Reinforcement Learning, Artificial Neural Networks and Support Vector Machines

Tiago Araújo (a71346) and Miguel Matos (pg35387)

Departamento de Engenharia Informática
Universidade do Minho

Abstract. We provide a concise introduction to basic approaches to reinforcement learning, artificial neural networks and support vector machines from the machine learning perspective. Some of the keywords are highlighted and most used methods mentioned.

Contents

1 Reinforcement Learning

1.1 Introduction

Reinforcement learning (RL) is an area of machine learning where an agent interacts with an unknown environment where you are presented with a problem that you do not know how it will be resolved. The solution is discovered through the attribution of **rewards** and **punishments** to the agent's **actions**, thus helping the agent to find the best solution to his **goal**. Actions may affect not only the immediate reward but also the subsequent rewards. Therefore, the two most important and distinguishing features of reinforcement learning are **trial-and-error search** and **delayed rewards**.

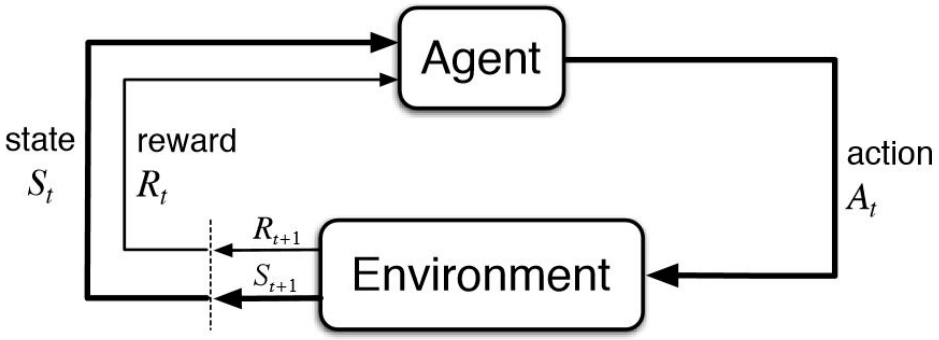

Fig. 1. Reinforcement Learning flow schema

As seen in the figure, an agent senses the state of the environment and takes an action on it which will result in a new state. The interpreter will evaluate this state and reward the agent. The agent will take rewards into consideration and another cycle starts until the agent reaches his goal.

Reinforcement learning therefore shines on situations where we don't have data, uncharted territory where the agent needs to **learn from his own experience**.

A good way to understand reinforcement learning is to consider some of the examples and possible applications that have guided its development.

- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find it's way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past;
- A gazelle calf struggles to its feet minutes after being born. Half an hour later it is running at 20 miles per hour.

1.2 Components

A RL agent includes one or more of the following components:

Policy is the agent's behaviour. It is a map from state to action:
Deterministic policy: $a = \pi(s)$
Stochastic policy: $\pi(a|s) = P[a|s]$

Value function is a prediction of future reward "How much reward will I get from action a in state s?".
Q-value function gives expected total reward I from state s and action a I under policy π I with discount factor γ.

$$Q^{\pi}(a|s) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

Fig. 2. Q-Value function

Reward function defines the goal in a reinforcement learning problem. Roughly speaking, it maps each state–action pair to a single number, a reward, indicating the desirability of that state. A learning agent's sole objective is to maximize the total reward it receives. The reward function defines what are the good and bad events for the agent. As such, the reward function must necessarily be unalterable by the agent. It may, however, serve as a basis for altering the policy. For example, if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future.

Model is the agent's representation of the environment. Is learnt from experience. Acts as proxy for environment and planner interacts with model.

1.3 Exploitation vs Exploration

One of the challenges that arise in reinforcement learning, and not in other kinds of machine learning, is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration or exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward.

1.4 Action-Value Methods

The n-armed bandit problem, so named by analogy to a slot machine, has n levers. Each action selection is like a play of one of the slot machine's levers, and the rewards are the payoffs for hitting the jackpot. Through repeated plays you are to **maximize** your winnings over a period of time by concentrating your plays on the best levers. In the n-armed bandit problem, each action has an expected reward if that action is selected, the value of that action. If you knew the value of each action, then it would be trivial to solve the n-armed bandit problem: you would always select the action with highest value. We assume that you do not know the action values with certainty, although you may have estimates.

