Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Application of Self-Organizing Maps in Astroinformatics

Lopatovský Lukáš

Supervisor: RNDr. Petr Škoda, Csc.

14th May 2014

Acknowledgements

I would like to thank my supervisor RNDr. Petr Škoda, Csc. for his advice and assistance, and to my family and friends for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

In Prague on 14th May 2014 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Lopatovský, Lukáš. *Application of Self-Organizing Maps in Astroinformatics.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2014.

Abstrakt

Astronómia je zdrojom veľkého množstva dát vhodných pre efektívne spracovanie za pomoci počítačov. Zamerali sme sa na data miningový algoritmus samoorganizujuce sa mapy a skúmali sme jeho možné využitie pre spracovávanie veľkých astronomických dat. Zvolili sme najlepšie spomedzi skúmaných implementácii a prepojili ich do jednotnej zapuzdrenej aplikácie. Na vyladovanie algoritmov sme použili známe data sety z úložiska UCI, následne sme sa zamerali na klasifikáciu hviezd na základe charakteristiky ich spektier. Samoorganizujuce sa mapy sa ukázali ako vhodný nástroj pre klasifikáciu. Využitím modelu učenia bez učiteľa umožňujú rozpoznávanie skupín spektier vymykajúcich sa normálu a dokážu odhaliť skryté, človekom len ťažko postrehnuteľné črty.

Klíčová slova Samoorganizujúce se mapy, Astroinformatika, Data mining, Umelá inteligencia, neuronové siete, Učenie bez učiteľa, Astronomická spektroskopia, Big Data

Abstract

The astronomy is source of big amount of data with possibility to be effectively examined by computer. For this purpose we focused on the machine learning algorithm called self-organizing Maps and its use on big astronomical data. We chose the best from considered implementations and wrapped tham in one user-friendly application. For scaling of algorithm we used well known data sets from the UCI repository and then concerned ourselves particularly with the classification of stars based on their spectral characteristics. The self-organizing maps are excelent tool for clustering data in a new way. The unsupervised learning paradigm enabled us to find the groups of self-similar outliers and hidden patterns in data that are hardly possible to find by eye. The algorithm shows very promising results.

Keywords Self-organizing maps, Astroinformatics, Data mining, Artificial intelligence, Neural networks, Unsupervised learning, Astronomical spectroscopy, Big Data

Contents

List of Figures

List of Tables

Introduction

From ancient times people were excited by knowledge. They were amazed by changing, but regularly periodic motions of stars on heaven and tried to find logic in infinitely remote space. They even built gigantic monuments to predict behaviour of cosmic bodies.

In centuries the science of astronomy has developed to form of complex science with lot of branches. As almost everything in the world, the process of modernization has not avoided this field. Now it is hardly to imagine, how it would be without using computational technologies. The computers became a thing of great significance. With its help, the big volume of astronomical data can be automatically processed, so instead of weeks of hard work, the data can be processed in scale of seconds.

However, sometimes the raw computing power is not enough. The intelligence and pattern recognition of human eye is irreplaceable. But what does it mean? Is it possible to find human resources for clustering of stellar spectra? What would it cost and would the received data even suitable for use and error-free?

This is the task we want to look on in this theses. More precisely we decided to make closer look on promising method of data mining: Self-organizing map (SOM) and examine its possible use in astronomy for stellar spectra classification.

The thesis is divided in following chapters:

Astronomy
: The brief introduction into terms as astroinformatics, astronomical spectroscopy and in astronomy broadly used file format `FITS`.

Knowledge discovery
: This is a theoretical introduction of Computer science fields, that are important to be acquaint with before reading following chapters. First the data mining is discussed, then we look deeper inside to neural networks and its learning paradigms.

Self-organizing maps
The closer look to the self-organizing maps (SOM), the type of neural networks. We discuss about its origin and aplications, look deeper into its inner structure and present the possibility of its visualization and error measurments.

Data sets
Introduction of the data sets that we used for testing and introduction of technics we used for their pre-processing.

Implementations
In this chapter are discussed several implementations of SOM algorithm. We mention their pro and contra. We also show some ideas how to boost the speed of the computation and discuss the possibility of parallelization (also using GPU).

Wrapping application
Description of architecture of wrapping application and several implemented features.

Tests
selected comparison tests of SOM implementations, used algorithms and features with focusing on astoinformatics problem, clustering of stellar spectra.

Other unsupervised methods
Introduction of other interesting algorithms that use unsupervised learning paradigm. Discussion about their possibilities for parallelization.

Chapter 1

Astronomy

1.1 Astroinformatics

Astroinformatics raised from natural need for interdisciplinary discipline, in which are represented aspect of astronomy, computer science and information technology. [32] It aims to enrich traditionally used methods of astronomy by finding new reliable and effective ways for processing of massive and complex data sets that can lead to better understanding of nature of astronomical objects.

Astroinformatics is using a new scientific methodology, where the new discoveries results often from outliers, founded by technique of machine learning and data mining, while benefiting from long-term skills of astronomy of building well-documented astronomical catalogs, automatically processed telescopes and satellite archives.

1.2 Astronomical spectroscopy

Spectroscopy is one of the most important astronomical tools, it studies the spectra of stars and other objects in the Universe dispersed according the wavelengths of their light. [26] This study allows to determine their characteristics as chemical compositions, radial velocities, relative motion or physical properties such as temperature or density.

Using Doppler shift can be measured dark matter Using a Doppler shift can be in spectroscopy measured dark matter content of galaxy, characteristics of Binary stars or exoplanets, the mass of galaxies or universe expansion rate.

1.3 FITS format

Flexible Image Transport System (`FITS`) is an open standard digital file format.[10] It is highly used in astronomy as it can store almost every type of data. Many

telescopes use it as a default storage format for their observations. [5]

`FITS` is interesting for a way how it stores data. The file consists of two parts. At the beginning it has a header containing meta-data. This part of the file is stored as ASCII text so it is easy readable also by human. It contains all important information about origin, coordinates, history of the data, etc. In the second part is a image itself stored in binary form.

The image can be classical two dimensional, but it can be also a one dimensional spectrum, usual spectrum, data cube or some other much more complex structure.

Chapter 2

Knowledge discovery

2.1 Data mining

Data mining is field of computer science involving methods of artificial intelligence, statistics and machine learning. It refers to the systematic application of statistical methods to a data set with the aim of new patterns identification. It also involves processing of very large data sets, which can no longer be processed manually. To process the big data the efficient methods should be used. In this case, it is needed to take special care for their time and memory complexity.

The term data mining is sometimes used for so-called "Knowledge Discovery in Databases (KDD)".[16] KDD includes both steps, pre-processing and analysis. Actually, the data mining refers just to second one.

Data mining does not mean just collection, storage and processing of large amounts of data. Mainly it refers to the extraction of previously unknown knowledge that are potentially useful, to determine certain regularities and hidden relationships in datasets. It helps to solve problems, that are not deterministically solvable in polynomial time (NP problems). There is no guarantee that the solution is optimal. However, if using a good model, it can be very similar to it.

2.2 Artificial neural networks

Artificial neural networks (ANN) were invented by Warren McCulloch and Walter Pitts (1943). [28] It is type of computational model based on mathematics and algorithms inspired by nature. It resembles processes in neural cortex of human or animal brain, by modeling cognitive operations at elementary basis, signals that are transmitted among neurons and creations and destruction of connection among them.

More formally ANN are massively parallel computational system with the ability to preserve information and enable its further processing. They re-

semble human learning in way it stores knowledge, via preserving them in inter-neural connections (synapses).

The base of neural network is neuron. Basically it can have several inputs coming from other neurons outputs or from outer environment. The operation that neuron use for transmission of inputs to output is called activation function. It used to be very simple. Actually, the combination of more simple elements is what makes the ANN complex. To compute activation value the connection weights are used. The weights are representations of inter-neural connection strength. 2.1

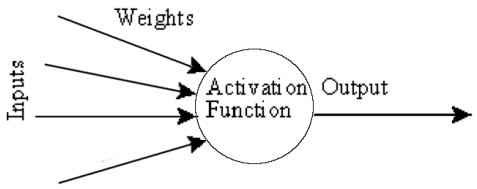

Figure 2.1: Model of artificial neuron. [18]

Prevailingly, the ANN used to be organized in layers. This approach significantly ease the program implementation and mathematical description of network. Usually it consists of input and output layer that communicate with environment. Optionally the one or more hidden layers can be included.

The main advantage of neural network is its ability to learn. It works like the black box. The output is created for every input, without response on inner structure of network. It is important to realize than ANN have ability of generalization, so they can manage the inputs they have never been in contact before.

2.3 Learning paradigms

In data mining we recognize two types of learning.[31] They are characterized by a way they gain information. It can be with a teacher (supervised) or by itself (unsupervised).

2.3.1 Supervised learning

The philosophy of supervised learning is the presence of supervisor during whole process of learning (training). It is needed to have data that have a corresponded correct output to each of inputs. The learning itself looks like following: The input data are given to a model input layer. The model transforms them to output, which is compared with the proper input. The comparison of the outputs is an error, that is consequently used by transforming the model in the way, it corresponds to given data better.

