

MODULE I:

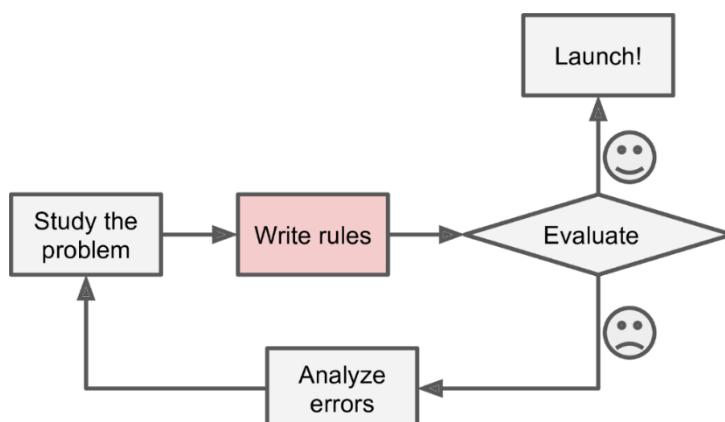
Definitions of Machine Learning

- 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.

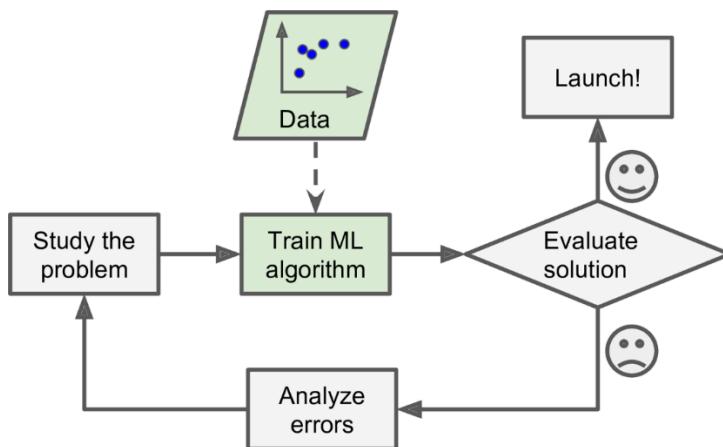
Need to use Machine Learning

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:



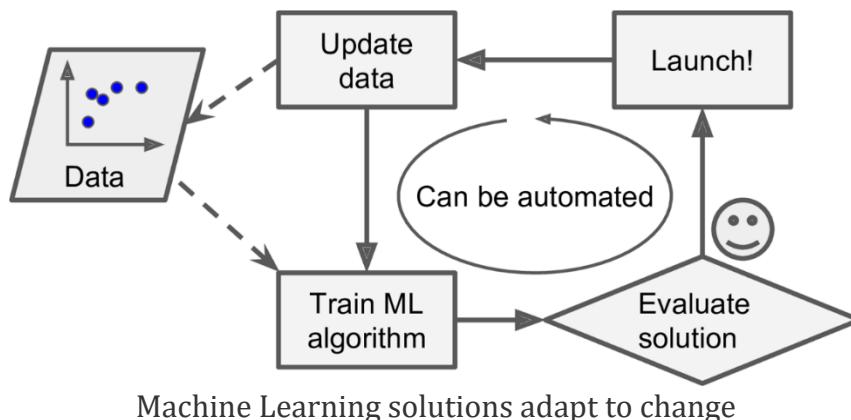
Traditional programming for spam filter detection



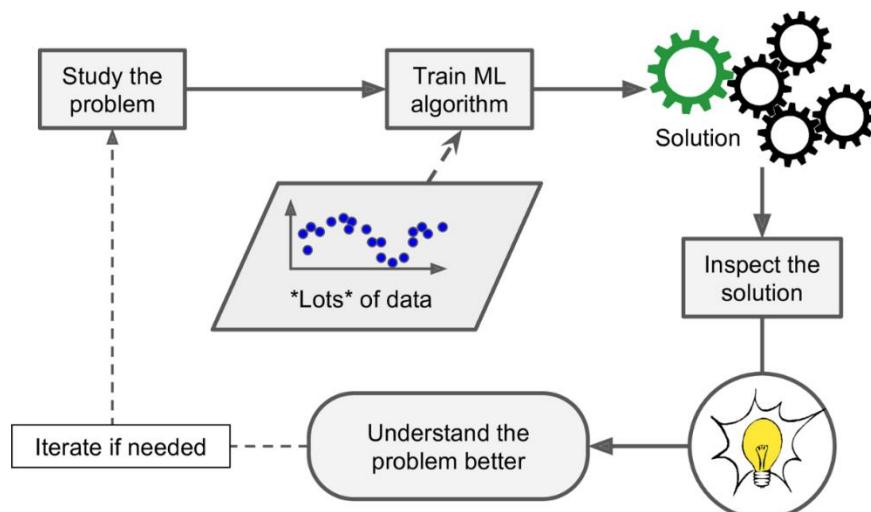
Machine Learning approach for spam filter detection

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 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.



- 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.