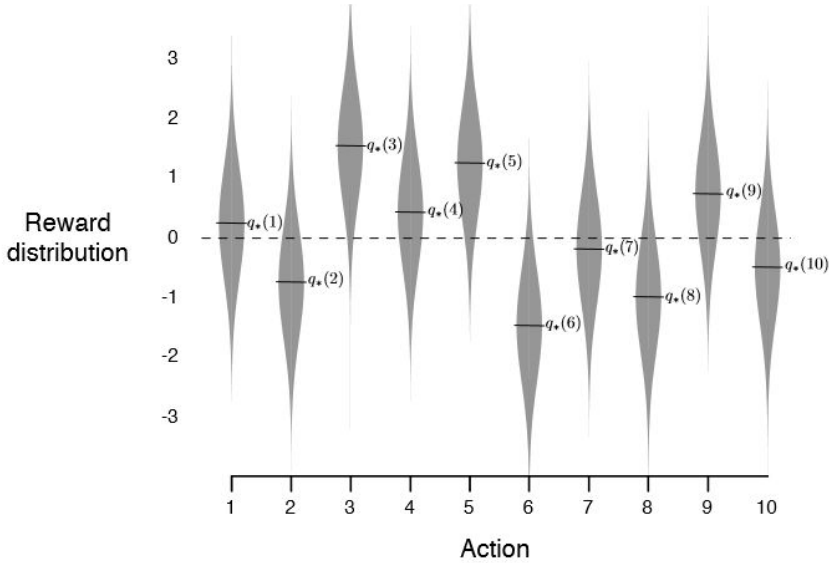

Fig. 3. An exemple bandit problem from the 10-armed testbed

If you maintain estimates of the action values, then at any time there is at least one action whose estimated value is greatest. We call this a **greedy** action. If you select a greedy action, you are **exploiting** your current knowledge of the values of the actions. If instead you select one of the **non greedy** actions, you are **exploring** because this enables you to **improve your estimate** of the non greedy action's value. Exploitation maximizes the expected reward on the one play, but exploration may produce the greater total reward in the long run.

For example, suppose the greedy action's value is known with certainty, while several other actions are estimated to be nearly as good but with substantial uncertainty. The uncertainty is such that at least one of these other actions probably is actually better than the greedy action, but you don't know which one. If you have many plays yet to make, then it may be better to explore the non greedy actions and discover which of them are better than the greedy action. Reward is lower in the short run, during exploration, but higher in the long run because after you have discovered the better actions, you can exploit them.

We call methods using this near-greedy action selection rule ε-greedy methods. An advantage of these methods is that, in the limit as the number of steps increases, every action will be sampled an infinite number of times, thus ensuring that all the Qt(a) converge to q∗(a).

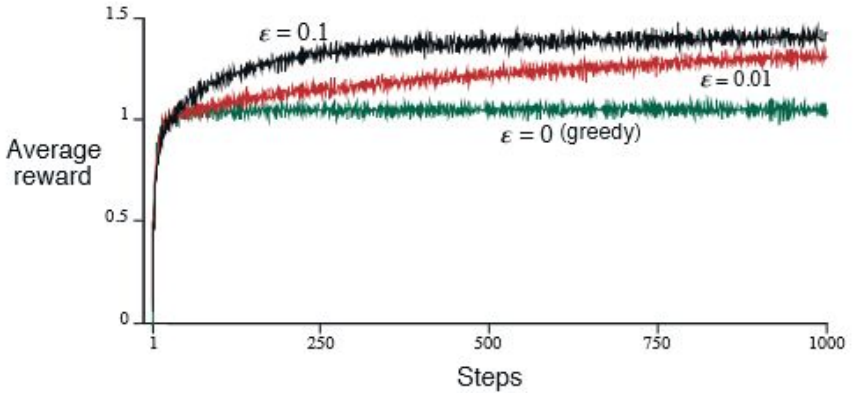

Fig. 4. Average performance of ε-greedy action-value methods on the 10-armed testbed

1.5 Markov Decision Processes (MDP)

The agent that interacts with the MDP is modeled in terms of a policy. A deterministic policy $\pi : S \rightarrow A$ is a mapping from the set of states to the set of actions. Applying π means always selecting action $\pi(s)$ in state s. This is a special case of a stochastic policy, which specifies a probability distribution over A for each state s, where $\pi(s,a)$ denotes the probability to choose action a in state s. The goal is to choose a policy π that will maximize some cumulative function of the random rewards, typically the expected discounted sum over a potentially infinite horizon:

There are three classes of fundamental methods for solving problems with MDP:

- **Dynamic programming methods** are well developed mathematically, but require a complete and accurate model of the environment;
- **Monte Carlo methods** don't require a model and are conceptually simple, but are not well suited for step-by-step incremental computation;
- **Temporal-difference methods** require no model and are fully incremental, but are more complex to analyze, Q-learning and SARSA are examples of Temporal-difference algorithms.

