Table of Contents

[1 Chapter 4 – Introduction to Machine Learning 2](#_Toc116909684)

[1.1 Machine Learning Basics 2](#_Toc116909685)

[1.1.1 Bayes theorem and maximum likelihood estimation 2](#_Toc116909686)

[1.1.2 Bias-variance tradeoff and model complexity 4](#_Toc116909687)

[1.2 Supervised Machine Learning 7](#_Toc116909688)

[1.2.1 Regression 7](#_Toc116909689)

[1.2.1.1 Linear and multivariate regression 7](#_Toc116909690)

[1.2.1.2 Ridge and LASSO regression 8](#_Toc116909691)

[1.2.2 Classification 10](#_Toc116909692)

[1.2.2.1 Support Vector Machine (SVM) 10](#_Toc116909693)

[1.2.3 Models for both regression and classification 11](#_Toc116909694)

[1.2.3.1 Decision Tree 11](#_Toc116909695)

[1.2.3.2 Random Forest 12](#_Toc116909696)

[1.2.3.3 Neural Networks 13](#_Toc116909697)

[1.2.3.3.1 Perceptrons 14](#_Toc116909698)

[1.2.3.3.2 Multilayer perceptron 15](#_Toc116909699)

[1.2.3.3.3 Generative Adversarial Networks 16](#_Toc116909700)

[1.2.4 Interpretability versus effectiveness 17](#_Toc116909701)

[1.2.5 Metrics 18](#_Toc116909702)

[1.2.5.1 Regression metrics 18](#_Toc116909703)

[1.2.5.2 Classification metrics 18](#_Toc116909704)

[1.2.5.3 Cross validation 19](#_Toc116909705)

[1.2.6 Uncertainty estimation 19](#_Toc116909706)

[1.2.6.1 Aleatoric versus epistemic uncertainty 19](#_Toc116909707)

[1.2.6.2 Models that incorporate uncertainty 20](#_Toc116909708)

[1.2.6.2.1 Gaussian Process 20](#_Toc116909709)

[1.2.6.2.2 Monte Carlo Dropout 21](#_Toc116909710)

[1.2.6.2.3 Bayesian Neural Network 21](#_Toc116909711)

[1.2.6.2.4 Mixed Density Network 22](#_Toc116909712)

[1.3 Unsupervised Machine Learning 23](#_Toc116909713)

[1.3.1 Dimensionality Reduction 23](#_Toc116909714)

[1.3.1.1 The curse of dimensionality 23](#_Toc116909715)

[1.3.1.2 Linear transformations 24](#_Toc116909716)

[1.3.1.2.1 Principal Component Analysis (PCA) 24](#_Toc116909717)

[1.3.1.2.2 Singular Value Decomposition (SVD) 24](#_Toc116909718)

[1.3.1.2.3 Nonnegative matrix factorization (NMF) 25](#_Toc116909719)

[1.3.1.3 Nonlinear transformations 25](#_Toc116909720)

[1.3.1.3.1 Variational Autoencoder (VAE) 26](#_Toc116909721)

[1.3.1.3.2 t-distributed Stochastic Neighbor Embedding (t-SNE) 27](#_Toc116909722)

[1.3.1.3.3 Uniform Manifold Approximation and Projection (UMAP) 28](#_Toc116909723)

[1.3.2 Clustering 28](#_Toc116909724)

[1.3.2.1.1 K-means 29](#_Toc116909725)

[1.3.2.1.2 K-nearest neighbors 29](#_Toc116909726)

[1.3.2.1.3 Density-based clustering 30](#_Toc116909727)

[1.3.2.1.4 Agglomerative hierarchical clustering 30](#_Toc116909728)

[1.3.3 Feature Selection 31](#_Toc116909729)

[1.3.3.1 Metrics 31](#_Toc116909730)

[1.3.3.2 Recursive Feature Elimination (RFE) 32](#_Toc116909731)

[1.4 References 34](#_Toc116909732)

# Chapter 4 – Introduction to Machine Learning

Machine learning (ML) has seen recent explosions in popularity and applications, catalyzed by advances in both models and accessibility of large data sets. In the most general sense, ML can be described as data-driven pattern recognition by performing linear decomposition onto nonlinear transformations; these transformations are often called *basis functions*. ML can be divided into three main categories: (1) supervised machine learning, (2) unsupervised machine learning, and (3) reinforcement learning.

Here, I will focus on supervised and unsupervised learning, which are often discussed together, as reinforcement learning is an exciting and complicated field unto itself. This chapter will give a brief overview of the most popular supervised and unsupervised methods as well as give context to the more specialized methods used in later chapters.

## Machine Learning Basics

In this section, I will cover the basics of machine learning. However, it is helpful to start with a statistical approach to model generation, as machine learning models and algorithms often pull from both frequentist and Bayesian approaches.

### Bayes theorem and maximum likelihood estimation

It is helpful to first consider the case where we observe a random variable, or a sampling from some inherent probability distribution. The goal of our model is to then use these observations to predict new outcomes based on initial conditions. To accurately do so, we want our model, characterized by the *parameters* , to have the best given our *data*, which is comprised of target variables and input parameters .

It is helpful to approach this problem from a Bayesian point of view using Bayes theorem, which states . Rephrased in words, this equation states the posterierior equals the likelihood times the prior, divided by the evidence. Let’s say we have no prior knowledge, and we can normalize the distribution later, so we can ignore the prior and evidence, respectively. Thus, the likelihood represents the probability of observing the data given the model parameters. Maximizing this likelihood, called *Maximum Likelihood Estimation (MLE)*, is a frequentist approach but is easily motivated by Bayesian inference (if using naive or uniform priors). However, if you have prior knowledge about the form of the solution, then you would want to use the purely Bayesian approach to solving this problem, called *Maximum a Posteriori (MAP)*.

Thus, given a set of observations for , the likelihood we want to maximize is given by . However, the maximum of any monotonic function is the same as the maximum of the log of that function. Thus, we can take the log of the entire equation and use log rules to transform this *cost, or objective, function* into . Here, we can start to make assumptions about the form of the probability distribution. If we assume is normally distributed around y, then we get an ordinary least squares solution (OLS). This is also equivalent to chi-square minimization. However, again, this is only true for Gaussian noise.

### Bias-variance tradeoff and model complexity

Assuming your data is being pulled from some inherent distribution and is thus a random variable, the expected error in your fit, i.e., , can be decomposed into three terms – the irreducible error, the bias-squared terms, and the variance term. The irreducible error comes from the noise in the data – here, we assume your target data y can be represented by , where is a deterministic function that depends on the input parameters and model parameters (or weights) , added to random fluctuations via , which we can model as , or sampling from a Gaussian distribution. [Bishop, #195]

The bias-squared term is the error, or deviation, of the data points from the model’s predictions, while the variance term represents the deviation of predictions from the real answer with each new draw (or realization) of the random variable. Essentially, the bias-squared term represents error in a single sample, while the variance represents the error from multiple samples. As can be expected, the bias-squared and variance terms come into play when choosing the complexity of the model.

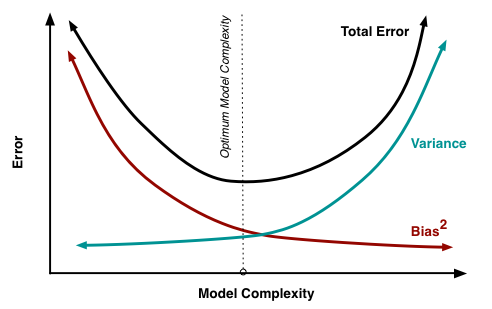


Fig. 1 Model complexity and total error – the bias versus variance tradeoff.

The model complexity, which can easily be seen as the number of hyperparameters or the degree of the polynomial we choose to fit, can be determined by looking at a similar figure as the one shown in Fig. 1. Here, we have plotted the total error (the sum of the irreducible, bias-squared, and variance terms) versus model complexity. A simple model might predict every point as the mean of all points; thus, the errors draw to draw will be similar (low variance) but within a draw might be high (high variance). On the other hand, a complex model might perfectly predict every data point it has seen (low bias) but will fail in regions where the model has not seen data before (high variance error). Thus, the goldilocks complexity occurs in the minimum of the total error, which represents the best balance between the bias-squared error and the variance error term.

The reason why bias-variance tradeoff is important is because it indicates how generalizable your predictions are when predicting a new draw of your random variable. Fig. 2 shows how the complex models with a high variance and low bias will overfit, meaning that even though accuracy on the training data is great, the accuracy on new or test data is bad. Underfitting occurs when simple models can’t predict well (training and test accuracies are both the same, but low). Ideally, both the training and test accuracies are high and the same, which occurs when a good balance between both error terms is met.

