**Machine Learning Rules of Thumb**

A few solid guidelines to give your model a good starting point

[[Hylke C. Donker](https://medium.com/@hylke.donker?source=post_page-----b50232b4b2f8--------------------------------)](https://medium.com/@hylke.donker?source=post_page-----b50232b4b2f8--------------------------------)

[[Towards Data Science](https://towardsdatascience.com/?source=post_page-----b50232b4b2f8--------------------------------)](https://towardsdatascience.com/?source=post_page-----b50232b4b2f8--------------------------------)

[Hylke C. Donker](https://medium.com/@hylke.donker?source=post_page-----b50232b4b2f8--------------------------------)

·

[Follow](https://medium.com/m/signin?actionUrl=https%3A%2F%2Fmedium.com%2F_%2Fsubscribe%2Fuser%2Fea1bfe4db7a8&operation=register&redirect=https%3A%2F%2Ftowardsdatascience.com%2Fmachine-learning-rules-of-thumb-b50232b4b2f8&user=Hylke+C.+Donker&userId=ea1bfe4db7a8&source=post_page-ea1bfe4db7a8----b50232b4b2f8---------------------post_header-----------)

Published in

[Towards Data Science](https://towardsdatascience.com/?source=post_page-----b50232b4b2f8--------------------------------)

·

7 min read

·

Mar 31, 2022

96

1

Sadly, the success of a machine learning project is not guaranteed. You therefore start with a quick and dirty implementation. One of the first questions that arises is “How much data do I need”? Ask any machine learning practitioner, and they will tell you: “it depends”. And they would be right! There are no universal truths when it comes to machine learning. It is as much an art, as it is a science. I do find, however, that for most aspects of machine learning, there are rules of thumb which tend to be good starting positions. Here, I’ve tried to collect a few common rules.

**How much Data?**

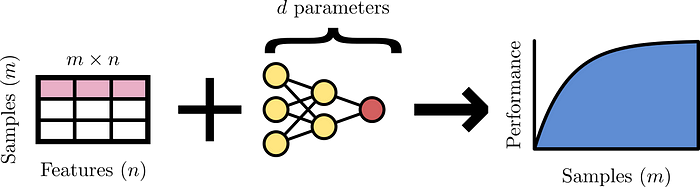


Fig. 1: Model performance is linked to the number and quality of samples collected, the amount and utility of the features, and the capacity of the model. Image by author.

Number of samples (*m*), features (*n*), and model parameters (*d*) form the holy trinity of machine learning. Most rules of thumb can largely be brought back to this triad (Fig. 1). Lets take a closer look.

**How many samples?**

Performance typically scales as log *m,* where *m* is the number of samples (Fig. 1), and is usually bounded from above by the noise in the labels. Therefore, when the training data is labelled by humans, this bound usually corresponds to human level performance. It may therefore be useful to focus on data quality, rather than quantity, as suggested by the [data-centric AI](https://datacentricai.org/) movement.

In general, more samples are required for regression than for classification [1]. Specifically, given *n* input features and *C* categories to classify, it is suggested to have at least [1]

*m* ≥ 10 *n* · *C*,

samples. For regression

*m* ≥ 50 *n,*

samples are suggested.

**How many parameters?**

Recall from your linear algebra class that to solve a linear system with *d* degrees of freedom, you need *d* constraints. For linear regression, each sample is a constraint. Therefore, to fix *d* parameters, you need *at least* as much samples — otherwise you system is said to be [underdetermined](https://en.wikipedia.org/wiki/Underdetermined_system). More generally, by interpreting a model’s parameters as degrees of freedom, a common heuristic is a ten-fold overdetermined system

*d* ≤ *m/*10 ,

although more conservative bounds for neural networks, such as *d* ≤ *m*/50, are also suggested [2]. In turn, having determined the number of parameters, *d,* can help you decide if the number of features, *n,* needs to be reduced.

Caution is, however, warranted because for many models, e.g., probabilistic graphical models, the number of constraints can be *O*(*n*) and independent of the sample size *m*.

**Which model?**

**Structured data**

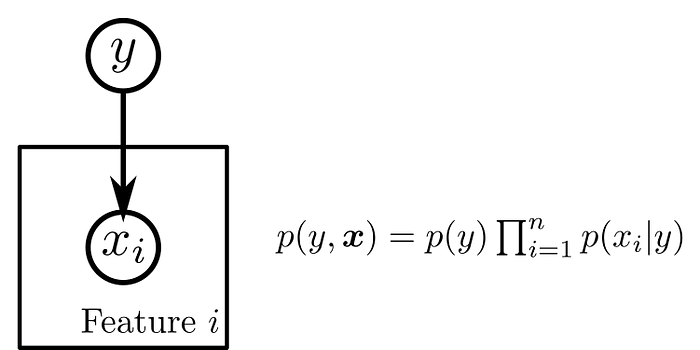


Fig. 2: Naive Bayes represented as a probabilistic graphic model. Nodes denote distributions and arrows indicate conditional dependence. The plate around the feature indicates that the relation is copied (in this case *n* times) and indexed by *i*. Image by author.

