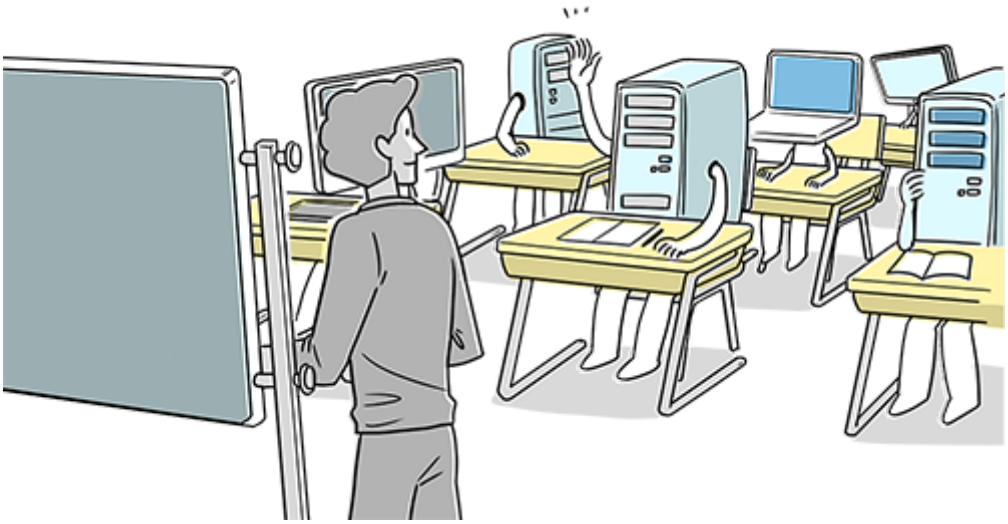
Introduction to Machine Learning

What is Machine Learning?

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.



How Machine Learning works?

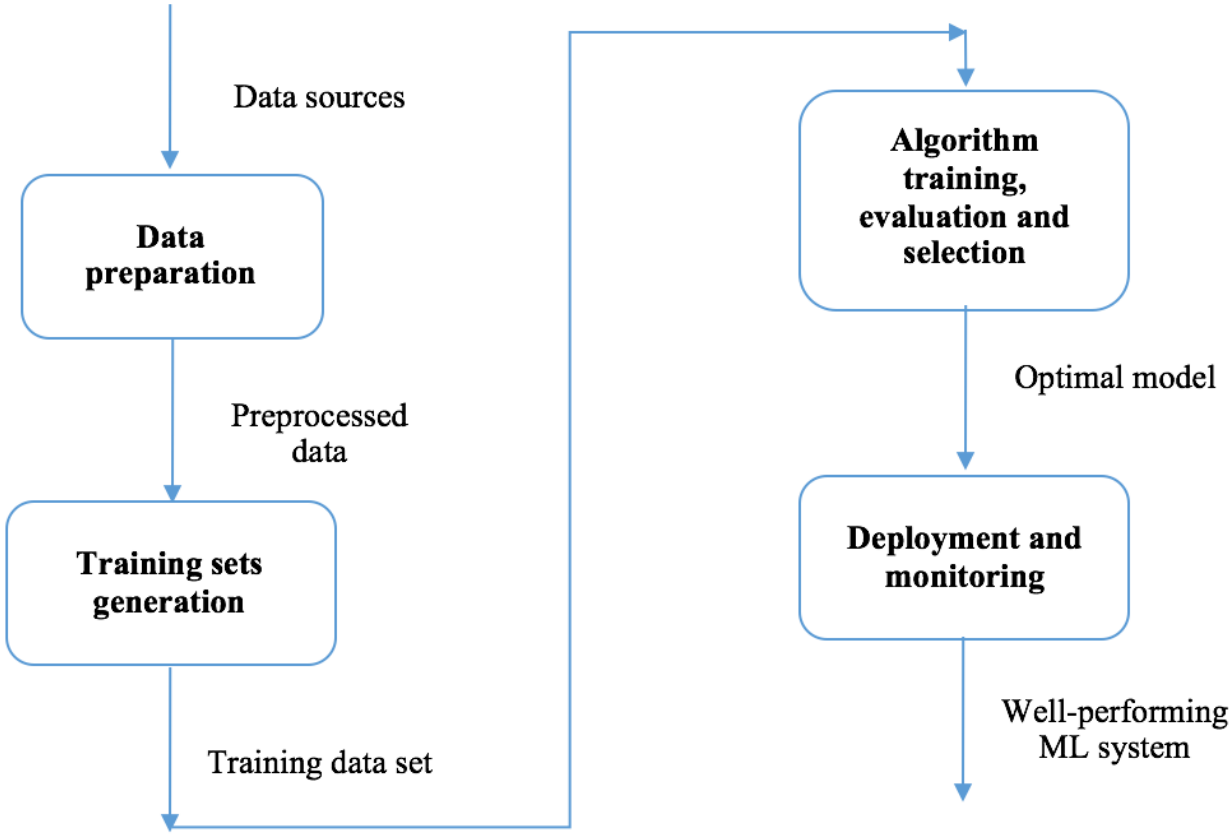
Suppose we want to buy the best web camera available in the market. In real life, the process we'd follow would be to look at several product reviews describing qualities about the model we are considering purchasing. For example, if we see that the reviews mostly consists of words like "good," "great," "excellent" etc. then we'd conclude that the webcam is a good product and we can proceed to purchase it. Whereas if the words like "bad," "not good quality," "poor resolution," then we conclude that it is probably better to look for another webcam. So you see, the reviews help us perform a "decisive action" based on the "pattern" of words that exist in the product reviews.