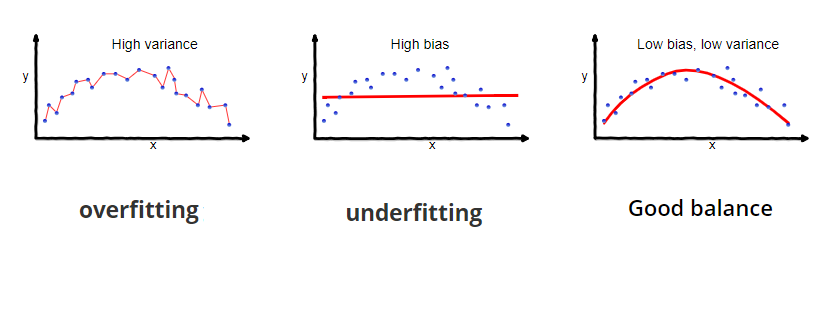


Fig. 2 The bias-variance tradeoff impacts the generalizability of the model.

To deal with this overfitting problem, you often have a validation set during model training. Throughout training, you check results on a dataset the model has not seen before (and is therefore not training data) but is not in the test set. The test set should only be used as a final metric and should not impact any decisions about model architecture; otherwise, predictions become biased, and the test set is no longer an accurate metric for the generalizability of your model. People often use a validation set to determine hyperparameters, model complexity, or any other training decision before a final evaluation.

## Supervised Machine Learning

### Regression

Oftentimes, people want a quantitative prediction of a target variable, or output, given a set of inputs, or features. Predicting a quantitative target is called *regression* and is one of the two types of supervised machine learning.

#### Linear and multivariate regression

Again, let’s say we have a target variable , where is a deterministic function that depends on the input parameters and model parameters (or weights) . We incorporate random fluctuations via , which we can model as , or Gaussian noise. The analytic solution to finding the optimal weights, or , is , where is a matrix composing of all observed . This solution, a line or hyperplane that can be visualized in Fig. 3, is thus determined by all the input and output variables and is linear because it is purely matrix multiplication and inversion.

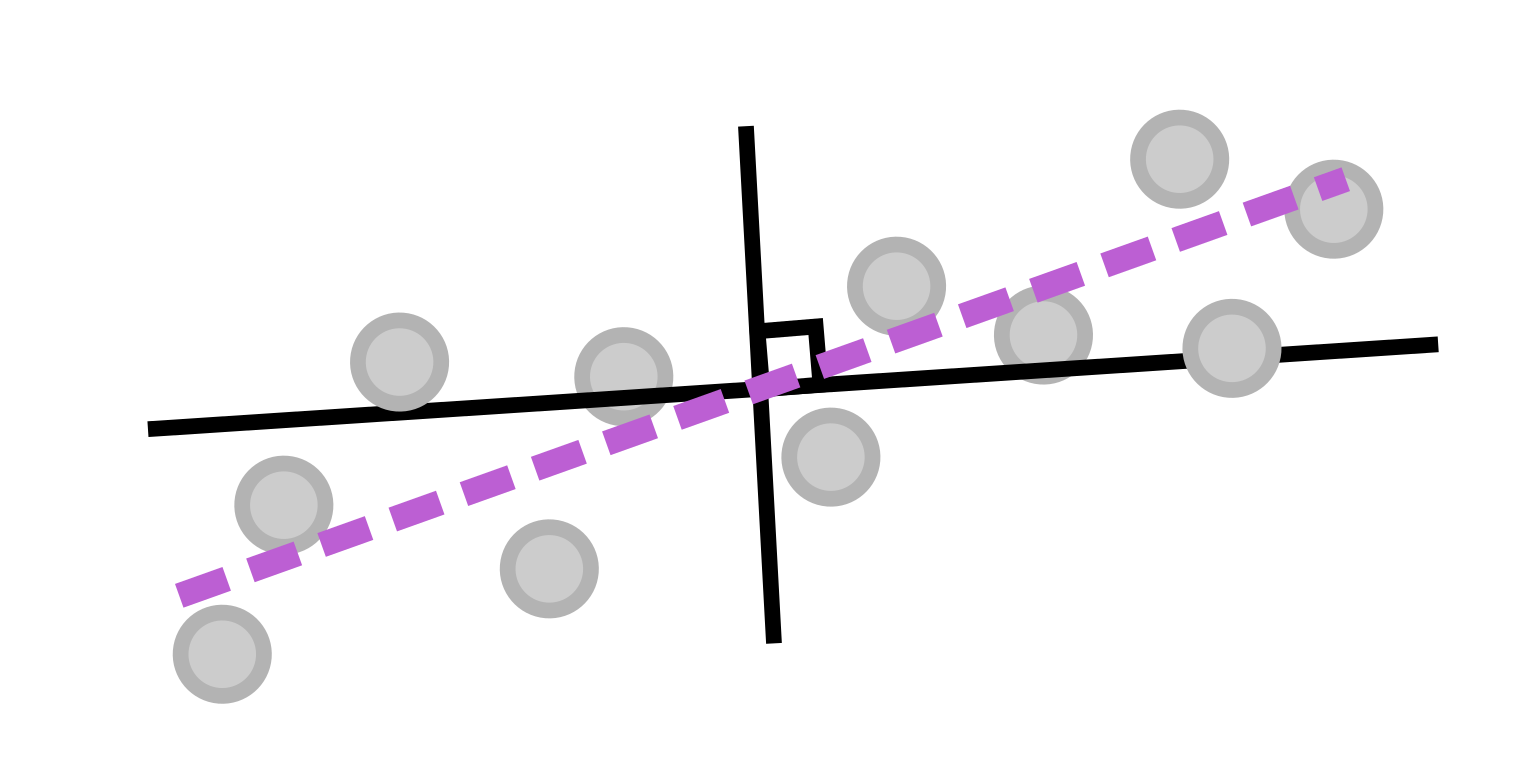


Fig. 3 Linear regression is finding a line (or hyperplane) that most follows the linear trends in the data, and as a bonus, it has an analytical solution.

Multivariate regression is the same as linear regression, except when the output variable () is multidimensional (and thus becomes a matrix Y). Thus, the epsilon term becomes a matrix (instead of a vector) and the weight vector becomes a matrix W as well. However, the solution looks the same – .

#### Ridge and LASSO regression

Including a regularization term in your fits, i.e., penalizing high weights by including a new term in your objective function (which is essentially a Lagrangian you are trying to minimize), has been shown to have better generalizability. First, a general norm looks like , where p identifies the type of norm. Thus, to add a regularization term to your objective function would results in something like , where the second term (the regularization term) is modified in scale by the Lagrange multiplier. Setting p to be two is called the L2 norm, which represents the standard Euclidean distance, and is called *ridge regression*. Setting p to be one is called the L1 norm, which is essentially taking the absolute value, and is called LASSO (Least Absolute Shrinkage and Selection Operator) regression.

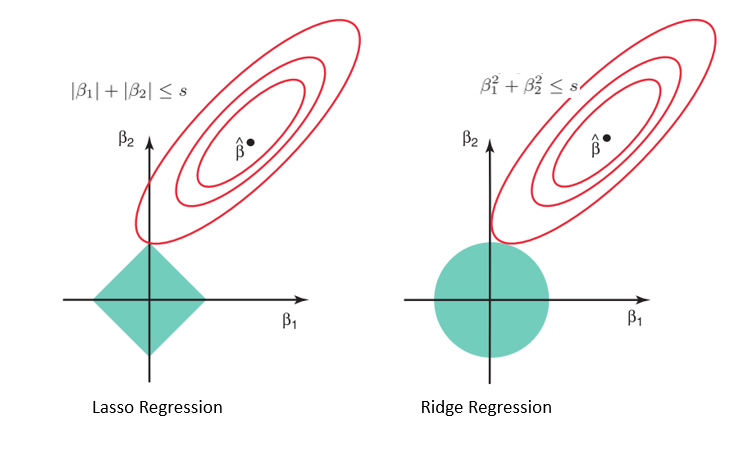


Fig. 4 The regularization term affects the sparsity of the solution.

The type of regularization is important because it impacts the sparsity of the solution. For example, let’s say you have two features, or input variables, and thus your weight vector is also two-dimensional. The equipotential surfaces (in other words, solutions where the overall loss (given by the norm) is the same) look like the green shapes in Fig. 4 – LASSO gives diamond shaped surfaces while Ridge gives circular surfaces. Solutions to the first term (i.e., the data-driven term) are represented by the concentric ellipses in red. All these solutions (i.e., any combination of the two weight components that fall along one of those red ellipses) may equally explain the data, but the individual elements of the weight vector will be different.

