https://pubs.acs.org/doi/book/10.1021/acsinfocus.7e5033

ACS IN FOCUS

Machine Learning in Materials Science

Keith T. Butler

Rutherford Appleton Laboratory

Felipe Oviedo

Microsoft AI for Good

Pieremanuele Canepa

National University of Singapore

https://pubs.acs.org/doi/book/10.1021/acsinfocus.7e5033

Preface

What We Cover

The key ambition of this book is to orient you in the areas of machine learning (ML) that we see as being very useful and likely to have great impact for a materials science researcher. ML, data science, or artificial intelligence, call it what you will, is a hugely diverse and fast-moving collection of subjects. Information on these subjects is abundant and almost necessarily highly diffuse—you can learn from books, from blogs, from videos, or from interpretive dance routines. The heterogeneous nature of information is a huge strength in this area but can also present a challenge.

Orientation in This Book

There is a wealth of great resources for learning many of the important basics for ML, not least in other titles in the *ACS In Focus* series, so we have chosen to minimize repetition. Where possible, we have pointed to other resources to provide information that is very important but not necessarily critical to understanding the contexts in which we describe the application of ML in materials science. For example, you may be surprised not to find a detailed introduction to many algorithms for supervised machine learning, such as random forests or multilayer perceptrons, but we do provide very clear signposts on where to look if you want to find these resources: we mention them in the text and also provide a *Read These Next* list at the end of each chapter.

We have attempted to primarily arrange this text around use cases, with the applications driving the introduction of new methods. We start with two chapters that concentrate on some fundamental methods and then have four chapters presenting use cases and example methods for ML applied to experiment and simulation. The first chapter deals with the early stages of building an ML solution for a materials science problem, concentrating on where and how to get data and some of the considerations when choosing an approach. CHAPTER 2 considers the ever important issue of how to build more robust models and how to make sure that your colleagues trust your results. CHAPTER 3 deals with using ML to accelerate or augment simulations. CHAPTER 4 introduces a number of ways in which ML can be applied to analyze and process experimental data. CHAPTER 5 looks at building integrated closed-loop experiments where ML is used to plan the course of a materials optimization experiment. CHAPTER 6 is focused on how ML can be used in the discovery of new materials on computers.

Another major consideration when we were writing this text was to try to ensure that for as many as possible examples that we provide we have also provided some information on how to *actually* implement the approaches. ML is an extraordinary field in that there is an abundance of high-quality, free, open-source tool kits available. We will give a little more overview of this in this book. It is our sincere hope that most readers of this book will end up putting into practice at least some of the methods that we present here.

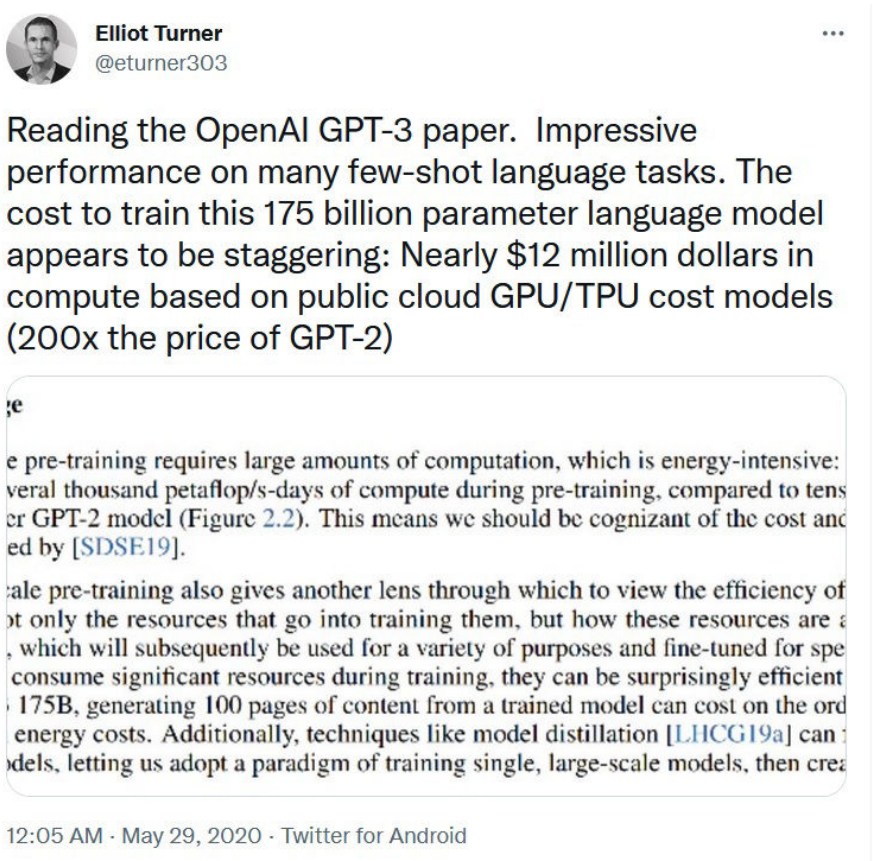
What You Need To Start Your ML Journey

We will briefly describe the hardware, software, and *"humanware"* that you need to start using ML for your research. Often, introductory texts do not deal with this and we feel that it is a rather large oversight. In our experience, practicing ML involves a significant proportion of our time making sure that we have right programs, sufficient compute resource, correct drivers, etc. to perform the task that we want to. In fact, combined with collecting and cleaning data, setting up of systems probably accounts for 70–80% of the work of a data scientist in materials science (and probably any field).

Hardware

There is a lot of optimistic talk about how ML will democratize research, and much of this is well founded, driven by the unprecedented availability and intellectual openness of much work in this area. However, it would be naïve to ignore the importance of compute resource in developing ML approaches. If we consider, for example, the GPT-3 (1) language model, it has 175 billion parameters and according to estimates based on commercially available graphical processing unit (GPU) resource cost around $12 million to train (FIGURE 0.1); clearly not accessible to an average nation-state, never mind research groups!

FIGURE 0.1 An Estimate of the Cost of Training GPT-3 Taken from Twitter.



https://twitter.com/eturner303/status/1266264358771757057

The kind of hardware required obviously depends on the task in hand, and luckily most models do not require the staggering levels of compute that GPT-3 requires. If we are interested in relatively small datasets and classical ML models (see SECTION 1.3 for the classical/deep distinction), then standard computer hardware with conventional central processing units (CPUs) are often sufficient. When we start to do **deep learning**, the situation changes, and often GPUs are necessary to efficiently use these methods. Deep learning algorithms run very efficiently on GPUs, and it is routine to get 5–10 × speed-ups just by running the same code on a GPU rather than a CPU.

It is not always necessary, however, to purchase and set up a local hardware stack for running ML training and inference. It is increasingly becoming standard practice to hire computational resources from cloud services. There are many vendors of cloud computing services available; the most notable examples are probably Google Cloud, Microsoft Azure, and AWS. These cloud services will often also provide some access to free GPU resource for starting projects. Projects such as Google Colaboratory make it possible to run electronic notebooks (see the Software section) on cloud compute resource, including some level of free access to GPUs (at the time of writing). Hopefully these resources mean that you will be able to run examples similar to the examples that we present throughout this book with a relatively low resource overhead, at least for getting started.

Software

Almost without doubt, if we are practicing ML these days, we will be working at least in part in the Python programming language. We do note that other languages are available: in particular, Julia is gaining significant popularity in the research community and promises the ease of use of Python with the performance of a compiled language such as C++. However, for the purposes of this book we concentrate primarily on Python.

One of the greatest strengths of Python is the software ecosystem that exists around it—for most scientific disciplines and subdisciplines, there are scores of readymade packages for tackling the problem. For almost any common computational task, it is probably unnecessary to rewrite our own code as the package to do it already exists. To take an example from the field of atomistic simulation for computational materials science, the

Atomic Simulation Environment is a suite of tools that performs many of the routine tasks of atomistic modeling automatically. Some studious internet searching would probably provide the same kind of package for almost any problem we are dealing with. However, this vast ecosystem of software does come at a cost: interoperability of packages is often challenging as they can sometimes rely on conflicting versions of other packages (for example, different releases of NUMPY). This situation is illustrated in a comic by the ever excellent XKCD.

The best way to try to manage packages and environments in Python is probably to use some combination of a package manager and virtual environments. Virtual environments allow us to create multiple different Python environments, so we can create a new environment and install a new package safe in the knowledge that we will not spoil an existing environment that was working perfectly well before. Package managers automatically download and install requested packages as well as dealing with any dependencies that they might have. The most popular package managers for Python are PIP and ANACONDA. ANACONDA provides both virtual environment and package manager functionality, and it also works with PIP, so we strongly suggest setting up ANACONDA on any machine where you intend to perform ML tasks. For using the code examples in this book, we have provided an ANACONDA environment file, which should allow us to build a virtual environment that will run all the codes we present.

Another resource that we would quickly like to mention is electronic notebooks, most notably Jupyter notebooks. Electronic notebooks allow for interactive running of code, with functionality for writing useful notes to accompany the code. Often for the exploratory stages of a project, before the code goes into full production, notebooks are very useful tools for tweaking code.

Finally, we strongly advocate for data and code sharing. Code that we write can easily be shared and made accessible on online repository services, such as GITLAB, BITBUCKET, or GITHUB. All of these resources provide a platform for sharing codes as well as an environment for collaborative code development. Often, researchers are bashful about sharing codes: **don't be**. When we share codes free, there is no obligation for it to be perfect and very often some codes are significantly better than none. If you become more serious about writing your own code and projects, we would also strongly advise you to learn about version control systems, most probably the GIT system, along with best coding practices and standards for your language of choice. Data sharing and open data are also critical for healthy ML practice; excellent initiatives include ZENODO, which offers services that host and share large datasets and also provide DOIs to ensure future "find-ability."

Humanware

The final piece of the jigsaw is you, the human in the loop. To optimize humanware, we strongly recommend reading and practice. We hope that the resources provided in this text serve as a good starting point, but there is no substitute for motivated self-discovery!

Keith T. Butler is a senior scientist at the Rutherford Appleton Laboratory, in the Scientific Machine Learning (SciML) team, where he leads projects that apply machine learning to the discovery and characterisation of materials. Keith obtained his bachelor's from Trinity College Dublin and his PhD from University College London. Before joining RAL, Keith spent time as a post-doctoral researcher in the groups of Aron Walsh (University of Bath/Imperial College London) and John Harding (University of Sheeld), and was a visiting researcher in The University of Toronto and Tokyo Institute of Technology. Keith's research focuses on using machine learning to accelerate the characterisation of materials and also to predict new, previously undiscovered materials for renewable energy applications, such as photovoltaics and photocatalysts. Keith is an active developer of several open source materials design packages (SMACT, SuperResTomo, Macro-Density) and a strong advocate of open science.

Felipe Oviedo is an applied researcher at Microsoft AI for Good, focusing on scientific machine learning for sustainability and healthcare applications. Prior to joining Microsoft, Felipe completed a PhD at the intersection of material science and computer science at MIT under the guidance of Prof. Tonio Buonassisi (MIT Mechanical Engineering) and Dr. John Fisher (MIT CSAIL). His dissertation was focused on accelerated development of photovoltaics by physics-informed machine learning. Felipe developed and deployed machine learning algorithms to accelerate the experimental screening and optimization of renewable energy materials and technologies. Before MIT, Felipe briefly worked in the energy industry and CERN.

