Maciej Wielgosz

Machine Learning Primer

October 2019

What this handout is (and isn't) about?

It is about:

- useful things to know (and hard to find in one place) when entering into Machine Learning (and especially Deep Learning) field,
- how to avoid common pitfalls.

It is not about:

- detailed algorithms' explanations,
- frameworks and programming patterns.

Chapter 1

Landscape

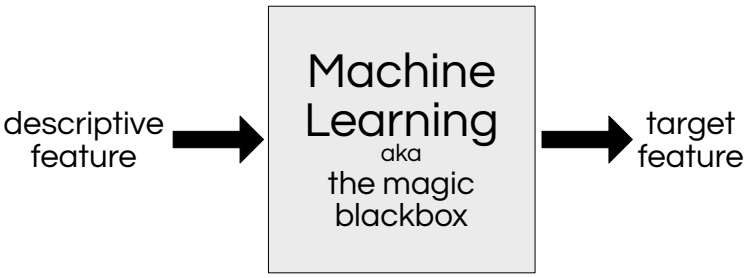
Machine Learning problem.
Relationship with other fields.
Classification of ML algorithms.
Classification.
Clustering.
Regression.
Dimensionality reduction.

Machine Learning problem

What is Machine Learning (ML)?

Giving computers the ability to learn without explicitly programming them.

— Arthur Samuel, 1959

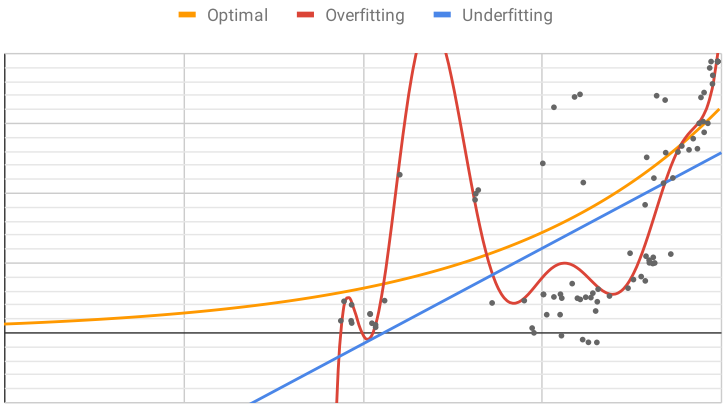


Why is this problem ill-posed?

There are many possible models that agree on what predictions should be for the existing (training) dataset but disagree about unavailable data and yield different potential results.

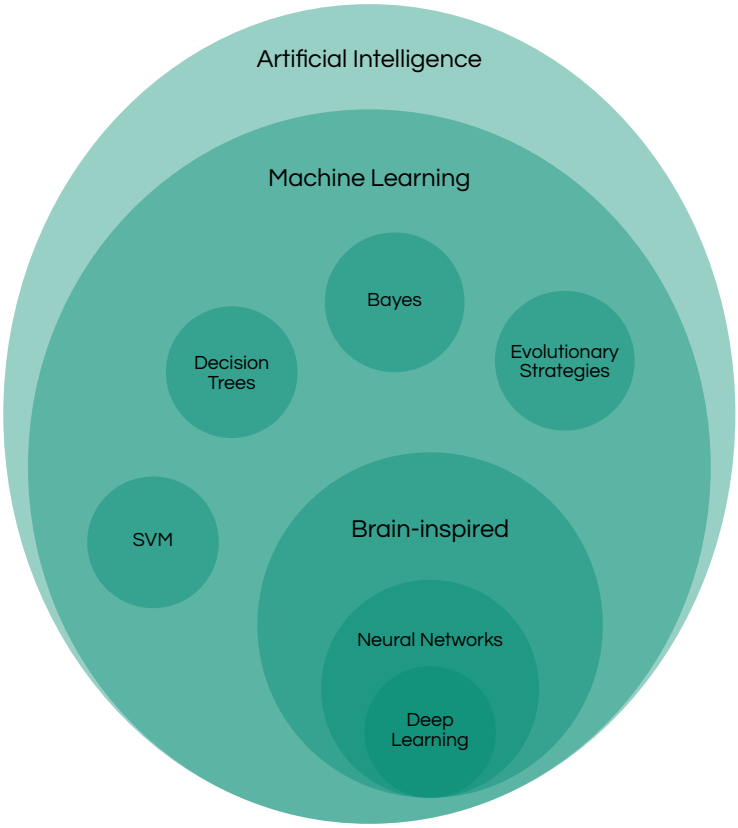
What to look out for when creating ML models?

