

T



ML OVERVIEW

introducing

ML TERMS

LEARN NOW HOW TO SPEAK ML !



P



E



T

Model.Type

P

Machine learning (ML) has strong roots in statistics, optimization theory, functional approximation, and signal processing with whom it shares most of its inner working apparatus.

But, it also developed its specific language different from the research branches mentioned above and in the center of its world is the model.

Therefore, the purpose of our course is to understand the specific language of machine learning with its distinguished terms and to analyze and understand a suite of selected algorithmic models developed in conjunction with the basic tasks suite, named ACCORDS: approximation, clustering, coding, orientation, recurrent, discrimination and seeing.

E

T



Models could be grouped in 3 major categories: **supervised, unsupervised and reinforcement learning.**

Apart from that, we could further discuss subcategories such as hybrid learning e.g. Semi-supervised learning, self-supervised learning, multi-instance learning or logical statistical inferences e.g. inductive learning, deductive inference, transductive learning. But these subcategories could be embedded in the one of the major categories, so we will not discuss extensively about them.

There are also many types of learning techniques: multi-task learning, transfer learning, ensemble learning, active learning, online learning, curriculum learning etc. We will discuss in this introductory book only about the 3 major categories from above, but it is worth mentioning their existence for the interested reader.

The application framework of ml algorithms is made of data and task together with algorithm and evaluation criteria. Recently, with explainability models, also the end-users were added to the pipeline. The difficulties encountered when constructing ML pipelines could be related to dataset that could be biased, the algorithm that could not learn what expected and to the end users that need specific types of explanation depending on their relation to the system and their knowledge

P



E



T



Model.Training.Dataset: usually the model is expected to train on the given dataset. Training implies the repeated processing of the dataset over a number of so-called epochs during which the model is fitting the appropriate values for its parameters e.g. weights and biases in case of a neural network and hyperparameters e.g. number of neuron and number of layers for a neural network such that we obtain the best approximation for the task with respect to some previously established metric and in the same time, we avoid overfitting i.e. using more parameters than can be justified by the dataset.

Training is done with the **Model.Compile.Fit** command and the dataset preparation will be introduced as **Model.Data.Preparation** for the specific tasks.

Depending on the data and the task at hand we will apply a supervised, unsupervised or reinforcement learning model.

P



E



T



Models.Type.Task: if the task is regression or classification and we have labeled data, we will use supervised models. Classification separates the data, regression fits the data.

If the task is clustering, dimensionality reduction, classification of unlabeled data, association mining, generative modelling, or dictionary learning and autoencoding, we will use an unsupervised learning model.

If the task is robot optimal mobility, computer games or adaptive control, we will choose reinforcement models. Sometimes generative models and reinforcement models are included in the class of semi-supervised learning when the reward function is known in some settings but cannot be evaluated in others.

If the task is natural language processing, the recurrent models and more recently the transformers are to chosen.

P



E



T



Models.Supervised: these types of models have access to a training dataset made out of data and their corresponding labels. Here, labels means that each data example in the training dataset is matched with the answer the algorithm should come up with on its own.

These models are usually trained using empirical risk minimization (ERM) which defines a family of learning algorithms that is used to give theoretical bounds on their performance.

Model.Dataset.Split: usually for the supervised models, the dataset is split between training, validation, sometimes also called development or dev-set and the test sets. The splitting is usually done in the following percent (70%,20%,10%), (70%,15%,15%), (80%,10%,10%) or (60%,20%,20%).

The training set is the portion of data used to fit the model. The validation set is made out of the part of data used to tune the model hyperparameters based on an unbiased evaluation of the trained model. Finally, the test set is the smallest dataset, and it is used to provide an unbiased evaluation of a final model fit on the training dataset and tuned on the validation set.

P



E



T



Models.Unsupervised: for these models the data is unlabeled, and their purpose is to solve the given task by automatically finding structure in the dataset.

We study in this course the following models: clustering or grouping tasks for which we briefly outline hierarchical clustering, k-means and dbscan, the apriori algorithm for association rules, generative adversarial network and autoencoder networks.

P



E



T



Models.Reinforce: for these models, we usually implement an agent which can take actions and can interact with an environment with the purpose to maximize the total rewards.

We study the following models: temporal-differences, q-learning, sarsa and deep q-network or DQN.

We use besides the mentioned libraries also the keras-rl library that works with openai gym.

P



E



T



Model.Explain: these are explainability algorithms for the machine learning models and they could be included in 3 major categories: visualization methods (CAM and grad-cam, layer-wise relevance propagation, deeplift etc.) That emphasize through visualization the characteristics of an input that strongly influence the output, model distillation (LIME, SHAP, distillation to decision tree or graphs etc.) Which develop a separate "white-box" ML model that is trained to mimic the input-output behavior and intrinsic methods (attention mechanisms, explanation association, model prototype etc.) That have been specifically created to render an explanation along with its output.