1.6 Development Tools

Some of the best tools for reinforcement learning are:

- **Tensorflow** - open source library for use of Machine Learning in general where it is possible to use reinforcement learning algorithms;
- **OpenAI Gym** - tool for developing and comparing reinforcement learning algorithms. It does not make assumptions about the agent structure and is compatible with any numerical computation library such as Tensorflow. It can be used from the Python language ;
- **PyBrain** - general library for Machine Learning that implements classic algorithms of Reinforcement Learning like Q-Learning;
- **BURLAP** - library of reinforcement learning in Java from Brown University;
- **RL Toolbox -** a C++ based, open-source, framework for all kinds of reinforcement learning algorithms.

1.7 Applications

Google AlphaGO. In October 2015, AlphaGO, an artificial intelligence system won a meeting of Go to the three-time European champion Mr. Fan Hui 5-0. Mechanisms of supervised learning, Monte Carlo tree research and reinforcement learning were used.

Manufacturing. In the Fanuc company, robots use reinforcement learning to pick up objects from a box to a container. If it succeeds or fails memorize the object and gain experience and train to get the job done quickly and accurately.

Finance Sector. Pit.AI is at the forefront leveraging reinforcement learning for evaluating trading strategies. It is turning out to be a robust tool for training systems to optimize financial objectives.

Inventory Management. Reinforcement learning algorithms can be built to reduce transit time for stocking as well as retrieving products in the warehouse for optimizing space utilization and warehouse operations.

Delivery Management. Reinforcement learning is used to solve the problem of Split Delivery Vehicle Routing. Q-learning is used to serve appropriate customers with just one vehicle.

2 Artificial Neural Networks

Artificial neural networks (ANN), also designed as **connectionist system**, are simplified models of the biological neural networks of humans. ANNs are structures of **computational units** extremely interconnected, named **neurons** or **nodes**, with the ability to learn.

Artificial neural networks are like the brain behaviour in two ways:

- The knowledge is acquired from an environment by a **learning process**.
- The knowledge is stored on the connections, also called **synapses**, of the nodes.

During the learning process, given by a learning algorithm, the **weight** of the connexions is adjusted to achieve a desired objective or network status. Although this being the traditional form of creating an ANN it is also possible to change their **inner structure**.

Artificial neural networks can be used on **supervised**, **reinforced** or **non-supervised**, for example the *Kohonen* networks, learning approaches.

To understand the way ANNs learn we have first to understand how the neurons/nodes work and how they can be structured.

2.1 Artificial neuron or node

The node is the key processing unit of ANNs. A node produces an output when the cumulative effect of the input exceeds a value. Therefore, a node contains:

- A **group of connections** each with its **own weight.** The signal is multiplied by the weight resulting on an excitation or inhibition of the node.
- An **integrator** which reduces the *n* inputs to one **single value**.
- An **activation function** which conditions the output signal adding a nonlinear component to the process.

Some of the most used functions are the **binary**, **linear** and **sigmoid** as seen in the figure 4 below in the same order.

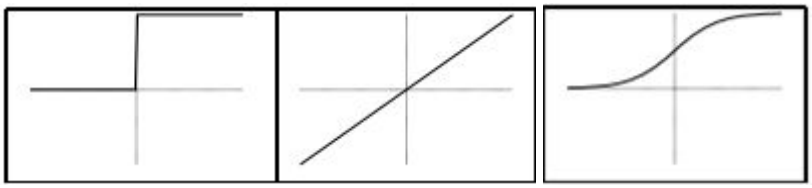

Fig. 5. Binary, linear and sigmoid function respectively

2.2 Network Architectures

The way the nodes spoken of previously connect between themselves it's named **architecture**. There are many types of architectures, but they can usually be generalized into three types:

- **Single Layer Feedforward Networks:** the connections in the network are always **unidirectional**, being either convergent (multiple connections to a single node) or divergent (from one node there are multiple connections). There is an **input layer** of nodes and an **output layer** of nodes.