- Data-induced biases
- Confidential/personal data leakage
- Generalization properties (is model under- or overfitting?)



Your summary

Relationship with other fields

How ML and its various sub- and supersets relate?	 Only selected algorithms included.
Why is Deep Learning (DL) called as such?	Because of the number of neural network layers involved. The "deep" started with about 16 layers in VGG-16 model and reaches values like 152 layers in ResNet-152 (notice the convention of adding the number of layers to the model name).

Your summary

Classification of ML algorithms

How the ML algorithms can be classified according to their usage?

	supervised	**unsupervised**
discrete	classification (categorization)	clustering
continuous	regression	dimensionality reduction

What is the difference between supervised and unsupervised learning?

In supervised learning, the model is shown the correct "answer" (label), while in unsupervised there are no or very few constraints on what the model can do with it (e.g., divide the data into three clusters).

How the reinforcement learning fits into this?

Reinforcement learning is a form of supervised learning, just stretched in time.
Usually during the supervised training the model performs only a single action and immediately knows whether it was correct ("I'm 75% sure this is a cat. — No, it's a bird.").
In the reinforcement learning, however, the model needs to generate a sequence of actions, observing its environments state, and the reward is given at the end ("Turn right, right, left and right again. — You've escaped the dungeon").

What if I have (semi-)continuous data, but want discreet results?

That depends on the model. Some will be able to handle it directly (e.g., neural networks) and yield expected results, while for others the manual feature extraction or data bucketization (converting it to a discreet form) may be required.

Your summary

Classification

What the classification is about?	Assigning a new sample (e.g., photo) to the predefined classes. It usually takes the form of the probability of the sample belonging to a particular class.
What are the common classification metrics?	• Top-1 accuracy – how often the model is correctly assigning the highest probability to the real sample class ("there's 60% probability that this is a cat. — It's a cat"). • Top-5 accuracy – how often the model is assigning to the real class one of the top five probabilities ("I'm 30% sure it's a pillow, 27% – a cat, 23% – a tiger, 9% – a wolf, and 7% – a lion — It's a tiger"). • F_β score – it combines into a single value how accurate the model is when assigning a particular class ("three out of five images that I said were cats really were cats" – precision) and how good it is at finding all samples that belong to that class ("I found only three real cats out of the 20 in the whole dataset" – recall). There β parameter controls how important (according to the system designer) is one versus other ("I think precision and recall are equally important – $\beta = 1$"). In its simplest form, the F_β score is designed for binary classification (only two classes). There are various strategies for combining the per-class F_β scores into a single score when there are many classes.

Your summary

Clustering

What the clustering is about?	Assigning a new sample to one of the clusters (groups, sets) determined during training ("this shape is most similar to ones in this group"). Depending on the algorithm, the target number of classes may need to be specified before training. It may be determined via some heuristics or during the optimization process.
What are the common clustering metrics?	The clustering metrics can be divided into two groups: internal and external. Internal evaluation is calculated using the clustering results, for example by measuring the distance between clusters centers (centroids) – inter-cluster distance – or points in the cluster and its center – intra-cluster distance. The distance itself can be measured in various ways, not only as Euclidean distance. Two of the common internal distance-based metrics are: • Davies–Bouldin Index (DBI) – compares the average distance of the points in the cluster from the centroid to the inter-cluster distance. • Dunn Index (DI) – compares the smallest distance between clusters to the biggest intra-cluster distance. The external metrics usually require labels to be known, and therefore are mostly similar to classification metrics (e.g., Jaccard Index, used to calculate the similarity between two datasets, or F_β score).

