The Engineering Guide to Machine Learning & Artificial Intelligence

Daniël Heres
Version 0.1

May 26, 2019

Contents

1 Introduction

What are the factors that makes Machine Learning projects successful? What should you focus on and what can be solved by tools? How do we bring models into production? Which skills do I need to develop to become productive? How do we tune models? How can we maintain models over time and understand the real time impact? How can we maintain and improve a model over time?

Machine Learning, being a relatively new field in industry, has many of these questions still open. Where more older profession in technology, such as Software Engineering, has developed lots of tools, patterns and ideas around it, in Machine Learning things are much more evolving and open. Also, as the field moves very fast, tools and ideas can become quickly outdated.

In this document I give an introduction into how to apply Machine Learning in practical settings. It is by no means complete or finished, and will need to be updated to stay actual.

In this guide we focus on the engineering side of machine learning, answering questions like these. I distill patterns that can be applied to the development cycle and popular tools that are used.

2 Data Collection

2.1 Labeling

Currently, the collection of labeled data is an important part of the machine learning practice.

Today's algorithms often need both data of a certain quality (e.g. should be very similar to the) and a certain quantity (the best algorithms need ¿ 1 million samples).

Even though in Machine Learning research approaches such as Transfer Learning and Self-supervised Learning reduce the need for labeled data, the need for large labeled data sets for accurate models is still big.

There are a couple of approaches towards collecting labeled data.

- **Automated**. This can be the case whenever there is existing historical data available, e.g. the number of page views at a certain day or the number of times a certain hashtag is used on Twitter. Although humans are part of the process, we don't need to manually collect the labels.
- **Semi-automated**. A human will provide feedback to a system's automatic suggestions. This example can then be used to improve the model's predictive performance and/or for measuring the performance. Sometimes labels can be collected without explicitly labeling
- **Manual**: Humans will completely label every example without the help of a computer.

2.2 Mechanical Turk Platforms

2.3 Labeling by Customers

A common strategy used by companies is to use an existing product to collect labels from customers, instead of paying per label.

This way, we can vastly expand our labeled data, and improve our existing labeled data by finding consensus in labeling with minimal cost.

2.4 Bootstrapping

3 Learning and modeling

Currently, the most successful and accur f ate models are using Supervised Learning. This means that a model is learned, from scratch, on a set of labeled data.

In this chapter we give some examples of commonly used models and how they work.

3.1 Train, validate, test

3.2 Cross-validation

3.3 Raw Feature Models

3.4 Image Models

3.5 Text Models

3.6 Sequence Models

3.7 Search & Ranking

3.8 Clustering

3.9 Anomaly Detection

4 Feature Engineering

Although we are moving more and more towards the fully automatic learning. Especially with numerical data (such as financial, sales-numbers, etc) feature engineering is an crucial part of models.

4.1 Normalization

Normalization is an important part of making features. Lots of algorithms such as neural networks work better when features are normalized.

For neural networks, the most important property of features is that the range of different features should be similar.

One basic way for normalization is to transform the features to have unit zero mean:

$$z = (x - \mu)/\sigma \tag{1}$$

Depending on the distribution of data, we need different ways of transformation. E.g. for a variable that is more exponential in nature, using a logarithm to transform it back. to a more linear distribution helps to make it a more useful feature to learning algorithms.

Similarly, if the data you have is quadratical, then taking the square root of the variable helps algorithms.

4.2 Cyclical Features

Cyclical features are very common in data involving time: we have often access to a timestamp or a day variable. Often there is a periodic pattern in this data. We want to use the fact that two times are similar when they are close to each other. E.g. 11:59 PM is very close to 00:00 AM. However, naively using the minutes since 00:00 AM as feature will have those features maximally apart!

One good way of using the time as a feature is to project them on a circle using sine and cosine.