Hence, the relationship among the buyers who purchased the webcam and wrote product reviews will influence other buyers, and their product reviews, in turn, will influence future purchases. Thus, a pattern exists across the people who already purchased the product and the future buyers of the product.

Workflow

In general, tasks in solving a machine learning problem can be summarized into four areas:

- Data Preparation
- Training set generation
- Algorithm Training
- Development and monitering



Data Preparation Process

The more disciplined you are in your handling of data, the more consistent and better results you are like likely to achieve. The process for getting data ready for a machine learning algorithm can be summarized in three steps:

Step 1: Select Data

Step 2: Preprocess Data

Step 3: Transform Data

You can follow this process in a linear manner, but it is very likely to be iterative with many loops.

Step 1: Select Data

This step is concerned with selecting the subset of all available data that you will be working with. There is always a strong desire for including all data that is available, that the maxim “more is better” will hold. This may or may not be true.

You need to consider what data you actually need to address the question or problem you are working on. Make some assumptions about the data you require and be careful to record those assumptions so that you can test them later if needed.

Below are some questions to help you think through this process:

What is the extent of the data you have available? For example through time, database tables, connected systems. Ensure you have a clear picture of everything that you can use. What data is not available that you wish you had available? For example data that is not recorded or cannot be recorded. You may be able to derive or simulate this data. What data don’t you need to address the problem? Excluding data is almost always easier than including data. Note down which data you excluded and why. It is only in small problems, like competition or toy datasets where the data has already been selected for you.

Step 2: Preprocess Data

After you have selected the data, you need to consider how you are going to use the data. This preprocessing step is about getting the selected data into a form that you can work.

Three common data preprocessing steps are formatting, cleaning and sampling:

Formatting: The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file. Cleaning: Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be sensitive information in some of the attributes and these attributes may need to be anonymized or removed from the data entirely. Sampling: There may be far more selected data available than you need to work with. More data can result in much longer running times for algorithms and larger computational and memory requirements. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset. It is very likely that the machine learning tools you use on the data will influence the preprocessing you will be required to perform. You will likely revisit this step.