The supervised learning paradigm is used for example in neural networks by back-propagation or by stochastic learning in random forest entropy counting.

2.3.2 Unsupervised learning

This type of learning paradigm does not need any output data. That is why it is often used in case we do not have sufficiently correct output or when it is not possible or very hard to create it. The model using unsupervised learning handles input data, based on the inner concepts. So it is able to find hidden structures and connections in data and compose a model which finds the statistical regularities in them and create their compact and expressive description. Such a data are easier for understanding and characterization then the raw one.

Since unsupervised learning used unlabeled data, there is not exact error measurement system to evaluate the solution. But there exists some methods of reliability measurement.

2.3.2.1 Competitive learning

Competitive learning is a type of unsupervised learning using the winner-take-all approach. From neurons that get signal, the most activated is selected as a winner. The winner neuron is set to value 1, all others to 0. Consequently only the weight with connection to winning neuron are updated.

2.4 Self-organization

Self-organization is a spontaneous creation of structures and system functions. [20] The self-organizing systems can reconfigure themself to changing environment, so they can flexible react to the new arising tasks without need to be specially trained for them. It gives them big amount of robustness and adaptiveness, what makes them ideal for solving of non-stationary or very large domain problems, which solution is not fixed or unknown. The special engineered system is capable to find a structure and connection in data space and help to solve many non trivial problems.

Chapter 3

Self-organizing map

Self-organizing Map (SOM) is a special type of artificial neural networks using paradigm of unsupervised learning. It is a computing method for mapping of multi-dimensional space into two dimensional lattice. It converts the statistically non linear connections among multi-dimensional data to simple relations visualized in low dimensional space. This is reached by using combination of two subsystems of different nature. [23] The winner-takes-all approach combining with plasticity control. The plasticity control is guaranteed by Neighborhood function that manage spreading of activation to best matching neuron neighborhood. It helps to create and maintain the topological organization of neuron. The use of these approaches together has created an effective and robust self-organizing system.

The SOM was first presented by Teuvo Kohonen, a professor of the Academy of Finland in work [22]. That is a reason why it is sometimes called Kohonen SOM.

"The algorithm of SOM is more similar to real brain, than other classical neural networks. It grew out of neural network models of associative memory and adaptive learning. The main stimuli to creation was to explain the spatial organization of brain's function in the cerebral cortex. It was not the first algorithm using power of self-organization. The idea was used by Von der Malsburg (1973) in spatially ordered line detectors and by Amari (1980) in neural field model. Nonetheless, the self-organization power of both algorithms was unsatisfactory."[23]

The SOM is used to reduce dimensions of data and visualize them in human readable way. The problem with more dimensions is that people can not effectively imagine more than three dimensional data, so there exist the techniques that help us understand them. The SOM reducing of dimensionality usually creates 1 or 2 dimensional lattice, which recognizes the similarities of the data and groups the similar data together. Eventually it mediates two things, reduction of dimensionality and display of similarities.

3.1 Applications

The SOMs were used for solving of broad scale of problems. it posses four important properties[14]:

1. **Approximation of the Input Space**
   By feature map we can find a prototype of input space. The goodness of approximation can be measured by quantization error presented in[odkaz].

2. **Topological Ordering**
   Spatial locations of a neurons are similar to particular domain of input pattern. Use of other words, the input vectors near to each other in resolving output grid are supposed to have a short distance in n-dimensional space too.

3. **Density Matching**
   Input vectors that are chosen from space with higher density are mapped onto larger part of lattice, so the output resolution of this vectors is higher then a resolution of vectors from lower density regions.

4. **Feature Selection**
   If input space is a non-linear distribution, the SOM is able to select a set of best features for its approximation. The SOM can be viewed as a non-linear generalization of PCA.

3.2 Network architecture

The map consists of two layers. The input layer is consisting of neurons, which number is equivalent to input data space. Second layer also called output layer or feature map is ordered in topological structure usually rectangular or hexagonal two dimensional lattice. All neurons (called also reference vectors) of lattice are connected by lateral connections. This connections are important for topology preserving and neuron neighborhood recognition.

In the Fig. 3.1 are displayed the connections among input and output layers. In this example is a network compounds from two dimensional input layer and square interconnected lattice output layer. The every neuron of output layer is connected to all neurons from input. Every such connection has its own weight value, from which the weight matrix is created.

3.3 Best matching unit

Best matching unit (BMU) is the label for neuron with a weight vector closest to the input vector. Such a neuron plays important role in process of weight actualization.

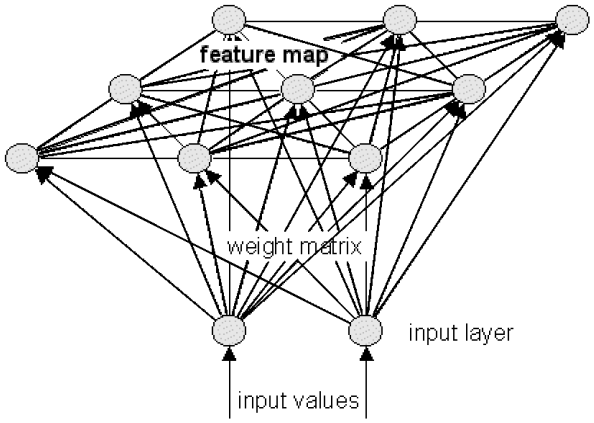

Figure 3.1: SOM topology [8]

The distance among neurons can be computed by any metric. The most common used is the Euclidean. 3.1. But also some other metric can be used, for example Manhattan.

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2+\cdots+(q_n-p_n)^2} = \sqrt{\sum_{i=1}^{n}(q_i-p_i)^2} \tag{3.1}$$

3.4 Neighborhood function

The neighborhood function is important part of SOM. It is responsible for topology preserving ability of map. The topological relations of input space are so maintained.

The neighborhood function is specified by the radius σ. It is a distance from neuron to the other neurons in its surrounding, in which they are consider as neighboring.

For a neighborhood function is often used the Gaussian function

$$\begin{aligned} t &= 1,2,3 \\ \Theta(t) &= exp\left(-\frac{dist^2}{2\sigma^2(t)}\right) \end{aligned} \tag{3.2}$$

or discrete computational easy bubble function. On figure 3.2 is displayed a Gaussian (a) respectively bubble function (b).

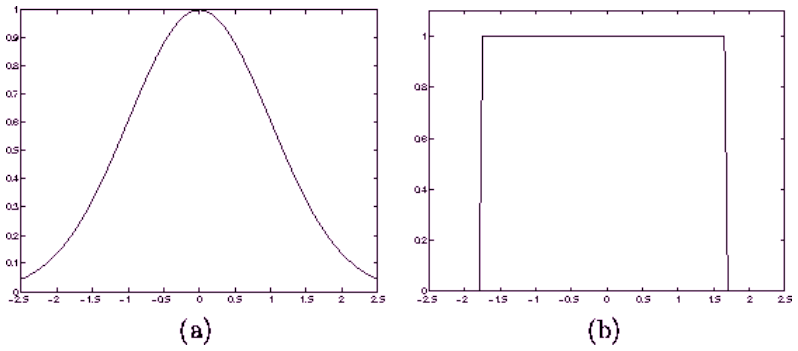

Figure 3.2: Possible neighborhood functions [21]

3.5 Learning process

During the training, the input vectors are selected in random order. All vectors are iteratively used multiple times to expose impact to the model. The time t is defined as discrete variable, increasing with every step in which the one input vector is processed.

Consequently the competition starts and BMU is found. The weight of BMU and its neighboring neurons are transforming in order to be more similar to input. The learning ratio $\mathcal{L}$ is used to count effect of transformation. It is chosen from interval $\mathcal{L} \in (0; 1\rangle$ by learning rate function $\mathcal{L}$.The $\mathcal{L}$ is usually decreasing in time. Such defined function leads map to converge to final state. There are several possible learning rate functions. For example exponential 3.3, inverse time 3.4 or linear 3.5 function, where the letter $\mathcal{L}_0$, is the learning rate of the lattice at time t_0 and the Greek letter lambda λ denotes a whole time constant.

$$t = 1, 2, 3$$
$$\mathcal{L}(t) = \mathcal{L}_0 \, exp\left(\frac{-t}{\lambda}\right) \tag{3.3}$$

$$t = 1, 2, 3$$
$$\mathcal{L}(t) = \mathcal{L}_0 \, \frac{c}{c+t}, \qquad c = \frac{\lambda}{100.0} \tag{3.4}$$

$$t = 1, 2, 3$$
$$\mathcal{L}(t) = \mathcal{L}_0 \, (1.0 - t/\lambda) \tag{3.5}$$

Similarly the neighborhood radius *sigma* is shrinking over time 3.6.

$$t = 1, 2, 3$$
$$\sigma(t) = \sigma_0 \, exp\left(\frac{-t}{\lambda}\right) \tag{3.6}$$

Such defined decrease rates are cosing a regular positioning of neurons at the beginning and its fine centering at the end.

3.6 Algorithm

1. **Initialization**
   Weights of each neuron of the grid are initialized to small random value.

2. **Input Selection**
   The random vector from input space is selected. It is recommended to iterate through all vector multiple times. The random order of every such iteration can stay the same.

