

A User's Guide to Machine Learning for Polymeric Biomaterials

Travis A. Meyer, Cesar Ramirez, Matthew J. Tamasi, and Adam J. Gormley*

Cite This: *ACS Polym. Au* 2023, 3, 141–157

Read Online

ACCESS |



Metrics & More



Article Recommendations

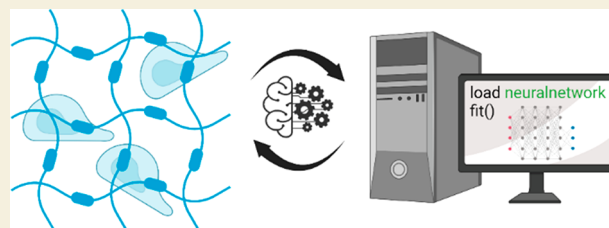


Supporting Information

ABSTRACT: The development of novel biomaterials is a challenging process, complicated by a design space with high dimensionality. Requirements for performance in the complex biological environment lead to difficult *a priori* rational design choices and time-consuming empirical trial-and-error experimentation. Modern data science practices, especially artificial intelligence (AI)/machine learning (ML), offer the promise to help accelerate the identification and testing of next-generation biomaterials. However, it can be a daunting task for biomaterial scientists unfamiliar with modern ML techniques

to begin incorporating these useful tools into their development pipeline. This Perspective lays the foundation for a basic understanding of ML while providing a step-by-step guide to new users on how to begin implementing these techniques. A tutorial Python script has been developed walking users through the application of an ML pipeline using data from a real biomaterial design challenge based on group's research. This tutorial provides an opportunity for readers to see and experiment with ML and its syntax in Python. The Google Colab notebook can be easily accessed and copied from the following URL: www.gormleylab.com/MLcolab

KEYWORDS: Biomaterials, Machine Learning, High throughput, Tutorial, Education and training



1. INTRODUCTION

Biomaterials development is a challenging enterprise—not only does the material need to be engineered to achieve the desired functionality, but it also needs to safely interface with the complex environment of human physiology. This is especially true as the field moves toward the development of “smart” or personalized biomaterials. As such, the traditional empirical approach to biomaterial product development including rational design based on prior knowledge/intuition followed by iterative, trial-and-error testing and redesign leads to a long development cycle for biomaterial innovation.¹

In recent decades, the field of data science has received an explosion of interest as a means for efficiently and effectively drawing conclusions from large amounts of data. This has been partially stimulated by the continued development of computationally efficient artificial intelligence (AI)/machine learning (ML) tools. AI and ML are often used interchangeably since ML is a subset of the broader category of AI. In this tutorial, we will refer to ML which is the use of algorithms that allow computer programs to iterate and evolve to accomplish certain tasks without being explicitly programmed. This is similar to how humans learn via observation and trial/error, but on an accelerated scale. The use of ML tools is particularly useful when data sets are too large, complex, and/or time-consuming to be practically understood by humans.² While ML has seen great success in computer science (computer vision, natural language processing, etc.) it is being increasingly adopted in other STEM fields, particularly in the fields of biology/bioinformatics³ and chemistry/materials science.⁴

While the implementation of ML in biomaterials development has been relatively limited, there are several examples in the literature, which have been extensively covered by a number of excellent reviews.^{1,5–7} Here, we briefly highlight several studies we believe demonstrate the breadth of ML usage in biomaterials. Our group has recently demonstrated the use of ML techniques to accelerate the development of novel polymer-protein hybrids capable of allowing proteins to maintain activity in harsh environments.^{8,9} For other polymeric materials, ML methods have been developed to predict whether peptide-functionalized building blocks could form hydrogels,¹⁰ to model the adhesion of cells and proteins on engineered surfaces,^{11,12} to classify the immunomodulatory behavior of synthetic copolymers,¹³ or to predict the physicochemical properties of polyurethanes.¹⁴ ML models have also been developed for inorganic biomaterials, including models to optimize the mechanical properties of metal alloys for biomedical implants^{15–17} and predict fracture behavior in ceramic materials.¹⁸ Additionally, ML techniques have been utilized for the development of nanomaterials, including the prediction of nanoparticle fate *in vivo*,¹⁹ the formation of the protein corona,²⁰ and immunomodulatory activity.²¹

Received: August 5, 2022

Revised: October 27, 2022

Accepted: October 27, 2022

Published: November 17, 2022



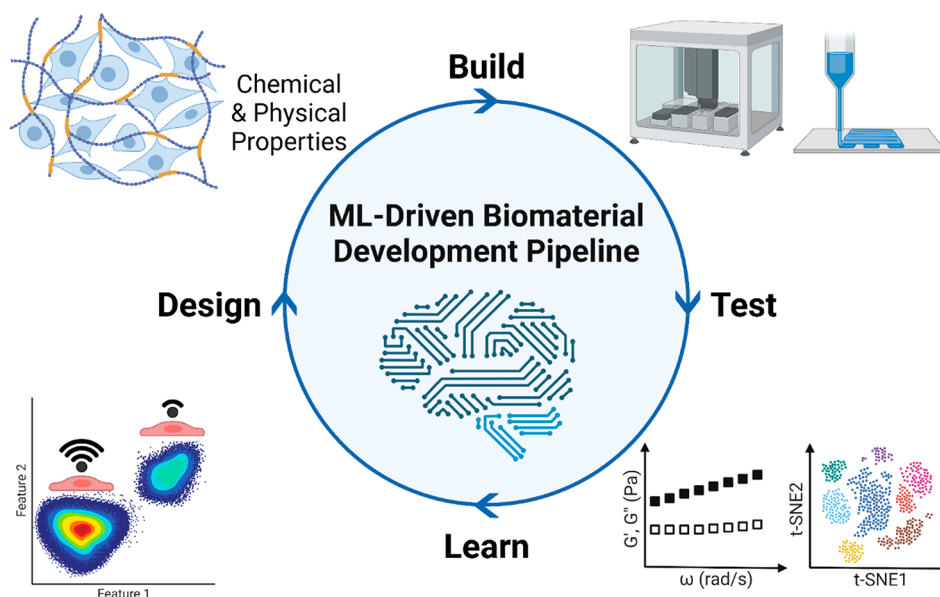


Figure 1. Schematic of an example Design-Build-Test-Learn paradigm for biomaterials development. Materials are initially designed with certain chemical and physical properties based on either rational design or a comprehensive survey of material features. Materials are then built and tested for desirable characteristics, ideally through high-throughput laboratory automation. This data, alongside design parameters, are then fed into a machine learning (ML) pipeline, where key patterns are extracted to create predictive models. These models can then be used to help design new material generations with targeted functionality. This figure was created with BioRender.

The success of these studies in utilizing ML techniques to facilitate and/or accelerate biomaterials innovation highlights how transformative the adoption of ML can be in the biomaterials field. The combinatorial complexity of most biomaterials means that material properties exhibit high dimensionality and thus rational design is a challenging and time-consuming process. To mitigate this, high-throughput combinatorial synthesis techniques have been developed to screen libraries of materials and thus generate data more rapidly,^{22,23} while enhancements in molecular modeling techniques enable generation of predicted properties and functions. ML is an efficient method of digesting large and complex data sets to extract patterns, identify key drivers of functionality, and make predictions on the behavior of future iterations.²⁴ ML algorithms are also helpful as a means of generating predictions from theoretical data (i.e., molecular dynamic simulations) in an effort to guide the synthesis and experimentation of materials in a more cost- and time-efficient manner.

We believe that the future of biomaterials development resides in the use of a Design-Build-Test-Learn paradigm, in which high-throughput material synthesis and characterization is paired with ML to accelerate the data-driven design of novel biomaterials with advanced functionality (Figure 1). Therefore, the goal of this tutorial is to provide an introductory “how-to” guide on using ML algorithms for biomaterial design, with the hope that this provides a solid foundation for other scientists to begin adapting and utilizing these tools for their own work. ML can be initially intimidating, so we hope to provide the community a concise, easy-to-understand introduction to the vocabulary and topics. In Section 2, we cover some general considerations for the collection of data necessary for training and evaluating ML models. In Section 3, we provide an introductory overview of ML implementation, focusing on the process of setup, implementation, validation, and analysis. Finally, in Section 4, we provide a hands-on tutorial utilizing data

from our lab to demonstrate how the topics covered are used in practice. All of the code, along with the example data set, can be found at www.gormleylab.com/MLcolab.

2. DATA COLLECTION

The first step in any study which seeks to use ML algorithms to model or predict material properties or functionality is the curation of data. In this section, we briefly cover considerations for data compilation/collection, quantity, mined vs. collected, and processing.

2.1. How Much Data Do I Need?

A common concern among researchers interested in pursuing ML-based studies is the large quantities of data that are seemingly needed to develop robust, accurate models. While this is certainly true for some algorithms and applications, successful implementation of ML can be done with much smaller data sets. For example, previous modeling studies have shown that ML processes can be successfully used on data sets including as small as 50 to several hundred polymers.^{25,26} A recent publication utilizing ML to design random copolymers for RNA transfection demonstrated successful model development with an initial library of 43 polymers.²⁷ Other work has shown that data sets on the order of ~100 observations had comparable predictive capacity to larger libraries of ~1000 observations, demonstrating the diminishing returns that can be observed with large data sets.²¹ In general, the choice of which ML algorithm to use is heavily influenced by data quantity where some high-performing models work well with smaller data sets. The choice of model as it pertains to the size of the data will be discussed more in Section 3.

There are other techniques and strategies which have been developed to facilitate effective modeling from sparse data. For example, one-shot learning, originally designed for computer vision problems,²⁸ has been demonstrated with small training data sets in the area of drug screening and could be useful for biomaterial-based projects.²⁹ In one-shot learning approaches,

model learning is accelerated by adding encoded chemical or physical properties which can then be used to inform similarity between known materials and new materials of interest. By including such similarity metrics, one-shot learning has been shown to be an effective method of informing models about complex behavior of molecules in which much larger data sets would normally be required to achieve.³⁰ Similarly, transfer learning (utilizing previously trained models as a starting point for a new task), latin hypercube sampling (a statistical method for generating random numbers that more closely represent the full distribution of existing data), or Bayesian inference (statistical technique in which *a priori* knowledge can be used to supplement data, allowing the development of models with minimal training data) are also strategies that can be used in cases where training data sets are small in order to improve model performance.

One ML paradigm that could be particularly relevant to biomaterials development is active learning. In active learning, ensemble or statistical ML methods return uncertainty values alongside predictions to map parameter spaces with high uncertainty. This information, along with techniques like Bayesian optimization, can then be used to help initialize new experiments with small, focused data sets that target regions of feature space that would be most fruitful for exploration.³¹ This explore vs exploit approach to active learning provides a balanced approach to simultaneously exploit known information, while exploring new feature spaces with high uncertainty. Our research group has utilized this strategy for the design of polymer–protein hybrids and demonstrated the superior efficiency and efficacy of ML-directed active learning data collection compared to large library screens.⁹

2.2. Mined vs Collected Data

Another consideration is whether to use mined or collected data. Mining refers to the practice of curating data by pulling information from previously published/developed studies, typically in the form of an Internet database. Generating data in-house allows precise control over the conditions under which materials are built and tested and is necessary for evaluating completely novel materials. However, mined data from databases is much more cost and time effective if good repositories are available. The use of mined data is common in fields where the presence of curated, large data sets are publicly available—this is especially true in the bioinformatics fields, where big data–omics studies are routinely available for others to access. Unfortunately, this kind of curation is rare in the biomaterials space; while comprehensive high-throughput combinatorial biomaterial studies have been undertaken over the past three decades, there does not exist a centralized repository for this data nor consistent methodology for its collection and representation.

There is recent emphasis on the use of data science in combination with high-throughput modular studies for biomaterialomics.¹ However, it is likely that the fruits of these initiatives will not be readily available in the near future. While limited relative to other areas of research, comprehensive material databases do exist such as AFLOW library, ICSD, Open Quantum Database, Materials Projects, and Citrine Informatic.⁶ Clinical trial data, some of which comes from studies of biomaterial implants, can also be sourced from several different areas, such as Vivli or YODA, to provide data for ML-driven analysis. Polymer Genome is a web-based ML pipeline for predicting polymer properties, which could be useful for

researchers looking to quickly and efficiently generate *in silico* polymer data sets.³² Additionally, work focused on using text mining tools to extract information from published biomaterial studies offers an exciting opportunity for curating data sets for machine learning moving forward.³³

If you are investigating a problem or area in which existing data sets might prove useful, it is important to keep in mind that data preprocessing techniques to clean and validate the data are especially critical. Investigation of the data set prior to training should include identifying and addressing missing or NaN (not a number) values as well eliminating observations which include spurious or obviously incorrect data. High level data set investigation methods, such as `.describe()` and `.info()` in Python, can be helpful tools for identifying outliers. An example of this kind of validation and cleaning is included in the accompanying tutorial (Section 4). It is good practice during preprocessing to maintain detailed records of every modification that was made to the original data set, which helps to ensure reproducibility for studies using ML.³⁴ Keep in mind, a major downside of using literature data is conditions for experimentation in biomaterials research are not standardized, and thus pooling observations across multiple studies can add unaccounted for variation into the data set which can obscure key signals within the data.⁶

2.3. Biomaterial Synthesis and Characterization

For most biomaterial researchers, the data used for ML will be collected in-house. Seed data experiments for initial model training benefit from the Design of Experiments (DOE) approach, which provides a framework for sampling independent variables from a range of possible options to maximize the information learned.³⁵ Starting with a DOE-inspired pilot material set can help facilitate successful ML implementation with smaller data sets.

