

MODULE I:

Definitions of Machine Learning

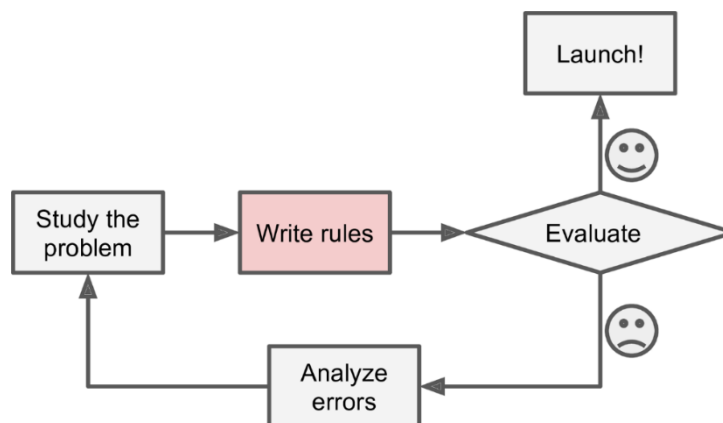
- Machine Learning is the science (and art) of programming computers so they can learn from data.
- Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.
- A computer program is said to learn from experience E with respect to some task T and some performance measure P , if its performance on T , as measured by P , improves with experience E .

Need to use Machine Learning

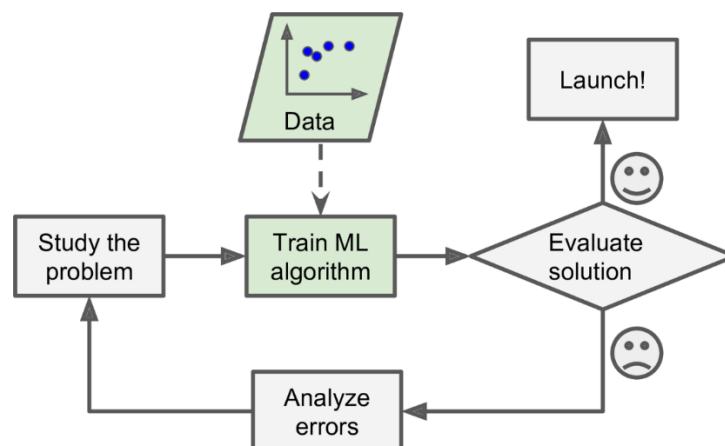
Machine Learning is great for:

- Problems for which existing solutions require a lot of hand-tuning or long lists of rules: one Machine Learning algorithm can often simplify code and perform better.

Eg: writing traditional programming vs machine learning approach for a spam filter:



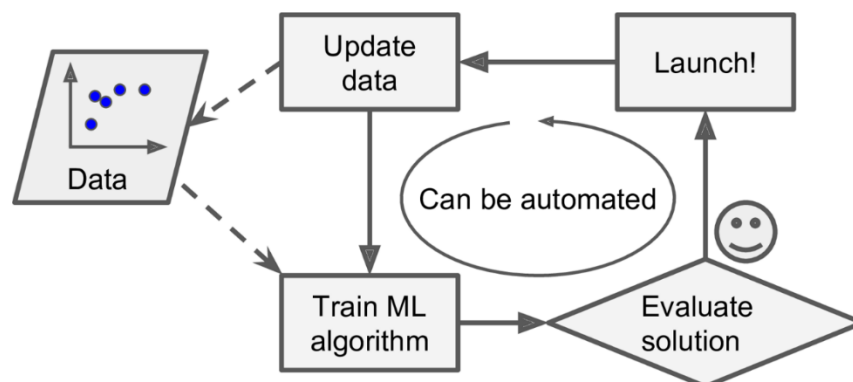
Traditional programming for spam filter detection



Machine Learning approach for spam filter detection

In traditional programming, one writes a detection algorithm for each of the patterns that you noticed, and the program would flag emails as spam if a number of these patterns are detected. Since the problem is not trivial, your program will likely become a long list of complex rules—pretty hard to maintain. In contrast, a spam filter based on Machine Learning techniques automatically learns which words and phrases are good predictors of spam by detecting unusually frequent patterns of words in the spam examples compared to the ham examples. The program is much shorter, easier to maintain, and most likely more accurate.

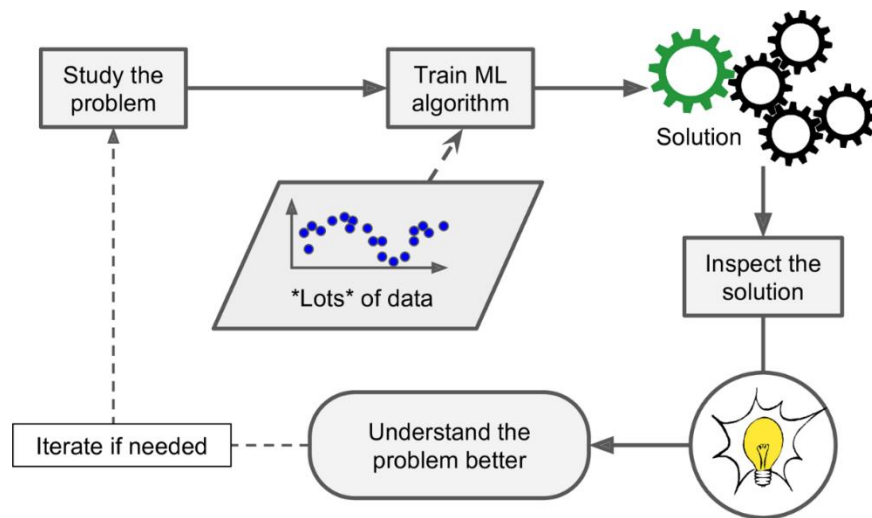
- ✚ Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- ✚ Machine Learning shines is for problems that either are too complex for traditional approaches or have no known algorithm. For example, consider speech recognition: say you want to start simple and write a program capable of distinguishing the words “one” and “two.” You might notice that the word “two” starts with a high-pitch sound (“T”), so you could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos. Obviously this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages. The best solution (at least today) is to write an algorithm that learns by itself, given many example recordings for each word.
- ✚ Fluctuating environments: a Machine Learning system can adapt to new data.
Eg: In the spam filter detection example, if spammers notice that all their emails containing “4U” are blocked, they might start writing “For U” instead. A spam filter using traditional programming techniques would need to be updated to flag “For U” emails. If spammers keep working around your spam filter, you will need to keep writing new rules forever. In contrast, a spam filter based on Machine Learning techniques automatically notices that “For U” has become unusually frequent in spam flagged by users, and it starts flagging them without your intervention.



Machine Learning solutions adapt to change

- ✚ Getting insights about complex problems and large amounts of data.
ML algorithms can be inspected to see what they have learned (although for some algorithms this can be tricky). For instance, once the spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam. Sometimes this will reveal

unsuspected correlations or new trends, and thereby lead to a better understanding of the problem. Applying ML techniques to dig into large amounts of data can help discover patterns that were not immediately apparent. This is called data mining.



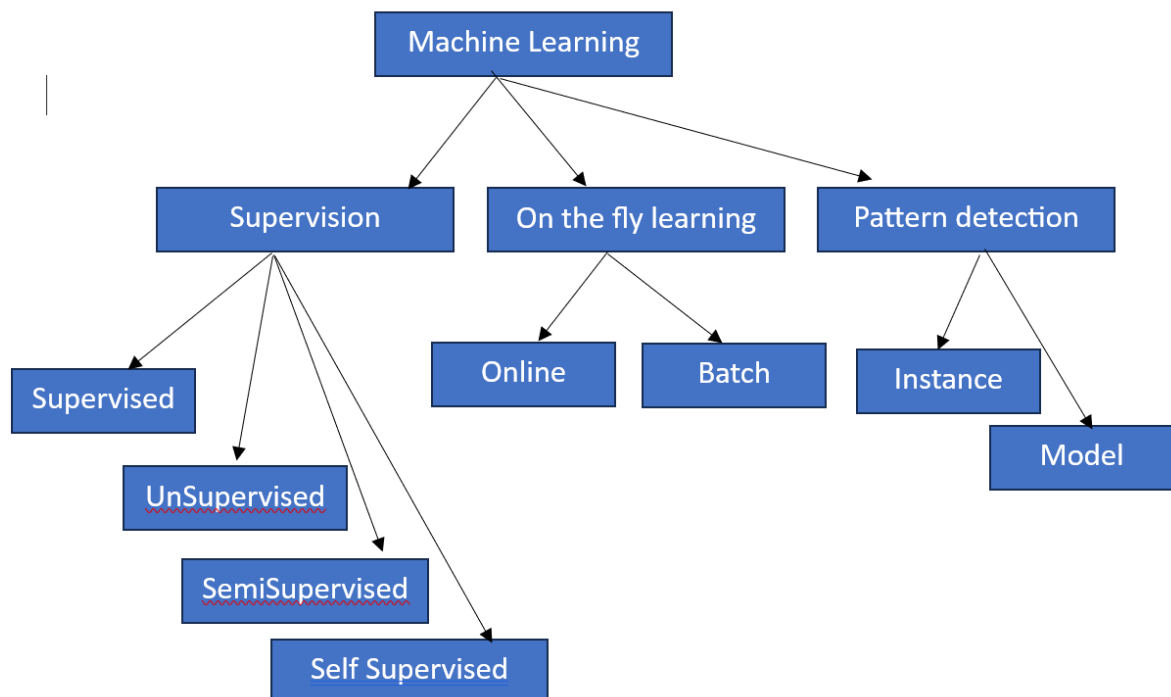
Machine Learning can help humans learn

Types of Machine Learning Systems

There are so many different types of Machine Learning systems that it is useful to classify them in broad categories based on:

- ✚ Whether or not they are trained with human supervision
- ✚ Whether or not they can learn incrementally on the fly
- ✚ Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data.