3. **Activation**
   Is the most computationally expensive part of algorithm. During this phase is counted a distance from input vector to all neurons on grid and a best matching one is selected and activated(BMU).

4. **Counting of Neighborhood Radius**
   In this phase is counted the radius, which takes informing size of radius of the map. By increasing time it is diminished and at the end it is usually in size of only one neuron.

5. **Weights actualization**
   The part of the algorithm in which the actual map is changing. The weights are changed for every neuron i that is within a neighborhood radius from BMU. The new value of weights W_{ij} is counted for every synapse j according to the following formula.

$$j = 1, , n\mathcal{W}_{ij}(t) \quad = \mathcal{W}_{ij}(t+1) + \Theta(t,i)\mathcal{L}(t)(\mathcal{V}_j(t) - \mathcal{W}_{ij}(t)) \qquad (3.7)$$

6. **Iteration**
   Return to point two, until there are vectors in input.

3.7 Visualization

3.7.1 U-Matrix

Unified Distance Matrix (U-Matrix) is important tool for visualization connections in SOM. It is created by visualization of output neurons, in such a way that new value of neuron is a sum of distances to its neighboring neurons. The acquired values are displayed in a color scale. The neurons that are close together used to have light color the more widely separated are darker. From Fig. 3.3 it is easy to predict two main clusters within data. They are the lighter areas of same color bordered by darker line.

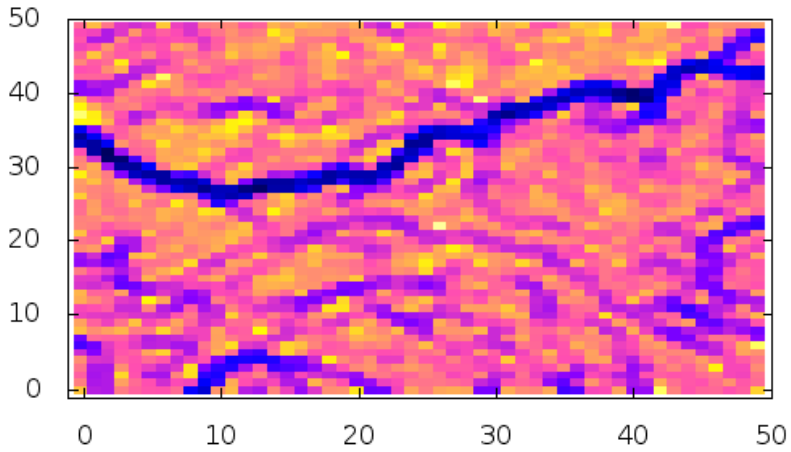

Figure 3.3: Example of U-Matrix

3.7.2 Map of activation

Map of activation visualize output neurons. Every neuron is colored in order how many of input vectors activate it. It can be used for example to determine a data density. After redistributing of input data by its source can be for every of the parts separately created map. Obtained maps can be compared and similarity among sources detected.

3.8 Errors measurements

In unsupervised learning algorithm it is not possible to define absolutely objective measure of errors. Nevertheless it is possible to measure its reliability by following methods[15].

3.8.1 Quantization error

By SOM the data space is quantized into finite number of output vectors. The vector quantization is used to compress data, to remove noise, etc. The (quadratic) quantization error is a squared distance between the input data i and its BMU neuron x. For distance $d(i, x)$ and n input vectors can be quantization error counted like in equation 3.8.

$$Err_{quant} = \frac{\sum_{i=1}^{n} d(i,x)^2}{n} \tag{3.8}$$

3.8.2 Measure of organization

It measures if a resulting map preserves topology. The neighboring neurons in the input space should be projected on the same or neighboring neuron of the map. Various procedures exists how to measure it. For example measure of the distance between all neurons examines if the nearest one is also the neighboring one. This method can be used to detect various twists, butterfly effects, rotations and mirrorings of the map.

Chapter 4

Data sets

The final application will be prevailingly used for spectral analysis. However for basic functionality tests we chose some well known data. For scaling of algorithms and time measuring, we used bigger astronomical data set.

4.1 Iris data

The most popular data set from UCI Repository [11] It is composed of 150 instances, 4 attributes each. Instances are categorized in three classes: Iris Setosa, Iris Versicolour, Iris Virginica. The class Setosa is fully separated. The two remaining intersect each other a bit. 4.1

The small size of data is not suitable for time measuring, however a well known data set is ideal for testing if a algorithm works properly.

The raw `Iris` data have attribute values in various intervals. To make all attributes equal in weight, they must be normalized. For normalization we used equation 4.1, where j is the concrete attribute and $max(j)$ resp. $min(j)$ is a maximal resp. minimal value of this attribute for every instance of data set i. Such a data have values in interval $\langle 0, 1 \rangle$. For testing we used only the normalized `Iris` data.

$$normData_{ij} = \frac{data_{ij} - min(j)}{max(j) - min(j)} \tag{4.1}$$

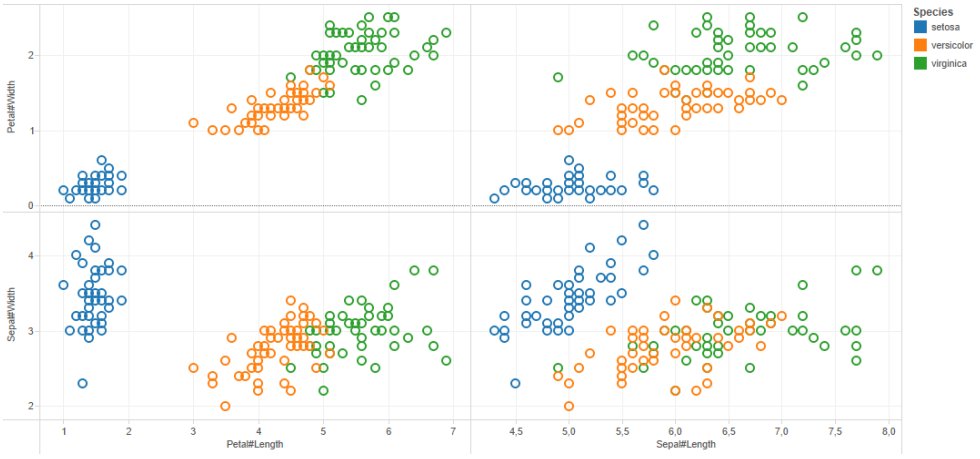

Figure 4.1: Basic visualizations of `Iris` data set.

4.2 Stellar spectra

This data set contains 1696 spectra of Be and normal stars from archive of the Astronomical Institute of the Academy of Sciences of the Czech Republic in Ondřejov.[12] The spectra were obtained with coudè spectrograph of Ondřejov Observatory 2m telescope. The data are divided into four classes dependent on shape of H_α line (see Fig. 4.2).and one special class containing outliers, spectra with some sort of oddity.

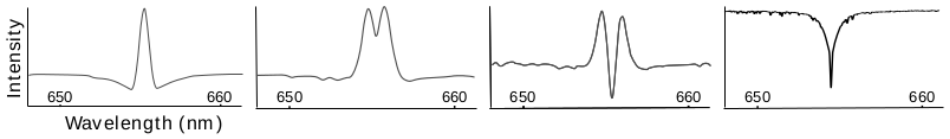

Figure 4.2: Characteristic shapes of spectra.

Originally, every spectrum was stored in separate `FITS` file containing information in 1997 points. The points were not distributed homogeneously for all spectra. The first points of spectra were in interval $\langle 622.266, 626.965\ nm\rangle$. The lasts in interval $\langle 673,504, 678.193\ nm\rangle$. Also the spacings between two neighboring points were heterogeneous.

To arrange data in usable shape we convert it into `CSV` file format. To eliminate the heterogeneous distribution of data we used the method of data binning: We set 1863 points in interval $\langle 626.975, 673.500\ nm\rangle$ with 0.025 nm length of step and linearly approximated the values for every spectrum. (For better understanding See the Fig. 4.3.) The data itself are a little noisy so we assumed the used method of linear binning is sufficient for approximating of values.

We used unnormalized data for testing, becouse the normalization is not needed. We need to look at every spectrum as whole, so we want the bigger values have a bigger impact for a result of classification.

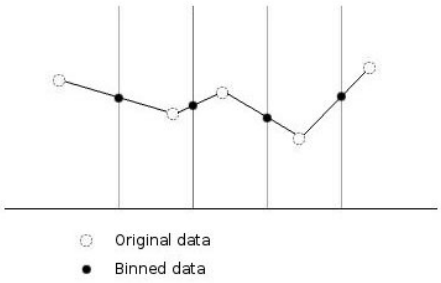

Figure 4.3: Linear data binning

Chapter 5

Implementations

The SOM algorithm has been implemented in many variants. It posses several attributes that in case of changing, have a considerable effect on the output. It makes the algorithm very sensible and any incorrect changing of implementation can make it unstable, topologically incorrect or bad oriented.

5.1 Requirements

When choosing the proper implementation of SOM algorithm, we had to consider the conditions under which it is going to be used.

As a data in astronomy can be really huge [4], the scalability of the algorithm was the main aspects we had to deal with. We had to consider its time performance and ability to process theoretically unbounded size of data.