An overall solution that minimizes the regularization term can be found by slowly growing the green equipotential surface from the origin until it intersects with one of the red curves. By doing so, the L1 norm will more likely converge on a solution with sparse solutions (or solutions along one axis of the weight vector) because most of the volume is distributed along the axis, whereas the Ridge equipotential surfaces jut out in all directions.

### Classification

Instead of learning a (multidimensional) line that follows the trends in the data as in regression, classification finds a *decision boundary*, which is often a line, or in higher dimensions, a (hyper)plane, that best separates data points. Thus, classification predicts categorical targets.

#### Support Vector Machine (SVM)

Support vector machines (SVMs) generate \*unique\* decision boundaries by including some leeway, or margin, in the fit in which they try to maximize. This unique solution is called the maximum-margin hyperplane. A demonstration of a SVM is seen in Fig. 5. Note that, in order to maximize the margin, the width of which is given by 2 / ||w||, one must minimize w. Because the solution is completely determined by points on or within the margin (indicated by the points on the dashed line in Fig. 5), these points are called *support vectors*. Moreover, the SVM activation function can be modified to be soft, meaning it can allow for a few outliers to either get close or cross the decision boundary, a beneficial property for nonlinearly separable data. [Russell, 2010 #229]

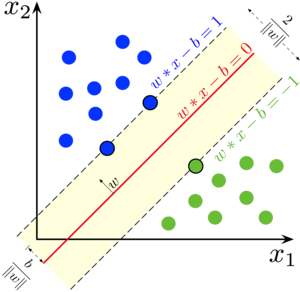


Fig. 5 The SVM maximizes the distance from the decision boundary between all data points **and** it maximizes the margin around this decision boundary. The margin ensures that the solution is unique.

### Models for both regression and classification

Most machine learning models can be adapted for both types of supervised machine learning. The next section will give an overview of the most popular models.

#### Decision Tree

Decision trees are easy to interpret, which is one of their biggest strengths, but decision trees easily overfit, which is their biggest downfall, because they look for the strongest correlations between input and output features. However, correlation does not necessarily mean causation, so having a good representative set of features is critical when using a decision tree. [Russell, 2010 #229] A diagram of a decision tree can be seen in Fig. 6.

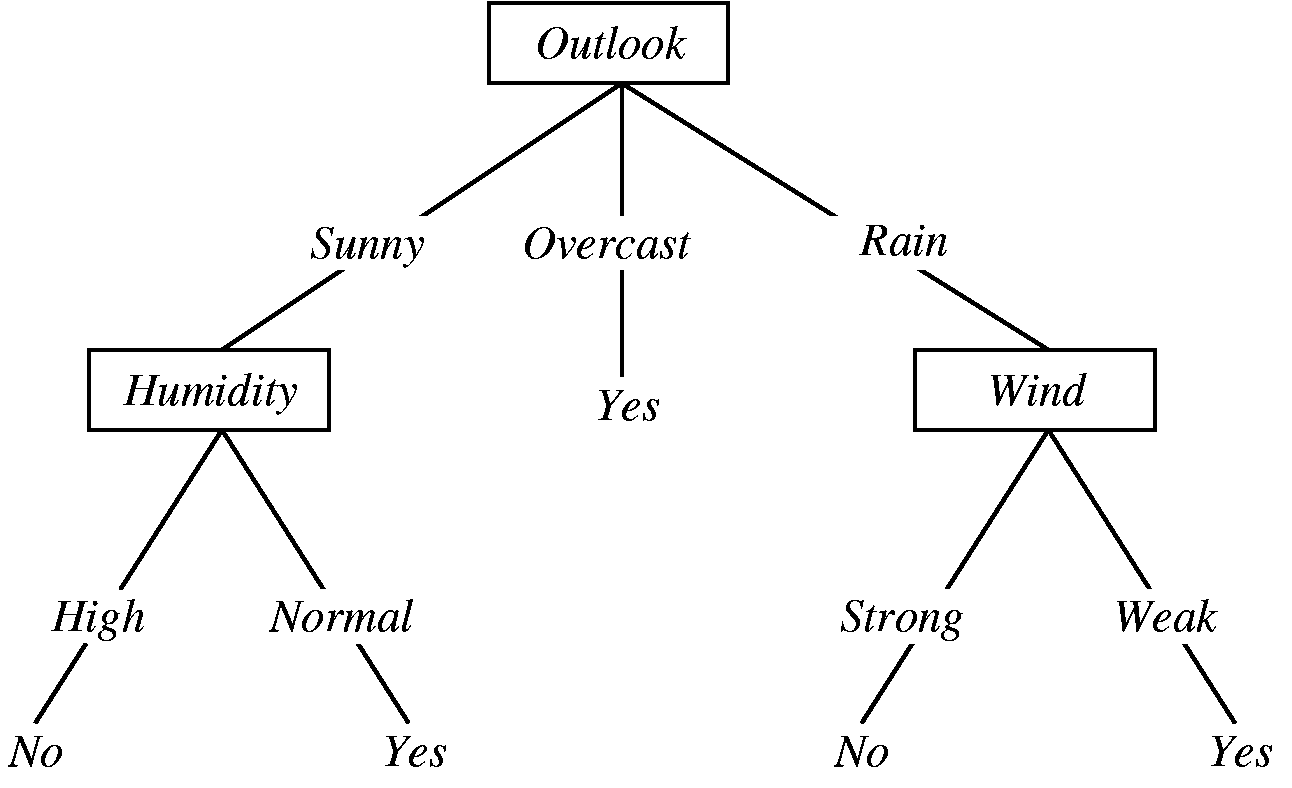


Fig. 6 A decision tree is composed of decision nodes (composed of a question about the data), branches (which depend on the answer to the question, which are typically interpreted as a “yes” or “no” answers), and leaf nodes (the target variables).

Decision trees make their branches by recursively looking for the strongest correlation between input features and the target variables. The strongest correlations appear near the top (or root) node, while further categorizations occur on lower branches. Thus, decision trees are most naturally used for classification problems.

#### Random Forest

A random forest is a collection of decision trees and gains its strength from “ensemble” learning. Essentially, random forests try to avoid the overfitting issues with decision trees by combining many models (decision trees) together and taking a majority vote of each tree. The individual trees are then “weakly” trained, meaning they are trained on different subsets of the training dataset, or on predicting different classes. [Breiman, 2001 #233]

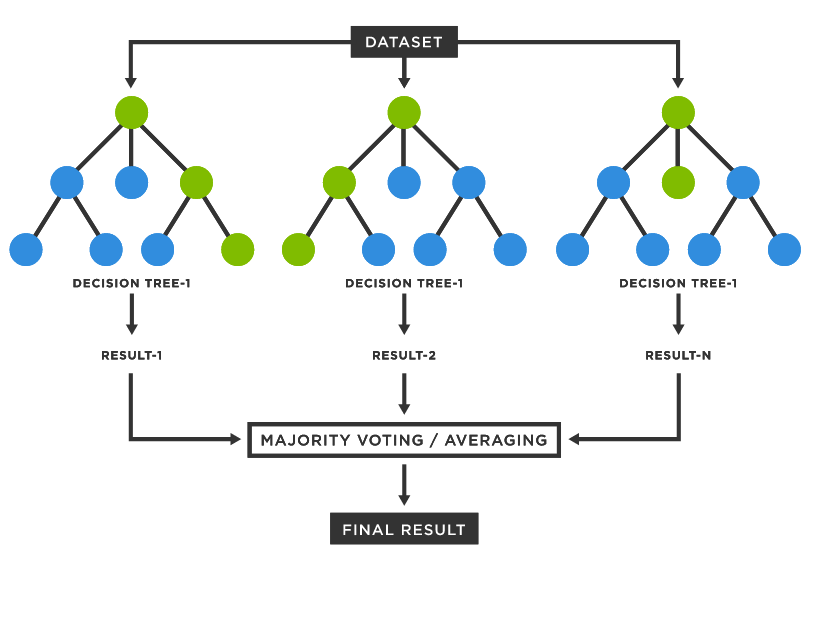


Fig. 7 A random forest is a collection of decision trees, where the overall decisions are a majority voting or averaged result from each tree.

#### Neural Networks

Neural networks are the most versatile machine learning model, and the hardest to interpret, which is why they are notoriously labeled as “black boxes.” However, their strengths are derived from the nonlinear transformations interspersed with linear transformations throughout the network. Here, nonlinear means that the output does not correlate 1:1 with the input and therefore the transformation cannot be inverted. Errors in the final predictions are then propagated throughout the model (starting from the output to the input) via *backprop*, which relies on the derivatives of the weights. However, the modern python tools use automatic differentiation, which numerically solves for the derivative (instead of finding an analytic solution), thus turning this issue into a quick and painless behind-the-scenes aspect of model training.