Classification chart of Machine Learning



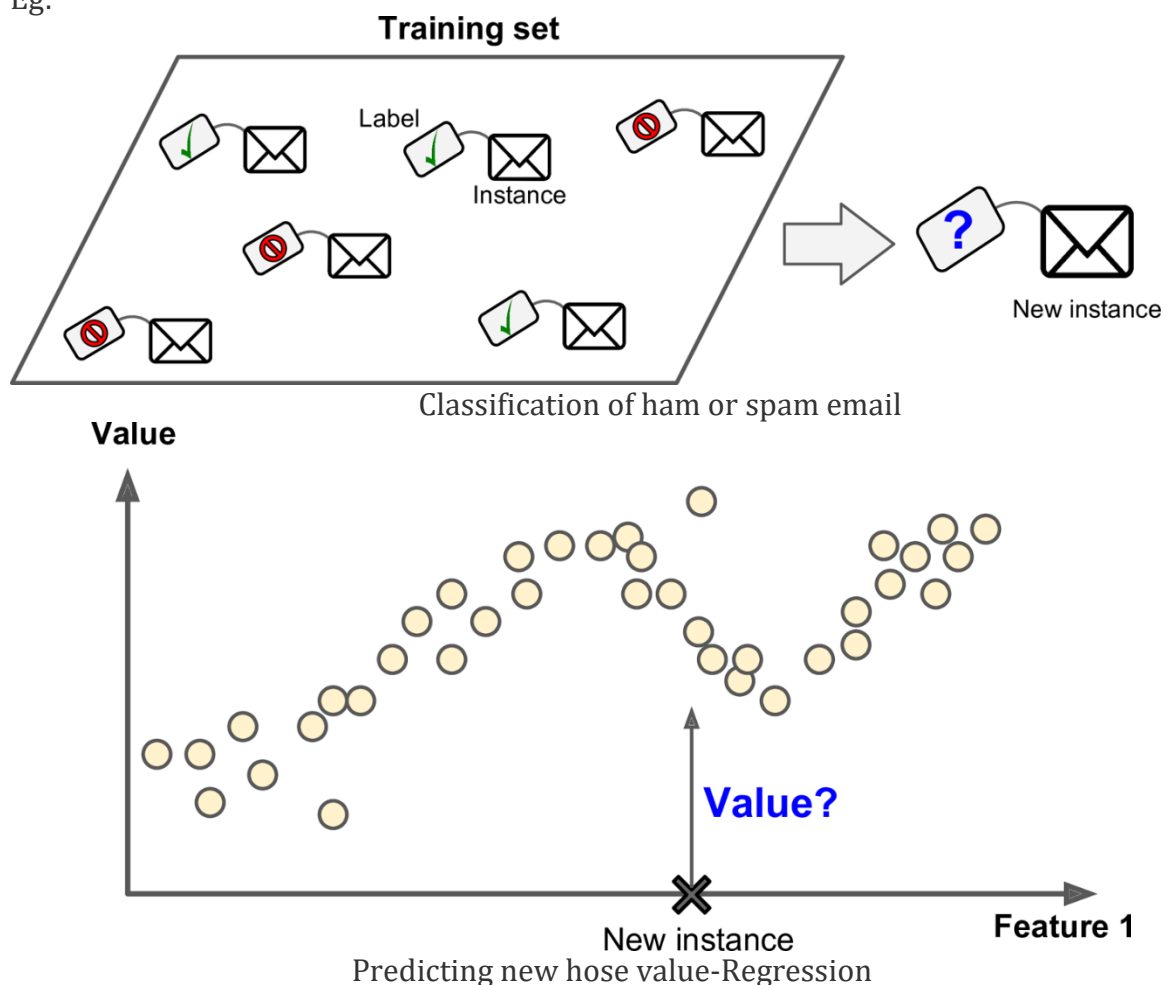
Supervised/Unsupervised Learning

Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning

In supervised learning, the training data you feed to the algorithm includes the desired solutions, called labels. A typical supervised learning task is classification. The spam filter is a good example of this: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.

Another typical task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called predictors. This sort of task is called regression.

Eg:



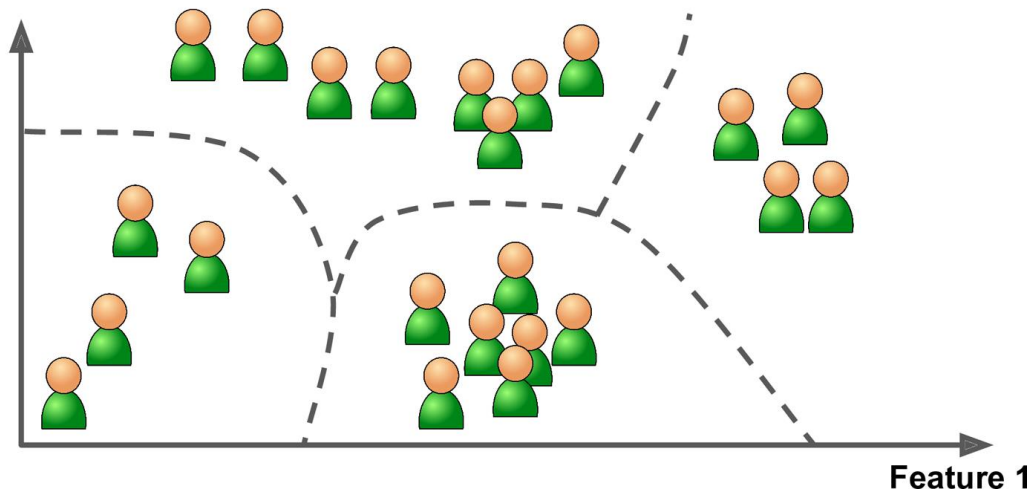
Unsupervised learning

In unsupervised learning, the training data is unlabeled. The system tries to learn without a supervisor. For example, say you have a lot of data about your blog's visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors. At no point do you tell the algorithm which group a visitor belongs to: it finds those connections without your help. For example, it might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on. If you use a hierarchical

clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.

Eg:

Feature 2



Clustering of visitors of a blog

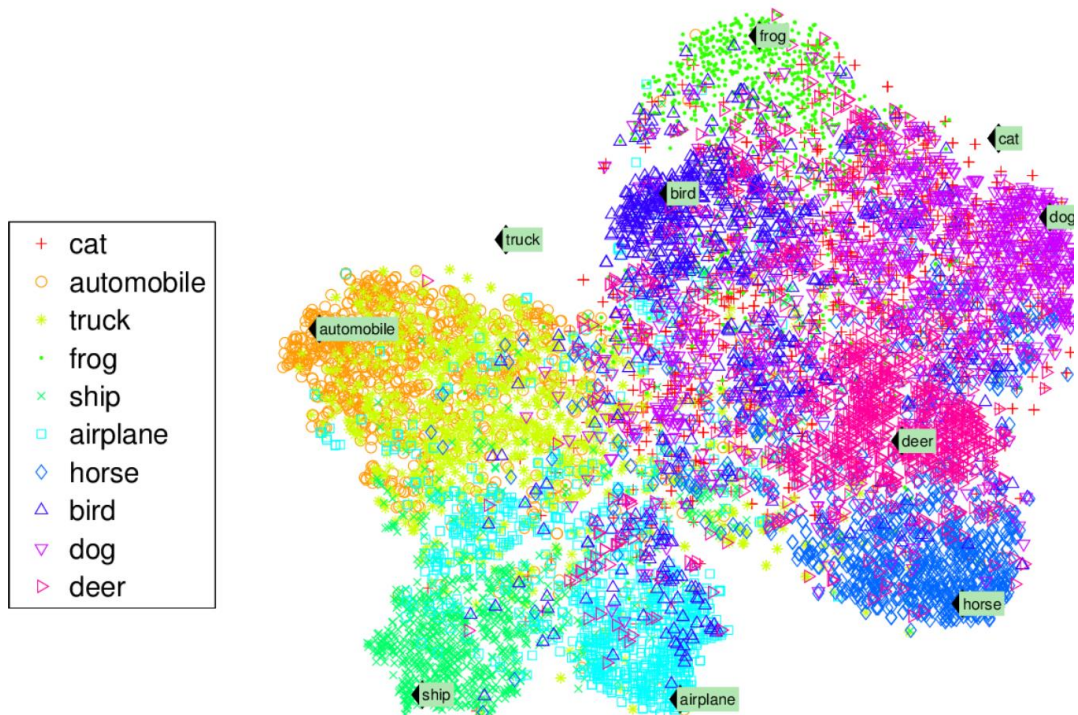
Useases/Applications of unsupervised learning

There are different use cases of unsupervised learning:

(i) Visualization

Visualization algorithms are also good examples of unsupervised learning algorithms: you feed them a lot of complex and unlabeled data, and they output a 2D or 3D representation of your data that can easily be plotted.

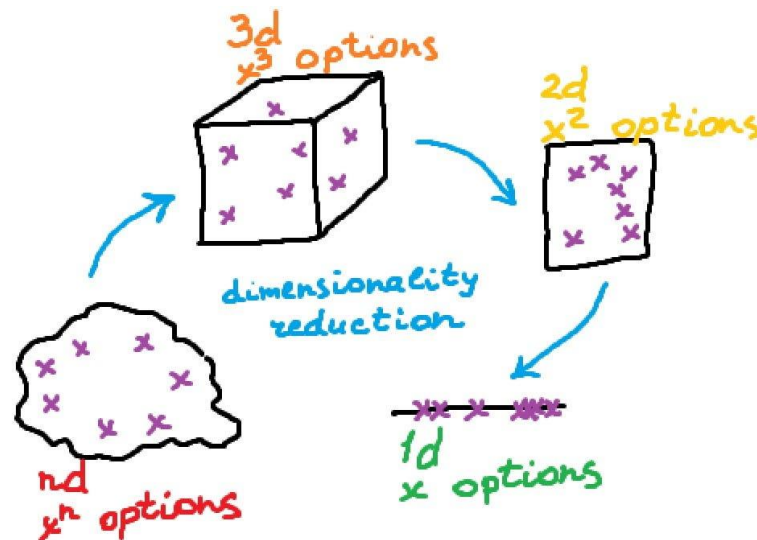
Eg: Visualization of various objects performing semantic clustering



(ii) Dimensionality Reduction

- ❖ The number of input features, variables, or columns present in a given dataset is known as dimensionality,
- ❖ and the process to reduce these features is called dimensionality reduction (Feature extraction).
- ❖ A dataset contains a huge number of input features in various cases, which makes the predictive modeling task more complicated.
- ❖ Because it is very difficult to visualize or make predictions for the training dataset with a high number of features, for such cases, dimensionality reduction techniques are required to use.

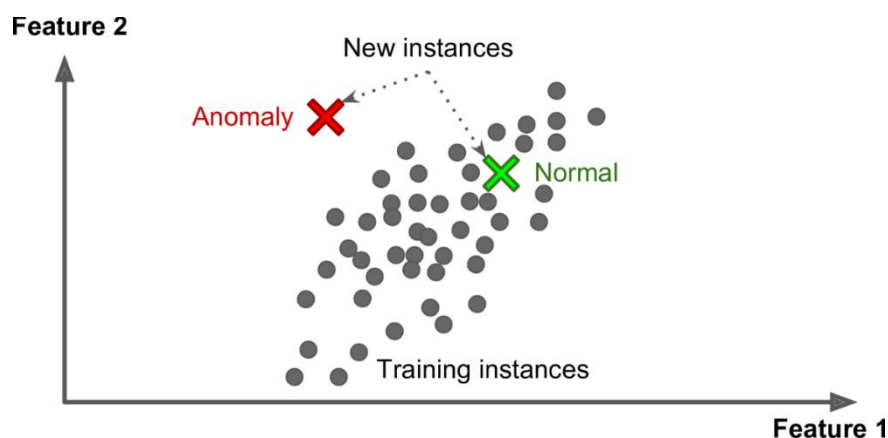
Eg:



(iii) Anomaly detection

Anomaly detection detects unusual credit card transactions to prevent fraud, catching manufacturing defects, or automatically removing outliers from a dataset before feeding it to another learning algorithm. The system is trained with normal instances, and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly.

Eg:

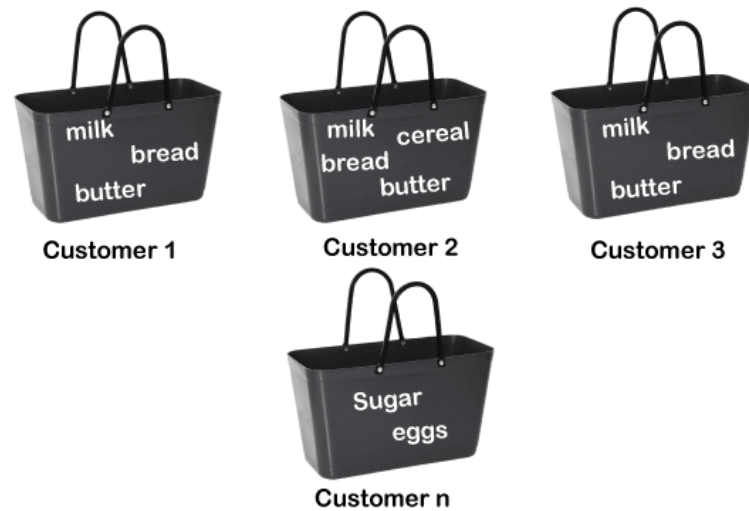


(iv) Association rule mining

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or

associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database..

