An Introduction to Machine Learning

Vanessa Nilsen

May 22, 2018

Contents

1 What is Machine Learning?

Machine learning is a field of computing that has recently seen an explosion in applications and household recognition. A loose definition of machine learning is using computers for tasks that they can learn from and become better at without explicit programming. Some examples of this have been particularly relevant in the medical field, where computer vision has become faster and more reliable than humans at the job. However, the ideas behind machine learning are far from new; Alan Turing, sometimes called the father of machine learning, had been talking about computers being able to improve their performance at their assigned tasks without additional programming since 1950, and John McCarthy, another frequently cited founder of the field, coined the term "artificial intelligence" in 1955. So while the rise in popularity of this powerful computing tool has been more pronounced in recent decades, the concepts have been around for over half a century.

There are also many different terms associated with this field, such as data science, statistical learning, deep learning, and others. A list of these terms, as well as some of the jargon associated with machine learning, can be found in the "Lingo and Jargon" section at the end of these notes. It may be helpful to familiarize yourself with these terms before using these notes.

2 Models in ML

Modelling in ML follows this general process:

1. Organize a data set.
2. Use the data in conjunction with an algorithm to train a model. Each algorithm has different variables that can be adjusted to produce better predictions on the data. These variables are known as *hyperparameters*.
3. Use the trained model to make predictions on the desired output variable.

3 Data Structure

Before discussing machine learning algorithms, it's helpful to understand what kind of data is required to train a model. This section will go over the way data is structured when it's used in ML applications.
Say, for example, we have some data on various alloys for which we'd like to predict the band gap:

Formula	Color	Crystallinity	T_{melt} (^oC)	Band Gap (eV)
Si	Grey	Single Crystalline	1414	1.2
GaAs	Grey	Single Crystalline	1238	1.4
CaO	White	Polycrystalline	2572	6.1
HgI_2	Red	Polycrystalline	259	2.1

Typically when using machine learning with materials data, there will be a some materials property or similar value we'd like to predict. In addition to the value we'd like to predict, we usually also have some data that has some correlation to the value we're predicting. Taking the above data as an example, we'd like to predict the band gap of a material using what we know about the chemical formula, the color of the material, the crystallinity of the sample, and the melting temperature of the material. The band gap is like the dependent variable for our analysis of this data set. When we use an algorithm to predict the band gap, the output from the model will be predictions of this variable in our data. Another term used for this portion of the data set is the class label, but that's a term more commonly used for categorical data than continuous data (discussed in the next section).
All the other data in the table is data that will be used as input to the algorithm we're using that will tell the algorithm about the data we're predicting. Typically called features, you can think of these as the independent variables in the analysis of our data, or a "fingerprint" that contains information describes the

output variable.
Any single row in this table, then, can be thought of as a *feature vector*, where the one instance of our data set would have the one piece of information we'd like to predict (the band gap, in this case), as well as a set of features that describe that information (the color, crystallinity, and melting temperature).

4 Data Types

Before using machine learning on a data set, it's important to understand the kind of data you're working with. Typically, there are two data types to concern ourselves with for machine learning applications:

1. Categorical data: Data that take discrete values and have no ordering.
2. Continuous data: Data that can take infinite values and are ordered.

Some examples of categorical data as compared to a similar type of continuous data are shown below:

	Categorical:	Continuous:
Weather:	Sunny, cloudy, or stormy	Temperature
Color:	Red, orange, etc.	Light wavelength
Question:	Does an alloy contain metal?	How much metal is in this alloy?

When making predictions, consider what's being predicted:

Categorical class labels →classification tasks
Continuous class labels →regression tasks

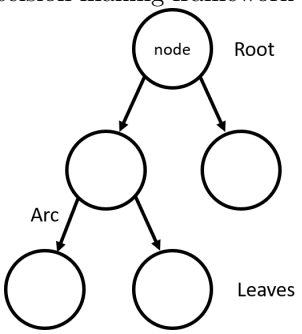
Important to note: Classification or regression tasks denote the type of data you're predicting. You can predict a continuous class label by feeding your model categorical training data, and vice versa.