##### Perceptrons

The basic building block of neural networks is the perceptron. Perceptrons were originally invented in 1943 by McCulloch and Pitts [McCulloch, 1943 #230] but were horrible at generalizing predictions and thus were ignored for a long time, and instead people focused on kernel-based models. That is, until 1986, when David Rumelhart, Geoffrey Hinton, and Ronald Williams [Rumelhart, 1986 #231] had the idea of combining them in “layers” and machine learning algorithms and their applications exploded.

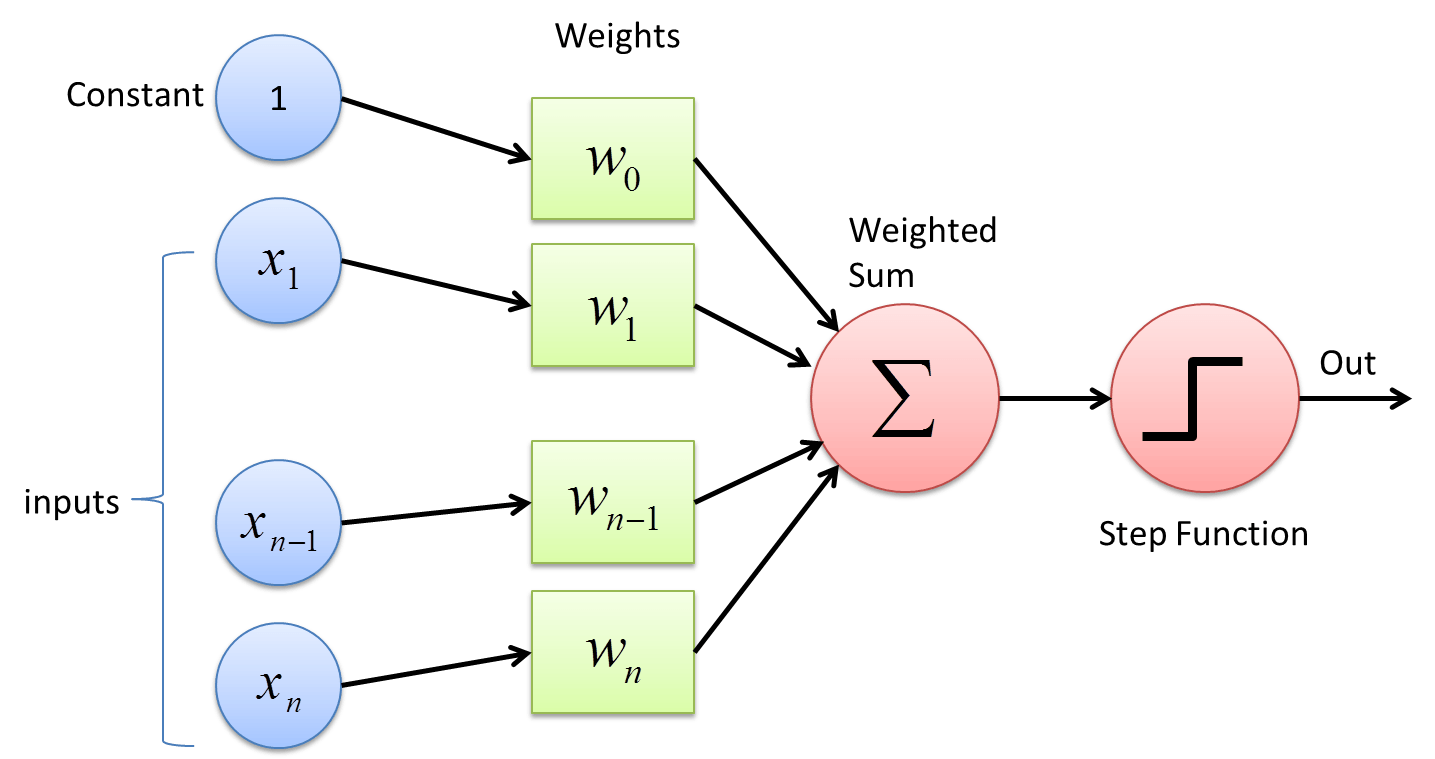


Fig. 8 The composition of a perceptron – linear combination of inputs via a weight vector which then get summed and passed to a nonlinear activation function.

Perceptrons are composed of four different steps: 1. Input, 2. weights, 3. Summation, and 4. activation function. The first three steps are the linear transformation, where inputs are weighted and summed together (the weight vector is what is “learned”). Then, that summation is passed to a nonlinear activation function. Activation functions that are commonly used are sigmoid, tanh, ReLu, step, and hinge. All these activation functions are zero for negative values and then (typically) move towards one around the origin (when the input is around zero). Activation functions like sigmoid, tanh, and the step function max out at one, making them good for output layers (where they could be interpreted as a probability, for example).

##### Multilayer perceptron

As mentioned earlier, perceptrons strung together in layers make for a powerful model and were thus called multilayer perceptrons (MLPs). MLPs are just another name for your standard neural networks. [Russell, 2010 #229] A diagram of this model can be seen in Fig. 9.

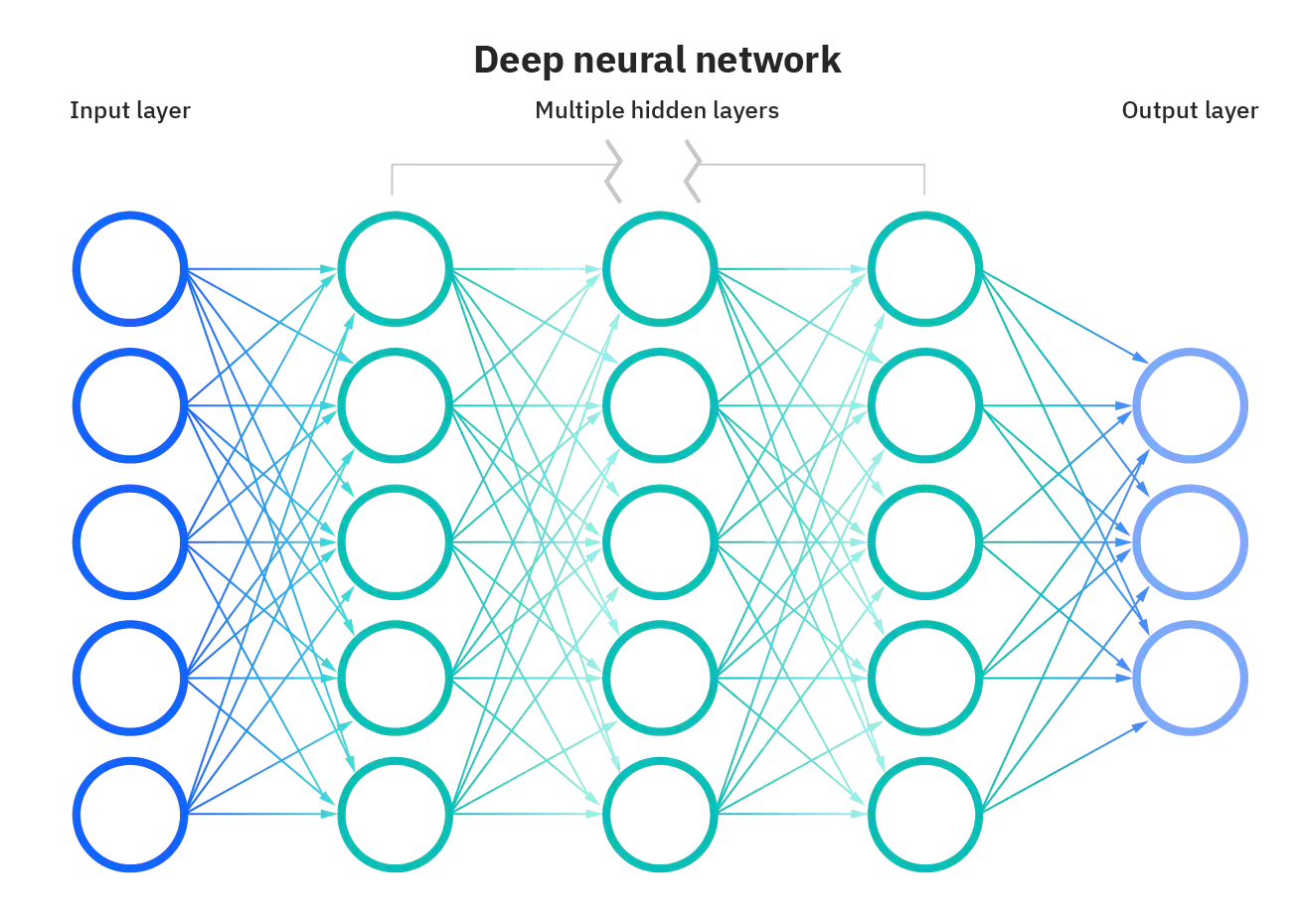


Fig. 9 Diagram of a fully connected MLP, or standard neural network, with three hidden layers.