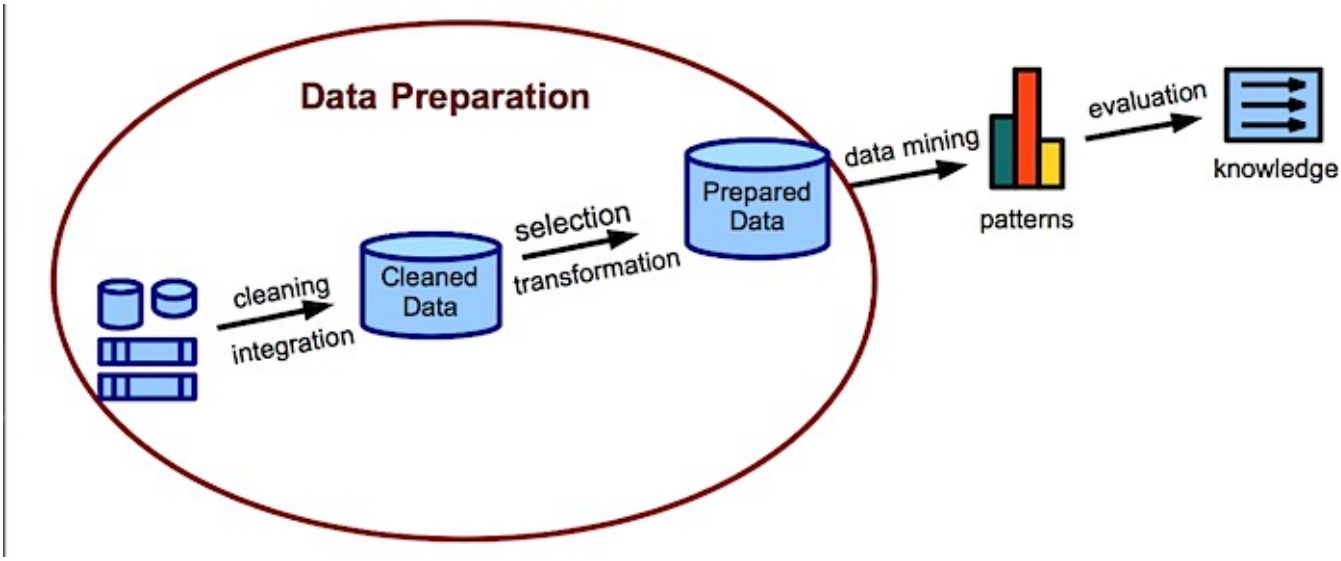
So much data So much data Photo attributed to Marc_Smith, some rights reserved

Step 3: Transform Data

The final step is to transform the process data. The specific algorithm you are working with and the knowledge of the problem domain will influence this step and you will very likely have to revisit different transformations of your preprocessed data as you work on your problem.

Three common data transformations are scaling, attribute decompositions and attribute aggregations. This step is also referred to as feature engineering.

- Scaling: The preprocessed data may contain attributes with a mixtures of scales for various quantities such as dollars, kilograms and sales volume. Many machine learning methods like data attributes to have the same scale such as between 0 and 1 for the smallest and largest value for a given feature. Consider any feature scaling you may need to perform.
- Decomposition: There may be features that represent a complex concept that may be more useful to a machine learning method when split into the constituent parts. An example is a date that may have day and time components that in turn could be split out further. Perhaps only the hour of day is relevant to the problem being solved. consider what feature decompositions you can perform.
- Aggregation: There may be features that can be aggregated into a single feature that would be more meaningful to the problem you are trying to solve. For example, there may be a data instances for each time a customer logged into a system that could be aggregated into a count for the number of logins allowing the additional instances to be discarded. Consider what type of feature aggregations could perform.



Training Set vs. Test Set

What is a Training Set?

