Andriy Burkov

MACHINE LEARNING ENGINEERING

"An optimist sees a glass half full. A pessimist sees a glass half empty. An engineer sees a glass that is twice as big as it needs to be."
— Unknown

"Death and taxes are unsolved engineering problems."
— Unknown

The book is distributed on the "read first, buy later" principle.

2 Before the Project Starts

Before a machine learning project starts, it has to be prioritized and the team working on the project has to be built. Prioritization is unavoidable, because the team and equipment capacity is typically limited, while the backlog of projects an organization has might be very long.

2.1 Prioritization of Machine Learning Projects

The key considerations in the prioritization of machine learning project are *impact* and *cost*.

The impact of using machine learning in a broader engineering project is high when 1) machine learning can replace a complex part in your engineering project or 2) there's a high benefit in getting inexpensive (and probably noisy) predictions.

For example, some complex part of an existing system can be rule-based, with many rules, rules inside rules, and exceptions from rules. Building and maintaining such a system can be extremely difficult, time-consuming and error-prone. It can also be a source of major frustration for software engineers when they are asked to work on maintaining that part of the system. Can the rules be learned instead of programming them? Can the existing system be used to generate labeled data easily? If yes, such a machine learning project would have a high impact and probably low cost.

Noisy inexpensive predictions can be valuable, for example, in a system that dispatches a big amount of requests. Let's say many such requests are "easy" and can be solved quickly using some existing automation. The remaining requests are considered "difficult" and must be addressed manually. A machine learning-based system that recognizes "easy" tasks and dispatches them to the automation will save a lot of time for humans who will only concentrate their effort and time on difficult requests. Even if the dispatcher makes an error in prediction, the complex request will reach the automation, the automation will fail on it, and the human will eventually receive that request. If the human receives a simple request by mistake, it's also not a problem: that simple request can still be sent to the automation or processed by the human (to avoid the overhead of forwarding the request to automation).

The cost of the machine learning project is highly influenced by three factors:

- the difficulty of the problem,
- the cost of data, and
- the needed accuracy.

Getting the right data and the right amount of it can be very costly, especially if manual labeling is needed. The need for high accuracy can translate into the requirement of getting more data or building a more complex model, such as a unique architecture of a deep neural network or a nontrivial ensembling architecture.

When you think about the problem's difficulty, the main considerations are:

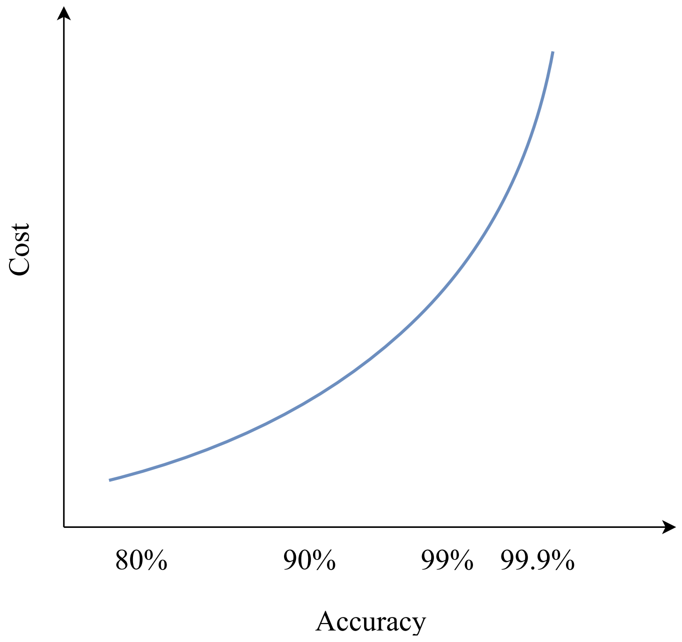

Figure 1: Superlinear growth of the cost as a function of accuracy requirement.

- whether an implemented algorithm or a software library capable of solving the problem is available (if yes, the problem is greatly simplified),
- whether a significant compute is needed to build the model,
- whether a significant compute is needed to run the model in the production environment.

The second driver of the cost is data. The following considerations have to be made:

- can data be generated automatically (if yes, the problem is greatly simplified),
- what is the cost of manual annotation of the data (i.e., assigning labels to unlabeled examples),
- how many examples are needed (usually, this cannot be known in advance, but can be estimated from known published results or the organization's own past experience).