Pieremanuele Canepa is an Assistant Professor in the Department of Materials Science and Engineering at the National University of Singapore (NUS). He received his bachelor's and master's degrees in Chemistry from the University of Torino (Italy) and a PhD from the University of Kent (UK). Prior to NUS, he was a Postdoctoral fellow at the Lawrence Berkeley National Laboratory and the Massachusetts Institute of Technology under the guidance of Prof. Gerbrand Ceder. His research contributes to the rational design of materials for clean energy technologies, including electrode materials for batteries, and electrolytes for sustainable energy storage devices. In 2021, Pieremanuele was elected as fellow of the Royal Society of Chemistry.

https://pubs.acs.org/doi/book/10.1021/acsinfocus.7e5033

Brief Table of Contents

Detailed Table of Contents

https://pubs.acs.org/doi/book/10.1021/acsinfocus.7e5033

CHAPTER 1

Applying Machine Learning (ML) to Materials Science

1.1 CHAPTER OVERVIEW

We will begin this book by giving a relatively high-level overview of the process involved in building a successful machine learning (ML) workflow for a problem of interest, which is depicted in FIGURE 1.1. We will concentrate here primarily on ML for materials science, but the process and principles outlined are general. Pedro Domingos, in his highly readable introductory article "A Few Useful Things To Know about Machine Learning" characterized ML as the combination of representation, evaluation, and optimization; for the sake of this book, we are also going to add a zeroth step to the process: data. In this chapter, we will cover data, representation, and evaluation. Optimization is discussed in other sections in the book (SECTIONS 5.3, 5.6, and 6.4).

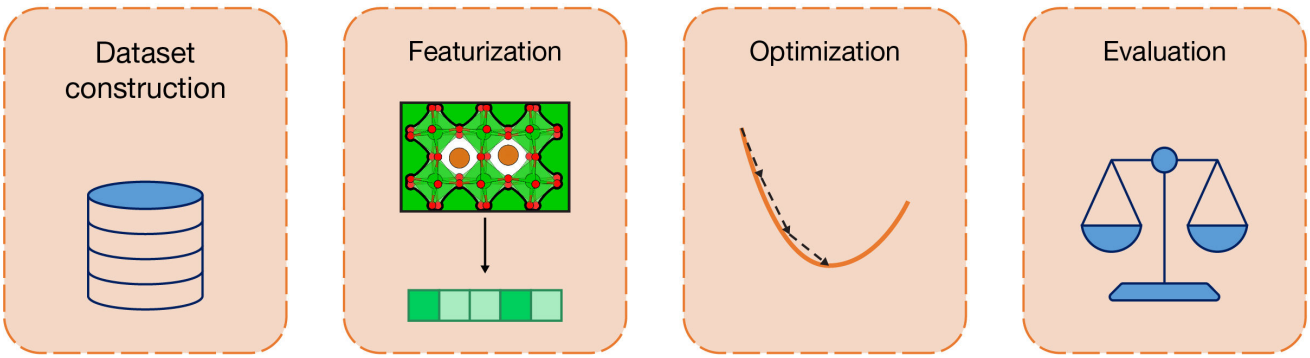
During the course of this chapter, we will introduce many useful ML concepts and techniques, including:

- databases;
- APIs;
- data visualization;
- representations and representation learning;
- **distance metrics**; and
- evaluation.

1.2 DATA AND DATABASES

Since the advent of artificial intelligence, the importance of data has become crucial to the development of any ML applications. In ML, a training data set is used to train the ML algorithm, which will be covered in most of the subsequent chapters in this book. To a large extent, the nature of data is the critical factor in deciding the choice and ultimate success or failure of an ML approach. It is remarkable that most deep learning methods were well established many decades before the recent explosion of deep learning applications, because the availability of large, high-quality, labeled data sets (such as **IMAGENET**) is a key enabling factor for these methods. As the field progresses, a strong underpinning for any researcher to understand is that training data requires some human intervention to analyze or process the data in a suitable format for its use in the ML model.

FIGURE 1.1 The Process of Building an ML Model.



First, the data are collected from various sources (experimental, simulated, etc.). Then the information is featurized, which involves choosing model types and converting materials into machine-readable formats. Then the model parameters are optimized so that they best fit the training data. Finally, the models are evaluated against independent test data sets. The various steps in this process are covered in this book.

1.2.1 The Five Vs of Data Science

To harness data's full capabilities in ML, a researcher must first consider the data's important attributes. For example, the quality of the data (veracity) and the quantity of data (the data volume) are fundamental attributes of data to develop accurate ML models. Any newcomers in data analytics and ML should familiarize themselves with the concepts of volume of data, velocity, veracity, validity, and volatility, which often are summarized as the 5 Vs of data science. The following paragraphs briefly explain each of the 5 Vs.

1. **Volume:** ML applications are enabled by the availability of large quantities of data to our disposal. In materials science, data can be acquired during an experiment or generated via simulations.

2. **Velocity:** in ML, velocity defines the speed at which data flows into a specific a database from various sources.

3. **Veracity:** in data science, the veracity of the data represents the degree of accuracy or truthfulness of a data set. Thus, veracity defines the quality of specific data sets, and generally data.

4. **Validity:** the concept of validity relates directly to veracity and is intended to define data that is accurate and appropriate for a specific application in data science.

5. **Volatility:** this refers to a data set's lifetime and its rate of change. Thus, the volatility of data in a database is whether a set of entries are relevant after a specific period of time.

1.2.2 Databases

A database is an organized collection of structured information, or data, typically stored electronically in a computer system (2). Many database types have been proposed over the years. In the last few decades, new types have been developed to address specific requirements. The simplest way to manage data on a computer outside of software is to store it in a basic file format.

To date, the most common database types are:

- **Relational databases.** Data are stored in multiple, related tables. Rows and columns of the tables store the data. The most common language to query relational databases is the structured query language (SQL). Common relational databases are: Oracle Database, MySQL, Microsoft SQL Server, and PostgreSQL.

- **Document-oriented and NoSQL databases.** Unlike in relational databases, data in a NoSQL database does not have to conform to a predefined **schema**, such as tables. Data is typically organized in **JSON**-like documents to model data instead of rows and columns

as in a relational database. Therefore, nonrelational databases are ideal to store and manage unstructured or semistructured data. One advantage of NoSQL databases is that developers can make changes to the database on the fly, without affecting applications that are using the database. Because of the nature of materials science problems, nonrelational database is commonly used to organize materials data. Common Nonrelational databases are: MongoDB, ApacheCassandra, CouchBase, and CouchDB.

- **Other database types.** These include cloud databases, columnar databases, key-value databases, object-oriented databases, wide column databases, and time series databases. These database types appear less popular in chemistry and materials science applications.

1.2.3 Interacting with Material Databases and Application Programming Interfaces

In materials science, data can be generated directly from a set of experimental measurements or from a large pool of simulations and archived in databases. Therefore, a central aspect to managing data and their attributes (the 5Vs) is the development, accessibility, and maintenance of databases.

Because a database is a dynamic organized collection of structured information, typically stored electronically in a computer, the typical operations performed by a user on a database are the addition of data, update of existing data, and query (retrieve) of data contained in a database. Indeed, precise languages to carry out these operations are specific to each specific type and the databases' architectures, which are not the object of this chapter.

Historically, several databases archiving data from specific experiments have been established. Indeed, decades ago scientists recognized databases' importance, producing archives of meticulously checked structural data from X-ray and neutron experiments of diffraction of molecules and materials. These are the Cambridge Structure Database (CSD), and its analogue database for inorganic structures and materials—the Inorganic Chemical Structure Database (3, 4). The popularization of density functional theory (DFT) provides a practical way to solve the complex many body Schrödinger equation molecules and extended inorganic materials has enabled the development of holistic high-throughput studies (5), whose data are then archived and redistributed through databases.

TABLE 1.1 shows some of the most popular databases containing thousands of material data entries obtained from computational data. Typically, databases summarized in TABLE 1.1 are not just the mere representation of data in a format of tables organized in collections of data, but they provide advanced interfaces that enable extremely sophisticated queries.

TABLE 1.1 Materials Databases Consolidated from Computed Data.[i]

Entries	Workflow	API	Provenance	References
Materials project				
711,561	Fireworks Automate Pymatgen Custodian	Yes	U.S.	(6, 7, 8, 9)
The Open Quantum Materials Database (OQMD)				
1,022,603	OQMD	Yes	U.S.	(10)
Automatic FLOW for Materials Discovery (AFLOW)				
3,482,348	AFLOW	Yes	U.S.	(11, 12)
The JARVIS Database				
65,286	JARVIS	Yes	U.S.	(13)
Materials Cloud				
7,502,686	AiiDA	Yes	Switzerland/Europe	(14, 15)
Novel Materials Discovery				
>11,000,000	Qmpy	Yes	Germany/Europe	(16)

Entries	Workflow	API	Provenance	References
BIG-MAP App Store				
N.A.	AiiDA SimStack	Yes	Denmark/Europe	(17)

The availability of vast amounts of data has fueled the revolution of data-driven materials science, which has enabled opportunities for in silico design of materials, mostly boosted by databases that can be queried by humans and machines via an application programming interface (API). Indeed, APIs have been described as "the currency in the digital world" (18), which are valuable to retailers for guaranteeing speed and safety in the information exchange. TABLE 1.1 informs whether materials databases provided dedicated APIs to query database properties. FIGURE 1.2 shows the Representational State Transfer (REST) that indicates the operation of a client (the user terminal) requesting information (data) about resources from a database. The database machine then transfers the current state of the resource back to the client machine.

FIGURE 1.2 Diagram of REST API Operation.

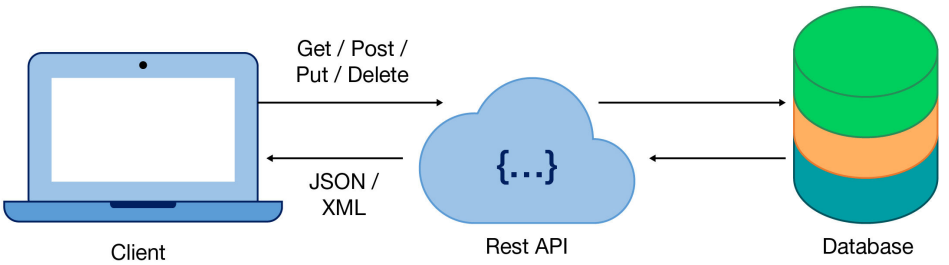


Figure adapted from https://www.seobility.net/ CC BY 4.0

In FIGURE 1.2, the client computer is the user's computer from where the request of data is initiated from a database server (a remote computer). Notably, all the communication occurs over the REST APIs. The API can deal with several basic operations:

- **GET**: Enable the client gets the data from the remote server.
- **POST**: Enable the client to write data to the remote server.
- **PUT**: Enable the client to update existing data on the remote server.
- **DELETE**: Enable the client to delete existing data on the server.

APIs can be coded using any programming language, such as C, C++, Java, and most recently Python, etc. For example, the REST API used to interface the database of Materials Project is based on Python (8).

If one is interested in retrieving data from a database, that implies performing a GET operation associated with a database query using the REST API from the client machine. As indicated in FIGURE 1.2, the result of this query is typically stored in a JavaScript Object Notation (JSON) or extensible **markup language** (XML) file. Indeed, the materials databases of TABLE 1.1 return JSON files upon queried. In the following, we provide a practical example of how to make a simple query to Materials Project and understands its results.

LISTING 1 shows a query done to Materials Project database using their Python RESTful API (8). Lines 9 and 10 in LISTING 1 defines the query according to the specification of the MaterialsProject RESTful API. In the specific example, the query is to GET all the materials' entry with a specific chemical formula, but the RESTful API enables more complex queries. The results of the query are then printed in lines 12–14. Specifically in line 12 of LISTING 1 prints the number of entries returned by the query. On line No. 15 of LISTING 1 the content of element [0] of the data object generated as a result of the query is printed.

LISTING 1.1 Python Snippet To Query the Materials Project Database via Their REST API.

```
1
2 from pymatgen.ext.matproj import MPRester
3 from pymatgen.core import Composition
4 import pprint
5
6 key_mp="xxxxxxxxxxx"
7 mprObj = MPRester(api_key=key_mp)
8 comp = Composition("Mn2O3")
9 dataQuery = mprObj.query({'pretty_formula': "Mn2O3"},
10              properties=["task_id", "pretty_formula", "
      structure"])
11
12 print("No. of queried compounds", len(dataQuery))
13 print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
14 print("Printing first compound of array")
15 pprint.pprint(dataQuery[0])
```

Notably, the attribute "properties = ["task_id", "pretty_formula", "structure"]" instructs the API to return the task_id of each material matching the query, the chemical minimal formula to represent the compound queried, and its structure. More options are clearly available, and increase as the database evolves.

In the current example, the first part of the query provides the number of entries available in the MaterialsProject database matching the chemical formula Mn_2O_3, as shown in LISTING 2.

LISTING 1.2 Number of Entries Matching the Chemical Formula Mn_2O_3 in LISTING 1.

```
1 No. of queried compounds 11
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The second part of the query of LISTING 2 returns the features of the first item of the query in the JSON format.

LISTING 1.3 Result of Query as in LISTING 1.

```
1 Printing first compound of array
2 {'pretty_formula': 'Mn2O3',
3  'structure': Structure Summary
4 Lattice
5     abc : 5.340933787647999 5.340933787647999 15.114359
6  angles : 90.0 90.0 146.28867925177863
```

```
volume : 239.28969097741458
      A : 1.548666 -5.111478 0.0
      B : 1.548666 5.111478 0.0
      C : 0.0 0.0 15.114359
PeriodicSite: Mn (1.5487, -1.3996, 0.7151) [0.6369, 0.3631, 0.0473]
PeriodicSite: Mn (1.5487, 1.3996, 14.3993) [0.3631, 0.6369, 0.9527]
PeriodicSite: Mn (1.5487, -1.3996, 6.8421) [0.6369, 0.3631, 0.4527]
PeriodicSite: Mn (1.5487, 1.3996, 8.2723) [0.3631, 0.6369, 0.5473]
PeriodicSite: Mn (1.5487, 4.1715, 9.6654) [0.0919, 0.9081, 0.6395]
PeriodicSite: Mn (1.5487, -4.1715, 5.4489) [0.9081, 0.0919, 0.3605]
PeriodicSite: Mn (1.5487, -4.1715, 2.1083) [0.9081, 0.0919, 0.1395]
PeriodicSite: Mn (1.5487, 4.1715, 13.0061) [0.0919, 0.9081, 0.8605]
PeriodicSite: O (0.0000, 0.0000, 7.5572) [0.0000, 0.0000, 0.5000]
PeriodicSite: O (0.0000, 0.0000, 0.0000) [0.0000, 0.0000, 0.0000]
PeriodicSite: O (1.5487, 0.2606, 12.8557) [0.4745, 0.5255, 0.8506]
PeriodicSite: O (1.5487, -0.2606, 2.2587) [0.5255, 0.4745, 0.1494]
PeriodicSite: O (1.5487, -0.2606, 5.2985) [0.5255, 0.4745, 0.3506]
PeriodicSite: O (1.5487, 0.2606, 9.8159) [0.4745, 0.5255, 0.6494]
PeriodicSite: O (1.5487, -2.2564, 13.9765) [0.7207, 0.2793, 0.9247]
PeriodicSite: O (1.5487, 2.2564, 1.1378) [0.2793, 0.7207, 0.0753]
PeriodicSite: O (1.5487, 2.2564, 6.4194) [0.2793, 0.7207, 0.4247]
PeriodicSite: O (1.5487, -2.2564, 8.6950) [0.7207, 0.2793, 0.5753]
PeriodicSite: O (1.5487, 3.3999, 11.3358) [0.1674, 0.8326, 0.7500]
PeriodicSite: O (1.5487, -3.3999, 3.7786) [0.8326, 0.1674, 0.2500],
'task_id': 'mvc-844'}
```

In particular, LISTING 3 contains:

- The chemical formula of the compound in the field "pretty formula."
- The structure of the chemical compound in the field "structure", defined by a number of attributes:
    - The length in Å of the lattice parameters, see the "abc" field.
    - The values of the angles separating each lattice parameter, see the "angles" field.
    - The volume of the unit cell, see the "volume" field.
    - The respective crystallographic vector, which are identified by "A", "B", and "C" fields.
    - The type of coordinates (periodic in this example), the chemical species, and the respectively coordinates in Cartesian and fractional units, respectively.
    - Finally, an identification code for the entry queried is in the "task_id" field.

The result of the query in LISTING 1 is highly specialized and conforms with the specifications of the REST API provided by MaterialsProject. The other databases shown in TABLE 1.1 utilize significantly different construct to produce the same query for the Mn_2O_3 chemical compounds. Furthermore, the output of the query may look slightly different, while the number of the item matching the compound formula Mn_2O_3 may be more or less.

As data stored in materials databases differ significantly in terms of scope and most importantly the 5 Vs exposed in SECTION 1.2.1, it appears extremely beneficial to unify data accessed from multiple sources. In particular, the retrieval of data from multiple databases is difficult as each material database provides its own specialized API that governs data access patterns, querying and the representation of the underlying data.

Furthermore, APIs have been developed to tailor the necessity of specific databases and their customers. To satisfy the user needs and the management of the database, significant effort is dedicated to translate the responses from the new API to the representation of the user.

To solve this issue, materials scientists have recently developed the OPTIMADE API (19), which provides a holistic standard for serving and accessing the information in compatible materials databases. According to Andersen et al. (19) the OPTIMADE API "strives to enable materials information to be filtered and retrieved in a straightforward and intuitive manner." The holistic OPTIMADE API is currently compatible with a broad range of materials databases, such as AFLOW, COD, TCOD, Materials Cloud, Materials Project, NOMAD, odbx, Open Materials Database (OMDB), and Open Materials Quantum Database (OQMD).

1.2.4 Common Materials Databases

When deciding whether or not we can build an ML model for a given problem, the first step is usually to consider what databases are available. In this section, we discuss some of the resources available for obtaining materials data; however, this discussion by no means covers all available databases and the number grows every month. Databases can be made up of experimental measurements or theoretical calculated properties (such as TABLE 1.1). Some databases cover broad ranges of materials, some are very specific to certain types (for example, porous materials), some cover generic properties like diffraction patterns or formation energies, and others are more application specific.

Generic computational materials databases: Possibly the largest growth in available materials data has been brought about by the advent of high-throughput computation in the past decade or so. There are many computational materials databases available now, and all have their own particular areas that they address. From TABLE 1.1, the Materials Project, AFLOW and OQMD databases have in common that they contain sets of materials properties calculated at a consistent level of theory, thereby allowing easy comparison internally. Because of the application history of when they were started, the Materials Project is skewed toward oxide and battery materials, while Aflow has more of an emphasis on intermetallic materials. Aflow has more entries than Materials Project, but for the majority of these only basic properties such as formation energy are reported (*i.e.*, no electronic structure information).

Databases such as the Materials Cloud and NOMAD resources have a somewhat different philosophy. In NOMAD the emphasis is on flexibility and this database contains inputs from many different types of calculations, so that the levels of theory or the code implementations could differ between entries. As a result, NOMAD is a much larger resource, but this does mean that some level of preprocessing is required before using the data for building an ML model for example the band gap of a material can be very sensitive to the level of theory used for calculation and artifacts of this sensitivity could easily pollute an ML mode if we do not first curate the data carefully. The Materials Cloud integrates with the AiiDA environment, providing a platform to develop high-throughput DFT workflows and also to construct and connect individual databases.

Generic experimental databases: The field of crystallography was very forward-thinking in terms of saving and curating data; as a result, the largest experimental databases contain primarily crystallographic data. The two main databases are the Cambridge Structural Database (CSD) (20), which contains over 1,000,000 entries, focused on small organic molecule and organometallic crystal structures, and the Inorganic Crystal Structure Database (ICSD) (21), which contains more than 240,000 inorganic crystal structures. Both of these databases are available on a subscription basis, an open version the Crystallography Open Database (COD) (22) is also available and contains more than 400,000 crystal structures of organic, inorganic, and organometallic materials.

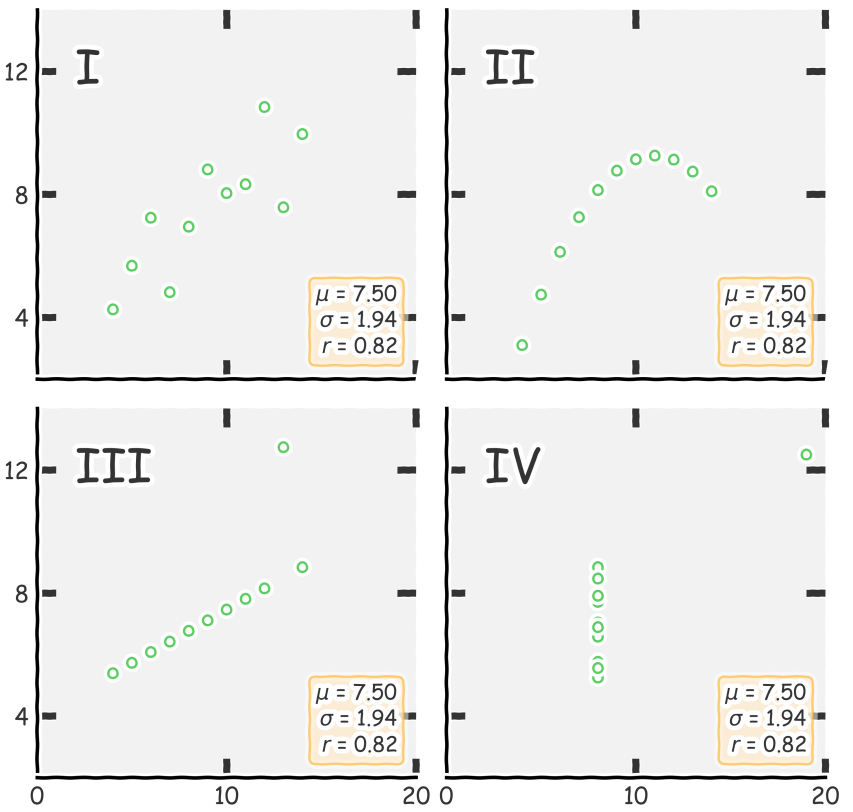
More specific databases: Databases particular to certain applications are beginning to appear more commonly. It is impossible to list all of the currently available resources. Some examples from the area of energy materials include the database of 2D metal halide perovskites (23) for solar energy applications, a recently published database of calculated thermoelectric materials properties (24), and the Catalysis-Hub.org database of calculated properties of catalytic materials (25).

1.2.5 Understanding Data Distributions and Relations

Before constructing an ML algorithm, it is usually advisable to get an understanding of how the data are distributed and how different aspects of the data relate to one another. This can be achieved in a number of different ways, looking at diagnostic metrics, looking at pair correlations, and visualizing the data. As a first step simply looking at metrics such as the mean and the **standard deviation** of parameters in the data can be very useful for telling us how much they vary, this will also be very important for normalizing the data (discussed below). We can also calculate correlations between factors in the data. Correlation metrics such as r^2 and Pearson correlation can tell us how closely factors are related. So, for example, if a single input parameter is highly correlated with the output that we are trying to predict, then it could be possible to construct a model based on just that parameter. Likewise, we might find that two representation parameters are closely related, for example, atomic number and atomic weight, in this case it could be redundant to use both parameters and we might consider dropping one of them.

It is important and often recommended, before passing data to an ML model to make sure that arbitrary differences in the data scales do not influence the model. For example, the melting point of a material will usually be measured on a scale of tens or hundreds, but the magnetic moments will typically be of the order of 1, this does not mean that in a given context the temperature is a thousand times more important than the magnetic moment. To ensure that the model is not presented with specious differences in scale it is important to normalize data before training. Normalizing can be achieved in many ways, such as standard normalization (dividing by the mean), min–max normalization (setting the minimum and maximum values and re-scaling the rest of the data) and many more—data normalization can be efficiently achieved using the preprocessing tools in `scikit-learn`.

FIGURE 1.3 The Famous Anscombe's Quartet.



All four data sets have the same mean and standard deviation, but look entirely different.

It is always important to visualize your data before doing any ML. Some time spent looking at the data can reveal many important characteristics and relationships. How the data is visualized will depend on the kind of data, time series, images, spectra, etc. The canonical example quoted to emphasize the importance of visualization as well as descriptive statistics is Anscombe's quartet, see FIGURE 1.3. Here the four distributions have identical diagnostic statistics, mean, standard deviation, r^2 and so on, but visualization of the data reveals that they are very different.

1.3 REPRESENTATIONS AND REPRESENTATION LEARNING

Representation in ML is literally how we present the problem that we wish to solve to the computer. This is actually a multifaceted issue and involves both how we set up the model and how we pass the training data to the model. For example, say we want to classify a set of materials as either metals or insulators, we would first assess what kind of model would be appropriate for solving this problem. To a large extent, the models that we choose are determined by a combination of the problem and data available. At this stage, we will introduce two different ways that we classify models throughout this book. **Representations** of materials systems will be discussed in more detail in SECTION 3.2.

First, we distinguish between supervised and unsupervised learning (there are many intermediate flavors such as self-supervised learning and semisupervised learning, but we will stick to the two main classes in this text). Which class of models we choose is determined by the data we have; if the data have labels, then we might want to do supervised learning, if the data are not labeled we can only do unsupervised learning. In supervised learning we are trying to learn the relationship between the input data and the labels, in the example of metal/insulator classification we might have a set of materials and labels of whether they are metals or insulators and we will train a model to learn this. Examples of supervised learning are presented in SECTIONS 4.2.3, 4.3.1, 4.4.1, 4.4.2, and 5.3.1.

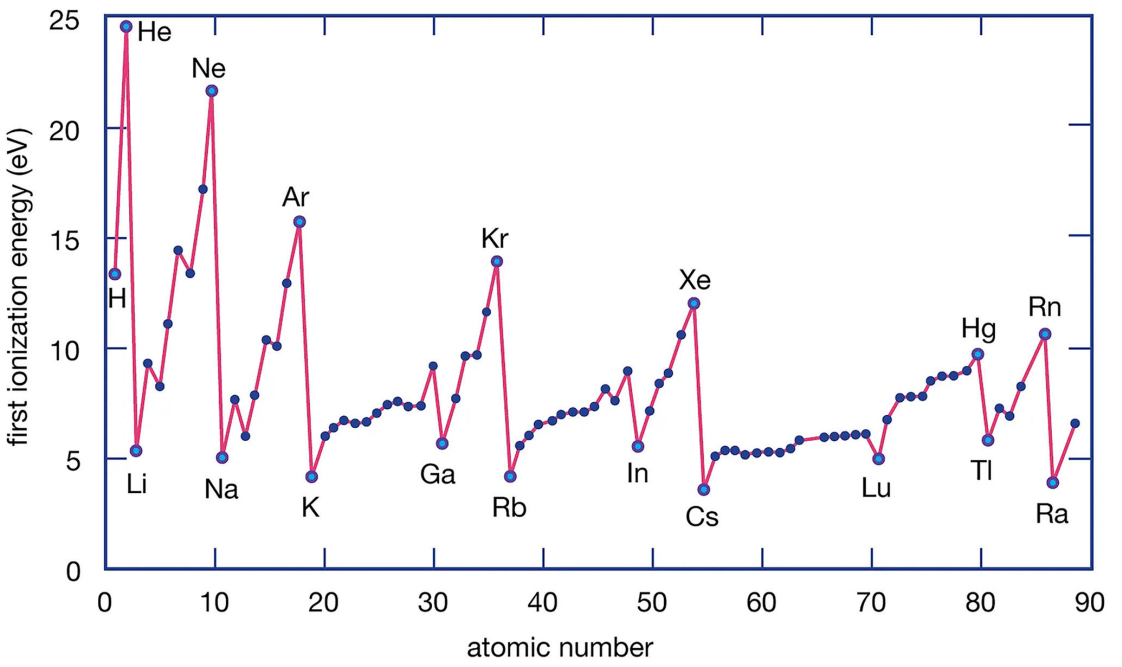
If on the other hand we have a set of materials and we know they contain metals and insulators but we have no labels, then we can only do unsupervised learning. In unsupervised learning, we are attempting to learn something about the structure of the data. To reuse the metal/

insulator example, it might be possible to take out input data and manipulate it in such a way that it forms two distinct sets; this can be performed by casting the data in an appropriate space, often using methods such as dimensionality reduction and then searching for structure in that space, often using methods such as clustering. Examples of unsupervised learning are presented in SECTIONS 4.2.2, 5.5, 2.4.2, 2.4.2.2, and 2.4.3.

Another distinction that we will make is between "classical" ML and deep learning (DL). Roughly speaking, we call any methods that are based on neural network structures as DL methods—technically to be "deep" a neural network should have more than one **hidden layer**, but for the sake of simplicity, we will call all neural network methods DL. Classical ML encompasses all other ML methods, such as decision trees, support vector machines, linear regression, naïve Bayes, Gaussian processes and so on. To understand where and when we might use DL or classical ML it is useful to understand the representation of the data.

Representations also called features are the format in which data are presented to an ML model. Building representations is arguably the most important part of many ML models. The way we represent our input data and what data we choose to represent offer a chance to impart our **domain knowledge** to the data before the ML algorithm has to learn anything. The Periodic Table is a fine example of featurization before the age of ML. In the Periodic Table, the atoms are assigned attributes, such as numbers, groups, rows, and columns, all of which reflect knowledge about the elements and how they relate to one another. Imagine trying to derive trends in ionization energies of the atoms if all we had to go on was the alphabetical order; now observe the order on this data set that is brought by arranging the values according to the Periodic Table (FIGURE 1.4). We can now clearly see that there is a trend in the data, and it could even be possible to develop a predictive model for points that we may not have yet observed. Hopefully, this simple example makes clear that the success or failure of many ML methods can rest on how we featurize the data.

FIGURE 1.4 Ionization Energies.



By featurizing the chemical species by atomic number clear trends in the ionization energy become apparent. Britannica, The Editors of Encyclopaedia. Encyclopedia Britannica, 2020. https://www.britannica.com/

An important exception to the discussion above about the importance of featurizing is DL. In DL approaches, the models are generally presented with largely unstructured data, for example images. A useful example might be if we wanted to use a spectrum to calculate the amount of a compound present in a mixture. If we were using a classical ML approach, we would convert the spectrum into a series of hand-crafted features, for example, we might give the position of the first peak, the ratio of the height of the first and second peak, the distance between the third and fourth peak, and so on. If we were using a DL approach, on the other hand, we could simply present an image of the spectrum, or a 1D vector of the intensities. The DL model will then learn representations of the spectra that are useful for the task we wish to achieve, how the model learns these representations is part of the training process. An example of classification learning is the convolutional neural network discussed in more detail in SECTION 2.2.

1.3.1 Distance Metrics

An important concept, when we have vector representations of data is measuring the distance between data points in this high-dimensional space. Distances are important, for example when we want to assign data to clusters and inherently points that are closer should correspond to data that is similar. However, measuring distances in multidimensional space can be tricky, here we cover some of the commonly applied distance measures and describe their strengths and weaknesses.

Euclidean distance is the simplest distance metric; it deals with the distance between vectors x and y with i dimensions:

$$D(x, y) = \sqrt{\sum_i (x_i - y_i)^2}. \tag{1.1}$$

The **Euclidean distance** is simple to calculate and can be useful. However, one must be careful that all units of each dimension have been normalized, or it will skew the distance. Also the Euclidean distance tends to become less useful as we move into spaces with dimensionality much greater than three.

Cosine distance works by calculating the cosine of the angle between x and y:

$$D(x, y) = \frac{x \cdot y}{||x||||y||}. \tag{1.2}$$

The cosine similarity overcomes some of the difficulties encountered by the Euclidean distance in higher dimensions. A major drawback of cosine similarity is that it only considers directions of vectors and not their magnitude.

Manhattan distance describes the distance between two points if they could only move at right angles (just like on the gridiron pattern of Manhattan streets):

$$D(x, y) = \sqrt{\sum_i |x_i - y_i|}. \tag{1.3}$$

As we can see, Manhattan distance is rather similar to the Euclidean distance. However, the Manhattan distance does not suffer as badly as Euclidean distance in higher dimensions and so it may be a better choice in such cases. As with Euclidean distance, care should be taken to normalize all dimensions before calculating the Manhattan distance.

The Minkowski distance is a generalization of the Manhattan and Euclidean distances.

$$D(x, y) = (\sum_i |x_i - y_i|^p)^{1/p} \tag{1.4}$$

The Minkowski distance allows for a great deal of flexibility, and in principle one can tune p to obtain the value that works best in a particular space. However, it is also advisable to explore the simpler metrics first and develop a feeling for how the choice of p might affect the performance of the distance metric that you choose.

1.4 EVALUATION

Evaluation functions are also commonly referred to as loss functions or objective functions. Evaluation functions are used in two contexts during ML; first, there is a function that the algorithm attempts to optimize during the training process, and second, there is a function that we use after training to compare different models and select the best one. These can be the same function or they can be different functions, depending on the problem that we are interested in addressing, for example, some loss functions like precision might be very good for characterizing how well our model performs, but since it is only evaluated on a large set of data, we cannot use it to update the parameters of the model on an individual case by case basis, which makes precision not optimal as a training loss function. We will also discuss more advanced validation and evaluation in SECTION 2.4.

When calculating the different loss functions for training and choosing between models, it is critical that we use different data for each. A model can often train very well and optimize the training loss function, but when presented with new data outside the training set it performs

very poorly, in this case the model is said to be overfitting to the training data. Generally, we can split data into training and validation sets, the training set is used to optimize the parameters of the model and the validation set is then used to compare between different models. Of course, keeping out a validation set reduces the amount of data that we have for training. One common way to overcome this and to have the best of both worlds is to use cross-validation training and validation. In **cross validation**, we train on a portion and withhold a portion of the data for validation, then we repeat this process but withholding a different portion of the data. Cross validation is often described as *n*-fold cross validation, where *n* is the number of different training/validation splits; so, 5-fold cross validation uses 5 different 80:20 splits for training/validation.

In most modern ML algorithms in addition to the parameters being optimized there are **hyperparameters** that we choose when setting up the model. Hyperparameters can be things such as the number of layers in a neural network, the number of learners in a random forest or the number of coefficients in a polynomial expression, they are adjustable parameters, but they do not update during the training. In order to find the best model, however, we often perform manual or automated hyperparameter tuning. During hyperparameter tuning we change some of the hyperparameters in a systematic way and observe how this affects the validation loss metric. This should raise suspicion as it means that we are using the validation data to fit some of the parameters in our data, and this can lead to an insidious leak of validation data into model training. In order to avoid this the standard practice to withhold another set of data, the test set, the test set is different from the training and validation sets and is only ever used to evaluate final model performance, never to make decisions about how to vary the parameters or the hyperparameters.

1.4.1 Evaluation Metrics

The evaluation metrics that we use are completely determined by the type of problem and the final objective that we are dealing with. One important distinction that we can make is whether we are doing a regression or classification problem. In regression, we are attempting to predict an output that is a continuous value, for example, the yield of a chemical synthesis, in classification we are attempting to separate data points into different classes, for example crystal structures based on diffraction patterns.

For regression-type problems, two of the most common loss functions are the mean absolute error (MAE) and the mean squared error (MSE):

$$\mathcal{L}_{\mathcal{MAE}} = \frac{1}{N}\sum |y - y'| \tag{1.5}$$

$$\mathcal{L}_{\mathcal{MSE}} = \frac{1}{N}\sum (y - y')^2 \tag{1.6}$$

where y and y' are the true and the predicted labels of the data.

The MSE has the advantage of being continuously differentiable and decreasing gradients toward the optimal value, which means that it is well behaved when optimizing close to the optimum. However, the MSE is quite susceptible to outliers in the data, because the loss increases with the square of the difference, a single point with a large difference can contribute disproportionately to the loss and the model may try too hard to minimize a spurious difference. The MAE is less affected by outliers; however, the functional form of the MAE means that optimization is less well behaved in the region close to the optimum value and the optimization can often take more steps to converge than the MSE. A regression loss that combines the best of MAE and MSE is the Huber loss, which behaves like MSE if the loss is small and like MAE if the loss is large:

$$\mathcal{L}_{\mathcal{H}\sqcap[]\nabla} = \begin{cases} \frac{1}{2}(y - y')^2 & if\ |(y - y')| < \delta \\ \delta((y - y') - \frac{1}{2}\delta) & otherwise \end{cases} \tag{1.7}$$

When we are interested in classification, there are other loss functions and evaluation metrics that we can use. Two of the most common loss functions used for training classifiers are the cross entropy and the hinge loss. Cross entropy compares the log of the true label with prediction:

$$\mathcal{L}_{\mathcal{BXE}} - (y\log(y') + (1 - y)\log(1 - y')) \tag{1.8}$$

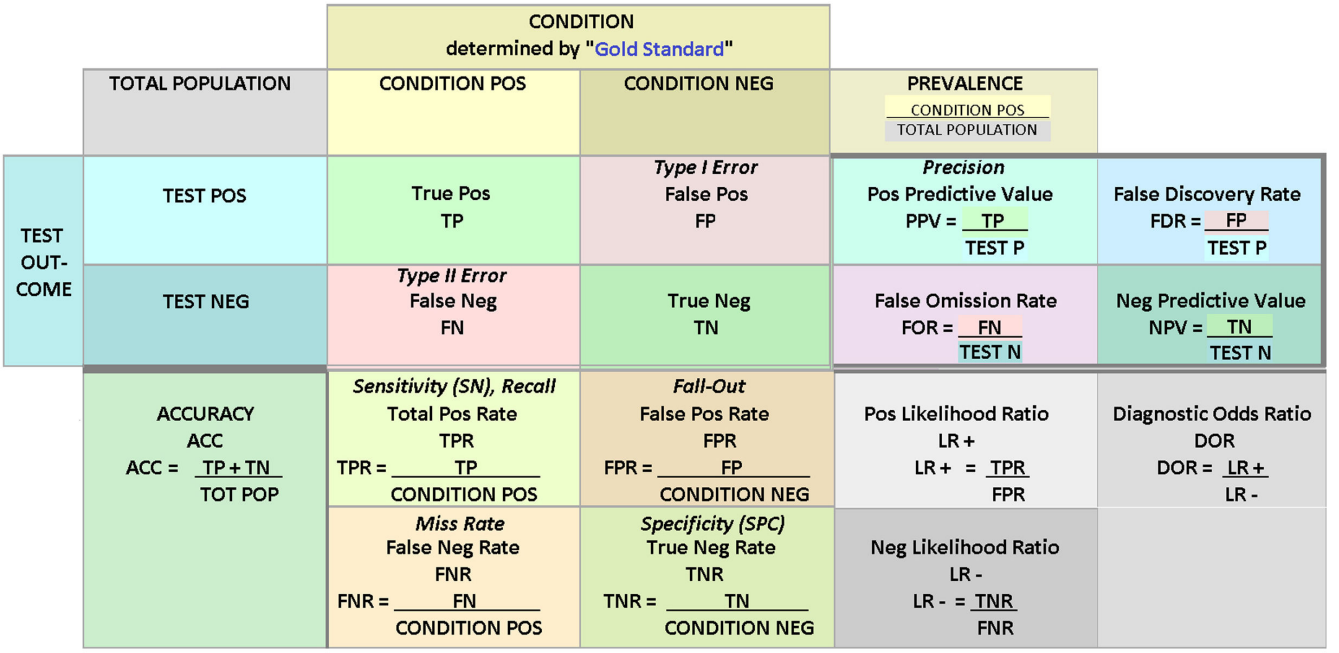
In this case we are looking at binary cross entropy as the equation assumes labels of 1 or 0; but this can readily be generalized to multiple labels. Note that if the true label is 1 the second term is zero and if the true label is zero the first term is zero. Binary cross entropy penalizes particularly strongly when the difference in labels is large, i.e., when the model is wrong and confident.

The hinge loss is not concerned with the actual values of the prediction, but with the classification that they correspond to. For example, in a binary classification, we may say that all values above 0.5 correspond to class 1; in binary cross entropy, there is still a loss even if the model predicts 0.95 for a point with label 1, but in hinge loss, the loss in this case is zero, because the classification is correct.

$$\mathcal{L}_{\mathcal{H}\backslash\backslash\}]} = max(0, 1 - y \cdot y') \tag{1.9}$$

For classification problems there are also a whole series of metrics that can be used to evaluate the classifier but that are not used in the training process. The confusion matrix is summarized in FIGURE 1.5. We now consider some of the most important metrics derived from the confusion matrix.

FIGURE 1.5 The Confusion Matrix.



Evaluation metrics that can be used to evaluate classification models. By Lavender888000 at https://commons.wikimedia.org/ CC BY-SA 4.0

Accuracy is calculated as $(TP + TN)/(TP + FP + FN + TN)$; it is a useful metric when the data we are dealing with is well balanced. However, one should be wary of using accuracy if the data is imbalanced—for example if we are trying to predict if a compound has critical temperature for superconductivity of greater than 200 K—obviously a classifier that always says no will be correct probably more than 99% of the time.

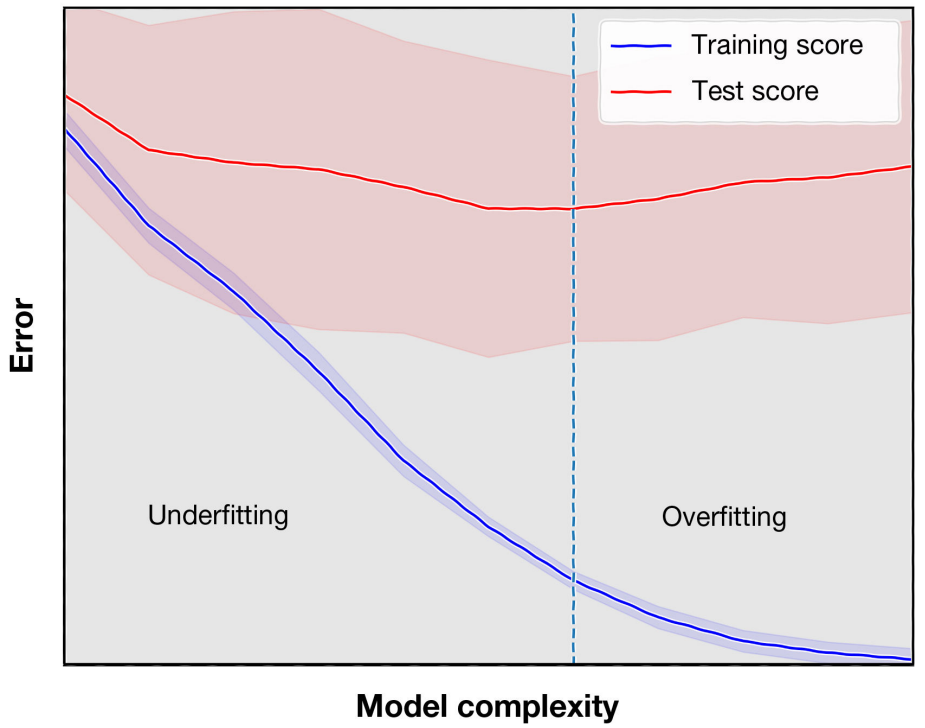
Precision is calculated as $(TP)/(TP + FP)$; this avoids the problem encountered for accuracy in the high-T superconductor classification above. If the model always says no then the precision for this prediction will be zero and hence we know that the model is not performing well. Precision is a good metric if we want to be sure that our positives are correct. For example, if a positive classification means that we will perform an expensive synthesis of an identified compound, then we want to be sure that the classification is correct. On the other hand, choosing models for optimal precision means that we will have a model that might miss some potential true positive classifications.

Recall is calculated as $(TP)/(TP + FN)$; it is similar to precision, in that it avoids the issue of accuracy for the imbalanced data set. However, a model with good recall is a model that is best for always identifying the positive cases. So, for example if we have a set of candidate materials and we want to definitely identify all of those that might display a particular characteristic, then choosing maximum recall is a good choice, but we must expect that the model might give us some more false positives than if we choose based on precision.

1.4.2 Protecting against Overfitting

We conclude the section on evaluation by looking at how we can prevent models from overfitting. Overfitting occurs because a model tried too hard to fit all of the observations in the training data and becomes needlessly complex. In FIGURE 1.6, it is clear that at a certain level of complexity the validation loss of the model starts to deteriorate, at this point it is overfitting. This highlights one of the best ways to protect against overfitting, that is, to validate the model properly and monitor validation loss.

FIGURE 1.6 Identifying Overfitting by Looking at Training and Validation Loss.



The training loss continually increases with increased model complexity, but the test loss at some point ceases to improve; after this point, the model is overfitting.

Models can also be protected against overfitting at the training stage. One of the most popular methods of avoiding overfitting is to use regularization terms in the loss function, the most common regularization terms are $L1$ and $L2$ regularization.

$$\mathcal{L} = \mathcal{L}' + \lambda \sum_{1}^{n} |w_i| \tag{1.10}$$

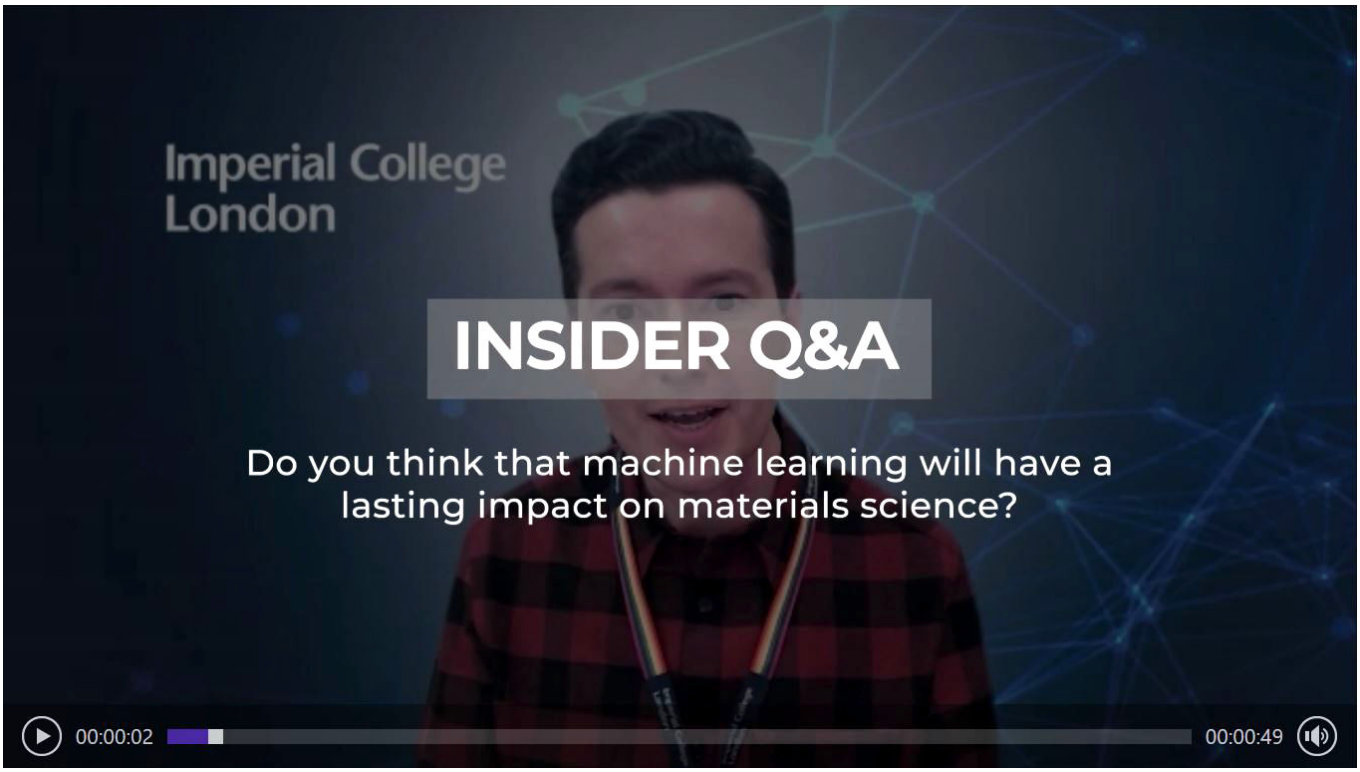
$$\mathcal{L} = \mathcal{L}' + \lambda \sum_{1}^{n} (w_i)^2 \tag{1.11}$$

where w are the weights of all the n parameters in the model and λ is a weighting term. Both $L1$ and $L2$ loss penalize in favor of models with lower weights on the parameters, $L1$ tends to optimize to weights of zero for unimportant parameters and $L2$ optimizes to small weights for unimportant parameters. Methods such as $L1$ and $L2$ are often useful in selecting parameters to represent data. By applying regularization, it is often possible to discard less meaningful features and to develop more compact and more generalizable representations.

For neural networks there is another form of regularization, known as dropout. In dropout, the network randomly sets the weights connecting a portion of the nodes to zero during each step of the training. At each epoch of training, a different set of weights is set to zero.

1.5 INSIDER Q&A: ARON WALSH

Do you think that machine learning will have a lasting impact on materials science?



0:51 | WATCH | **ANSWER** Professor Aron Walsh, Imperial College London

Read the Transcript:

I would say that machine learning is already becoming a core research tool. And, I don't think it's going to be going away anytime soon. It's giving us efficient ways to process and analyze data sets to sort of training more predictive models if we want to sort of extrapolate to new systems. So, already, I think machine learning is starting to be integrated into undergraduate courses that will really ensure that the next generation of chemists will be fluent with the terminology and some of the methods. So, it won't just be for the research specialist. So, definitely, I think machine learning is here to stay.

1.6 THAT'S A WRAP

- Machine learning consists of data, representation building, model evaluation, and model optimization.
- There are many high-quality materials science databases emerging; often the most useful way to access them is via a computer code known as an API.
- Representation building involves choosing a model and processing data before passing to the model—it is often the most important step for successful machine learning.
- Deep learning models do not require extensive manipulation or featurization of data; they can learn representations, but they generally require much more data than classical ML methods.
- Evaluating models on independent data is critical for an unbiased assessment of performance or usefulness.

1.7 READ THESE NEXT

1. **For an introduction to different model types:**

   Janet, J. P.; Kulik, H. J. *Machine Learning in Chemistry*; American Chemical Society, **2020**, 10.1021/acs.infocus.7e4001

2. For a nice general overview of the elements of ML as well as discussion of important tacit knowledge:

Domingos, P. A Few Useful Things To Know About Machine Learning. *Commun. ACM* **2012**, 55 (10): 78–87, 10.1145/2347736.2347755

3. For a hands on guide to programming neural networks in Python:

Chollet, F. *Deep learning with Python*; Manning Publications, **2017**.

4. For an excellent hands on guide to data science:

VanderPlas, J. *Python Data Science Handbook*; O'Reilly, **2016**.

Bibliography

1. Brown, T., et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*; Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; Lin, H. Eds.; Curran Associates, Inc., **2020**; vol. 33, 1877–1901. https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.
2. Oracle. *Oracle database online documentation, 10g release 2 (10.2)*; **2022**. https://docs.oracle.com/cd/B19306_01/server.102/b14220/intro.htm.
3. Groom, C. R.; Bruno, I. J.; Lightfoot, M. P.; Ward, S. C. The cambridge structural database. *Acta Crystallogr., Sect. B: Struct. Sci., Cryst. Eng. Mater.* **2016**, *72*, 171–179, 10.1107/S2052520616003954.
4. Bergerhoff, G.; Hundt, R.; Sievers, R.; Brown, I. D. The inorganic crystal structure data base. *J. Chem. Inf. Comput. Sci.* **1983**, *23*, 66–69, 10.1021/ci00038a003.
5. Hautier, G.; Fischer, C. C.; Jain, A.; Mueller, T.; Ceder, G. Finding nature's missing ternary oxide compounds using machine learning and density functional theory. *Chem. Mater.* **2010**, *22*, 3762–3767, 10.1021/cm100795d.
6. Jain, A. et al. Commentary: The materials project: A materials genome approach to accelerating materials innovation. *APL Mater.* **2013**, *1*, 011002, 10.1063/1.4812323.
7. Ong, S. P. et al. Python materials genomics (pymatgen): A robust, open-source python library for materials analysis. *Comput. Mater. Sci.* **2013**, *68*, 314–319, 10.1016/j.commatsci.2012.10.028.
8. Ong, S. P. et al. The materials application programming interface (API): A simple, flexible and efficient API for materials data based on REpresentational state transfer (REST) principles. *Comput. Mater. Sci.* **2015**, *97*, 209–215, 10.1016/j.commatsci.2014.10.037.
9. Mathew, K. et al. Atomate: A high-level interface to generate, execute, and analyze computational materials science workflows. *Comput. Mater. Sci.* **2017**, *139*, 140–152, 10.1016/j.commatsci.2017.07.030.
10. Saal, J. E.; Kirklin, S.; Aykol, M.; Meredig, B.; Wolverton, C. Materials design and discovery with high-throughput density functional theory: The open quantum materials database (OQMD). *JOM* **2013**, *65*, 1501–1509, 10.1007/s11837-013-0755-4.
11. Taylor, R. H. et al. A RESTful API for exchanging materials data in the AFLOWLIB.org consortium. *Comput. Mater. Sci.* **2014**, *93*, 178–192, 10.1016/j.commatsci.2014.05.014.
12. Calderon, C. E. et al. The AFLOW standard for high-throughput materials science calculations. *Comput. Mater. Sci.* **2015**, *108*, 233–238, 10.1016/j.commatsci.2015.07.019.
13. Choudhary, K. et al. The joint automated repository for various integrated simulations (jarvis) for data-driven materials design. *npj Comput. Mater.* **2020**, *6*, 1–13, 10.1038/s41524-020-00440-1.
14. Pizzi, G.; Cepellotti, A.; Sabatini, R.; Marzari, N.; Kozinsky, B. AiiDA: automated interactive infrastructure and database for computational science. *Comput. Mater. Sci.* **2016**, *111*, 218–230, 10.1016/j.commatsci.2015.09.013.
15. Uhrin, M.; Huber, S. P.; Yu, J.; Marzari, N.; Pizzi, G. Workflows in AiiDA: Engineering a high-throughput, event-based engine for robust and modular computational workflows. *Comput. Mater. Sci.* **2021**, *187*, 110086, 10.1016/j.commatsci.2020.110086.
16. Draxl, C.; Scheffler, M. NOMAD: The FAIR concept for big data-driven materials science. *MRS Bull.* **2018**, *43*, 676–682, 10.1557/mrs.2018.208.
17. BIG-MAP; *BIG-MAP app store*; **2021**. https://big-map.github.io/big-map-registry/.
18. Goetsch, K. *APIs for modern commerce: enable rich customer experiences everywhere*; O'Reilly Media: Sebastopol, CA, **2017**.
19. Andersen, C. W. et al. OPTIMADE, an API for exchanging materials data. *Sci. Data* **2021**, *8*, 217, 10.1038/s41597-021-00974-z.
20. Groom, C. R.; Bruno, I. J.; Lightfoot, M. P.; Ward, S. C. The Cambridge structural database. *Acta Crystallogr., Sect. B: Struct. Sci., Cryst. Eng. Mater.* **2016**, *72*, 171–179, 10.1107/S2052520616003954.
21. Hellenbrandt, M. The inorganic crystal structure database (ICSD)—present and future. *Crystallogr. Rev.* **2004**, *10*, 17–22, 10.1080/08893110410001664882.
22. Gražulis, S. et al. Crystallography Open Database – an open-access collection of crystal structures. *J. Appl. Crystallogr.* **2009**, *42*, 726–729, 10.1107/S0021889809016690.
23. Marchenko, E. I. et al. Database of two-dimensional hybrid perovskite materials: open-access collection of crystal structures, band gaps, and atomic partial charges predicted by machine learning. *Chem. Mater.* **2020**, *32*, 7383–7388, 10.1021/acs.chemmater.0c02290.
24. Ricci, F. et al. An ab initio electronic transport database for inorganic materials. *Sci. Data* **2017**, *4*, 1–13, 10.1038/sdata.2017.85.
25. Winther, K. T. et al. Catalysis-Hub.org, an open electronic structure database for surface reactions. *Sci. Data* **2019**, *6*, 1–10, 10.1038/s41597-019-0081-y.
26. VanderPlas, J. *Python data science handbook: Essential tools for working with data*; O'Reilly Media, Inc., **2016**.

27. Goldstein, A.; Kapelner, A.; Bleich, J.; Pitkin, E. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *J. Comput. Graph. Stat.* **2015**, *24*, 44–65, 10.1080/10618600.2014.907095.

28. Davies, D. W.; Butler, K. T.; Walsh, A. Data-driven discovery of photoactive quaternary oxides using first-principles machine learning. *Chem. Mater.* **2019**, *31*, 7221–7230, 10.1021/acs.chemmater.9b01519.

29. Jha, D. et al. ElemNet: Deep learning the chemistry of materials from only elemental composition. *Sci. Rep.* **2018**, *8*, 1–13, 10.1038/s41598-018-35934-y.

30. Tshitoyan, V. et al. Unsupervised word embeddings capture latent knowledge from materials science literature. *Nature* **2019**, *571*, 95–98, 10.1038/s41586-019-1335-8.

31. Antunes, L. M., Grau-Crespo, R.; Butler, K. T. Distributed representations of atoms and materials for machine learning. *arXiv preprint arXiv:2107.14664*, **2021**.

32. Faber, F.; Lindmaa, A.; von Lilienfeld, O. A.; Armiento, R. Crystal structure representations for machine learning models of formation energies. *Int. J. Quantum Chem.* **2015**, 115, 1094–1101, 10.1002/qua.24917.

33. Himanen, L. et al. DScribe: Library of descriptors for machine learning in materials science. *Comput. Phys. Commun.* **2020**, *247*, 106949, 10.1016/j.cpc.2019.106949.

34. Vinyals, O.; Bengio, S.; Kudlur, M. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, **2015**.

35. Fung, V.; Zhang, J.; Juarez, E.; Sumpter, B. G. Benchmarking graph neural networks for materials chemistry. *npj Comput. Mater.* **2021**, *7*, 1–8, 10.1038/s41524-021-00554-0.

36. Miksch, A. M.; Morawietz, T.; Kästner, J.; Urban, A.; Artrith, N. Strategies for the construction of machine-learning potentials for accurate and efficient atomic-scale simulations. *Mach. Learn.: Sci. Technol.* **2021**, *2*, 031001, 10.1088/2632-2153/abfd96.

37. Deringer, V. L.; Caro, M. A.; Csányi, G. Machine learning interatomic potentials as emerging tools for materials science. *Adv. Mater.* **2019**, *31*, 1902765, 10.1002/adma.201902765.

38. Ward, L.; Agrawal, A.; Choudhary, A.; Wolverton, C. A general-purpose machine learning framework for predicting properties of inorganic materials. *npj Comput. Mater.* **2016**, *2*, 1–7, 10.1038/npjcompumats.2016.28.

39. Oviedo, F. et al. Fast and interpretable classification of small x-ray diffraction datasets using data augmentation and deep neural networks. *npj Comput. Mater.* **2019**, *5*, 1–9, 10.1038/s41524-019-0196-x.

40. Maffettone, P. M.; Daly, A. C.; Olds, D. Constrained non-negative matrix factorization enabling real-time insights of *in situ* and high-throughput experiments. *arXiv preprint arXiv:2104.00864*, **2021**.

41. Van Amersfoort, J.; Smith, L.; Teh, Y. W.; Gal, Y. Uncertainty estimation using a single deep deterministic neural network. In *Proceedings of the 37th International Conference on Machine Learning, vol. 119 of Proceedings of Machine Learning Research*; Daume, H., III, Singh, A., Eds.; PMLR, 2020; pp 9690-9700. https://proceedings.mlr.press/v119/van-amersfoort20a.html.

42. Butler, K. T.; Le, M. D.; Thiyagalingam, J.; Perring, T. G. Interpretable, calibrated neural networks for analysis and understanding of inelastic neutron scattering data. *J. Phys.: Condens. Matter* **2021**, *33*, 194006, 10.1088/1361-648X/abea1c.

43. Herbol, H. C.; Hu, W.; Frazier, P.; Clancy, P.; Poloczek, M. Efficient search of compositional space for hybrid organic–inorganic perovskites via bayesian optimization. *npj Comput. Mater.* **2018**, *4*, 1–7, 10.1038/s41524-018-0106-7.

44. Sun, S. et al. A data fusion approach to optimize compositional stability of halide perovskites. *Matter* **2021**, *4*, 1305–1322, 10.1016/j.matt.2021.01.008.

45. Lookman, T.; Balachandran, P. V.; Xue, D.; Yuan, R. Active learning in materials science with emphasis on adaptive sampling using uncertainties for targeted design. *npj Comput. Mater.* **2019**, *5*, 1–17, 10.1038/s41524-019-0153-8.

46. Hase, F.; Roch, L. M.; Kreisbeck, C.; Aspuru-Guzik, A. Phoenics: a Bayesian optimizer for chemistry. *ACS Cent. Sci.* **2018**, *4*, 1134–1145, 10.1021/acscentsci.8b00307.

47. Shields, B. J. et al. Bayesian reaction optimization as a tool for chemical synthesis. *Nature* **2021**, *590*, 89–96, 10.1038/s41586-021-03213-y.

48. Zhou, Z.; Li, X.; Zare, R. N. Optimizing chemical reactions with deep reinforcement learning. *ACS Cent. Sci.* **2017**, *3*, 1337–1344, 10.1021/acscentsci.7b00492.

49. Sakurai, A. et al. Ultranarrow-band wavelength-selective thermal emission with aperiodic multilayered metamaterials designed by bayesian optimization. *ACS Cent. Sci.* **2019**, *5*, 319–326, 10.1021/acscentsci.8b00802.

50. Yamawaki, M.; Ohnishi, M.; Ju, S.; Shiomi, J. Multifunctional structural design of graphene thermoelectrics by bayesian optimization. *Sci. Adv.* **2018**, *4*, eaar4192, 10.1126/sciadv.aar4192.

51. Attia, P. M. et al. Closed-loop optimization of fast-charging protocols for batteries with machine learning. *Nature* **2020**, *578*, 397–402, 10.1038/s41586-020-1994-5.

52. Griffiths, R.-R.; Hernández-Lobato, J. M. Constrained Bayesian optimization for automatic chemical design using variational autoencoders. *Chem. Sci.* **2020**, *11*, 577–586, 10.1039/C9SC04026A.

53. Ziatdinov, M. et al. Hypothesis learning in an automated experiment: application to combinatorial materials libraries. *Adv. Matter.* **2021**, 2201345. 10.1002/adma.202201345.

54. Ziatdinov, M.; Ghosh, A.; Kalinin, S. V. Physics makes the difference: Bayesian optimization and active learning via augmented gaussian process. *arXiv preprint arXiv:2108.10280*, **2021**.

55. Shahriari, B.; Swersky, K.; Wang, Z.; Adams, R. P.; De Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* **2015**, *104*, 148–175, 10.1109/JPROC.2015.2494218.

56. Frazier, P. I. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, **2018**.

57. Ax adaptive experimentation platform. https://ax.dev/ (accessed December, 30, 2021).

58. Lindauer, M. et al. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *arXiv preprint arXiv:2109.09831*, **2021**.

59. Fauvel, T.; Chalk, M. Contextual Bayesian optimization with binary outputs. *arXiv preprint arXiv:2111.03447*, **2021**.

60. Venkatram, S. et al. Predicting crystallization tendency of polymers using multifidelity information fusion and machine learning. *J. Phys. Chem. B* **2020**, *124*, 6046–6054, 10.1021/acs.jpcb.0c01865.

61. Tran, A.; Tranchida, J.; Wildey, T.; Thompson, A. P. Multi-fidelity machine-learning with uncertainty quantification and bayesian optimization for materials design: Application to ternary random alloys. *J. Chem. Phys.* **2020**, *153*, 074705, 10.1063/5.0015672.

62. Swersky, K.; Snoek, J.; Adams, R. P. Multi-task Bayesian Optimization. *Advances in Neural Information Processing Systems 26 (NIPS 2013)*; NeurIPS Proceedings, **2013**.

63. Sbalzarini, I. F.; Müller, S.; Koumoutsakos, P. Multiobjective optimization using evolutionary algorithms. In *Proceedings of the summer Program*; Citeseer, **2000**; Vol. 2000, pp 63–74.

64. Rolland, P.; Scarlett, J.; Bogunovic, I.; Cevher, V. High-dimensional bayesian optimization via additive models with overlapping groups. In *International conference on artificial intelligence and statistics*; PMLR, **2018**; pp. 298–307.

65. Moriconi, R.; Deisenroth, M. P.; Kumar, K. S. High-dimensional Bayesian optimization using low-dimensional feature spaces. *Mach. Learn.* **2020**, *109*, 1925–1943, 10.1007/s10994-020-05899-z.

66. Eriksson, D.; Pearce, M.; Gardner, J.; Turner, R. D.; Poloczek, M. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*; The MIT Press, **2019**; Vol. *32*, pp 5496–5507.

67. Turner, R. et al. Bayesian optimization is superior to random search for machine learning hyperparameter tuning: Analysis of the black-box optimization challenge 2020. *arXiv preprint arXiv:2104.10201*, **2021**.

68. Korovina, K. et al. Chembo: Bayesian optimization of small organic molecules with synthesizable recommendations. In *International Conference on Artificial Intelligence and Statistics*; PMLR, **2020**; pp. 3393–3403.

69. Botorch bayesian optimization in pytorch. https://botorch.org/ (accessed December 30, 2021).

70. Emukit | emukit is a highly adaptable python toolkit for enriching decision making under uncertainty. https://emukit.github.io/ (accessed December 30, 2021).

71. https://pymoo.org/.

72. perone/pyevolve: Pyevolve. https://github.com/perone/Pyevolve (accessed December 30, 2021).

73. Hyperopt documentation. http://hyperopt.github.io/hyperopt/ (accessed December 30, 2021).

74. Optuna - a hyperparameter optimization framework. https://optuna.org/ (accessed December 30, 2021).

75. Davies, D. W. et al. Computational screening of all stoichiometric inorganic materials. *Chem* **2016**, *1*, 617–627, 10.1016/j.chempr.2016.09.010.

76. Davies, D. W. et al. Smact: Semiconducting materials by analogy and chemical theory. *J. Open Source Softw.* **2019**, *4*, 1361, 10.21105/joss.01361.

77. Gaultois, M. W. et al. Data-driven review of thermoelectric materials: performance and resource considerations. *Chem. Mater.* **2013**, *25*, 2911–2920, 10.1021/cm400893e.

78. Ward, L. et al. Matminer: An open source toolkit for materials data mining. *Comput. Mater. Sci.* **2018**, *152*, 60–69, 10.1016/j.commatsci.2018.05.018.

79. Davies, D. W. et al. Computer-aided design of metal chalcohalide semiconductors: from chemical composition to crystal structure. *Chem. Sci.* **2018**, *9*, 1022–1030, 10.1039/C7SC03961A.

80. Sanchez-Lengeling, B.; Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science* **2018**, *361*, 360–365, 10.1126/science.aat2663.

81. Coulom, R. Efficient selectivity and backup operators in monte-carlo tree search. In *International conference on computers and games*; Springer, **2006**; pp. 72–83.

82. More, T. *Utopia*; **1551**.

83. https://github.com/jmetal/jmetalpy.

84. Burger, B. et al. A mobile robotic chemist. *Nature* **2020**, *583*, 237–241, 10.1038/s41586-020-2442-2.

85. King, R. D. et al. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature* **2004**, *427*, 247–252, 10.1038/nature02236.

Footnotes

i. The database is used to compute, archive, analyze experimentally reported or **in silico** materials for advanced technological applications. The total number of entries is reported. The "Workflow" column displays existing automation tools that preprocess, execute, and analyze each new material entry. The availability of application programming interfaces (**APIs**) to interface with the database is also indicated.

ii. Utopia was first used by Thomas More in his 16th-century book (82) of that name and derives from the Greek word meaning no-place.

iii. Genetic algorithms may be within the scope of this book, but they are covered well elsewhere.

Glossary

API: Application programming interface a software interface, which allows two or more applications to interact.

AUTOGLUON: A package for AutoML of deep and classical ML https://auto.gluon.ai/stable/index.html

AUTOKERAS: A package for AutoML of deep neural networks https://autokeras.com/.

AutoML: A field of machine learning interested in automated design of ML models.

Autoencoders: Unsupervised statistical model that learns a low-dimensional representation of data by reconstructing it.

Bayesian information criterion (BIC): Criterion for model selection from a finite set of models based on a penalization of additional parameters.

Bayesian neural networks (BNNs): Neural networks defined by a distribution over parameters and target variables.

BM3D method: A classical denoising technique https://pypi.org/project/bm3d/.

CAPTCHA: Completely Automated Public Turing—tools used by computers to distinguish humans from bots—often involving identifying objects from images.

Cauchy distribution: Continuous distribution of the ratio of two independent normally distributed random variables with mean zero.

Chemical space: The set of possible chemistries in which we are interested for a given problem.

Child nodes: Any subnode of a given node is called a child node, and the given node, in turn, is the child's parent.

Class activation maps: A method for interpreting CNNs (see SECTION 2.6.2).

Clustering algorithm: Methods for picking out classes in data (see SECTION 2.4.3).

Clustering: Unsupervised machine learning method to extract structured partitions from a dataset.

Compressed Representation: A lower dimensional representation of data, eg converting a large image to a smaller one by jpeg compression.

Concatenation: The operation of joining character objects end-to-end.

Confidence measure: A measure of how reliable we think the output of model is.

Convolutional Neural networks: A specific neural net type of architecture for dealing with large input data—see SECTION 3.2 for more details.

Convolutional layers: Elements that are assembled to make a convolutional neural network.

Coulomb matrix: Molecular featurization of the atoms or species in a molecule based on pairwise interactions.

Covariance matrix: Matrix that measures the statistical dependence of the random variables in a system.

Covariance: Statistical measurement of the relationship between two variables.

Cross validation: Statistical technique to validate a model by repeatedly splitting the available data into subsets of training and testing data.

Cumulative probability: The aggregate probability based on integrating over a probability density function.

Data augmentation: Extending the training data set by performing simple operations on inputs, for example rotating or translating images for training.

Deep learning: Broadly any machine learning method based on neural networks.

Denoising: Removing noise from experimental measurements.

Descriptors: Vectors used as input to a ML algorithm.

Design space: Possible combination of continuous, categorical or discrete variables that define a target variable or figure of merit.

Deterministic uncertainty quantification: An approach for adding confidence measures to classification ML models.

Discriminative model: Model of the conditional probability distribution of a target variable given an observed variable.

Distance metrics: Ways of measuring similarity of data points see SECTION 1.3.1.

Domain knowledge: Theoretical or practical knowledge for a specific scientific or engineering domain.

Downsampling procedure: Methods to reduce the size of the representation in a deep learning model.

ElemNet: A particular NN architecture for predicting materials properties.

Embeddings: Low dimensional representations of data.

Ensemble methods: Machine learning methods that combine various independent models to provide improved performance. Methods where several models are used at once.

EPR: Electron paramagnetic resonance.

Euclidian distance: A measure of distance based on the squares of the differences in dimensions.

Expectation: Weighted average of the value of a random variable.

Figure of merit: Target variable that we are interested in optimizing or engineering.

Filters: Matrices within a CNN they serve to pick out certain features from data.

Fitness factor: In genetics—it is the quantitative representation of individual reproductive success.

Gaussian distribution: Continuous bell-shaped or normal probability distribution.

Gaussian mixture model (GMM): Probabilistic model that assumes that a dataset is generated from a finite mixture of Gaussian distributions with unknown parameters.

Gaussian processes (GP): Non-parametric statistical method that models the data as a Gaussian process defined by a prior covariance.

Generative adversarial neural network (GAN): Generative model that works by combining two neural networks, a generator and a discriminator network, trained on adversarial fashion.

Generative model: Model of the joint probability distribution of an observed variable and a target variable.

Herfindahl–Hirschman index: A common measure of market concentration and is used to determine market competitiveness.

Heuristics: "rule-of-thumb" to approximate the knowledge of a system or solve a problem.

Hidden layer: The layers in the middle of an NN, all layers except input and output layers are hidden layer.

High-performance computing: Large scale computing resources that typically perform many complex operations in parallel.

Hypervolume: A volume in more than three dimensions.

Hyperparameter: Parameters that control the learning process in machine learning.

IMAGENET: A large database of labelled photographs often used for training and assessing computer vision tasks https://www.image-net.org/.

INS: Inelastic neutron scattering.

In silico: Experiments performed on a computer as opposed to in a physical laboratory.

Interpretability: The quality of a statistical or machine learning model to explain its inner workings.

Invariant: An operation that changes the appearance but not the class/value of a sample.

Inverse design: Paradigm in which we start with target variables or figures of merit and design a material or molecule to satisfy the desired properties, often in generative manner.

JSON: JavaScript Object Notation a popular file format for storing and transporting data.

K-means clustering: Distance-based clustering method.

Kernel function: Weighting function used in a non-parametric statistical model.

Kullback–Leibler divergence: Statistical distance measuring how one probability distribution is different from a second.

Label mask: A map of labels for each pixel in an image.

Latent space: Low-dimensional representation of the variable space of machine learning problem.

LOCO-CV: Leave-cluster-out cross validation, a variation of cross-validation that biases a particular cluster of data in the data split.

Magnon spectra: The spectrum of energy transfers due to excitations of magnon states in a magnetic material.

Manifold Learning: Learning of a low-dimensional representation (manifold) of a system.

Marginalization: Method that requires summing over the possible values of one variable to determine the marginal contribution of another.

Markup language: A computer language that uses tags to define objects in a document—e.g., html.

Max pooling: A pooling operation that replaces a vector with the maximum value of that vector.

Maximum likelihood estimation: Method of estimating the parameters of an assumed probability distribution, given some observed data.

MC methods: A broad class of computational algorithms that rely on repeated random sampling to obtain numerical results.

MCMC inference: Numerical approximation of the posterior probability of unobserved variables.

Model capacity: Power of a statistical model to capture a real functional form.

Multilayer perceptron: A simple neural network consisting of densely connected layers only.

Natural language processing (NLP): A field of study concerned with understanding how language works.

NEURAL NETWORK INTELLIGENCE: A package for AutoML of deep neural nets https://nni.readthedocs.io/en/stable/.

(NP)-hard: The defining property of a class of problems that are informally at least as hard as the hardest problems in NP. Here NP means that the solution to the problems scales polynomially.

NMR: Nuclear magnetic resonance.

Non-negative matrix factorization: See SECTION 4.3.2.

Normal distribution: A probability distribution used to model phenomena that have a default behaviour and cumulative possible deviations from that behaviour.

Out-of-distribution detection method: A way to detect new data that does not look like the training data and therefore is likely to not work well with a model.

Parametric: Statistical model that assumes an explicit functional form or probability distribution.

Pooling: Collecting together elements from a vector and producing a new lower dimension vector, e.g., Summing or averaging.

Posterior: An update probability distribution after a data sampling step.

Principal component analysis: A method for dimensionality reduction see SECTION 2.4.2.1.

PyTorch: Tensor computation and deep learning library. https://github.com/pytorch/pytorch.

Pyro: Deep probabilistic programming library built on PyTorch. https://github.com/pyro-ppl/pyro.

Reconstruction error: In an autoencoder, the difference between the input and the output.

Representations: The form in which data is present in an ML model.

Sampling: Statistical process to extract a group of observation from a large population.

Schema: The organization of data as a blueprint for how a database is structured.

Self-supervised neural network: Neural network that learns useful representations from unlabeled data.

Semantic segmentation: Separating images into different regions based on what is depicted in each region.

Shapley Analysis: explanation technique for the outputs of non-linear models.

Sigmoid function: A type of activation function based on the sigmoid form https://en.wikipedia.org/.

Sigmoid squashing: Passing data through a sigmoid function.

Softmax: A function that takes a vector and normalizes it to a probability, similar to a partition function in statistical mechanics.

Solution space: The set of all possible solutions to a model.

Square root squashing: Taking the square root of an entire dataset.

Standard deviation: Measurement of the amount of variation of dataset or population.

Student's t-distribution: Continuous probability distribution defined by estimating the mean of a normally distributed population with small sample size and unknown standard deviation.

Surrogate models: A ML model that replaces a more expensive to compute mathematical model.

TEM: Transmission electron microscopy.

Tensorflow-probability: Probabilistic regression and statistical analysis library on Tensorflow. https://www.tensorow.org/probability.

TPOT: A package for AutoML of classical models http://epistasislab.github.io/tpot/.

t-SNE: t-Stochastic neighbor embedding.

U-net architecture: A structure of of CNN which compresses and then reconstructs data (see SECTION 4.2.1).

Uncertainty: Measure of the variability or randomness of a system. Can be epistemic, caused by incomplete knowledge of a system, or aleatoric, caused by natural randomness of a system.

Upsampling procedure: Methods to increase the size of the representation in a deep learning model.

Variational autoencoders: Generative version of an autoencoder model.

Variational inference: Analytical approximation of the posterior probability of unobserved variables.

Weak learners: ML models that cannot learn well on their own.

Weight matrix: A matrix of the weights associated with a layer in an NN.

XAS: X-ray absorption spectroscopy.

XPS: X-ray photoemission spectroscopy.