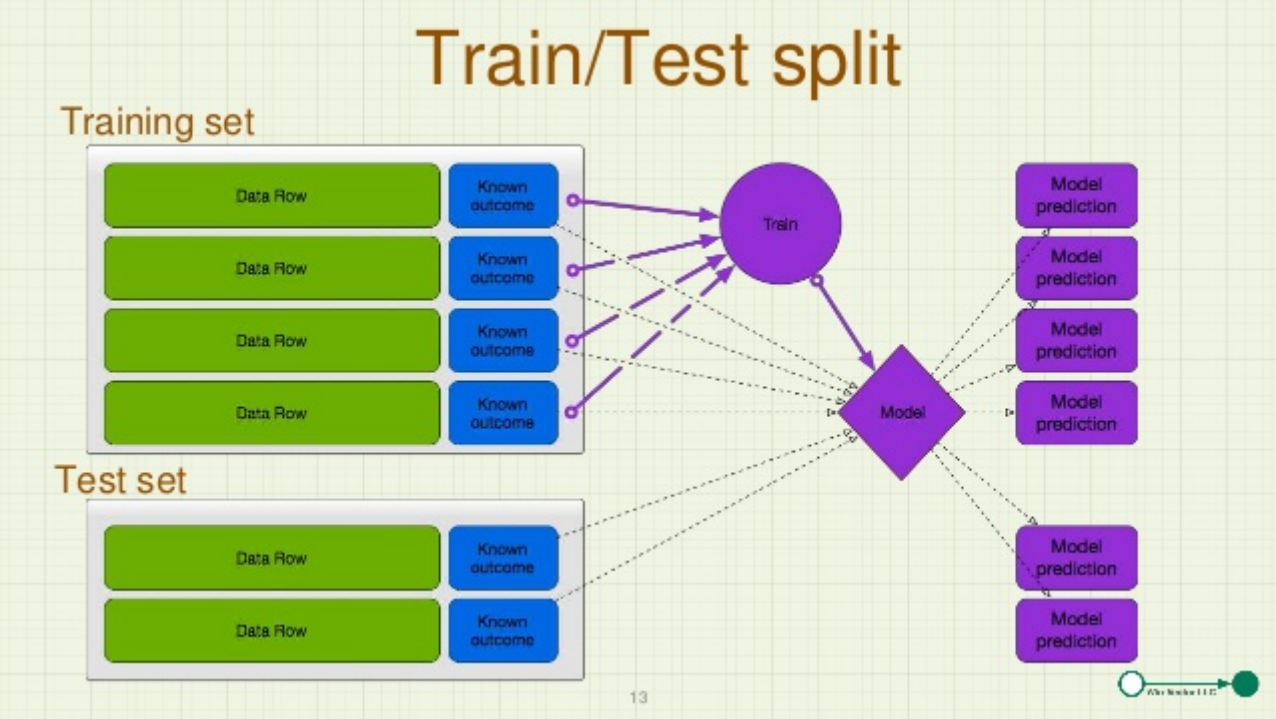
In machine learning, a training set is a dataset used to train a model. In training the model, specific features are picked out from the training set. These features are then incorporated into the model. Thereby, if the training set is labeled correctly, the model should be able to learn something from these features.

What is a Test Set?

The test set is a dataset used to measure how well the model performs at making predictions on that test set. If the prediction scores for the test set are unreasonable, we'll need to make some adjustments to our model and try again.

But why not just use the data from the training set to test the performance of our model?

The issue here is that our test would yield misleading results if we test our model with the training data. The model itself was created by learning from the training set, so it will likely do quite well at making predictions on the training set itself- it knows this data too well. We need to test the model with a test set, i.e. a dataset the model hasn't seen before.



Algorithm training

Selecting an algorithm is based on the problem we are trying to solve. There are following type machine learning algorithms:

- Supervised
- Unsupervised
- Reinforcement
- Semisupervised

Supervised

Supervised learning algorithms are trained using labeled examples, such as an input where the desired output is known. For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data. Supervised learning is commonly used in applications where historical data predicts likely future events. For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim. Supervised algorithms can further devided into following:

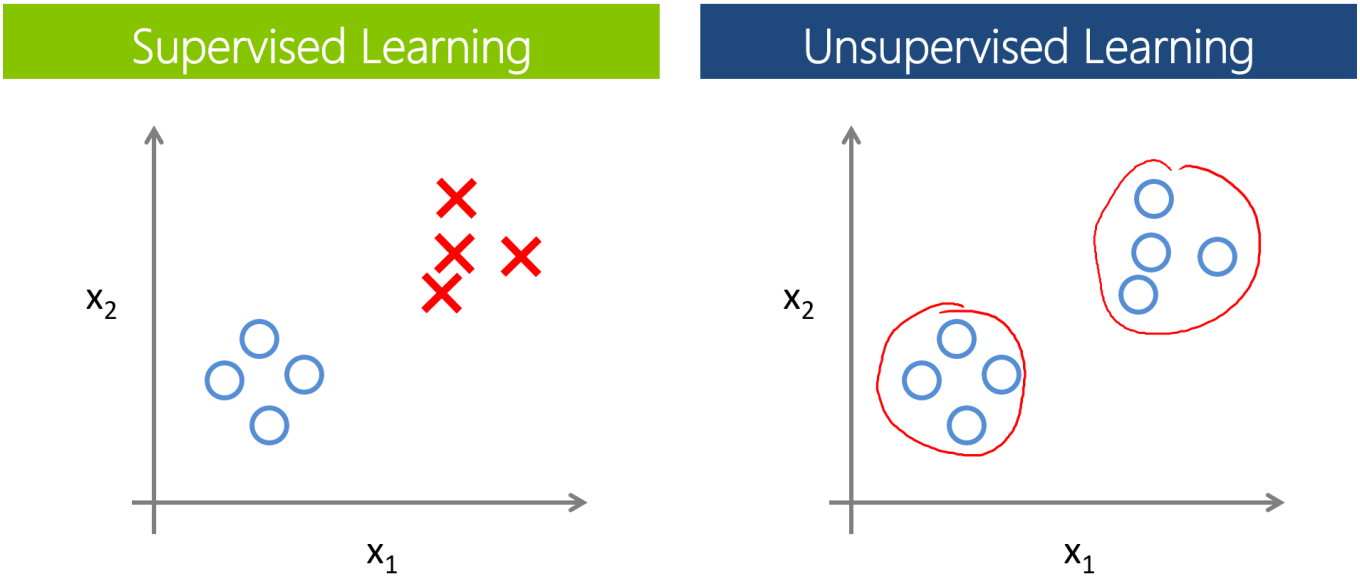
- Classification. When the data are being used to predict a category, supervised learning is also called classification. This is the case when assigning an image as a picture of either a 'cat' or a 'dog'. When there are only two choices, it's called two-class or binomial classification. When there are more categories, as when predicting the winner of the NCAA March Madness tournament, this problem is known as multi-class classification.

- Regression. When a value is being predicted, as with stock prices, supervised learning is called regression.

- Anomaly detection. Sometimes the goal is to identify data points that are simply unusual. In fraud detection, for example, any highly unusual credit card spending patterns are suspect. The possible variations are so numerous and the training examples so few, that it's not feasible to learn what fraudulent activity looks like. The approach that anomaly detection takes is to simply learn what normal activity looks like (using a history non-fraudulent transactions) and identify anything that is significantly different.