If we have a (Unix) timestamp variable with the number of seconds, projecting it to a circle for minutes in an hour, hours in a day, and days in a week is easy:

$$\begin{aligned}
min_{sin} &= \sin(\frac{t \cdot 2\pi}{3600}) \\
min_{cos} &= \cos(\frac{t \cdot 2\pi}{3600}) \\
hour_{sin} &= \sin(\frac{t \cdot 2\pi}{24 \cdot 3600}) \\
hour_{cos} &= \cos(\frac{t \cdot 2\pi}{24 \cdot 3600}) \\
day_{sin} &= \sin(\frac{t \cdot 2\pi}{24 \cdot 3600 \cdot 7}) \\
day_{cos} &= \cos(\frac{t \cdot 2\pi}{24 \cdot 3600} \cdot 7)
\end{aligned} \tag{2}$$

We can do the same for yearly patterns, but the usefulness will depend on how many years of data you have.

4.3 Categorical Embeddings

If you have categorical variables. One basic way is to use One Hot Encoding.

Some examples of categories are

Figure 1: Thermometer 8/9

- Shoe color
-

4.4 Thermometer Encoding

When we have a feature bounded between two numbers we can. However, for example in neural networks, the way we use this feature will be linear. The idea of thermometer encoding is to transform one feature into n features, where each feature will be active at a certain threshold where each feature holds roughly the same amount of data points.

For example, when we have a variable from 0 to 10 and we want to transform it to a thermometer using stepsize of 1, the thresholds are at the values $1, 2, 3, 4, 5, 6, 7, 8, 9$.

A value 8.5 can be visualized as this "thermometer":

An implementation in NumPy:

```
import numpy as np

def thermometer(x, start, end, step_size=1):
    return ((x > np.arange(start + step_size, end, step_size))
            .astype(float))
```

This function gives the result:

```
>>> thermometer([[8.5]], 0, 10)
array([[1., 1., 1., 1., 1., 1., 1., 1., 0.]])
```

4.5 Positional Encoding

5 Version Control for Data, Models and Experiments

6 Data Augmentation

7 Feature Databases

8 Metrics

8.1 Mean Opinion Score

9 Hyperparameter Optimization

Usually, models contain lots of parameters that are chosen before training a model. Some examples are the number of layers in a neural network and the number of convolutional filters, the features that are used in the model or the learning rate that is used for the optimization algorithm.

We usually do hyperparameter on a metric which we directly want to optimize, such as top-1 accuracy. This is in contrast with gradient-based optimization were we often use a surrogate metric such as cross-entropy loss.

9.1 Bayesian Optimization

9.2 Neural Architecture Search

10 Optimization

10.1 Stochastic Gradient Descent

10.1.1 SGD with momentum

10.1.2 Adam

10.2 Evolutionary Algorithms

11 Complexity and Maintainability in Machine Learning

11.1 Feature Selection

11.2 Ablation Studies

11.3 Expressiveness

11.4 Don't repeat yourself

11.5 Feature Store

12 Numerical Libraries and Frameworks

To define and train Machine Learning models efficiently, we need libraries and frameworks.

12.1 TensorFlow

12.2 PyTorch

12.3 MxNet

12.4 Scikit-learn

12.5 NumPy

12.6 Keras

12.7 XGBoost

12.8 LightGBM

12.9 FastAI

13 Machine Learning on Big Data

13.1 Distributed Machine Learning

13.2 Data Formats

14 Productionizing Models

14.1 Model Formats

After learning a ML model such as a neural network or a random forest, we have to store the *weights* of the model and the *structure* of the model.

- ONNX
- TensorFlow SavedModel
- TensorFlow Lite
- Keras File. A binary file format from Keras that can be used to save / share. It uses the HDF5 format to save the model weights and
- Pickle. This is the built-in object serialization functionality of Python. It is meant to temporarily write an (Python) object to disk, to load it again later within the same environment.

14.2 Serving Models

- TensorFlow Serving
- MXNet Model Server