JINAV: Journal of Information and Visualization *ISSN: 2746-1440 (online)*
Vol. 2 No. 2 (2021)
https://doi.org/10.35877/454RI.jinav373

Python for Automating Machine Learning Tasks

Tinatin Mshvidobadze

Gori State University, Georgia

Abstract

Machine learning is used in a variety of computational tasks where designing and programming explicit algorithms with good performance is not easy. Applications include email filtering, recognition of network intruders or malicious insiders working towards a data breach. In this article we will focus on basics of machine learning, tasks and problems and various machine learning algorithms. The article discusses the Python programming language as the best language for automating machine learning tasks.

Keywords: Machine learning; artificial intelligence; neural network; python; tensors.

1. Introduction

Machine learning (ML) is a branch of artificial intelligence. According to Mitchell (2011): "Machine learning is the study of computer algorithms that allow computer programs to automatically improve through experience."

The type of machine learning:

- *Supervised learning* - where supervised learning algorithms try to model relationship and dependencies between the target prediction output and the input features, such that we can predict the output values for new data based on those relationships, which it has learned from previous datasets (Kotsiantis, 2007).
- *Unsupervised learning* - another type of machine learning, is the family of machine learning algorithms, which have main uses in pattern detection and descriptive modeling. These algorithms do not have output categories or labels on the data. According to Talwar A., a central case of unsupervised learning is the problem of density estimation in statistics (Talwar & Kumar, 2013).
- *Reinforcement learning* - aims at using observations gathered from the interaction with its environment to take action that would maximize the reward or minimize the risk. A great example of reinforcement learning are computers reaching a super-human state and beating humans on computer games (Sutton & Barto, 2018).

Machine learning began to perform tasks that were impossible to do with classic rule-based programming.

A machine-learning system is trained rather than explicitly programmed. It's presented with many examples relevant to a task, and it finds statistical structure in these examples that eventually allows the system to come up with rules for automating the task.

2. Machine Learning Algorithms, Tasks and Applications

Machine learning systems using a computer can perform complex processes by learning data rather than following pre-programmed rules. Increasing data availability facilitates the preparation of machine building systems for a large

[*] Corresponding author.

E-mail address: xxxx@xxxxxx.edu (First Author) set of examples, while the increase in computer processing energy has supported the critical capabilities of these systems. Algorithmic advances are also observed in the field, which give machine learning better power.

As a outcome of these advances, systems which performed at noticeably below-human levels can now go better than humans at some definite tasks. According to Singh, S., the concept of machine learning is used in many applications and is a core concept for intelligent systems (Singh, Kumar, & Kaur, 2014). As the field develops further, machine learning shows promise of supporting potentially transformative advances in a range of areas, and the social and economic opportunities which follow are significant.

According to Muhammad & Yan (2015), there are number of machine learning algorithms such as Linear Regression, Logistic Regression, Decision Tree, SVM, and KNN. Linear Regression is used to estimate real values based on continuous variable(s). Decision Tree is a type of supervised learning algorithm that is mostly used for classification problems. SVM is a classification method. In this algorithm, we plot each data item as a point in n-dimensional space with the value of each feature being the value of a particular coordinate. K nearest neighbors is a simple algorithm which stores the entire available cases and classifies new cases by a majority vote of its k neighbors.

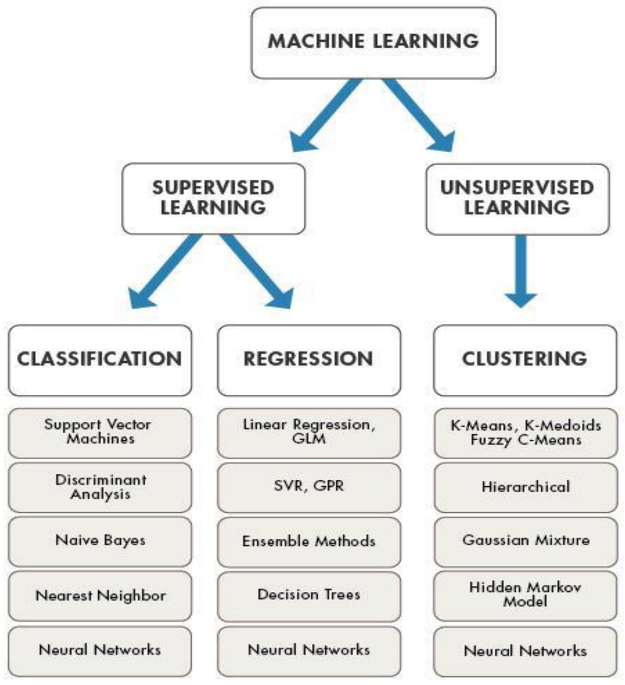

Fig. 1. Machine learning algorithms

According to Kumar, N. Machine learning algorithms are widely used in variety of applications like digital image processing (Singh, Kumar, & Kaur, 2014), big data analysis, Speech Recognition, Medical Diagnosis, Statistical Arbitrage, Learning Associations, Classification, Prediction etc. (Sharma, Pabby, & Kumar, 2017).