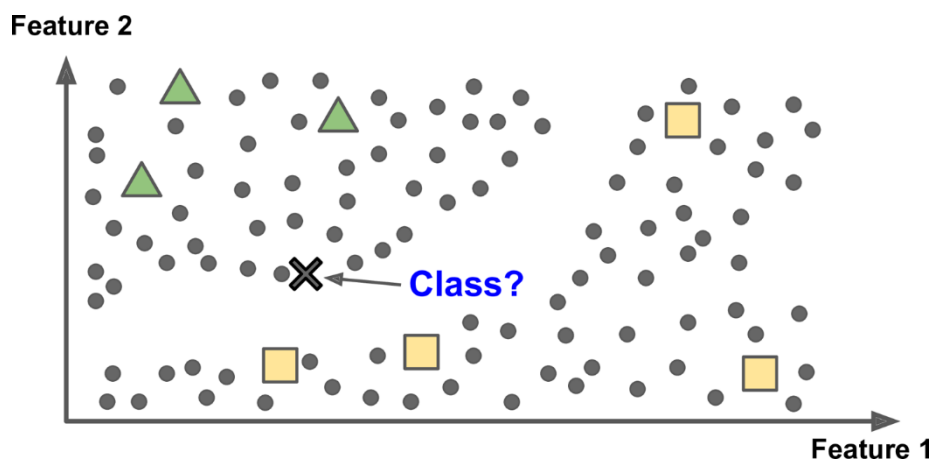
For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:



Semi-supervised learning

Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning. Some photo-hosting services, such as Google Photos, are good examples of this. Once you upload all your family photos to the service, it automatically recognizes that the same person A shows up in photos 1, 5, and 11, while another person B shows up in photos 2, 5, and 7. This is the unsupervised part of the algorithm (clustering). Perform one label per person and it is able to name everyone in every photo, which is useful for searching photo. Most semisupervised learning algorithms are combinations of unsupervised and supervised algorithms.

Eg:



Self supervised learning

Self-supervised learning is a deep learning methodology where a model is pre-trained using unlabelled data and the data labels are generated automatically, which are further used in subsequent iterations as ground truths. The fundamental idea for self-supervised learning is to create supervisory signals by making sense of the unlabeled data provided to it in an unsupervised fashion on the first iteration. Then, the model

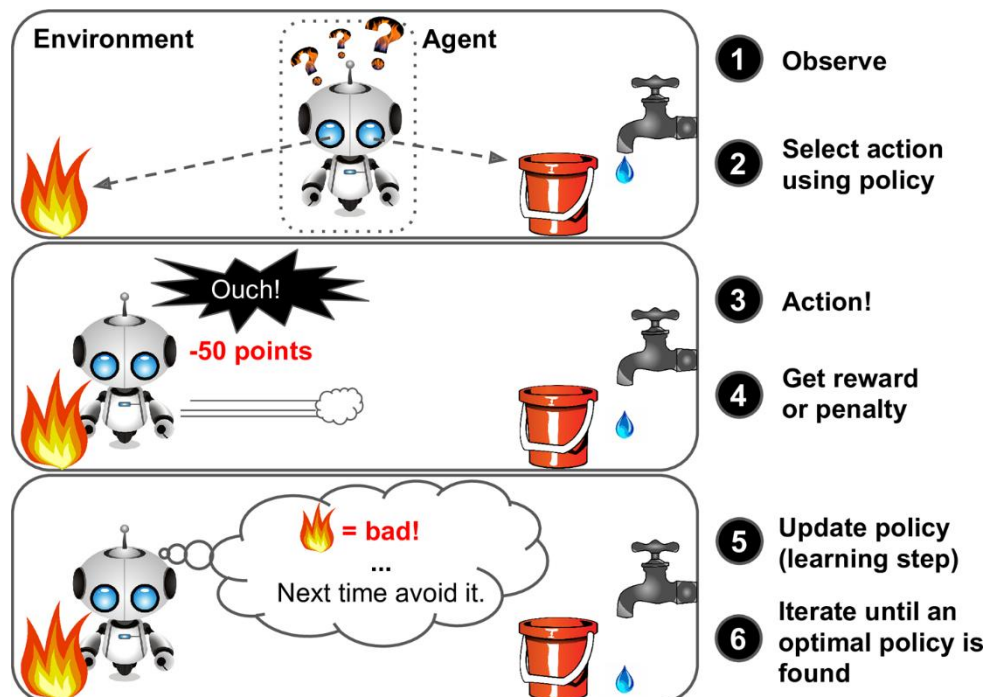
uses the high-confidence data labels among those generated to train the model in subsequent iterations.

For example, a self-supervised learning model might be trained to predict the location of an object in an image given the surrounding pixels to classify a video as depicting a particular action.

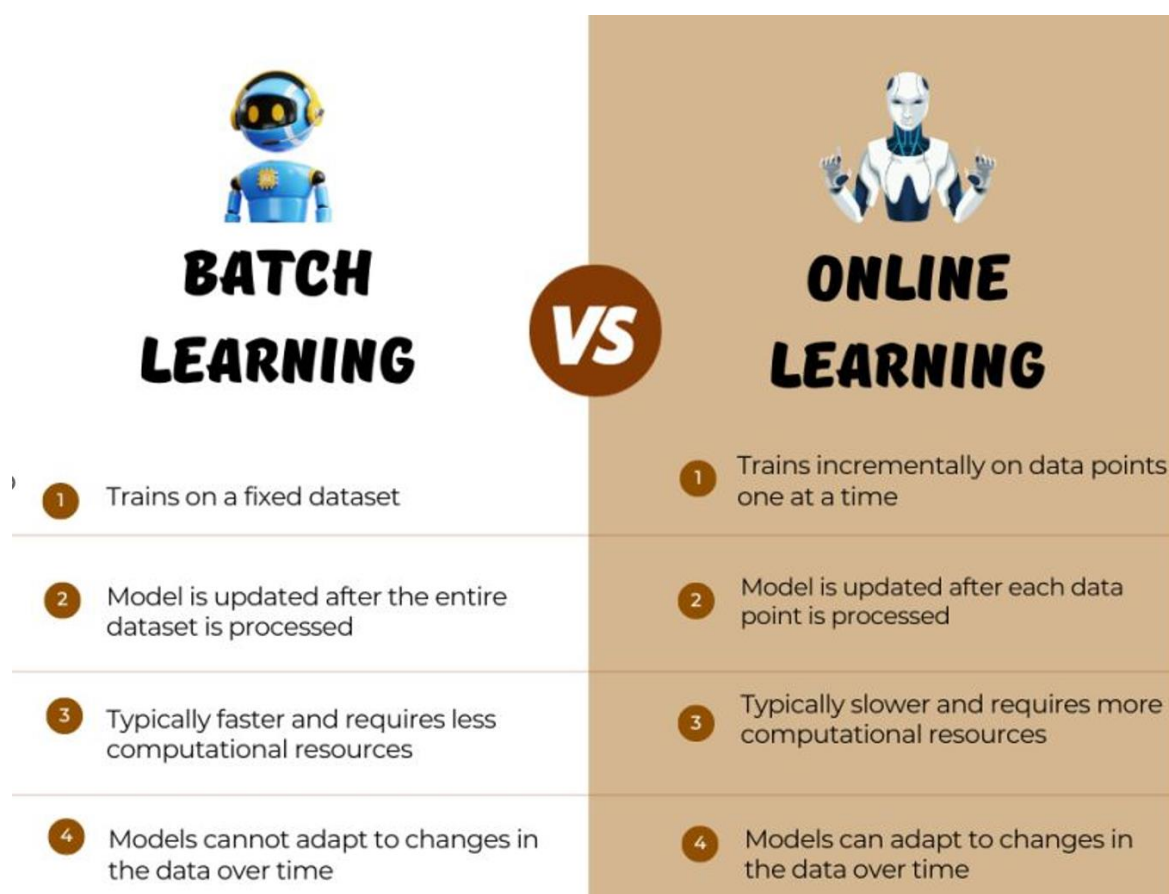
Reinforcement Learning

Reinforcement Learning system uses an agent in this context which can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards). It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

Eg:



Batch and Online Learning



In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning.

Drawbacks:

Handling large amounts of data: Batch learning requires loading the entire dataset into memory for training. This becomes a challenge when dealing with large datasets that exceed the available memory capacity.

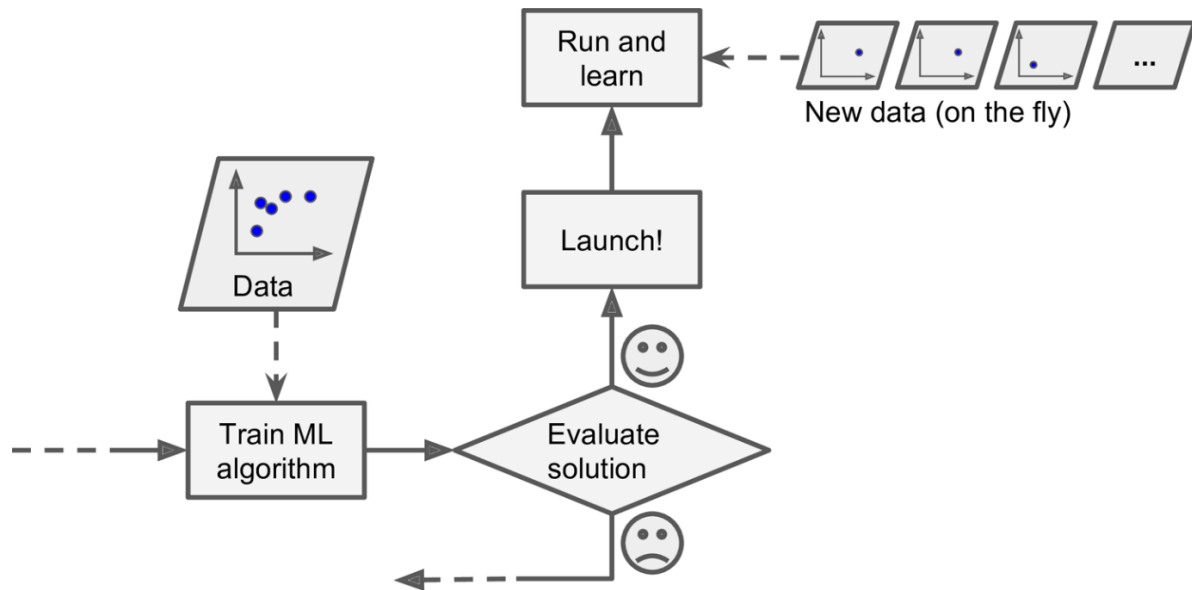
Hardware limitations: Batch learning can be computationally expensive, especially when dealing with complex models or large datasets. Training a model on a single machine may take a significant amount of time and may require high-performance hardware, such as GPUs or specialized processing units.