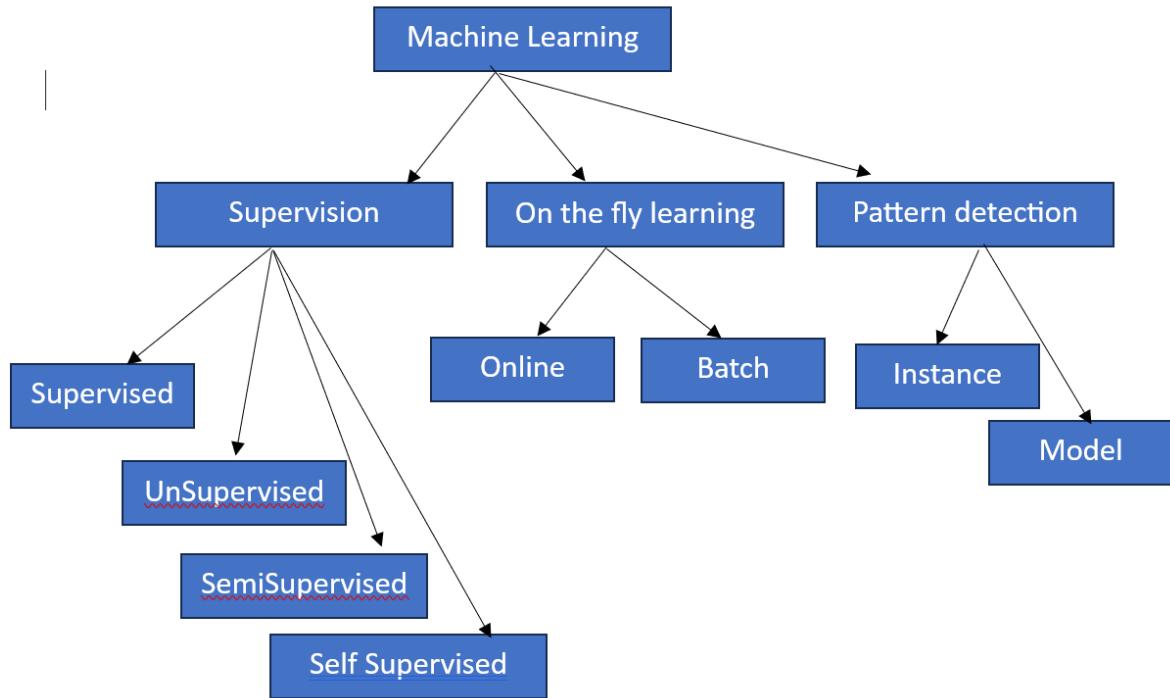


Types of Machine Learning Systems

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



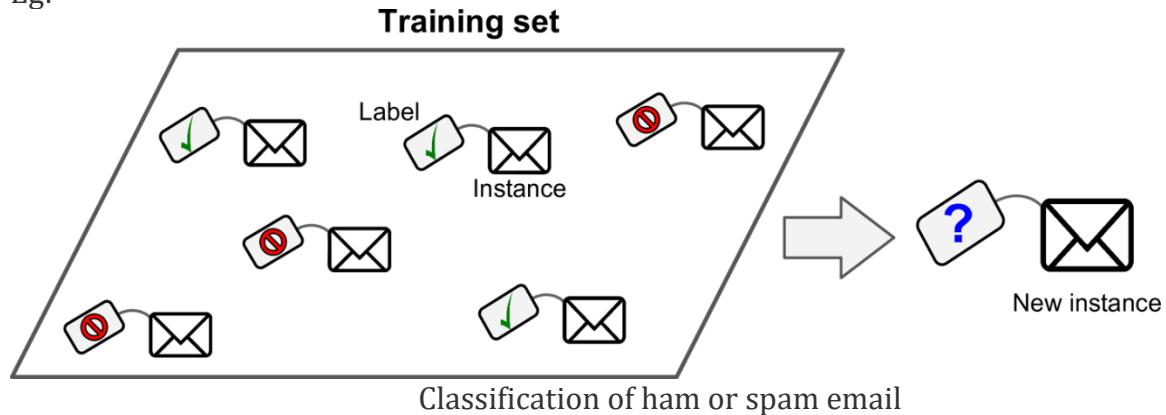
Supervised/Unsupervised Learning

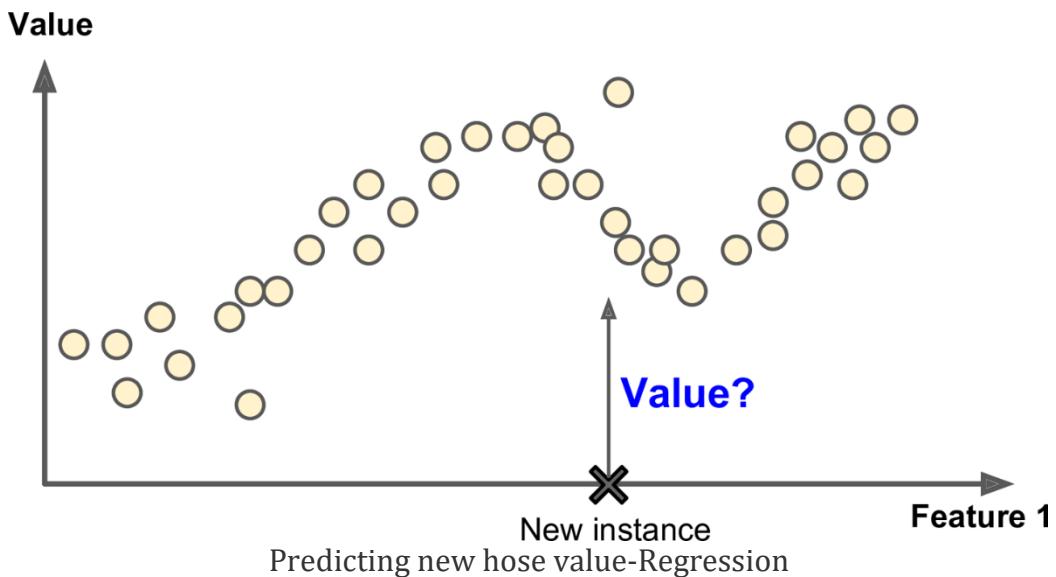
Machine Learning systems can be classified according to the amount and type of supervision they get during training. There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning

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:

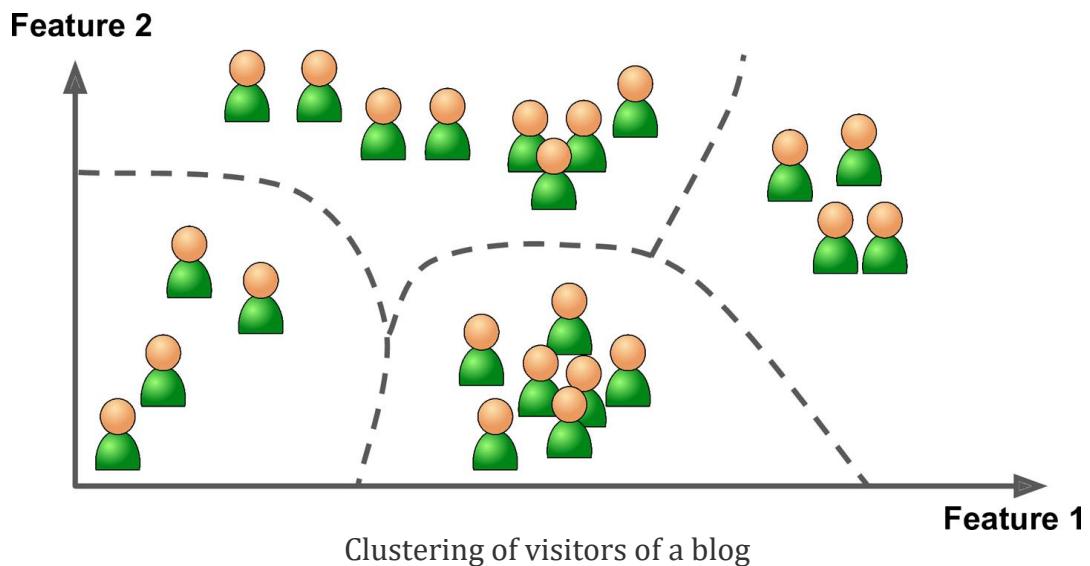




Unsupervised learning

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:



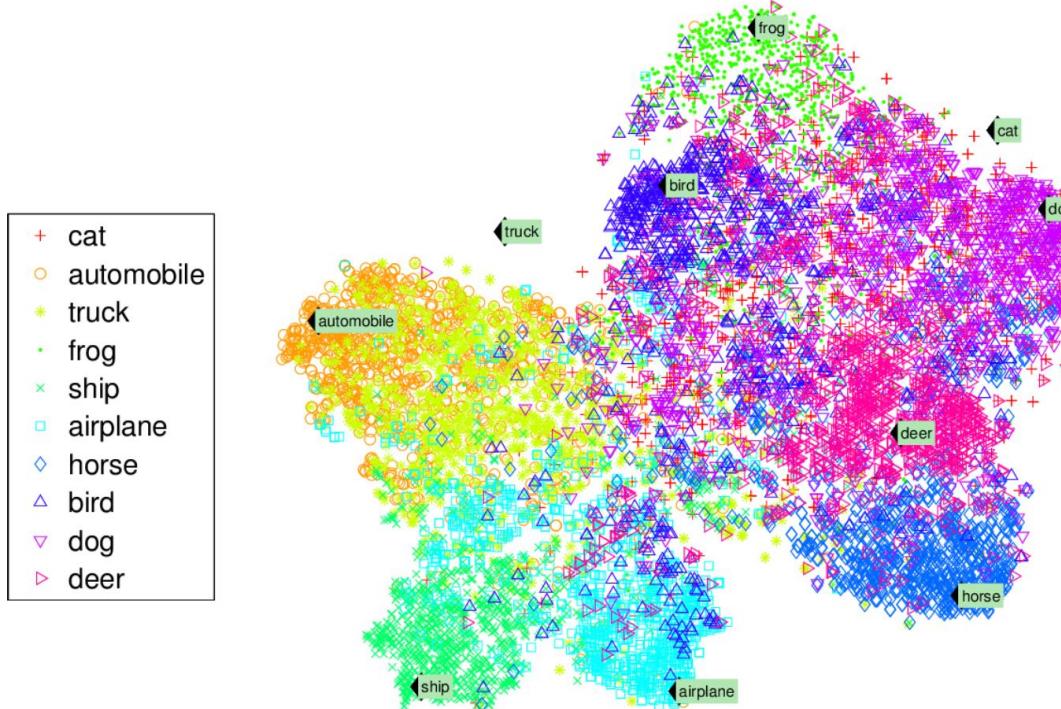
Use cases/Applications of unsupervised learning

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.

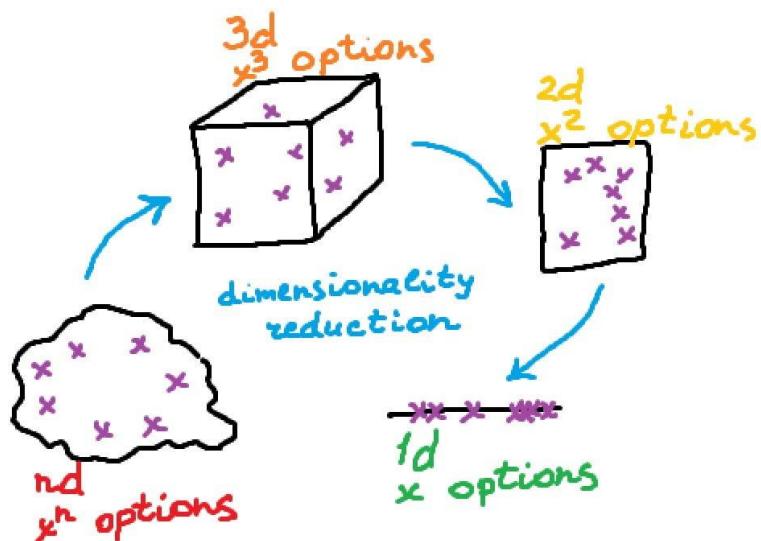
Eg: Visualization of various objects performing semantic clustering



(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.