It is important to consider how the throughput and scale of synthesis and characterization will impact a planned design campaign. High-throughput material synthesis is one area which has seen considerable growth in recent decades, and thus several well-established techniques exist for generating high quality biomaterial libraries for screening. There have been several excellent reviews with extensive coverage on different experimental techniques for both synthesis and characterization.^{22,23,36} Here, we only provide a brief overview of various techniques.

For the high-throughput synthesis of polymeric materials, photoinduced free radical polymerization of polymeric microarrays has been extensively used to generate large combinatorial libraries.^{37,38} Oxygen-tolerant living polymerization techniques, such as Enz-RAFT and PET-RAFT, are recent improvements in this area which enable the synthesis of well-controlled polymer architectures in well-plate formats that are ideally suited for high-throughput synthesis and ML investigation.^{39–46} For nano/microparticle development, layer-by-layer processes offer an ability to easily synthesize combinatorial libraries of particles with different characteristics based on the core/coating combinations used during synthesis.⁴⁷ Combinatorial libraries of hydrogels can be synthesized in which material properties are varied through the use of 3D molds,⁴⁸ or chemical properties can be varied through covalent modification of components such as alginate.⁴⁹ The introduction of gradients into a continuous material is another efficient means of screening material properties, and synthesis methods that allow for variations in stiffness,⁵⁰ topography,⁵¹ and biochemical properties^{52–55} have been developed.

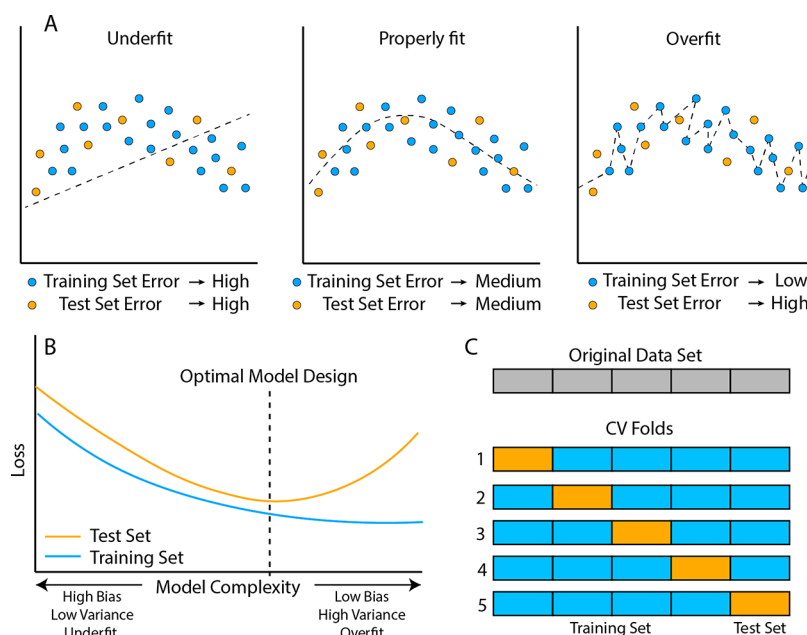


Figure 2. (A) Two-dimensional data set, split into training (blue) and test (orange) subsets, with the modeled function (dashed line) shown. In the case of model underfitting (left), the modeled function does a poor job capturing the variance of the underlying data, leading to high errors for both subsets. Model overfitting (right) leads to a function that perfectly fits the underlying training set profile, leading to low training set error but high test set error. A properly fit function (middle), appropriately captures the underlying pattern in the data and is equally generalizable to new data in the test set. (B) Bias/variance trade off showing loss function error for both the training set (blue) and test set (orange) as a function of model complexity. A model with low complexity has high bias and does not accurately capture data patterns, leading to an underfit model. As the complexity increases, loss function error decreases for both training and test data sets. Eventually, a point is reached where the model begins to overfit the training data, leading to a rise in test set error and an increase in model variance. (C) Example of 5-fold cross validation, where data set is split in five bins containing equal proportions of data and training/validation is completed five times, using different subsets for training (blue) and test (orange) data. Loss can then be compared across all five iterations to get an average loss for the entire data set.

After the materials of interest have been synthesized, their properties and functionality must then be characterized to build the data set for model training. Techniques for measuring the mechanical properties of biomaterials in a fashion suitable to high-throughput processing have been previously developed, such as spherical nanoindentation.⁵⁶ For polymeric materials, experimental methods such as small-angle X-ray scattering (SAXS) or dynamic light scattering (DLS) are available in high-throughput for characterizing libraries.⁵⁷ Biocompatibility studies for cytotoxicity, immunogenicity, and blood compatibility can also be included in a high-throughput experimental pipeline. This can include data on the gene expression profile of cells cultured with different biomaterials.⁵⁸ High-content imaging (HCI) is a microscopy technique for monitoring cell-material interactions in a high-throughput manner, allowing the rapid testing of materials.⁵⁹ Microfluidic devices can be used to monitor the response of cell-material interactions on things like chemical gradients,⁶⁰ mechanical stress,⁶¹ or in environments which more closely mimic *in vivo* conditions.⁶² Finally, the process of barcoding materials using DNA-labels can be used to label materials for tracking in complex/pooled environments,⁶³ while barcoded heterogeneous cell mixtures can be used to generate large data sets regarding material-biology interactions with fewer materials but larger libraries of cell types/lineages.⁶⁴

Computational simulations can also be used to characterize materials and generate features for ML algorithms in a high-throughput, time-efficient manner. Several techniques for the theoretical modeling of polymers exist which can be leveraged for data collection.⁶⁵ The development of coarse-grained models used in conjunction with techniques such as Monte Carlo or Molecular Dynamics simulations allows for the efficient

theoretical analysis of complex materials at spatiotemporal scales inaccessible by traditional atomistic computational processes.^{66,67} Some examples of computationally focused biomaterial studies include the design of injectable biomaterial scaffolds,⁶⁸ the characterization of bulk mechanical and surface properties of common polymeric materials,⁶⁹ and the analysis of pH-responsive protein adsorption to polymeric hydrogels.⁷⁰ Additionally, a computational framework known as cellular automata can be used to model phenomenon such as material degradation and drug release.⁷¹

Nonexperimental data, such as material/polymer composition, can be used as inputs for ML algorithms. Polymeric descriptors taken from existing databases or other software can be used to generate additional features that do not necessitate experimental determination.⁵ Additionally, techniques such as composition-based feature vectors (CBFV),⁷² in which categorical values of materials, such as element/monomer composition, are encoded in a manner that facilitates ML-usage. One-hot encoding, a method of converting categorical data into numerical values that do not carry inherent ordinal assumptions, is another common method for converting chemical compositions into features that perform well in ML algorithms. Further discussion of feature encoding and data representation is included in Section 3.3.

Finally, it can be helpful to consider the final presentation of data prior to collection, as this can save time in the future. All data collected for ML-based biomaterials work should be deposited into an online repository. Often, universities have data storage infrastructure in place already—in our group's recently published work,⁹ the data was uploaded to a Princeton University repository.⁷³ This ensures transparency and

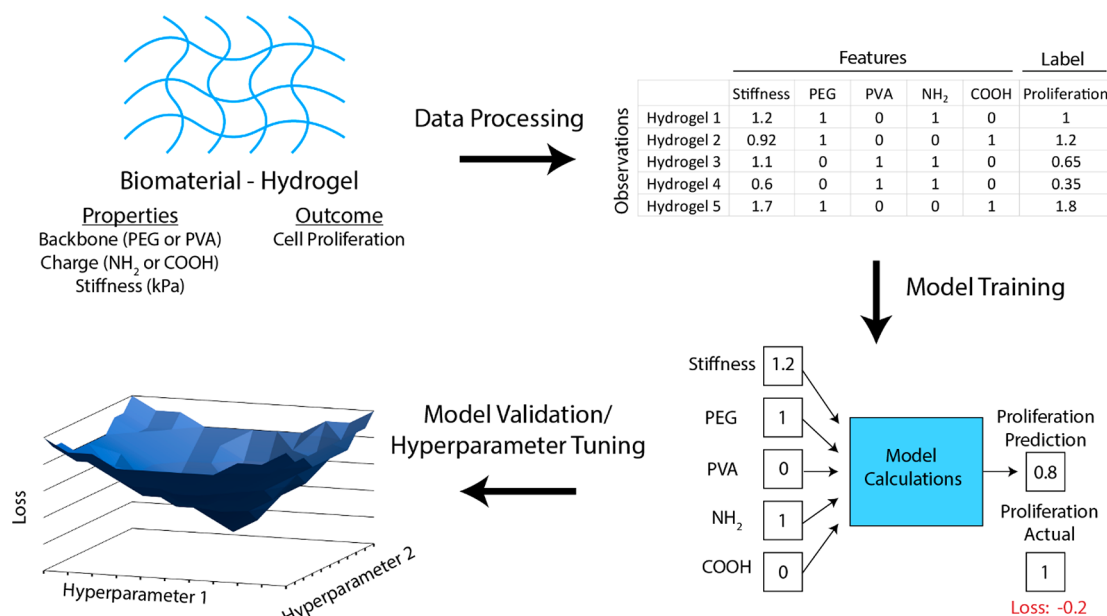


Figure 3. General ML pipeline. The first step involves the conversion of material features and labels into tabular data where categorical data has been encoded and numerical data has been scaled and normalized. In the second step, the model is trained such that the features are fed into the model and a predicted label is compared to the actual label to generate a loss term. Training iterations focus on modifying model parameters to minimize loss across the training data set. Finally, the model is validated against a test data set and model hyperparameters can be systematically tuned to minimize test set loss and find the optimal model design.

reproducibility for the published work, while also serving as a means for future researchers to potentially mine and utilize this data for other data science-focused investigations. When compiling these data stores, FAIR (findable, accessible, interoperable, and reusable) guidelines should be followed.⁷⁴

3. ML FOR BIOMATERIALS DEVELOPMENT

After data has been collected and curated, the next step is to select and use a ML algorithm. Before discussing the specific types of ML models as well as the steps necessary to effectively utilize that model, we first start with definitions and a general overview of the ML process. An alphabetized list of these definitions, plus others, is included in the [Supporting Information](#). Two types of ML are most commonly used in the biological/material fields; **supervised learning**, where the data used to train the program contains labels that the program uses to judge performance and to predict on new data; and **unsupervised learning**, where unlabeled data is used and the task is to find patterns/groups within new data sets so that the data can be clustered or simplified via dimensionality reduction. Of these, supervised learning tasks are the most commonly used in biomaterials research, as labels/objectives are typically known for an initial data set. Supervised learning can be further subdivided based on the kind of labels that the program seeks to predict. If these labels are defined classes (i.e., yes/no cell adhesion, pro/anti-inflammatory, etc.), then the task is **classification** and the algorithm is known as a **classifier**. If the labels take the form of a continuous variable (i.e., degree of cytotoxicity, % cell adhesion, etc.), then the task is **regression** and the algorithm is known as a **regressor**. Often the same ML models can be used for both classification and regression, with only minor changes to their implementation. **Features** are the predictive data that are fed into a ML algorithm, which the program then uses to arrive at a prediction or grouping; these are analogous to independent variables. **Labels** are the values or

classes that one seeks to predict for new data during supervised learning; these are analogous to dependent variables.

During the process of ML, the program takes the feature values and manipulates them in some way using preset **parameters**. At the conclusion of one iteration, the program thus has a predicted label (for supervised learning) or grouping/pattern (for unsupervised learning). The quality of the output is then assessed, typically through the use of a **loss function**, also sometimes referred to as a **cost function**. There are numerous kinds of loss functions that can be used, though common ones are mean squared error for regressors and binary cross entropy for classifiers. Based on the results of this loss function, the algorithm proceeds through the training data set again, updating the parameters to minimize the loss function (or error of the model). This process repeats until the loss function has reached a minimum. This is known as the **training process**. **Hyperparameters** are values that govern how the algorithm accomplishes this goal; these values are chosen by the modeler during model initiation and unchanged by the program during the training process. Typically, a range of hyperparameter values are sampled during modeling to tune or alter the performance of a given algorithm for a given task.

