataset is stored in the “optdigits-tra.txt” file under Resources directory of the sample application. The dataset that we use has been taken from UC Irvine Machine Learning Repository. The name of the dataset is called "Optical Recognition of Handwritten Digits Data Set" (a.k.a. the Optdigits Dataset). http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

1. To start the learning process

Click the button "Start training". Using the default settings, it should take few seconds to complete.

1. To start the classification process

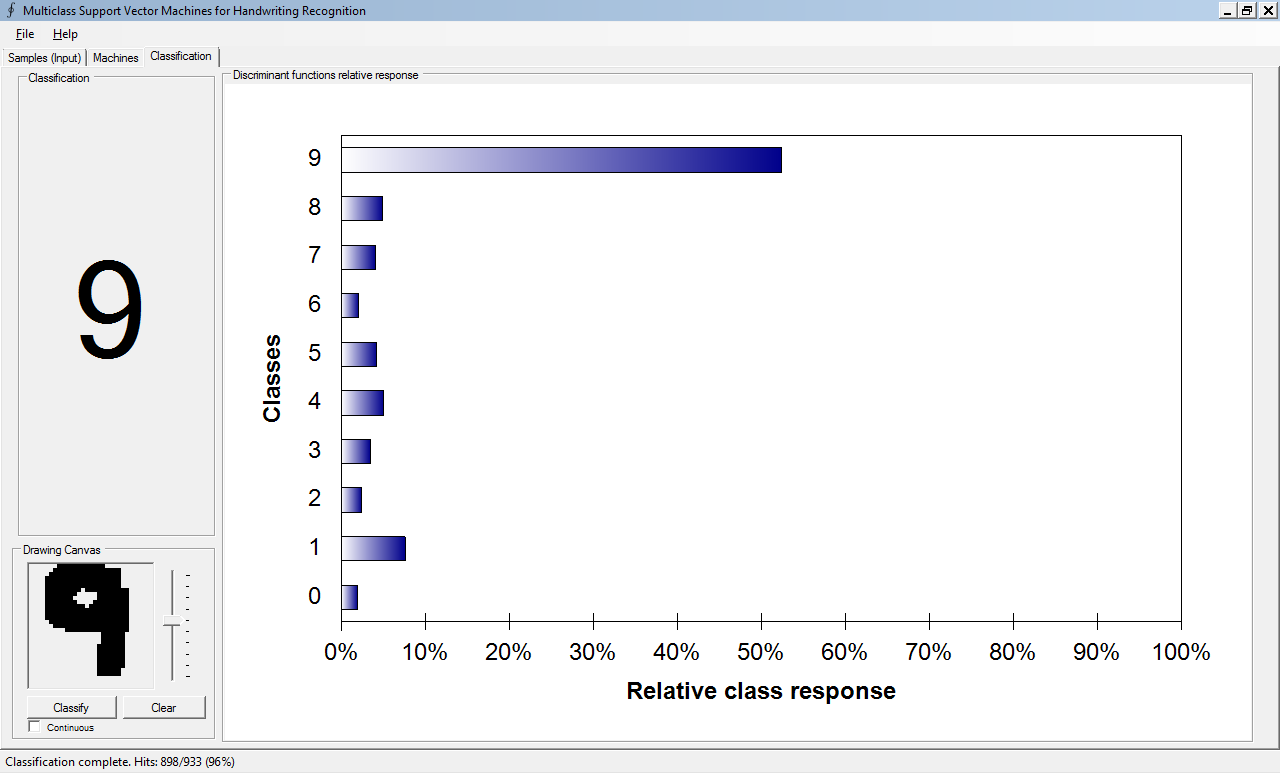
After training is completed, click a decision strategy either "Classify by voting" or “Classify by elimination” to start the classification. Using the default values, it has achieved 96% accuracy in my machine for “Classify by elimination”.

Note: The sample application was implemented two decision strategies. 1) Max-voting method (also known as 1vs1 decision). 2) Elimination method (also known as DAG decision).

After training, the created SVM model can be seen in the "Machines" tab. The support vectors and the bias (threshold) for each machine can be seen by selecting one of the entries in the first DataGridView. The darker the vector, the more weight it has in the decision process.

To test the model with unseen data

By clicking on the "Classification" tab, we can manually test the Multi-class Support Vector Machine for user drawn digits on the “drawing canvas”, on the right side, you could see that the horizontal bars, higher the bar height then closer the match. Fig 1.2



Insertimage\_9348OS\_01\_02.png

## 1.4.4 Data classification using Kernel Support Vector Machines

The second sample application explains the data classification kernel SVM.