In 2012, machine learning, deep learning, and neural networks made great strides and found use in a growing number of fields. Organizations suddenly started to use the terms of “machine learning” and “deep learning” for advertising their products (Wikipedia, n.d.).

2.1. A first look at a neural network

Early iterations of neural networks have been completely supplanted by the modern variants covered in these pages, but it’s helpful to be aware of how deep learning originated.

As neural networks started to gain some respect among researchers, thanks to this first success, a new approach to machine learning rose to fame and quickly sent neural nets back to oblivion: kernel methods. Kernel methods are a group of classification algorithms, the best known of which is the support vector machine (SVM) (Cortes & Vapnik, 1995).

The first successful practical application of neural nets came in 1989 from Bell Labs, when Yann LeCun combined the earlier ideas of convolutional neural networks and backpropagation, and applied them to the problem of classifying handwritten digits.

Let's look at a concrete example of a neural network that uses the Python library Keras to learn to classify handwritten digits. The Python programming language best fits machine learning due to its independent platform and its popularity in the programming community.

The problem we're trying to solve here is to classify grayscale images of handwritten digits (28 × 28 pixels) into their 10 categories (0 through 9). We'll use the MNIST dataset, a classic in the machine-learning community, which has been around almost as long as the field itself and has been intensively studied. It's a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST).

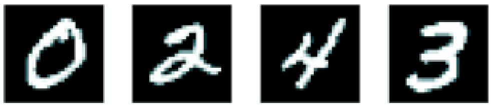

Fig. 1. MNIST sample digits

The MNIST dataset comes preloaded in Keras, in the form of a set of four Numpy arrays (Wan, Zeiler, Zhang, Le Cun, & Fergus, 2013).

2.2. Loading the MNIST dataset in Keras

```
from keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

`train_images` and `train_labels` form the *training set*, the data that the model will learn from. The model will then be tested on the *test set*, test_images and test_labels.

The images are encoded as Numpy arrays, and the labels are an array of digits, ranging from 0 to 9. The images and labels have a one-to-one correspondence.

Let's look at the training data:

```
>>> train_images.shape
(60000, 28, 28)
>>> len(train_labels)
60000
>>> train_labels
Array ([5, 0, 4, ..., 5, 6, 8],  dtype=uint8)
```

and here's the test data:

```
>>> test_images.shape
(10000, 28, 28)
>>> len(test_labels)
10000
>>> test_labels
Array ([7, 2, 1, ..., 4, 5, 6],  dtype=uint8)
```

The workflow will be as follows: First, we'll feed the neural network the training data, train_images and train_labels. The network will then learn to associate images and labels. Finally, we'll ask the network to produce predictions for test_images, and we'll verify whether these predictions match the labels from test_labels.

This example shows how it is possible to build and train a neural network to classify handwritten numbers in less than 20 lines of Python code.

In the following example will discuss all the moving pieces that we just reviewed and explained what is going on behind the scenes. Discuss data storage facilities in the network; The actions of the tensor from which the layers are made; And a gradient descent that allows your network to learn from its training examples.

In general, all current machine-learning systems use tensors as their basic data structure. Tensors are fundamental to the field—so fundamental that Google's TensorFlow was named after them.

At its core, a tensor is a container for data—almost always numerical data, also a container for numbers.

2D tensors are a generalization of matrices to an arbitrary number of dimensions.

If pack such matrices in a new array, you obtain a 3D tensor, which you can visually interpret as a cube of numbers. Following is a Numpy 3D tensor:

```
>>> x = np.array   ([[[5, 78, 2, 34, 0],
                     [6, 79, 3, 35, 1],
                     [7, 80, 4, 36, 2]],
                     [[5, 78, 2, 34, 0],
                     [6, 79, 3, 35, 1],
                     [7, 80, 4, 36, 2]],
                     [[5, 78, 2, 34, 0],
                     [6, 79, 3, 35, 1],
                     [7, 80, 4, 36, 2]]])
>>> x.ndim
3
```

By packing 3D tensors in an array, you can create a 4D, 5D tensor, and so on.

A tensor is defined by three key attributes (Shukla & Fricklas, 2018):

- *Number of axes (rank)*;
- *Shape;*
- *Data type* (usually called dtype in Python libraries).

The *relu* operation and addition are *element-wise* operations: operations that are applied independently to each entry in the tensors being considered. This means these operations are highly amenable to massively parallel implementations. If need a naive Python implementation of an element-wise operation, we use a for loop, as in this naive implementation of an element-wise relu operation:

```
def naive_relu(x):

assert len(x.shape) == 2        - x is a 2D Numpy tensor
x = x.copy()                    - avoid overwriting the input tensor
for i in range(x.shape[0]):
for j in range(x.shape[1]):
x[i, j] = max(x[i, j], 0)
return x
```

It is possible to add:

```
def naive_add(x, y):
assert len(x.shape) == 2                 - x and y are 2D Numpy tensors
assert x.shape == y.shape
x = x.copy()                             - avoid overwriting the input tensor
for i in range(x.shape[0]):
for j in range(x.shape[1]):
x[i, j] += y[i, j]
return x
```

2.3. Geometric interpretation of tensor operations

Because the contents of the tensors manipulated by tensor operations can be interpreted as coordinates of points in some geometric space, all tensor operations have a geometric interpretation. For instance, let's consider addition. We'll start with the following vector:

A = [0.5, 1]

It's a point in a 2D space (see figure 2). It's common to picture a vector as an arrow linking the origin to the point, as shown in figure 3.

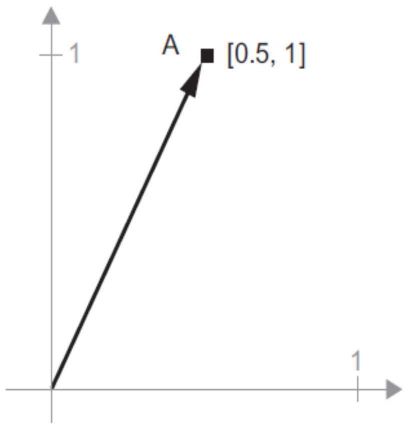

Fig. 2. A point in a 2D space

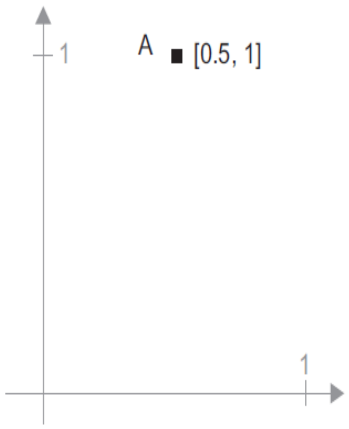

Fig. 3. A point in a 2D space pictured as an arrow

In general, elementary geometric operations such as affine transformations, rotations, scaling, and so on can be expressed as tensor operations (Chollet, 2018).

Thus neural networks consist entirely of chains of tensor operations and that all of these tensor operations are just geometric transformations of the input data. It follows It is possible interpret a neural network as a very complex geometric transformation in a high-dimensional space, implemented via a long series of simple steps. Each layer in a deep network applies a transformation that disentangles the data a little—and a deep stack of layers makes tractable an extremely complicated disentanglement process.

3. Conclusion

The article illustrates the concept of machine learning with its tasks and applications. Machine learning and AI, as a unit, are still developing but are rapidly growing in usage due to the need for automation. The demand for smart solutions to real-world problems necessitates the need to develop AI further in order to automate tasks that are tedious to program without AI. Python programming language is considered the best algorithm to help automate such tasks, and it offers greater simplicity and consistency than other programming languages.

References

Chollet, F. (2018). *Deep learning with Python* (Vol. 361). New York: Manning.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine learning*, *20*(3), 273-297.

Kotsiantis, S. (2007). Supervised machine learning: A review of classification techniques. *Informatica Journal, 31*, 249–268.

Mitchell, T. M. (2011). *For pioneering contributions and leadership in the methods and applications of machine learning.* National Academy of Engineering. Retrieved October 2, 2011.

Muhammad, I., & Yan, Z. (2015). Supervised Machine Learning Approaches: A Survey. *ICTACT Journal on Soft Computing*, *5*(3).

Sharma, D., Pabby, G., & Kumar, N. (2017). Challenges Involved in Big Data Processing & Methods to Solve Big Data Processing Problems. *International Journal for Research in Applied Science and Engineering Technology* (*IJRASET), 5*(8), 841-844.

Shukla, N., & Fricklas, K. (2018). *Machine learning with TensorFlow*. Greenwich: Manning.

Singh, S., Kumar, N., & Kaur, N. (2014). Design and development of RFID Based Intelligent Security System. *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET)*, *3*(1).

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Talwar, A., & Kumar, Y. (2013). Machine Learning: An artificial intelligence methodology. *International Journal of Engineering and Computer Science*, *2*(12), 3400-3405.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., & Fergus, R. (2013, May). Regularization of neural networks using dropconnect. In *International conference on machine learning* (pp. 1058-1066). PMLR.

Wikipedia. (n.d.). *Deep Learning Revolution*. https://en.wikipedia.org/wiki/Deep_learning#Deep_learning_revolution