Also we had to take in mind the environment under which it will be executed. It is Linux based cloud web server `VO-Cloud` (also named after its first application `VO-KOREL`)[29], place where astronomers can in familiar and friendly interface conduct their computationally intensive experiments and use post-processed outputs and visualizations.

For better understanding and ability to read information from SOM output grid, it is necessary to find a way how to visualize it. However, raw visualization of grid is not sufficient. For spectral analyses astronomers need to be as well in touch with input data. So there is a need for another visualization of high dimensional input vectors that would display their connections to BMUs in grid.

Whole application should have user friendly environment. It should be configurable by one easy-readable file and executed by a single command.

We consider following open-sourced implementations of algorithm:

5.2 Rapid Miner

`Rapid Miner` is widely used data mining tool developed by Technical University of Dortmund[6].It offers big variety of data mining algorithms and user friendly interface. The earlier versions and core of the application are available under open-source license. The SOM algorithm implemented in `Rapid Miner` allows to change only a small number of attributes. It use immutable toroidal topology of map and linear learning rate function that are not possible to change.

There exist some enriching plug-ins, however the `Java` based environment showed to be insufficient for bigger data. We encountered a low memory limits and it is not very time efficient as shown in comparing tests (see Tab.7.1).

5.3 WEKA

Waikato Environment for Knowledge Analysis developed by University of Waikato has a plug-in providing implementations of Classification algorithms[7]. It use hight quality Kohonens `SOM_PAK` algorithm [24]implemented in `Java` environment. We decided not to use this implementation for same reason as `Rapid Miner`.

5.4 SOM_PAK

The `SOM_PAK` is a collection of tools for the correct application of the SOM. This original program package was created by the SOM Programming Team of the Helsinki University of Technology with consideration for large problems and incomplete data. Program is available for download at [24]under free license for science. The distribution includes also reference guide.

We decided to use this implementation, because it is created by using a pure `C` language code, what makes it very light-weight and speed oriented. Also we have chosen it for its reliability (It was developed by team led by Professor Teuvo Kohonen, inventor of SOM and it exceeded 1500 literature references by year 1995) and its interesting selectable features.

5.5 Our implementation

For better understanding of a inner structures of SOM algorithm, we decided to create our own implementation. We aimed to make it as fast as possible in a way it could be complementary to `SOM_PAK` algorithm. For performance reasons we chose a language `C++`. The implementation was inspired by algorithm from [13].It uses exponential learning rate function, Gaussian neighborhood function and rectangular shaped lattice, but it is easily extensible for any possible feature.

We experimented with several boosting methods:

For a distance measurement it is sufficient to use square of Euclidean distance. With such a implementation, we spare callings of square root function without any effect to computation as we use the distances only for comparison and it is evident that if $x > y$ than $x^2 > y^2$ is also valid.

Standard algorithm iterate through the whole lattice, when determining the neurons in BMU neighborhood. It is redundant, as a neighborhood radius at the end of training decrease to the size of one. To avoid unnecessary iterations we used the BFS (Breadth-first search) algorithm. This algorithm is not so friendly for compiler optimization, so we tried to find the best possible combination of both approaches.

It is important to remark that such a optimization become less evident with increasing size of input vector. It is so, because of increasing time consuming by actual weight adjusting.

The most time expensive part of algorithm is function for finding BMU. This function time complexity is $\mathcal{O}(mapSizeX * mapSizeY * dim)$, where $mapSizeX$ and $mapSizeY$ represent sizes of mapping grid and dim is a dimension of input vector. This function is called for every input vector in each iteration.

To boost speed of BMU finding, we consider following algorithms:

5.5.1 Spatial access methods

It exists a group of methods for finding a closest point in n-dimensional space. For example structures as r-tree or k-d tree (short for k-dimensional tree). Using these methods we can decrease the number of comparisons among neurons in one BMU finding from usual $\mathcal{O}(mapSizeX * mapSizeY)$ to $\mathcal{O}(\log(mapSizeX * mapSizeY))$.

Unfortunately, as mentioned in [19], the real speed boost gained by algorithm is quickly diminished by increasing number of input vector dimensions. This makes it not suitable for high-dimensional astronomical data.

5.5.2 Probing algorithm

It is the algorithm based on gradient descent search approach. It was presented in article written by J. Lampinen and E. Oja [25].[29] Algorithm consists of two step.

1. Choose the one neuron from SOM lattice as a candidate.
2. Examine its grid neighbors and compare its distances to an input vector. While it exists closest neuron, make it new candidate and iterate the whole process, otherwise terminate the search.

For better results it is recommended to repeat whole procedure 2–6 times.

The algorithm is not a general purpose nearest neighbor method, since there is no way to assure that the result is the true nearest neighbor. However, as mentioned in the article, the real goal of the self-organizing is not to find the exact nearest neighbors but to develop a topological feature map. The algorithm will make errors in finding the closest reference vector to an input vector, but in such a consistent way, that it will not disturb the self-organization.

The algorithm designed in such a way does not suffer from any slow-down in high dimensions and offers significant speed boost for large grids. That are reasons, we tried to implement the algorithm.

For choosing of initialization neuron we use random strategy and Friedman's algorithm for finding the nearest neighbor [17]. Friedman's algorithm chooses the neuron that is the closest in projection using one random dimension.

5.6 Parallel implementations

The increasing number of data naturally arises the question of parallelization. In this section we discuss three possible approaches of parallel implementations.

5.6.1 MapReduce

The `MapReduce` programming paradigm is commonly used when handling big data. The idea consists of dividing problem into more same small subproblems, that are distributed into separate devices. The solutions are consecutively sent to main machine. It combines them in some way to get a final result.

The SOM algorithm has an iterative nature. Every step is directly dependent on the values computed in previous one. It makes the classical SOM algorithm inappropriate for `MapReduce`.

However, it exists variants of SOM adapting to the `MapReduce`. We refer about it later in chapter 8

5.6.2 Thread parallelism

The SOM algorithm has naturally parallel character. It makes it ideal for thread based parallelism. Parallelization can be used for redistributing work of BMU finding as well as weight actualization. We parallelized the implementation using OpenMP directives.

5.6.3 GPU

Similarly as a thread parallelism, the SOM is very suitable for use on Graphics processing units. It was achieved the speedup factor about eighty six (compared to CPU) with NVIDIA's `CUDA` architecture on a Tesla C1060 GPU.[27] This implementation of SOM is quite straightforward. The challenging part is to fit it for a very large map sizes or large number of attributes that would be to big for a memory of single device. In this case it would be needed to split the data according to the memory availability and make computations independently for all of the blocks.

Chapter 6

Wrapping application

The application is controlled by a python script reading of the configurating information from a single user written file. For implementation of wrapper we chose the `Python` as it seems to be an ideal language for making such a job, because of its simple and clear syntax. All computationally expensive and critical parts are implemented in `C/C++`. This language is a logical choice as the application is meant to be used on a big amount of data.

The application used has a following characteristics:

- From the user view looks the application very simple6.1.
- The making of changes, adding new or removing existing features and implementations is very easy to do.

The final application consist of tree main parts.

1. **Data pre-processing**
   The data files are convert to the form used by algorithms.
2. **Core application**
   The selected algorithms are performing computations on input data sets.
3. **Output post-processing**
   The result data files are used for error counting and visualizations creating.

6.1 Configuration File

Configuration file is in a `JSON` format. It helps the user to define proper input files, it gives him a possibility to choose, which output files will be generated, and set the parameters of SOM algorithm. Furthermore it gives him possibility

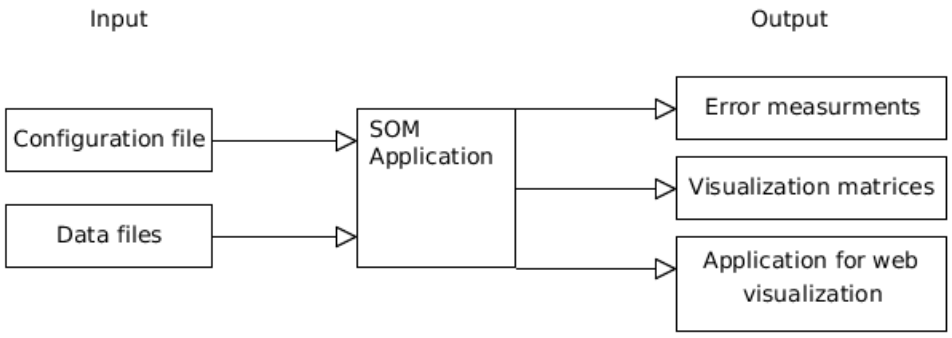

Figure 6.1: Outer look of application

to choose which output files will be generated and set the algorithm to run in the parallel mode.

The complete guide for input parameters can be find in Appendix C.

6.2 Input of data

Input data can be in `CSV` format. The `FITS` format can be converted using included conversion programm. The data sets can be separated in arbitrary number of files. They common size has no limitations as the files are read sequentially. However, the average time of running is slightly bigger when using more files, as a data have to be for each iteration repeatedly read from the disk.

Optionally, it can be added the file containing data classes and names. It is option that can help recognize quality of chosen algorithm (by visualizating it). This data files should be equivalent to main data in a way how they are separated in files and rows.