Steps to run the application:

1. Open the application.

(From the installed location. C:\Program Files\Accord.NET\Framework\Samples\MachineLearning\Classification (SVMs)\bin\x86\Release\Classification (SVMs).exe

1. To load the dataset:

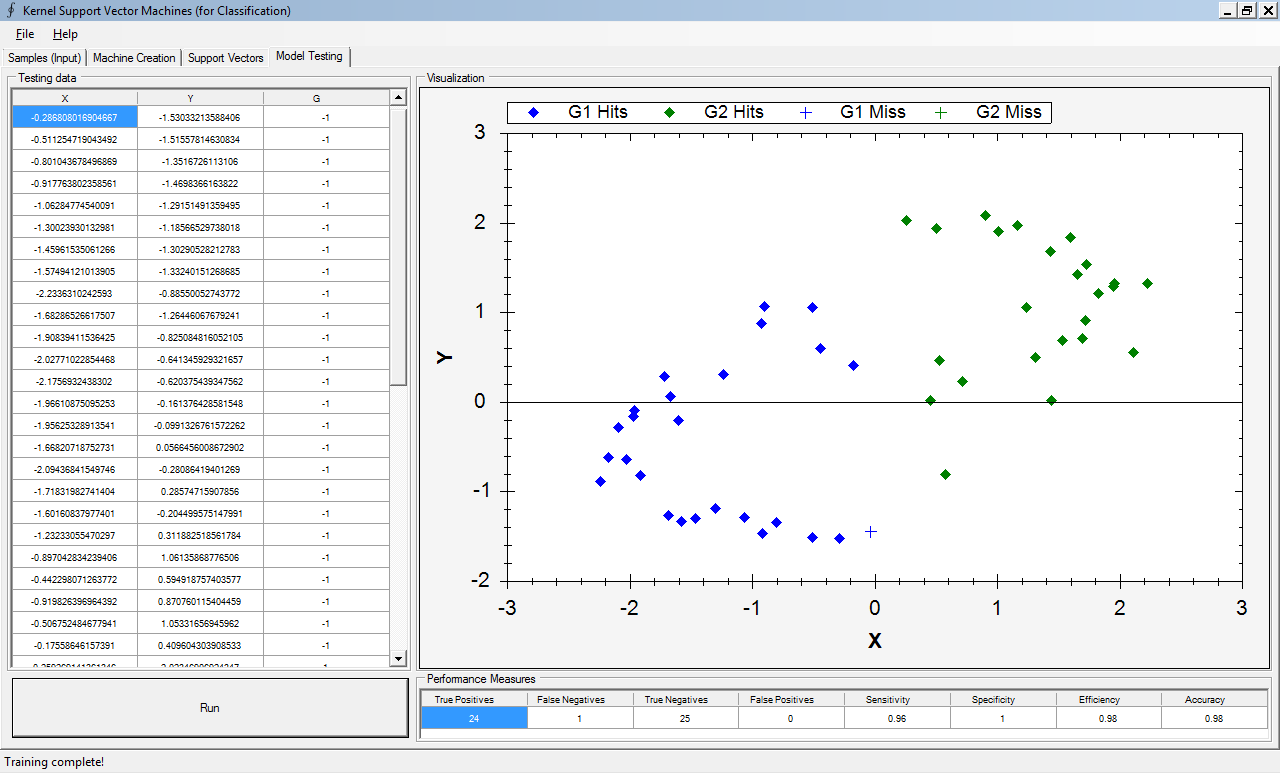
Select File | Open. This would open a dialog box where we should choose the examples.xls file.

Note: The dataset is stored in the examples.xls file under Resources directory of the sample.

1. Then the application opens an another popup, Select the option “Classification - Yin Yang” and Click “Ok”.
2. After the dataset loaded, you could see that there are Blue and Green dots on the scatter plot in the 2-dimentioanl space.
3. Switch to the "Machine Creation" tab
4. Click the button “Create Machine”, now that training process has been completed.
5. To test the model with unseen data

Switch to the "Model Testing" tab, and click “Run”. We could see that performance measures efficiency and accuracy is 0.98. (Fig 1.3)

Fig 1.3



Insertimage\_9348OS\_01\_03.png

The performance metrics are computed by confusion matrix (this module is a part of Accord.Statistics.Analysis). A confusion matrix contains information about expected and predicted classifications done by the SVMs. Performance of such systems is commonly evaluated using the data in the matrix. The following code shows the confusion matrix for a two class classifier.