In an MLP, each input node corresponds to an input feature. Then, the inputs are typically passed to any number of hidden layers, or layers that are neither input nor output layers, where each input goes to every node in the next layer, and every node in the next layer receives every input (at least for a fully connected model). Finally, the weights are passed to an output layer, where the dimension of the target variables (often) dictates how many output nodes there are. A deep neural network is simply an MLP with many hidden layers.

##### Generative Adversarial Networks

Neural networks start to get fancy very quickly. There are recurrent neural networks, transformer neural networks, and convolutional neural networks. One interesting network of note is generative adversarial networks (GANs). [Ian J. Goodfellow, 2014 #234] GANs generate new data by having one model that creates data, seeded with random noise, called a generator. A parallel model, called the discriminator, compares real samples to the generated, or fake, samples and the entire model is penalized if the discriminator can distinguish between real and fake inputs. Obviously, this process requires an intricate balance between training the generator and discriminator, and many modifications have been made to encourage this, such as using the Wasserstein distance metric in a Wasserstein GAN (WGAN).

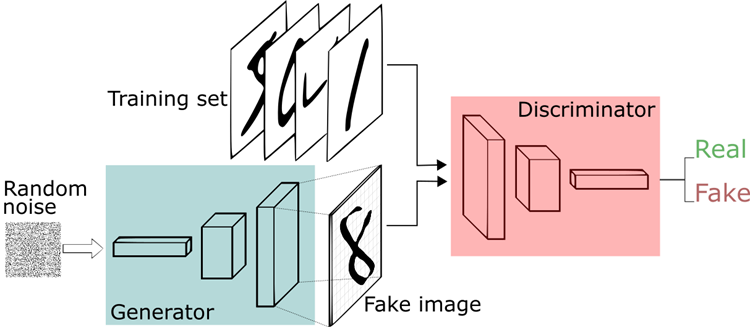


Fig. 10 A generative adversarial network (GAN) can make new realistic looking samples by balancing a discriminator and generator during training.

### Interpretability versus effectiveness

In general, deciding what model to use depends on the problem, but also on the desired interpretability of the results. While linear regression is the most limited in its scope, only working well for linear data, it is the most interpretable as the importance of features can be directly interpreted using the learned weights. Decision trees and SVMs are likely easy to interpret. The second tier in interpretability includes ensemble models, such as a random forest, as the majority voting hides the interpretation behind another layer. Finally, neural networks are the hardest to interpret as the vast amount of weights and layers make it difficult to correlate features to specific outcomes.

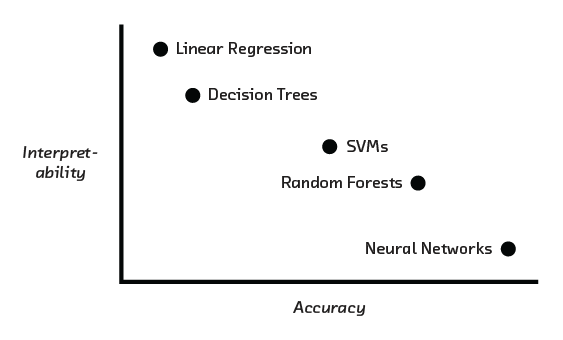


Fig. 11 Summary of the interpretability versus accuracy (or strength) of each machine learning model.

### Metrics

#### Regression metrics

Regression metrics quantitatively compare predictions to true values. The most common are mean squared error (MSE), root mean squared error (RMSE), and mean average error (MAE). A good way to visualize regression predictions is by making a correlation curve, or plotting true (target) values versus predictions, where a perfect model would fall along the y = x line.

#### Classification metrics

Reporting an appropriate classifications metric is critical as there are often drastically different interpretations of results depending on the metric you use; for example, it is more important to identify the probability of false negatives for cancer detection that it is to purely report accuracy. Here are some common metrics to consider.

Accuracy is the sum of true positives and true negatives divided by the total number of predictions. Accuracy is a good metric where distinguishing between false positives (Type 1 errors) and false negatives (Type 2 errors) does not matter. A good way to visualize all predictions is by forming a confusion matrix. Other metrics include precision (true positive divided by true positive plus false positive) and recall (true positive divided by true positive plus false negative). An important metric that is often reported is the F1 score as it combines precision and recall. It is defined as precision times recall dived by the sum of precision and recall. This metric punishes extremes values more and places importance on false positives and false negatives.

#### Cross validation

As stated earlier, cross validation is super important for determining any model hyperparameters and gauging the generalizability of your model. The most basic type of cross validation is k-fold cross validation. Here, the training data is split into k different chunks, where the model will be trained on k – 1 of the sections, and the last section will be used as a validation set. Throughout training repetitions, often called epochs in the case of neural networks, the validation set will systematically move between every k section. Then, an average and standard deviation can be calculated using the k predictions on the different validation sets.

### Uncertainty estimation

There are a handful of ways to formally estimate uncertainty of predictions from machine learning models (which is essential for scientific interpretation) that are better than just rerunning the model and seeing how much its predictions change.

#### Aleatoric versus epistemic uncertainty

First, let’s understand the basics of uncertainty from a statistical perspective. Uncertainty can be divided into two different types – aleatoric and epistemic. Aleatoric uncertainty is from the internal randomness of phenomena while epistemic uncertainty is from the lack of knowledge of the system, i.e., hidden variables in your system that may be affecting the outcome of the event. [Hüllermeier, 2021 #235]

#### Models that incorporate uncertainty

The following are a few models used in machine learning that formally incorporate uncertainty into their predictions.

##### Gaussian Process

The first model to discuss is a Gaussian process (GP). GPs are non-parametric kernel-based methods that incorporate Bayes rule into their predictions. [Rasmussen, 2006 #145] Thus, they can give an estimate of uncertainty, or conversely confidence, to every output. A GP performs well on smoothly varying data, such as a multidimensional gaussian, but poorly on discontinuous or sharply featured data, such as a step function.

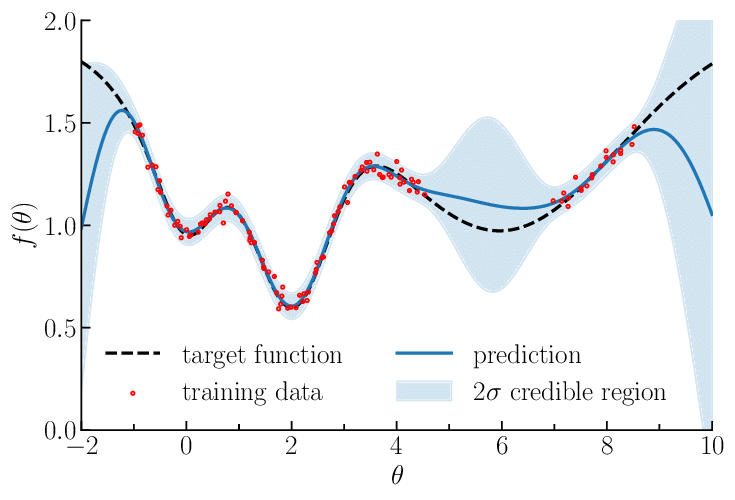


Fig. 12 A Gaussian process gives estimates of uncertainty depending on the location of the training data.

##### Monte Carlo Dropout

The second and most common way to estimate uncertainty of a prediction from a neural network is by using dropout during predictions. Dropout means randomly setting weights in the network to zero, thus randomly canceling out some correlations. [Yarin Gal, 2016 #236] This process has been shown to estimate Bayesian uncertainty, and it is the easiest to implement as it does not require any architecture modifications.

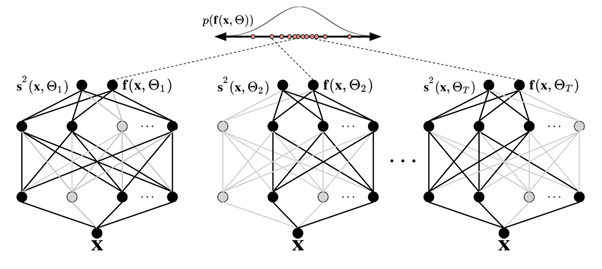


Fig. 13 Monte Carlo dropout during test predictions is the easiest and most common way to estimate uncertainty because it does not require special model architecture.

