Data Miner VCL 1.2

A quick guide to artificial intelligence and classification

1 Why classification

People’s learning seems slow. The learning experience is difficult to transfer and human failings are potentially always present: one gets tired, forgets quickly, gets sick, is not in the mood…etc.

It is therefore desirable that knowledge might be built automatically by computers. The computers would then be able to help experts or even replace experts. People prefer systems whose decisions they can easily explain and comprehend. Black boxes are not a desirable feature.

There is one standard procedure when we want to detect what is happening or predict or diagnose:

Try to find one single parameter that is able to indicate the state of the process that you are observing. For example, to detect whether the noise is loud or quiet, we can record the noise with the microphone and look at the level of the incoming volume and do not need any further processing. We need just one threshold which will define the limit from which point on we will classify the noise as loud. The approach with one single parameter will always give clear results.

If it is difficult or impossible to derive one single parameter that would describe the problem well enough, we can try using more parameters. This is usually the case where the level of knowledge about the specific field is not yet advanced enough to analytically compute the result. This is where classification algorithms come in.

Classification algorithms are able to compute many probable results across many parameters, if we have sufficient examples with a known outcome from the past. This is sometimes called “**induction**” or “**inference**” (depends on which you school you went to).

We do not need to know, what is the actual analytical relation between the parameters and the outcome. Classification algorithms will internally detect this relation, although they will not be able to express it. If an algorithm is able to express the detected relation, it is called a machine learning algorithm. Because not all relations can always be expressed with signs and words, the classification algorithms have higher classification accuracy then machine learning algorithms. Today’s machine learning algorithms usually express the relations between data and outcome in terms of: IF-THEN sentences, or fairly simple equations.

It also has to be taken into account that conclusions can be soundly drawn from statements which are FALSE! Some of these conclusions may be true and others false i.e. both true and false statements may be drawn from given falsehoods. It all starts with the quality of original gathered data.

2 Classes, attributes, learn and test examples

First, we need to define some terms which we will be using when talking about classification algorithms. Example:

My friends are going sailing. I would like to guess, if they will go or not, based on the history of their decisions:

ID	**Weather**	**Friends**	**Humidity [%]**	**Boat**	**Sailing**
1	good	many	61	small	Yes
2	bad	few	39	medium	No
3	bad	few	82	big	No
4	good	many	75	medium	Yes
5	good	many	66	big	Yes
6	bad	many	77	small	Yes
7	bad	few	92	big	No

And today I know that:
weather is good, there are many friends, air humidity is 80% and the boat is big.

Will my friends go sailing or not? The methods that are able to compute the answer (or estimate the probability) of such things are called **classification algorithms.**

Specific fields of expertise are called **domains**. The more examples we have from each domain the more accurate can be the classification. In our case we are looking at a “domain of sailing”.

The parameters that are used to make a decision (Weather, number of friends, etc…) are called **attributes**. Sailing defines the **class.** There are two possible class values in this case: Yes and No. We have 7 tutorial examples in the table.

The attributes that work with a finite number of values are called **discrete attributes.** (Weather, Friends, Boat). The attributes that work with real numbers are called **continuous attributes.** (Humidity).

Because some algorithms are not able to work with real valued attributes, these attributes have to be transformed to discrete values. This process is called **discretization.** Example:

We have one continuous attribute: Humidity. Discretization would look like this:

If the value falls in	70-100 %	50-70 %	40-50 %	0-40 %
then Humidity is	High	Medium	Low	Very low

Instead of a large number of different values, we now obtained only four different cases:

High, Medium, Low, Very Low.

There are different methods for high quality automatic discretization, but these methods will not be discussed here. The most common one, known from statistics, is a procedure used to build a histogram. It is best, if discretization is not required, because this always results in a loss of classification accuracy.

When discrete attributes are entered into the classification algorithms, they have to be **indexed.** They main reason for this operation is performance. Example: string “High” is replaced with integer 1, string “Medium” with integer 2 and so on…. (For each of the attributes). This lowers storage requirements and comparison time in a computer.

3 Incremental learning

Some classification algorithms assume that all the required data is available at the outset. If an algorithm is able to adapt to the new data, without the need to relearn the whole database again, we say that is capable of incremental learning.

4 Implicit versus explicit

Best classification algorithms store the knowledge implicitly. This means, that they are not able to convert this knowledge to rules, which could be expressed explicitly in the form of if-then sentences or decision trees. If an algorithm is able to express its knowledge in explicit form, it is called a "**machine learning algorithm"**. While extracting explicit rules from large databases may be interesting in some cases, the resulting classification success using these explicit rules is usually lower then when using classifiers, which store the knowledge implicitly.