It is important to add, that class and name data are not used in training phase of algorithm. They are exclusively used for visualization.

6.3 Core application

In this phase the SOM algorithm is executed. The concrete implementation is chosen according to configuration file defined by user. As each of the implementations contains some specific features, it does not support every possible combination of input parameters.

This is usually the most time-consuming part of algorithm. After terminating, it outputs a trained map of feature vectors.

6.4 Post-processing

The post-processing phase is common for all used algorithms. All of the files creating in this step are stored in the file "results".

6.4.1 Errors

The program counts the quadratic quantization error. In variant with information about classes it is also counted error that reflects how many of the input data vectors associate with a neuron that associate with more input vectors of another class. This error can be helpful while scaling the algorithm.

6.4.2 Visualization Matrices

The output matrices are visualized by program `Gnuplot`. The program creates usual U-Matrix and a matrix created by plotting information about data (optionally also with classes) in a way they associate with neurons (we called it association matrix).

6.4.3 Web visualizations

To add a interactivity in output visualization we created a small web application that enable interactively plotting of spectra for spectral classification. It is also possible to examine artificial spectra of feature vectors and find a spectra associated to them as well as examining the relations in visualized matrices.

For plotting of spectra we uses `JavaScript` library `dygraphs`.[2] Other interactivity is created by using `PHP` and `HTML`. The variable parts of `HTML` and `PHP` code are generated in post-processing via `C` program.

Chapter 7

Tests

In this chapter we show some of the test, we used for choosing of implementations and scaling of algorithm.

We focused mainly on run-time measurement and on the impact of spared time on quality of the solution. The quality can be compared using visualized matrices or quadratic quantization error. It is important to note that the value of quantization error is not an absolute, rather a relative indicator of quality, as it is strongly dependable on data set characteristic and size of the neuron lattice.

We decided to add just the most important tests, as adding of visualizations in the text of thesis is very space consuming.

When not stated otherwise, the test were performed on Pentium(R) Dual-Core CPU T4400, 2.20GHz.

7.1 Comparing of implementations

When choosing implementations for wrapper aplication we tested several alternatives. We focused mainly on time of running and also on quality of results.

While the quality was comparable, the time performance differs a lot. As we expected, the `C/C++` written implementations showed to be faster. We trained a map of size 30x30 using stellar spectra data set, eight iterations of data and neighborhood radius of size 30. For results of time measurement see Tab.7.1.

	learning rate function	time [s]
Rapid Miner	linear	215
WEKA	linear	202
SOM_PAK	inverse time	116
SOM_PAK	linear	103
Our Implementation	exponential	68

Table 7.1: Time comparison of implementations

7.2 Learning rate functions

In this test we compared learning rates function of algorithms used in wrapper application. For testing on `Iris` data was used 10x10 grid, ten iterations and neighborhood radius of size 10. For testing on stellar spectra were used same parameters as in section above.

Both implementation show the high performance of clustering. The test reveals a different nature of algorithms in `SOM_PAK` vs. our implementation. Our algorithms deploys input data more regularly over the whole area of lattice, whereas the `SOM_PAK` implementations grouped them denser into clusters. This is the reason for differences in measured error rates (see Tab.7.2). The lower error in our implementation is caused by less number of input vectors associated to one neuron so it can be better approximated.

To summarize we can say that our implementation uses more flexible grid and can recognize finer differences in data set, however sometimes it can be more useful to use more stable `SOM_PAK` implementation.

In Figures 7.1 7.2 7.3 you can see the association matrices from the `Iris` data experiment, similar results were obtained on stellar spectra data set.

	learning rate function	quant. error (iris)	quant. error (stellar spectra)
SOM_PAK	inverse time	0.035	104.19
SOM_PAK	linear	0.018	86.92
Our Implem.	exponential	0.005	7.78

Table 7.2: Time comparison of implementations

7.3 Data binning

It is rare to find data that are ready for training without any form of preprocessing. So the Spectra data set needed to unify the values of light wavelength. We achieved it by using the technique of linear data binning in Section 4.2. After its application the quality of self-organization rapidly increased.

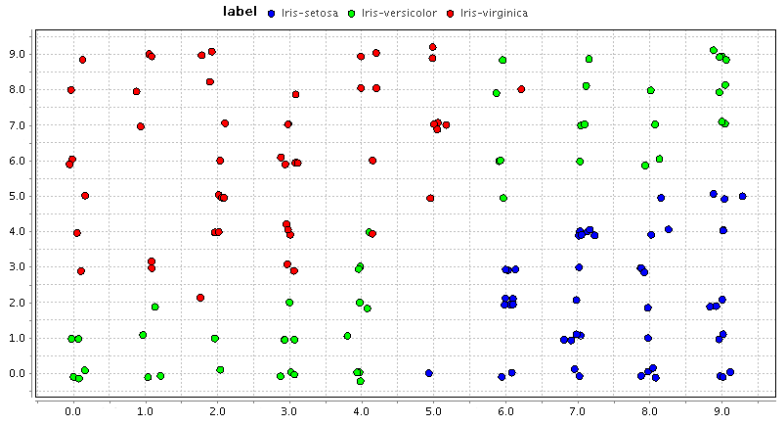

Figure 7.1: Association matrix of `Iris` data set after using exponential learning rate function.

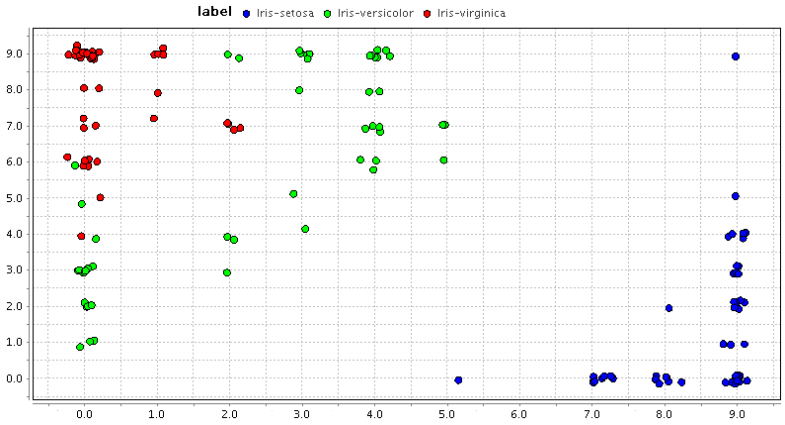

Figure 7.2: Association matrix of `Iris` data set after using linear learning rate function.

The experiment was conducted on the reduced stellar spectra set (we omitted fifth group of outliers). We used exponential learning rate function with starting value 1.0, eight iterations of learning cycle and Gaussian neighborhood function. For illustration you can see association matrices of the raw data in Fig. 7.4 and of the binned in Fig. 7.4. Comparing the figures, it is clearly visible that the binned input vectors are classified much more precisely. The whole quantization error decreased from 12.70 to 2.99 when using binned data.

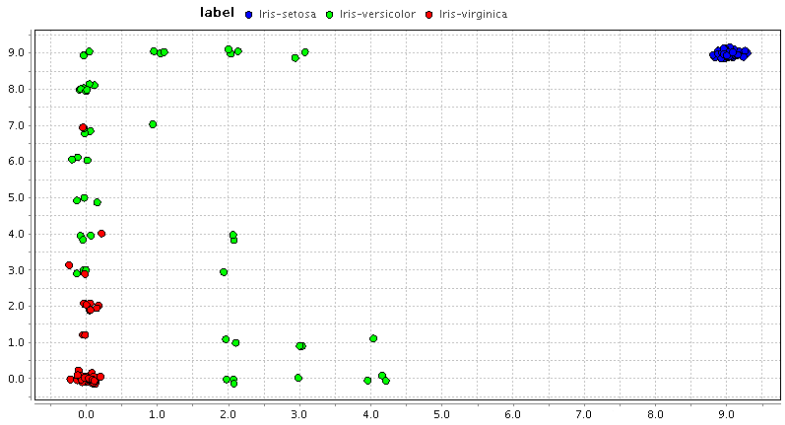

Figure 7.3: Association matrix of `Iris` data set after using inverse time learning rate function.

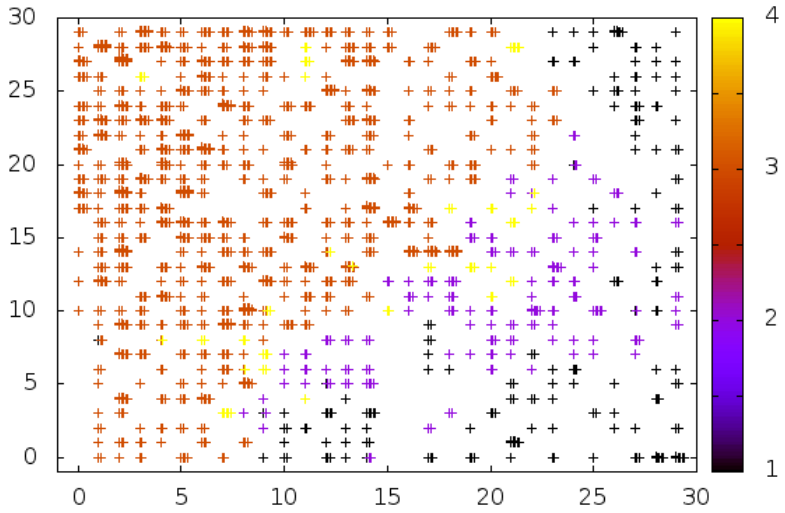

Figure 7.4: Association matrix of raw data.