5 Machine Learning Algorithms

This section will go over the details of several ML algorithms, particularly decision trees, random forests, and neural networks. This is not an exhaustive list of all ML algorithms currently being used in industry or research, but it does include ones that are widely used today. Each section will introduce the concept of the algorithm, explain the structure that can be visualized for a given algorithm, and discuss any variables (hyperparameters) the researcher can control directly related to the algorithm.

5.1 Decision Trees

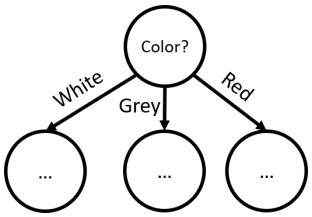
Decision trees are easy to think of as a decision making framework you've used before, similar to a flowchart.



In Figure 1, above, is a decision tree. The general structure of the tree as follows:

- White circles are called nodes. They represent places where data is split on, or where predictions are made with the data if the node doesn't have any more nodes below it (i.e. is a "leaf node").
- Arrows connecting nodes are known as arcs.
- The node at the top of the tree is the root. When feeding information through the tree, it will start at the root.
- At the bottom of the tree are the leaf nodes. These nodes contain a simple regression model for the subset of data points that fall into this leaf node.

The concept of a decision tree can be thought of as a succession of questions and answers starting at the root of the tree and ending at the leaves, where the nodes hold questions about the data set and the arcs represent possible answers to the question posed at the originating node. To analyze data as a decision tree, you would examine your data set and pick one feature to ask a question about. Taking again the semiconductor data as an example, we could first ask a question about what color the semiconductor is. The decision tree resulting from just this question would be as follows:



As seen above, the question we'd like to ask is written in the node, and possible answers to the question lie on the arcs from the node. This process can repeat until you've asked a question about every feature in the data set.

The order in which these questions about features are asked is also important for decision trees. Since decision trees are based heavily in information theory, where we seek the minimization of the data set rapidly, features that are split on earlier in the decision making process are more impactful on what's being predicted.

When implementing a decision tree, some of the hyperparameters (variables) you can control are the maximum depth of the tree, the maximum number of differently labelled samples at the leaves, as well as a few others. The most common hyperparameter is by far the maximum depth of the tree.

In practice, decision trees are fairly easy to implement, but they tend to overfit. The amount of overfitting can be moderated by changing the maximum depth of trees when considering rather large data sets. This concept relates to the application of Ockham's Razor (the simplest answer is the right answer); in this case, this would say to pick the shallower of two decision trees, as the smaller tree has likely made fewer assumptions and is more likely to be the correct solution to the problem at hand. It is important to note that this generalization is not a blanket statement that is a solution to every problem of overfitting decision trees, but it is a concept to consider when predictive capabilities are less than desirable.