5 Prior class probability

The initial probability of the classes is called prior class probability. Example:

Sailing:	4 (yes)/ 7 (all the cases) *100 %= 57 %
Sailing:	3 (no) / 7 (all the cases) *100 %= 43 %

Prior probability is important, because it gives us the probability in % of how many cases would be classified correctly, if we could always classify to the majority class. In our case 57% of classifications would be correct, if we could always answer "yes".

But in some cases, making use of prior probability is not a desirable feature. Imagine a set of tutorial examples describing the cause of illnesses in some industrially developed country. The major cause of problems would be completely different from the reality in some third world country. We would therefore not be able to successfully apply this knowledge in the third world country, because the frequency of different cases of illnesses would be different.

Obtained knowledge is therefore not usable in cases where the prior probabilities of classes in the learning domain differs from those in the test domain, if we choose to rely on prior probabilities.

One other example is the case of **prevalence** of the majority class. If there is one class that has very high prior probability (> 90%), then the other classes could get starved, because the prior probability could prevail over all other issues. Imagine a machine that works well 99% of time and we have something like 50 different possible fault cases, representing only 1% of the total acquired learning examples.

If the classification algorithm is capable of ignoring the prior probabilities of classes we will call it to be **prior probability invariant.**

6 Missing values

Sometimes we are not able to obtain values for all attributes of a new learning example. But it might be useful, if could use these examples also, because usually there are not enough acquired examples to describe the domain well enough. Different algorithms are differentially sensitive to missing values.

7 Quality of attributes

We might use more than one attribute to attempt a successful classification, because a single attribute is not consistent with the value of the class. We try to look over many parameters and estimate a class to which the new example may belong.

Example:

	A1	A2	Class
Record 1	0	0	0
Record 2	1	1	1
Record 3	1	0	1
Record 4	0	1	0

In this case the attribute A1 is able to classify to the correct class on its own. Attribute A2 just causes useless noise. If there are many attributes present that are not descriptive they can cause enough noise so that the “good” attribute response gets averaged out. If the classification algorithm is not able to find “noisy” attributes on its own, it is best to try to find them by other means and exclude them from the classification. That procedure is called **pruning**.

It may be worth spending some time, to see which attribute types actually carry some useful information. One measure of the quality of the attribute might be its ability to separate all classes on its own. This will suffice in most of practical cases but in general case things are not so simple. Example:

A specific attribute might seem noisy across all classes but is able to separate well between two classes. Noisy attributes cause problems only, if there are not sufficient acquired examples available (which is often the case).

8 Pruning methods

There are two general approaches to finding attributes which actually improve the classification accuracy:

Pre-pruning: We sort the attributes by quality. We start with the classification of one best attribute and add any others that are able to improve classification accuracy.
Post-pruning: We start with all the attributes and remove all those, whose removal is able to improve classification accuracy.

Ideally, we would be checking all combinations of attributes.

9 Overfitting

If we fit our classification model very close to the acquired examples, the accuracy of the classification will start falling. An example of such a behavior is the parameter K in the K ‘nearest neighbors’ algorithm. If we classify only by looking up the class of one close example (K = 1), the overall classification accuracy will be lower than if we looked at more examples at the same time, because our data is not deterministic. On the other hand, the more determinism the lower can be the K parameter.

Sometimes ‘overfitting’ refers to too many attributes being used and, in such cases, we can help ourselves with pruning. Usually, the more attributes there are, the more examples are required for reliable classification.

10 ‘Myopia’ of attributes

Sometimes it takes two or more attributes to separate classes well enough, because the attributes work in pairs. They are said to be ‘**dependent attributes’**. If the domain contains dependent attributes the classification problem is also called to be **non-linear**. The algorithms that are unable to handle **dependent attributes** are called **naïve** or **myopic,** because they do not see the information that it is present in the data. Naive Bayes is a classification algorithm giving theoretically best classification accuracy, but its main drawback is the fact that it is naive.

Example:

	A1	A2	Class
Record 1	0	0	0
Record 2	0	1	1
Record 3	1	0	1
Record 4	1	1	0

The problem can be handled by the classification algorithm internally or by hand externally. If we detect that two attributes are dependent, we can combine them into one and in that way eliminate the need for the classification algorithm to be able to handle them. Usually people building databases define independent attributes. While in the laboratories, where algorithms are tested on artificial data, the myopia of classification algorithms poses a major problem, the real-world knowledge databases do not seem to be affected by the problem [2].