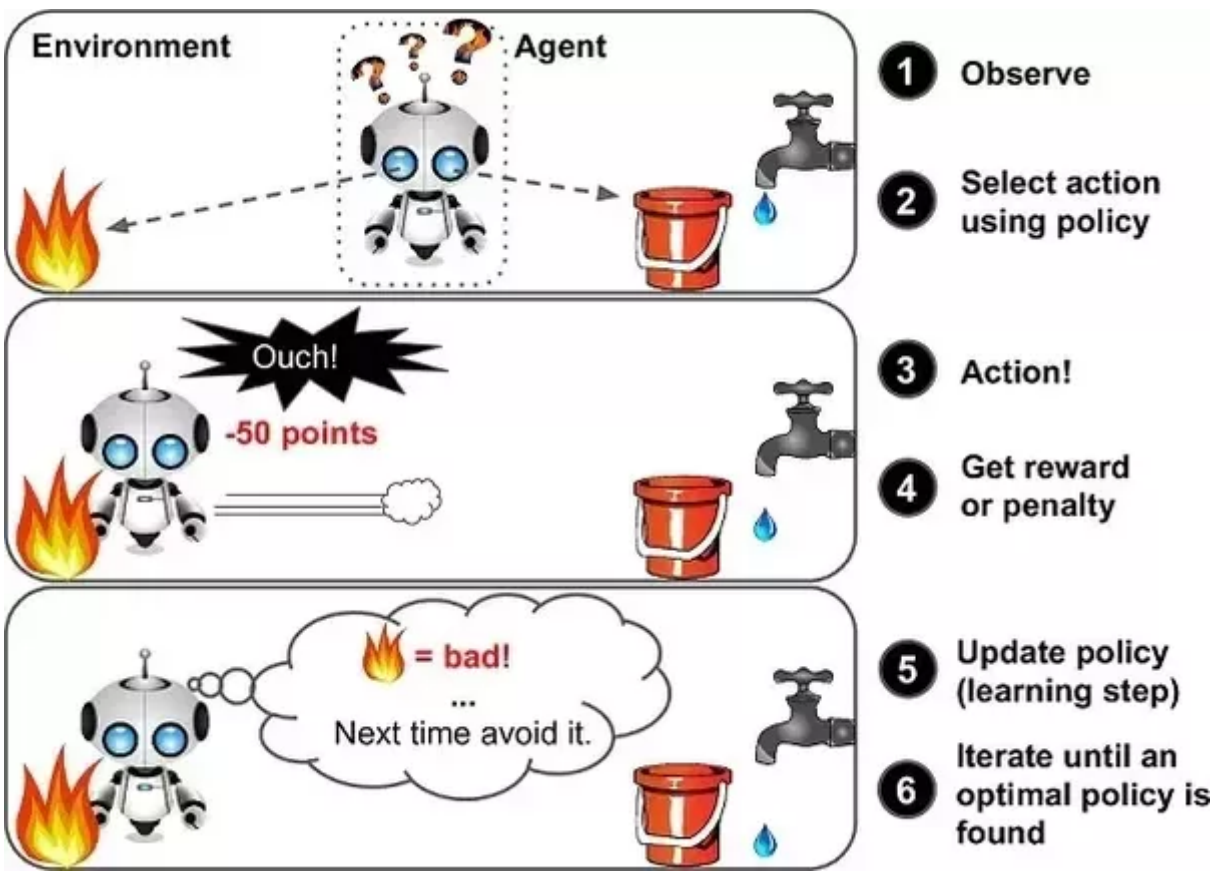
Unsupervised

Unsupervised learning is used against data that has no historical labels. The system is not told the "right answer." The algorithm must figure out what is being shown. The goal is to explore the data and find some structure within. Unsupervised learning works well on transactional data. For example, it can identify segments of customers with similar attributes who can then be treated similarly in marketing campaigns. Or it can find the main attributes that separate customer segments from each other. Popular techniques include self-organizing maps, nearest-neighbor mapping, k-means clustering and singular value decomposition. These algorithms are also used to segment text topics, recommend items and identify data outliers.