The below code from MainForm.cs explains how the performance metrics are calculated.

// Extract the first and second columns (X and Y)

double[][] inputs = table.GetColumns(0, 1).ToArray();

// Extract the expected output labels

int[] expected = table.GetColumn(2).ToInt32();

// The output will contain values as predicted by the decision system

int[] output = new int[expected.Length];

// Compute the actual machine outputs

for (int i = 0; i < expected.Length; i++)

output[i] = System.Math.Sign(svm.Compute(inputs[i]));

// Use confusion matrix to compute some performance metrics, In this test,

// 1 means positive, 0 means negative. The result is bound with DataGridView control

ConfusionMatrix confusionMatrix = new ConfusionMatrix(output, expected, 1, -1);

## 1.4.5 Data Classification using Support Vector Machine

In this section, we will see a very simplified example code for Support Vector Machine using c# and Accord.NET Framework.

1. Load the Visual studio (A free version of Visual studio is available at http://www.microsoft.com/express)
2. After Visual Studio has finished loading, click the File | New Project menu item to start creating a new project
3. Select Visual C# | Windows | Console Application
4. Provide a name of the application “ConsoleApplication1”
5. We have to add references to our project. Right-click the project name on the Solution Browser and select "Add References"
6. We will want to add references to the Accord.MachineLearning, Accord.Statistics, Accord.Math and Accord.Core namespaces. Use Browse option and select the path where Accord.NET installed, Be default the framework has been installed all dll files at C:\Program Files\Accord.NET\Framework\Release

Add the below code in the Main function.

using Accord.MachineLearning.VectorMachines;

using Accord.MachineLearning.VectorMachines.Learning;

using Accord.Math;

using Accord.Statistics.Kernels;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

// Load the data set (hard coded values)

double[][] inputs =

{

new double[] { 0, 0 }, // 0 and 0: 0 (label -1)

new double[] { 0, 1 }, // 0 and 1: 0 (label -1)

new double[] { 1, 0 }, // 1 and 0: 0 (label -1)

new double[] { 1, 1 } // 1 and 1: 1 (label +1)

};

// SVM outputs should be given as [-1;+1]

int[] labels =

{

-1, -1, -1, 1

};

// Create a Support Vector Machine for the given inputs

KernelSupportVectorMachine machine = new KernelSupportVectorMachine(new Gaussian(0.1), inputs[0].Length);

// Instantiate a new learning algorithm for SVMs

SequentialMinimalOptimization smo = new SequentialMinimalOptimization(machine, inputs, labels);

// Set up the learning algorithm

smo.Complexity = 1.0;

// Run the learning algorithm

double error = smo.Run();

// It should return zero

Console.WriteLine("error:" + error);

// Compute the decision output for one of the input vectors

int decision = System.Math.Sign(machine.Compute(inputs[3]));

// Show results on screen

Console.WriteLine("Result:" + decision);

Console.ReadKey();

}

}

}

## 1.4.6 Data Classification using Multi-class Support Vector Machines

One of the ways to extend the original SVM algorithm to multiple classes is to build a one- against-one scheme where multiple SVMs specialize to recognize each of the available classes. By using the scheme, the original multi-class classification problem is then reduced to smaller binary problems.

Use the same steps above to create another console application. This is an example code for Multi-class Support Vector Machines.

using Accord.MachineLearning.VectorMachines;

using Accord.MachineLearning.VectorMachines.Learning;

using Accord.Math;

using Accord.Statistics.Kernels;

namespace ConsoleApplication2

{

class Program

{

static void Main(string[] args)

{

// Load the Sample input data

double[][] inputs =

{

new double[] { 1, 4, 2, 0, 1 },

new double[] { 1, 3, 2, 0, 1 },

new double[] { 3, 0, 1, 1, 1 },

new double[] { 3, 0, 1, 0, 1 },

new double[] { 0, 5, 5, 5, 5 },

new double[] { 1, 5, 5, 5, 5 },

new double[] { 1, 0, 0, 0, 0 },

new double[] { 1, 0, 0, 0, 0 },

};

int[] outputs =

{

0, 0,

1, 1,

2, 2,

3, 3,

};

// Create a new Linear kernel

IKernel kernel = new Linear();