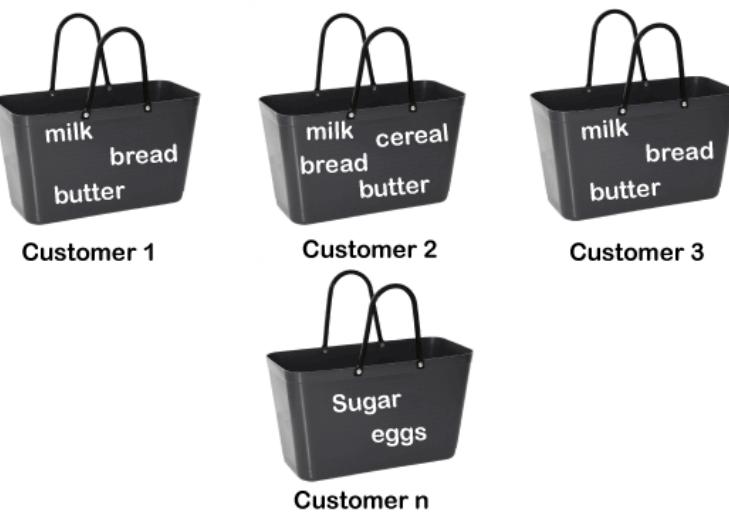
Eg:



(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..

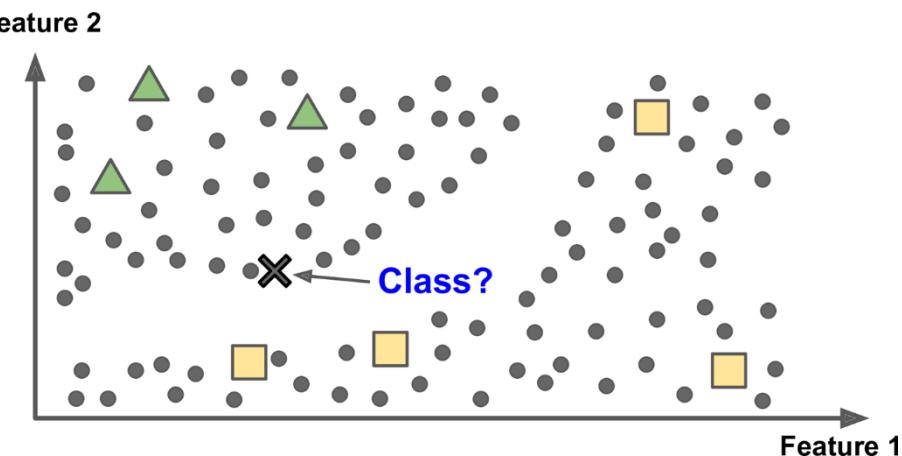
For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:



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

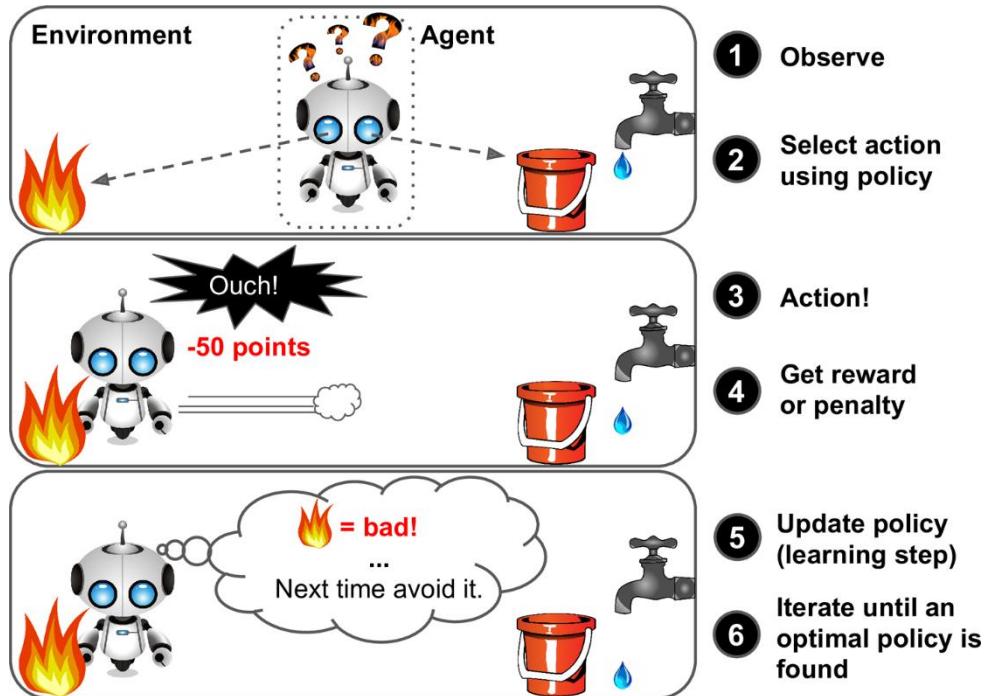
Self-supervised learning is a deep learning methodology where a model is pre-trained using unlabelled data and the data labels are generated automatically, which are further used in subsequent iterations as ground truths. The fundamental idea for self-supervised learning is to create supervisory signals 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.