7.4 Outliers classification

One of the advantages of unsupervised learning paradigm against the supervised is its ability to find previously undefined classes of data. For example in astronomy, finding the group of spectra with similarly odd features can lead to interesting discoveries of new atypical or yet not observed groups of objects.

For experiment we used same parameters as in previous section, except the data set that contains also fifth group of outliers.

The experiment confirmed SOMs ability of organization and clustering. The outliers are grouped in four main clusters (see Fig. 7.6). Each of them contain the spectra of the same characteristic. In the Fig. 7.7 there is an

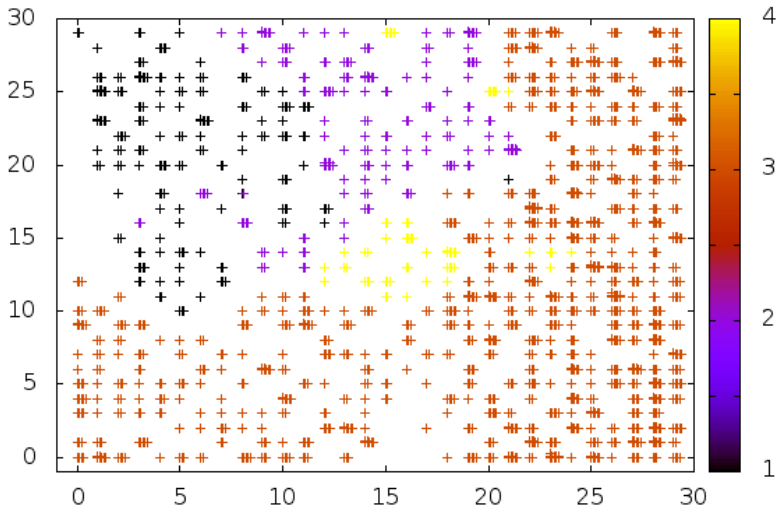

Figure 7.5: Association matrix of binned data.

average representative of each cluster marked with corresponding letter.

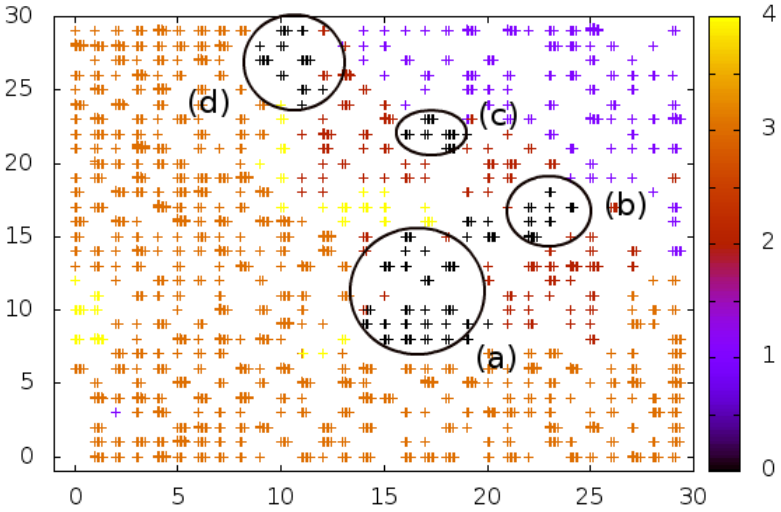

Figure 7.6: Four main clusters of outliers.

7.5 Timing Test of Functions Determining Neighborhood

We implemented two functions for determining Neighborhood. The algorithms are mentioned in 5.5. As we assumed, the time gain of BFS algorithm is not

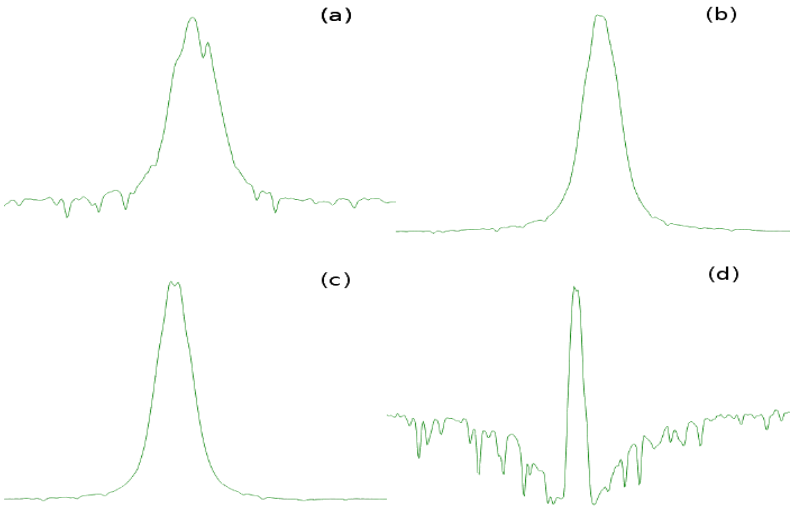

Figure 7.7: Average representatives of outliers clusters.

evident when using with high dimensional vectors. Actually, it is even a bit slower.

The most visible advantage of BFS is at the end of algorithm, when the size of neighborhood radius shrink. We tried to find ideal ratio of algorithm combination, so it gives the best performance.

For testing we used `Iris` data set, exponential learning rate function, lattice of size 50 x 50 and for easier measurement of time, 1000 iterations over data. In this measurement we are interested only in time performance. The 1000 iteration over 150 vectors means the same as for example 10 iteration over 15000. The results are plotted in Fig. 7.8. The ratio is labeled in percents. For example the ratio 10% means, that we used standard algorithm for first ten percent of time and BFS for the rest.

The application is mainly created for highly dimensional data, so we decided to set the stable ratio of 100%. Although we showed the tiny time gain while training low dimensional data it is too small to have visible effect.

7.6 Probing Algorithm

In this test we compared classical and probing algorithm. We used lattice of 30x30 with initial size of neighborhood 30, exponential learning rate function and 8 iterations of SOM algorithm.

We combined algorithms in various ratios, starting with classical algorithm and ending with probing. However, the quality of trained map when using probing algorithm (does not matter, how many percent) was clearly worst (see Fig. 7.9 resp. 7.10 for classical resp. probing algorithm) In this example we

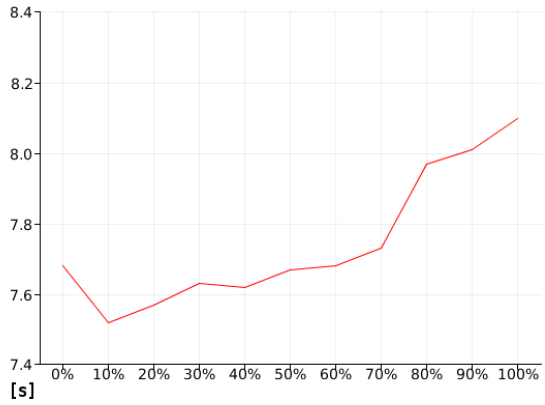

Figure 7.8: running times for different ratios of functions determining neighborhood.

used four iteration of algorithm.[1]).

We tested random and Friedman's strategy for choosing initialization neuron. Both of them showed the result of same quality. On the contrary increasing number of iteration boost the resulting organization of map. Naturally more iterations consume a longer period of time (Classical version: 57.9s, 1 iteration: 13.1s, 4 iterations: 20.8s, 10 iterations: 31.1s). Observations suggest that a quality of solution is proportional to running time, so the probing algorithm is good alternative, in case we prefer time sparing and do not need highest quality or in case we have an easy separable and predictable data set.

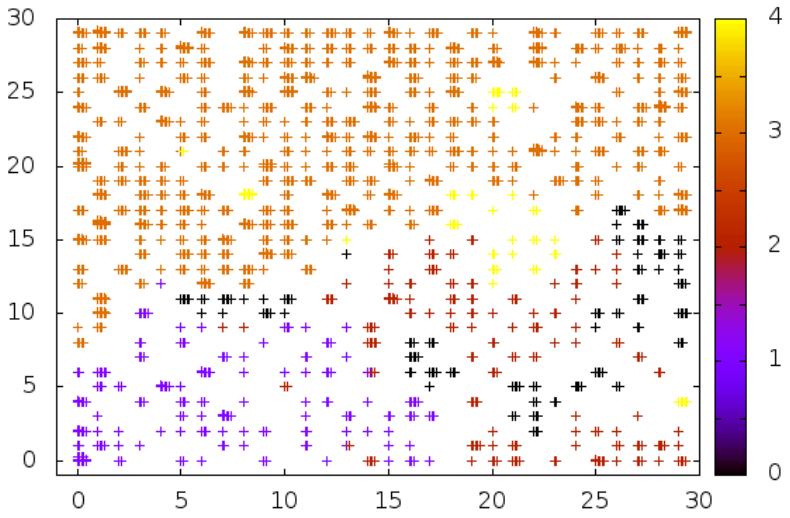

Figure 7.9: Association matrix after using of classical algorithm.