// Create a new Multi-class Support Vector Machine with five input,

// using the linear kernel and for four disjoint classes.

var machine = new MulticlassSupportVectorMachine(5, kernel, 4);

// Create the Multi-class learning algorithm for the machine

var teacher = new MulticlassSupportVectorLearning(machine, inputs, outputs);

// Configure the learning algorithm to use SMO to train the

// underlying SVMs in each of the binary class subproblems.

teacher.Algorithm = (svm, classInputs, classOutputs, i, j) =>

new SequentialMinimalOptimization(svm, classInputs, classOutputs);

// Run the learning algorithm

double error = teacher.Run();

// It should return zero

Console.WriteLine("error:" + error);

// Compute the decision output for one of the input vectors

double decision = machine.Compute(inputs[3]);

Console.WriteLine("Result:" + decision);

Console.ReadKey();

}

}

}

## 1.4.7 Data classification with Decision Tree

Decision trees are very simple models, the representation of a classification rule as a tree, with nodes labelled as features, edges labelled as values (or a set of values) of those features and leaves as class labels enables the interpretation of the classifier.

A decision tree uses multistage decision process. Instead of using the complete set of features jointly to make a decision, different subsets of features are used at different levels of the tree.

There are three types of node:

1. Root node: this is the node that has no incoming edges. It is usually at the top of the tree
2. Internal nodes: nodes with one incoming edge and two or more outgoing edges.
3. Leaf nodes: nodes with one incoming edge.

Consider the problem of deciding whether to make a loan to an applicant based on three variables: Applicant’s income, marital status and credit rating. By asking a series of questions about the applicant. The first question we ask concerns the applicant’s income: high, medium or low? If it is high, we decide that a loan may be made. If the income is medium, we ask a further question: What is the applicant’s marital status? If the Applicant is married, we make the loan; if not, then a loan is not made. If the income is low, we ask a question concerning the credit rating of the applicant. If the credit rating is good, we make the loan, if not then a loan is not made.

The series of questions above and their answers could be represented as a decision tree

Fig 1.4

Loan

No Loan

Loan

No Loan

Loan

High

Medium

Low

Married

Not Married

Poor

Good

Insertimage\_9348OS\_01\_04.png

Let’s consider another example.

Create a console application, the steps to create the application has been described in the previous section. And the comments in the below c# code explains the concepts. In this example, we will be using the famous Play Tennis example by Tom Mitchell (1998). In Mitchell's example, one would like to infer if a person would play tennis or not based solely on four input variables. Those variables are all categorical, meaning that there is no order between the possible values for the variable. Refer Fig: 1.5

using System.Data;

using Accord.Statistics.Filters;

using Accord.MachineLearning.DecisionTrees;

using Accord.MachineLearning.DecisionTrees.Learning;

namespace ConsoleApplication3

{

class Program

{

static void Main(string[] args)

{

DecisionTree tree;

double[][] inputs;

int[] outputs;

// consider the example, the famous Play Tennis example by Tom Mitchell (1998):

DataTable data = new DataTable("Mitchell's Tennis Example");

// infer if a person would play tennis or not based solely on four input variables.

//Those variables are all categorical,

//meaning that there is no order between the possible values for the variable

data.Columns.Add("Day", typeof(string));

data.Columns.Add("Outlook", typeof(string));

data.Columns.Add("Temperature", typeof(double));

data.Columns.Add("Humidity", typeof(double));

data.Columns.Add("Wind", typeof(string));

data.Columns.Add("PlayTennis", typeof(string));

data.Rows.Add("D1", "Sunny", 50, 85, "Weak", "No");

data.Rows.Add("D2", "Sunny", 50, 90, "Weak", "No");

data.Rows.Add("D3", "Overcast", 83, 78, "Weak", "Yes");

data.Rows.Add("D4", "Rain", 70, 96, "Weak", "Yes");

data.Rows.Add("D5", "Rain", 68, 80, "Weak", "Yes");

data.Rows.Add("D6", "Rain", 65, 70, "Weak", "No");

data.Rows.Add("D7", "Overcast", 64, 65, "Weak", "Yes");

data.Rows.Add("D8", "Sunny", 50, 95, "Weak", "No");

data.Rows.Add("D9", "Sunny", 69, 70, "Weak", "Yes");

data.Rows.Add("D10", "Rain", 75, 80, "Weak", "Yes");

data.Rows.Add("D11", "Sunny", 75, 70, "Weak", "Yes");

data.Rows.Add("D12", "Overcast", 72, 90, "Weak", "Yes");

data.Rows.Add("D13", "Overcast", 81, 75, "Weak", "Yes");

data.Rows.Add("D14", "Rain", 50, 80, "Weak", "No");

// In order to try to learn a decision tree, we will first convert this problem to a more simpler representation.