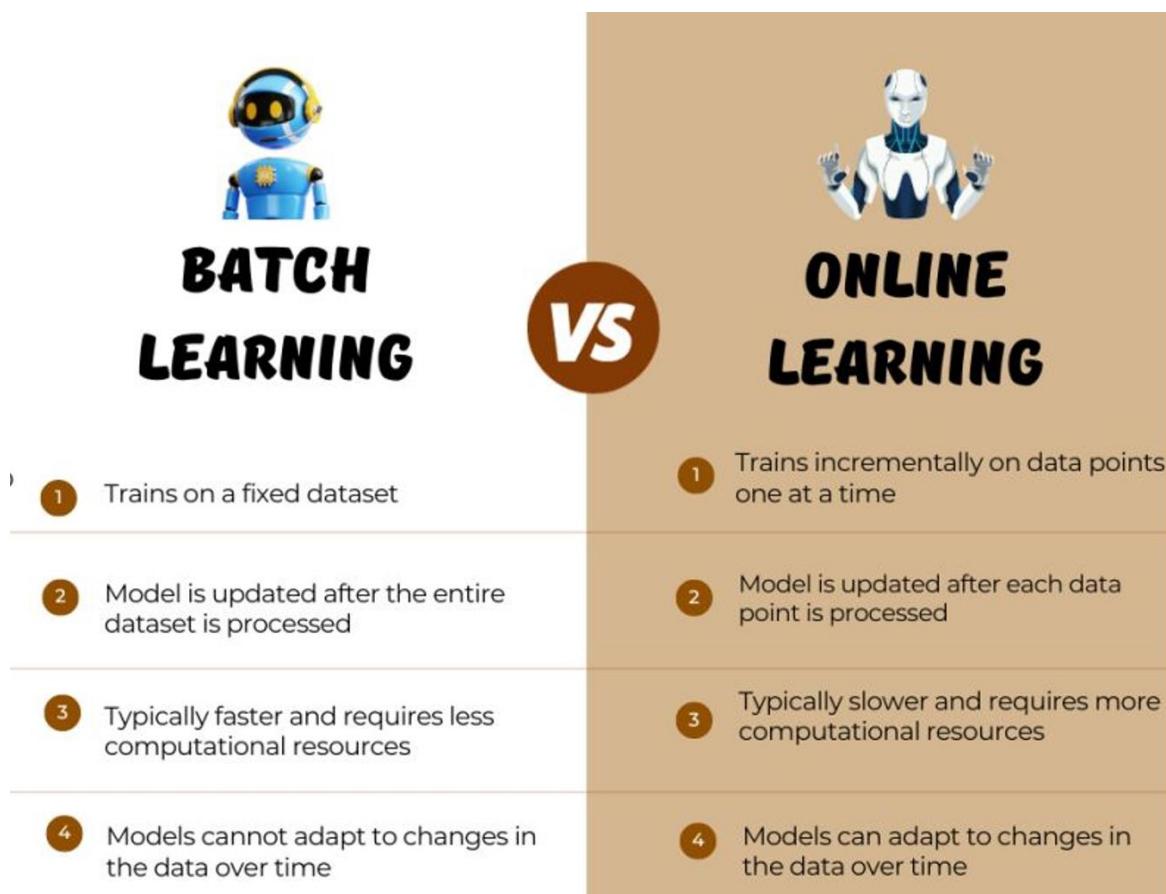
Reinforcement Learning

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.

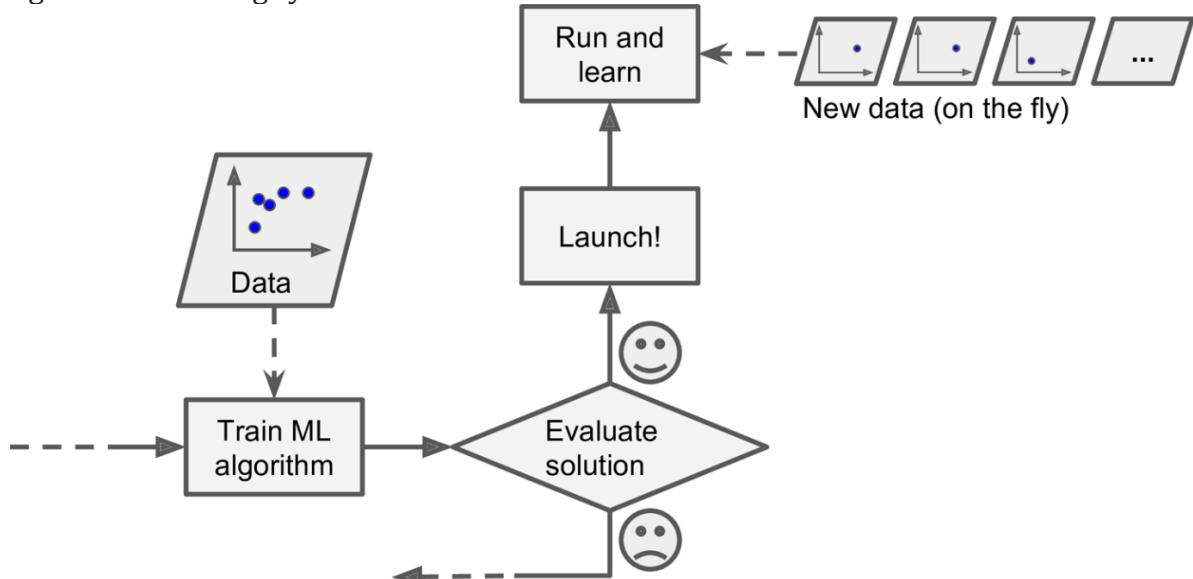
Hardware limitations: Batch learning can be computationally expensive, especially when dealing with complex models or large datasets. Training a model on a single machine may take a significant amount of time and may require high-performance hardware, such as GPUs or specialized processing units.

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

online learning

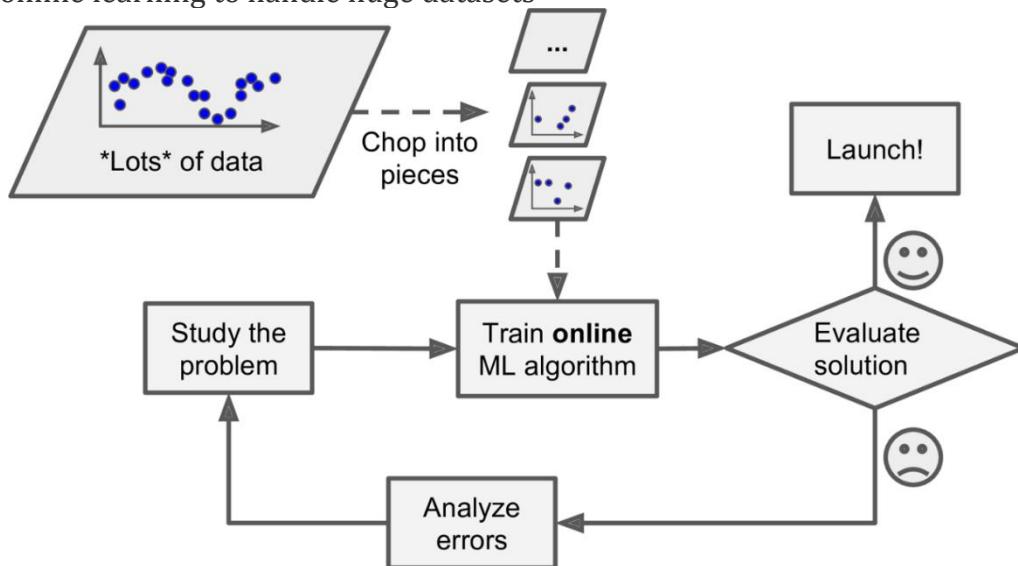
In the online learning, data is fed to the model in small batches, sequentially. These batches are called mini batches. After, each batch of training, your model gets better. since these batches are small chunks of data. so you can perform this training on server (in production) That's why it is called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