Poor model performance can typically be classified into one of two types: **underfitting** or **overfitting** (Figure 2A). When a model is underfit, it does a poor job capturing the reasons for variance within the data and thus the predicted labels or groupings are often highly divergent from the actual values (i.e., large loss function). In comparison, an overfit model is one in which the variations within the training data set are modeled too well, to the point that noise within the data is treated as signal and mapped efficiently. While overfit models perform very well in regards to loss function minimization on the training data, they are not generalizable and thus perform poorly when new data is presented. There is typically a trade off in model complexity as it relates to underfitting and overfitting. In general, less complex models are more likely to underfit the data, while

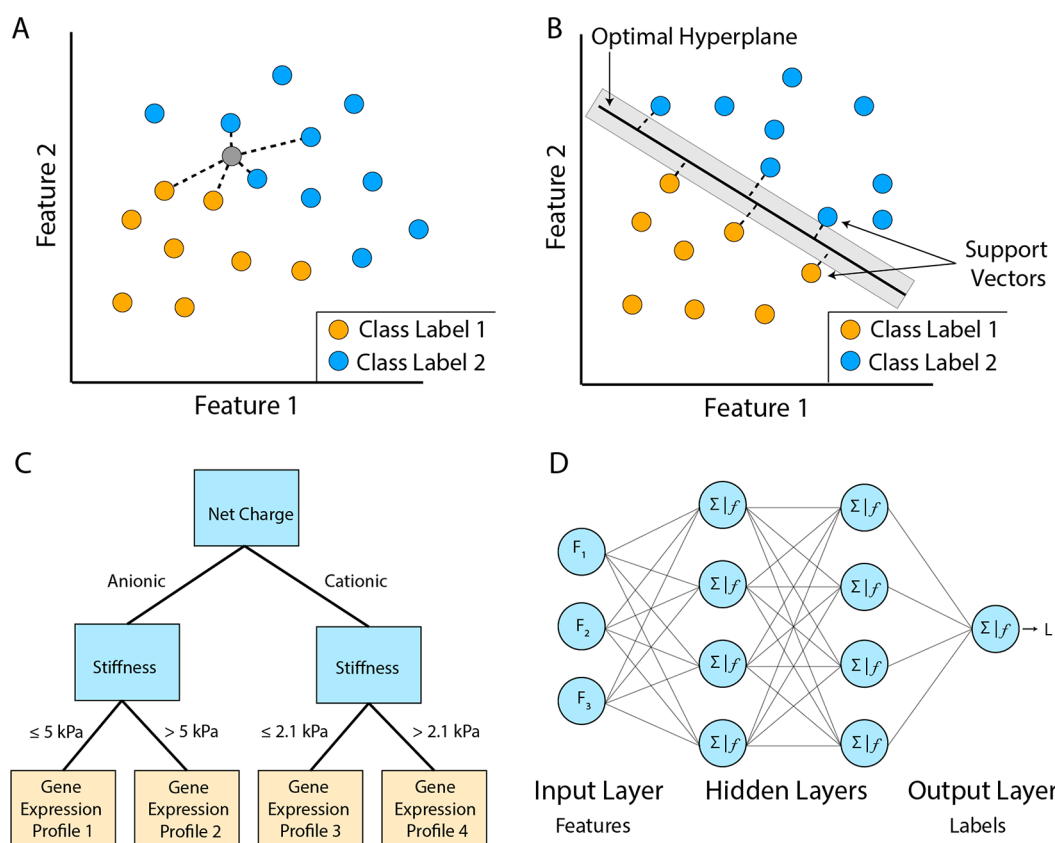


Figure 4. (A) K-nearest neighbors (KNN) classifier with unlabeled data point (gray) assigned a label (blue or orange) based on voting by the $k = 4$ closest labeled data points in feature space. (B) Two-dimensional support vector machine (SVM) classifier where optimal hyperplane separating classes is chosen by maximizing the size of the margin (gray box) based on support vector data points. (C) Hypothetical decision tree used to classify gene expression profiles based on hydrogel characteristics. Features are used as decision points at each node (blue), and observations are sorted into child nodes based on feature values. Eventually, leafs (orange) are reached which contain no further subdivisions, and labels are assigned based on leaf composition. (D) Artificial neural network consisting of an input layer, two hidden layers, and an output layer. Features (F_1 – F_3) are fed into the input layer, where they are subsequently sent to hidden layers for dot-product summation (Σ) and activation function processing (f). This process repeats until the final predicted label is calculated in the output layer.

more complex models can lead to overfitting. This is also known as the **bias-variance** trade off (Figure 2B). Properly designed models attempt to maximize the ability to capture complex relationships within the data (low bias), while still maintain generalizability to unseen data (low variance). The process of **validation/testing** is done after training in order to assess the degree to which a model suffers from underfitting and overfitting. A common technique for validation is known as **cross-validation** (Figure 2C) and will be discussed further in Section 3.5.

The general process for a ML pipeline is described below and illustrated in Figure 3. For novice users, the first choice to be made will be which programming language to use (Section 3.1). After the specific type of ML algorithm is chosen (Section 3.2), the data must then be processed to be machine interpretable (Section 3.3). The model is then initiated and trained (Section 3.4). Validation is then used to assess model performance (Section 3.5), and hyperparameters can be tuned to optimize model behavior (Section 3.6). Often ensemble methods are used to generate more robust models as well as to estimate model uncertainty (Section 3.7). Finally, the model can be used to generate predictions on new data, or analyzed to provide insight into the fundamental mechanisms behind the modeled process (Section 3.8).

Finally, a number of other excellent tutorials/perspectives have been written regarding the implementation of ML in biology² and materials science,³⁴ which we highly recommend readers use for additional reference.

3.1. Programming ML Models

At their core, all ML models are mathematical algorithms attempting to replicate functions that describe the underlying data. As such, ML models can be built in most modern computer languages. However, the explosion in ML interest in recent years has led to the development of prebuilt libraries for running ML in a number of common programming languages. These include scikit-learn,⁷⁵ Keras,⁷⁶ and PyTorch⁷⁷ in Python, as well as caret⁷⁸ in R and MLJ⁷⁹ in Julia. These libraries greatly simplify the implementation of ML by providing prebuilt functions for training, validation, and analysis of many ML algorithms. For those just getting started, we highly recommend the use of scikit-learn and Keras within Python. Python is one of the more common and generalizable programming languages in use today, and numerous free online tutorials exist to help with getting started. Scikit-learn and Keras are popular ML toolkit libraries with excellent documentation and resources. Open-source cloud computing services like Google Colab, which runs natively on Python, are excellent for programming, running, and sharing small ML models, and is another reason to consider using Python. The tutorial section of this paper (Section 4) will

utilize the scikit-learn package within Google Colab. A wide variety of online tools exist to facilitate learning both Python in general and ML specifically. Our group has utilized DataCamp (www.datacamp.com) and highly recommends it, though other free/open-source resources also exist, including LearnPython (www.learnpython.org) and CodeCademy (www.codecademy.com), among others. Online course repositories such as Udemy (www.udemy.com) and Coursera (www.coursera.org) also have units on Python and ML.

3.2. ML Algorithms

Several different ML algorithms exist whose choice depends on a number of considerations such as the type of task to be performed, the type of data to be used, the size of the training set, computational cost, etc. All models have an inductive bias that favors them toward certain types of solutions based on either the mathematical techniques used or the type of loss function that is optimized. As such it is important to consider how the characteristics of the data/experimentation matches the underlying mathematical process of the model used, known as inductive bias.² In this section, we will briefly summarize a few of the more popular algorithms in use, which will hopefully provide a good foundation for choosing how to build your first model. However, the best course of action is to search the literature for previous models which try to address a similar problem or complete a similar task and use that as a starting point. It is also recommended that different classes of algorithms be used to develop models for the same task with the same training data, such that the results can be compared to allow for selection of the most appropriate/best performing model.

Dimensionality Reduction is an unsupervised learning task that can be accomplished by several different algorithms, most notably principal component analysis (PCA). PCA clusters data by reducing the number of features/dimensions taken into account while maintaining the relationships between data points as much as possible.² Ideally, this involves a reduction to either two or three dimensions such that the clustering of data can be visualized more readily. Dimensionality reduction algorithms can be used on data with both linear and nonlinear relationships and can be a helpful step to engineer features prior to the introduction of other supervised ML models. Other algorithms for dimensionality reduction include t-distributed stochastic neighbor embedding (t-SNE),⁸⁰ uniform manifold approximation and projection (UMAP),⁸¹ and diffusion maps.⁸²

Logistic Regression is a simple, linear algorithm which performs well for binary classification tasks, in which the coefficients of a linear combination of features are optimized to predict the probability of a certain event (or label) occurring.⁸³ **Linear Regression** models can also be developed using ML processes for regression-based tasks. In practice, most linear regression models developed using ML strategies implement a regularization process in which overfitting is minimized through the restriction of the values of coefficients optimized during training. Common regularization strategies include **Ridge Regression**,⁸⁴ which is useful when features are highly correlated, and **Lasso Regression**,⁸⁵ which acts to reduce or remove the effect of features which have lower predictive power. These techniques, while simple, are quick and powerful tools for modeling linear relationships, and are frequently good starting points for initial model development.

K-Nearest Neighbors (KNN) is a simple, robust supervised learning algorithm well-suited for classification problems, in which the identity of new, unknown observations is established

via the distance in the feature space from labeled data (Figure 4A).⁸⁶ The hyperparameter *k* dictates the number of closest data points which contribute to the prediction of the new observation and can be varied to optimize model performance. While primarily used for classification tasks, KNN can be used for regression tasks, in which the assigned label is the weighted average of the labels of the closest neighbors.

Support Vector Machines (SVM) are a class of algorithms which are some of the more robust and heavily used within the ML field.⁸⁷ These models derive a mathematical equation, known as a hyperplane, that is used to separate clusters of data by focusing on observations that are closest to the decision barrier between classes (the support vectors) (Figure 4B). While this is inherently a linear-process, SVM uses a mathematical transformation to the data, known as a kernel function, to add additional dimensions, transforming nonlinear problems into linear problems which are easier to solve. Different kernel functions exist which allow for the ability to tailor SVMs to different kinds of nonlinear data. A good practice is to train both linear and nonlinear SVM models (a good option for initial nonlinear models is the radial basis function kernel) in order to quickly ascertain if the use of a nonlinear model will be beneficial. While SVMs have been traditionally used for classification problems, they can also be implemented for regression tasks where the optimal hyperplane is chosen to maximize data point inclusion and thus used as a line of best fit. SVM models are robust and highly customizable due to nature of kernel-function based implementation, in which different kernel functions can be used based on the nature of the data/task. Additionally, SVM works well with smaller training data sets and when there are a large number of features, which makes it an excellent choice for certain ML tasks. However, the computational complexity of these algorithms limits their suitability for larger data sets, and SVM models do not perform as well when the data set contains overlapping classes or imbalanced data.

Gaussian Process Model (GPM) is a type of supervised learning algorithm in which the optimal predicted function describing the data is modeled as a distribution of a number of possible functions that could all fit the data.⁸⁸ Gaussian process models can be used for both regression and classification tasks, though it is typically used for regression and are typically known as Gaussian process regressor or GPRs. These models make use of Bayesian statistics as a means of estimating uncertainty, and kernel functions are specified by the user as a means of establishing *a priori* that are then updated using the training data. A major advantage of GPMs is the inclusion of uncertainty/variance within every predicted value, which can be very helpful in terms of interpreting model outputs and identifying regions of the feature space where the model does not perform as well. However, these models are computationally expensive and thus perform best with smaller data sets.

Decision Trees, of which the most common versions are known as classification and regression trees (CARTs), are very popular supervised ML algorithms useful for a wide variety of tasks and data types.⁸⁹ These algorithms utilize a sequence of if/else questions about features in order to assign a label. Trees are made up of nodes, at which point a certain feature is interrogated; that feature of an observation is compared to a set value, and the observation is thus sorted into one of two paths to subsequent child nodes. This process repeats until a final decision node, known as a leaf, is reached (Figure 4C). During the training process, the nodes of a tree are grown recursively and the choice of feature/set value at each node is varied to

maximize the information gain at each decision point. The decision tree as a whole is optimized in order to maximize the homogeneity of labels, known as purity, of the leaves. Leaf purity can be assessed via various computational processes such as the gini-index⁹⁰ or mean-squared error. For classification tasks, the label assigned to a new observation is simply the label that is most prevalent in the leaf node, while regression tasks assign the average value of the observations within the leaf. Decision trees have a number of benefits that make them well suited for a variety of ML tasks: they are able to capture nonlinear relationships, they do not require features to be on the same scale, they are easy to understand and interpret, and they are capable of creating multiple rectangular decision regions within a parameter space. However, models built from decision trees are very sensitive to the training set data and are thus easily overfit, and the values they output for regression tasks are discontinuous. As such, ensemble methods such as **Random Forests** or **Gradient Boosted Trees** are almost always implemented—these will be discussed in Section 3.7.

The final ML algorithm to be summarized here are **Artificial Neural Networks (ANNs)**, often referred to as deep learning techniques. These models can be used for both classification and regression problems and have revolutionized the use of ML for certain tasks, particularly those with large data sets or where features are highly structured, as with images, text, or video.² ANNs are built from a number of nodes situated in layers where information from previous layers are fed into nodes of subsequent layers, processed, and then passed along (Figure 4D). Features from an observation start in the input layer, are modified during passages through hidden layers, and then emerge as a predicted label in the output layer. This architecture enables the model to take into account how features within an observation are related or correlated, with patterns within the data built into the hidden layers during the training process. Typically, the values of subsequent layers are generated by taking the dot-product of connections from nodes in the previous layer, followed by modulation via the use of an activation function, which serves to impart nonlinearity to the algorithm. The choice of activation function is one area in which models can be optimized or modulated for different tasks; some common activation functions include tanh, rectified linear activation (ReLU), and softmax, with the development of new activation functions an area of active research.⁹¹ The training process involves the generation of a predicted label based on initial random weightings within the network (forward propagation), the calculation of the loss function in relation to the actual labels, and then the updating of the weights based on a technique to find the loss function minimum known as gradient descent (backpropagation). This process occurs iteratively across the training set until the loss function has been minimized.

ANNs are especially powerful ML algorithms because they enable representation learning, in which case internal components of the model are built to represent patterns observed within the data. This minimizes the need for feature engineering or *a priori* intuition about patterns within the data set. For example, autoencoders are a type of ANN used for unsupervised learning tasks, which works by creating a compressed representation of the initial data (encoding) into what is known as a latent space representation; this can later be “extracted” to the original form (decoding).⁹² ANNs have other benefits, including the ability to capture complicated relationships between features, the ability to handle a wide variety of

data types (text, image, video, audio), and the ability to have asymmetric feature presentation within observations. However, ANNs traditionally require very large training data sets, which mitigates their usage in certain areas, and it can be challenging to natively interrogate feature importance or model logic, making most ANNs a “black-box” model. However, improving upon these limitations is an area of active research, and it is likely that new techniques to mitigate these concerns will soon be prevalent. Deep neural network implementation is not included in the Python package scikit-learn, but must be used with other libraries such as Keras or PyTorch.

There are several variations of neural networks that are worth mentioning. The standard ANN described above is also known as a multilayer perceptron. When multiple hidden layers are included within the model, this is called **deep learning**. One variation is **Convolutional Neural Networks (CNNs)**, which are models best suited for image-based data, where data possesses a local structure and recognition of that structure is the task.⁹³ CNNs utilize single-layer convolutional layers, which apply a “filter” to local groups of input features (i.e., smaller regions of an image). The filter is then passed across every region of the data to produce the output, and these convolutional layers begin to learn the local structure of the input data. A related variation are **Graph Convolutional Neural Networks**, which utilize a similar process but are best suited for data in which observations are connected by specific relationships or interactions.⁹⁴ In this case, interactions between observations are graphed as edges and nodes, and the structure of this graph guides the flow of information through the neural network. Finally, **Recurrent Neural Networks (RNNs)** are another variation that is best suited for data in which a direct connection exists between subsequent observations in an ordered sequence (i.e., time- or order-dependent data).⁹⁵ In this instance, the output of one observation is passed back into the network as an additional variable, known as the internal state update, to be used during the prediction of subsequent data points.

3.3. Data Processing

Most data sets, whether mined or generated in-house, need to be preprocessed before model training. For simple categorical data, techniques such as one-hot encoding can be used to make class labels machine-interpretable. However, it can be much more challenging to represent complex biomaterials and it is critical to consider how features and data should be represented before starting a new project. Doing so ensures that past, present, and future data can be equally used to continuously evolve models as new knowledge is gained. This is perhaps the most challenging aspect of using ML for biomaterials science. For example, how does one quantitatively represent a peptide-functionalized hydrogel that mimics a stem cell niche? Proteins, peptides, and nucleic acids are conveniently represented by the primary sequence making them attractive candidates for data-directed design. Further inspiration can be found in the small molecule community which routinely uses “Simplified Molecular Input Line Entry System” (SMILES) string notations that encode chemical structure thus providing a machine-interpretable format for representing molecules. BigSMILES is a modification on this technique for polymeric materials which uses normal SMILES syntax to represent polymeric fragments by a list of repeating units.⁹⁶ Composition-based feature vectors is another technique for converting material structure and properties into a machine-interpretable form.⁷² The Webb group at Princeton also published a comprehensive study of different polymer

featurization techniques which can prove useful for biomaterial modeling.⁹⁷

Quantitative, continuous variable data also needs to be manipulated for featurization. One common practice is the scaling of data, particularly when certain features have values which fall in drastically different ranges. Most ML algorithms take the distance between values in a feature as an important signal, and one feature containing values with a range of 0–3 will be treated drastically differently from a feature containing values with a range of 10 000–1 000 000. There are several different techniques for scaling features, including converting data to have a mean of 0 and variance of 1, or a minimum 0 and maximum of 1—the specific choice of scaling process will likely depend on the data/task. Regardless of which scaling strategy you choose, keep in mind that the values used for feature scaling (i.e., mean and standard deviation) must only come from the training data set in order to ensure that successful predictions are robust.³⁴ Normalization of data can also be helpful for regressions tasks—in all cases, scaling should be done first, followed by normalization.³⁴

Data augmentation is a preprocessing strategy that can be used to alter and enhance the training data set and thus improve the robustness of the subsequent model. This is especially common with the use of image-based data sets and convolutional neural networks: the addition of common image transformations such as rotation, zoom, and reflection to the training data set can be helpful in improving model performance. Augmentation can be used to improve training with other kinds of data sets as well—for example, libraries have been created to assist in the augmentation of chemical structure data sets for use with graphical neural networks.⁹⁸

Class imbalance is another example of an issue with data that can benefit from preprocessing. Imbalanced data is where more data exists for certain labels than others, which can hinder the ability for the model to correctly predict the minority label. In this instance, researchers can use resampling techniques to overcome class imbalance problems. These techniques include random oversampling, which simply involves resampling in-class data, or more complex strategies like adaptive synthetic sampling and synthetic minority oversampling, which create new synthetic data points based on linear combinations of in-class existing data.¹⁰ The dimensionality reduction techniques t-SNE and UMAP can be used for visualizing high-dimensionality data in ways that allows researchers to more easily identify class imbalance or clustering issues prior to starting ML analysis.

3.4. Model Initiation and Training

The first step after data processing and before model training is the establishment of training and test data subsets. Because all ML models are built based on the training data given to them, it is critical that models are evaluated and validated based on data that was not available during training. Often, this involves the fractionation of the data set into randomly distributed train and test subsets, with some predefined proportion of the data assigned to each group. Both scikit-learn and Keras have prebuilt methods for accomplishing this task (`train_test_split` and `validation_split`, respectively). It is important to make sure that not only is the data split done randomly, but that the data is properly stratified (i.e., average and standard deviation are same between subsets and class representation is roughly equivalent).⁵ This is also a point to be wary of data leakage, which is where connections exist between data in the training set and test set, such that validation overestimates the predictive capacity for

unrelated data.² This is typically more relevant for biological applications (ensuring no homologous proteins, data from same patient, etc.) but is an important consideration for all studies to ensure the developed model is as robust as possible.

After subsetting the data, the next step is to create and train the model on the training data set. The development of ML packages such as scikit-learn and Keras has drastically simplified the initiation and training of models; in most instances, you simply import the library containing the model you want, instantiate the model, and then train the model using the `.fit()` method. It is during these steps that the various hyperparameters are established. It is important to keep in mind that models often utilize the computers random number generator to produce initial trainable parameters for the model (weights, etc.). Some models can be sensitive to the initialization state, and thus it is recommended to also assign a seed value to the RNG algorithm during model initiation in order to ensure reproducibility.³⁴ Sensitivity to initialization can then be assessed by comparing model performance using different seed values.

3.5. Validation

At this point, you should have a model capable of making predictions on unseen data. How good are these predictions? The process of validation is done to assess model performance—observations with known labels are fed into the trained model, and the validity of the predictions is established by comparison to ground truth values. There are a number of different metrics which are used to judge model performance. For regression tasks, mean squared error (MSE) and r^2 are typically used to assess accuracy. For classification tasks, a few different metrics exist, all of which provide slightly different information on model performance. These metrics include classification accuracy, log loss, receiver operating characteristic (ROC) curves and the area under the curve (AUC), and F1 score.⁹⁹

It is also important to highlight that there are two different steps in the validation process. The first step involves characterization of model performance during model development, which subsequently allows for the tuning of model hyperparameters as a means of improving performance. After this process has been complete, a final testing phase on an entirely new set of test data should be completed with the final, optimized model, and then performance on this data set is what should be reported. This paradigm necessitates two unique testing subsets from the data, and thus has more stringent requirements for the quantity of initial data. Another alternative that is commonly used is the process of cross-validation on the initial training data set. Cross-validation involves the fragmentation of the data set into a few groups, or folds (Figure 2C). The model is then trained and validated on different subsets of the folds, and the process is repeated until all of the data has been used for validation. As an example, 5-fold cross validation involves the fragmentation of the data into groups containing 20% of the original data set. The first iteration uses the first four folds for training and the fifth fold for validation. The second iteration uses the first, second, third, and fifth fold for training and the fourth fold for validation, etc. At the completion of the process, you thus have five different accuracy scores for the five iterations with which to assess overall model performance. Cross-validation is an excellent tool for minimizing model overfitting and ensuring model generalizability, particularly when using smaller data sets. Cross-validation can be easily implemented using the `cross_val_score` method in scikit-learn.

The use of cross-validation methods also allows for the estimation of the bias/variance trade-off discussed previously. If each individual validation score is lower than that overall cross-validated score, the model likely suffers from high variance where each individual training process leads to an overfit model. In this instance, decreasing the complexity of the model can be helpful. On the other hand, if the training and cross-validated scores are similar but worse than the desired score, then the model likely suffers from high bias and model complexity should be increased to improve overall model performance.

3.6. Hyperparameter Tuning

Following the initial validation of a model to judge its performance, the next step is likely the tuning of the model to improve performance. This process is typically done through modification to the model hyperparameters and is thus known as hyperparameter tuning. The choice of which hyperparameters to vary and which to leave as default, as well as the extent of variation, is a skill that is honed through practice, and we highly recommend new users experiment with hyperparameter modification extensively. One method for hyperparameter tuning is known as the grid search method—in this technique, a range of preset values for all hyperparameters are established on a grid, and the model is subsequently trained and validated using every combination. At the conclusion of the search, the combination of hyperparameters giving the highest model performance can be identified. This process can be automated using the scikit-learn GridSearchCV class. This grid search method is computationally expensive and can frequently offer only minor improvements in model performance but may still be useful. Random search is a comparable technique which is computationally less expensive but also less robust. Grid search is limited in scope by the initial hyperparameter grid established, and thus functions best when a relatively small number of hyperparameters needs to be screened. In contrast, random search is capable of sampling larger hyperparameter spaces due to its lower computational cost, which increases the chances of identifying promising hyperparameter regions at the expense of overlooking optimal groupings. More complex hyperparameter tuning methods, such as Bayesian optimization and genetic algorithms, have been developed to address the inefficiencies of grid/random search algorithms and can be excellent options for certain applications/models.¹⁰⁰

3.7. Ensemble Techniques

Ensembling refers to the process of creating multiple different models and then combining their predictions together via some form of weighted averaging to arrive at a single prediction. Ensembling has several benefits, including the establishment of more robust predictions which incorporate the complementary strengths of different models, as well as the calculation of uncertainty or variance within predictions. Ensembling can be accomplished using a single type of ML algorithm, with different models trained either with different data sets or different hyperparameters, or it can involve the combination of multiple different types of ML models. AutoML is a Python package which automates the training of different classes of models on the same data set, and it is an excellent option for one wishing to ensemble different algorithms. Similarly, VotingClassifier is a method built into scikit-learn which ensembles various classification models and outputs the best performer.

Decision trees in particular benefit from the use of ensemble techniques, as a single decision tree is prone to overfitting and only outputs discontinuous data. A common implementation of

decision trees is the use of a random forest model. Random forests, as the name implies, involve an ensemble of numerous different decision trees together in one model. Random forests utilize a process known as bootstrap aggregation, or bagging, in which individual decision trees are trained on a different random subset of the training data set. Differences between trees are further established by limiting which features can be interrogated at each node, forcing individual trees to take new paths through the data. The end result is a number of predictions given for any new data, which can then be averaged (for regression) or selected via majority voting (for classification) to arrive at a single predicted label.

Another ensembling decision tree method are boosted tree models, including adaptive boosting and gradient boosting techniques. This process works by training numerous simple decision trees in an iterative manner, with the results of one decision tree being fed as an input into subsequent trees. For adaptive boosting, incorrectly predicted observations are weighted more heavily in subsequent iterations in order to drive improvement, while gradient boosting involves the use of residual prediction errors as a label in subsequent tree training. The final predicted label is then assigned based on weighted average or weighted voting from all predictors used. One form of gradient boosted decision trees, known as XGBoost, has proven to be an especially accurate ML model for a variety of different tasks.¹⁰¹

3.8. Model Usage/Interpretation

After a ML model has been trained, tuned, and validated against the final test data set, the final step involves actually using the model. Often, this utility arises from the ability of the model to predict values or classes from completely new data. These predictions can then be used to help guide the design of new materials or experiments in a time- and cost-effective manner. Ideally, the results of the new experiments can then be compared back to the original prediction to further validate the effectiveness of the model.

Another important use for developed models is the interrogation of the prediction process to understand the patterns or insights gleaned by the ML algorithm. This interrogation serves two purposes—it provides a useful check that allows users to judge whether or not the model can be trusted, and it can be used to gain a better understanding of the underlying structure—function behavior the model is trying to simulate.¹⁰² Different kinds of ML models offer different inherent capabilities for understanding the mechanisms behind the prediction. For example, the individual component weights of linear or logistic regression models can be evaluated to conceptually grasp relationship of different features to the prediction. For tree-based models, feature importance can be extracted from the trained models in scikit-learn to see what features were most important in improving leaf purity. Saliency maps are used to help identify which regions of data, like images, contribute most to certain classifications, and can thus be a way to try to understand the predictive behavior of neural networks.²

However, other models behave more as black-boxes, and other tools are necessary to deconstruct those models and provide a visual or textual description of feature importance. Local interpretable model-agnostic explanations (LIME) is a technique used to help explain predictions through the development of an interpretable model which can recreate the original model performance on small subsets of the data. LIME works by making variations to features in a local (i.e., one

observation) region and monitoring how these variations alter the predicted output using a simple, interpretable model like lasso regression or decision trees.¹⁰³ A similar technique is Shapely Additive Explanations (SHAP),¹⁰⁴ an explanatory model which assigns values to individual features in proportion to the amount they contribute (both positively and negatively) toward a given prediction in relation to the average prediction across the whole data set. This technique is based on the concept of Shapley values from game theory.¹⁰³ In SHAP, Shapely values are combined using a linear additive feature attribution method and are thus a combination of Shapely values and LIME. The SHAP process produces a plot of feature importance, as well as a summary plot illustrating the Shapely value for each feature. Observation/dependence plots are also generated to interpret how different features interact with each other while influencing model performance. SHAP has been used to interrogate predictions of high performing polycationic delivery vehicles,²⁷ to facilitate feature engineering for models of immunomodulatory osteoinductive biomaterials,¹⁰⁵ and to conduct a sensitivity analysis for models predicting the enzymatic activity of synthetic nanomaterials.¹⁰⁶ SHAP calculation can be implemented using the Python package *shap*, works well with tree-based models in *scikit-learn*, and is already implemented in several boosted-tree frameworks.

4. HANDS-ON EXAMPLE IN GOOGLE COLAB

To provide readers with a hands-on demonstration of the techniques and process described here, we have created an example ML notebook for modeling a real biomaterial design challenge. We hope this example provides insight into how biomaterial-based problems support the need for ML-driven experimentation and how that experimentation can be successfully implemented. The scripts used to generate, train, validate, and analyze the models are written in Python and included in an open access Google Colaboratory (Colab) notebook, which can be easily accessed using any Internet browser and does not require the installation of any software. This notebook can be accessed at www.gormleylab.com/MLcolab. An introductory understanding of Python syntax and the use of common data science libraries, including NumPy and Pandas, is assumed though not necessary to get started. We highly encourage interested readers to utilize this code as a means of practicing the techniques we've discussed and experiment on their own. After all, the best way to learn a new technique is to practice using it yourself.

While we encourage readers to engage with the tutorial through the interactive Google Colab notebook linked above, a brief overview of the interactive tutorial is included here. This work, including the data used,⁷³ is based on published work by our group to accelerate the development of novel polymer-protein hybrids.⁹ The example focuses on a supervised learning regression task in which the length and monomer composition of complex heterocopolymers (features) are used to predict the retained activity of polymer-enzyme complexes following exposure to denaturing conditions (labels). In this tutorial we will use a training data set consisting of an initial seed library of copolymers complexed with the enzyme glucose oxidase to help us design new polymer architectures with improved stabilization performance.

4.1. Data Preprocessing

The first step is to import the data set into our working environment within Python. The *pandas* python library can be

used to read .csv files hosted online. We follow this by removing columns from the data set that are not relevant to our objective.

```
import pandas as pd
url = 'https://raw.githubusercontent.com/webbtheosim/PPH_public/main/data/gox.csv' #URL from Github raw data
df = pd.read_csv(url) #Read data into a pandas dataframe
dataset = df.copy()
dataset = dataset.drop(labels = ['Polymer Number', 'Generation', 'Gelation'], axis = 1) #Dropping columns not needed for model training
```

We can now do a high level examination of a data set using *dataset.info()*, which provides information such as the data type for each column, non-null values and memory usage. The data type information is often quite useful in finding discrepancies or issues within the data set, including situations where the values of a column could have been stored as a string or non-numeric data type.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 624 entries, 0 to 623
Data columns (total 10 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   DP                                           624 non-null    int64
1   2-(diethylamino)ethyl methacrylate         624 non-null    float64
2   2-hydroxypropyl methacrylate                624 non-null    float64
3   2-sulfopropyl methacrylate                  624 non-null    float64
4   butyl methacrylate                          624 non-null    float64
5   3-(dimethylamino)propyl methacrylate        624 non-null    float64
6   methyl methacrylate                         624 non-null    float64
7   poly(ethylene glycol) methyl ether methacrylate 624 non-null    float64
8   trimethylammonium chloride ethyl methacrylate 624 non-null    float64
9   REA                                           624 non-null    float64
dtypes: float64(9), int64(1)
memory usage: 48.9 KB
```

Following the initial cleaning/quality check on the data set, we can now prepare our train and test data. First, we will need to import the *train_test_split* function from *scikit-learn* which will be used to split our data into the different sets and *StandardScaler* which will be used to scale our data

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

The next step is to isolate the feature columns (monomer composition and length) as well as the label column (retained enzyme activity or REA). We will save the data frame containing the feature columns as “X” with the target column dropped from this. Our target column will be saved as “y”. Once this is done, we can populate the arguments of the *train_test_split* function, passing our features (X), our target (y), the *test_size* which is the portion of data that we want to save for testing, and a *random_state* which will aid in assuring we have reproducible results. In our case, since we have a small amount of data we want to use as much training data as possible so we will use only 10% of our data for testing and the rest will be used for training.

```
X = dataset.drop('REA', axis = 1) # Dropping our target column
y = dataset['REA'] # Extracting the GOx REA column
X_train_raw, X_test_raw, y_train, y_test = train_test_split(X,y,test_size = 0.10, random_state = 42)
```

Now that we have split our data into the test and training sets, we still need to scale our data as some of the features exists on different scales (e.g., monomer composition in percentages, polymer length in hundreds of units). For this, we will first instantiate a *StandardScaler()* object to which we will fit our training data and then we will transform this data to the scaled form. It is important to remember that only the training data should be fit to the scaler.

```
scaler = StandardScaler() #Instantiating scaler to prepare our final data to use during model development
X_train = scaler.fit_transform(X_train_raw) # fitting the scaler and transforming our train data using .fit_transform
X_test = scaler.transform(X_test_raw) #Using .transform to scale our test data with the settings obtained from fitting the training data
```

4.2. Model Training

Once we have prepared our data, we can begin model training by first importing *RandomForestRegressor* from scikit-learn and instantiating a Random Forest Regressor. We will give this a *random_state* argument to ensure reproducible results. Then we can continue to train our model on the features (X) and labels (y) of the training data set by passing these objects to the *fit()* function.

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(random_state=42) #Using default settings but setting a seed = 42
rf.fit(X_train,y_train) #Fitting the model to the training data
```

4.3. Validation

Following initial model training, one should validate model performance. For this, we will use 5-fold cross validation, imported from scikit-learn via the *cross_val_score* method. We will look at three metrics to describe our model performance, negative mean squared error (MSE), negative mean absolute error (MAE) and R^2 by passing each as a “scoring” keyword argument. The final metric is reported as the mean of the result of the 5 individual scores for each validation split.

```
from sklearn.model_selection import cross_val_score
import numpy as np
cv_results_r2 = np.mean(cross_val_score(rf, X_train, y_train, cv=5, scoring = 'r2')) #CV R2 scoring, setting cv=5 for 5-fold cross validation
cv_results_nmse = np.mean(cross_val_score(rf, X_train, y_train, cv=5, scoring = 'neg_mean_squared_error')) #CV training negative MSE
cv_results_nmae = np.mean(cross_val_score(rf, X_train, y_train, cv=5, scoring = 'neg_mean_absolute_error')) # CV training MAE

cv_dict = {'R2':[cv_results_r2], 'MSE':[cv_results_nmse], 'MAE':[cv_results_nmae]}
cv_df = pd.DataFrame.from_dict(cv_dict)
cv_df
```

The cross-validation results are displayed on the data frame below. These metrics correspond to a relatively average performance of the model, so the next step is to try to improve model performance via hyperparameter tuning.

	R2	MSE	MAE
	0.635365	-56.347711	-5.196263

4.4. Hyperparameter Tuning and Model Evaluation

Once we have done an initial evaluation of our model after fitting to the training data, we can investigate improving model performance through hyperparameter tuning. While there are a number of hyperparameters that can be altered for a Random Forest model (a full list can be found in scikit-learn’s documentation), the hyperparameters that are tuned for this tutorial include:

n_estimators - Number of decision trees (we will try 20 evenly spaced values in the range 50–2000)

max_features - Maximum number of features to be considered for a node split (we will try ‘sqrt’, ‘auto’, and ‘log2’)

min_samples_split - The minimum number of observations necessary to split a node in a tree (we will try 2, 3, 5, and 10)

max_depth - The depth of a decision tree (we will try 5, 10, 20, and 30)

There are a number of different methods to perform hyperparameter tuning in scikit-learn, each with their own benefits and costs. For this tutorial, we will focus on two common algorithms - Grid Search and Random Search. For both methods, the user creates a grid which contains the parameters to tune and the search space for these parameters. If Grid Search is used, every parameter combination is used to create a model and evaluate it, which is a robust but

computationally expensive task. A less computationally expensive alternative is using Random Search, where only some random combinations of hyperparameters are used to create models and evaluate them. For this tutorial, we will use Random Search as this will provide results in a more efficient manner. Whether one used Grid Search or Random Search, the models created will only be as good as your grid defining the parameter spaces to test. Let us begin by setting up the hyperparameters to tune and follow that by creating our grid using a dictionary.

#Setting up the hyperparameters to tune

```
n_estimators = [int(x) for x in np.linspace(50,2000,20)] # here we use int(x) as cannot have half a tree
max_features = ['sqrt', 'auto', 'log2'] #The maximum number of features to consider at a split
min_samples_split = [2,3,5,10] # Minimum to split node
max_depth = [5, 10, 20, 30]
```

#Creating the grid

```
grid = {'n_estimators': n_estimators,
        'max_features': max_features,
        'min_samples_split': min_samples_split,
        'max_depth': max_depth}
```

After choosing the hyperparameters to tune and the parameter space over which to search, we can now import *RandomizedSearchCV*, which is a scikit-learn function for random search hyperparameter optimization with cross validation. We will look at 100 random models created using our grid, set a *random_state*, and use the negative MSE to evaluate the models created. Some important parameters of *RandomizedSearchCV* are

estimator - The model that we want to tune hyperparameters for

param_distributions - The parameters and the space we want to search

n_iter - How many models or combinations do we want to try

refit - When set to “True”, the best model will be stored for future use and refit to the data

cv - The type of cross validation we want to do, here we are doing 5-fold

scoring - Metric to evaluate the models, we will use MSE

```
from sklearn.model_selection import RandomizedSearchCV
rf_RS = RandomizedSearchCV(estimator= rf, param_distributions= grid,
                           n_iter = 100, cv = 5, random_state= 42,
                           refit = True, return_train_score= True,
                           scoring='neg_mean_squared_error')
```

```
rf_RS.fit(X_train,y_train)
```

The *.best_score_* and *best_estimator_* attributes on *rf_RS* can be used to display the best score and best estimator found during the search.

```
best_score = rf_RS.best_score_
best_estimator = rf_RS.best_estimator_
print(' Best score: ', best_score)
print('Best estimator: ', best_estimator)
```

```
Best score: -54.82880683041073
RandomForestRegressor(max_depth=30, max_features='log2', n_estimators=768,
                      random_state=42)
```

Here we see a slight improvement in MSE (from −56.3 to −54.8) following hyperparameter tuning. We can also see that our best estimator uses a max depth of 30, with 768 trees and max features using ‘log2’. We can further evaluate our tuned model by making some predictions with our test data and calculating the three metrics we mentioned earlier. Here we can use *rf_RS.best_estimator_* and can call methods such as *.predict()* as we would with any other model.

```

predictions = rf_RS.best_estimator_.predict(X_test) # Predictions made on
the test set
mse_test = mean_squared_error(y_test,predictions) #Evaluation of MSE based
on actual label (y_test) and predicted label
mae_test = mean_absolute_error(y_test, predictions) #Evaluation of MAE bas
ed on actual label (y_test) and predicted label
r2_test = r2_score(y_test, predictions) #Evaluation of r2 based on actual
label (y_test) and predicted label
print('MSE on test set: ',mse_test)
print('MAE on test set: ', mae_test)
print('R2 on test set: ', r2_test )

```

```

MSE on test set: 63.34329778984832
MAE on test set: 4.765689452373583
R2 on test set: 0.7124321789522972

```

We can see our tuned model improved its performance on the test set, with lower error metrics (MSE and MAE) and a higher r_2 value. Another way to evaluate model performance is by plotting predicted labels vs actual labels, which can be accomplished using the python library matplotlib (Figure 5).

```

import matplotlib.pyplot as plt
plt.plot(y_test,predictions, 'o')
plt.xlabel('Actual REA')
plt.ylabel('Predicted REA')
plt.plot([0,70],[0,70])
plt.title('Random Forest score: '+ str(round(r2_test,2))+ ' MAE: ' + str(r
ound(mae_test,2)))
plt.show()

```

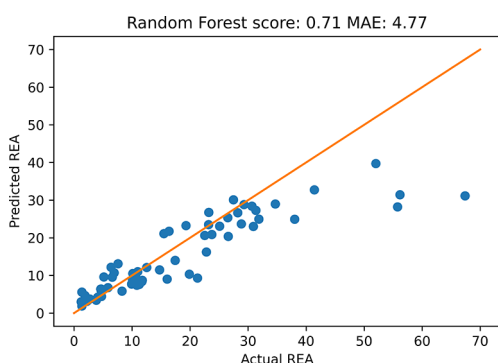


Figure 5. Model performance is evaluated by plotting the actual REA labels from the data (x-axis) against the predicted REA labels from the model output of the test data set (y-axis). The orange line shows the 1:1 equivalence that would be obtained with a “perfect” model, with values below the line representing an underprediction of the actual REA and values above the line representing an overprediction. Quantitative metrics of model performance (Random Forest R^2 score and MAE) are listed above the graph for reference.

Within our training and hyperparameter tuning, we find that our Random Forest model scores $R^2 = 0.71$. This leads us to the question of “How good is this score given our problem?” While the answer depends on the machine learning application and difficulty of the problem that we are investigating, successful demonstrations of using machine learning for biomaterials design in our case study, and other examples have demonstrated a wide range of R^2 from $\sim R^2 = 0.5$ – 0.98 . Such differences in accuracy are generally attributable to key factors: (1) Relevance of training data to the target prediction label, (2) quantity of training data, (3) data representation, and (4) many of the other considerations discussed in this tutorial (model selection, hyperparameter selection, etc.)

Some of these examples are provided below:

- Machine Learning on a Robotic Platform for the Design of Polymer–Protein Hybrids⁹
- Machine-Assisted Discovery of Chondroitinase ABC Complexes toward Sustained Neural Regeneration⁸
- Machine-Learning-Guided Discovery of 19F MRI Agents Enabled by Automated Copolymer Synthesis¹⁰⁷

- Featurization strategies for polymer sequence or composition design by machine learning⁹⁷

4.5. Using the Model

Finally, we want to use our model to make predictions on new data. Here we will create a dictionary in which we will input the values for each feature (monomer composition and degree of polymerization) of new, untested polymer designs, followed by feature scaling.

```

dictionary = {'DP':100, '2-(diethylamino)ethyl methacrylate':0.0,
'2-hydroxypropyl methacrylate':0.1, '2-
sulfopropyl methacrylate':0.10,
'butyl methacrylate':0.10, '3-
(dimethylamino)propyl methacrylate':0.3,
'methyl methacrylate':0.2,'poly(ethylene glycol) methyl ether metha
crylate':0.0,
'trimethylammonium chloride ethyl methacrylate':0.2}

```

```

df = pd.DataFrame([dictionary])
possible_polymer= df.copy()
possible_polymer_scaled =scaler.transform(possible_polymer)

```

We can now pass these features into our tuned model to make a prediction for the REA of the untested polymers.

```

preds_new = rf_RS.best_estimator_.predict(possible_polymer_scaled) #Genera
te predictions of REA using the possible polymer features input previously
possible_polymer_2 = possible_polymer.copy()
possible_polymer_2['Predicted REA'] = preds_new #Store the generated predi
ctions in a new column called Predicted REA
possible_polymer_2

```

butyl methacrylate	3-(dimethylamino)propyl methacrylate	methyl methacrylate	poly(ethylene glycol) methyl ether methacrylate	triethylammonium chloride ethyl methacrylate	Predicted REA
0.1	0.3	0.2	0.0	0.2	24.412464

4.6. Model Interpretation

Shapley values are one method for interpreting model predictions that can be extremely helpful in validating model performance and understanding the process behind model predictions. After installing SHAP, we can create an explainer object to which we pass the feature names and the model.

```

!pip -q install shap
import shap
explainer = shap.Explainer(rf_RS.best_estimator_.predict,X_test, feature_n
ames= X_train_raw.columns)
shap_vals = explainer(X_test)

```

The summary plot is a helpful tool for evaluating the contribution of each feature on model performance (Figure 6). Each feature is given a separate row, with every observation of the feature given a point that is color-coded based on feature

```
shap.summary_plot(shap_vals)
```

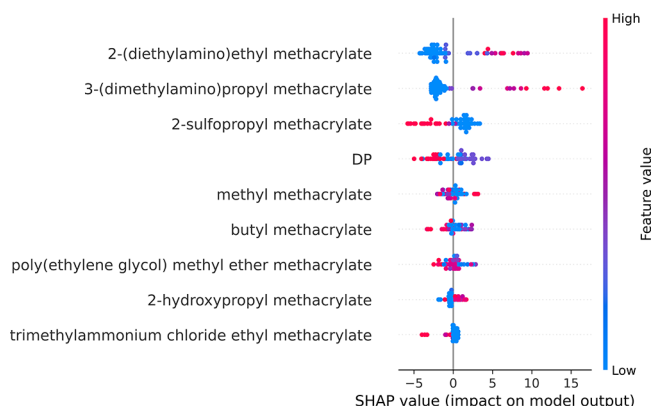


Figure 6. Summary plot of SHAP analysis of random forest regression model built here. The relative impact of each feature on model output (SHAP value) is displayed as horizontal position, and the relative magnitude of the feature value for each data point is color-coded (red = high value, blue = low value).

value. The relative contribution of each point toward the prediction (SHAP value) is then plotted on the horizontal axis. In this instance, we see that the incorporation of cationic monomers had the largest impact on model predictions, with higher incorporation of these monomers (red dots) contributing to higher REA (higher SHAP values). In contrast, the anionic 2-sulfopropyl methacrylate monomer appears to have a hindrance on enzyme protection, with observations containing higher values of this monomer having negative SHAP values.

5. SUMMARY

Biomaterials offer the promise to help solve a wide variety of pressing biomedical and biotechnological problems. However, the highly dimensional design space for material development, alongside the requirements for compatibility and efficacy in complex biological environments, means the development of novel biomaterials is fraught with time-consuming trial-and-error experimentation. We believe that the adoption of high-throughput experimental techniques alongside the implementation of ML driven data science can significantly accelerate biomaterial design campaigns, and that these approaches will become commonplace for the next-generation of biomaterial scientists. The goal of this Perspective was to provide readers with a solid foundation for understanding the definitions and process of ML and how it can be implemented in biomaterials science. It also provides a useful hands-on experience for learning and practicing with real-life data sets.

■ ASSOCIATED CONTENT

Supporting Information

The Supporting Information is available free of charge at <https://pubs.acs.org/doi/10.1021/acspolymersau.2c00037>.

Alphabetized list of ML vocabulary and definitions used throughout the article (PDF)

■ AUTHOR INFORMATION

Corresponding Author

Adam J. Gormley – Department of Biomedical Engineering, Rutgers, The State University of New Jersey, Piscataway, New Jersey 08854, United States; orcid.org/0000-0002-2884-725X; Email: adam.gormley@rutgers.edu

Authors

Travis A. Meyer – Department of Biomedical Engineering, Rutgers, The State University of New Jersey, Piscataway, New Jersey 08854, United States

Cesar Ramirez – Department of Biomedical Engineering, Rutgers, The State University of New Jersey, Piscataway, New Jersey 08854, United States

Matthew J. Tamasi – Department of Biomedical Engineering, Rutgers, The State University of New Jersey, Piscataway, New Jersey 08854, United States

Complete contact information is available at:

<https://pubs.acs.org/doi/10.1021/acspolymersau.2c00037>

Author Contributions

The manuscript was written through contributions of all authors. All authors have given approval to the final version of the manuscript. CRediT: **Travis A Meyer** investigation (lead), writing-original draft (lead), writing-review & editing (equal); **Cesar Ramirez** software (lead), writing-review & editing

(equal); **Matthew J. Tamasi** conceptualization (supporting), investigation (supporting), software (supporting), writing-original draft (supporting), writing-review & editing (equal); **Adam J. Gormley** conceptualization (lead), investigation (supporting), supervision (lead), writing-review & editing (equal).

Funding

T.A.M. acknowledges support from NIH IRACDA K12GM09385, M.J.T. acknowledges support from NIH NIGMS T32GM135141, and A.J.G. acknowledges support from NIH NIGMS R35GM138296.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

The authors would like to thank Dr. Christopher Radford for assistance in reviewing the manuscript.

■ REFERENCES

- (1) Basu, B.; Gowtham, N. H.; Xiao, Y.; Kalidindi, S. R.; Leong, K. W. Biomaterialomics: Data science-driven pathways to develop fourth-generation biomaterials. *Acta Biomaterialia* **2022**, *143*, 1–25.
- (2) Greener, J. G.; Kandathil, S. M.; Moffat, L.; Jones, D. T. A guide to machine learning for biologists. *Nat. Rev. Mol. Cell Biol.* **2022**, *23* (1), 40–55.
- (3) Inza, I.; Calvo, B.; Armañanzas, R.; Bengoetxea, E.; Larranaga, P.; Lozano, J. A. Machine learning: an indispensable tool in bioinformatics. In *Bioinformatics methods in clinical research*; Springer, 2010; pp 25–48.
- (4) Butler, K. T.; Davies, D. W.; Cartwright, H.; Isayev, O.; Walsh, A. Machine learning for molecular and materials science. *Nature* **2018**, *559* (7715), 547–555.
- (5) Kerner, J.; Dogan, A.; von Recum, H. Machine learning and big data provide crucial insight for future biomaterials discovery and research. *Acta Biomaterialia* **2021**, *130*, 54–65.
- (6) Suwardi, A.; Wang, F.; Xue, K.; Han, M.-Y.; Teo, P.; Wang, P.; Wang, S.; Liu, Y.; Ye, E.; Li, Z.; et al. Machine Learning-Driven Biomaterials Evolution. *Adv. Mater.* **2022**, *34* (1), 2102703.
- (7) Xue, K.; Wang, F.; Suwardi, A.; Han, M.-Y.; Teo, P.; Wang, P.; Wang, S.; Ye, E.; Li, Z.; Loh, X. J. Biomaterials by design: Harnessing data for future development. *Materials Today Bio* **2021**, *12*, 100165.
- (8) Kosuri, S.; Borca, C. H.; Mugnier, H.; Tamasi, M.; Patel, R. A.; Perez, I.; Kumar, S.; Finkel, Z.; Schloss, R.; Cai, L.; et al. Machine-Assisted Discovery of Chondroitinase ABC Complexes toward Sustained Neural Regeneration. *Adv. Healthcare Mater.* **2022**, *11* (10), 2102101.
- (9) Tamasi, M. J.; Patel, R. A.; Borca, C. H.; Kosuri, S.; Mugnier, H.; Upadhyay, R.; Murthy, N. S.; Webb, M. A.; Gormley, A. J. Machine Learning on a Robotic Platform for the Design of Polymer-Protein Hybrids. *Adv. Mater.* **2022**, *34* (30), 2270219.
- (10) Li, F.; Han, J.; Cao, T.; Lam, W.; Fan, B.; Tang, W.; Chen, S.; Fok, K. L.; Li, L. Design of self-assembly dipeptide hydrogels and machine learning via their chemical features. *Proc. Natl. Acad. Sci. U. S. A.* **2019**, *116* (23), 11259–11264.
- (11) Kwaria, R. J.; Mondarte, E. A. Q.; Tahara, H.; Chang, R.; Hayashi, T. Data-Driven Prediction of Protein Adsorption on Self-Assembled Monolayers toward Material Screening and Design. *ACS Biomaterials Science & Engineering* **2020**, *6* (9), 4949–4956.
- (12) Epa, V. C.; Yang, J.; Mei, Y.; Hook, A. L.; Langer, R.; Anderson, D. G.; Davies, M. C.; Alexander, M. R.; Winkler, D. A. Modelling human embryoid body cell adhesion to a combinatorial library of polymer surfaces. *J. Mater. Chem.* **2012**, *22* (39), 20902–20906.
- (13) Rostam, H. M.; Fisher, L. E.; Hook, A. L.; Burroughs, L.; Luckett, J. C.; Figueredo, G. P.; Mbadugha, C.; Teo, A. C. K.; Latif, A.; Kämmerling, L.; et al. Immune-Instructive Polymers Control Macrophage Phenotype and Modulate the Foreign Body Response In Vivo. *Matter* **2020**, *2* (6), 1564–1581.

- (14) Pugar, J. A.; Childs, C. M.; Huang, C.; Haider, K. W.; Washburn, N. R. Elucidating the Physicochemical Basis of the Glass Transition Temperature in Linear Polyurethane Elastomers with Machine Learning. *J. Phys. Chem. B* **2020**, *124* (43), 9722–9733.
- (15) Wu, C.-T.; Chang, H.-T.; Wu, C.-Y.; Chen, S.-W.; Huang, S.-Y.; Huang, M.; Pan, Y.-T.; Bradbury, P.; Chou, J.; Yen, H.-W. Machine learning recommends affordable new Ti alloy with bone-like modulus. *Mater. Today* **2020**, *34*, 41–50.
- (16) Zhang, Y.; Wen, C.; Wang, C.; Antonov, S.; Xue, D.; Bai, Y.; Su, Y. Phase prediction in high entropy alloys with a rational selection of materials descriptors and machine learning models. *Acta Mater.* **2020**, *185*, 528–539.
- (17) Zou, C.; Li, J.; Wang, W. Y.; Zhang, Y.; Lin, D.; Yuan, R.; Wang, X.; Tang, B.; Wang, J.; Gao, X.; et al. Integrating data mining and machine learning to discover high-strength ductile titanium alloys. *Acta Mater.* **2021**, *202*, 211–221.
- (18) Hsu, Y.-C.; Yu, C.-H.; Buehler, M. J. Using Deep Learning to Predict Fracture Patterns in Crystalline Solids. *Matter* **2020**, *3* (1), 197–211.
- (19) Lazarovits, J.; Sindhwani, S.; Tavares, A. J.; Zhang, Y.; Song, F.; Audet, J.; Krieger, J. R.; Syed, A. M.; Stordy, B.; Chan, W. C. W. Supervised Learning and Mass Spectrometry Predicts the in Vivo Fate of Nanomaterials. *ACS Nano* **2019**, *13* (7), 8023–8034.
- (20) Ban, Z.; Yuan, P.; Yu, F.; Peng, T.; Zhou, Q.; Hu, X. Machine learning predicts the functional composition of the protein corona and the cellular recognition of nanoparticles. *Proc. Natl. Acad. Sci. U. S. A.* **2020**, *117* (19), 10492–10499.
- (21) Yamankurt, G.; Berns, E. J.; Xue, A.; Lee, A.; Bagheri, N.; Mrksich, M.; Mirkkin, C. A. Exploration of the nanomedicine-design space with high-throughput screening and machine learning. *Nat. Biomed Eng.* **2019**, *3* (4), 318–327.
- (22) Yang, L.; Pijuan-Galito, S.; Rho, H. S.; Vasilevich, A. S.; Eren, A. D.; Ge, L.; Habibović, P.; Alexander, M. R.; de Boer, J.; Carlier, A.; et al. High-Throughput Methods in the Discovery and Study of Biomaterials and Materiobiology. *Chem. Rev.* **2021**, *121* (8), 4561–4677.
- (23) Soheilimoghaddam, F.; Rumble, M.; Cooper-White, J. High-Throughput Routes to Biomaterials Discovery. *Chem. Rev.* **2021**, *121* (18), 10792–10864.
- (24) Gormley, A. J.; Webb, M. A. Machine learning in combinatorial polymer chemistry. *Nature Reviews Materials* **2021**, *6* (8), 642–644.
- (25) Webb, M. A.; Jackson, N. E.; Gil, P. S.; de Pablo, J. J. Targeted sequence design within the coarse-grained polymer genome. *Sci. Adv.* **2020**, *6* (43), 1–10.
- (26) Kumar, R.; Le, N.; Tan, Z.; Brown, M. E.; Jiang, S.; Reineke, T. M. Efficient Polymer-Mediated Delivery of Gene-Editing Ribonucleoprotein Payloads through Combinatorial Design, Parallelized Experimentation, and Machine Learning. *ACS Nano* **2020**, *14* (12), 17626–17639.
- (27) Kumar, R.; Le, N.; Oviedo, F.; Brown, M. E.; Reineke, T. M. Combinatorial Polycation Synthesis and Causal Machine Learning Reveal Divergent Polymer Design Rules for Effective pDNA and Ribonucleoprotein Delivery. *JACS Au* **2022**, *2* (2), 428–442.
- (28) Lake, B. M.; Salakhutdinov, R.; Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science* **2015**, *350* (6266), 1332–1338.
- (29) Altae-Tran, H.; Ramsundar, B.; Pappu, A. S.; Pande, V. Low Data Drug Discovery with One-Shot Learning. *ACS Central Science* **2017**, *3* (4), 283–293.
- (30) Yamada, H.; Liu, C.; Wu, S.; Koyama, Y.; Ju, S.; Shiomi, J.; Morikawa, J.; Yoshida, R. Predicting Materials Properties with Little Data Using Shotgun Transfer Learning. *ACS Central Science* **2019**, *5* (10), 1717–1730.
- (31) Shmilovich, K.; Mansbach, R. A.; Sidky, H.; Dunne, O. E.; Panda, S. S.; Tovar, J. D.; Ferguson, A. L. Discovery of self-assembling π -conjugated peptides by active learning-directed coarse-grained molecular simulation. *J. Phys. Chem. B* **2020**, *124* (19), 3873–3891.
- (32) Doan Tran, H.; Kim, C.; Chen, L.; Chandrasekaran, A.; Batra, R.; Venkatram, S.; Kamal, D.; Lightstone, J. P.; Gurnani, R.; Shetty, P.; et al. Machine-learning predictions of polymer properties with Polymer Genome. *J. Appl. Phys.* **2020**, *128* (17), 171104.
- (33) Hakimi, O.; Krallinger, M.; Ginebra, M.-P. Time to kick-start text mining for biomaterials. *Nature Reviews Materials* **2020**, *5* (8), 553–556.
- (34) Wang, A. Y.-T.; Murdock, R. J.; Kauwe, S. K.; Oliynyk, A. O.; Gurlo, A.; Brgoch, J.; Persson, K. A.; Sparks, T. D. Machine Learning for Materials Scientists: An Introductory Guide toward Best Practices. *Chem. Mater.* **2020**, *32* (12), 4954–4965.
- (35) Lundstedt, T.; Seifert, E.; Abramo, L.; Thelin, B.; Nyström, Å.; Pettersen, J.; Bergman, R. Experimental design and optimization. *Chemometrics and intelligent laboratory systems* **1998**, *42* (1–2), 3–40.
- (36) Simon, C. G., Jr.; Lin-Gibson, S. Combinatorial and High-Throughput Screening of Biomaterials. *Adv. Mater.* **2011**, *23* (3), 369–387.
- (37) Anderson, D. G.; Levenberg, S.; Langer, R. Nanoliter-scale synthesis of arrayed biomaterials and application to human embryonic stem cells. *Nat. Biotechnol.* **2004**, *22* (7), 863–866.
- (38) Hook, A. L.; Chang, C.-Y.; Yang, J.; Luckett, J.; Cockayne, A.; Atkinson, S.; Mei, Y.; Bayston, R.; Irvine, D. J.; Langer, R.; et al. Combinatorial discovery of polymers resistant to bacterial attachment. *Nat. Biotechnol.* **2012**, *30* (9), 868–875.
- (39) Chapman, R.; Gormley, A. J.; Herpoldt, K.-L.; Stevens, M. M. Highly Controlled Open Vessel RAFT Polymerizations by Enzyme Degassing. *Macromolecules* **2014**, *47* (24), 8541–8547.
- (40) Chapman, R.; Gormley, A. J.; Stenzel, M. H.; Stevens, M. M. Combinatorial Low-Volume Synthesis of Well-Defined Polymers by Enzyme Degassing. *Angew. Chem., Int. Ed.* **2016**, *55* (14), 4500–4503.
- (41) Gormley, A. J.; Chapman, R.; Stevens, M. M. Polymerization Amplified Detection for Nanoparticle-Based Biosensing. *Nano Lett.* **2014**, *14* (11), 6368–6373.
- (42) Gormley, A. J.; Yeow, J.; Ng, G.; Conway, Ó.; Boyer, C.; Chapman, R. An Oxygen-Tolerant PET-RAFT Polymerization for Screening Structure–Activity Relationships. *Angew. Chem., Int. Ed.* **2018**, *57* (6), 1557–1562.
- (43) Tamasi, M.; Kosuri, S.; DiStefano, J.; Chapman, R.; Gormley, A. J. Automation of Controlled/Living Radical Polymerization. *Advanced Intelligent Systems* **2020**, *2* (2), 1900126.
- (44) Yeow, J.; Chapman, R.; Gormley, A. J.; Boyer, C. Up in the air: oxygen tolerance in controlled/living radical polymerisation. *Chem. Soc. Rev.* **2018**, *47* (12), 4357–4387.
- (45) Yeow, J.; Chapman, R.; Xu, J.; Boyer, C. Oxygen tolerant photopolymerization for ultralow volumes. *Polym. Chem.* **2017**, *8* (34), 5012–5022.
- (46) Xu, J.; Jung, K.; Atme, A.; Shanmugam, S.; Boyer, C. A Robust and Versatile Photoinduced Living Polymerization of Conjugated and Unconjugated Monomers and Its Oxygen Tolerance. *J. Am. Chem. Soc.* **2014**, *136* (14), 5508–5519.
- (47) Correa, S.; Boehnke, N.; Barberio, A. E.; Deiss-Yehiely, E.; Shi, A.; Oberlton, B.; Smith, S. G.; Zervantonakis, I.; Dreaden, E. C.; Hammond, P. T. Tuning Nanoparticle Interactions with Ovarian Cancer through Layer-by-Layer Modification of Surface Chemistry. *ACS Nano* **2020**, *14* (2), 2224–2237.
- (48) Duffy, C.; Venturato, A.; Callanan, A.; Lilienkamp, A.; Bradley, M. Arrays of 3D double-network hydrogels for the high-throughput discovery of materials with enhanced physical and biological properties. *Acta Biomaterialia* **2016**, *34*, 104–112.
- (49) Vegas, A. J.; Veisheh, O.; Doloff, J. C.; Ma, M.; Tam, H. H.; Bratlie, K.; Li, J.; Bader, A. R.; Langan, E.; Olejnik, K.; et al. Combinatorial hydrogel library enables identification of materials that mitigate the foreign body response in primates. *Nat. Biotechnol.* **2016**, *34* (3), 345–352.
- (50) Kim, T. H.; An, D. B.; Oh, S. H.; Kang, M. K.; Song, H. H.; Lee, J. H. Creating stiffness gradient polyvinyl alcohol hydrogel using a simple gradual freezing–thawing method to investigate stem cell differentiation behaviors. *Biomaterials* **2015**, *40*, 51–60.
- (51) Zhou, Q.; Kühn, P. T.; Huisman, T.; Nieboer, E.; van Zwol, C.; van Kooten, T. G.; van Rijn, P. Directional nanotopographic gradients:

a high-throughput screening platform for cell contact guidance. *Sci. Rep.* **2015**, *5* (1), 16240.

(52) Mei, Y.; Wu, T.; Xu, C.; Langenbach, K. J.; Elliott, J. T.; Vogt, B. D.; Beers, K. L.; Amis, E. J.; Washburn, N. R. Tuning cell adhesion on gradient poly (2-hydroxyethyl methacrylate)-grafted surfaces. *Langmuir* **2005**, *21* (26), 12309–12314.

(53) Li, B.; Ma, Y.; Wang, S.; Moran, P. M. Influence of carboxyl group density on neuron cell attachment and differentiation behavior: gradient-guided neurite outgrowth. *Biomaterials* **2005**, *26* (24), 4956–4963.

(54) Camacho, P.; Behre, A.; Fainor, M.; Seims, K. B.; Chow, L. W. Spatial organization of biochemical cues in 3D-printed scaffolds to guide osteochondral tissue engineering. *Biomaterials Science* **2021**, *9* (20), 6813–6829.

(55) Camacho, P.; Busari, H.; Seims, K. B.; Schwarzenberg, P.; Dailey, H. L.; Chow, L. W. 3D printing with peptide–polymer conjugates for single-step fabrication of spatially functionalized scaffolds. *Biomaterials Science* **2019**, *7* (10), 4237–4247.

(56) Pathak, S.; Kalidindi, S. R. Spherical nanoindentation stress–strain curves. *Materials Science and Engineering: R: Reports* **2015**, *91*, 1–36.

(57) Upadhyay, R.; Murthy, N. S.; Hoop, C. L.; Kosuri, S.; Nanda, V.; Kohn, J.; Baum, J.; Gormley, A. J. PET-RAFT and SAXS: High Throughput Tools To Study Compactness and Flexibility of Single-Chain Polymer Nanoparticles. *Macromolecules* **2019**, *52* (21), 8295–8304.

(58) Chaudhuri, O.; Gu, L.; Klumpers, D.; Darnell, M.; Bencherif, S. A.; Weaver, J. C.; Huebsch, N.; Lee, H.-p.; Lippens, E.; Duda, G. N.; et al. Hydrogels with tunable stress relaxation regulate stem cell fate and activity. *Nat. Mater.* **2016**, *15* (3), 326–334.

(59) Celiz, A. D.; Smith, J. G.; Patel, A. K.; Hook, A. L.; Rajamohan, D.; George, V. T.; Platt, L.; Patel, M. J.; Epa, V. C.; Singh, T.; et al. Discovery of a novel polymer for human pluripotent stem cell expansion and multilineage differentiation. *Advanced materials* **2015**, *27* (27), 4006–4012.

(60) Frank, T.; Tay, S. Flow-switching allows independently programmable, extremely stable, high-throughput diffusion-based gradients. *Lab Chip* **2013**, *13* (7), 1273–1281.

(61) Kim, L.; Vahey, M. D.; Lee, H. Y.; Voldman, J. Microfluidic arrays for logarithmically perfused embryonic stem cell culture. *Lab Chip* **2006**, *6* (3), 394–406.

(62) Huh, D.; Matthews, B. D.; Mammoto, A.; Montoya-Zavala, M.; Hsin, H. Y.; Ingber, D. E. Reconstituting organ-level lung functions on a chip. *Science* **2010**, *328* (5986), 1662–1668.

(63) Dahlman, J. E.; Kauffman, K. J.; Xing, Y.; Shaw, T. E.; Mir, F. F.; Dlott, C. C.; Langer, R.; Anderson, D. G.; Wang, E. T. Barcoded nanoparticles for high throughput in vivo discovery of targeted therapeutics. *Proc. Natl. Acad. Sci. U. S. A.* **2017**, *114* (8), 2060–2065.

(64) Yu, C.; Mannan, A. M.; Yvone, G. M.; Ross, K. N.; Zhang, Y.-L.; Marton, M. A.; Taylor, B. R.; Crenshaw, A.; Gould, J. Z.; Tamayo, P.; et al. High-throughput identification of genotype-specific cancer vulnerabilities in mixtures of barcoded tumor cell lines. *Nat. Biotechnol.* **2016**, *34* (4), 419–423.

(65) Gartner, T. E.; Jayaraman, A. Modeling and Simulations of Polymers: A Roadmap. *Macromolecules* **2019**, *52* (3), 755–786.

(66) Grünwald, F.; Alessandri, R.; Kroon, P. C.; Monticelli, L.; Souza, P. C. T.; Marrink, S. J. Polypy: a python suite for facilitating simulations of macromolecules and nanomaterials. *Nat. Commun.* **2022**, *13* (1), 68.

(67) Dhamankar, S.; Webb, M. A. Chemically specific coarse-graining of polymers: Methods and prospects. *J. Polym. Sci.* **2021**, *59* (22), 2613–2643.

(68) Bédier, A.; Bonini, F.; Verheyen, C. A.; Genta, M.; Martins, M.; Brefie-Guth, J.; Tratwal, J.; Filippova, A.; Burch, P.; Naveiras, O.; et al. An Injectable Meta-Biomaterial: From Design and Simulation to In Vivo Shaping and Tissue Induction. *Adv. Mater.* **2021**, *33* (41), 2102350.

(69) Raffaini, G.; Elli, S.; Ganazzoli, F. Computer simulation of bulk mechanical properties and surface hydration of biomaterials. *J. Biomed. Mater. Res., Part A* **2006**, *77A* (3), 618–626.

(70) Sun, T.-Y.; Liang, L.-J.; Wang, Q.; Laaksonen, A.; Wu, T. A molecular dynamics study on pH response of protein adsorbed on peptide-modified polyvinyl alcohol hydrogel. *Biomaterials Science* **2014**, *2* (3), 419–426.

(71) Johannes Laaksonen, T.; Mikael Laaksonen, H.; Tapio Hirvonen, J.; Murtomäki, L. Cellular automata model for drug release from binary matrix and reservoir polymeric devices. *Biomaterials* **2009**, *30* (10), 1978–1987.

(72) Murdock, R. J.; Kauwe, S. K.; Wang, A. Y.-T.; Sparks, T. D. Is Domain Knowledge Necessary for Machine Learning Materials Properties? *Integrating Materials and Manufacturing Innovation* **2020**, *9* (3), 221–227.

(73) Mugnier, H.; Upadhyay, R.; Murthy, N. S.; Webb, M. A.; Patel, R. A.; Gormley, A. J.; Tamasi, M.; Borca, C. H.; Kosuri, S. *Data on Enzyme Activity Retention in glucose oxidase, lipase, and horseradish peroxidase*; Princeton University, 2022.

(74) Wilkinson, M. D.; Dumontier, M.; Aalbersberg, I. J.; Appleton, G.; Axton, M.; Baak, A.; Blomberg, N.; Boiten, J.-W.; da Silva Santos, L. B.; Bourne, P. E.; et al. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* **2016**, *3* (1), 160018.

(75) Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V. Scikit-learn: Machine learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.

(76) Ketkar, N. Introduction to keras. In *Deep learning with Python*; Springer, 2017; pp 97–111.

(77) Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L. Pytorch: An imperative style, high-performance deep learning library. *arXiv (Machine Learning)*, December 3, **2019**, 1912.01703, ver. 1. DOI: 10.48550/arXiv.1912.01703

(78) Kuhn, M. Building predictive models in R using the caret package. *Journal of statistical software* **2008**, *28*, 1–26.

(79) Blaom, A. D.; Kiraly, F.; Lienart, T.; Simillides, Y.; Arenas, D.; Vollmer, S. J. MLJ: A Julia package for composable machine learning. *Journal of Open Source Software* **2020**, *5* (55), 2704.

(80) Van der Maaten, L.; Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **2008**, *9* (11), 2579–2605.

(81) McInnes, L.; Healy, J.; Melville, J. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv (Machine Learning)*, September 18, **2020**, 1802.03426, ver. 3. DOI: 10.48550/arXiv.1802.03426 (accessed 9/29/2022).

(82) Coifman, R. R.; Lafon, S. Diffusion maps. *Applied and Computational Harmonic Analysis* **2006**, *21* (1), 5–30.

(83) Dreiseitl, S.; Ohno-Machado, L. Logistic regression and artificial neural network classification models: a methodology review. *J. Biomed Inform* **2002**, *35* (5–6), 352–359.

(84) Schreiber-Gregory, D. N. Ridge Regression and multicollinearity: An in-depth review. *Model Assisted Statistics and Applications* **2018**, *13* (4), 359–365.

(85) Muthukrishnan, R.; Rohini, R. LASSO: A feature selection technique in predictive modeling for machine learning. In *2016 IEEE international conference on advances in computer applications (ICACA)*; IEEE, 2016; pp 18–20.

(86) Abu Alfeilat, H. A.; Hassanat, A. B. A.; Lasassmeh, O.; Tarawneh, A. S.; Alhasanat, M. B.; Eyal Salman, H. S.; Prasath, V. B. S. Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. *Big Data* **2019**, *7* (4), 221–248.

(87) Cervantes, J.; Garcia-Lamont, F.; Rodríguez-Mazahua, L.; Lopez, A. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing* **2020**, *408*, 189–215.

(88) Wang, J. An intuitive tutorial to Gaussian processes regression. *arXiv (Machine Learning)*, April 18, **2022**, 2009.10862, ver. 4. DOI: 10.48550/arXiv.2009.10862 (accessed 9/29/2022).

(89) Kotsiantis, S. B. Decision trees: a recent overview. *Artificial Intelligence Review* **2013**, *39* (4), 261–283.

- (90) Raileanu, L. E.; Stoffel, K. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence* **2004**, 41 (1), 77–93.
- (91) Apicella, A.; Donnarumma, F.; Isgro, F.; Prevete, R. A survey on modern trainable activation functions. *Neural Netw* **2021**, 138, 14–32.
- (92) Bank, D.; Koenigstein, N.; Giryas, R. Autoencoders. *arXiv (Machine Learning)*, April 13, **2021**, 2003.05991, ver. 2. DOI: [10.48550/arXiv.2003.05991](https://doi.org/10.48550/arXiv.2003.05991) (accessed 9/29/2022).
- (93) O'Shea, K.; Nash, R. An introduction to convolutional neural networks. *arXiv (Neural and Evolutionary Computing)*, December 2, **2015**, ver. 2. DOI: [10.48550/arXiv.1511.08458](https://doi.org/10.48550/arXiv.1511.08458) (accessed 9/29/2022).
- (94) Zhang, S.; Tong, H.; Xu, J.; Maciejewski, R. Graph convolutional networks: a comprehensive review. *Computational Social Networks* **2019**, 6 (1), 1–23.
- (95) Lipton, Z. C.; Berkowitz, J.; Elkan, C. A critical review of recurrent neural networks for sequence learning. *arXiv (Machine Learning)*, October 17, **2015**, 1506.00019, ver. 4. DOI: [10.48550/arXiv.1506.00019](https://doi.org/10.48550/arXiv.1506.00019) (accessed 9/29/2022).
- (96) Lin, T.-S.; Coley, C. W.; Mochigase, H.; Beech, H. K.; Wang, W.; Wang, Z.; Woods, E.; Craig, S. L.; Johnson, J. A.; Kalow, J. A.; et al. BigSMILES: A Structurally-Based Line Notation for Describing Macromolecules. *ACS Central Science* **2019**, 5 (9), 1523–1531.
- (97) Patel, R. A.; Borca, C. H.; Webb, M. A. Featurization strategies for polymer sequence or composition design by machine learning. *Molecular Systems Design & Engineering* **2022**, 7 (6), 661–676.
- (98) Magar, R.; Wang, Y.; Lorsung, C.; Liang, C.; Ramasubramanian, H.; Li, P.; Farimani, A. B. AugLiChem: Data Augmentation Library of Chemical Structures for Machine Learning. *arXiv (Machine Learning)*, December 1, **2021**, 2111.15112, ver. 2. DOI: [10.48550/arXiv.2111.15112](https://doi.org/10.48550/arXiv.2111.15112) (accessed 9/29/2022).
- (99) Brown, J. B. Classifiers and their Metrics Quantified. *Molecular Informatics* **2018**, 37 (1–2), 1700127.
- (100) Yang, L.; Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* **2020**, 415, 295–316.
- (101) Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016; pp 785–794.
- (102) Ribeiro, M. T.; Singh, S.; Guestrin, C. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016; pp 1135–1144.
- (103) Molnar, C. *Interpretable Machine Learning: A Guide for Making Black Box Models Explainable*, 2nd ed.; Self-Published, 2022.
- (104) Lundberg, S. M.; Lee, S.-I. A unified approach to interpreting model predictions. *Advances in neural information processing systems* **2017**, 30, 4768–4777.
- (105) Burroughs, L.; Amer, M. H.; Vassey, M.; Koch, B.; Figueredo, G. P.; Mukonoweshuro, B.; Mikulskis, P.; Vasilevich, A.; Vermeulen, S.; Dryden, I. L.; et al. Discovery of synergistic material-topography combinations to achieve immunomodulatory osteoinductive biomaterials using a novel in vitro screening method: The ChemoTopoChip. *Biomaterials* **2021**, 271, 120740.
- (106) Wei, Y.; Wu, J.; Wu, Y.; Liu, H.; Meng, F.; Liu, Q.; Midgley, A. C.; Zhang, X.; Qi, T.; Kang, H.; et al. Prediction and Design of Nanozymes using Explainable Machine Learning. *Adv. Mater.* **2022**, 34 (27), 2201736.
- (107) Reis, M.; Gusev, F.; Taylor, N. G.; Chung, S. H.; Verber, M. D.; Lee, Y. Z.; Isayev, O.; Leibfarth, F. A. Machine-Learning-Guided Discovery of 19F MRI Agents Enabled by Automated Copolymer Synthesis. *J. Am. Chem. Soc.* **2021**, 143 (42), 17677–17689.

Recommended by ACS

Data-Driven Prediction of Janus/Core–Shell Morphology in Polymer Particles: A Machine-Learning Approach

Bahareh Esteki, ChangKyoo Yoo, *et al.*

MARCH 30, 2023

LANGMUIR

READ 

Community Resource for Innovation in Polymer Technology (CRIPT): A Scalable Polymer Material Data Structure

Dylan J. Walsh, Bradley D. Olsen, *et al.*

FEBRUARY 20, 2023

ACS CENTRAL SCIENCE

READ 

Multitask Neural Network for Mapping the Glass Transition and Melting Temperature Space of Homo- and Co-Polyhydroxyalkanoates Using σ_{Profiles} Molecular Inputs

Abir Boublia, Inas M. AlNashef, *et al.*

DECEMBER 23, 2022

ACS SUSTAINABLE CHEMISTRY & ENGINEERING

READ 

Machine Learning with Enormous “Synthetic” Data Sets: Predicting Glass Transition Temperature of Polyimides Using Graph Convolutional Neural Networks

Igor V. Volgin, Sergey V. Lyulin, *et al.*

NOVEMBER 17, 2022

ACS OMEGA

READ 

Get More Suggestions >