Finally, one of the most important factors influencing the cost is the desired accuracy of the model. The property to be aware of is that the cost of the machine learning project grows superlinearly in the accuracy requirement as illustrated in Figure 1. Low accuracy can also be a source of significant loss in the future when the model will be deployed in the production environment. The considerations to make:

- How costly is each wrong prediction?
- What is the lowest accuracy level below which the model becomes too noisy for being practical?

2.2 Estimating Complexity of a Machine Learning Project

There is no standard method of estimation of how complex a machine learning project is other than by comparison with other projects executed by the organization or reported in the literature. There are several major unknowns that are almost impossible to guess with confidence unless you worked on a similar project in the past or read about such a project. The unknowns are:

- whether the required accuracy level (or the value of any other metrics important to you) is attainable in practice,
- how much data you will need to reach the required accuracy level,
- how many features (and which ones) are needed so that the model can learn and generalize sufficiently well,
- how large the model has to be (especially relevant for neural networks and ensemble architectures),
- how long will it take to train one model (in other words, how much time is needed to run one **experiment**) and how many experiments will be needed to reach the desired level of performance.

One thing you can almost be sure of is that if the required accuracy level is above 99% you can expect complications related to an insufficient quantity of labeled data. In some problems, even 95% accuracy is considered very hard to reach.

Another useful reference is the human performance on the task. If you want your model to perform as well as a human, this is typically a hard problem.

One way to make a more educated guess is to simplify the problem and try to solve it first. For example, you have a problem of classifying a set of documents into 1000 topics. Run a pilot project by focusing on 10 topics first, by considering documents belonging to other 990 topics as "Other". Manually label the data for these 11 classes (10 real topic plus Other). The logic here is that it's much simpler for a human to keep in mind the definitions of only 10 topics compared to memorizing the difference between 1000 topics[1].

Once you have your simplified problem with 11 classes, solve it and measure time on every stage. Once you see that the problem for 11 classes is solvable you can hope that it will be solvable for 1000 classes as well. Your measurements can then be used to estimate the time required to solve the full problem, though you cannot multiply this time by 100 to get an accurate estimate. The quantity of data needed to learn to distinguish between more classes usually grows superlinearly with the number of classes.

An alternative way of obtaining a simpler problem from a potentially complex one is to split the problem into several simple ones by using the natural slices in the available data. For example, you want to predict something about your customers and you have customers in

[1] To save even more time, apply clustering to the whole collection of unlabeled documents and only label documents belonging to one or a few clusters.

multiple locations. Solve the problem for one location only, or for customers in a specific age range.

The progress of machine learning project is nonlinear. The accuracy usually grows fast in the beginning, but then the growth slows down. Sometimes you see no growth and decide to add additional features potentially depending on external databases or knowledge bases. While you are working on a new feature or labeling more data (or outsourcing this task to an external vendor or team), no progress in the level of accuracy is happening at all.

Because of this non-linearity in progress, you have to make sure that the product leader (or the client) understands the constraints and the risks. Carefully log every activity and track the time it took. This will help not only in reporting but also in simplifying complexity estimations for similar projects in future.

2.3 Defining the Goal of a Machine Learning Project

The **goal** of a machine learning project is to build a model that solves or helps solving a certain business problem. The model is specified by the structure of its input (or inputs) and output (or outputs) and the minimum acceptable level of performance (as measured by accuracy of prediction or another performance metric).

The model will then be used as a part of a system that serves some purpose. In particular, the model can be used in a broader system to:

- *automate* (for example by taking an action on the user's behalf or by starting or stopping a certain activity on a server),
- *alert* or *prompt* (for example by asking the user if an action should be taken or a system administrator if the traffic seems suspicious),
- *organize* by presenting a set of items in an order that might be useful for a user (for example by sorting pictures or documents in the order of similarity to a query or according to user's preferences),
- *annotate* (for example, by adding contextual annotations to a displayed information or by highlighting, in a text, phrases relevant to user's task),
- *extract* (for example, by detecting smaller pieces of relevant information in a larger input, such as named entities in the text: proper names, companies, and locations),
- *recommend* (for example, by detecting and showing to a user highly relevant items in a large collection based on item's content or user's reaction to the past recommendations),
- *classify* (for example, by dispatching input examples into one, or several, of a predefined set of distinct named groups),
- *predict a quantity* (for example, by assigning a number, such as a price, to an object, such as a house),
- *synthesize* (for example, by generating new text, image, sound or an a diffrent object similar to other objects in a collection),
- *answer an explicit question* (for example, "Does this text describe this image?" or "Are these two images similar?"),

- *transform the input* (for example, by reducing its dimensionality for visualization purposes paraphrasing a long text as a short abstract, translating a sentence into another language, or augmenting an image by applying a filter to it),
- *detect novelty* or an *anomaly.*

Almost any business problem solvable with machine learning can be defined in a form similar to one from the above list. If you cannot define your business problem in such a form, it's likely that machine learning is not the best solution in your case.