Availability constraints: In some scenarios, obtaining the entire dataset required for batch learning may not be feasible or practical.

online learning

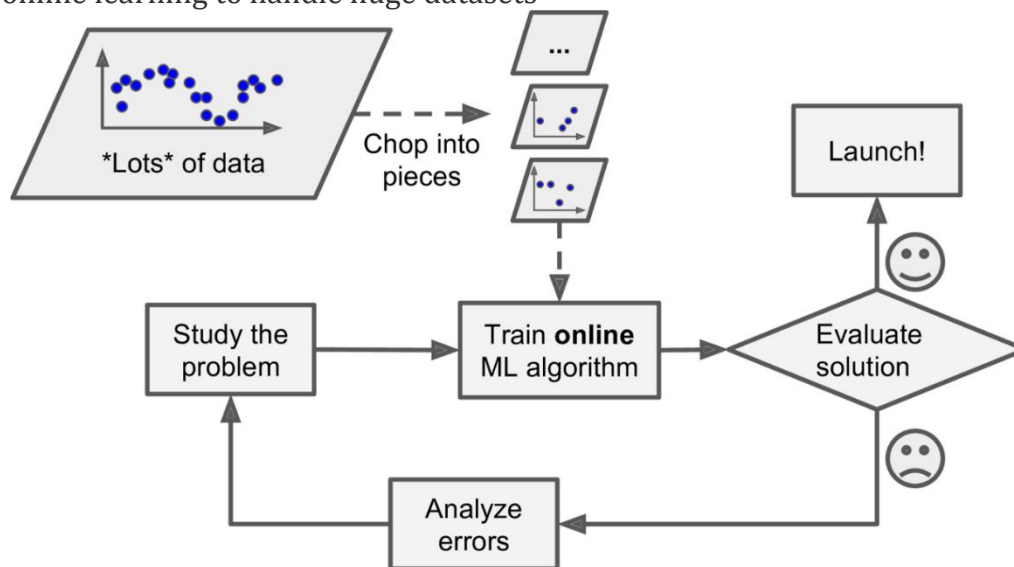
In the online learning, data is fed to the model in small batches, sequentially. These batches are called mini batches. After, each batch of training, your model gets better. since these batches are small chunks of data. so you can perform this training on server (in production) That's why it is called online learning means your model is getting trained when your model is on server.

Eg: Online learning system



Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. It is also a good option if you have limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them.

Using online learning to handle huge datasets



One important parameter of online learning systems is how fast they should adapt to changing data: this is called the learning rate. If you set a high learning rate, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data. Conversely, if you set a low learning rate, the system will have more inertia; that is, it will learn more slowly, but it will also be less sensitive to noise in the new data or to sequences of nonrepresentative data points. A big challenge with online learning is that if bad data is fed to the system, the system's performance will gradually decline.

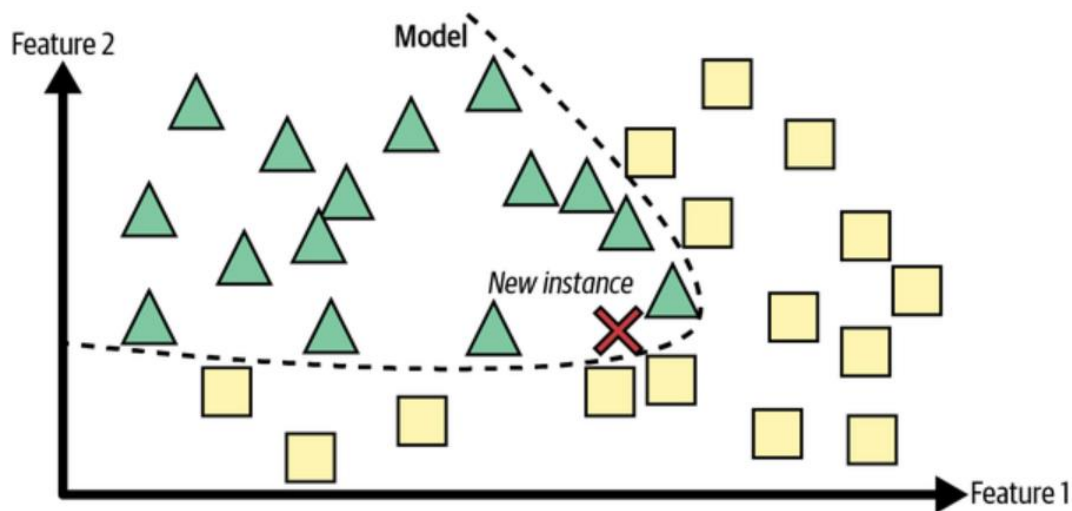
Instance-Based Versus Model-Based Learning

One more way to categorize Machine Learning systems is by how they generalize. There are two main approaches to generalization: instance-based learning and model-based learning.

Model-Based Learning

Model-based learning involves creating a mathematical model that can predict outcomes based on input data. The model is trained on a large dataset and then used to make predictions on new data. The model can be thought of as a set of rules that the machine uses to make predictions. In model-based learning, the training data is used to create a model that can be generalized to new data. The model is typically created using statistical algorithms such as linear regression, logistic regression, decision trees, and neural networks. These algorithms use the training data to create a mathematical model that can be used to predict outcomes.

Eg:



Advantages of Model-Based Learning

1. **Faster predictions:** Model-based learning is typically faster than instance-based learning because the model is already created and can be used to make predictions quickly.
2. **More accurate predictions:** Model-based learning can often make more accurate predictions than instance-based learning because the model is trained on a large dataset and can generalize to new data.
3. **Better understanding of data:** Model-based learning allows you to gain a better understanding of the relationships between input and output variables.

Disadvantages of Model-Based Learning

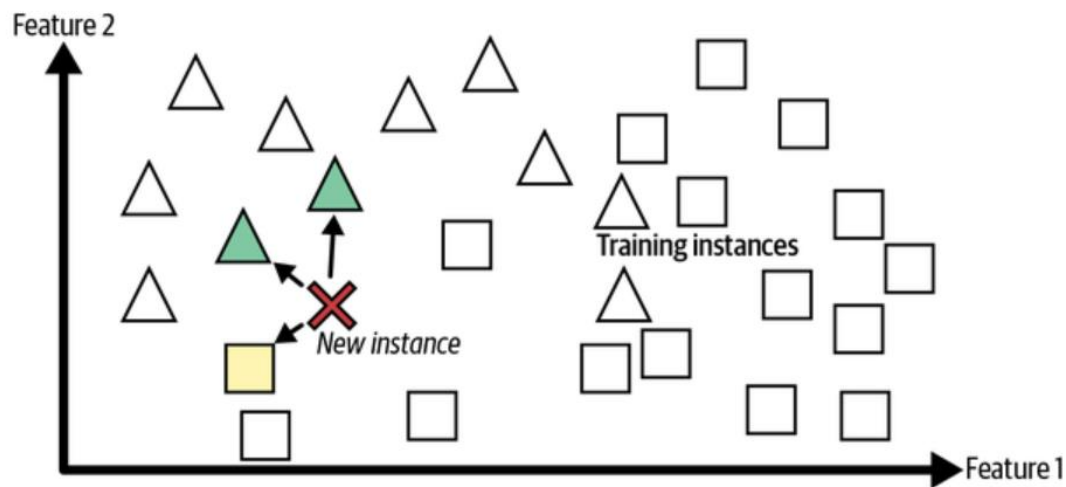
1. **Requires a large dataset:** model-based learning requires a large dataset to train the model.
2. **Requires expert knowledge:** Model-based learning requires expert knowledge of statistical algorithms and mathematical modeling.
3. **Requires expert knowledge:** Model-based learning requires expert knowledge of statistical algorithms and mathematical modeling.

Instance-Based Learning

Instance-based learning involves using the entire dataset to make predictions. The machine learns by storing all instances of data and then using these instances to make predictions on new data. The machine compares the new data to the instances it has seen before and uses the closest match to make a prediction. In instance-based learning, no model is created. Instead, the machine stores all of the training data and uses this data to

make predictions based on new data. Instance-based learning is often used in pattern recognition, clustering, and anomaly detection.

Eg:



Advantages of Instance-Based Learning

1. No need for model creation: Instance-based learning doesn't require creating a model.
2. Can handle small datasets: Instance-based learning can handle small datasets because it doesn't require a large dataset to create a model.
3. More flexibility: Instance-based learning can be more flexible than model-based learning because the machine stores all instances of data and can use this data to make predictions.

Disadvantages of Instance-Based Learning

1. Slower predictions: Instance-based learning is typically slower than model-based learning because the machine has to compare the new data to all instances of data in order to make a prediction.
2. Less accurate predictions: Instance-based learning can often make less accurate predictions than model-based learning because it doesn't have a mathematical model to generalize from.
3. Limited understanding of data: Instance-based learning doesn't provide as much insight into the relationships between input and output variables as model-based learning does.

Usual/Conventional Machine Learning	Instance Based Learning
Prepare the data for model training	Prepare the data for model training. No difference here
Train model from training data to estimate model parameters i.e. discover patterns	Do not train model. Pattern discovery postponed until scoring query received
Store the model in suitable form	There is no model to store
Generalize the rules in form of model, even before scoring instance is seen	No generalization before scoring. Only generalize for each scoring instance individually as and when seen
Predict for unseen scoring instance using model	Predict for unseen scoring instance using training data directly
Can throw away input/training data after model training	Input/training data must be kept since each query uses part or full set of training observations
Requires a known model form	May not have explicit model form
Storing models generally requires less storage	Storing training data generally requires more storage

Main Challenges of Machine Learning

Insufficient Quantity of Training Data

For a toddler to learn what an apple is, all it takes is for you to point to an apple and say “apple” (possibly repeating this procedure a few times). Now the child is able to recognize apples in all sorts of colors and shapes. Machine Learning is not quite there yet; it takes a lot of data for most Machine Learning algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples (unless you can reuse parts of an existing model).

Nonrepresentative Training Data

In order to generalize well, it is crucial that your training data be representative of the new cases you want to generalize to. This is true whether you use instance-based learning or model-based learning. It is crucial to use a training set that is representative of the cases you want to generalize to. This is often harder than it sounds: if the sample is too small, you will have sampling noise (i.e., nonrepresentative data as a result of chance), but even very large samples can be nonrepresentative if the sampling method is flawed. This is called sampling bias.

Eg: you want to build a system to recognize funk music videos. One way to build your training set is to search “funk music” on YouTube and use the resulting videos. But this assumes that YouTube’s search engine returns a set of videos that are representative of all the funk music videos on YouTube. In reality, the search results are likely to be biased

toward popular artists (and if you live in Brazil you will get a lot of “funk carioca” videos, which sound nothing like James Brown)

Poor-Quality Data

If the training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the system to detect the underlying patterns, so your system is less likely to perform well. It is often well worth the effort to spend time cleaning up your training data. The truth is, most data scientists spend a significant part of their time doing just that.

Irrelevant Features

As the saying goes: garbage in, garbage out. Your system will only be capable of learning if the training data contains enough relevant features and not too many irrelevant ones. A critical part of the success of a Machine Learning project is coming up with a good set of features to train on. This process, called feature engineering, involves:

- Feature selection: selecting the most useful features to train on among existing features.
- Feature extraction: combining existing features to produce a more useful one (as we saw earlier, dimensionality reduction algorithms can help).
- Creating new features by gathering new data.

overfitting:

it occurs when your model is too simple to learn the underlying structure of the data. For example, a linear model of life satisfaction is prone to underfit; reality is just more complex than the model, so its predictions are bound to be inaccurate, even on the training examples.