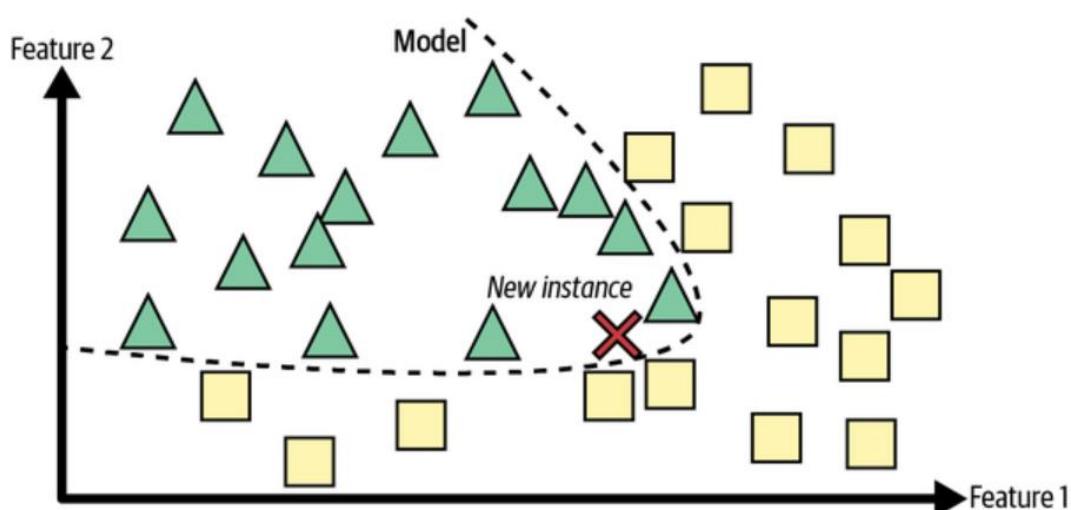
Instance-Based Versus Model-Based Learning

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.

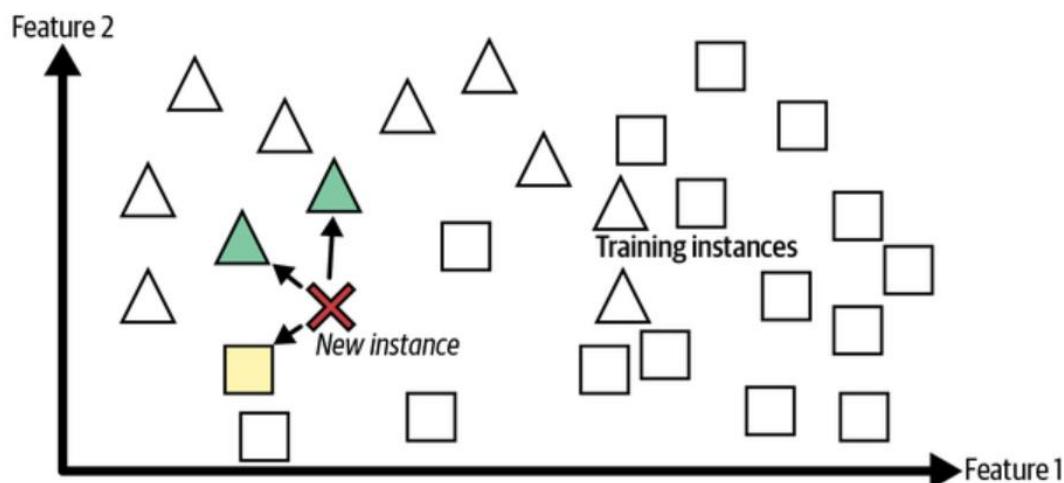
Disadvantages of Model-Based Learning

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.

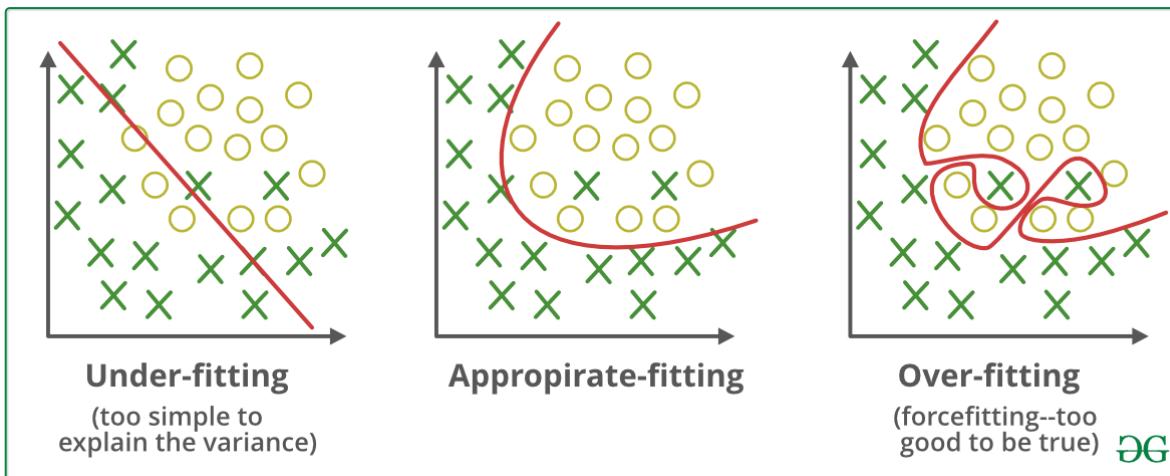
The main options to fix this problem are:

- 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:

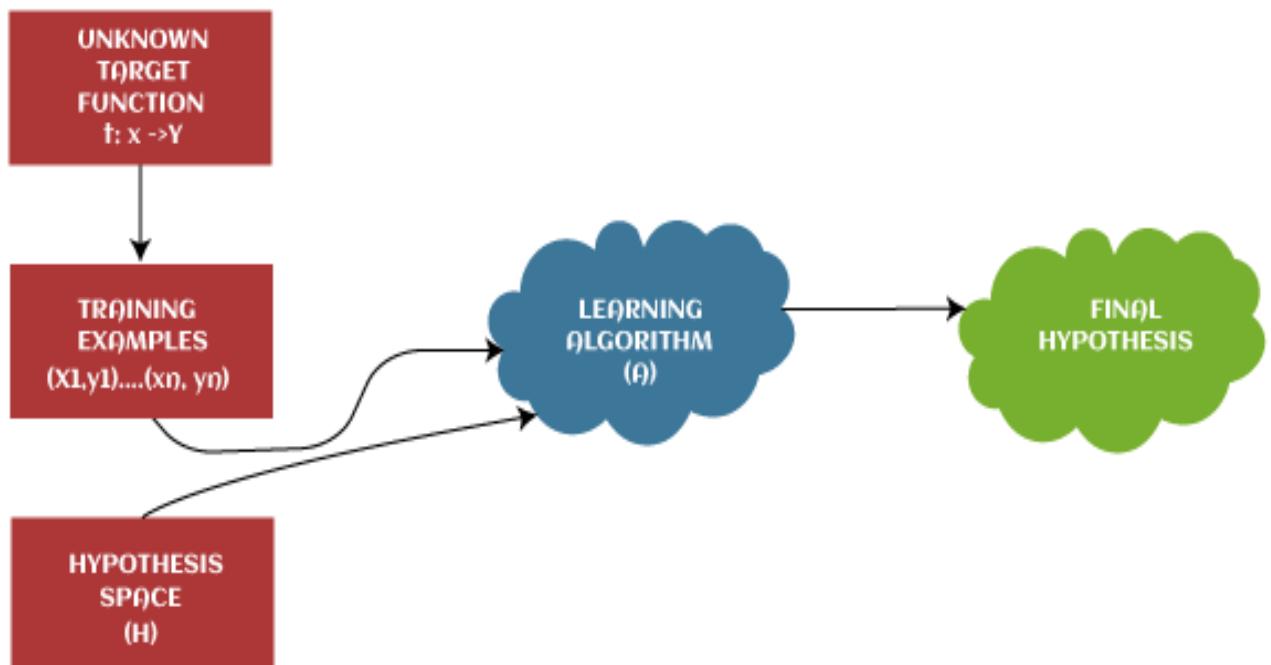
$$h(x) = c(x) \text{ for all } x \in X$$

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

A Robot Driving Learning Problem:

For a robot to drive on a four-lane highway it needs a human-like understanding of all the possibilities it might encounter. With the use of sight scanners and advanced machine learning algorithms, it can be made possible.

T -> To drive on public four-lane highways using sight scanners.

P -> the average distance progressed before an error.

E -> the order of images and steering instructions noted down while observing a human driver.

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.

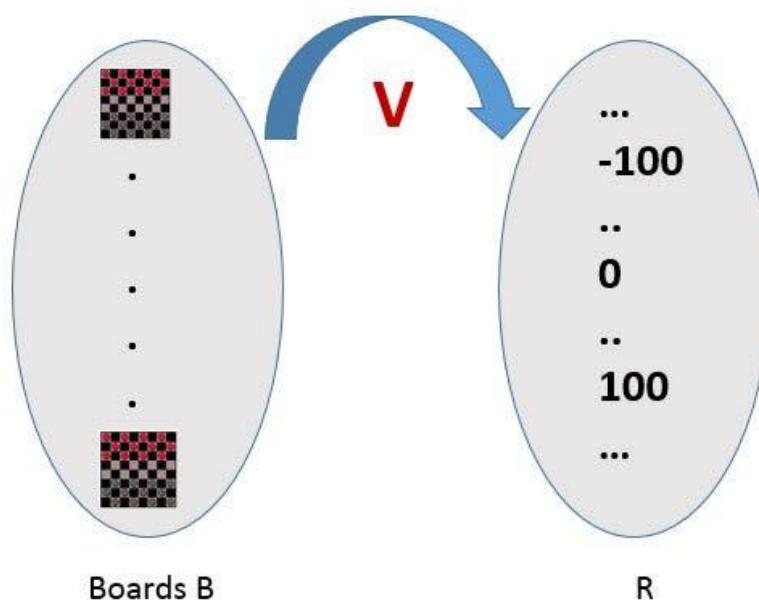
1. Direct or Indirect training experience — In the case of direct training experience, an individual board states and correct move for each board state are given. In case of indirect training experience, the move sequences for a game and the final result (win, loss or draw) are given for a number of 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 it's time to choose a representation that the learning program will use to describe the function \hat{V} that it will learn.