* Small dataset (20 ≤ *m* ≤ 10³): [naive Bayes](https://scikit-learn.org/stable/modules/naive_bayes.html), [elastic net](https://scikit-learn.org/stable/modules/linear_model.html#elastic-net)

When samples are scarce, use interpretable models with large inductive bias. Specifically, when [calibration](https://en.wikipedia.org/wiki/Calibration_(statistics)) is a concern, go for logistic regression. Otherwise, its generative counterpart naive Bayes [3,4] is a good candidate (Fig. 2). (For a discussion of their equivalence see Ref. [3]). The latter outperforms the former in the small data regime [5]. The intuition here, is that imposing additional stringent model assumptions reduces the variance of the model’s coefficients. Elastic net (or, linear regression with both a *L*₁ and *L*₂ penalty on the weights) works in a similar way, *de facto* codifying a preference for sparse and shrunken coefficients.

* Intermediate dataset (10³ ≤ *m* ≤ 10⁴): [gradient boosted trees](https://catboost.ai/)

Kaggle competitions have learned us that gradient boosted trees are incredibly versatile and work well on many real life, noisy, datasets. For larger datasets, the more flexible gradient boosted trees can squeeze more juice from your data relative to simpler models, at the expense of inspectability. Fortunately, advancements in explainable AI (e.g., [SHAP](https://github.com/slundberg/shap)) have made these previously blackbox models fairly easy to understand and interrogate.

* Large dataset (10⁴ ≤ *m*): neural networks

Large training corpora combined with neural networks are a match made in heaven, as demonstrated by most advancements in AI. Training scales linearly with the size of the dataset (i.e., ∝ *m*, for a given set of weights) making neural networks specifically suitable for large datasets. Moreover, TensorFlow and PyTorch have made neural networks blazingly fast and easy to scale across multiple machines.

**Unstructured data**

When your dataset consists of unstructured data such as images, text, or audio, it is recommended that you piggyback on existing pre-trained models. To fine tune an image classifier, you may need as little as *m* = 10 examples per class. You can browse [model zoo](https://modelzoo.co/) or [TensorFlow Hub](https://tfhub.dev/) to look for pre-trained models that fit your need. Alternatively, there are no-code cloud services such as Amazon’s [SageMaker](https://aws.amazon.com/sagemaker/) or Google’s[Vertex AI](https://cloud.google.com/vertex-ai) to help you do all the heavy lifting.

**How much regularisation?**

One way to help your model generalise beyond the training set is to put penalties on the size of your model’s weights **w**. This is called regularisation. Two popular penalties are the [Manhatten norm](https://en.wikipedia.org/wiki/Norm_(mathematics)#Taxicab_norm_or_Manhattan_norm) (or, *L*₁ norm) — after the city, because distances resemble travelling along a rectangular street grid — and the “standard” Euclidean norm (or, *L*₂ norm).

When regularising, verify that *all* features are scaled to order unity (dimensionless) by, e.g., standardisation. This ensures that the penalty affects all weights equally. Ball park estimates can be obtained by studying two linear regression cases that can be solved in closed form, namely, [Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html) (i.e., *L*₁ regularisation *λ*₁|***w***|₁ of weights ***w***) and [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) regression (*L*₂ regularisation *λ*₂|***w***|²₂ of weights ***w***). When the features are centred and uncorrelated, their solutions can be expressed in terms of the unpenalised solution [a,b]. Lasso clips (or, truncates) all unpenalised coefficients below *λ*₁ [a]. A reasonable starting point can therefore be *λ*₁ = 0.1. Ridge, on the other hand, only shrinks their size towards zero [b]. To pick λ₂, you might want to take into account to what extent your system is overdetermined. A note of warning: scikit-learn uses slightly different conventions for their objectives (cf. the 1/(2 *m*) factor in [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) versus [Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)).

**How many hash buckets?**

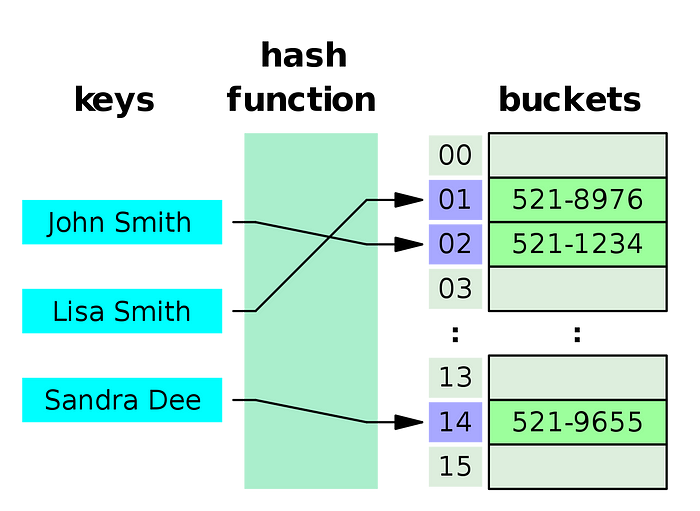


Fig. 3: Illustration of the hashing trick, where items are assigned to a bucket by computing a hash. Image by Jorge Stolfi and obtained from Wikipedia’s [Hash table](https://en.wikipedia.org/wiki/Hash_table) article, copyright under [CC BY-SA 3.0](https://creativecommons.org/licenses/by-sa/3.0/).

The hashing trick groups items by generating a new index based on the item’s hash (Fig. 3). For example, the English language contains *n* = 0.6 million words [6] which we might want to reduce to a fixed size *n*↦ *n’* before training. In addition, new — out-of-vocabulary — words are being invented as we speak (literally!). This poses a problem during serving, since these new words weren’t part of the training set. Feature hashing resolves both problems at the expense of fidelity. The fidelity is controlled by the number of hashing collisions which, in turn, is determined by the number of hash buckets *n*’. To tackle out-of-vocabulary items, it is suggested to take [1]

*n*’ = *n*/5,

hash buckets. I have yet to find a good heuristic when the goal is to reduce the vocabulary size. However, choosing *n*’ hash buckets so that the model’s size *d ≤ m/10*seems like a reasonable guesstimate.

**Size of my embedding?**

In neural networks, embedding layers condensesparse vectors — containing mostly zeroes — into a small dense representation. Think of converting a one-hot encoded word — where all entries are zero except for the word’s index — into a word embedding. But how much should I compress my high cardinal input data? Given a list of *q* categories to embed, I’ve came across the following two heuristics for choosing the size *d*ₑ of your embedding. Namely, square root scaling [1]

*d*ₑ = 1.6 √*q,*

or taking the fourth root [7]

*d*ₑ = ∜*q*.

**Discussion**

The [no free lunch theorem](https://en.wikipedia.org/wiki/No_free_lunch_theorem) tells us that there is no silver bullet when it comes to machine learning. Therefore, these rules of thumb are necessarily more wrong than right. Nevertheless, they can be good starting points to home in on estimates specific to your dataset. Hyperparameter tuning together with cross validation can help you find your dataset’s sweet spot.

As you’ve noticed, this document is far from exhaustive. For example, we haven’t touched upon all the hyperparameters to train a neural network. For this, I would recommend Jeff Macaluso’s excellent [deep learning list](https://jeffmacaluso.github.io/post/DeepLearningRulesOfThumb/).

I would love to know your heuristics. Please expand this collection by leaving your suggestions in the comments.

**Acknowledgements**

I would like to thank [Rik Huijzer](https://huijzer.xyz/) for the discussions that initiated this post and for proofreading.

**Footnotes**

[a]: Assume the features are centred and uncorrelated. Let **w\*** be the maximum likelihood estimate of linear regression without regularisation. The Lasso regression (linear regression with *L*₁ regularisation *λ*₁|***w***|₁ for weights ***w***) solution turns out to be [3]

***w*** = sgn(***w*\***) ReLU(***w*\*** -*λ*₁),

where sgn(*x*) is the sign of *x* and ReLU(*x*) is the rectified linear unit.

[b]: Like in [a], assume the features are centred and uncorrelated and **w\***the unregularised solution. For [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html) regression (linear regression with *L*₂ regularisation *λ*₂|***w***|²₂ of weights ***w***) the weights shrink their size [3]

***w*** = ***w***\* / (1 + λ₂),

towards zero.

**References**

[1]: Valliappa Lakshmanan, Sara Robinson, and Michael Munn. *Machine learning design patterns*. O’Reilly Media, 2020.

[2]: Ahmad Alwosheel, Sander van Cranenburgh, and Caspar G. Chorus. “Is your dataset big enough? Sample size requirements when using artificial neural networks for discrete choice analysis.” *Journal of choice modelling* 28 (2018): 167–182.

[3]: Kevin P. Murphy, *Probabilistic machine learning: an introduction*. MIT press, 2022.

[4]: Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[5]: Andrew Ng and Michael Jordan. “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes.” *Advances in neural information processing systems* 14 (2001).

[6]: <https://en.wikipedia.org/wiki/List_of_dictionaries_by_number_of_words>

[7]: <https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html>