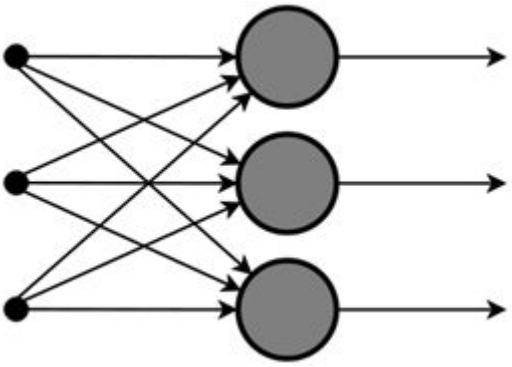

Fig. 6. Example of Single Layer Feedforward Network

- **Multi-Layer Feedforward Networks:** Relative to the single layered ones they add **one or more intermediary layers** and their nodes are called intermediary nodes. By adding these layers, it increases the capacity of the network to **learn more complex functions**, useful for when the number of input nodes is high. Although it also has a downside, which is the **increase of the learning time exponentially**. One of the most used algorithms for this type of network is **backpropagation**. It is a method that calculates the error contribution of each neuron and then an optimization algorithm adjusts the weight accordingly. It is also called a **back propagation of errors**, because the error is calculated at the output and distributed back through the network layers.

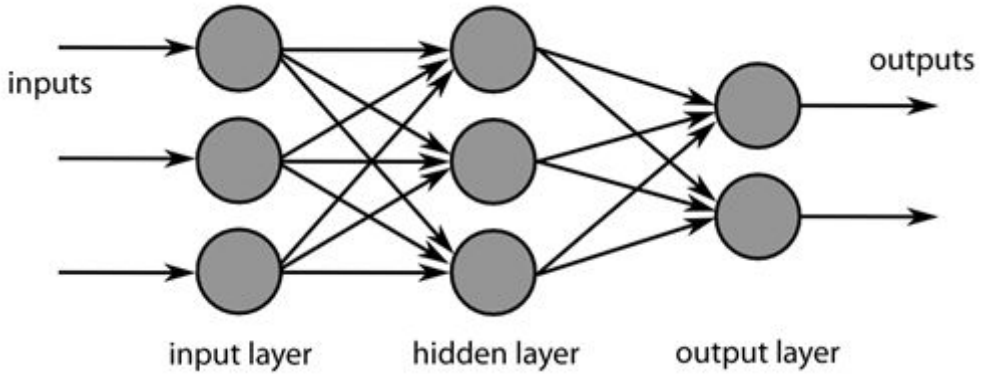

Fig. 7. Example of Multi-Layer Feedforward Network

- **Recurrent Networks:** the connections in this type of network form a **directed cycle**, meaning that nodes can have connections to themselves or other nodes in a previous or in the same layer as seen in the figure. This allows it to exhibit a **dynamic temporal behaviour**.

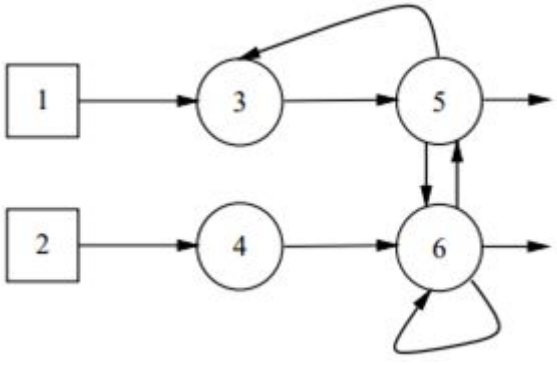

Fig. 8. Example of Recurrent Network

2.3 Learning

After seeing the elements of a neural network and how they work together we can say that the learning of an artificial network consists of the following steps:

- The ANN is stimulated by an **event**.
- The **weights** of the nodes are changed because of this event.
- The ANN gives a **new output** related to the alterations in the internal structure.

This learning is executed based on a set of **rules**, the learning algorithm.

2.4 Development Tools

Some of the most well-known tools to develop artificial neural networks are:

- **Theano** is a CPU/GPU symbolic expression compiler in python.
- **Torch** provides a Matlab-like environment for state-of-the-art machine learning algorithms in lua.
- **Tensorflow** is an open source software library for numerical computation using data flow graphs. It was developed by Google.
- **Lasagne** is a lightweight library to build and train neural networks in Theano.
- **Keras** is a Theano based deep learning library like Lasagne.