Your summary

Regression

What the regression is about?	Creating a model which outputs match the real data best (e.g., given a set of 2D points, find a line that will best represent their positions). The regression can be used for modeling the so-called "normal" data – predicting the next value(s) or showing how the data should look like if no anomaly occurred. Therefore, usually, before training the regression models, the abnormal data is removed from the dataset.
What are the common regression metrics?	• Mean Absolute Percentage Error (MAPE) – how much (on average) the modeled signal differs from the original as a **percentage of the original value**. Will not work directly if original signal value equals zero at some point. • Mean Squared Error (MSE) – average squared difference between the modeled signal and the original. The squaring allows to **penalize bigger errors** more than small ones. • Root Mean Squared Error (RMSE) – root taken from MSE. Similarly to MAPE, allows to somewhat correlate the error value to the original signal values.

Your summary

Dimensionality reduction

What the dimensionality reduction is about?	Taking the multidimensional data and reducing it to a form that, e.g. human can use in visualization (2D, 3D) or some algorithm (ineffective/inefficient for multidimensional data) can use. The basic idea is to take the data and transform it so that if two samples are "close" to each other in the original space, they will also be "close" in the reduced space. Of course, what is understand by "close" depends on the applied distance metrics.
What are the common dimensionality reduction algorithms?	• Principal Component Analysis (PCA), and all its variants, • T-distributed Stochastic Neighbor Embedding (t-SNE) – developed with visualization in mind.
How to know if dimensionality reduction works correctly?	While the metrics allowing to compute some form of distance between two vectors can be used, they are not really useful, since they will favour the algorithms that use similar metrics at their base. Usually, when dimensionality reduction is a part of the data preprocessing, its quality is determined by how well the whole system performs. In such case, the dimensionality reduction parameters are optimized by looking at how system quality changes while the top part (usually classifier) is left as similar as possible.

Your summary

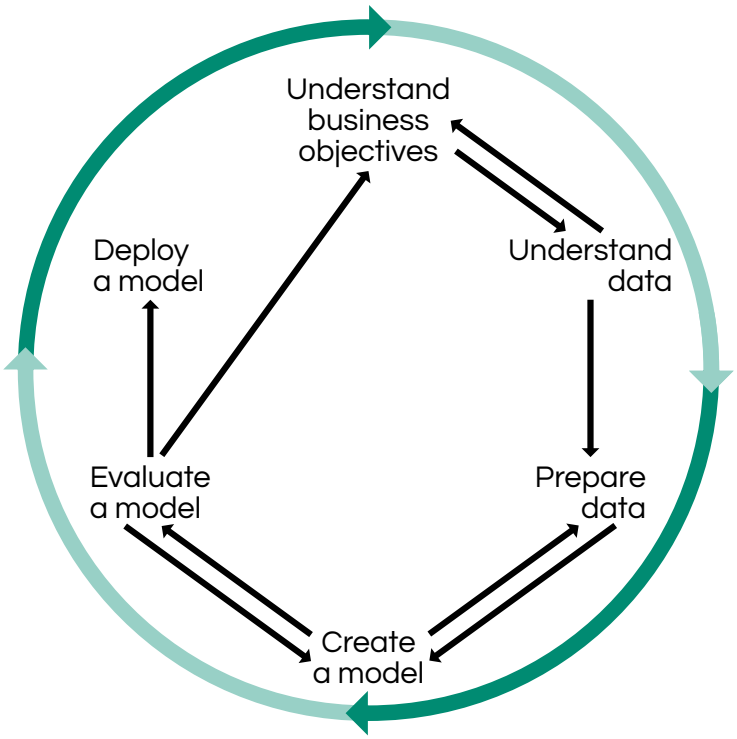
Chapter 2

From data to an application

Data Science process.
A model's life-cycle.
Data flow through the software.
Understanding business objectives.
Data understanding.
Data cleaning.
Dataset preparation.
Data normalization.
Data normalization II.
Dataset maintenance.
Results logging.
Computing portability.