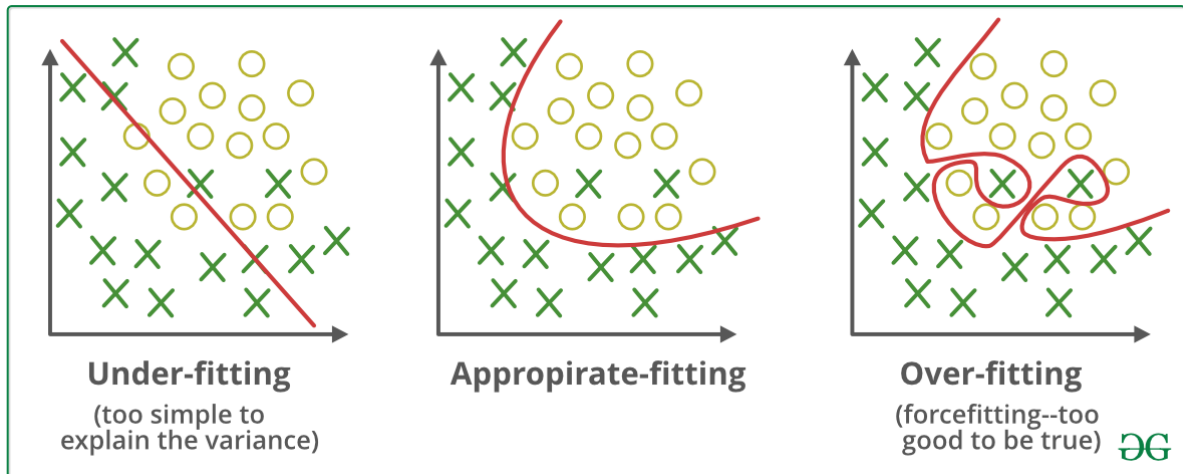
The main options to fix this problem are:

- Selecting a more powerful model, with more parameters
- Feeding better features to the learning algorithm (feature engineering)
- Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

Underfitting

A statistical model or a machine learning algorithm is said to have underfitting when a model is too simple to capture data complexities. It represents the inability of the model to learn the training data effectively result in poor performance both on the training and testing data. In simple terms, an underfit model's are inaccurate, especially when applied to new, unseen examples. It mainly happens when we uses very simple model with overly simplified assumptions. To address underfitting problem of the model, we need to use more complex models, with enhanced feature representation, and less regularization.

Graphs to differentiate overfitting, underfitting and appropriate-fitting



Techniques to fight underfitting and overfitting

Underfitting	Overfitting
More complex model	More simple model
Less regularization	More regularization
Larger quantity of features	Smaller quantity of features
More data can't help	More data can help

Learning

Learning is the action or process of obtaining information or ability through studying, practicing, being instructed, or experiencing something. Learning techniques can be split into five categories:

1. Rote Learning (Memorizing): Memorizing things without understanding the underlying principles or rationale.
2. Instructions (Passive Learning): Learning from a teacher or expert.
3. Analogy (Experience): We may learn new things by applying what we've learned in the past.
4. Inductive Learning (Experience): Formulating a generalized notion based on prior experience.
5. Deductive Learning: Getting new information from old information.











Concept Learning

Concept learning, as a broader term, includes both case-based and instance-based learning. At its core, concept learning involves the extraction of general rules or patterns

from specific instances to make predictions on new, unseen data. The ultimate goal is for the machine to grasp abstract concepts and apply them in diverse contexts.

Concept learning in machine learning is not confined to a single pattern; it spans various approaches, including rule-based learning, neural networks, decision trees, and more. The choice of approach depends on the nature of the problem and the characteristics of the data.

each concept can be thought of as a boolean-valued function defined over this larger set. Example: using concept to classify birds:

Bird	Beak	Shape of the beak	Purpose of the beak
Eagle 		It has a strong, sharp and hooked beak	This shape helps eagle to catch animals as it flies.
Parrot 		It has a sharp and curved beak	This shape helps crack nuts and seeds and to tear fruits.
Duck 		It has a flat and broad beak.	This shape helps it to catch fish and worms in the water.
Sparrow 		It has a small pointed beak.	This shape helps it to pick small grains.
Hummingbird 		It has a straw-like, long and slender beak.	This shape helps it to suck nectar from flowers.

Understanding the Concept:

The set of instances, represented by X , is the list of elements over which the notion is defined. The target idea, represented by c , is the notion of action to be learned. It's a boolean-valued function that's defined over X and may be expressed as:

$$c: X \rightarrow \{0, 1\}$$

So, when we have a subset of the training with certain attributes of the target concept c , the learner's issue is to estimate c from the training data.

The letter H stands for the collection of all conceivable hypotheses that a learner could explore while determining the identification of the target idea.

A learner's objective is to create a hypothesis h that can identify all of the objects in X in such a way that:

$$h(x) = c(x) \text{ for all } x \text{ in } X$$

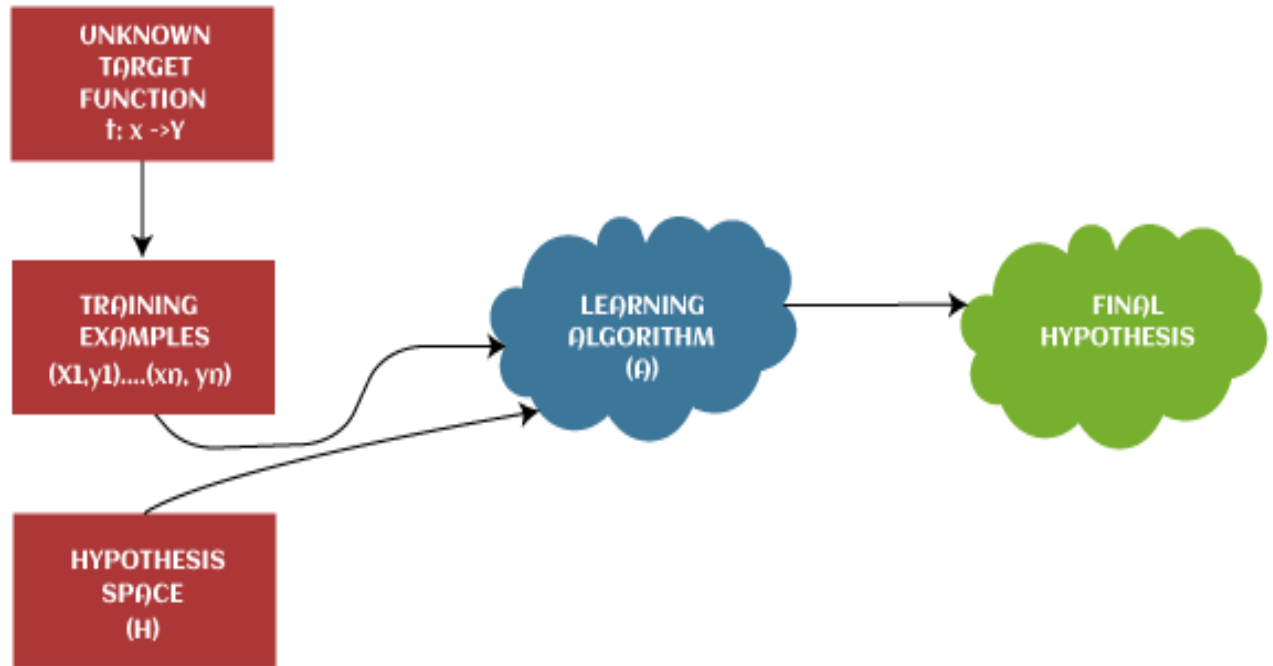
In this sense, there are three things that an algorithm that enables concept learning must have:

1. Details about the training (Past experiences to train our models)
2. Target Conception (Hypothesis to identify data objects)

3. Data objects themselves (For testing the models)

Hypothesis in Machine Learning (ML)

The hypothesis is one of the commonly used concepts of statistics in Machine Learning. It is specifically used in Supervised Machine learning, where an ML model learns a function that best maps the input to corresponding outputs with the help of an available dataset.



In supervised learning techniques, the main aim is to determine the possible hypothesis out of hypothesis space that best maps input to the corresponding or correct outputs. There are some common methods given to find out the possible hypothesis from the Hypothesis space, where hypothesis space is represented by uppercase-h (H) and hypothesis by lowercase-h (h).

Hypothesis space (H):

Hypothesis space is defined as a set of all possible legal hypotheses; hence it is also known as a hypothesis set. It is used by supervised machine learning algorithms to determine the best possible hypothesis to describe the target function or best maps input to output.

Hypothesis (h):

It is defined as the approximate function that best describes the target in supervised machine learning algorithms. It is primarily based on data as well as bias and restrictions applied to data.

Designing a Learning System in Machine Learning

Designing a learning system in machine learning requires careful consideration of several key factors, including the type of data being used, the desired outcome, and the available resources.

- ✚ The first step in designing a learning system in machine learning is to identify the type of data that will be used. This can include structured data, such as numerical and categorical data, as well as unstructured data, such as text and images. The

type of data will determine the type of machine learning algorithms that can be used and the preprocessing steps required.

- ✚ Once the data has been identified, the next step is to determine the desired outcome of the learning system. This can include classifying data, making predictions, or identifying patterns in the data. The desired outcome will determine the type of machine learning algorithm that should be used, as well as the evaluation metrics that will be used to measure the performance of the learning system.
- ✚ Next, the resources available for the learning system must be considered. This includes the amount of data available, the computational power available, and the amount of time available to train the model. These resources will determine the complexity of the machine learning algorithm that can be used and the amount of data that can be used for training.
- ✚ Once the data, desired outcome, and resources have been identified, it is time to select a machine-learning algorithm and begin the training process. Decision trees, SVMs, and neural networks are examples of common algorithms. It is crucial to assess the effectiveness of the learning system using the right assessment measures, such as recall, accuracy, and precision.
- ✚ After the learning system is trained, it is important to fine-tune the model by adjusting the parameters and hyperparameters. This can be done using techniques such as cross-validation and grid search. The final model should be tested on a hold-out test set to evaluate its performance on unseen data.

checkers learning problem

For a checkers learning problem, the three elements will be,

1. Task T: To play checkers
2. Performance measure P: Total percent of the game won in the tournament.
3. Training experience E: A set of games played against itself

Training experience

During the design of the checker's learning system, the type of training experience available for a learning system will have a significant effect on the success or failure of the learning.

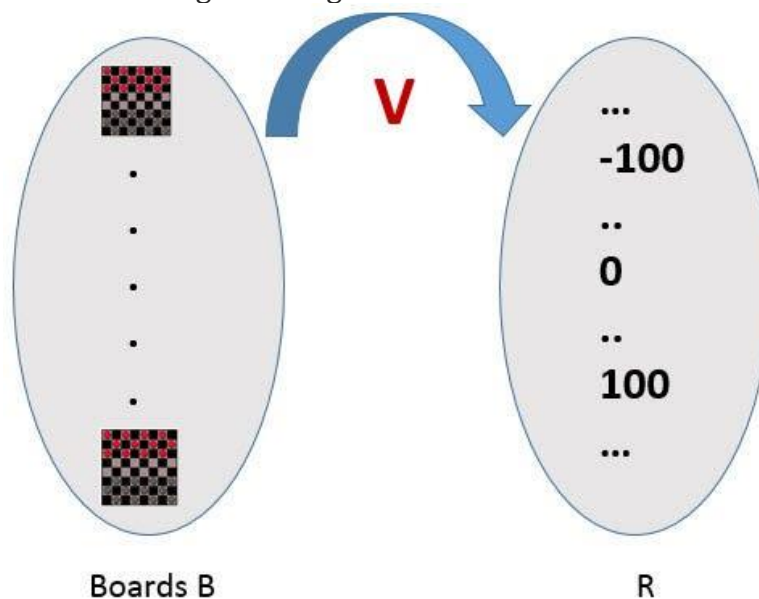
1. Direct or Indirect training experience — In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of games.
2. Supervised — The training experience will be labeled, which means, all the board states will be labeled with the correct move. So the learning takes place in the presence of a supervisor.
Unsupervised — The training experience will be unlabeled, which means, all the board states will not have the moves. So the learner generates random games and plays against itself with no supervision involvement.
Semi-supervised — Learner generates game states and asks the supervisor for help in finding the correct move if the board state is confusing.
3. Is the training experience good —
Performance is best when training examples and test examples are from the same/a similar distribution.