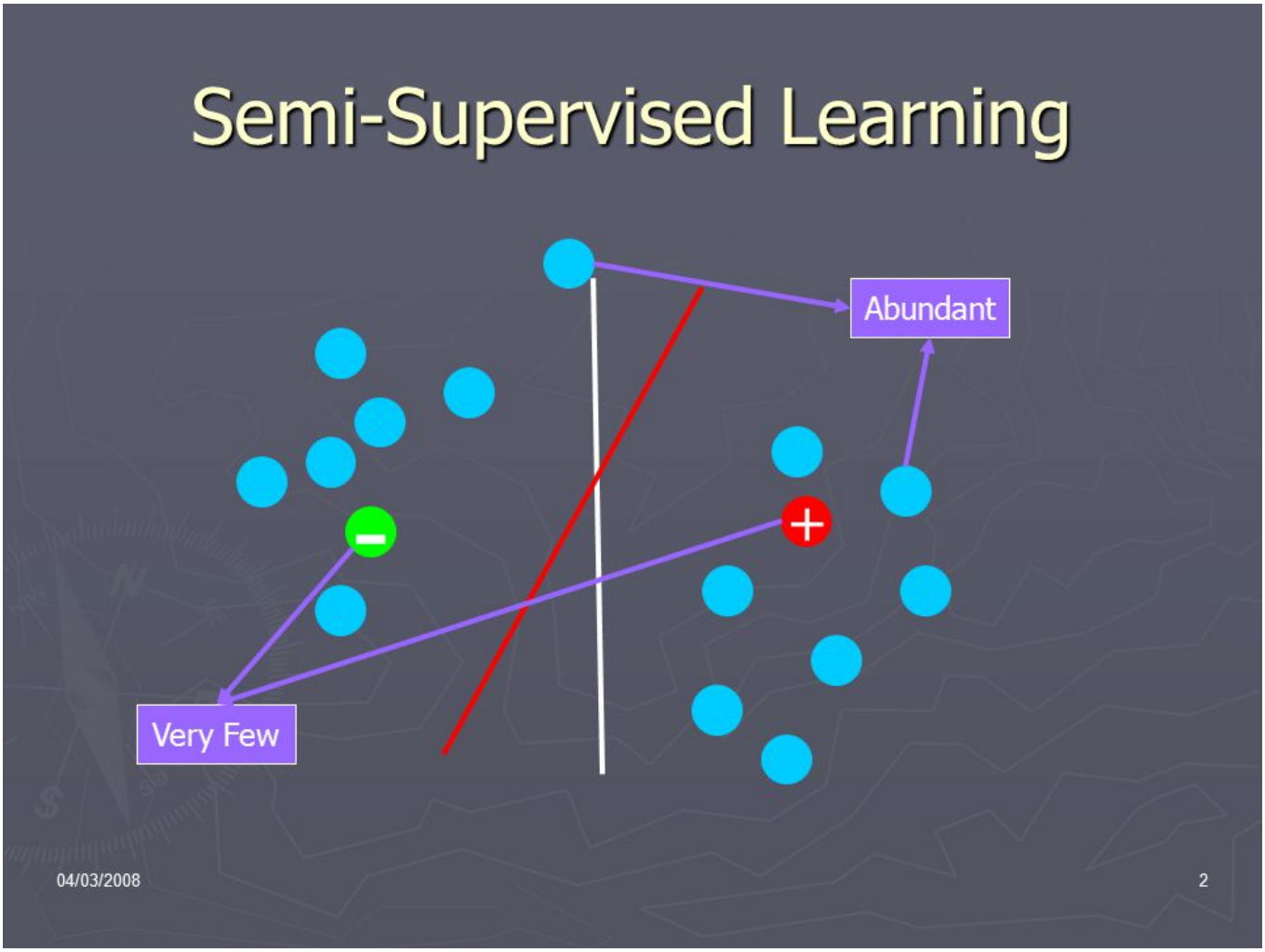
Reinforcement

Reinforcement learning is often used for robotics, gaming and navigation. With reinforcement learning, the algorithm discovers through trial and error which actions yield the greatest rewards. This type of learning has three primary components: the agent (the learner or decision maker), the environment (everything the agent interacts with) and actions (what the agent can do). The objective is for the agent to choose actions that maximize the expected reward over a given amount of time. The agent will reach the goal much faster by following a good policy. So the goal in reinforcement learning is to learn the best policy.



Semisupervised

Semisupervised learning is used for the same applications as supervised learning. But it uses both labeled and unlabeled data for training – typically a small amount of labeled data with a large amount of unlabeled data (because unlabeled data is less expensive and takes less effort to acquire). This type of learning can be used with methods such as classification, regression and prediction. Semisupervised learning is useful when the cost associated with labeling is too high to allow for a fully labeled training process. Early examples of this include identifying a person's face on a web cam.



Choosing an algorithm

Accuracy

Getting the most accurate answer possible isn't always necessary. Sometimes an approximation is adequate, depending on what you want to use it for. If that's the case, you may be able to cut your processing time dramatically by sticking with more approximate methods. Another advantage of more approximate methods is that they naturally tend to avoid overfitting.

Training time

The number of minutes or hours necessary to train a model varies a great deal between algorithms. Training time is often closely tied to accuracy—one typically accompanies the other. In addition, some algorithms are more sensitive to the number of data points than others. When time is limited it can drive the choice of algorithm, especially when the data set is large.

Linearity

Lots of machine learning algorithms make use of linearity. Linear classification algorithms assume that classes can be separated by a straight line (or its higher-dimensional analog). These include logistic regression and support vector machines. Linear regression algorithms assume that data trends follow a straight line. These assumptions aren't bad for some problems, but on others they bring accuracy down.

Non-linear class boundary

Non-linear class boundary relying on a linear classification algorithm would result in low accuracy

Data with a nonlinear trend

Data with a nonlinear trend using a linear regression method would generate much larger errors than necessary Despite their dangers, linear algorithms are very popular as a first line of attack. They tend to be algorithmically simple and fast to train.