##### Bayesian Neural Network

The third way to incorporate uncertainties is to use a Bayesian neural network, which essentially learns distributions instead of weights. This difference means twice as many parameters to learn, assuming Gaussian distributions, which can be described by a mean and standard deviation. [Goan, 2020 #237]

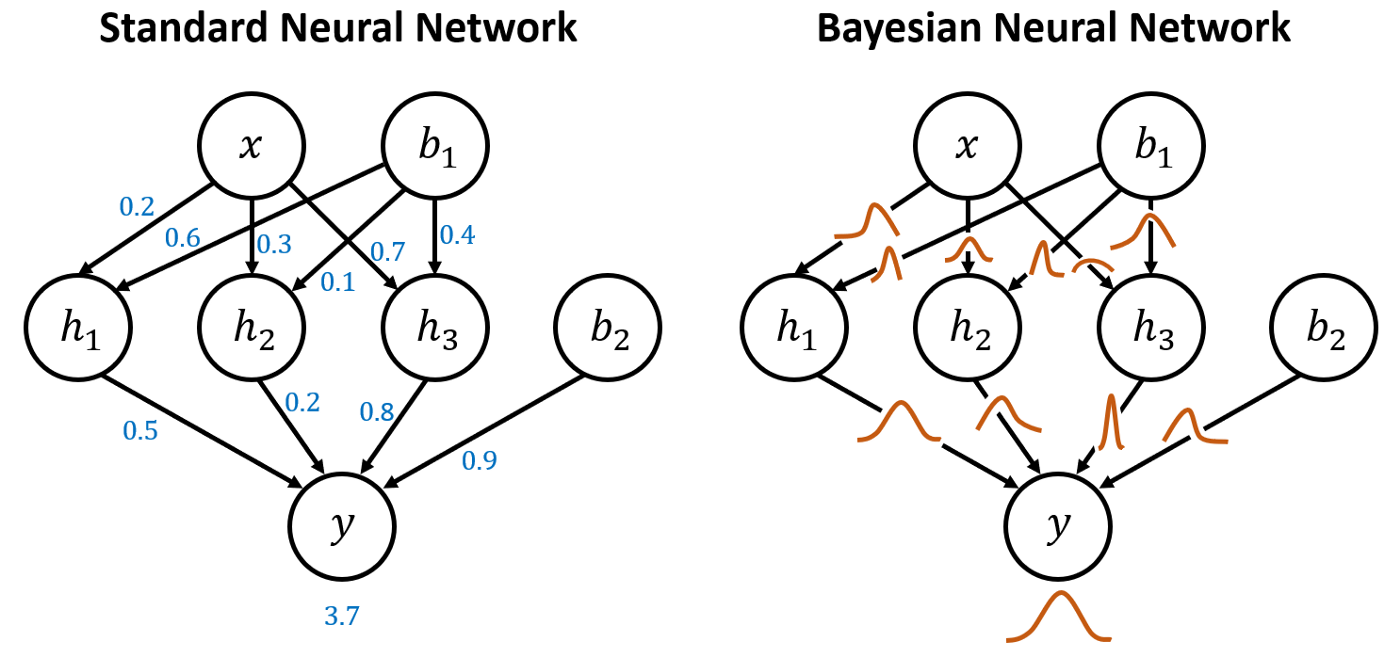


Fig. 14 A Bayesian neural network learns distributions of weights instead of just the weights themselves.

##### Mixed Density Network

Finally, the last model that incorporates uncertainty into predictions is a mixed density network. Here, the final output layer is expanded to include the parameters of the desired distributions (such as mean and standard deviation, not just the mean) and the loss function is adjusted accordingly to reflect this interpretation. [MacKay, 1995 #238]

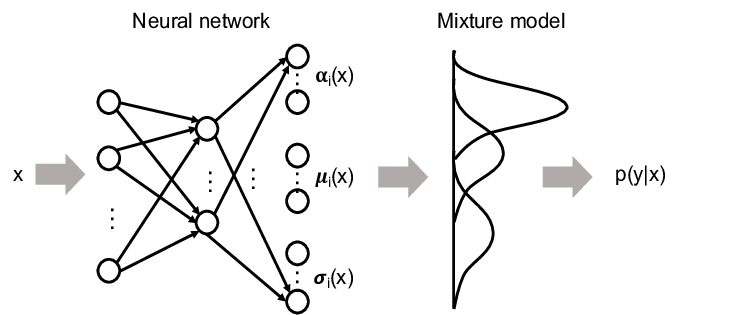


Fig. 15 A mixed density network is another easy implementation of a neural network that can formally account for uncertainty.

## Unsupervised Machine Learning

Most of my work focused on unsupervised machine learning. Unsupervised machine learning is looking for trends in data, but with no target variable. Instead, it is purely a data-driven pattern recognition technique. Often, unsupervised machine learning is used to disentangle correlated features (features selection) or look for the most important information in your data. This analysis can speed up computations or make predictions more interpretable.

### Dimensionality Reduction

#### The curse of dimensionality

The curse of dimensionality refers to the phenomenon that points that are far away in some dimensional space look even farther away in higher dimensions, following an exponential explosion. This trend means that points that might have started close now look just as far away from every other point. This phenomenon is from the r3 law, or that your Jacobian in high dimensions depends on the radius r by increasing orders of magnitude. This curse is an issue when trying to identify similarities in high-dimensional data. Thus, dimensionality reduction tries to solve this issue by finding a lower-dimensional representation of your original dataset. [Russell, 2010 #229]

#### Linear transformations

##### Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is an algorithm that finds the eigenvalue and eigenvector decomposition of a dataset such that the eigenvectors are specially ordered to be in order of importance, where importance is described as “explaining the most variance in the dataset” or, equivalently, minimizing the distance data points need to be projected onto that eigenvector. [Press, 2007 #211]

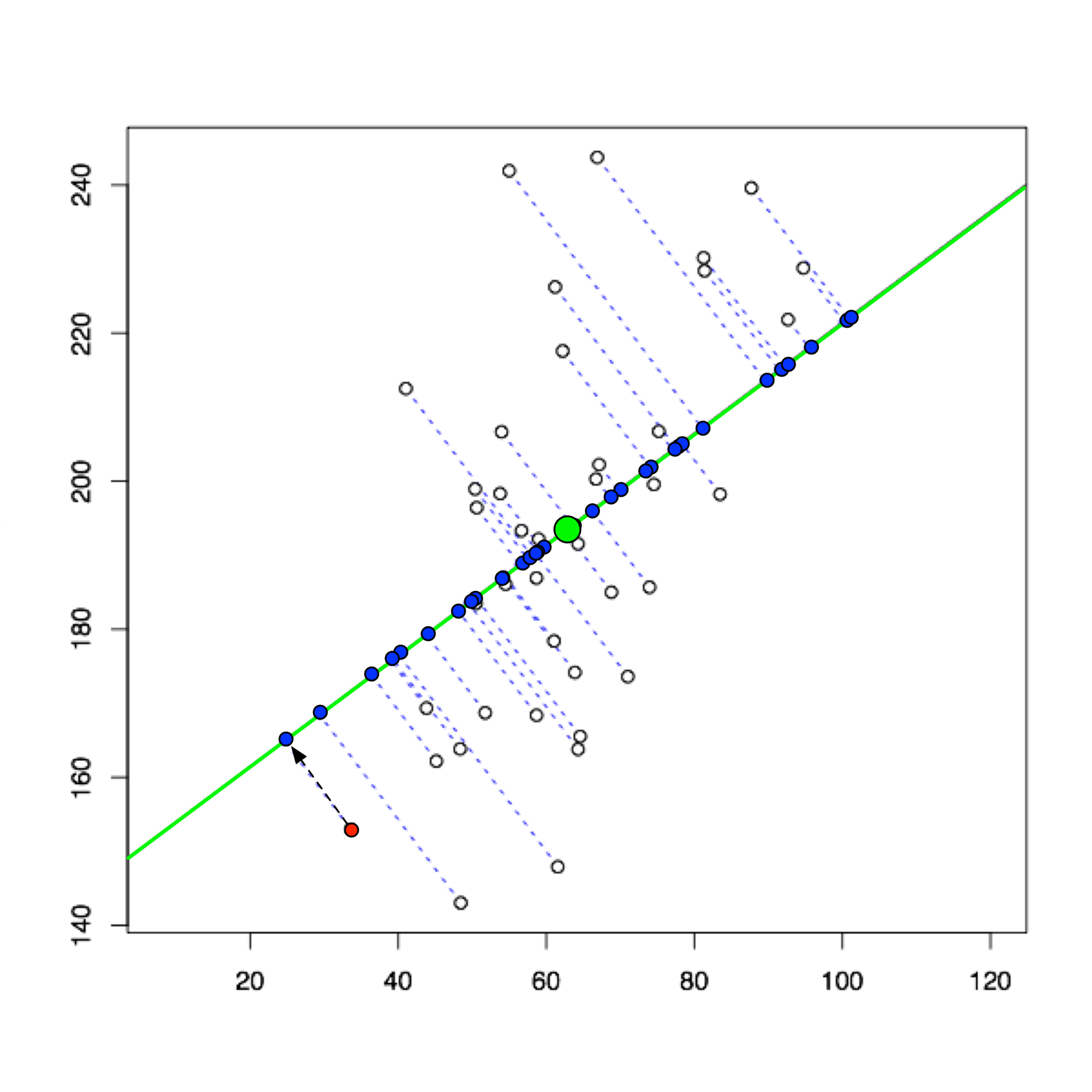


Fig. 16 PCA tried to maximize explained variance, or equivalently minimize the distances needed for the data points to be projected onto that eigenvector (or basis vector).