We will exemplify in this course the post-hoc model agnostic algorithms lime and shap using their respective python implementations in the dedicated chapter.

P



E



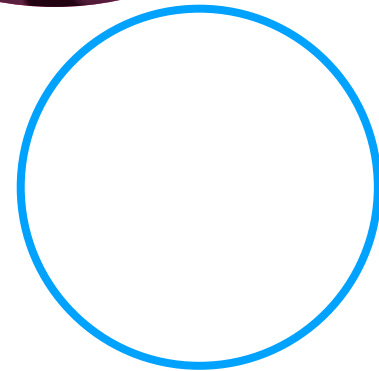


MODEL . MATH . FUNCTION

There is a real need of advanced mathematics for the understanding of machine learning models. For example, the supervised learning tasks can be understood as function approximation problems in the following way: given a label dataset of inputs and outputs, we assume that there is an unknown closed form function that is mapping the inputs to the outputs in the target domain and resulted in the dataset. We then use supervised learning algorithms to approximate this function.

Supervised machine learning algorithms can be modeled in a classical way with e.g. support vector machines, where we need to compute the support vectors or with modern deep learning neural networks.

Also, for shallow deep neural networks, we have the UAP.





Model.UAP: In 1989, Hornik, Stinchcombe, and White published a proof of the fact that for any continuous function f on a compact set K , there exists a feedforward neural network, having only a single hidden layer, which uniformly approximates f to within an arbitrary $\varepsilon > 0$ on K . This property is known as the Universal Approximation Theorem, UAP.

Now, recall the uniform norm of a function $f:A \rightarrow B$, $\|f\| = \sup\{|f(x)| : x \in A\}$. In other words, the uniform norm gives an upper bound on all possible values of f , and for two functions f, g we see that $\|f - g\|$ gives a uniform bound on the extent to which f and g differ from one another.





MODEL.DENSE


A feedforward neural network having N dense units or neurons arranged in a single hidden layer is a function $y: \mathbb{R}^d \rightarrow \mathbb{R}$ of the form $y(x) = \sum a_i \sigma(w_i^T x + b_i)$, where $w_i, x \in \mathbb{R}^d$, $a_i, b_i \in \mathbb{R}$, and σ is a nonlinear activation function.

The w_i are the weights of the individual units and are applied to the input x . The a_i are the network weights and are applied to the output of each unit in the hidden layer. Finally, b_i is the bias of unit i .

Therefore, a dense neural network is just a linear combination of affine transformations of its inputs under non-linear activations.

By an affine transformation we mean the $w_i x + b_i$ part of the model. Without the non-linear activation, we would have just linear functional approximation. But the non-linearity makes the model more complex and it allows to better represent complicated functions spaces that are out of reach for the linear models.

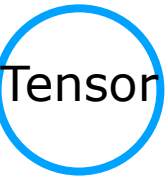
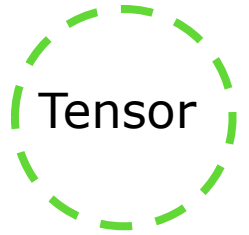




Model.Parameters.Tensors: A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array. Therefore, a vector is a one-dimensional or first order tensor and a matrix is a two-dimensional or second order tensor.

Examples: scalar, $s=5$; vector $v = [1, 2, 3, 4, 5]$ with dimension 1×5 ; matrix $M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$ with dimension 2×5 ; tensor $\begin{bmatrix} [1,2,3], [4,5,6], [7,8,9] \\ [11,12,13], [14,15,16], [17,18,19] \\ [21,22,23], [24,25,26], [27,28,29] \end{bmatrix}$ with dimension 3×3 .

For manipulating tensors, we use the Numpy library of Python: `import numpy as np`. Then one can represent them using `np.arrays`. To find their dimension you can use `np.dim`

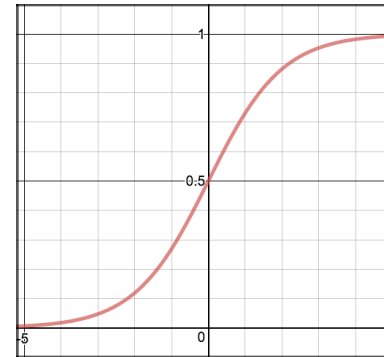




MODEL.FUNCTION.ACTIVATION: THE ACTIVATIONS FUNCTIONS ALSO KNOWN AS TRANSFER FUNCTIONS, ARE VERY USEFUL IN INCREASING THE MODEL APPROXIMATION PROPERTIES. WITH THE INTRODUCTION OF NON-LINEARITY, THE NEURON ACTIVATION FUNCTION ENHANCES THE NETWORK CAPACITY OF ACCURATELY REPRESENTING COMPLEX MODELS.