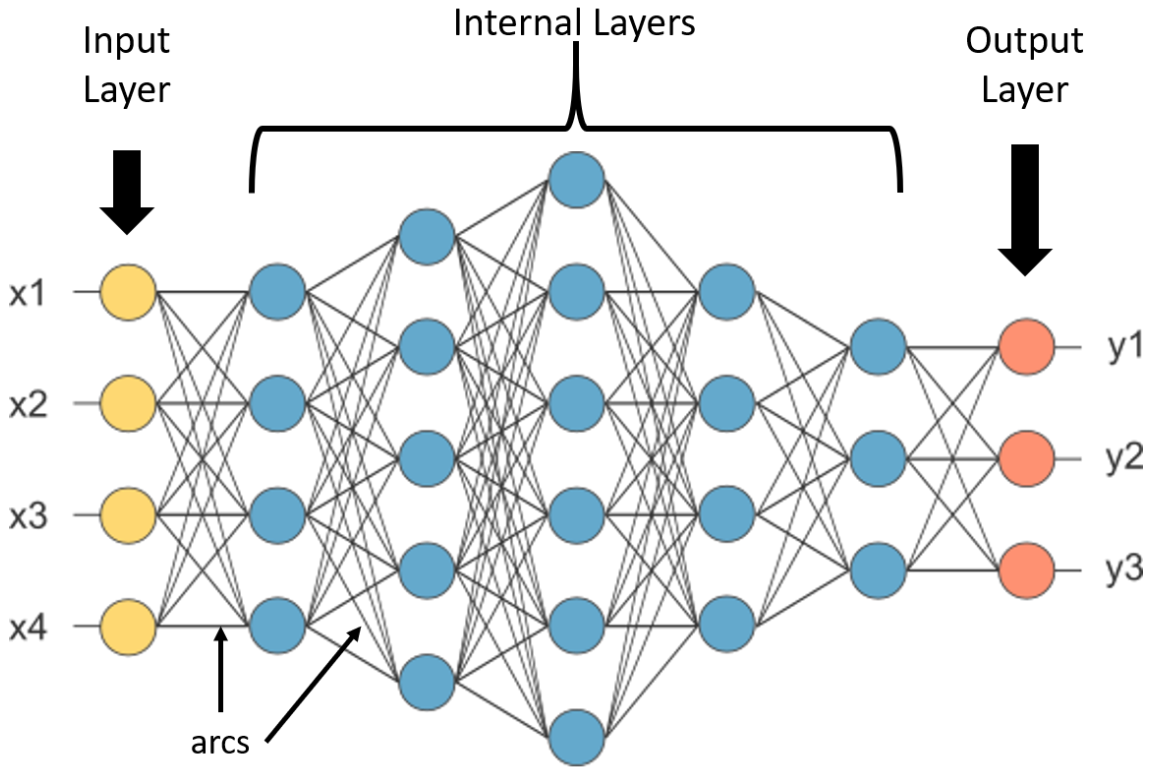
5.2 Random Forests

There are two important pieces of the name "random forest." The easier of these two to guess is why they're called forests. Since forests are places with many trees, it's logical to assume random forests would involve many (decision) trees. In fact, that's exactly what a random forest is: an algorithm that takes note of the output of many decision trees to give one output. are known as an ensemble method because we typically choose to focus on the performance of the forest as a whole rather than each individual tree in the forest. However, the reason for these forests to be called random is slightly less obvious. Where decision trees work to minimize the entropy of the data set at each split considering all possible features to split on, when building trees for random forests, only a random subset of features are available to choose to split upon. Now that we know a bit about the name, how's a random forest work?

The way a random forest works is fairly simple: many decision trees are built and fitted to the data set, and the output of the forest is the either the majority classification of all decision trees in the forest (for a classification task) or find the average value output by all the trees in the forest (for a regression task). Typically, random forests also employ a type of bootstrap aggregating (*bagging*, or randomly sampling from a set with replacement) to randomly resample the decision trees being used to vote on the output of the model. The output of the forest can then be analyzed just as you would for any other algorithm. One of the most common hyperparameters to consider when implementing a random forest is the number of trees to include in the forest, and the hyperparameters for the decision trees composing the forest are still applicable for each individual tree.
One of the reasons to implement a random forest over a single decision tree concerns the problem of overfitting. Random forests typically have fewer problems with overfitting, and they still are fairly easy to implement like decision trees. The reason overfitting is unlikely for a random forest is because a majority of the decision trees in the forest would have to incorrectly predict a sample's class or value. Using multiple decision trees reduces this risk.

5.3 Neural Networks

Another frequently used machine learning algorithm is the neural network (NN). Many famous neural networks are being used today for everything from playing board games to steering self-driving cars (e.g. AlphaGo, TensorFlow, Siri, and Uber, to name a few), and they come in many varieties that are named differently for variations in their structure. We'll go over possible variations of neural networks after we discuss the general structure of a neural network. Again, the specific structure is dependent on the type of NN, but the general components are shown in the image below:



- Some number of nodes (the circles), which hold either data, a function used to transform that data, or a prediction on a class label for the given data point.
- Some arcs, connecting the nodes (the black lines).
- An input layer (the yellow circles), where data is fed into the network.

- 1+ internal layers (the blue circles), that perform a series of transformations on the input data using activation functions and connect the input layer to the output layer.
- An output layer (the orange circles), which details the output of the network (the predictions on the class label). For every possible class in a classification task and for most values that could be predicted in a regression task, there will be a node in the output layer.

Some possible combinations of the above components are a simple, one node networks (called perceptrons), more complex network containing multiple layers, each with multiple nodes and unique functions embedded in these nodes (such as a convolution neural network), or even have thousands of nodes in a network with many layers (called a deep neural network). The main difference between the types of neural networks lies in the internal layers. The nodes in internal layers are connected to previous layers and receive the outputs of previous layers. Each input typically has a weight associated with it, which are used as inputs to activation function in the node. The activation function is a function that outputs a value dependent on the data received. Some examples of activation functions are:

- Linear threshold unit
- Hyperbolic tangent
- Sigmoid
- Rectified linear unit (ReLU)

This activation function typically has a threshold value that determines whether or not it can . The node then outputs the if the result of the activation function is less than or greater than this threshold value. This output goes to the next layer of nodes in the network. This process repeats for however many layers are in the network.

Training the neural network involves sending data through the network, observing the error, and updating the weights of the nodes that are causing the error. One method for updating the weights is known as Back Propagation, and helps distribute the error from any given node across the entire network to minimize the erroneous node's impact. The nodes can be updated many times or few times, with the end goal being finding the global minimum of the error. Can error as a function of the weights and visualize this as a surface, where the lowest point on the surface is the minimum error. Finding this global minimum of the error is ideal, but often takes very long. Output layers are typically simple and output the results of the predictions, whether they be classification labels or regression tasks. Each possible value for a class label, whether it be a class for classification or a value for regression, is represented by a node in the output layer. Each node in the output layer will always output a value when making predictions, but only one node should indication the sample being predicted falls under its category or regression value.

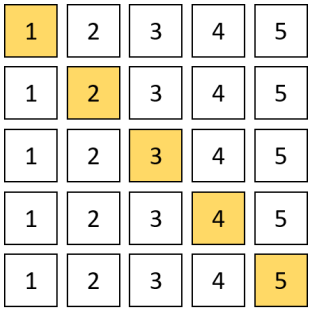
6 Cross Validation

Since predicting information we already know isn't very useful, we'd like to have a process to test how well our models can predict new data. However, a frequent problem when using materials data for machine learning is having enough data to adequately train a model in the first place! If possible, we want to minimize the amount of data reserved for validating the performance of our model and maximize the amount used for training. A common way to have data reserved for testing while not drastically reducing the size of a training data set is to use cross validation (CV) while training the model. Cross validation has numerous variations including leave-one out (LOOCV) and k-fold cross validation. While the details of these are slightly different, both work on the same premise of splitting a data set into *training* and *testing* (or *validation*) sets, where the model is trained only on the training data and predictions are made on the testing data that was left out during training. The main difference between LOOCV and k-fold CV is deciding how to partition the data into the training and testing subsets. LOOCV, as the name implies, functions by leaving out 1 random data point for testing and training using the rest of the data set. k-fold CV, on the other hand, splits the data into k equally sized, randomly assigned subgroups (often called *folds*) and using 1 fold for testing and the remaining folds for training. In addition to randomly grouping the data into k folds and using one fold for

testing, k-fold CV also ensures that every fold is left out at some point to be used for testing. The typical process for this repeated leaving out one fold in k-fold CV is as follows:

- Split data into k subgroups (k typically = 5 or 10)
- Train the model w/ k-1 of the folds and test using the fold
- Repeat until every subset has been the testing subset, then average the errors on the training sets.

The graphic below can help visualize this process for k-fold CV. Each row of squares is known as a *round* of cross validation, and the number of rounds of CV that will be performed is equal to k. In each round, the shaded square would be the fold of data left out for training, and each fold is left out exactly one time throughout the entire process.



The practice of using cross validation is useful because it can help assess if a model is overfitting, which would be indicated by a large increase in the error when predicting unseen data as compared to the error attained when training.

7 Assessing Model Performance

Now that we've discussed several machine learning algorithms, we'll switch our discussion to assessing how well models are predicting. Depending on whether a classification or regression task is being performed, methods to assess model performance vary, so we'll be addressing each separately.

7.1 Assessing Classifier Performance

When assessing how well a classifier is predicting, there are several ways to quantify the quality of the model's predictions. For example, one might want to know how many A *confusion matrix* nicely recaps this. A confusion matrix looks like the following table, which will use the example of trying to predict if an alloy is metallic or non-metallic:

	Metal (Actual Class)	Non-Metal (Actual Class)
Metal (Predicted Class)	True Positives	False Positives
Non-Metal (Predicted Class)	False Negatives	True Negatives

Where the upper left entry in the table is the number of data points that were predicted metals and are actually metals (true positives), the upper right is the number of data points predicted as non-metals that are metals (false positives), the lower left is the data points predicted as non-metals that are actually metals (false negatives), and the lower right are the data points that were predicted as non-metals that are actually non-metals. There are several standardized percentages that can be obtained from this matrix and used to describe classifier performance:

- True Positive Rate: Fraction of items predicted true when they should be predicted true

$$\frac{TruePositive}{TruePositive + FalseNegatives} \tag{1}$$

- False Positive Rate: Fraction of items predicted true when they should be predicted false

$$\frac{FalsePositive}{TruePositive + FalseNegatives} \tag{2}$$

- True Negative Rate: Fraction of items predicted false when they should be predicted false

$$\frac{TrueNegative}{TrueNegative + FalsePositive} \tag{3}$$

- False Negative Rate: Fraction of items predicted false when they should be predicted true

$$\frac{FalseNegative}{TrueNegative + FalsePositive} \tag{4}$$

- Precision:

$$\frac{TruePositive}{TruePositive + FalsePositive} \tag{5}$$

- Recall:

$$\frac{TruePositive}{TruePositive + FalseNegative} \tag{6}$$

Depending on the project, precision or recall may be more important to have high numbers for. However, if neither recall nor precision are more important to your project, they can be used together in the F1 score, which is the harmonic mean of the recall and the precision:

$$F1 = 2 * \frac{1}{\frac{1}{Precision} + \frac{1}{Recall}} \tag{7}$$

Possible values for the F1 score range from 0 (worst performance, no correct predictions) to 1 (best performance, all correct predictions).

7.2 Assessing Regressor Performance

There are several ways to quantify regressor performance, but the main one we'll focus on is the Root Mean Squared Error ($RMSE$). The RMSE tells how far a predicted value lies from the actual value using the following equation:

$$RMSE = \sqrt{\frac{\hat{y}_i - y_i}{n}} \tag{8}$$

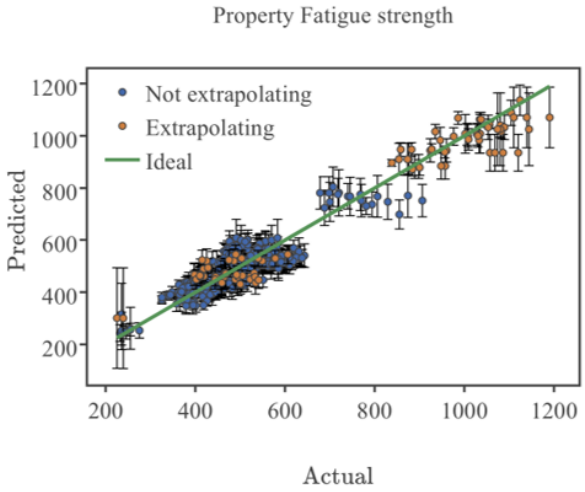
To assess the RMSE for an entire data set, the difference between the predicted and actual values would be summed over all data points. One important note is that there's no "good" value of RMSE to look for across every different data set and problem. To get around this problem, it can be beneficial to consider RMSE normalized by the standard deviation (σ) of the data set at hand to consider what the error on your predictions is as a fraction of the spread of the data. Some ballpark estimates for the quality of RMSE/σ values are summarized in the table below:

RMSE/σ	Prediction Quality
$>.9$	Poor
.5 - .9	Ok
.3 - .5	Good
$<.3$	Excellent

A good example of why RMSE/σ If you add more data to your data set and the mean of the data increases drastically, the RMSE will likely increase as well. However, the RMSE/σ may have decreased, indicating better performance despite the higher RMSE.

Parity plots are a good way to summarize the results of a classifier and quickly inspect the predictions for any outlying predictions. Each point on the plot represents a sample in the data set, where:
X coord of point: Actual value of sample
Y coord of point: Value of sample predicted by the model.

A perfect prediction would have all points fall perfectly on the line y = x. An example of a parity plot for predicting the fatigue strength for some metallic alloys is shown as an example of a parity plot.



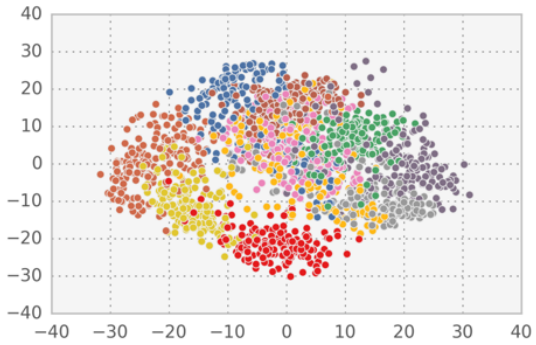
When looking at this plot, a quick way to tell if a particular point isn't predicting well is checking for any points that fall far from the ideal line, shown in green on the figure.

8 Visualizing High Dimensional Data

Finally, we will discuss one persistent problem that can impact large data sets being used for classification or regression, regardless of algorithm being implemented. This problem is with high dimensional data, and a common problem when using high dimensional data is visualization. Since visualizing data is one of the fastest ways for humans to briefly analyze data to find trends, problems, or other meaning, the task of visualizing a high dimensional data set is quite pressing.
To do this, need to reduce the dimensions of the data set using methods that often result in clusters of similar data points.

8.1 Principal Component Analysis (PCA)

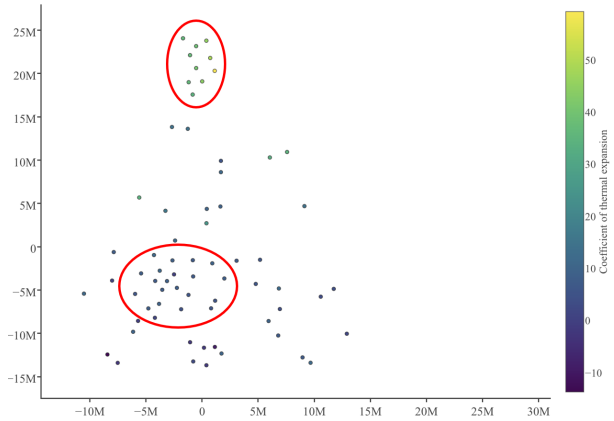
Principal component analysis (PCA) is a linear method to project data from high dimensions to 2 or 3 dimensions. Visualizing data using this method can tell that near points are similar to one another and far points are dissimilar. However, it's better at telling what's dissimilar than what's similar, and if data isn't linearly separable, PCA may not be able to give good separation of different clusters, resulting in little information to be gained from this method. However, PCA does retain some of the global structure of the data. An example of a PCA plot is shown below:



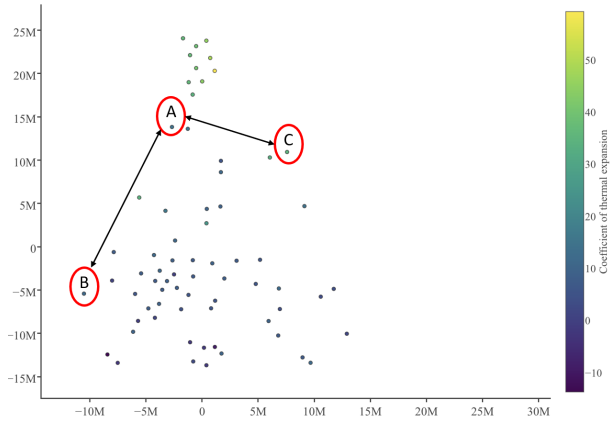
With this plot, it's easy to see the separate clusters of differently colored points, particularly in the clusters are far apart from one another (for example, the orange and purple clusters are quite far apart and it's easy to see the data points in these 2 clusters are separate from one another). However, there are sections of this plot, such as the center, where clustering is harder to see and clusters of colors run together. This goes to reinforce the point that PCA is better at telling what's dissimilar than what's similar.

8.2 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE, pronounced *t-SNEE*), is a non-linear method to reduce dimensions. This method works as a probability function, where you're assessing the probability 2 points are likely to be paired near to one another because they're similar. Shown in the figure below is an example of a t-SNE plot that has some clusters of data points circled. What we can tell about the points from this plot is that the points within the circles are similar to one another. For example, this plot is visualizing data on the coefficient of thermal expansion for some concrete, where the data set also contains information about the type of larger rocks that make up the concrete, which has a large impact on the CTE. Each of the circled clusters represents alloys that have either crushed stone or a glacial granite as this rock type in the data set.



In the above plot, you may notice the axes have no labels. This is because of the non-linear way data is separated when assessing the probability function used to plot the data. While the axes aren't entirely unimportant, they aren't incredibly useful when considering relative distance between 2 points, so the axes on this plot typically don't need to be worried about. As stated before, t-SNE works through finding a probability that 2 points are similar and plotting them near one another. While we can also say 2 points that aren't clustered together aren't similar, that's about all the analysis we can make about dissimilar data points. This can be illustrated using the same t-SNE plot as before, now with points A, B, and C marked, shown below.



Let's say we're trying to see if there are some points that are dissimilar to point A, and we find points B and C in this plot. Point C is closer to point A than point B, but that doesn't mean point C is more similar to point A than point B. This is because the function that plots the data points on the t-SNE plot only preserves *local* structure, not global structure of the data set, so once again, we can say points that are near one another are similar and far points are dissimilar, but since global structure of our high dimensional data hasn't been preserved, we can't say anything about comparing far apart points.

9 Lingo and Jargon

This section serves as a quick overview of some machine learning-related jargon or buzzwords you may encounter frequently.

- Artificial Intelligence: The development of computer systems that can be used to perform tasks that would typically require a human's intelligence to complete the task. Usually used synonymously with machine learning.
- Class Label: In a classification problem, the class label is the value being predicted.
- Descriptor: Data that describes the output value of the model. During training, the algorithm will use this data to "learn" about the output variable being predicted. Also known as a feature.
- Features: Another term for a descriptor (see above).
- Hyperparameters: Certain variables that can be changed in a specific machine learning algorithm that may impact how well a trained model can predict on a data set. Some examples include the number of trees used in a random forest or the number of epochs used during the training of a neural network.
- Materials Informatics: Applying principles of informatics, such as techniques for data processing, storage, and analysis, to the materials science field.
- Neural net, NN, ANN...: Various abbreviations for "neural network".
- Overfitting: A phenomena that may occur when an ML algorithm hasn't learned any patterns in the data, but has simply "memorized" the data that was used to train it. The result is poor predictions on data the model hasn't seen, but deceptively good performance during training.
- Statistical Learning: A framework for machine learning that draws on the concepts of statistics to find a way to describe a value as a function of its features. Some examples of statistical learning are various types of regressions (linear, polynomial, etc.), decision trees, and support vector machines.