##### Singular Value Decomposition (SVD)

Singular value decomposition (SVS) is the more complete form of PCA and thus takes more computational power (because of a matrix inversion). [Press, #211] SVD represents the original matrix A as A = UDVT, where U is composed of left singular vectors, D is a matrix with the singular values along its diagonal, and V is composed of right singular vectors. SVD is unique up to rearranging the order of the singular values.

##### Nonnegative matrix factorization (NMF)

Nonnegative matrix factorization (NMF) is like PCA, except it forces the eigenvalues to be positive, which is beneficial for physical variables that likewise can’t be negative. [Lee, 1999 #136] NMF calculates two non-negative matrices whose product reproduces the original dataset, or A = WH, where the column vectors in W are linearly combined using the coefficients in the columns of H.

#### Nonlinear transformations

Nonlinear transformations have strengths over linear methods because they can capture nonlinear trends in data. Effectively, they aren’t constrained to basis vectors that are (hyper)planes but can have a weaving manifold to explain the data, as shown in Fig. 17a.

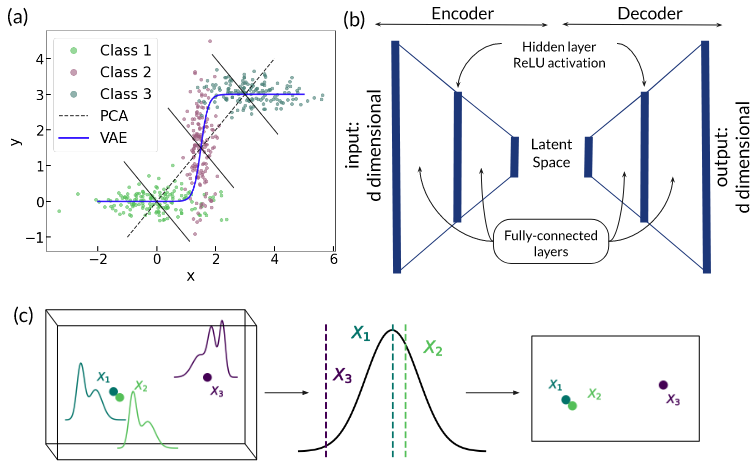


Fig. 17 The benefits of nonlinear dimensionality reductions algorithms. [Tetef, 2021 #140]

##### Variational Autoencoder (VAE)

An autoencoder, the architecture shown in Fig. 17b, is a special type of neural network that is composed of two sequential networks – an encoder and a decoder. The encoder takes data, in the original d dimensions, and reduces it to a lower dimensional representation called a *latent space*. The decoder then takes the latent space and expands the dimensionality back to the original d dimensions. By trying to reconstruct the original input in the output layer, the autoencoder iteratively learns how to maximize the information being squeezed through the bottleneck layer, or latent space.

An autoencoder can then be modified to be a variational autoencoder (VAE) by learning a distribution in the latent space instead of a deterministic embedding. This modification is done by learning two parameters for each latent space dimension (instead of one) and interpreting one as a mean and the other as a standard deviation. Then, the model will randomly vary the input going to the decoder by sampling from that learned mean and standard deviation. This property allows the latent space to be generative and thus create new data that it hasn’t seen before by interpolating through the latent space. [Hinton, 2006 #101]

##### t-distributed Stochastic Neighbor Embedding (t-SNE)

t-distributed stochastic neighbor embedding (t-SNE) [van der Maaten, 2008 #114] is a tool that is often used for visualizing data in two or three dimensions (it doesn’t work in higher dimensions) but can be used as a dimensionality reduction technique as well. t-SNE creates a graph-based similarity representative of the original data by calculating the joint conditional probability distribution between every pair of points to effectively get a similarity matrix. It then projects the data points to a lower dimension (say two) and tries to match the distances between each data point in the reduced space such that the subsequently calculated similarity matrix in the lower dimensions matches the one originally generated in the higher dimension. Ideally, this process means that there is no data compression or loss of information since the similarity between points is retained. This process is demonstrated in Fig. 17c.

However, t-SNE generates a “non-parametric” embedding, which means that it needs the entire dataset every time it makes a lower dimensional representation. There are no weight vectors it can save to do a transformation later. “Non-parametric” does not mean it has no hyperparameters. In fact, the one hyperparameter it does have is called perplexity, which represents your suspected minimum cluster size.

Moreover, t-SNE is excellent for looking at local similarities, but at the cost of loss of global structure. That means you must interpret clustering the t-SNE reduced space carefully. Data points in the same cluster can be interpreted as similar, but distances between clusters cannot tell you exactly how similar or different the two clusters are, only that they are different.

##### Uniform Manifold Approximation and Projection (UMAP)

Uniform Manifold Approximation and Projection (UMAP) [McInnes, 2020 #142] is very similar to t-SNE in that it constructs a graphic similarity representation of the original high-dimensional data and tries to match that similarity metric in the reduced dimensional embedding. However, because of a different choice in the cost function, it retains global similarity, unlike t-SNE. UMAP has two hyperparameters – the minimum distance between clusters and the average number of neighbors a data point in a cluster is expected to have. These hyperparameters let you tune how global versus local you want your similarity metric and thus how tightly your clusters appear in your reduced space.

### Clustering

Although a reduced representation of your data is beneficial in and of itself, it can be helpful to apply a clustering algorithm to that reduced space. Clustering algorithms work better on fewer dimensions exactly because of the curse of dimensionality. Moreover, clustering can help identify unbiased classes in your dataset and be a precursor to supervised machine learning.

##### K-means

K-means clustering is a centroid-based clustering algorithm, i.e., it tries to balance the center of mass among all the clusters. K-means randomly place k centroids, where k is the number of clusters, and then it moves the centroid locations until they are the furthest apart from each other while being the center of mass for the nearby data points. Thus, k-means clustering works well if you know the expected number of clusters and the data has a Gaussian distribution (no Swiss rolls).

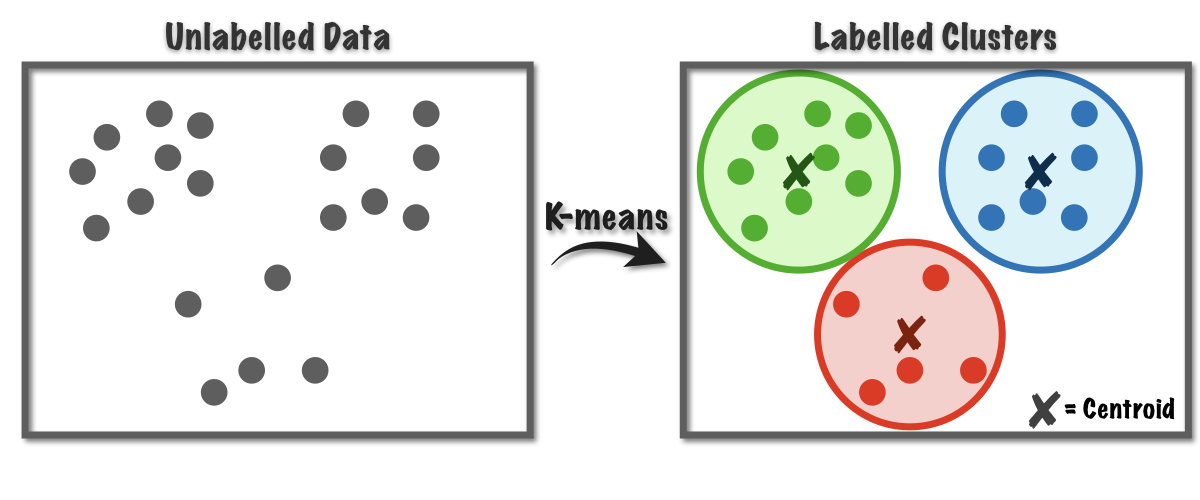


Fig. 18 K-means clustering balances center of mass.