[1]Do not confuse inside iteration of probing algorithm with iteration of SOM.

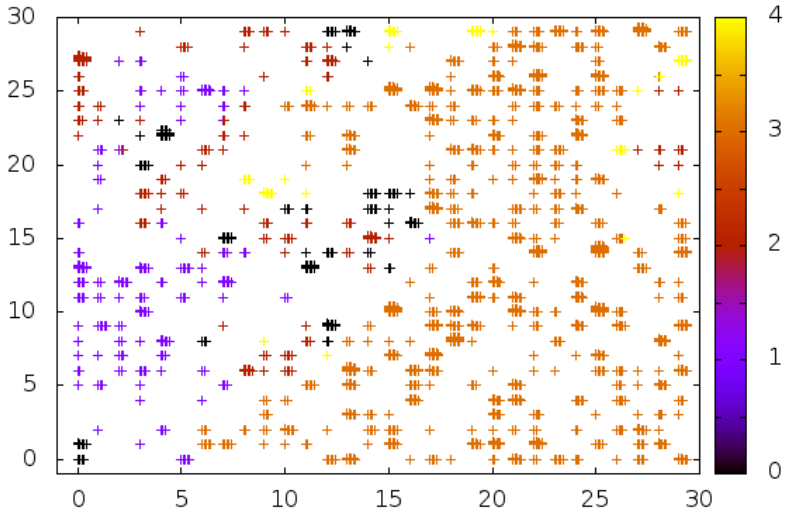

Figure 7.10: Association matrix after using of probing algorithm.

7.7 Parallel implementations

For testing of acceleration of parallel algorithm we used stellar data trained using map of size 30x30 in eight iterations. For performance measurment we used 12-core computional cluster STAR [3].

After parallelization of part where BMU is found, the algorithm using 10 threads accelerated almost four times. For the information about other number of threads see a graphs on Fig. 7.11, The red line in the image (a) is a run-time of training in seconds, in the image (b) is showed how much is a parallel implementation faster than sequential.

We can count that unparallelized part of algorithm takes now something around 3.5 seconds (app. 18% of run-time), while the time of BMU finding becomes only 1.5 seconds (using 10 threads). So we decided to parallelize also the second most time consuming part, the weight actualization.

By parallelizing weight actualization we get the time of unparallelized part only something about 0.6 seconds (app. 3%). The measured times and accelerations for other number of threads is marked by blue line in graphs.

By comparing both lines we can see that acceleration of full parallel algorithm is not very significant for small number of threads. We assumed it is caused by initialization cost of openMP library.

It is interesting to notice that by using 10 threads and the equivalent parameters, we could process over 2.5 million spectra in one hour.

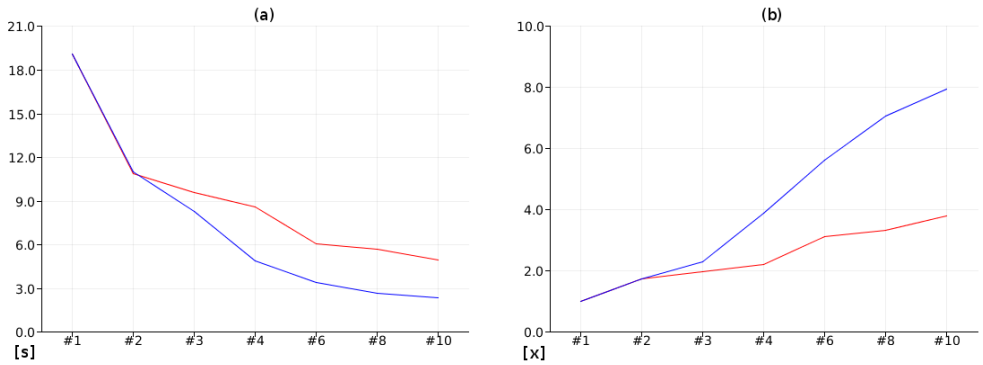

Figure 7.11: Measures of parallel implementations.

Chapter 8

Other unsupervised methods

In this section we try to compare the SOM algorithm with other methods using unsupervised learning paradigm. It is impossible to set the tests that would objectively compare their performance and suitability. In unsupervised learning we lack unified objective output as every method is using its own strategy of approximation of solution.

However we can compare the ways how the algorithms handle data and we can discuss its feasibility for massive parallel environment.

8.0.1 GSOM

Growing self-organizing maps (GSOM) is a variant of SOM solving the dilemma of choosing appropriate grid size. It usually starts with only a few nodes and dynamically adds a new one (see Fig 8.1).

This technique can better fit the data space as it develops in shapes depending on the structure of clusters. [9]

Unfortunately the irregular shape of map condemns the GSOM for being very hard to parallelized.

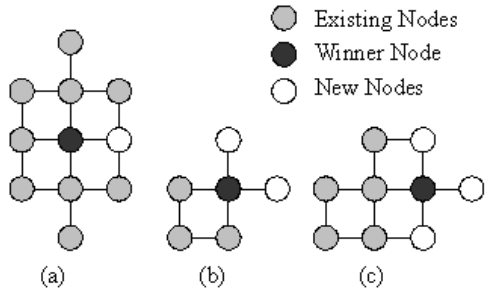

Figure 8.1: Various strategies of GSOM growing.

8.0.2 CSOM

Concurrent self-organizing maps (CSOM) [30] are a collection of more small SOM. Every small map use the usual SOM algorithm, however it is only partially unsupervised learning method as every map is trained using only exact class of data. After such training the input vectors are classified by the map in which they reach minimal quantization error.

The advantage of CSOM is its great possibility for parallelization also by using of MapReduce paradigm. The maps can be easily divided among nodes in cluster. In the reduce phase of algorithm would be simply chosen the result with minimal quantization error.

The possible implementation for GPU would be very similar to SOM algorithm. Even it would not encounter the problems with large maps as a map in CSOM used to be very small.

By using a CSOM we loose the unsupervised learning advantage of finding previously unknown features, but it has a nice ability for classification of evenly distributed classes.

We assumed this technique could be confusing when using over classes containing data of different densities. For example an input vector belonging to spare class, which is close to the dense one can be easily miss-classified.

8.0.3 K-means clustering

K-means clustering is like SOM a method of vector quantization. The input parameter of the algorithm K is a number of clusters that are expected. It is needed to set it at the beginning, so when the number of clusters is unknown, it is needed to start the algorithm multiple times and find the best result (for example using quantization error).

The algorithm starts with random selection of k "means". To each of the data inputs is found the nearest mean. From the data associated to neuron is counted a centroid that becomes a new mean for next iteration. This step is iterated until it reaches the convergence to a local minimum.

Possibility of parallelization are similar to SOM algorithm. Because of its iterative nature it is not possible to use MapReduce paradigm, however parallelization of nearest mean finding is easily implementable for threads or for GPU.

8.0.4 DAME-grid

`DAME` (DAta Mining & Exploration)[1] is a web based data mining infrastructure developed to deal with massive and distributed datasets and perform complex knowledge extraction task. It offers broad scale of machine learning methods that can be examined via a web application `DAMEWARE` (Web Application REsource of DAME).

`DAME` offers many interesting algorithms of unsupervised learning. Self-organizing maps with 2-stage clustering and also all above mentioned methods: CSOM, GSOM and K-means.

We examined above mentioned methods, however only using low dimensional `Iris` data as an upload of data of higher dimension failures permanently without leaving any error message. It is probably because it is still in beta release. We appreciate the DAME project, especially its big variety of algorithms (also implemented for GPU). We also liked a high number of parameters that can be customized while doing experiments.

Conclusion

This bachelor thesis provides a detail description of self-organizing map (SOM) algorithm and its possibilities for solving problems in newly arisen science of informatics. As a astronomy is source of large data sets we specially focused on its performance under increasing number of data.

We compared several existing implementation of SOM, from which we chose a `C`-written Kohonen implementation `SOM_PAK` and wrapped it together with our own created complementary parallel implementation in `C++`. This application take special look for time effectivity and is able to process theoretically unlimited number of data files.

By making a tests using astronomical data sets, we found that SOM is promising method, mainly for finding of previously unknown or in some way odd spatial objects, what can eventually lead to discoveries that will help us to understand the character of universe.

We also examined and tested several interesting boost possibilities and heuristics for SOM algorithm, however we find out that most of them do not work ideally for high dimensional data, but they can be interesting consideration for making of experiments.

At the end we briefly discussed and compared several similar algorithms and considered their feasibility for massively parallel environment.

The bachelor thesis was presented at the conference "New challenges in astro- and environmental informatics in the Big Data era" in Szombathely, Hungary in May 2014.

Future

The created application is planned to be set for VO-Cloud environment. It should be done easily as we have taken this into account during its development. It also needs to be tested more properly and to be discussed with astronomers to know their specific needs.

The application is easily modifiable so there could be arbitrarily added new features or implementations.

In longer horizon it could be used on data obtained from LAMOST telescope [4] in China that produces 4000 spectra in a single exposure and have created catalog containing over two millions spectra.