Choosing the Target Function

In this design step, we need to determine exactly what type of knowledge has to be learned and it's used by the performance program. Here there are 2 considerations — direct and indirect experience.

During the direct experience, the checkers learning system, it needs only to learn how to choose the best move among some large search space. We need to find a target function that will help us choose the best move among alternatives. Let us call this function ChooseMove and use the notation $\text{ChooseMove} : B \rightarrow M$ to indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M .

When there is an indirect experience, it becomes difficult to learn such function. How about assigning a real score to the board state. So the function be $V : B \rightarrow R$ indicating that this accepts as input any board from the set of legal board states B and produces an output a real score. This function assigns the higher scores to better board states.



Choosing a representation for the Target Function

Now its time to choose a representation that the learning program will use to describe the function \hat{V} that it will learn.

choose a simple representation for any given board state, the function \hat{V} will be calculated as a linear combination of the following board features:

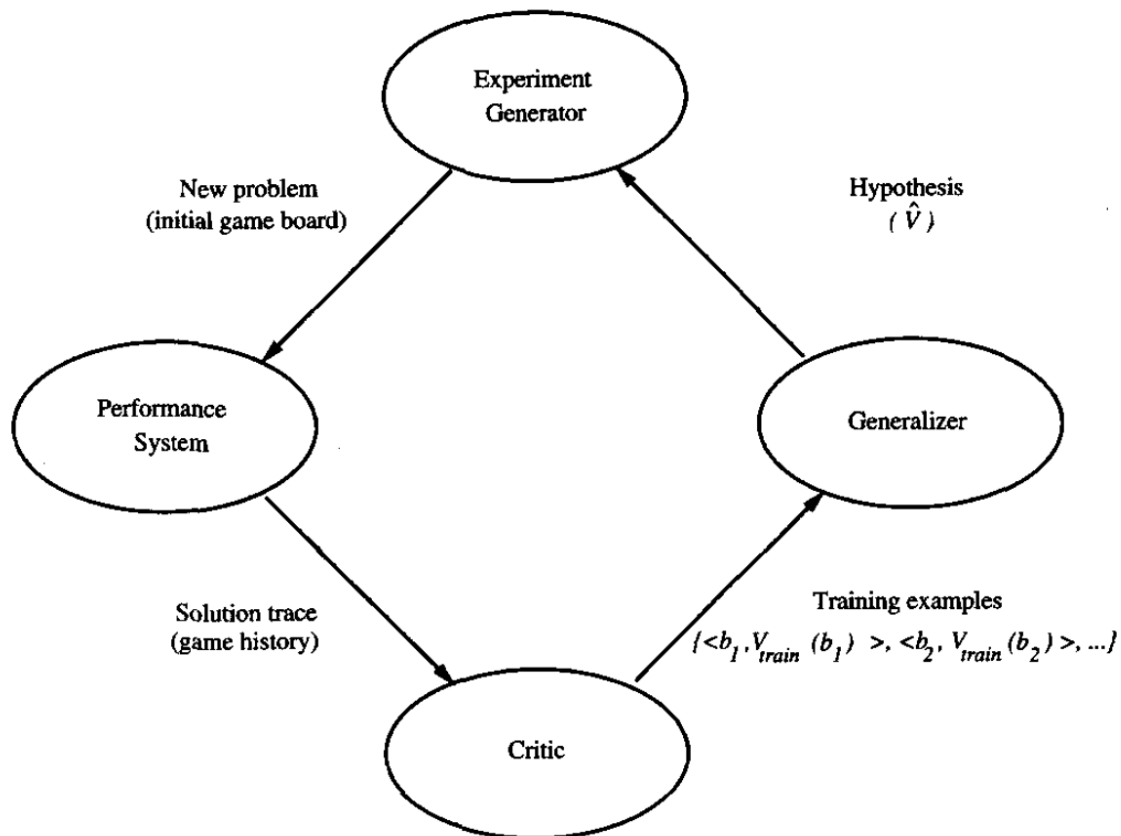
- $x_1(b)$ — number of black pieces on board b
- $x_2(b)$ — number of red pieces on b
- $x_3(b)$ — number of black kings on b
- $x_4(b)$ — number of red kings on b
- $x_5(b)$ — number of red pieces threatened by black (i.e., which can be taken on black's next turn)
- $x_6(b)$ — number of black pieces threatened by red

$$\hat{V} = w_0 + w_1 \cdot x_1(b) + w_2 \cdot x_2(b) + w_3 \cdot x_3(b) + w_4 \cdot x_4(b) + w_5 \cdot x_5(b) + w_6 \cdot x_6(b)$$

Final Design for Checkers Learning system

1. The performance System — Takes a new board as input and outputs a trace of the game it played against itself.

2. The Critic — Takes the trace of a game as an input and outputs a set of training examples of the target function.
3. The Generalizer — Takes training examples as input and outputs a hypothesis that estimates the target function. Good generalization to new cases is crucial.
4. The Experiment Generator — Takes the current hypothesis (currently learned function) as input and outputs a new problem (an initial board state) for the performance system to explore.



Final design of the checkers system

Perspectives and Issues in Machine Learning

Following are the list of issues in machine learning:

1. What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
2. How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
3. When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
4. What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
5. What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?

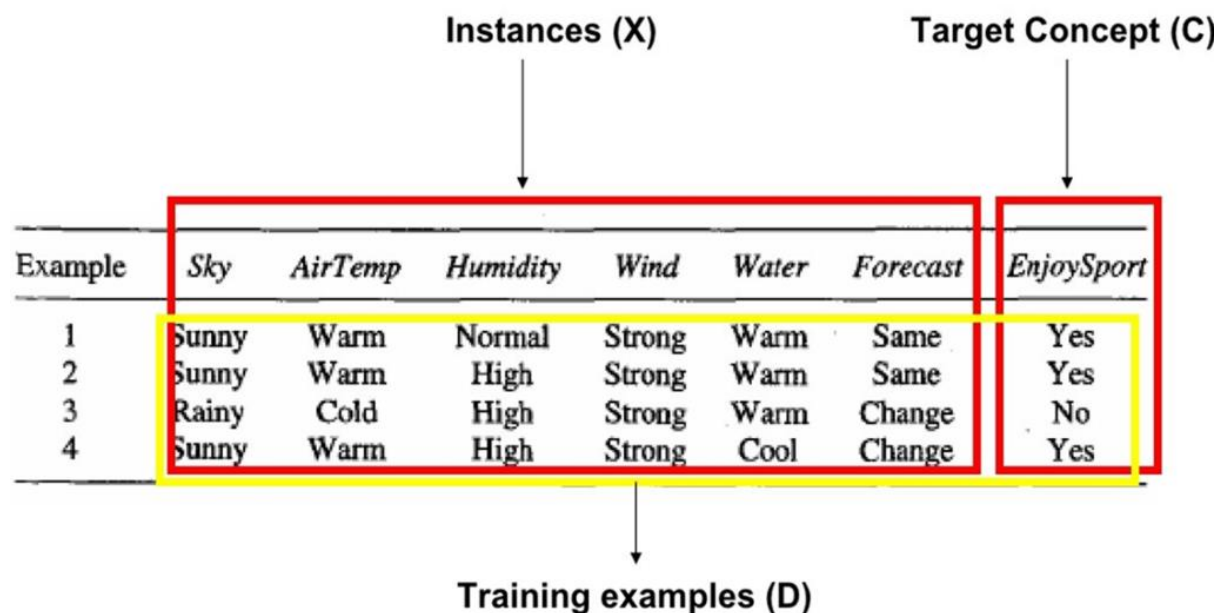
5. How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

Concept Learning in Machine Learning

Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples. Consider the example task of learning the target concept “days on which person enjoys his favorite water sport.” Below Table describes a set of example days, each represented by a set of attributes. The attribute EnjoySport indicates whether or not a person enjoys his favorite water sport on this day. The task is to learn to predict the value of EnjoySport for an arbitrary day, based on the values of its other attributes.

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Concept Learning Notations



Hypothesis representation for Machine Learning

In particular, let each hypothesis be a vector of six constraints, specifying the values of the six attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.

For each attribute, the hypothesis will either

- indicate by a “?” that any value is acceptable for this attribute,
- specify a single required value (e.g., Warm) for the attribute, or
- indicate by a “ \emptyset ” that no value is acceptable.

If some instance x satisfies all the constraints of hypothesis h , then h classifies x as a positive example ($h(x) = 1$).

To illustrate, the hypothesis that a person enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression

$(?, \text{Cold}, \text{High}, ?, ?, ?)$

Most General and Specific Hypothesis

The most general hypothesis-that every day is a positive example-is represented by
 $(?, ?, ?, ?, ?, ?)$

and the most specific possible hypothesis-that no day is a positive example-is represented by

$(\emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset)$

Instance Space

Consider, for example, the instances X and hypotheses H in the EnjoySport learning task. Given that the attribute *Sky* has three possible values, and that *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast* each have two possible values, the instance space X contains exactly $3 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 96$ distinct instances.

Example:

Let's assume there are two features $F1$ and $F2$ with $F1$ has A and B as possibilities and $F2$ as X and Y as possibilities.

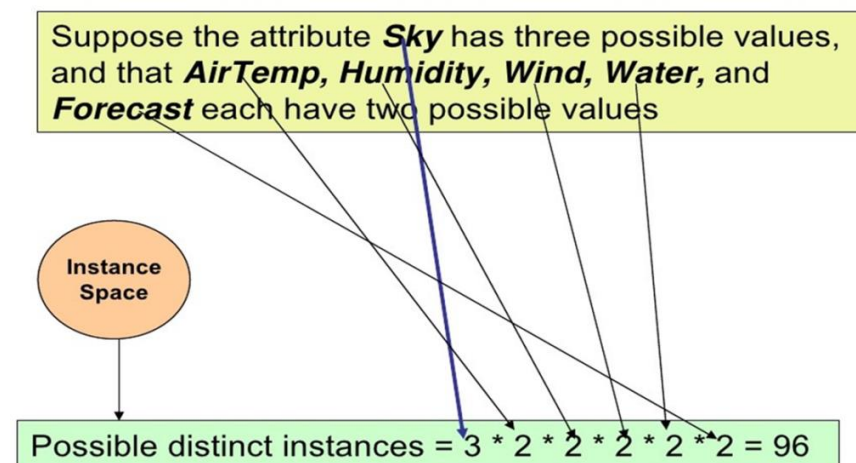
$F1 \rightarrow A, B$

$F2 \rightarrow X, Y$

Instance Space: $(A, X), (A, Y), (B, X), (B, Y)$ – 4 Examples

Hypothesis Space: $(A, X), (A, Y), (A, \emptyset), (A, ?), (B, X), (B, Y), (B, \emptyset), (B, ?), (\emptyset, X), (\emptyset, Y), (\emptyset, \emptyset), (\emptyset, ?), (? , X), (? , Y), (? , \emptyset), (? , ?)$ – 16

Hypothesis Space: $(A, X), (A, Y), (A, ?), (B, X), (B, Y), (B, ?), (? , X), (? , Y), (? , ?)$ – 10



Hypothesis Space

Similarly there are $5 \cdot 4 \cdot 4 \cdot 4 \cdot 4 \cdot 4 = 5120$ syntactically distinct hypotheses within H . Notice, however, that every hypothesis containing one or more “ \emptyset ” symbols represents the empty set of instances; that is, it classifies every instance as negative.