2.5 Applications

Artificial neural networks have been employed on multiple areas such as Robotic, Medicine, Economy and Computer Science. Some examples of more concrete applications are:

- **System identification and control** such as vehicle control and trajectory prediction.
- **Game playing and decision making** used in backgammon, chess and poker for example.
- **Pattern recognition** with uses in radar systems, face identification, signal classification and object recognition.
- **Sequence recognition** of handwritten text and speech.
- **Medical diagnosis**
- **Social Network and e-mail spam filtering.**
- **Cancer diagnosis** by using cell shape information.
- Automated trading systems in the **finance** area.

3 Support Vector Machines

In machine learning, **support vector machines (SVMs)** are supervised learning models with associated learning algorithms that analyse data used for **classification** (the problem of identifying to which of a set of categories a new observation belongs) and **regression analysis** (set of statistical processes for estimating the relationships among variables). Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic **binary linear classifier** . An 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** (hyperplane) 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.

In general, complex real-world applications require more expressive hypothesis spaces than linear functions. SVMs can efficiently perform a nonlinear classification using what is called the **kernel trick**, implicitly mapping their inputs into **high-dimensional feature spaces**.

When **data are not labelled**, an **unsupervised learning approach** is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups. The clustering algorithm which provides an improvement to the support vector machines is called **support vector clustering**.

3.1 Linear SVM

Suppose some given data points each belong to one of two classes, and the goal is to decide which class a new data point will be in. In the case of support vector machines, a data point is viewed as a p-dimensional vector (a list of p numbers), and we want to know whether we can separate such points with a (p-1) dimensional **hyperplane**. This is called a linear classifier. There are multiple hyperplanes that can classify the data. The best hyperplane is the one that represents the **largest gap**, or margin, between the two classes. If this hyperplane exists it is known as the **maximum-margin hyperplane** and the linear classifier it defines is known as the **maximum margin classifier**. As seen in the figure 8 the maximum-margin hyperplane is H3 (red).

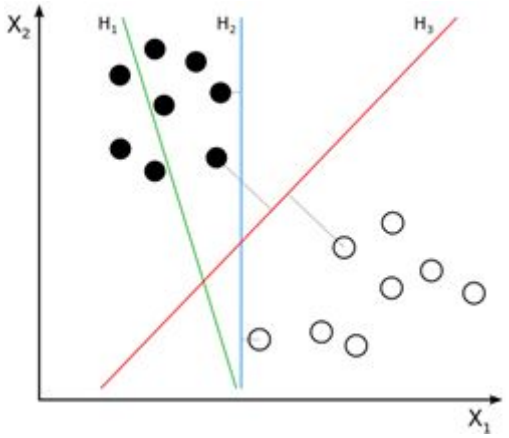

Fig. 9. Representation of various hyperplanes in space

The support vector machine is better than other classifiers because when you get a new sample (new points), you will have already made a line that keeps B and A as far away from each other as possible, and so it is less likely that one will spill over across the line into the other's territory.

3.1.1 Hard Margin

If the training data are **linearly separable**, we can select two parallel hyperplanes (**support vectors**) that separate the two classes of data, so that the distance between them is as large as possible. The region bounded by these two hyperplanes is called the "margin", and the **maximum-margin hyperplane** is the hyperplane that lies **halfway between them**.

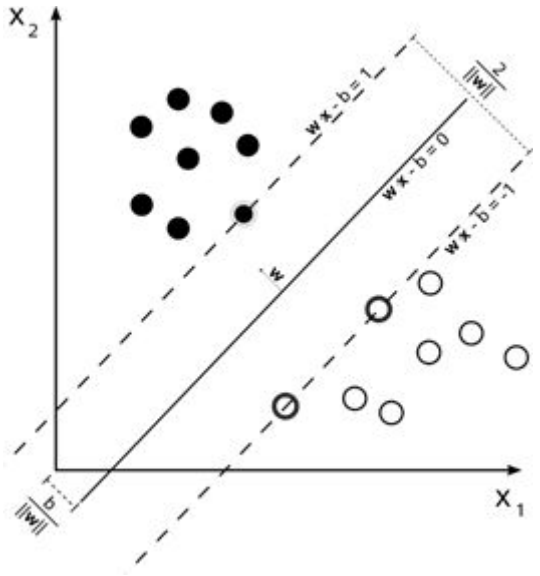

Fig. 10. Representation of a Hard Margin case with one hyperplane and two support vectors

3.1.2 Soft Margin

To extend SVM to cases in which the data are **not linearly separable** (when the data has noise) we need some relaxation of the constraints by adding **slack variables** and using the **hinge-loss function**. After this we proceed to minimization as it can be seen in the figure 10.

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{\|\mathbf{w}\|}{2} + C\sum_{i=1}^{N}\xi^{(i)}, \\ s.t. \quad & y^{(i)}(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)} + b) \geq 1 - \xi^{(i)}, \qquad \forall i \in \{1,\ldots,N\} \\ & \xi^{(i)} \geq 0, \qquad \forall i \in \{1,\ldots,N\} \end{aligned}$$

Fig. 11. Minimization problem

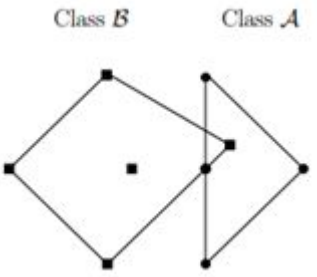

Fig. 12. Convex hull of class A and B

As we can see in the figure 11 the classes **intersect** making our objective reducing the influence of outlying points on the solution.

These linear SVMs can be optimized with the **Lagrangian dual problem**.

3.2 Nonlinear SVM

The original maximum-margin hyperplane algorithm proposed by Vapnik in 1963 constructed a linear classifier. However, in 1992, Bernhard E. Boser, Isabelle M. Guyon and Vladimir N. Vapnik suggested a way to create nonlinear classifiers by applying the **kernel trick** to maximum-margin hyperplanes. The resulting algorithm is formally similar, except that **every dot product is replaced by a nonlinear kernel function**

(similarity function over pairs of data points). This allows the algorithm to fit the maximum-margin hyperplane in a **transformed feature space**. The transformation may be nonlinear and the transformed space **high dimensional**. Although the classifier is a **hyperplane in the transformed feature space, it may be nonlinear in the original input space** as seen in the figure 12.

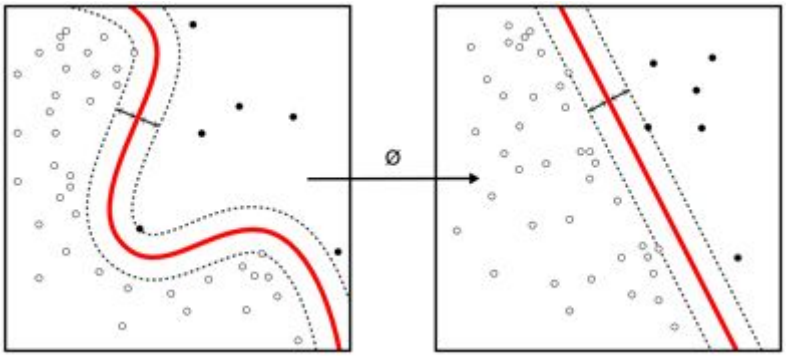

Fig. 13. Representation of hyperplane in the original and transformed space by the kernel trick

Another of the advantages of the kernel method is that the learning algorithms and theory can be largely decoupled from the specifics of the application area, which must simply be encoded into the design of the kernel function.

Working in a higher-dimensional feature space **increases the generalization error** of support vector machines, although given enough samples the algorithm still performs well. Generalization in this context is the ability of a learning machine to **perform accurately on unseen examples** after having experienced a learning data set. Successfully controlling the increased flexibility of kernel-induced feature spaces requires a sophisticated theory of generalisation, which is able to precisely describe which factors have to be controlled in the learning machine in order to guarantee good generalisation.

The most common kernels are the **polynomial** and **gaussian radial basis function** kernels.

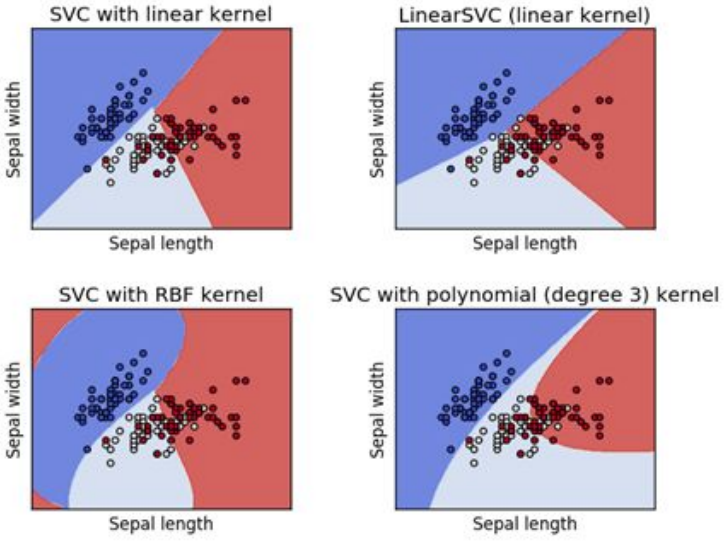

Fig. 14. Examples of classification using different types of kernels

In figure 13 we have three classes instead of two and we can see examples of the nonlinear kernels mentioned above.

3.3 Multiclass classification

As seen in figure 13 there can be situations where we have more than two classes. The most dominant approach is to reduce the **single multiclass problem into multiple binary classification problems**.

There are two methods for this not specific to SVMs:

- **One-vs-all (OVA)**: Classification of new instances in this case is done by a winner takes all strategy in which the classifier with the highest output function assigns the class. For example, if we have A, B and C classes we train A vs not-A, B vs not-B, C vs not-C. This means we **train one classifier per class**. The class with the highest confidence score wins.

- **One-vs-one (OVO)**: In this case classification is done by a max-wins voting strategy in which every classifier assigns the new instance to one of the two classes, then the vote for the assigned class is increased by one vote. The class with most votes determines the instance classification. This means we **train all possible pairs of classifications** therefore having to train way more binary classifiers then the other method.

3.4 Development Tools

The most used SVM tools are :

- **LIBSVM** – is a sophisticated SVM library with interfaces for Python, R, S+, MATLAB, Perl, Ruby and LABVIEW make it particularly popular.
- **Weka** – one of the best known collection of data mining tools which also contains an SVM implementation.

3.5 Applications

Algumas das aplicações de SVM em problemas reais são:

Text Categorisation: The task of text categorisation is the classification of natural text (or hypertext) documents into a fixed number of predefined categories based on their content. This problem arises in a number of different areas including email filtering, web searching, office automation, sorting documents by topic, and classification of news agency stories.

Image Recognition: The automatic categorisation of images is a crucial task in many application areas, including Information Retrieval, filtering Internet data, medical applications, object detection in visual scenes, etc. Much of the research in image categorisation focuses on extraction of high level features from images, for example using edge detection and shape description techniques, in order to capture relevant attributes of the image without increasing the number of features by too much traditional classification approaches perform poorly when working directly on the image because of the high dimensionality of the data, but Support Vector Machines can avoid the pitfalls of very high dimensional representations

Handwritten characters: The first real-world task on which Support Vector Machines were tested was the problem of hand-written character recognition. This is a problem currently used for benchmarking classifiers, originally motivated by the need of the US Postal Service to automate sorting mail using the handwritten Zip codes.

Bioinformatics:

- **Protein Homology Detection**. A protein is formed from a sequence of amino-acids from a set of 20, there exist thousands of different proteins, and one of the central problems of bioinformatics is to predict structural and functional features of a protein based on its amino-acidic sequence. This can be done by relating new protein sequences to proteins whose properties are already known, i.e. by detection of protein homologies. In a sense, proteins are clustered in families, which are then grouped together into superfamilies based on their properties.

- **Gene Expression**. Another application of SVMs to biological data mining is given by the automatic categorisation of gene expression data from DNA microarrays. The DNA microarray technology is revolutionising genetic analysis by making it possible to detect what genes are expressed in a particular tissue, and to compare the levels of expression of genes between the tissue in two different conditions.

References

https://en.wikipedia.org/wiki/Reinforcement_learning
Richard S. Sutton and Andrew G. Barto, “Reinforcement Learning: An Introduction”, The MIT Press, 2nd edition, 2012.
https://en.wikipedia.org/wiki/Artificial_neural_network
Paulo Cortez, José Neves, “Redes neuronais artificiais”, Universidade do Minho, 2000
https://stats.stackexchange.com/questions/23391/how-does-a-support-vector-machine-svm-work
https://en.wikipedia.org/wiki/Support_vector_machine
Nello Cristianini, John Shawe-Taylor, “An Introduction to Support Vector Machines and other kernel-based learning methods”, Cambridge University Press, 2000.