// Create a new codification codebook to

// convert strings into integer symbols

// For example, “Sunny” could as well be represented by the integer label 0 and so on

// A codebook effectively transforms any distinct possible value for a variable into an integer symbol

Codification codebook = new Codification(data);

// specify the predictors (variables which will we will use for our decision).

DecisionVariable[] attributes =

{

new DecisionVariable("Outlook", codebook["Outlook"].Symbols), // 3 possible values (Sunny, overcast, rain)

new DecisionVariable("Temperature", DecisionVariableKind.Continuous),

// continuous values

new DecisionVariable("Humidity", DecisionVariableKind.Continuous), // continuous values

new DecisionVariable("Wind", codebook["Wind"].Symbols + 1) // 1 possible value (Weak)

};

int classCount = codebook["PlayTennis"].Symbols; // 2 possible values (yes, no)

// Let's now proceed and create our DecisionTree:

tree = new DecisionTree(attributes, classCount);

C45Learning c45 = new C45Learning(tree);

// Translate our training data into integer symbols using our codebook:

// i.e Extract symbols from data and train the classifier

DataTable symbols = codebook.Apply(data);

inputs = symbols.ToArray("Outlook", "Temperature", "Humidity", "Wind");

outputs = symbols.ToArray<int>("PlayTennis");

// learn a decision tree

double error = c45.Run(inputs, outputs);

// To compute decisions

for (int i = 0; i < inputs.Length; i++)

{

int y = tree.Compute(inputs[i]);

Console.WriteLine(y);

}

}

}

}

Below is the decision tree of the above code. Fig 1.5

No

Yes

Yes

Yes

Sunny

Overcast

Rainy

Normal

High

Normal

High

Yes

No

Strong

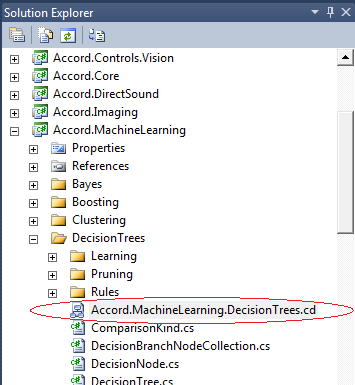
Weak

Insertimage\_9348OS\_01\_05.png

The source code of the Accord.NET has also includes class diagrams that describes relationship among the classes. In order to see the class diagram

1. Open the solution file Accord.NET.sln from C:\Program Files\Accord.NET\Framework\Sources
2. From Solution explorer, open any machine learning project. For example, open | Accord.MachineLearning (Refer Fig 1.6)
3. Double click Accord.MachineLearning.DecisionTrees.cd file, this will open a class diagram of the Decision tree.

Fig 1.6



Insertimage\_9348OS\_01\_06.png

## 1.4.8 Data classification with Naive Bayes classifier

The Naive Bayes classifier is a simple and general-purpose algorithm for most applications. it is based on the Bayesian Theorem. The Naive Bayes Classifier strongly depends on the assumption of the independence. A more descriptive term for the underlying probability model would be "independent feature model". It assumes that the presence or absence of a particular feature is unrelated to the presence or absence of any other feature.

For example, to classify a fruit, an apple is classified by (red color, round shape) and a banana is classified by (yellow color, long shape). A Naive Bayes Classifier considers each of these attributes (color and shape) independently to the probability that the fruit is of a particular class regardless of the presence or absence of the other attributes.

Despite of their naive design is over-simplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations.

In this example, we will be using the famous Play Tennis example by Tom Mitchell (1998). In Mitchell's example, one would like to infer if a person would play tennis or not based solely on four input variables. Those variables are all categorical, meaning that there is no order between the possible values for the variable.