Therefore, the number of semantically distinct hypotheses is only $1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$.

General-to-Specific Ordering of Hypotheses

To illustrate the general-to-specific ordering, consider the two hypotheses

$h_1 = (\text{Sunny}, ?, ?, \text{Strong}, ?, ?)$

$h_2 = (\text{Sunny}, ?, ?, ?, ?, ?)$

Now consider the sets of instances that are classified positive by h_1 and by h_2 . Because h_2 imposes fewer constraints on the instance, it classifies more instances as positive.

In fact, any instance classified positive by h_1 will also be classified positive by h_2 . Therefore, we say that h_2 is more general than h_1 .

For any instance x in X and hypothesis h in H , we say that x satisfies h if and only if $h(x) = 1$.

We define the `more_general_than_or_equal_to` relation in terms of the sets of instances that satisfy the two hypotheses.

FIND-S algorithm

Find-S algorithm, is a machine learning algorithm that seeks to find a maximally specific hypothesis based on labeled training data. It starts with the most specific hypothesis and generalizes it by incorporating positive examples. It ignores negative examples during the learning process. The algorithm's objective is to discover a hypothesis that accurately represents the target concept by progressively expanding the hypothesis space until it covers all positive instances.

-
1. Initialize h to the most specific hypothesis in H
 2. For each positive training instance x
 - For each attribute constraint a_i in h
 - If the constraint a_i is satisfied by x
 - Then do nothing
 - Else replace a_i in h by the next more general constraint that is satisfied by x
 3. Output hypothesis h
-

TABLE 2.3
FIND-S Algorithm.

Inner working of Find-S algorithm

The Find-S algorithm operates on a hypothesis space to find a general hypothesis that accurately represents the target concept based on labeled training data. Let's delve into the inner workings of the algorithm –

- Initialization – The algorithm starts with the most specific hypothesis, denoted as h . This initial hypothesis is the most restrictive concept and typically assumes no positive examples. It may be represented as $h = \langle \emptyset, \emptyset, \dots, \emptyset \rangle$, where \emptyset denotes "don't care" or "unknown" values for each attribute.
- Iterative Process – The algorithm iterates through each training example and refines the hypothesis based on whether the example is positive or negative.
 - ✚ For each positive training example (an example labeled as the target class), the algorithm updates the hypothesis by generalizing it to include the attributes of the example. The hypothesis becomes more general as it covers more positive examples.

- ✚ For each negative training example (an example labeled as a non-target class), the algorithm ignores it as the hypothesis should not cover negative examples. The hypothesis remains unchanged for negative examples.
- Generalization – After processing all the training examples, the algorithm produces a final hypothesis that covers all positive examples while excluding negative examples. This final hypothesis represents the generalized concept that the algorithm has learned from the training data.

During the iterative process, the algorithm may introduce "don't care" symbols or placeholders (often denoted as "?") in the hypothesis for attributes that vary among positive examples. This allows the algorithm to generalize the concept by accommodating varying attribute values. The algorithm discovers patterns in the training data and provides a reliable representation of the concept being learned.

Example:

Consider the data set:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Step1: Initialization

$H = [<0,0,0,0,0,0>]$

Step 2: Consider first sample, compare the sample value and hypothesis values one by one and make changes:

$H = [<\text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same}>]$

Step 3: Consider second sample as it is also positive

$H = [<\text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same}>]$

Step 4: Skip third sample as it is negative and then consider fourth sample

$H = [<\text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ?>]$

The key property of the FIND-S algorithm —

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples

FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct

Unanswered Questions by Find-S algorithm in Machine Learning

1. Has the learner converged to the correct target concept? Although FIND-S will find a hypothesis consistent with the training data, it has no way to determine whether it has found the only hypothesis in H consistent with the data

2. Why prefer the most specific hypothesis? In case there are multiple hypotheses consistent with the training examples, FIND-S will find the most specific.
3. Are the training examples consistent? In most practical learning problems there is some chance that the training examples will contain at least some errors or noise.
4. What if there are several maximally specific consistent hypotheses? In the hypothesis language H for the EnjoySport task, there is always a unique, most specific hypothesis consistent with any set of positive examples.

Consistent and Version Space

A hypothesis h is consistent with a set of training examples D if and only if $h(x) = c(x)$ for each example $(x, c(x))$ in D .

$$\text{Consistent}(h, D) \equiv (\forall \langle x, c(x) \rangle \in D) h(x) = c(x)$$

The version space, denoted $VS_{H,D}$ with respect to hypothesis space H and training examples D , is the subset of hypotheses from H consistent with the training examples in D

$$VS_{H,D} \equiv \{h \in H \mid \text{Consistent}(h, D)\}$$

The List-Then-Eliminate algorithm

One obvious way to represent the version space is simply to list all of its members. This leads to a simple learning algorithm, which we might call the List-Then-Eliminate algorithm. The algorithm is as follows :

1. **VersionSpace** a list containing every hypothesis in H
2. For each training example, $(x, c(x))$
remove from **VersionSpace** any hypothesis h for which $h(x) \neq c(x)$
3. Output the list of hypotheses in **VersionSpace**

Representation for Version Spaces

we can represent the version space in terms of its most specific and most general members. For the enjoysport training examples D , we can output the below list of hypothesis which are consistent with D . In other words, the below list of hypothesis is a version space.

h1	Sunny	?	?	?	?	?
h2	?	Warm	?	?	?	?
h3	Sunny	?	?	Strong	?	?
h4	Sunny	Warm	?	?	?	?
h5	?	Warm	?	Strong	?	?
h6	Sunny	Warm	?	Strong	?	?

In the list of hypothesis, there are two extremes representing general (h1 and h2) and specific (h6) hypothesis. Let's define these 2 extremes as general boundary G and specific boundary S .

Definition — G

The general boundary G, with respect to hypothesis space H and training data D, is the set of maximally general members of H consistent with D.

Definition — S

The specific boundary S, with respect to hypothesis space H and training data D, is the set of minimally general (i.e., maximally specific) members of H consistent with D.

Candidate Elimination algorithm

The Candidate-Elimination algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training example.

Step 1: Read the dataset

Step 2: Initialize S and G

Step 3: Read a sample from dataset

Step 4: If sample is not positive go to Step 5

Step 4(a): If first sample, store all features to S. Go to Step 4(d)

Step 4(b): Otherwise, check if feature not same as S.

Step 4(c): If not, store '?' to S.

Step 4(d): Check if G is not '?' and S is '?', then make G as '?'

Step 5: For negative sample, check feature is not same as S and S is not '?'

Step 5(a): If yes, then store S to G. Else G='?'

Step 6: Check whether all samples are over. If no goto Step 3

Step 7: Display S and G

Example:

Consider the dataset:

Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

Initial Values:

$G_0 = [<?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>]$

$S_0 = [<0, 0, 0, 0, 0, 0>]$

Step1: For first sample (positive – update S)

$G_1 = [<?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>]$

$S_1 = [<\text{Sunny}, \text{Warm}, \text{Normal}, \text{Strong}, \text{Warm}, \text{Same}>]$

Step2: For second sample (positive – update S)

$G_2 = [<?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?>]$

$S_2 = [<\text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same}>]$

Step 3: For third sample (negative – update G)

G3=[<Sunny,?,?,?,?,>, <?,Warm,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?Same>]

S3=[<Sunny,Warm,?,Strong,Warm,Same>]

Step 4: For fourth sample (positive – update S)

G4=[<Sunny,?,?,?,?,>, <?,Warm,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?>, <?,?,?,?,?>]

S4=[<Sunny,Warm,?,Strong,?,?>]

Inductive Bias

Every machine learning algorithm with any ability to generalize beyond the training data that it sees has, by definition, some type of inductive bias. That is, there is some fundamental assumption or set of assumptions that the learner makes about the target function that enables it to generalize beyond the training data. The candidate elimination algorithm converges towards the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.

- What if the target concept is not contained in the hypothesis space?
- Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
- How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
- How does the size of the hypothesis space influence the number of training examples that must be observed?

The following three learning algorithms are listed from weakest to strongest bias.

1. Rote-learning : storing each observed training example in memory. If the instance is found in memory, the stored classification is returned.

Inductive bias : nothing — Weakest bias

2. Candidate-Elimination algorithm : new instances are classified only in the case where all members of the current version space agree in the classification.

Inductive bias : Target concept can be represented in its hypothesis space

3. Find-S : find the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances.

Inductive bias : Target concept can be represented in its hypothesis space + All instances are negative instances unless the opposite is entailed by its other knowledge — Strongest bias

MODULE-II

Working with real data

Many of the Machine Learning Crash Course Programming Exercises use the California housing data set, which contains data drawn from the 1990 U.S. Census. The following table provides descriptions, data ranges, and data types for each feature in the data set.

Column title	Description
longitude	A measure of how far west a house is; a more negative value is farther west
latitude	A measure of how far north a house is; a higher value is farther north
housingMedianAge	Median age of a house within a block; a lower number is a newer building
totalRooms	Total number of rooms within a block
totalBedrooms	Total number of bedrooms within a block
population	Total number of people residing within a block
households	Total number of households, a group of people residing within a home unit, for a block
medianIncome	Median income for households within a block of houses (measured in tens of thousands of US Dollars)
medianHouseValue	Median house value for households within a block (measured in US Dollars)
Ocean proximity	The distance from the house to ocean expressed as different categories

Exploring the dataset

Google colab is used to operate on this data set and perform machine learning preprocessing operations and machine learning techniques.

#Code snippet to load the dataset

```
import pandas as pd
housing=pd.read_csv("/content/sample_data/housing.csv",sep=",")
```

#Code snippet for descriptive statistics

```
housing.head() #Display first five records
```

```
housing.info()
```

#Get metadata information like number of samples and datatypes of each column and number of non-null values.

#Working with categorical column attribute

#Number of instances of each category

```
housing["ocean_proximity"].value_counts()
```

Get descriptive stats

```
housing.describe()
```

Different statistics like count, mean, standard deviation, minimum, 25%, 50% and 75% data and maximum values are displayed.

```
#Visualization
```

```
import matplotlib.pyplot as plt
housing.hist(bins=50,figsize=(12,8))
plt.show()
```

Dataset splitting

Two standard techniques for splitting data set is random shuffling and stratified sampling. A simple random sample is used to represent the entire data population and randomly selects individuals from the population without any other consideration.

A stratified random sample, on the other hand, first divides the population into smaller groups, or strata, based on shared characteristics. Therefore, a stratified sampling strategy will ensure that members from each subgroup are included in the data analysis.