In our example we use the **XOR** operator to map attributes A1 and A2 to A3:

	A1	A2	A3 = A1 XOR A2
Record 1	0	0	0
Record 2	0	1	1
Record 3	1	0	1
Record 4	1	1	0

We can see that Attribute 3 now completely defines the “Class”. (first table)

11 Feature comparison of some classification algorithms:

	Naive Bayes	Linear classifier	KNN
Incremental Learning	✓*	✓	✓
Forgetting capable	✓	✓	✓
Learning complexity	n · a	n · a	n · a
Classification complexity	c · a	c · a	n · a
Discrete attributes	✓	✓	✓
Continuous attributes	✗	✓	✓
Prior probability invariant	✗	✓	✗
Easily handles missing values	✓	✓	✓

*Not for continuous attributes, because they require discretization and discretization requires complete learning set.

n = number of examples
c = number of classes
a = number of attributes

12 General recommendations and info on different algorithms

Naive Bayes
The veteran of classification. It will furnish best classification results, regardless of the many methods introduced, especially due to the use of m-estimate of probability [4] and Laplace-law of probability. It is very fast in learning and classification. Natively handles only discrete attributes and requires a special discretization engine for real valued attributes. The algorithm will compute probability distributions for all attributes for each class and classify in to the most probable class.

K-NN or K nearest neighbors
Another veteran of classification. Gives best classification results especially with real valued attributes. Its biggest setback is performance. The learning phase is fast, but each classification requires that all the examples are read again. (Naive Bayes requires only that all the classes are read again.) As the name suggests, the algorithm will search up K nearest neighbors and then classify in to the majority class in this selection of K neighbors.

Linear Classifier
Specifically designed to address the problems of prior probabilities of classes and is prior probability invariant. (Solves problems of majority class prevalence and differences in probability distribution of classes between acquired and test examples). As fast as Naive Bayes and also able to handle real valued attributes. It is a compromise between the speed of naive Bayes and the ability of K-NN to handle real valued attributes. The algorithm will compute probability distributions for all attributes for each class and classify to the most probable class. Main difference to the Naive Bayes: Training one class does affect the response of other classes. This allows the response of specific classes to be hand engineered (fuzzy logic design). The response of the class is interpreted as: Average probability across all attributes, that this new example belongs to this class. We classify to the class with the strongest response.

13 Testing of classification algorithms

To test the success of classification algorithms, we split the test databases into two sets: the learning set (66%) and testing set (33%). We design 30 such splits for each database and then run all algorithms on these thirty splits per database. Finally, we compute average accuracy and standard deviation.

14 Which algorithm is best

In most cases Naïve Bayesian will give highest classification results outclassing backpropagation neural networks, k nearest neighbors and many others, while offering best performance, since it is probably the fastest classification algorithm of all in learning and classifying. Its greatest weakness is the requirement for discretization of continuous attributes and myopicy. Linear classifier adds the ability to natively handle real valued attributes, while it achieves almost the same or higher classification accuracy in most test domains. Today there is not one best algorithm for all domains. It is therefore meaningful to try out several algorithms and then select the one that best serves your needs.

15 Classification results interpretation traps

If the number of test examples is low (< 100) then, even if the algorithm misses only one example, the accuracy will fall by 1. Therefore, differences by a few (1..2) percent between classification accuracy of different algorithms should be treated with reservations, if the number of examples is low. Also, if the classification accuracy does not achieve prior probability of the majority class, the results are not very meaningful, because it seems that looking at more attributes does not reveal any additional information. The only information available seems to be the relative frequency of the classes.

16 Recommendations for achieving high classification results:

- Pay special attention to selection of good attributes. Good attributes are able to separate the classes well on their own already.
- Try to keep the number of attributes low. Usually the knowledge databases contain around 30 attributes. Be on the lookout for noisy attributes.
- Myopia is usually not an issue, but if found, it could improve your classification accuracy.
- If you turn off prior probability (Linear Classifier) and the classification accuracy falls below the probability of the majority class, that is a clear indication that the attributes are not good, because you would achieve better classification accuracy by always classifying to the majority class.
- You have to have sufficient learning examples available or carefully design/select those that are available.
- Be careful when relying on prior probabilities of classes. If you have problems with majority class prevalence, use the Linear classifier with prior probability switched off.

17 References:

- Bratko I. (1986,1990) Prolog programming for Artificial Intelligence (2nd edition 1990) , Adison-Wesley.
- Overcoming the Myopia of Inductive Learning Algorithms with RELIEF, Igor Kononenko, Edvard Šimec, Marko Robnik-Šikonja, Applied Inteligence 7, 39-55 (1997))
- Machine learning, Igor Konenko, Ljubljana 1997.
- Cestnik, B (1990) Estimating probabilites: A Cruical task in machine learning, Proc. European Conf. on Artificial Inteligence 90, Stockholm.

18 Credits

Many thanks to prof. David Bell for his suggestions and editing of the english language.