##### K-nearest neighbors

K-nearest neighbors (KNN) [Fukunaga, 1975 #239] clustering is similar to K-means in that it is centroid-based in that it calculates the probability of a new point belonging to a class based on the nearby data points, where the hyperparameter k represents the expected number of members in each cluster. Thus, KNN is a **supervised** clustering approach, unlike most other unsupervised clustering algorithms. However, the algorithm allows clusters to be more globular or amorphous in shape.

##### Density-based clustering

Dbscan [Hahsler, 2019 #147] is a density-based clustering algorithm, which means it looks at the density of nearby data to determine whether the points belong in the same cluster or not. It has one main hyperparameter, epsilon, which is effectively the expected radius of a cluster. The algorithm will glob data points together with the expectation that almost all points in one cluster will fall within a radius of epsilon from another other point in the same cluster.

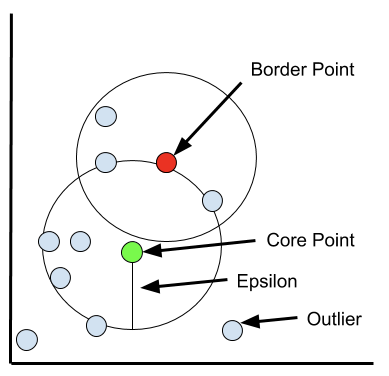


Fig. 19 dbscan uses the hyperparameter epsilon to determine the radius of an expected cluster.

##### Agglomerative hierarchical clustering

Agglomerative hierarchical clustering [Murtagh, 1983 #240] is a recursive process that groups data points together one at a time until all data points belong to the same cluster. Using a divisive algorithm does the opposite – splitting the dissimilar data points from each other until every point belongs to its own cluster. Both result in a dendrogram, a tree-like graphical representation of your data, where “cuts” can be taken horizontally across them to determine clusters.

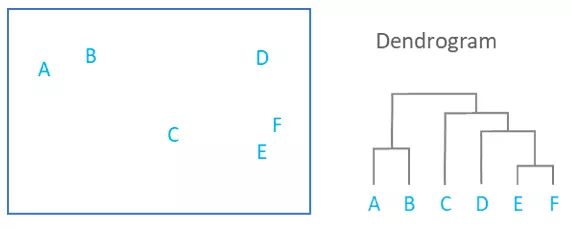


Fig. 20 Dendrograms represent similarity of points using the distances between them as a metric.

### Feature Selection

#### Metrics

Feature selection is critical when input variables are correlated as it makes models more interpretable. The type of metric one might use to perform feature selection depends on the type of input and output variables, as shown in Fig. 21. Notice that all the metrics are supervised. For categorical inputs and outputs, as is often the case in scientific research, a chi-squared metric is traditionally used. Chi-squared can be separated into three main types: tests for goodness of fit, tests for independence, and tests for homogeneity. The “goodness of fit” chi-square test is its most common usage, which determines the probability that a variable comes from a specific distribution.

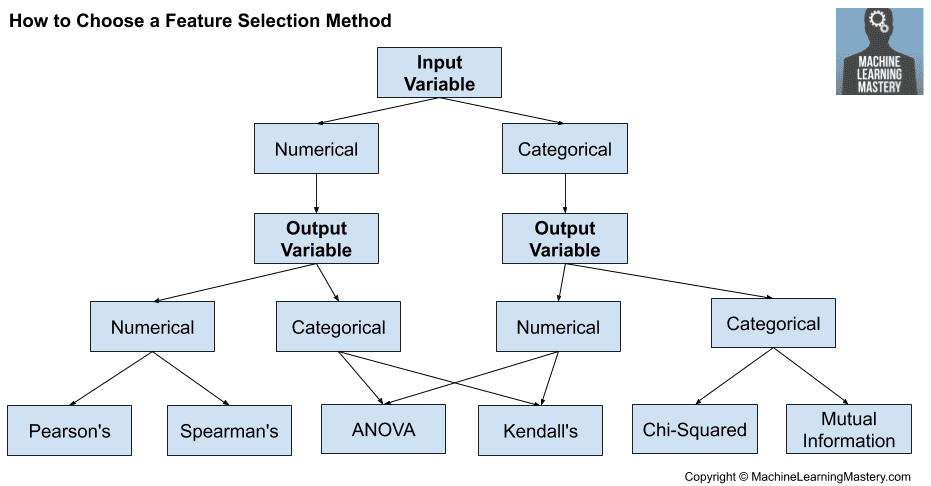


Fig. 21 Supervised feature selection metrics for different types of input of output variables.

#### Recursive Feature Elimination (RFE)

Recursive feature elimination [Jeon, 2020 #185] is a wrapper-based feature selection method that, as the name suggests, recursively prunes the input, or feature space, such that the most important features remain. The algorithm decides the importance of features by training a base machine learning model, such as a linear regressor as shown in Fig. 22, and then calculating the prediction accuracy for that set of features. To remove a feature, it will repeat this process by enumerating through every combination of N- 1 features, where N is the current size of the feature set. In each step, it removes the feature that corresponded to the lowest accuracy. The algorithm continues until a desired number of features remains.

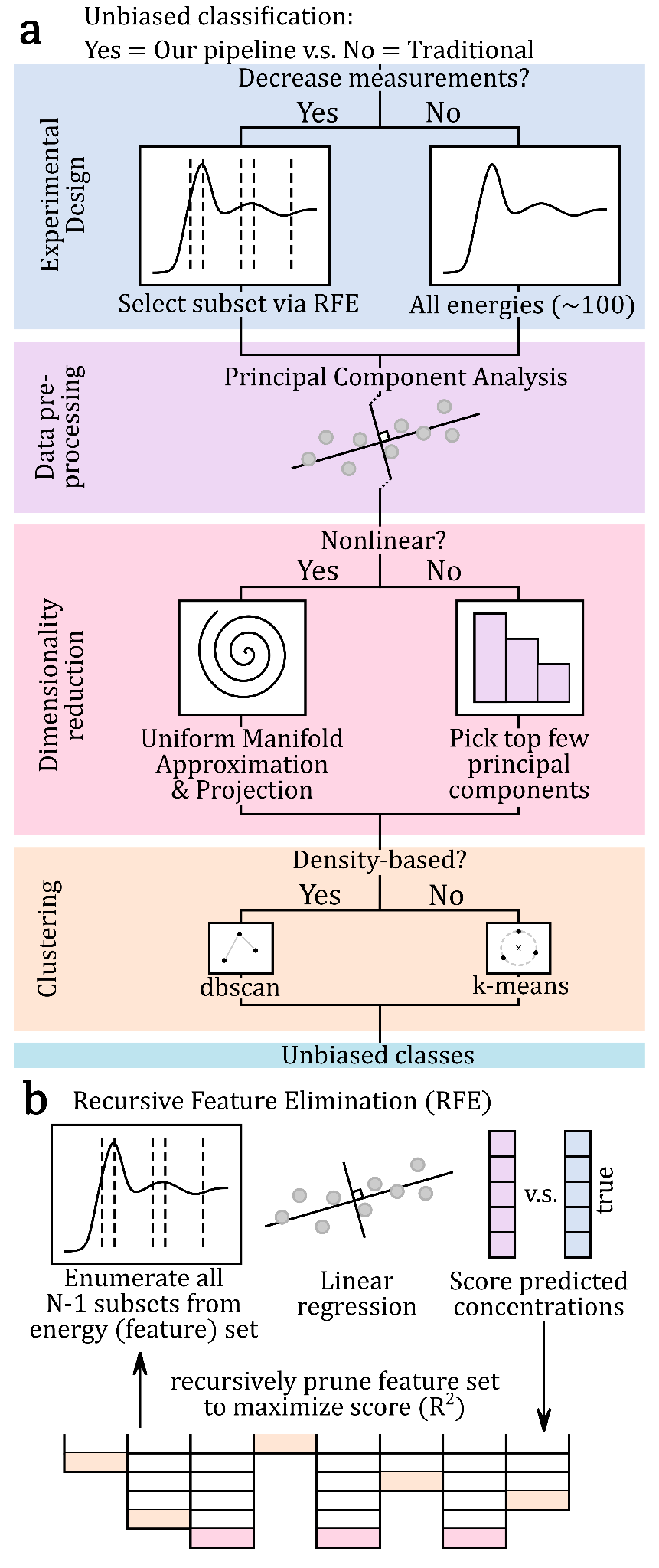


Fig. 22 Recursive feature elimination picks the best features by correlating them with the best score (e.g., accuracy) of predicting the target variables. [Tetef, 2023 #232]

## References