Code for simple random sampling:

```
import numpy as np
def shuffle_and_split(dataset,test_ratio):
    test_size=int(test_ratio*len(dataset))
    np.random.seed(42)
    shuffled_indices=np.random.permutation(len(dataset))
    test_indices=shuffled_indices[:test_size]
    train_indices=shuffled_indices[test_size:]
    return dataset.iloc[train_indices],dataset.iloc[test_indices]
train_data,test_data=shuffle_and_split(housing,0.2)
```

Code for stratified random sampling:

```
#create income categories
import matplotlib.pyplot as plt
housing["income_cat"]=pd.cut(housing["median_income"],
                             bins=[0,1.5,3.0,4.5,6,np.inf],labels=[1,2,3,4,5])
#Stratified sampling
from sklearn.model_selection import StratifiedShuffleSplit
splitter=StratifiedShuffleSplit(n_splits=1,test_size=0.2,random_state=42)
strat_splits=[]
for train_index,test_index in splitter.split(housing,housing['income_cat']):
    train_set=housing.iloc[train_index]
    test_set=housing.iloc[test_index]
    strat_splits.append([train_set,test_set])
```

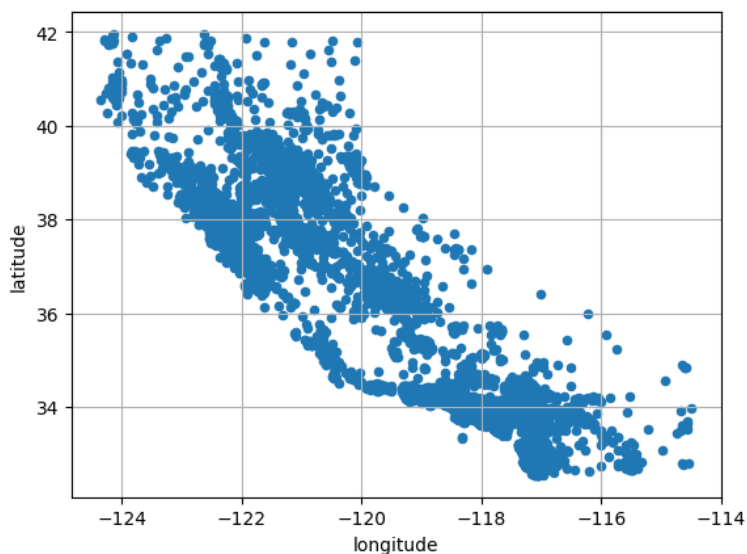
Exploratory data analysis

Exploratory data analysis is one of the basic and essential steps of a data science project. A data scientist involves almost 70% of his work in doing the EDA of the dataset. In this article, we will discuss what is Exploratory Data Analysis (EDA) and the steps to perform EDA. Key aspects of EDA include:

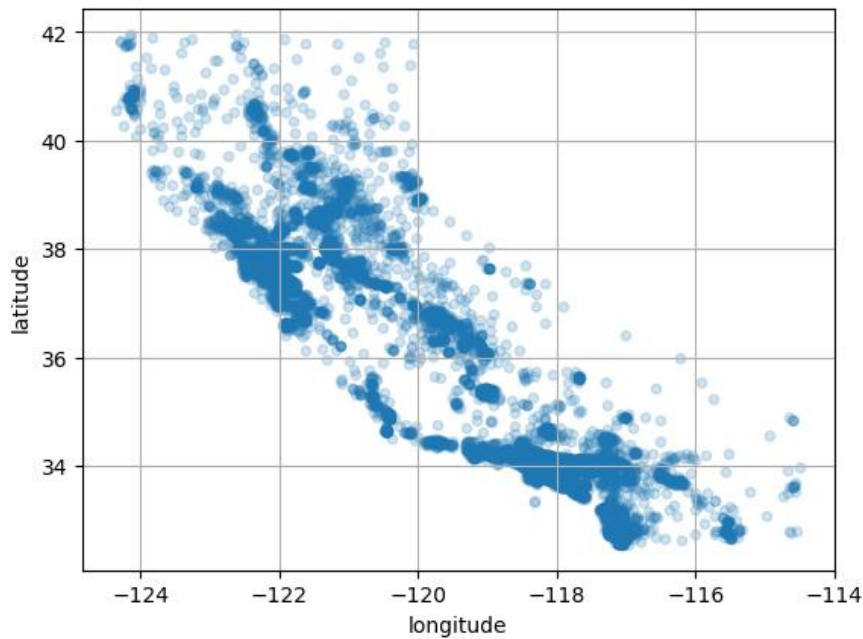
- **Distribution of Data:** Examining the distribution of data points to understand their range, central tendencies (mean, median), and dispersion (variance, standard deviation).
- **Graphical Representations:** Utilizing charts such as histograms, box plots, scatter plots, and bar charts to visualize relationships within the data and distributions of variables.
- **Outlier Detection:** Identifying unusual values that deviate from other data points. Outliers can influence statistical analyses and might indicate data entry errors or unique cases.
- **Correlation Analysis:** Checking the relationships between variables to understand how they might affect each other. This includes computing correlation coefficients and creating correlation matrices.
- **Handling Missing Values:** Detecting and deciding how to address missing data points, whether by imputation or removal, depending on their impact and the amount of missing data.
- **Summary Statistics:** Calculating key statistics that provide insight into data trends and nuances

Code for EDA using scatter plot of geography

```
housing1=train_set.copy()
housing1.plot(kind="scatter",x="longitude",y="latitude",grid=True)
plt.show()
```

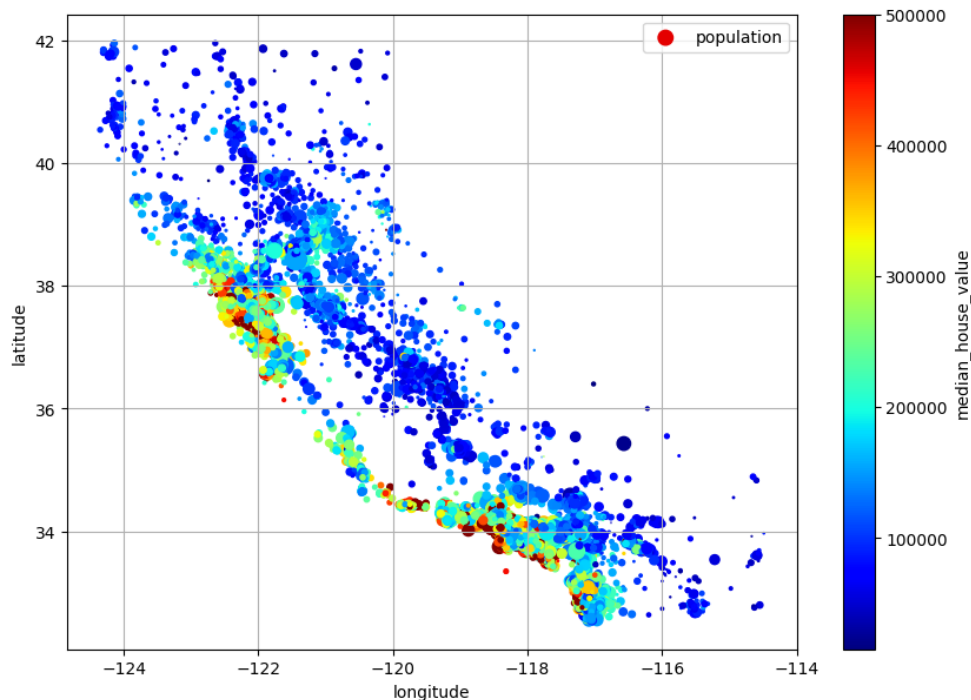


```
housing1.plot(kind="scatter",x="longitude",y="latitude",grid=True,alpha=0.2)
plt.show()
```



Population and expensive relationship

```
housing1.plot(kind="scatter",x="longitude",y="latitude",grid=True,
s=housing1["population"]/100,label="population",
c="median_house_value",cmap="jet",colorbar=True,figsize=(10,7))
plt.show()
```



#Studying correlation

Correlation is a key statistical concept that researchers employ to analyze connections within their data. It helps us to Understand the Relationship Between Variables. Knowing the correlation helps uncover important relationships between elements we are investigating. It provides insight into how changes in one variable may correlate with or predict changes in another. As researchers we rely on correlation to better understand the links between different phenomena.

The correlation coefficient quantifies the strength and direction of the correlation. Values closer to 1 or -1 represent stronger correlations, while those closer to 0 indicate little connection between the variables.

Code for correlation

```
housing2=housing1.drop(['ocean_proximity','income_cat'],axis=1)
corr_matrix=housing2.corr()
#Correlation with target attributes
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Output:

```
median_house_value    1.000000
median_income         0.688380
total_rooms           0.137455
housing_median_age    0.102175
households            0.071426
total_bedrooms        0.054635
population            -0.020153
longitude             -0.050859
latitude              -0.139584
```

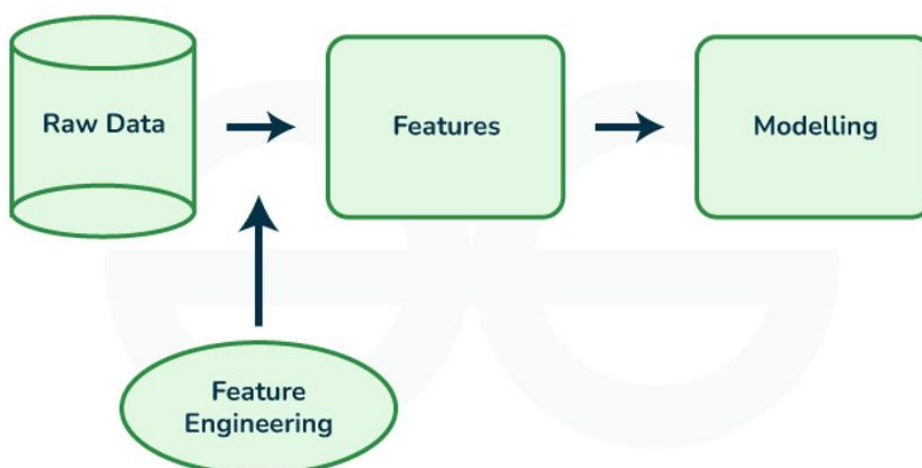
Name: median_house_value, dtype: float64

Hence target attribute median_house_value is highly correlated to median_income

Feature Engineering

Feature engineering is the process of transforming raw data into features that are suitable for machine learning models. In other words, it is the process of selecting, extracting, and transforming the most relevant features from the available data to build more accurate and efficient machine learning models.

The success of machine learning models heavily depends on the quality of the features used to train them. Feature engineering involves a set of techniques that enable us to create new features by combining or transforming the existing ones. These techniques help to highlight the most important patterns and relationships in the data, which in turn helps the machine learning model to learn from the data more effectively.



Code for Feature Engineering

```
housing2["rooms_per_house"]=housing2["total_rooms"]/housing2["households"]
housing2["bedrooms_ratio"]=housing2["total_bedrooms"]/housing2["total_rooms"]
housing2["people_per_house"]=housing2["population"]/housing2["households"]
```


Study their impact

```
corr_matrix=housing2.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Output:

```
median_house_value  1.000000
median_income      0.688380
rooms_per_house    0.143663
total_rooms        0.137455
housing_median_age  0.102175
households         0.071426
total_bedrooms     0.054635
population         -0.020153
people_per_house   -0.038224
longitude          -0.050859
latitude           -0.139584
bedrooms_ratio     -0.256397
```

Hence new attributes are much more correlated with target attribute than the older features.

Handling Missing data**#Old approaches****#Get rid of corresponding districts**

```
housing1.dropna(subset=["total_bedrooms"],inplace=True)
```

#Get rid of corresponding column

```
housing1.drop("total_bedrooms",axis=1)
```

#Imputation

```
median=housing1["total_bedrooms"].median()
housing1["total_bedrooms"].fillna(median,inplace=True)
```

New approach

```
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(strategy="median")
housing_num=housing1.select_dtypes(include=[np.number])
imputer.fit(housing_num)
X=imputer.transform(housing_num)
housing_tr=pd.DataFrame(X,
columns=housing_num.columns,index=housing_num.index)
housing_tr.info()
```