When and how to apply Statistics, Machine Learning and Deep Learning techniques

Josep Lluis Berral-García, *Member*, IEEE
Barcelona Supercomputing Center, Jordi Girona 29-31, 08034, Barcelona, Spain
Tel: 0034 93 4137716, Fax: 0034 93 4137721, e-mail: josep.berral@bsc.es

ABSTRACT

Machine Learning has become 'commodity' in engineering and experimental sciences, as calculus and statistics did before. After the hype produced during the 00's, machine learning (statistical learning, neural networks, etc.) has become a solid and reliable set of techniques available to the general researcher population to be included in their common procedures, far from the mysticism surrounding this field when only ML experts could solve modeling and prediction problems using such novel algorithms. But while knowledge on this field has settled among professionals, novice ML users still have trouble to decide when determined techniques could and should be applied to solve a given problem, sometimes ending with over-complicated solutions for simplistic problems, or complex problems partially solved by simplistic methods. This tutorial wants to introduce the most common techniques on statistical learning and neural networks, towards showing the proper techniques for each given scenario.

Keywords: Predictive Analytics, Machine Learning, Deep Learning, Knowledge Discovery

1. INTRODUCTION

Day by day, more professionals from different science and engineering fields decide to include machine learning techniques on their procedures and methodologies. Machine Learning (ML) provides methods and algorithms to treat and extract information from data automatically, letting computing frameworks to model and predict the behaviour of observed systems, people and scenarios. The relevance of ML comes up when such observed system cannot be modelled manually by human operators or experts, as these are too complex and dynamic or simply there are no experts for a new given scenario. Almost all the different ML algorithms are based on collecting data from an observed system to create a model; this model can be used for explaining its behaviour (discovering behaviours and properties), for relating a-priori observed features against a-posteriori observed ones (prediction and classification), or for relating actions on the system against the system status (heuristics and decision making oracles) among other uses.

Interest in machine learning has arisen among the scientific community as many unsolved problems, infeasible through classical and exhaustive computing, can be solved through approximation and automatic modelling. ML techniques can provide models as oracles or heuristics, that although being approximate or produce values with a certain error, can be used with confidence to guide problem solving better than guessing or not having any value. ML can also provide pattern recognition techniques, computationally expensive in the past due to lacking high performance computing resources, but feasible with present technologies. Each ML algorithm has different characteristics and purposes, also complexity and capabilities, always depending on the kind of data to learn. Therefore, like in every other field of science and engineering, professionals must choose their tools and methods according to each problem to be faced, knowing the capabilities and requirements of each method, also avoiding oversimplification or complication.

2. STATISTICAL LEARNING

Most machine learning techniques are based on statistical approximation, towards modelling and prediction, data clustering and characterization, and approximate heuristics towards decision making [1]. These processes can be done off-line, when data is collected from the observed system *a-priori* and a model is trained to be used *a-posteriori*, or on-line, when data is continuously collected and a model is updated using the incoming data. When creating a model, we must know what it is for. The most general classification of machine learning problems considers the Supervised problems, the Unsupervised problems, the Reinforcement learning problems and the Data Stream problems.

2.1 Supervised Learning

Supervised problems mostly focus on having a *model* (formula, heuristic or oracle) that, given a new input (observation), an output can be predicted or classified. The training method consists on having some input examples with their relative outputs, then the ML algorithm attempts to create a model by fitting the input features to their outputs. The accuracy and precision of the model can be measured by comparing the predicted outputs against the real outputs, guiding the training process towards tuning the model.

Regressions attempt to find a formula $f(x) = \hat{y} \approx y$ by minimizing $\varepsilon = (\hat{y} - y)^2$ between real and predicted outputs. Usual methods consider the regression as linear or polynomial, increasing the degree of the polynomial to overfit the regression to our observation points, aware of losing generality on our model. Also we can use multinomial regressions if we consider our features related. In situations where our target output is a label or class, logistic regressions can be also considered, where outputs are passed through a logistic function to obtain the probability of an input to belong to a given class.

Trees and Forests build models shaped as decision trees, with a label or a regression on each tree leaf [2]. Trees compute the *relevance* (e.g. Information Gain or Gini Index) of features to create a decision tree, dividing the datasets recursively by class or value. New inputs are passed through the tree until reaching a leaf, then a class or value is assigned, or a regression is performed. Forests are ensembles of trees, useful when models are unstable (different samples of the same data produce very different trees). Trees are especially useful for their interpretability, as they can be visualized and show the importance of each feature to classify or predict data.

Space and Neighbour models are based on similarity of data in order to classify it [3]. Those models memorize training data to compare new inputs with, then classified or predicted as the most similar inputs. The classical algorithm for this is *k*-Nearest Neighbours, where a new input is predicted as the average or vote of the *k* nearest memorized samples. This kind of models have the *curse of dimensionality*, and are not adequate for high dimensional data without transforming it into a lower dimension space, but they tend to be accurate when data classes are homogeneous.

Bayesian methods compute probabilities of observed inputs for a given class, to use the Bayes theorem to obtain the probabilities of a class given new inputs. We can use the Naïve Bayes algorithm when assuming independence between features, or Bayesian networks when not. Such methods are easy to train as they depend uniquely on collecting the support of each feature or set of features for each class, then using Bayes theorem to classify new data, also are resilient to noise and high dimensionality data.

2.2 Unsupervised Learning

Unsupervised problems mostly focus on discovering data relations from a model without output features. Classical examples are Clustering or Dimensionality reduction problems. Models are fitted by minimising distances inside clusters, maximising distances among clusters, reducing the variance of data, etc. The objective is to discover in data relations unknown to the scientist. Created models can be used for predicting new data into the found clusters, or transform it into new reduced dimensions.

Clustering methods attempt to group data by similarity on features or behaviour, so these groups can be studied later by scientists, or be used as reference groups for new inputs. One of the first problems when clustering is deciding how many groups we want to produce. Methods like *k*-means attempt to cluster data in *k* groups, where *k* is provided by the user, while hierarchical cluster does not require *a-priori* the number of groups but the user decides a similarity threshold to indicate when stop joining samples and groups. Other methods like DBSCAN or kernel density estimation attempt to cluster data without knowing the number of cluster *a-priori*, by determining a threshold for variance metrics on the produced groups.

Dimensionality reduction methods attempt to reduce the number of features on a dataset while not losing much information on data. While this reduction can be done though feature selection methods, by dropping those features apparently not correlated to our outputs, in unsupervised scenarios where no outputs exist, methods rely on which features (or combination of features) maximise the variance of data, dropping those that do not. A classic example is Principal Component Analysis, a method to transform data features into uncorrelated linear combination of features, also ranking those by their provided variance on data, then removing new dimensions with low variance. The user must decide the cutting threshold for this variance, and once data is transformed, it can be passed through other ML methods, with lower dimensionality.

2.3 Reinforcement Learning

Reinforcement Learning (RL) methods are based on trial and error, adjusting models from feedback. Decision making is a classical use of RL, where each model produced output receives a reward or penalty depending on feedback. These techniques are useful on State-Action problems, where a system must decide which action to recommend given a status; the action maximizing a utility function upon the current status is proposed, then the real utility function is evaluated *a-posteriori* to reward or penalize the action for such state. On stable systems, models tend to converge into a policy indicating the best action for each state, and in evolving systems the model is constantly updated. The set of state-action pairs can be represented by a Markov chain, where each state is connected to other states by actions, and each state selects that action leading towards higher utility state, also this Markov chain can have memory remembering the last *n* visited states and performed actions.

The most used algorithms for RL are Q-Learning and SARSA, based on Bellman equations [4]. Q-Learning implements a state-action policy algorithm, attempting to find the optimal policy *P* on state-action pairs as a function $Q(s_t,a_t)$, updated through a reward *r*, a learning rate α, and a discount factor γ for each action (eq.1). This method updates the status-action value pair based on the maximum possible reward after such state-action, and is done offline by learning about the state-actions performed on an example run. On the other hand, the SARSA algorithm (State-Action-Reward-State-Action) updates the status-action value pair based on its own selected actions (e.g. action with highest immediate reward), faster and able to be updated on-line (eq.2).

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot max_a\, Q(s_{t+1},a) - Q(s_t,a_t)] \qquad eq.1$$
$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha \cdot [r_{t+1} + \gamma \cdot Q(s_{t+1},a_{t+1}) - Q(s_t,a_t)] \qquad eq.2$$

E.g., RL for decision making is seen in resource management, learning policies for enabling resources. A utility function must be defined (e.g., benefits versus costs on enabled resources) in order to reward or penalize actions. Then, a self-optimization system can determine when enabling or disabling resources at each moment will produce better long term results.

2.4 On-line Learning

While most cases of use for modelling and prediction can be performed off-line, with Big Data and systems continuously producing data, being able to create and update models on-line is a desirable quality on ML methods. On-line learning methods are proposed for handling streams of data while modelling them on-line.

On-line learning strategies can consist on updating models by buffering data and renewing the model with certain periodicity or when the model becomes outdated, also building an ensemble of different models, training new models with new data while discarding old models. For this, we must define a policy for updating models and buffering data, and define the decision policy in the created ensembles.

Further, data stream methods consist on algorithms allowing to treat data immediately as it arrives, updating models in real time. Data stream models are used on scenarios where data volume and periodicity is high enough that analysing data more than once (or sometimes all data once) is prohibitive. *Sketches* define which and how data is kept in the system, and how data is updated with the arrival of new inputs.

3. NEURAL NETWORKS AND DEEP LEARNING

Neural network techniques can be included in the previous categories of statistical learning for supervised and unsupervised learning, but we give them a special focus because of their specific structure and training methodology.

3.1 Neural Networks

Artificial Neural Network (ANNs or NNs) are artefacts based on the collection of connected *neurons*, each one processing inputs into an output towards other neurons [5]. NNs have different useful configurations, being popular Feed Forward NNs for regression and classification problems, Convolutional NNs for image recognition, Recurrent NNs for series. NNs are trained by defining their configuration (how many neurons and their function, and how are they connected) then data is passes through the network repeated times. Each neuron attempts to predict its output value from the input values using its function, compute the error, and then adjust its function through weights. A classical neuron known as *perceptron* performs a weighted aggregation of inputs towards an output $f(x) = w_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \ldots$, and depending on whether the output is result of a prediction or classification, the output can be the identity of the aggregation $g(x) = f(x)$ or its sigmoid function $g(x) = sigmoid(f(x))$. The perceptron actually performs a linear regression by adjusting weights using gradient descent to reach the least square error. Training a perceptron requires to pass data several times (*epochs*) for weights to adjust towards the minimal error. Neural networks usually are composition of neurons using specific f and g functions. Here we list some examples of used NN configurations:

Feed Forward ANNs are composed by layers (*arrays*) of neurons, where inputs are connected to all neurons on the first layer, then the outputs are connected to all neurons of the second, until reaching the output neuron, aggregating results for regression or classification. The intermediate layers are called *hidden layers*. Data flows forward until the output layer, where values are compared to the real one, then the error is computed and propagated backwards, adjusting all neuron weights using techniques like, e.g., gradient descent. Methodology and generated models work like in supervised learning. **Convolutional NN**s are FFANNs including *convolutional* and *pooling* layers, neurons performing convolution on inputs, used in image recognition to detect pattern in image and video. **Recurrent NN**s are similar to FFANNs with the exception that layers are also retro-fed and may contain memory units, useful for learning series. **Auto-Encoders** (AE) can be considered as a specific NN as the architecture behaves like a FFANN with one hidden layer and an output layer with as many outputs as input features, attempting to learn the input from itself passing through a dimensionality reduction. AEs can be used, e.g., as dimensionality reduction mechanisms, like with **Restricted Boltzmann Machines** (RBM), with a single hidden layer, where inputs are converted to hidden values (new features), the reconstructed to the initial inputs using the inverse transformation matrix (not like AEs with different transformation and reconstruction matrices). RBMs can also be used to expand dimensions, when hidden layers have more units than inputs; the hidden units produce new features (activations) as a linear combination of the inputs; also this can be used to correct and complete inputs, as partial inputs producing activations that later reconstruct the missing inputs (usually performing Gibbs sampling).

3.2 Deep Learning

Recent experimentation with Neural Networks has shown the capabilities of NNs with more than one hidden layer. Deep Learning refers to those neural network deployments. As one hidden layer can learn the non-linear relations of its inputs, a second hidden layer can detect the relations of the outputs of the first hidden layer, and so on [6]. An example for image recognition, networks can be trained with more than 5 hidden layers, where the first layer detects spots and lines, the second detects shapes built from the detected spots and lines, up to the fifth layer detecting objects, that an output layer classifies with labels. For this example, those hidden layers are actually composed by a convolutional layer, a pooling layer and a rectifier layer.

Training deep network training can be a costly process, the more layers you add, also considering that convolutional layers have a high complexity cost. But some solutions exist, as for the example of object recognition, trained neural network models are published open to the public and everybody can download and apply. As these public models are usually general, it has been shown that they can be retrained focusing on the target subsets of elements for a given application towards specializing the model for a specific application. Other uses of those trained networks are "transfer learning", to transform elements from one dataset to the "style" of another dataset. For the image example, this can be done retraining specific layers of the already trained object recognition network, and instead of using it to classify, new images are passed forward and then reconstructed backward, gaining the new "style".

It is hard to find a human readable interpretation for Neural network models, but this is not always impossible. As an example, in models obtained from image recognition, each neuron can provide its "kernel" as representation of the shapes it is recognizing. If we are not treating images but structured data, we can also easily visualize such information, to discover what is each layer recognizing, and use it to discover relevant patterns in what we are classifying.

4. USE OF MACHINE LEARNING METHODS

Having available so many ML methods, it is important to know which ones to use for each problem. Statistical learning, with regressions or Bayesian methods are the easiest to test, and usually provide good results. Often we just need the formula relating our measured inputs to a future output metric, or an output hard to compute. Or we just need to classify our observations, having lots of already classified examples, where Bayesian methods or k-Nearest Neighbours already do the trick. Then, as complexity of data grows, we can jump to classification or regression trees or forests, as they can, e.g., produce piece-wise linear regressions, also allow to read the model to understand what it has learned. Different scenario is when data is not "complex" but "wide" (high dimensionality), and then we can attempt feature selection or dimensionality reduction, and see if the new data works with the previous methods. Then, reaching datasets with apparent high complexity, we can start with small neural networks, and increase their power (i.e. number of neurons at the hidden layer). As the target in our data is not trivial, representing complex elements, we can start putting additional layers to our network.

For those problems representing series of data (e.g. time series), depending on our goals, we can choose data stream mining methods, to keep aggregations and statistics about our data constantly updated, or in case of requiring prediction, we can check first statistical signal processing techniques (auto-regressions or filters), then in case of complex series we can choose neural network configurations allowing series like RNNs and conditional RMBs.

5. CONCLUSIONS

In this paper I briefly overviewed the basic methods used in machine learning, towards modelling and prediction, considering statistical learning and neural networks. Each ML method has its complexity and its capabilities, and their application depends on the kind of data to learn, and the final application for our ML application. While simpler methods close to statistical analysis are valid for most problems, more complex methods exist to treat problems with higher difficulty or oriented towards understanding data.

ACKNOWLEDGEMENTS

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 639595).

REFERENCES

[1] Hastie, T.; Tibshirani, R. & Friedman, J. (2001), The Elements of Statistical Learning, Springer.

[2] Breiman, L. et al. (1984). Classification and regression trees. Wadsworth & Brooks/Cole Advanced Books & Software.

[3] Altman, N. S. (1992). An introduction to kernel and nearest-neighbour nonparametric regression. The American Statistician 46 (3): 175–185.

[4] Bellman, R. (1957) Dynamic Programming. Princeton University Press, NJ, USA.

[5] Bishop, C.M. (1995) Neural Networks for Pattern Recognition, Oxford: Oxford University Press.

[6] LeCun, Y.; Bengio, Y. & Hinton G. (2015). Deep learning, Nature 521, no. 7553: 436-444.