Of course, the application do not have to be narrowed only for use in astronomy, it is also possible to use it for objects recognition or classification in others branches.

Bibliography

[1] DAME — DAta Mining & Exploration) [online]. http://dame.dsf.unina.it/index.html, [Cited 2014-05-12].

[2] Dygraphs [online]. http://dygraphs.com/, [Cited 2014-05-12].

[3] HPC Star [online]. http://star.fit.cvut.cz/dokuwiki/star:hpc:description:node, [Cited 2014-05-12].

[4] LAMOST — Large Sky Area Multi-Object Fiber Spectroscopic Telescope [online]. http://www.lamost.org/public/?locale=en, [Cited 2014-05-12].

[5] National Aeronautics and Space Administration: FITS. [online]. http://fits.gsfc.nasa.gov/fits_documentation.html, [Cited 2014-05-12].

[6] Predictive Analytics, Data Mining, Self-service, Open source - RapidMiner [online]. http://rapidminer.com/, [Cited 2014-05-12].

[7] WEKA Classification Algorithms [online]. http://wekaclassalgos.sourceforge.net/, [Cited 2014-05-12].

[8] Cartes de Kohonen & loi de Grossberg — TPE 2012 [online]. http://tpebrodinserise.wordpress.com/kohonen/, March 2012, [Cited 2014-05-12].

[9] Alahakoon, D.; Halgamuge, S. K.; Srinivasan, B.: Dynamic Self Organising Maps with Controlled Growth for Knowledge Discovery. *IEEE Transactions on Neural Networks*, 2000.

[10] Allegrezza, S.: Flexible Image Transport System: a new standard file format for long-term preservation projects? [online]. http://www.vaticanlibrary.va/moduli/Allegrezza_EWASS2012.pdf, July 2012, [Cited 2014-05-12].

[11] Bache, K.; Lichman, M.: UCI Machine Learning Repository. "http://archive.ics.uci.edu/ml", 2013.

[12] Bromová, P.; Bařina, D.; Škoda, P.; etc.: Classification of Spectra of Emission-line Stars Using Feature Extraction Based on Wavelet Transform. In *Proceedings of 23rd Annual Astronomical Data Analysis Software and Systems (ADASS) Conference*, 2013, pp. 1–9999.

[13] Buckland, M.: Kohonen's Self Organizing Feature Maps [online]. http://www.ai-junkie.com/ann/som/som1.html, [Cited 2014-05-12].

[14] Bullinaria, J. A.: Self Organizing Maps: Algorithms and Applications [online]. http://www.cs.bham.ac.uk/~jxb/NN/l17.pdf, 2004, [Cited 2014-05-12].

[15] Cottrell, M.; de Bodt, E.; Verleysen, M.: A statistical tool to assess the reliability of self-organizing maps. In *Advances in Self-Organising Maps*, Springer, 2001, pp. 7–14.

[16] Fayyad, U. M.; Piatetsky-Shapiro, G.; Smyth, P.: From Data Mining to Knowledge Discovery in Databases. *AI Magazine*, volume 17, no. 3, 1996: pp. 37–54.

[17] Friedman, J. H.; Baskett, F.; Shustek, L. J.: An Algorithm for Finding Nearest Neighbors. *IEEE Transactions on Computers*, volume C-24, October 1975: pp. 1000–1006.

[18] Gershenson, C.: Artificial Neural Networks for Beginners [online]. http://turing.iimas.unam.mx/~cgg/cogs/doc/FCS-ANN-tutorial.htm, [Cited 2014-05-12].

[19] Guttman, A.: R-trees: a dynamic index structure for spatial searching. In *ACM SIGMOD*, 1984, pp. 47–57.

[20] Haken, H.: Self-organization. *Scholarpedia*, volume 3, no. 8, 2008: p. 1401.

[21] Hollmen, J.: Self-Organizing Map (SOM) [online]. http://users.ics.aalto.fi/jhollmen/dippa/node9.html, March 1996, [Cited 2014-05-12].

[22] Kohonen, T.: Self-Organized Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, volume 43, no. 1, 1982: pp. 59–69.

[23] Kohonen, T.; Honkela, T.: Kohonen network. *Scholarpedia*, volume 2, no. 1, 2007: p. 1568, revision 122029.

[24] Kohonen, T.; Hynninen, J.; Kangas, J.; etc.: SOM_PAK: The Self-Organizing Map Program Package [online]. `http://www.cis.hut.fi/research/som_lvq_pak.shtml`, 1996, [Cited 2014-05-12].

[25] Lampinen, J.; Oja, E.: Fast self-organization by the probing algorithm. *ijcnn*, volume 2, 1989.

[26] Massey, P.; Hanson, M. M.: Astronomical Spectroscopy. *ArXiv e-prints*, Oct. 2010.

[27] Mathew, S.; Joy, P.: Ultra Fast SOM using CUDA [online]. `http://nestsoftware.com/nest/whitepapers/Ultra_Fast_SOM_using_CUDA.pdf`, [Cited 2014-05-12].

[28] McCulloch, W.; Pitts, W.: A logical calculus of ideas immanent in nervous activity. *Bulletin of Math. Biophysics*, volume 5, 1943: pp. 115–133.

[29] Mrkva, L.: *VO-KOREL, server for astronomical cloud computing*. Bachelor's thesis, Czech Technical University in Prague, Faculty of Information Technology, Czech Republic, 2012.

[30] Neagoe, V.; Ropot, A. D.: Concurrent self-organizing maps for pattern classification. In *Cognitive Informatics, 2002. Proceedings. First IEEE International Conference on*, 2002, pp. 304–312.

[31] Sathya, R.; Abraham, A.: Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification. *International Journal of Advanced Research in Artificial Intelligence*, volume 2, 2013: pp. 34–38.

[32] Škoda, P.: Optical Spectroscopy with the Technology of Virtual Observatory. *Baltic Astronomy*, volume 20, 2011: pp. 531–539.

Appendix A

Acronyms

ANN Artificial neural networks

BFS Breadth-first search

BMU Best matching unit

CSOM Concurrent self-organizing maps

DAME DAta Mining & Exploration

DAMEWARE Web Application REsource of DAME

FITS Flexible Image Transport System

GPU Graphics processing unit

HTML HyperText Markup Language

JSON JavaScript Object Notation

KDD Knowledge Discovery in Databases

k-d tree K-dimensional tree

LAMOST Large Sky Area Multi-Object Fiber Spectroscopic Telescope

NP-hard Non-deterministic Polynomial-time hard

PHP Hypertext Preprocessor

SOM Self-organizing map

U-Matrix Unified distance matrix

WEKA Waikato Environment for Knowledge Analysis

Appendix B

CD contents

- `readme.txt` ..... brief description of the CD content
- `wrapper` ..... created wrapping application
  - `compile.sh` ..... script for compilation
  - `config.json` ..... user configuration script
  - `run.py` ..... script for running of application
  - `bin` ..... all binary files
  - `src` ..... all source files
    - `web` ..... source files for web application
  - `input` ..... sample inputs
  - `result` ..... outputs from algorithms
  - `som_pak-3.1` ..... SOM_PAK library
- `misc` ..... miscellaneous
  - `tests` ..... sample tests
- `text` ..... text of the thesis
  - `latex` ..... text of the thesis in latex
  - `BP_Lopatovsky_Lukas_2014.pdf` ..... text of the thesis in pdf

Appendix C

User manual

C.1 First run

1. Compile the application by running a compile script:

   ```
   $> ./compile.sh
   ```

2. Insert data files into input folder `Wrapper/input/`.
3. Arbitrary change the user configuration file `config.json`.
4. Run the application:

   ```
   $> python run.py
   ```

5. Look for a results in result file `Wrapper/result`.
6. Arbitrary run created web application by opening `Wrapper/result/index.php` file in your browser.

C.2 Parameters of configuration file

Name arbitrary name of the experiment.

Algorithm definition of algorithm

- **BMU** "normal" or "probing" algorithm
- **Threads** select number of computational threads (integer)

Data definition of data input

- **Path** one or more paths to input files or whole directories.
- **File_type** "csv" or "fits"
- **Delimiter** delimiter of csv file.
- **Columns** columns in file or dimension of input vector.

Parameters SOM parameters

- **Topology** "rect" for rectangular, "hex" for hexagonal
- **Neighborhood_fcion** "gaussian" or "bubble" function
- **Neighborhood_radius** size of initial neighborhood radius (integer)
- **Map_size_x** horizontal size of SOM grid (integer)
- **Map_size_y** vertical size of SOM grid (integer)
- **Iteration** number of iterations over whole data set (integer)
- **Learning_rate** initial learning rate (float in interval 0–1)
- **Probing_iter** number of iterations of probing algorithm (integer, used only if probing algorithm is selected)

Output selection of output files

- **Visualization** values true or false, create visualization or not.
- **Error** values true or false, count errors or not.
- **Web** values true or false, create web application or not.

Optional_info selection of optional functionality

- **Names** values true or false, add files with data names or not (number of files and rows must be equivalent to input data)
- **Names_path** one or more paths to names files or whole directories
- **Classes** values true or false, add files with classes or not (number of files and rows must be equivalent to input data)
- **Classes_path** one or more paths to classes files or whole directories