A successful model has the following four properties:

- it respects the input and output specifications and the minimum performance requirement,
- it benefits the organization (measured via economies, increased sales or profit),
- it benefits the user (measured via productivity, engagement, and sentiment),
- it's scientifically rigorous.

A scientifically rigorous model is characterised by a *predictable behavior*, for the input examples that are similar to the examples used for training, and is *reproducible*. The former property means that if input feature vectors come from the same range of values as the training data, then the model, on average, has to make the same amount of errors as was observed on the holdout data when the model was built. The latter property means that a model with similar properties can be *easily* built once again from the same training data using the same algorithm and values of hyperparameters. The word *easily* means that no additional analysis work is necessary to rebuild the model, only the compute power.

When defining the goal of machine learning, make sure you solve the right problem. To give an example of the incorrectly defined goal, imagine that your client has a cat and a dog and needs a system that lets their cat in the house but keeps their dog out. You might decide to train the model to distinguish cats from dogs. However, this model will also let *any cat* in and not just *their* cat. Alternatively, you may decide that because the client only has two animals, you will train a model that distinguishes between those two. In that latter case, because your classification model is binary, a raccoon will be classified as either the dog or the cat. If it's classified as the cat, it will be let in the house[2].

2.4 Structuring a Machine Learning Team

There are two cultures of structuring a machine learning team depending on the organization.

The first culture says that a machine learning team has to be composed of data analysts (or scientists) who collaborate closely with software engineers. In such a culture, the software engineer doesn't need to have very deep expertise in machine learning but has to understand the vocabulary of their fellow data analysts or scientists.

[2]This is why to have the class "Other" in your classification problem is almost always a good idea.

According to the second culture, all engineers in a machine learning team must have a combination of machine learning and software engineering skills.

There are pros and cons in each culture. The proponents of the former say that each member of the team has to be the best in what they do. A data analyst must be an expert in many machine learning techniques and have a deep understanding of the theory to come up with an effective solution to most problems fast and with the least effort. Similarly, a software engineer has to have a deep understanding of various computing frameworks and be capable of writing efficient and maintainable code.

The proponents of the latter say that scientists are hard to integrate with software engineering teams. Scientists care more about how accurate their solution is and often come up with solutions that are impractical and cannot be effectively executed in the production environment. Also, because scientists usually don't write efficient and well-structured code, the latter has to be rewritten into the production code by a software engineer, which, depending on the project, can turn out to be a daunting task.

Besides machine learning and software engineering skills, a machine learning team may include experts in data engineering (or data engineers) and experts in data labeling.

Data engineers are software engineers responsible for ETL (for Extract, Transform, Load). These three conceptual steps are part of a typical data pipeline. Data engineers use ETL techniques and create an automated pipeline in which raw data is transformed to analysis-ready data. Data engineers design how data must be structured and integrated from various resources. They write on-demand queries on that data or wrap the most frequent queries into fast APIs to make sure that the data is easily accessible by analysts and other data consumers. Generally, data engineers are not expected to know any machine learning.

In most big companies data engineers work separately from machine learning engineers in a data engineering team.

Experts in data labeling are responsible for three activities:

- manually or semi-automatically assign labels to unlabeled examples according to the specification provided by machine learning engineers or analysts,
- build labeling tools,
- manage outsourced labelers,
- validate labeled examples for quality.

Again, in big companies, data labeling experts may be organized in two or three different teams: one or two teams for data labelers (for example, one local and one outsourced) and a team of software engineers and a UX specialist responsible for building labeling tools.

Finally, DevOps engineers work closely with machine learning engineers to automate model deployment, loading, monitoring, and occasional or regular model maintenance. In smaller companies and startups, a DevOps engineer may be part of the machine learning team, or a machine learning engineer could be responsible for the DevOps activities. In big companies, DevOps engineers employed in machine learning projects usually work in a larger DevOps team. Some companies introduced the MLOps role whose responsibility is to deploy machine learning models in production, upgrade those models and build data processing pipelies involving machine learning models.

2.5 Contributors

I'm grateful to the following people for their valuable contributions to the quality of this chapter: Christopher Thompson, Brad Ezard, Sylvain Truong, Niklas Hansson, Zhihao Wu, Max Schumacher, and Kelvin Sundli.