Data Science process

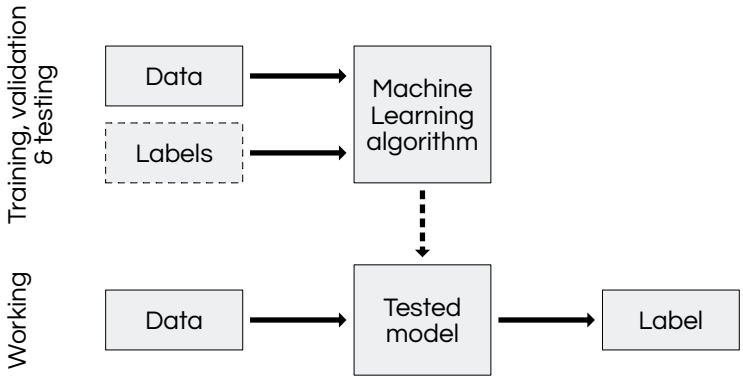
How does the model creation process look like from a **human** perspective?



Your summary

A model's life-cycle

What phases there are in a standard model's life-cycle?



1. Training, validation, and testing.
   (a) During **supervised** training, the model is "shown" both data and labels. In **unsupervised** learning, the model adjusts itself to the data to, e.g., find the clusters' centroids.
   (b) The validation set is used to see how well model performs on data it didn't see before. Usually it is interleaved with the training phases. The metrics acquired from validation data can be used to, e.g., earlier stop the learning process if there is no further progress or when the models starts to overfit.
   (c) The testing dataset is used to take a final measure of how well model performs, and how various **models compare to each other**. It shouldn't contain any data from training or validation sets, and shouldn't change between tests.
2. Working – the model is used on the new, unlabelled data.

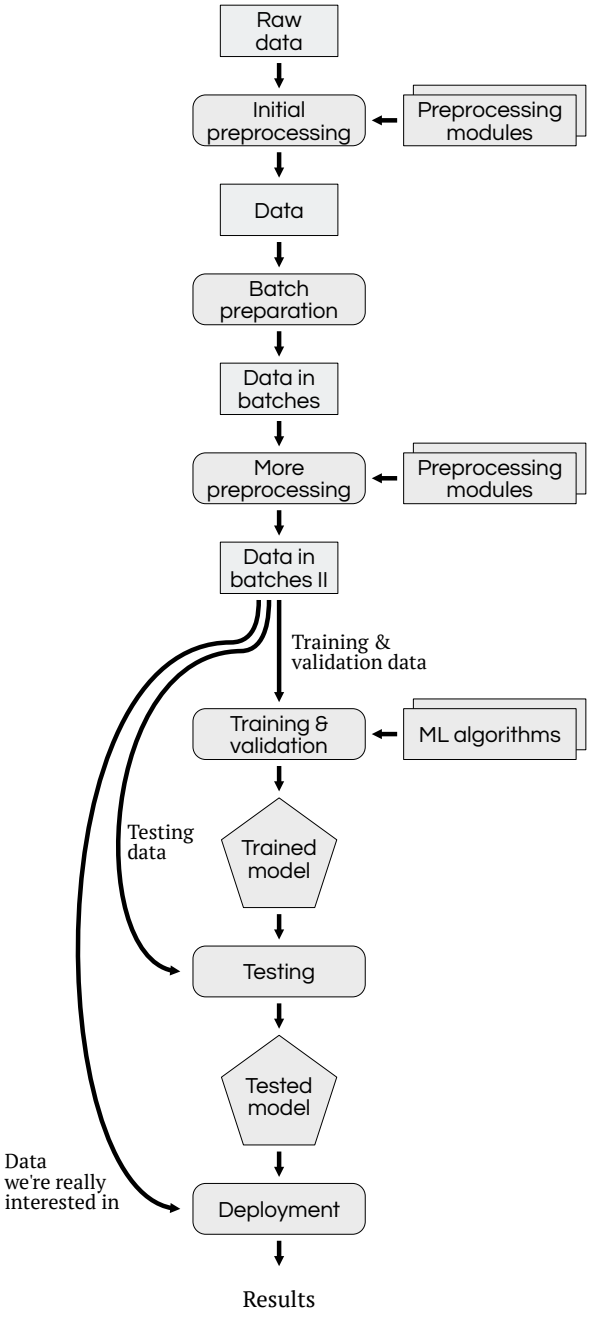
Is it possible to integrate new data into the model after it was trained?

Yes, depending on the model various strategies can be adopted. Sometimes the model can be updated offline (outside of the working system) and then replaced, the other times (usually in the case of novelty detectors) it may be possible to integrate the new data as it arrives (online learning).

Your summary

Data flow through the software

What is the standard data flow in the software?



Your summary

Understanding business objectives

What is the key task at this stage?	Selection of **appropriate quality assessment measures**.
Why is this so important?	So that the project expectations and outcomes can be communicated to non-ML people: • domain experts, • project managers, • end clients, • and various other stakeholders.
How to know if appropriate metrics were chosen?	They should be: • understood by all involved parties (to avoid "but we thought it would do something else" at later stages), • address the real business need (because why optimize something with no consequence?), • calculable from the available data (i.e., no expert is needed each time to verify how well the model performs), • **interpretable** – to allow comparison between models (e.g., "this model yields three times as many errors as the other").
What else to do before proceeding?	• Establish current "system" performance if possible (e.g., "trained personnel member can check five objects per minute and finds eight out of ten broken ones, and there are on average ten broken ones in every thousand") – often companies do not know that! • Establish a dummy model performance – what if the model would always say "it's ok" or assigned "ok/broken" labels at random with known probability (10/1000 is broken)? • Knowing the above – what **performance the final model needs to achieve** to justify its deployment?

Your summary

Selection of the appropriate quality assessment measures is the key of the successful project.

Data understanding

What should you know about the data?	• Possible value ranges, • How (and if) the series synchronize with one another, • Relationships between the series (are they correlated?), • How much data (if any) is missing? • Anything else you or data provider deem important – and keep asking until you have a sufficient grasp, • Visualize the data – ask the provider, maybe they already have some tools they use.
What to do if the data is heterogenous?	Design a common container for them (e.g., common image format, all numbers in float64). Ask the provider about raw data, which may be more suitable as a model's input.

Your summary

Data cleaning

How to handle missing data?	There are several strategies, among them: • ignore – the gaps split the series into smaller fragments, • fill with first/last available value – effective for very short gaps, • interpolate – selection of the interpolation strategy can be important.
What to do with noise/outliers/anomalies?	That depends on the target application. In some cases, when the data is abundant, and the model should work on as raw data as possible, removing them may not make much sense. Other times the algorithm may expect "clean" data, e.g., to later be able to detect anomalies. The key point is to understand what really is an outlier – for that, thorough data understanding and/or consultations with domain experts may be needed.
Should data be normalized at this point?	**No.** The data should only be normalized (assuming the algorithm requires it) **after it was split into subsets**. The parameters required for normalization (e.g., minimum, maximum, mean, and standard deviation) should only be found for the **training set** and then applied for the other subsets.

Your summary

Dataset preparation

How to split the data?	Separate the data into three (or even four) sets. Following doing it according to Pareto rule is usually a good option: 1. 20% of the data goes to the **testing set**, 2. 20% of the rest, so 16% of the full dataset goes into the **validation** set, 3. if hyper-parameters **optimization** is anticipated, in may be worth to set aside the dataset for that purpose (approximately 13% of the full dataset), 4. the rest (64% or about 51% if optimization set was separated) goes into the **training** set.
Why create a separate set for the optimization?	• If training dataset is used for optimization, there is very high chance the most-overfitting model will be selected as the best – a very undesirable outcome. • If testing dataset is used for optimization, a "data leak" occurs – it becomes a kind of indirect training data for the meta-model (the one controlling the optimization process). As a result, there is no way to truly measure how the target model behaves for unseen data (e.g., how well it will perform after deployment). • If validation dataset is used, it may affect training process, e.g., when validation metrics are used to control when the training should stop (so-called early stopping) or are used to adjust parameters like learning rate. However, if the dedicated optimization set is not available, using the validation one in its place seems to be the safest.
What if there is not enough data to create so many subsets and still get meaningful results?	The **cross-validation** approach should be used. The dataset is first divided into several (e.g., five) subsets, and each of the subsets is used as a test set while the rest is used for training/validation (the same model hyper-parameters, but training from scratch). Afterward, the test metrics are compared – the more similar they are, the better.

Your summary

Data normalization

Why normalize data?	There are several reasons: • when data channels have different ranges, it may be necessary to bring them to common one to calculate the overall metrics (especially in regression case), • it may allow for easier interpretation of the regression quality metrics (e.g., if we know the data is n range $[-1, 1]$, the RMSE at the level of 0.8 is not looking very promising), • some algorithms handle normalized data better than not normalized (yield better results or arrive at them faster), • especially in neural networks case, the range of the data affect the resulting rage of the trained network weights; this may influence, e.g., how well such a network can be quantized (and consequently, if it can be put into the hardware), • when working with fixed-point representation, it allows to avoid the under-usage of the data representation's dynamic range.
What are the common normalization approaches?	• Min-Max Scaling – this is probably the most common one when someone is talking about normalization. It is based on finding the minimum and maximum values in the data, and then scaling it so that minimum equals 0 (or sometimes -1) and maximum equals 1. It is very sensitive to outliers, therefore they should not be taken into account during minimum/maximum search. • Studentization[a] – based on finding the mean and the standard deviation of the data. It is more resistant to outliers, however there is no guaranteed output range, since it depend on the data distribution. Sometimes the studentization outputs are additionally clipped.

[a] named after William Sealy Gosset, who wrote under the pseudonym Student

Your summary

Data normalization II

How to determine the normalization parameters?	They should be calculated using **training set only**, and then applied when normalizing other subsets. For **each of the data channels** (e.g., red, green and blue in case of the images) separate values should be found, however they should be based on **all available series**, i.e., they should not be calculated on per-series basis. If there is too much data to fit into the memory at once, appropriate techniques (e.g., running mean and variance) should be used.
What is data bucketization and how it fits into all of this?	It is a process of mapping input data into bins (buckets) – similarly to how calculating a histogram works. Data bucketization can be used as an alternative to the normalization, since the bins can be then re-mapped to semi-continuous values (e.g., by using the bin edge or middle point). There can be various strategies for finding the bins' edges – starting with the simplest equal-width bins, through more advanced taking data distribution into account. Data bucketization can be used as a form of data stream compression. Additionally, sometimes mapping input data to the limited and well-defined range may also positively affect the training process of the model.

Your summary

Dataset maintenance

What points to consider when designing the dataset format?	• Format and location of the raw data: – CSV, JSON, XML, etc. kept on some hard drive? – Maybe stored in a database? • How will the data be fed into the model? – Will all data fit into the memory? – Look for the data streaming methods in the chosen framework: iterators, FIFOs, queues, etc. • Serialization: – Is it needed (e.g., to speed up the computations or because framework can easily handle it)? – Check what formats are supported out-of-the-box by your framework of choice. • Does the approach match the size of the dataset and how it will be **used in production**?

Your summary

Results logging

What things to keep in mind about results logging?	The most important point about results logging is to **do it**. The rest are just enhancements. • Start small, e.g., keep a spreadsheet with a few columns and extend it as needed. • Keep checkpoint of the models (and be able to associate it with specific result). • If dataset changes or you have several versions (e.g., small for local code debugging, medium for initial hyper-parameters optimization and full for actual tests) – keep tabs on that too, and indicate in results what dataset was used. • Use version control for software and, again, associate used version (tag, commit, branch) with the results. • Dump the hyper-parameters used during particular test. • Get team involved – work out the common logging scheme. • Use online visualization, such as Tensorboard or Visdom. That way you can observe the progress and "babysit" the training process. • When using multiple machines, log the name of the machine that was used for running the experiment. • Avoid logging a lot of stats which you cannot directly digest or share with the team members – although "hide" function for spreadsheet columns can help. • Automate results logging as much as possible (e.g., automatic dump of core settings to CSV).

Your summary

The most important point about results logging is to **DO IT**. The rest are just enhancements.

Computing portability

How to develop and debug models?	• Do it on a local machine. • Prepare a dataset which is appropriate (small and representative) – just enough to run basic tests and see if anything fails.
How to ensure the results are repeatable?	• Prepare a single dataset version and spread it across all the test machines you are using. • Make the software between machines as consistent as possible (e.g., the same version of framework, Python, serialization libraries).
How to ensure models portability?	• Keep models as a separate module and import as needed. • Use the general switches for GPU/CPU for debugging. • Always save checkpoints in the "basic" CPU format, and only convert for a particular GPUs setup on load.

Your summary

Chapter 3

When you need a hint

Recommended resources.
How to digest (arXiv) papers.
How to deal with "scientific" code.

Recommended resources

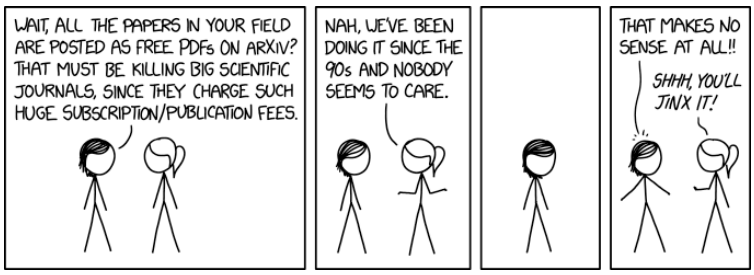
John D. Kelleher, Brian Mac Namee and Aoife D'Arcy

Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies, The MIT Press, 2015

Ian Goodfellow, Yoshua Bengio and Aaron Courville

Deep Learning, The MIT Press, 2016

arXiv



"ArXiv", https://xkcd.com/2085/, CC BY NC

Your summary

How to digest (arXiv) papers

What is the recommended "reading procedure"?	Do it in three steps (and stop if you notice it does not meet your needs): 1. Skim through, looking for main ideas. 2. Read skipping the math. 3. Read in detail.
What to look at in particular?	• Is the source code available? Usually in the form of GitHub/Bitbucket repo. It may contain a lot of details which are not provided in the paper. • Is the dataset available? So you ma start with reproducing the results. • Is the trained model provided ("checkpoint")? It may save a lot of time. • How detailed is the description? Does it contain all crucial parameters, necessary to run it from scratch on your machine? Info about learning rate, batch size, used optimizer etc. may be crucial to replicate the results. • Who are the authors? Or more specifically: what their affiliations are. Companies often have a dedicated department to ensure their proprietary data will not leak, which results in obfuscation of the published research results.

Your summary

How to deal with "scientific" code

How does it differ from "normal" code?	Very often it is only created for the purpose of the single experiment and then abandoned. Documentation is scarce at best, and support non-existing.
What to look for?	• Which framework and in what exact version it was designed to work with? Data science frameworks, especially for neural networks, are evolving very fast and version mismatch can cause very unexpected errors. • How many people use it? The bigger the number, the better. • How many ports of the original version are available? You will have something for the reference in case the original one does not work (or you need one using different framework).

Your summary