Model.Function.Sigmoid: It is a classical activation function and it is also known as the logistic activation function. It has the following definition in Python.

```
def sigmoid(z):  
    return 1.0 / (1 + np.exp(-z))
```



It has an S-shape curve and it can be used for models where we must predict the probability as an output. It is differentiable and monotonic, and its derivative has the following definition in Python

```
def sigmoid_prime(z):  
    return sigmoid(z) * (1-sigmoid(z))
```



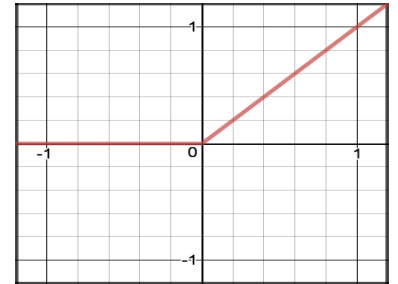
Model.Function.Relu: Rectified Linear Unit Activation Function is a novel activation function and it belongs to the new wave of transfer functions. It is simple and very facile to evaluate. The function is zero when x is less than zero and it is equal to x when x is above or equal to zero. It has the following definition in Python:

```
def relu(z):  
    return max(0, z)
```

It is monotonic and it avoids and rectifies vanishing gradient problem. Its derivative in Python is:

```
def relu_prime(z):  
    return 1 if z > 0 else 0
```

Connected to ReLU: Exponential Linear Unit or its widely known name ELU is a function that tends to converge cost to zero faster and produce more accurate results. Different to other activation functions, ELU has an extra alpha constant which should be a positive number. LeakyRelu is a variant of ReLU. Instead of being 0 when $z < 0$, a leaky ReLU allows a small, non-zero, constant gradient α (Normally, $\alpha = 0.01$). Other related functions are Bipolar rectified linear unit (BReLU), Parametric rectified linear unit (PReLU), Randomized leaky rectified linear unit (RReLU), Gaussian Error Linear Unit (GELU), Scaled exponential linear unit (SELU), S-shaped rectified linear activation unit (SReLU) etc.



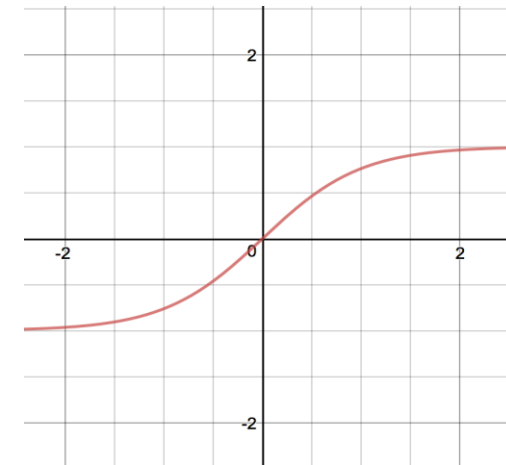
Model.Function.Tanh: The hyperbolic tangent Activation Function is also an S-shape function similar and from the same generation with the sigmoid function. Different from the sigmoid, its output is zero-centered and not 0.5 centered. Its definition in Python is the following:

```
def tanh(z):  
    return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
```

The function is monotonic while its derivative is not monotonic and has the following Python definition:

```
def tanh_prime(z):  
    return 1 - np.power(tanh(z), 2)
```

Close connected is the arctan function or inverse tan, which is also a S-shape function and it maps input to output ranging between $[-\pi/2, \pi/2]$.



Model.Activate.Softmax: This function is used to impart probabilities when you have more than one outputs you get probability distribution of outputs. Basically, the softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1. The softmax function is sometimes called the softargmax function, or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used for multi-class classification, and its formula is very similar to the sigmoid function which is used for logistic regression. The softmax function can be used in a classifier only when the classes are mutually exclusive. Mathematically, it has the following expression



Softmax

$$S(y_i) = \frac{e^{y_i}}{\sum_{j=1}^J e^{y_j}}$$

Its implementation in Python is the following:

```
def softmax(x):  
    return np.exp(x) / np.sum(np.exp(x), axis=0)
```



Softmax



ACCORDS

Approximation, Clustering, Coding, Orientation, Recurrent, Discrimination and Seeing.



Approximation and Discrimination

Supervised learning
models

Seeing

Deep
Convolutional
Neural Networks,
CNN

Coding and Recurrent

Recurrent
networks,
LSTMs and
Autoencoders,
GANs

Clustering and dimensionality reduction

Unsupervised
learning models

Orientation and Agents

Reinforcement
learning models