Number of parameters

Parameters are the knobs a data scientist gets to turn when setting up an algorithm. They are numbers that affect the algorithm's behavior, such as error tolerance or number of iterations, or options between variants of how the algorithm behaves. The training time and accuracy of the algorithm can sometimes be quite sensitive to getting just the right settings. Typically, algorithms with large numbers parameters require the most trial and error to find a good combination.

The upside is that having many parameters typically indicates that an algorithm has greater flexibility. It can often achieve very good accuracy. Provided you can find the right combination of parameter settings.

Number of features

For certain types of data, the number of features can be very large compared to the number of data points. This is often the case with genetics or textual data. The large number of features can bog down some learning algorithms, making training time unfeasibly long.

Overfitting vs. Underfitting

Overfitting

Overfitting refers to a model that models the training data too well.

Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.

Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function. As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.

For example, decision trees are a nonparametric machine learning algorithm that is very flexible and is subject to overfitting training data. This problem can be addressed by pruning a tree after it has learned in order to remove some of the detail it has picked up.

Undefitting

Underfitting refers to a model that can neither model the training data nor generalize to new data.

An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.

Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.

Good fit

Ideally, you want to select a model at the sweet spot between underfitting and overfitting.

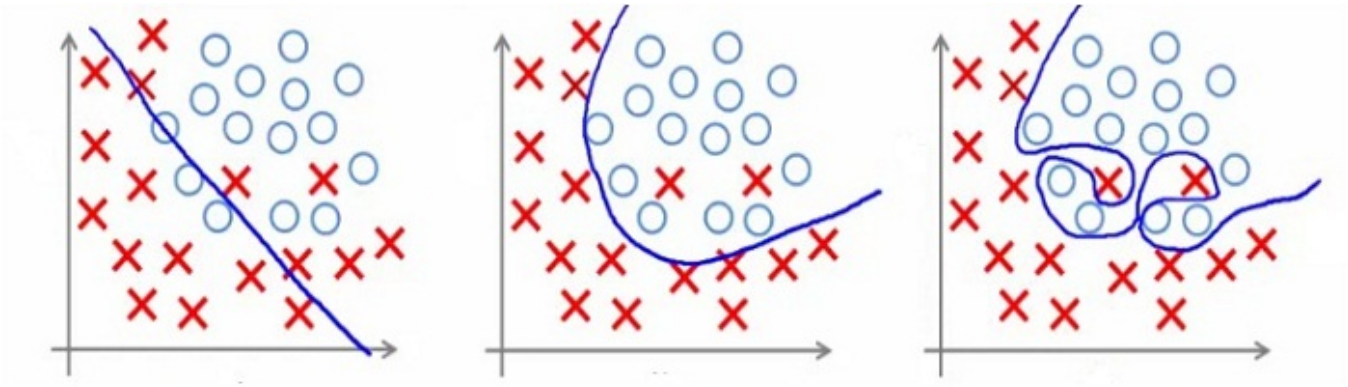
This is the goal, but is very difficult to do in practice.

To understand this goal, we can look at the performance of a machine learning algorithm over time as it is learning a training data. We can plot both the skill on the training data and the skill on a test dataset we have held back from the training process.

Over time, as the algorithm learns, the error for the model on the training data goes down and so does the error on the test dataset. If we train for too long, the performance on the training dataset may continue to decrease because the model is overfitting and learning the irrelevant detail and noise in the training dataset. At the same time the error for the test set starts to rise again as the model's ability to generalize decreases.

The sweet spot is the point just before the error on the test dataset starts to increase where the model has good skill on both the training dataset and the unseen test dataset.

You can perform this experiment with your favorite machine learning algorithms. This is often not useful technique in practice, because by choosing the stopping point for training using the skill on the test dataset it means that the testset is no longer "unseen" or a standalone objective measure. Some knowledge (a lot of useful knowledge) about that data has leaked into the training procedure.



Under-fitting

(too simple to explain the variance)

Appropriate-fitting

Over-fitting

(forcefitting -- too good to be true)

In [ ]: