Computational Materials Science 152 (2018) 60–69

Contents lists available at ScienceDirect

Computational Materials Science

journal homepage: www.elsevier.com/locate/commatsci

Matminer: An open source toolkit for materials data mining

Logan Ward[a,b], Alexander Dunn[c,d], Alireza Faghaninia[c], Nils E.R. Zimmermann[c], Saurabh Bajaj[c,e], Qi Wang[c], Joseph Montoya[c], Jiming Chen[f], Kyle Bystrom[d], Maxwell Dylla[g], Kyle Chard[a,b], Mark Asta[d], Kristin A. Persson[c], G. Jeffrey Snyder[g], Ian Foster[a,b], Anubhav Jain[c,*]

[a] Computation Institute, University of Chicago, Chicago, IL 60637, United States
[b] Data Science and Learning Division, Argonne National Laboratory, Argonne, IL 60439, United States
[c] Lawrence Berkeley National Laboratory, Energy Technologies Area, 1 Cyclotron Road, Berkeley, CA 94720, United States
[d] Department of Materials Science and Engineering, University of California, Berkeley CA 94720, University of California, Berkeley, CA 94720, United States
[e] Citrine Informatics, Redwood City, CA 94063, United States
[f] Department of Chemical Engineering, University of Illinois, Urbana, IL 61801, United States
[g] Department of Materials Science and Engineering, Northwestern University, Evanston, IL 60208, United States

ARTICLE INFO

Keywords:
Data mining
Open source software
Machine learning
Materials informatics

ABSTRACT

As materials data sets grow in size and scope, the role of data mining and statistical learning methods to analyze these materials data sets and build predictive models is becoming more important. This manuscript introduces matminer, an open-source, Python-based software platform to facilitate data-driven methods of analyzing and predicting materials properties. Matminer provides modules for retrieving large data sets from external databases such as the Materials Project, Citrination, Materials Data Facility, and Materials Platform for Data Science. It also provides implementations for an extensive library of feature extraction routines developed by the materials community, with 47 featurization classes that can generate thousands of individual descriptors and combine them into mathematical functions. Finally, matminer provides a visualization module for producing interactive, shareable plots. These functions are designed in a way that integrates closely with machine learning and data analysis packages already developed and in use by the Python data science community. We explain the structure and logic of matminer, provide a description of its various modules, and showcase several examples of how matminer can be used to collect data, reproduce data mining studies reported in the literature, and test new methodologies.

1. Introduction

Recently, the materials community has placed a renewed emphasis in collecting and organizing large data sets for research, materials design, and the eventual application of statistical or "machine learning" techniques. For example, the mining of databases comprised of density functional theory (DFT) calculations has been used to identify materials for batteries [1,2], to aid the design of metal alloys [3,4], and for many other applications [5]. Importantly, such data sets present new opportunities to develop predictive models through machine learning techniques: rather than designing and programming such models manually, such techniques produce predictive models by learning from a body of examples. Machine learning models have been demonstrated to predict properties of crystalline materials much faster than DFT [6–9], estimate properties that are difficult to access via other computational tools [10,11], and guide the search for new materials [12–16]. With the continued development of general-purpose data mining methods for many types of materials data [17–19] and the proliferation of material property databases [20], this emerging field of "materials informatics" is positioned to have a continued impact on materials design.

In this paper, we describe a new software library, "matminer", for applying data-driven techniques to the materials domain. The main roles of matminer are depicted in Fig. 1: matminer assists the user in retrieving large data sets from common databases, extracts features to transform the raw data into representations suitable for machine learning, and produces interactive visualizations of the data for exploratory analysis. We note that matminer does not itself implement common machine learning algorithms; industry-standard tools (e.g., scikit-learn or Keras) are already developed and maintained by the larger data science community for this purpose. Instead, matminer's role is to *connect* these advanced machine learning tools to the materials domain.

* Corresponding author at: Lawrence Berkeley National Laboratory, Energy Technologies Area, 1 Cyclotron Road, Berkeley, CA 94720, United States.
E-mail addresses: loganw@uchicago.edu (L. Ward), AJain@lbl.gov (A. Jain).

https://doi.org/10.1016/j.commatsci.2018.05.018
Received 16 April 2018; Accepted 7 May 2018
Available online 25 May 2018

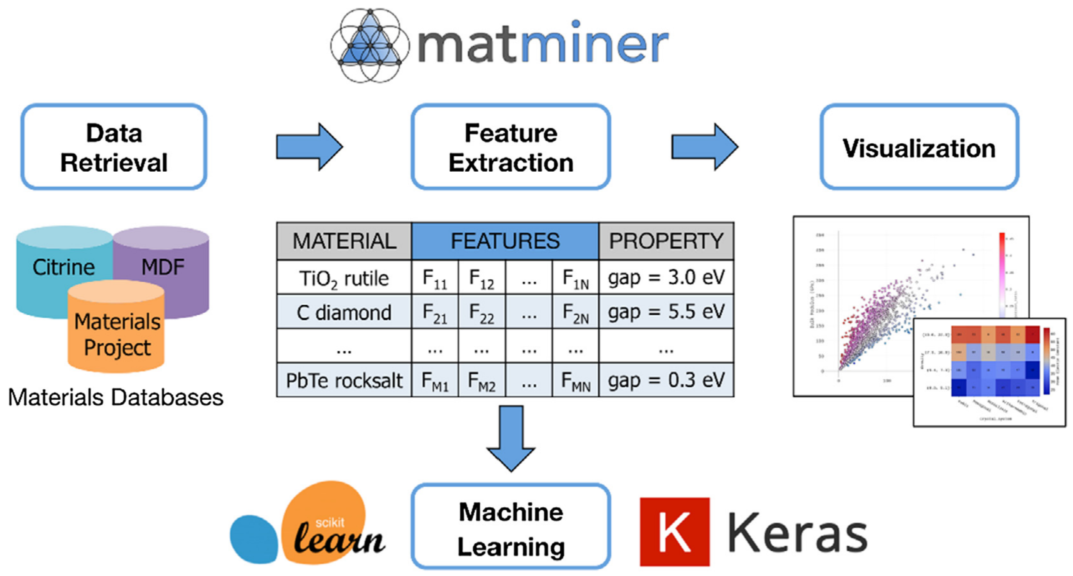

Fig. 1. Overview of the capabilities of matminer. Matminer aids the user in constructing a data pipeline for materials informatics and is composed of three main components: (1) tools for retrieving data from a variety of materials databases, (2) tools for extracting features (or descriptors) from materials data, and (3) re-useable and customizable recipes for visualizing materials data. Data is retrieved and processed in a way that makes it simple to integrate matminer with external machine learning libraries such as scikit-learn and Keras.

Matminer solves many problems encountered when conducting data-driven research. For example, learning the Application Programming Interface (API) for each data source and preprocessing retrieved data adds significant complexity to the task of building new machine learning models. Matminer provides a simplified interface that abstracts the details of these API interactions, making it easy for the user to query and organize large data sets into the standard *pandas* [21] data format used by the Python data science community. Furthermore, as we will further discuss later in the text, matminer implements a suite of 47 distinct feature extraction modules capable of producing thousands of physically relevant descriptors that can be leveraged by machine learning algorithms to more efficiently determine input-output relationships. Although many such feature extraction methods are reported in the literature, many lack an open source implementation. Matminer not only implements these domain-specific feature extraction methods but provides a unified interface for their use, making it trivial to reproduce or compare (and, eventually, extend) these methods. Finally, matminer contains many pre-defined recipes of visualizations for exploring and discovering different data relationships. In aggregate, these features allow for cutting edge materials informatics research to be conducted with a high-level, easy-to-use interface.

We note that prior efforts have produced software for computing features for materials (e.g., Magpie[22,23], pyMKS [24]), building deep learning models of molecular materials (e.g., deepchem [25,26]), providing turnkey machine learning estimates of various properties, or integrating machine learning with other software [27–29]. In contrast to these prior efforts (which have their own intended applications and scope), matminer is designed to interact and integrate with standard Python data mining tools such as *pandas* and *scikit-learn* [30], implements a library of feature generation methods ("featurizers") for a wide variety of materials science entities (e.g., compositions, crystal structures, and electronic structures), and includes tools to assist with data retrieval and visualization.

The source code for the version of matminer described in this manuscript (version 0.3.2) and examples of its use are available as supplementary information. Updated versions are regularly published to the Python Package Index (https://pypi.python.org/pypi/matminer). The actively developed version of matminer is available on GitHub at https://github.com/hackingmaterials/matminer. Matminer also includes a dedicated repository of examples and tutorials (many in an interactive, runnable Jupyter notebook format [31]) for using the data retrieval, featurization, and visualization tools, located at https://github.com/hackingmaterials/matminer_examples. Full documentation for matminer is also available from https://hackingmaterials.github.io/matminer/. The matminer code currently contains 109 unit tests to ensure the integrity of the code, which are run automatically with each code commit through a continuous integration process. A help forum for matminer is available at: https://groups.google.com/forum/#!forum/matminer.

2. Software architecture and design principles

A guiding principle of matminer is to integrate domain-specific knowledge and data about materials into larger ecosystem of Python data analysis software. The Python community has developed a rich suite of interoperable tools for data science, which are broadly used across the data science community and occasionally known as the "PyData" or "SciPy" stacks [32]. These libraries include NumPy and Scipy [33], which provide a suite of high-performance numerical methods, and Jupyter [31], which facilitates interactive data analysis. Matminer is designed to allow users to leverage these professional-level data science libraries for materials science studies.

A central tool in the PyData stack is the pandas DataFrame, which is a tabular representation of data similar to (but more powerful than) a virtual spreadsheet [21]. Pandas makes it possible, for example, to load a data set and perform many common data post-processing procedures, such as filtering, grouping, joining, computing rolling averages, and producing descriptive statistics. Additionally, data formatted into a pandas DataFrame can be easily used with other Python data analysis libraries, such as scikit-learn, numpy, and matplotlib. DataFrames can also be visualized as interactive tables within Jupyter notebooks. They can also be serialized into multiple formats to allow them to be archived and shared. Because of all the benefits and features that are achieved by transforming data into the DataFrame format, matminer's data retrieval API automatically formats data that it retrieves from external sources into this format. Data retrieved through matminer is thus immediately ready for a wide variety of tasks, including data cleaning, data exploration, data transformations, data visualization, and machine learning. As described in later sections, all data extraction, featurization, and visualization tools in matminer can generate or operate on pandas DataFrame objects.

Matminer is also designed to integrate closely with the scikit-learn machine learning library [30]. Scikit-learn is the *de facto* standard machine learning library for Python. In addition to its rich suite of machine learning algorithms, scikit-learn contains utilities useful for all aspects of the machine learning process (e.g., data preprocessing, model selection, hyperparameter tuning). Other machine learning libraries, such as Keras [34] and TensorFlow [35], also provide scikit-learn-compatible wrappers for their models, which further motivates the importance of making matminer easily compatible with scikit-learn. Matminer achieves integration with scikit-learn in two ways. First, the pandas DataFrame objects produced by matminer are tightly integrated with scikit-learn through the interoperability built in to the PyData stack. Second, the feature extraction methods implemented by matminer follow the same model (and, more formally, subclass) scikit-learn's preprocessing methods. This allows matminer feature extraction methods to be used with scikit-learn's Pipeline functionality and makes it easy to combine data processing methods present in the two libraries.

Matminer also heavily leverages the pymatgen [36] materials science library. Matminer's use of the pymatgen library makes it unnecessary to recreate complex or materials-science-specific algorithms (e.g., space group determination) when implementing new feature extraction methods. Overall, the software architecture of matminer is designed to bridge the gap between the professional-level data science tools developed by the Python community and the tools, techniques, and data specific to the materials domain.

3. Components of matminer

We now describe the main functions of matminer. We describe each of the three major components. data retrieval, featurization, and visualization, separately.

3.1. Data retrieval

The first step in data mining is to obtain a data set that is ideally large and diverse. There are several efforts underway in the materials community to build such databases of materials properties [37–44]. However, while the proliferation of databases is a great benefit to materials informatics, the use of these data sources is complicated by the fact that each database implements a different API, authentication method, and schema. One core function of matminer is to provide a consistent API around different databases and return the data in a form that is suitable for use in data mining tools.

At the time of writing, matminer supports data retrieval from four commonly used materials databases: Citrination [40,43], Materials Project (MP) [39], Materials Data Facility (MDF) [44], and Materials Platform for Data Science (MPDS) [45]. In addition, a generic MongoDB interface supports data retrieval from any MongoDB resource [46]. Below, we describe these data retrieval tools in detail:

(i) Citrination, developed by Citrine Informatics [40], is a centralized database that contains a variety of materials data, including experimental measurements and computational results, all in a common data schema – the "pif" [47]. The matminer data retrieval tool uses Citrine's citrination-client library to retrieve data from Citrination, and then converts the data from the hierarchical pif format to a tabular DataFrame format. In the process of converting the pif records, matminer retrieves all details describing a material (e.g., composition), its known properties, and how these properties were determined.

(ii) The Materials Project (MP) [39] primarily contains DFT [48,49] computed properties for over 60,000 compounds. In a similar fashion to the Citrination data extractor, matminer uses the existing MP API [50] (as implemented in the "MPRester" class of the Python Materials Genomics (pymatgen) library [36]) to query the database. MPDataRetrieval allows users to access a wide variety of properties of crystalline materials, including their crystal structures, electronic band structure, phonon dispersion, piezoelectric, dielectric and elastic constants.

(iii) The Materials Data Facility (MDF) is geared towards enabling researchers to publish their own data sets across a wide array of data types and materials subdisciplines. Matminer contains an MDFDataRetrieval class that uses the MDF's own Forge library [51] to perform the bulk of the search function but assists the user in formatting the final data to a standardized pandas DataFrame object.

(iv) The Materials Platform for Data Science (MPDS) [45] is a commercial database that includes phase diagram data (~60,000 entries), crystal structure data (~400,000 entries), and materials property values (~800,000 entries). The MPDSDataRetrieval class in matminer can retrieve and format information from this database.

(v) MongoDB is a popular tool in the data mining community due to its efficient and flexible data model [46]. For example, data generated through the atomate [52] computational suite is stored in such databases. The "MongoDataRetrieval" class of matminer converts MongoDB documents to rows of a pandas DataFrame.

All database tools are consistent in that they (i) contain a "get_dataframe" method that makes a query to the database and (ii) returns the data in a Pandas DataFrame object. The "get_dataframe" method for each source takes query instructions in a simple, standard format. We also provide the ability to run queries in the language specific to each source. In so, we provide both a novice-friendly route for using new data sources and maintain the ability for experts to access all features of a familiar data source. However, matminer does standardize the output such that data mining tools written for one database can be easily applied to another. One benefit of the uniformity of the APIs and output formats provided by matminer is that these features make it easy to combine data from multiple sources. The data merging tools built into the pandas DataFrame object facilitate this procedure. For example, it is straightforward to retrieve experimental band gap energies from Citrination and then easily compare those values with computed band gap energies from Materials Project or the OQMD (this specific example is described in detail in Section 4.2).

Matminer also contains several built-in datasets that can be loaded directly with a single line of Python and do not require external database calls or setting any options. These built-in datasets include: 1181 DFT-based elastic tensors [53], 941 DFT-based piezoelectric tensors [54], 1056 DFT-based dielectric constants [55], and 3938 DFT-based formation energies [39,56]. The built-in data sets make it simple to begin testing and developing data mining methods.

Finally, a user can load their own data set using the built-in tools of the pandas library, which can load data from CSV, Excel, or various other formats. This process can be conducted independently of matminer but the final data format will be compatible with the subsequent data featurization tools of matminer.

3.2. Data featurization: Transforming materials-related quantities into physically relevant descriptors

Typically, machine learning employs an intermediate step between compiling raw data and applying a machine learning algorithm. This step converts data from a raw format (often specialized for parsing by a particular software package or formatted for human readability) into a numerical representation that is useful for visualization or machine learning software. This process is called "feature extraction", "featurization", or generating "descriptors". Featurization transforms or augments the raw data (which might have a very complicated and difficult to learn relationship between inputs and outputs) into a set of physically relevant quantities that reflect the relationships between the input

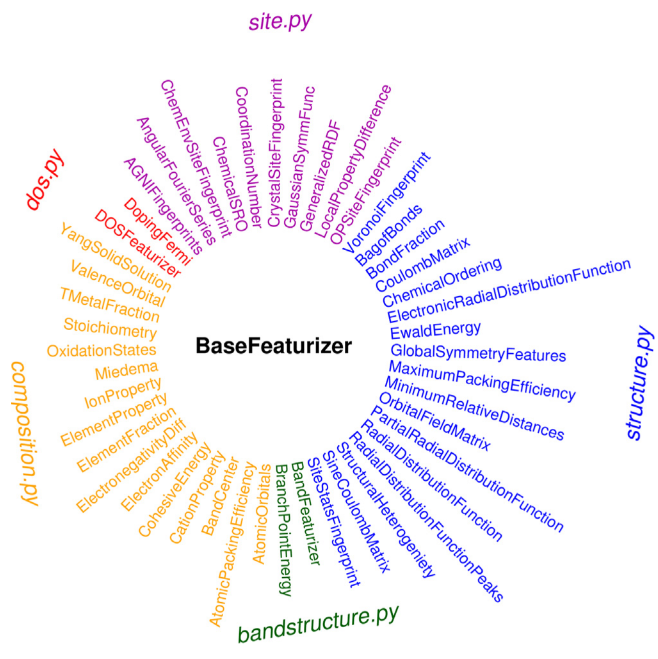

Fig. 2. Overview of the 47 featurizers that are currently available in five different modules (composition, site, structure, bandstructure, dos) of matminer. Each featurizer can generate one or hundreds of features, such that matminer as a whole is capable of producing thousands of individual features.

and output variables. The feature extraction step is one of the main ways in which one can exploit domain knowledge to vastly improve the performance of a machine learning algorithm. For example, common features that are extracted from a chemical composition include the differences in electronegativities of the component elements or the sum of atomic radii of the various elements.

Many generalizable featurization approaches have been proposed in the literature for different types of materials data [18,22,25,56–61]. However, the software required to use them are often unavailable, not open-source, or are distributed across many repositories. The lack of published software means that employing these methods in practice requires a significant time investment. Through matminer, we make these community developments in machine learning available to the community by providing open-source implementations of various featurization methods. Furthermore, despite the diversity of methodologies, matminer provides a uniform interface to all featurizers, freeing researchers to rapidly iterate through different approaches and determine the method best suited to their application.

All featurizer classes in matminer follow a common code-design pattern by inheriting from a base class, BaseFeaturizer, which defines the template for all featurization classes. BaseFeaturizer prescribes the four methods that must be implemented by each new featurizer:

1. The “featurize” method does the core work. It transforms materials data (e.g., a composition) into the desired feature values (e.g., element properties such as atomic weight, atomic radii, and Mendeleev number).
2. The “feature_labels” method provides descriptive labels that correspond to the feature values computed in the “featurize” method. These feature_labels can be thought of as column labels for the various features (and are indeed used as column labels when featurizing an entire DataFrame).
3. The “citations” method returns a list of BibTex-formatted references that a user should read to fully understand the features and cite if they are used. The citations method thus provides background and context for the featurizers and appropriate attribution to the original developers of the methodology.
4. The “implementors” method provides the name of the person(s) who implemented and are responsible for maintaining the featurizer. This is useful if one has a question, comment, or suggestion regarding the specific implementation details of a featurization method.

BaseFeaturizer provides additional functions that a user can call once these four methods are implemented. For example, the “featurize_dataframe” method uses the “featurize” and “feature_labels” operations to add the features to an entire pandas DataFrame. That is, featurize_dataframe will process potentially thousands or millions of rows of data, exploiting Python's multiprocess functionality to parallelize over available cores. The BaseFeaturizer class also follows the pattern used by featurizers in the scikit-learn machine learning library, which allows matminer featurization classes to be integrated easily with existing scikit-learn tools. For example, one can build a data processing pipeline that mixes some of the data normalization tools present in scikit-learn with the materials-specific features implemented in matminer.

Matminer contains, at the time of writing, a total of 47 featurizers that support the generation of features for diverse types of materials data. Each of these featurizers can produce many individual features/descriptors, such that it is possible to generate thousands of total features with the matminer code. For example, the ElementProperty featurizer will convert a chemical composition into various summary statistics of the properties of that composition's component elements (e.g., average ionic radius or standard deviation of elemental melting points). The BandFeaturizer will convert a complex electronic band structure into quantities such as band gap and the norm of k point coordinates at which the conduction band minimum and valence band maximum occur.

We have grouped the featurizers into five different Python modules based on the input data type: (i) composition, (ii) (crystal) structure, (iii) density of (electronic) states, (iv) band structure, and (v) (atomic) site. The featurizers available in matminer in each module are presented in Fig. 2. In Table 1, we briefly describe each featurizer and provide the canonical reference(s). The complete source code for each featurizer is available in matminer such that users can employ, fully inspect, and modify the implementations of these methods.

In addition to these individual featurizers, we provide a FunctionFeaturizer that combines individual features into functions such as products, quotients, logarithms, or any arbitrary mathematical expression. This procedure allows one to generate a large space of candidate features from even a small number of initial input features and has been observed to be useful in several previous works in the materials domain [18,62]. The implementation in matminer leverages the sympy library [63] which can eliminate symbolically redundant features.

3.3. Data visualization

A crucial step of a materials informatics workflow is visualizing data, which is helpful in understanding outliers, selecting features, and guiding the machine learning process. Many data-driven materials studies generate a standard suite of similar charts, such as heatmaps or two-dimensional scatter plots, which condense multiple complex relationships into simple, informative figures. For example, visualizing distributions of data (such as histograms and violin plots) at intermediate steps in the workflow process is a useful tool for pruning data and identifying outliers. Matminer drastically simplifies making many common visualizaitons.

Although there exist several excellent plotting libraries in Python (e.g., matplotlib [81] and seaborn [82]), these libraries are not designed to generate *interactive* plots that are also easy to share and serialize to a raw data format. Fortunately, the Plotly library [83] provides the needed functionality; however, its integration with standard

Table 1
A list of the featurizers currently implemented in matminer. Each row in the table provides the name of the relevant Python class, a concise description of the features it computes, and the appropriate references to the original methodology.

Featurizer	Description	Reference
	composition.py	
AtomicOrbitals	Highest occupied molecular orbital (HOMO) and lowest unoccupied molecular orbital (LUMO) using orbital energies from NIST.	[64]
AtomicPackingEfficiency	Packing efficiency based on a geometric theory of the amorphous packing	[90]
BandCenter	Estimation of absolute position of band center using geometric mean of electronegativity.	[65]
CationProperty	Element property attributes of cations in a composition	[66]
CohesiveEnergy	Cohesive energy per atom of a compound by adding known elemental cohesive energies from the formation energy of the compound.	[67]
ElectronAffinity	Average electron affinity times formal charge of anion elements.	[66]
ElectronegativityDiff	Statistics on electronegativity difference between anions and cations.	[66]
ElementFraction	Fraction of each element in a composition.	–
ElementProperty	Statistics of various element properties	[22,36,66]
IonProperty	Maximum and average ionic character, whether a composition is charge-balanced	[22]
Miedema	Formation enthalpies of intermetallic compounds, solid solutions, and amorphous phases using semi-empirical Miedema model (and some extensions).	[68–70]
OxidationStates	Statistics of oxidation states.	[66]
Stoichiometry	L^p norm-based stoichiometric attributes.	[22]
TMetalFraction	Fraction of magnetic transition metals.	[66]
ValenceOrbital	Valence orbital attributes such as the mean number of electrons in each shell.	[22,66]
YangSolidSolution	Mixing thermochemistry and size mismatch terms of Yang and Zhang (2012)	[91]
	structure.py	
BagofBonds	Representation where each structure is represented based on the types of and distances between each pair of sites	[71]
BondFraction	Fraction of nearest neighbors between each element (e.g., C-O vs C-C) bonds	[71]
ChemicalOrdering	How much the ordering of species in the structure differs from random	[6]
ColoumbMatrix	Coulomb matrix ($M_{ij} = Z_i Z_j / \lvert R_i - R_j \rvert$ for $i \neq j$, $Z_i^{2.4}/2$ *for* $i = j$, with Z_i and R_i the nuclear charge and the position of atom i).	[7]
ElectronicRadialDistributionFunction	RDF in which the positions of neighboring sites are weighted by electrostatic interactions inferred from atomic partial charges.	[72]
EwaldEnergy	Energy from Coulombic interactions based on charge states of each site	[73]
GlobalSymmetryFeatures	Symmetry information such as spacegroup number and (enumerated) crystal system type.	–
MaximumPackingEfficiency	Maximum possible packing efficiency of this structure	[6]
MinimumRelativeDistances	Closest neighbor distances for all sites, where relative distance are used $f_{ij} = r_{ij}/(r_i^{\text{atom}} + r_j^{\text{atom}})$ with r_i^{atom} being radius of atom or ion i.	[74]
OrbitalFieldMatrix	Average of the 32 by 32 matrix descriptions of the chemical environment of each atom in the unit cell, based on the group numbers, row numbers (optional), distances of coordinating atoms, and Voronoi Polyhedra weights.	[75]
PartialRadialDistributionFunction	Frequency of bonds across varied ranges of length between certain pairs of elements	[58]
RadialDistributionFunction	Conventional radial distribution function (RDF) of a crystal structure.	–
RadialDistributionFunctionPeaks	Distances of the largest peaks in the RDF of a structure	–
StructuralHeterogeneity	Variance in the bond lengths and atomic volumes in a structure	[6]
SineCoulombMatrix	Same as the CoulombMatrix, except the nondiagonal elements are weighted by $\boldsymbol{B}\cdot \sum_{k=\{x,y,z\}} \hat{e}_k \sin^2 [\pi \hat{e}_k \boldsymbol{B}^{-1}\cdot \boldsymbol{r_{ij}}]_2^{-1}$, where $\boldsymbol{r_{ij}}$ is the vector between atoms i and j and $\boldsymbol{B}$ is the lattice matrix, rather than $1/r_{ij}$.	[56]
SiteStatsFingerprint	Generates features pertaining to an entire structure by computing statistics across the features of all sites in the unit cell	–
	bandstructure.py	
BandFeaturizer	Non-zero band gap, direct band gap, k-point degeneracy, relative energy to CBM/VBM at arbitrary list of k-points and at conduction/valence bands.	–
BranchPointEnergy	Branch-point energy by averaging the energy of arbitrary number of conduction and valence bands throughout the full Brillouin zone.	[76]
	dos.py	
DopingFermi	Fermi level associated with a specified carrier concentration and temperature	–
DOSFeaturizer	The top N contributors to the density of states at the valence and conduction band edges. Includes chemical specie, orbital character, and orbital location information.	–
	site.py	
AGNIFingerprints	Fingerprints based on integrating the distances product of the radial distribution function with a gaussian window function	[77]
AngularFourierSeries	Encodes both radial and angular information about site neighbors. Each feature is a sum of the product of two distance functions between atoms that share the central site and the cosine of the angle between them.	[17]
ChemEnvSiteFingerprint	Local site environment fingerprint computed with the chemenv module in pymatgen.	[74,78]
ChemicalSRO	Chemical short-range ordering features to evaluate deviation of local chemistry with the nominal composition of entire structure.	[79]
CoordinationNumber	Number of first nearest neighbors of a site	[74]
CrystalSiteFingerprint	Coordination number percentage and local structure order parameters computed from the neighbor environment of a site; Voronoi decomposition-based neighbor finding.	[74]
GaussianSymmFunc	Gaussian radial and angular symmetry functions originally proposed for fitting machine learning potentials.	[28,80]
GeneralizedRadialDistributionFunction	A radial distribution function where the bins do not need to act in a "histogram" mode. The bins can be any arbitrary function such as Gaussians, Bessel functions, or trig functions.	[17]
LocalPropertyDifference	Differences in elemental properties between site and its neighboring sites	[6]
OPSiteFingerprint	Local structure order parameters computed from the neighbor environment of a site; distance-based neighbor finding.	[74]
VoronoiFingerprint	Voronoi indices, i-fold symmetries and statistics of Voronoi facet areas, sub-polyhedron volumes and distances derived by Voronoi tessellation analysis.	[79]

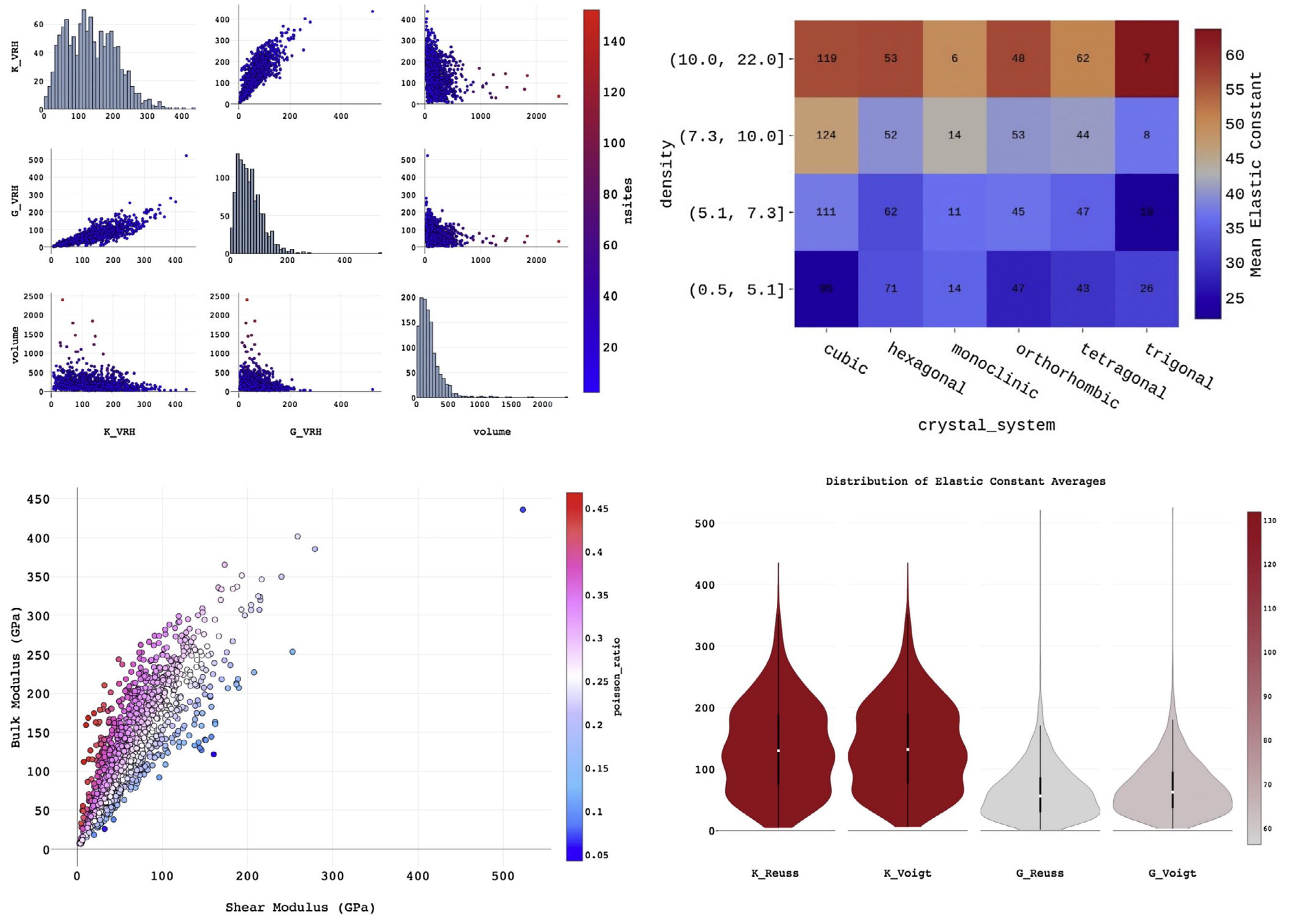

Fig. 3. Examples of plots based on a built-in data set of elastic tensors [53] and generated through the FigRecipes interface. Clockwise from top-left: a scatter matrix, a heat map, a violin plot, and an x-y plot with color dimension that represents Poisson ratio.

Python data libraries such as pandas remains minimal. Thus, to accelerate visualization, matminer includes its own module, FigRecipes, that provides a set of pre-defined methods for creating well-formatted, common figures (Fig. 3). Plotly was selected as the backend of FigRecipes because (1) its interactivity enables the rapid identification (via Plotly “hoverinfo”) of outliers in data sets, which are frequently the most important data points in materials informatics studies, and (2) it uses a portable JSON representation of Plotly plots, which enables FigRecipes to output fine-tunable Plotly figure templates with a few lines of code. Furthermore, interactive Plotly figures can be shared easily on the web via URL, which facilitates making figures collaboratively.

The PlotlyFig class in matminer's FigRecipes module supports seven types of plots: x-y plots, scatter matrices, histograms, bar charts, heatmaps, parallel plots, and violin plots. FigRecipes also facilitates generating often-overlooked figures, such as parallel coordinate plots [84], which have been found to be useful in materials science applications as they provide a technique for representing relationships between variables in high dimensional spaces. PlotlyFig can generate several plots using the same DataFrame content, automatically determining relevant labels and legend information from DataFrame column headers. PlotlyFig can also automatically bin and transform data to be compatible with the selected plot type; for example, PlotlyFig can automatically bin data in a DataFrame to create a heatmap and can generate multiple violin plots from a DataFrame lacking an explicit 'group' column. PlotlyFig's succinct syntax and automatic conversions provide robust extensions of Plotly's plotting functionality.

PlotlyFig interfaces with several Plotly options for visualization, such as interactive offline plotting, static images, and the online Plotly interface. All figures generated with FigRecipes can be returned as a PlotlyDict object, a JSON-like dict representation of a figure that can be serialized and stored for reproducibility and sharing. This ability makes FigRecipes a useful plotting tool for creating scientific representations of data; complex data can first be easily converted into a PlotlyDict template, and this figure template specifically edited to create custom-made publication-quality images.

4. Examples of using matminer

Next, we present four usage examples that showcase the capabilities of matminer. The source code for these and other examples are available as part of the matminer_examples GitHub repository (https://github.com/hackingmaterials/matminer_examples). Users can download, inspect, and execute the full code for these examples themselves and modify them for their own applications.

4.1. Retrieving data sets and visualizing them

In our first example, we use matminer's CitrineDataRetrieval tool to collect the experimental thermoelectric materials properties reported by Gaultois et al. [85] and compiled in the Citrine database. We then, with the help of FigRecipes, visualize this data in just a few lines of code. An example output is depicted in Fig. 4, in which electrical conductivity, Seebeck coefficient, thermal conductivity and the figure of merit of thermoelectric materials (zT) are visualized in a single plot. This example effectively recreates Fig. 3 of Ref. [85] but allows the user

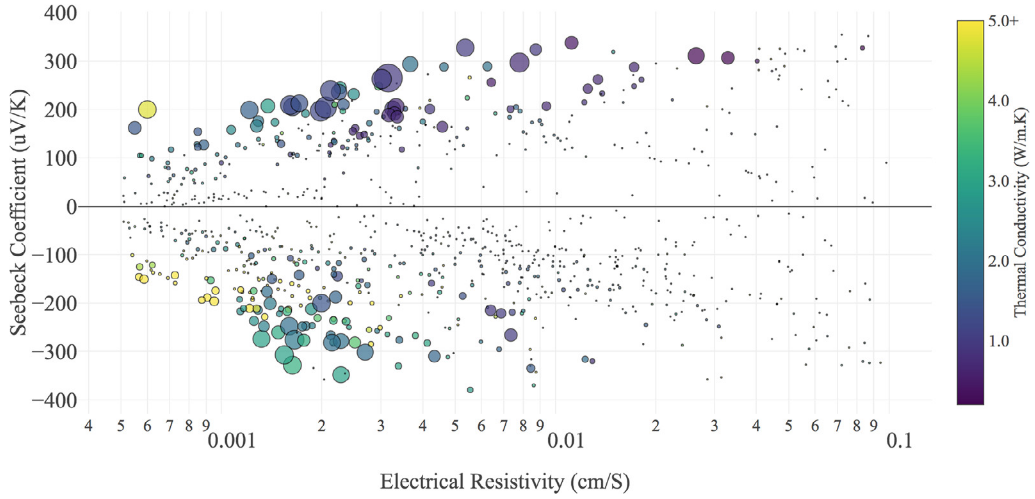

Fig. 4. Thermoelectric properties of nearly 1000 materials compiled by Gaultois et al. [85] and as retrieved and visualized with matminer. The marker size is scaled according to the figure of merit, zT.

to process the data locally, perhaps adding in their own data filtering or featurization procedure. Once the data set is loaded into a DataFrame called "df_te", re-creating this figure can be accomplished by two Python commands, as follows:

```
pf = PlotlyFig(df_te, x_scale='log',
   x_title='Electrical Resistivity (cm/S)',
        y_title='Seebeck Coefficient (uV/K)',
        colorbar_title='Thermal Conductivity (W/
   m.K)')
pf.xy(('Electrical resistivity', 'Seebeck
   coefficient'),
      labels='chemicalFormula', sizes='zT',
      colors='Thermal conductivity', color_range=[0,
   5])
```

The first line defines the data used by the charts and names for the axes. The second line defines the data being plotted. Further details are handled automatically. For example, zT values are normalized for better visualization. In addition, because the user specified a color_range of [0, 5] for the thermal conductivity values, all thermal conductivity values equal or greater than 5 are denoted by a bright yellow color with a "5+" tick label is automatically added to the colorbar. Thus, FigRecipes includes both automatic and customizable options that balance speed and flexibility of visualization.

4.2. Comparing experiment and theory data

In another example, we retrieve all the experimental band gap data available in Citrine and compare them with the calculated values available in the Materials Project [39]. Comparing data from two different sources is often complicated by the need to match records from one system to another. In this example, we need to find records in Materials Project with the same composition. As many entries in Citrination lack an associated crystal structure, we match each band gap to the ground-state structure with the same composition in Materials Project. Merging these data sources also demonstrates how combining data sources can fill in missing information from each database. Owing to the CitrineDataRetrieval class, the Material Project API and Pandas, merging the two data sources requires only 9 lines of code:

```
c = CitrineDataRetrieval() # Create an adapter to the
   Citrine Database.
df = c.get_dataframe(prop='band gap',
   data_type='EXPERIMENTAL',
        show_columns=['chemicalFormula', 'Band
   gap'])
mpr = MPRester()
def get_MP_bandgap(formula):
  formula = Composition
   (formula).get_integer_formula_and_factor()[0]
  strcs = mpr.get_data(formula)
  if strcs:
    return sorted(strcs, key = lambda e: e
   ['energy_per_atom'])[0]['band_gap']
df['DFT Band gap'] = data['chemicalFormula'].apply
   (get_MP_bandgap)
```

As shown in Fig. 5, most computed DFT band are lower than the experimental values, which is a known drawback of DFT calculations performed using LDA or GGA functionals [86–88]. Because the comparison is performed automatically, minimal human effort is required to update the result as new experimental band gaps are added to Citrination or new calculations are performed by Materials Project. As exemplified by this example, the tools matminer provides to automate data-driven analyses can make reproducing data-driven materials studies much simpler.

4.3. Building a machine learning model using OQMD data

To demonstrate how matminer can facilitate the process of machine learning, we recreate a machine learning model from a 2016 paper by Ward et al. [22] In this work, the authors trained a machine learning model using data from the Open Quantum Materials Database (OQMD) [42,92] to predict the formation enthalpy of crystalline materials given their composition.

The first step is to retrieve the OQMD data used by Ward et al., which is available through the Materials Data Facility [44]. We can use matminer's data retrieval tools to access this data directly with only three lines of code:

```
mdf = retrieve_MDF.MDFDataRetrieval
   (anonymous = True)
query_string = 'mdf.source_name:oqmd_v3 AND
  (oqmd_v3.configuration:static OR
  oqmd_v3.configuration:standard) AND
  dft.converged:True'
data = mdf.get_data(query_string,
  unwind_arrays=False)
```

The next step is to process the dataset to create a suitable training set:

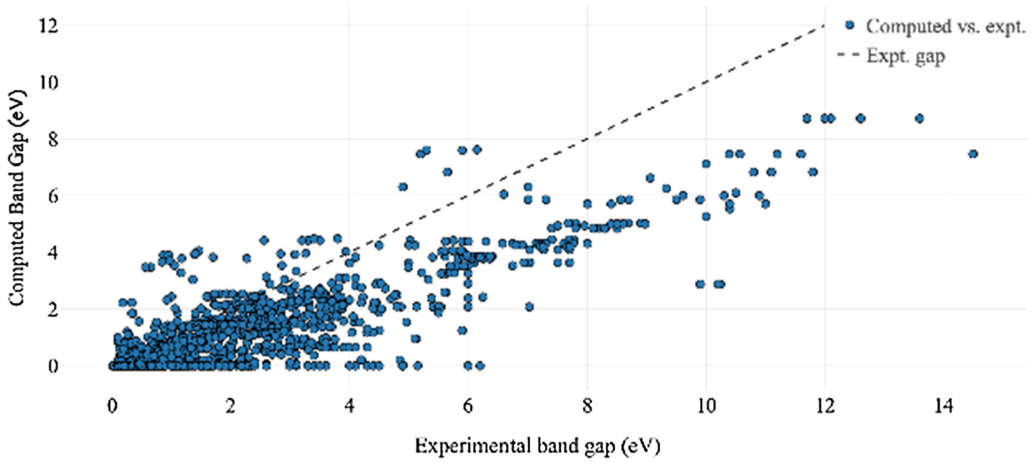

Fig. 5. Comparison of experimentally-measured band gap energies retrieved from the Citrine database to DFT-PBE computed electronic band gaps retrieved from the Materials Project. As expected, the data set demonstrates that computed band gaps underestimate experimental values [86–88].

removing errors, duplicates, and outliers. For example, removing all entries which lack a computed formation enthalpy can be achieved in a single line of Python:

```
data = data[~ data['oqmd_v3.delta_e.value'].isnull
   ()]
```

The third step in building a machine learning model is computing a representation. We have implemented the techniques developed by Ward et al. into matminer as Featurizer classes. These Featurizers, which operate on DataFrame objects, are also simple to run:

```
featurizer = MultipleFeaturizer([
  cf.Stoichiometry(),
   cf.ElementProperty.from_preset("magpie"),
  cf.ValenceOrbital(props=['avg']), cf.IonProperty
   ()])
featurizer.featurize_dataframe(data,
   col_id='composition_obj')
```

These two lines of code generate the 145 features used by Ward et al. and store them within the DataFrame object. At this point, the data are in a form that is compatible with existing machine learning libraries, such as scikit-learn or Keras. After using scikit-learn's Random Forest implementation and cross-validation utilities, we find that our model achieves a MAE of 0.071 eV/atom in 10-fold cross-validation, which is consistent with the results reported by Ward et al. (as low as 0.088 eV/atom using a different tree-based ML method). Overall this example serves to demonstrate how matminer, combined with community-standard data analysis and machine learning libraries, facilitates the construction of machine learning models from materials data.

4.4. Comparing crystal structure featurization methods

Another benefit of matminer is that it simplifies comparing machine learning methods. To illustrate, we used matminer to compare three methods for predicting the formation energy for a given crystal structure: the Sine Coulomb Matrix (SCM) [56], the Orbital Field Matrix (OFM) [75], and a recent modification to the OFM in development that also includes the row of each element in the periodic table in addition to the column (OFMR).

The first step in comparing the models is to gather training sets. For this task, we use the original 3938 structures selected by Faber et al. from the Materials Project (FLLA) [56] and a dataset of all 7735 stable ternary oxides in the Materials Project with unit cell size at most 30 atoms (TER_OX). Gathering the data is simple with matminer. The FLLA data set is built into matminer and the TER_OX dataset can be gathered with a single MPDataRetrieval query:

```
from matminer.data_retrieval.retrieve_MP import
   MPDataRetrieval
mpr = MPDataRetrieval()
criteria = '*-*-O'
properties = ['structure', 'nsites',
   'formation_energy_per_atom',
'e_above_hull']
df = mpr.get_dataframe(criteria = criteria,
   properties = properties)
df = df[df['e_above_hull'] < 0.1]
df = df[df['nsites'] <= 30]
```

Each of the three methods use Kernel Ridge Regression (KRR) as the machine learning algorithm; we employ the implementation of this method from scikit-learn. scikit-learn includes a well-optimized implementation of KRR, and has a tool – GridSearchCV – for easily selecting the optimum kernel and regularization parameter for KRR [30]. We tested each method using five-fold cross validation, and used four-fold cross-validation when selecting optimizing hyperparameters for each fold. We tested Laplacian and RBF (radial basis function) kernels for both features, and used the r^2 value of the formation energy per atom predictions to score each hyperparameter set [30].

The orbital field matrix can be time consuming to calculate for a large dataset because of its size; however, the process can be accelerated by the parallelization feature of matminer. Matminer automatically runs in parallel across all available CPU cores using Python's multiprocessing package. The following code computes the OFM representation and automatically runs in parallel:

```
from matminer.featurizers.structure import
   OrbitalFieldMatrix
ofm = OrbitalFieldMatrix()
df = ofm.featurize_dataframe(df, 'structure')
```

The cross-validation results for the FLLA and TER_OX datasets are presented in Table 2. We find very close agreement between the Mean Absolute Error (MAE) reported by Faber et al. for the SCM (0.37 eV/atom) and our result with matminer of 0.387 eV/atom, despite minor differences in the cross-validation procedure [56]. This demonstrates that we are able to reproduce the methodology of a published machine learning paper and compare it with a new featurization method (OFMR) with very little effort.

Our results indicate that for both data sets, the OFMR outperforms the OFM featurizer, which in turn outperforms the SCM (Table 2). All methods perform better on the TER_OX dataset than the FLLA dataset, demonstrating that the specific data set influences both absolute and relative model performance. Featurization and evaluation of the OFM and OFMR take much longer than for the SCM because of the size of the

Table 2
Performance (in terms of both accuracy and time needed to featurize) of several machine learning methods on two different datasets: the FLLA [56] and TER_OX datasets. We compare the Sine Coulomb Matrix (SCM) [56], Orbital Field Matrix (OFM) [75], and Orbital Field Matrix + row in periodic table (OFMR). The performance scores are for each model in 5-fold cross-validation. Each model was run on 24, 2.3 GHz processor cores on a system with 64 GB of RAM.

Dataset	Descriptor	MAE (eV/atom)	RMSE (eV/atom)	r^2	Featurize time (s)	Cross-validation time (h:mm:ss)
FLLA	SCM	0.387	0.575	0.708	2.0	0:07:42
	OFM	0.229	0.346	0.894	138.	0:50:40
	OFMR	0.171	0.277	0.932	138.	1:20:14
TER_OX	SCM	0.123	0.220	0.917	5.0	0:30:16
	OFM	0.090	0.140	0.967	366.	4:30:16
	OFMR	0.059	0.100	0.983	363.	7:06:42

descriptors, which may result in a time-accuracy tradeoff in some applications. We also note that Faber et al. have been developing updated structure representations [89] that in the future might be further compared to the current results. Being able to probe the applicability of different featurization methods for different data sets is significantly simplified by the ability to easily swap out different machine learning methods and datasets within a machine learning pipeline. This allows for rapid testing of new methods against various data sets.

5. Conclusion

Performing materials informatics requires developing a data pipeline that encompasses data retrieval, feature extraction, and visualization prior to the actual machine learning step. The matminer software described in this manuscript is designed to facilitate the development, reuse, and reproducibility of data pipelines for materials informatics applications. We have designed matminer to connect the domain-specific aspects of materials informatics (i.e., materials data extraction, feature extraction of materials science concepts, common plotting routines) with the professional level machine learning and data processing software already developed and in use by the Python community. It is our hope that matminer can serve as a community repository for new materials data analytics techniques as they become available such that researchers can rapidly develop and test new methods against standard techniques, accelerating the use of data mining in the materials community at large.

Acknowledgements

This code was intellectually led and primarily developed using funding provided by U.S. Department of Energy, Office of Basic Energy Sciences, Early Career Research Program, which funded the efforts of AJ, AD, AF, SB, and QW. LW and IF were supported by financial assistance award 70NANB14H012 from U.S. Department of Commerce, National Institute of Standards and Technology as part of the Center for Hierarchical Material Design (CHiMaD), by the National Science Foundation as part of the Midwest Big Data Hub under NSF Award Number: 1636950 "BD Spokes: SPOKE: MIDWEST: Collaborative: Integrative Materials Design (IMaD): Leverage, Innovate, and Disseminate," and by the Department of Energy contract DE-AC02-06CH11357. NER, JM, MA, and KAP were funded by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, Materials Sciences and Engineering Division under Contract No. DE-AC02-05-CH11231: Materials Project program KC23MP. JC and KC were supported by NSF, United States grant 1541450 (CC∗DNI DIBBS: Merging Science and Cyberinfrastructure Pathways: The Whole Tale). KWB acknowledges the University of California, Berkeley College of Chemistry for a summer research stipend. MD and GJS were funded by NSF DMR program Grant nos. 1334713 and 1333335.

This research used the Savio computational cluster resource provided by the Berkeley Research Computing program at the University of California, Berkeley (supported by the UC Berkeley Chancellor, Vice Chancellor for Research, and Chief Information Officer). This research used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

We thank all those in the materials community who have contributed code commits to matminer, including Ashwin Aggarwal, Evgeny Blokhin, Jason Frost, Matthew Horton, Kiran Mathew, Shyue Ping Ong, Sayan Rowchowdhury, and Donny Winston.

Appendix A. Supplementary material

Supplementary data associated with this article can be found, in the online version, at http://dx.doi.org/10.1016/j.commatsci.2018.05.018.

References

[1] H. Chen, G. Hautier, A. Jain, C. Moore, B. Kang, R. Doe, L. Wu, Y. Zhu, Y. Tang, G. Ceder, Chem. Mater. 24 (2012) 2009.
[2] M. Aykol, S. Kim, V.I. Hegde, D. Snydacker, Z. Lu, S. Hao, S. Kirklin, D. Morgan, C. Wolverton, Nat. Commun. 7 (2016) 13779.
[3] C. Nyshadham, C. Oses, J.E. Hansen, I. Takeuchi, S. Curtarolo, G.L.W. Hart, Acta Mater. 122 (2017) 438.
[4] S. Kirklin, J.E. Saal, V.I. Hegde, C. Wolverton, Acta Mater. 102 (2016) 125.
[5] A. Jain, K.A. Persson, G. Ceder, APL Mater. 4 (2016) 53102.
[6] L. Ward, R. Liu, A. Krishna, V.I. Hegde, A. Agrawal, A. Choudhary, C. Wolverton, Phys. Rev. B 96 (2017) 24104.
[7] M. Rupp, A. Tkatchenko, K.-R. Müller, O.A. von Lilienfeld, Phys. Rev. Lett. 108 (2012) 58301.
[8] J. Carrete, W. Li, N. Mingo, S. Wang, S. Curtarolo, Phys. Rev. X 4 (2014) 11019.
[9] L. Ward, C. Wolverton, Curr. Opin. Solid State Mater. Sci. 21 (2017) 167.
[10] J.C. Mauro, A. Tandia, K.D. Vargheese, Y.Z. Mauro, M.M. Smedskjaer, Chem. Mater. 28 (2016) 4267.
[11] E.W. Bucholz, C.S. Kong, K.R. Marchman, W.G. Sawyer, S.R. Phillpot, S.B. Sinnott, K. Rajan, Tribol. Lett. 47 (2012) 211.
[12] T.D. Sparks, M.W. Gaultois, A. Oliynyk, J. Brgoch, B. Meredig, Scr. Mater. 111 (2015) 10.
[13] R. Yuan, Z. Liu, P.V. Balachandran, D. Xue, Y. Zhou, X. Ding, J. Sun, D. Xue, T. Lookman, Adv. Mater. 1702884 (2018) 1702884.
[14] A. Mannodi-Kanakkithodi, A. Chandrasekaran, C. Kim, T.D. Huan, G. Pilania, V. Botu, R. Ramprasad, Mater. Today (2017), http://dx.doi.org/10.1016/j.mattod.2017.11.021.
[15] F.A. Faber, A. Lindmaa, O.A. von Lilienfeld, R. Armiento, Phys. Rev. Lett. 117 (2016) 135502.
[16] F. Ren, L. Ward, T. Williams, K.J. Laws, C. Wolverton, J. Hattrick-Simpers, A. Mehta, Sci. Adv. 4 (2018) eaaq1566.
[17] A. Seko, H. Hayashi, K. Nakayama, A. Takahashi, I. Tanaka, Phys. Rev. B 95 (2017) 144110.
[18] R. Ramprasad, R. Batra, G. Pilania, A. Mannodi-Kanakkithodi, C. Kim, Npj Comput. Mater. 3 (2017) 54.
[19] S.R. Kalidindi, ISRN Mater Sci. 2012 (2012) 1.
[20] J. Hill, G. Mulholland, K. Persson, R. Seshadri, C. Wolverton, B. Meredig, MRS Bull. 41 (2016) 399.
[21] W. McKinney, Proc. 9th Python Sci. Conf. 1697900 (2010) 51.
[22] L. Ward, A. Agrawal, A. Choudhary, C. Wolverton, Npj Comput. Mater. 2 (2016) 16028.
[23] http://bitbucket.org/wolverton/magpie.
[24] W. Daniel, B. David, F. Tony, K. Surya, R. Andrew, PyMKS: Materials Knowledge System in Python, 2014. doi: 10.6084/m9.figshare.1015761.
[25] Z. Wu, B. Ramsundar, E.N. Feinberg, J. Gomes, C. Geniesse, A.S. Pappu, K. Leswing, V. Pande, Chem. Sci. 9 (2018) 513.
[26] https://github.com/deepchem/deepchem.
[27] E. Gossett, C. Toher, C. Oses, O. Isayev, F. Legrain, F. Rose, E. Zurek, J. Carrete, N. Mingo, A. Tropsha, S. Curtarolo (2017) arXiv:1711.10744v1.
[28] A. Khorshidi, A.A. Peterson, Comput. Phys. Commun. 207 (2016) 310.
[29] https://github.com/libAtoms/QUIP.
[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, J. Mach. Learn. Res. 12 (2011) 2825.
[31] F. Perez, B.E. Granger, Comput. Sci. Eng. 9 (2007) 21.
[32] K.J. Millman, M. Aivazis, Comput. Sci. Eng. 13 (2011) 9.
[33] S. van der Walt, S.C. Colbert, G. Varoquaux, Comput. Sci. Eng. 13 (2011) 22.
[34] https://github.com/keras-team/keras.
[35] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A.

Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, 2015. < https://www.tensorflow.org/ > .
[36] S.P. Ong, W.D. Richards, A. Jain, G. Hautier, M. Kocher, S. Cholia, D. Gunter, V.L. Chevrier, K.A. Persson, G. Ceder, Comput. Mater. Sci. 68 (2013) 314.
[37] A. Frantzen, J. Scheidtmann, G. Frenzer, W.F. Maier, J. Jockel, T. Brinz, D. Sanders, U. Simon, Angew. Chemie Int. Ed. 43 (2004) 752.
[38] Y. Xu, M. Yamazaki, P. Villars, Jpn. J. Appl. Phys 50 (2011) 11RH02.
[39] A. Jain, S.P. Ong, G. Hautier, W. Chen, W.D. Richards, S. Dacek, S. Cholia, D. Gunter, D. Skinner, G. Ceder, K.A. Persson, APL Mater. 1 (2013) 11002.
[40] https://citrination.com.
[41] S. Curtarolo, W. Setyawan, S. Wang, J. Xue, K. Yang, R.H. Taylor, L.J. Nelson, G.L.W. Hart, S. Sanvito, M. Buongiorno-Nardelli, N. Mingo, O. Levy, Comput. Mater. Sci. 58 (2012) 227.
[42] J.E. Saal, S. Kirklin, M. Aykol, B. Meredig, C. Wolverton, JOM 65 (2013) 1501.
[43] J. O'Mara, B. Meredig, K. Michel, JOM 68 (2016) 2031.
[44] B. Blaiszik, K. Chard, J. Pruyne, R. Ananthakrishnan, S. Tuecke, I. Foster, JOM 68 (2016) 2045.
[45] https://mpds.io/.
[46] https://www.mongodb.com/.
[47] K. Michel, B. Meredig, MRS Bull. 41 (2016) 617.
[48] P. Hohenberg, W. Kohn, Phys. Rev. 136 (1964) B864.
[49] L.O. Wagner, T.E. Baker, E.M. Stoudenmire, K. Burke, S.R. White, Phys. Rev. B 90 (2014) 45109.
[50] S.P. Ong, S. Cholia, A. Jain, M. Brafman, D. Gunter, G. Ceder, K.A. Persson, Comput. Mater. Sci. 97 (2015) 209.
[51] https://github.com/materials-data-facility/forge.
[52] K. Mathew, J.H. Montoya, A. Faghaninia, S. Dwarakanath, M. Aykol, H. Tang, I. Chu, T. Smidt, B. Bocklund, M. Horton, J. Dagdelen, B. Wood, Z.-K. Liu, J. Neaton, S.P. Ong, K. Persson, A. Jain, Comput. Mater. Sci. 139 (2017) 140.
[53] M. De Jong, W. Chen, T. Angsten, A. Jain, R. Notestine, A. Gamst, M. Sluiter, C.K. Ande, S. Van Der Zwaag, J.J. Plata, C. Toher, S. Curtarolo, G. Ceder, K. a Persson, M. Asta, Sci. Data (2015) 1.
[54] M. de Jong, W. Chen, H. Geerlings, M. Asta, K.A. Persson, Sci. Data 2 (2015) 150053.
[55] I. Petousis, W. Chen, G. Hautier, T. Graf, T.D. Schladt, K.A. Persson, F.B. Prinz, Phys. Rev. B 93 (2016) 115151.
[56] F. Faber, A. Lindmaa, O.A. von Lilienfeld, R. Armiento, Int. J. Quantum Chem. 115 (2015) 1094.
[57] T. Fast, S.R. Kalidindi, Acta Mater. 59 (2011) 4595.
[58] K.T. Schütt, H. Glawe, F. Brockherde, A. Sanna, K.R. Müller, E.K.U. Gross, Phys. Rev. B 89 (2014) 205118.
[59] A. Seko, A. Takahashi, I. Tanaka, Phys. Rev. B 90 (2014) 24101.
[60] O. Isayev, C. Oses, C. Toher, E. Gossett, S. Curtarolo, A. Tropsha, Nat. Commun. 8 (2017) 15679.
[61] K.T. Schütt, F. Arbabzadah, S. Chmiela, K.R. Mu, A. Tkatchenko, Nat. Commun. 8 (2017) 13890.
[62] L.M. Ghiringhelli, J. Vybiral, S.V. Levchenko, C. Draxl, M. Scheffler, Phys. Rev. Lett. 114 (2015) 105503.
[63] A. Meurer, C.P. Smith, M. Paprocki, O. Čertík, S.B. Kirpichev, M. Rocklin, Am. Kumar, S. Ivanov, J.K. Moore, S. Singh, T. Rathnayake, S. Vig, B.E. Granger, R.P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M.J. Curry, A.R. Terrel, Š. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, A. Scopatz, PeerJ Comput. Sci. 3 (2017) e103.
[64] S. Kotochigova, Z.H. Levine, E.L. Shirley, M.D. Stiles, C.W. Clark, Phys. Rev. A 55 (1997) 191.
[65] M.A. Butler, J. Electrochem. Soc. 125 (1978) 228.
[66] A.M. Deml, R.O. Hayre, C. Wolverton, V. Stevanovic, Phys. Rev. B 93 (2016) 85142.
[67] C. Kittel, Introduction to Solid State Physics, 8th ed., Wiley, 2005.
[68] F.R. de Boer, Cohesion in Metals: Transition Metal Alloys, North-Holland, Amsterdam, 1988.
[69] R.F. Zhang, S.H. Zhang, Z.J. He, J. Jing, S.H. Sheng, Comput. Phys. Commun. 209 (2016) 58.
[70] L.J. Gallego, J.A. Somoza, J.A. Alonso, J. Phys. Condens. Matter 2 (1990) 6245.
[71] K. Hansen, F. Biegler, R. Ramakrishnan, W. Pronobis, O.A. Von Lilienfeld, K.-R.R. Müller, A. Tkatchenko, J. Phys. Chem. Lett. 6 (2015) 2326.
[72] E.L. Willighagen, R. Wehrens, P. Verwer, R. de Gelder, L.M.C. Buydens, Acta Crystallogr. Sect. B Struct. Sci. 61 (2005) 29.
[73] P.P. Ewald, Ann. Phys. 369 (1921) 253.
[74] N.E.R. Zimmermann, M.K. Horton, A. Jain, M. Haranczyk, Front. Mater. 4 (2017) 1.
[75] T. Lam Pham, H. Kino, K. Terakura, T. Miyake, K. Tsuda, I. Takigawa, H. Chi Dam, Sci. Technol. Adv. Mater. 18 (2017) 756.
[76] A. Schleife, F. Fuchs, C. Rödl, J. Furthmüller, F. Bechstedt, Appl. Phys. Lett. 94 (2009) 12104.
[77] V. Botu, R. Ramprasad, Phys. Rev. B 92 (2015) 94306.
[78] D. Waroquiers, X. Gonze, G.-M. Rignanese, C. Welker-Nieuwoudt, F. Rosowski, M. Göbel, S. Schenk, P. Degelmann, R. André, R. Glaum, G. Hautier, Chem. Mater. 29 (2017) 8346.
[79] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tesselations, 2009.
[80] J. Behler, J. Chem. Phys. 134 (2011) 74106.
[81] J.D. Hunter, Comput. Sci. Eng. 9 (2007) 90.
[82] M. Waskom, O. Botvinnik, D. O'Kane, P. Hobson, S. Lukauskas, D.C. Gemperline, T. Augspurger, Y. Halchenko, J.B. Cole, J. Warmenhoven, J. de Ruiter, C. Pye, S. Hoyer, J. Vanderplas, S. Villalba, G. Kunter, E. Quintero, P. Bachant, M. Martin, K. Meyer, A. Miles, Y. Ram, T. Yarkoni, M.L. Williams, C. Evans, C. Fitzgerald, Brian, C. Fonnesbeck, A. Lee, A. Qalieh, 2017. doi: 10.5281/ZENODO.883859.
[83] https://plot.ly/.
[84] J.M. Rickman, Npj Comput. Mater. 4 (2018) 5.
[85] M.W. Gaultois, T.D. Sparks, C.K.H. Borg, R. Seshadri, W.D. Bonificio, D.R. Clarke, Chem. Mater. 25 (2013) 2911.
[86] J.P. Perdew, M. Levy, Phys. Rev. Lett. 51 (1983) 1884.
[87] L.J. Sham, M. Schlüter, Phys. Rev. Lett. 51 (1983) 1888.
[88] M.K.Y. Chan, G. Ceder, Phys. Rev. Lett. 105 (2010) 196403.
[89] F.A. Faber, A.S. Christensen, B. Huang, O.A. von Lilienfeld, J. Chem. Phys. 148 (2018) 241717.
[90] K.J. Laws, D.B. Miracle, M. Ferry, A predictive structural model for bulk metallic glasses, Nat. Commun. 6 (2015) 8123.
[91] X. Yang, Y. Zhang, Prediction of high-entropy stabilized solid-solution in multi-component alloys, Mater. Chem. Phys. 132 (2012) 233–238.
[92] S. Kirklin, J.E. Saal, B. Meredig, A. Thompson, J.W. Doak, M. Aykol, et al., The Open Quantum Materials Database (OQMD): assessing the accuracy of DFT formation energies, Npj Comput. Mater. 1 (2015) 15010, http://dx.doi.org/10.1038/npjcompumats.2015.10.