DataTable data = new DataTable("Mitchell's Tennis Example");

data.Columns.Add("Day", "Outlook", "Temperature", "Humidity", "Wind", "PlayTennis");

data.Rows.Add("D1", "Sunny", "Hot", "High", "Weak", "No");

data.Rows.Add("D2", "Sunny", "Hot", "High", "Strong", "No");

data.Rows.Add("D3", "Overcast", "Hot", "High", "Weak", "Yes");

data.Rows.Add("D4", "Rain", "Mild", "High", "Weak", "Yes");

data.Rows.Add("D5", "Rain", "Cool", "Normal", "Weak", "Yes");

data.Rows.Add("D6", "Rain", "Cool", "Normal", "Strong", "No");

data.Rows.Add("D7", "Overcast", "Cool", "Normal", "Strong", "Yes");

data.Rows.Add("D8", "Sunny", "Mild", "High", "Weak", "No");

data.Rows.Add("D9", "Sunny", "Cool", "Normal", "Weak", "Yes");

data.Rows.Add("D10", "Rain", "Mild", "Normal", "Weak", "Yes");

data.Rows.Add("D11", "Sunny", "Mild", "Normal", "Strong", "Yes");

data.Rows.Add("D12", "Overcast", "Mild", "High", "Strong", "Yes");

data.Rows.Add("D13", "Overcast", "Hot", "Normal", "Weak", "Yes");

data.Rows.Add("D14", "Rain", "Mild", "High", "Strong", "No");

In order to estimate a discrete Naive Bayes, we will first convert this problem to a more simpler representation. Since all variables are categories, it does not matter if they are represented as strings, or numbers, since both are just symbols for the event they represent. Since numbers are more easily representable than text strings, we will convert the problem to use a discrete alphabet through the use of a codebook.

A codebook effectively transforms any distinct possible value for a variable into an integer symbol.

For example, “Sunny” could as well be represented by the integer label 0, “Overcast” by “1”, Rain by “2”, and the same goes by for the other variables. So:

// Create a new codification codebook to

// convert strings into integer symbols

Codification codebook = new Codification(data);

// Translate our training data into integer symbols using our codebook:

DataTable symbols = codebook.Apply(data);

int[][] inputs = symbols.ToIntArray("Outlook", "Temperature", "Humidity", "Wind");

int[] outputs = symbols.ToIntArray("PlayTennis").GetColumn(0);

Now that we already have our learning input/ouput pairs, we should specify our Bayes model. We will be trying to build a model to predict the last column, entitled “PlayTennis”. For this, we will be using the “Outlook”, “Temperature”, “Humidity” and “Wind” as predictors (variables which will we will use for our decision). Since those are categorical, we must specify, at the moment of creation of our Bayes model, the number of each possible symbols for those variables.

// Gather information about decision variables

int[] symbolCounts =

{

codebook["Outlook"].Symbols, // 3 possible values (Sunny, overcast, rain)

codebook["Temperature"].Symbols, // 3 possible values (Hot, mild, cool)

codebook["Humidity"].Symbols, // 2 possible values (High, normal)

codebook["Wind"].Symbols // 2 possible values (Weak, strong)

};

int classCount = codebook["PlayTennis"].Symbols; // 2 possible values (yes, no)

// Create a new Naive Bayes classifiers for the two classes

NaiveBayes target = new NaiveBayes(classCount, symbolCounts);

// Compute the Naive Bayes model

target.Estimate(inputs, outputs);

Now that we have created and estimated our classifier, we can query the classifier for new input samples through the Compute() method

// We will be computing the label for a sunny, cool, humid and windy day:

int[] instance = codebook.Translate("Sunny", "Cool", "High", "Strong");

// Now, we can feed this instance to our model

int output = model.Compute(instance, out logLikelihood);

// Finally, the result can be translated back to one of the codewords using

string result = codebook.Translate("PlayTennis", output); // result is "No"

The second category of algorithm are clustering algorithms, the sample solution contains following projects,

Color clustering with K-Means and Meanshift.

Point cloud clustering with Gaussian Mixture Models.

Readers are encouraged to run the these clustering algorithms, The source code contains the enough comments to understand the code. K-Means clustering will be covered in detail at chapter 2 - News clustering.

# Summary

In this chapter, we have just scratched the surface of the subject, we have seen how to run sample applications, and basic concepts of few algorithms. Few more use-cases for building a classifier includes - Email Spam detection, Automatic assignment of article to a category, Automatic detection of natural language (e.g. English), Sentiment analysis. etc.

At this point, you should be able to use ML libraries to do machine learning tasks. If you able to get data of your own or some else, you should split it into training, development and test portions. Using the training and development data, you’ll be able to develop the system then run the resulting model on the test data to get required quality.

From next chapter onwards, we would develop complete machine learning systems which are real word examples. For example, in chapter 2 we would built a "news clustering" system which groups news articles under various categories.