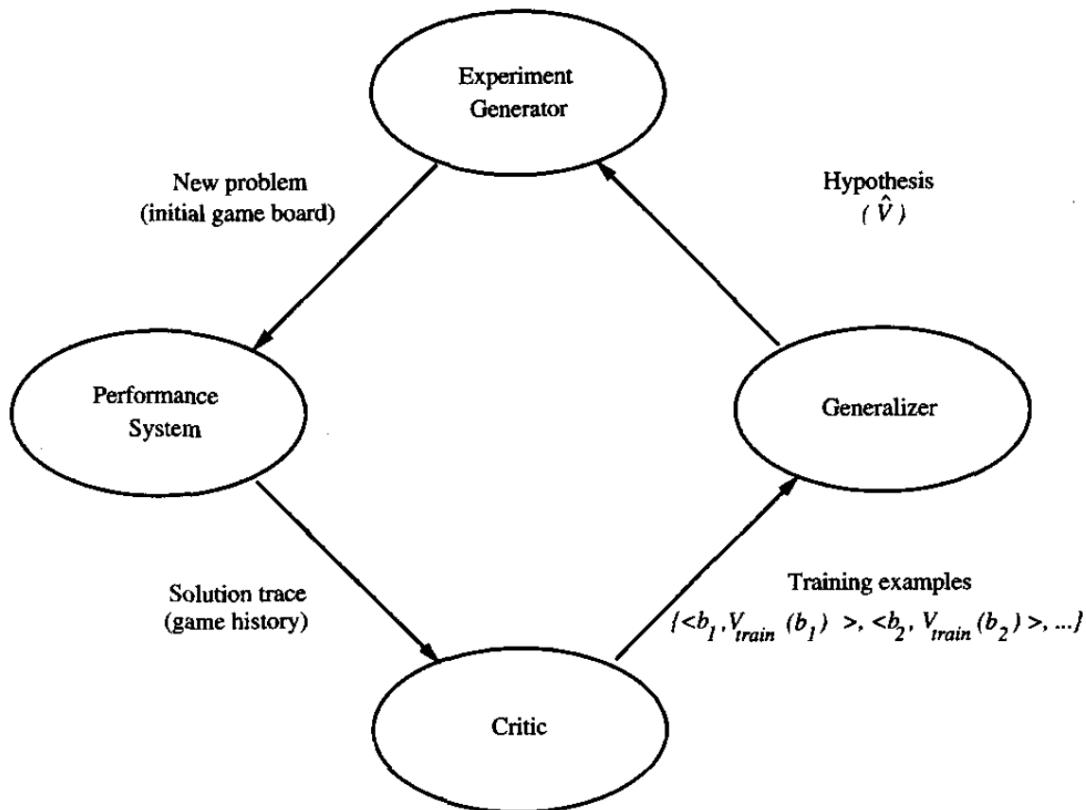
choose a simple representation for any given board state, the function \hat{V} will be calculated as a linear combination of the following board features:

- $x_1(b)$ — 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:

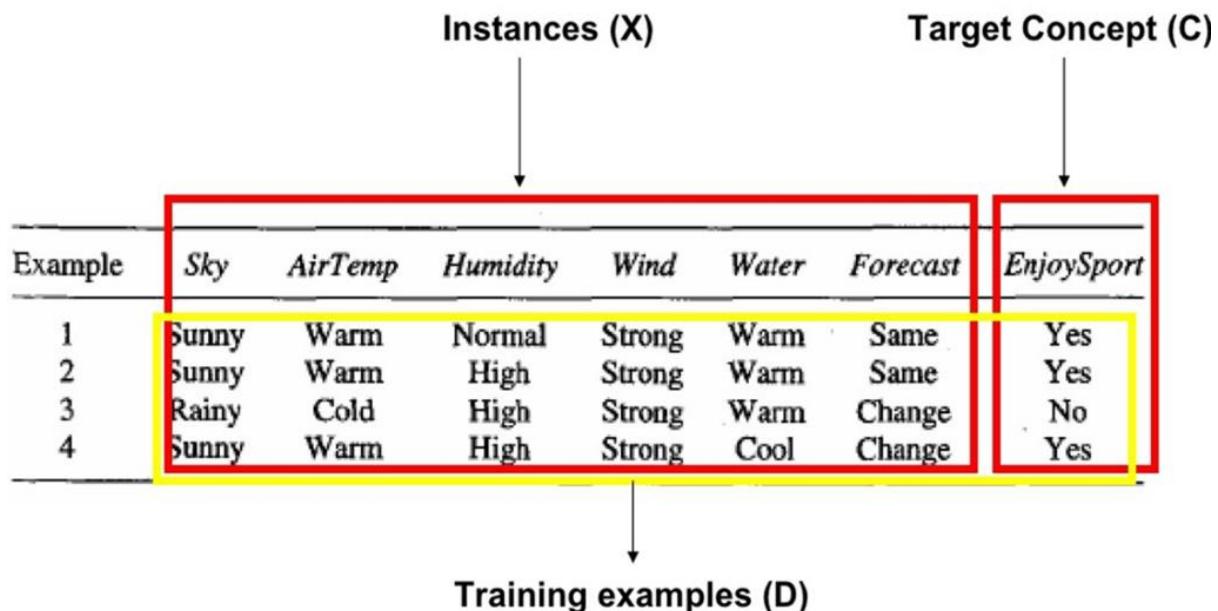
1. What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
2. How much training 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 “ø” 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

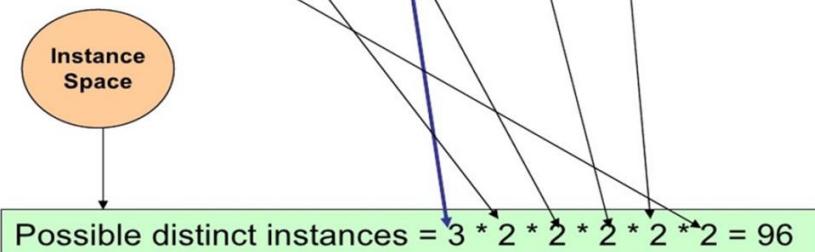
F2 \rightarrow X, Y

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

Suppose the attribute **Sky** has three possible values,
 and that **AirTemp**, **Humidity**, **Wind**, **Water**, and
Forecast each have two possible values



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 $1 + (4 \cdot 3 \cdot 3 \cdot 3 \cdot 3) = 973$.

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 $h = <\emptyset, \emptyset, \dots, \emptyset>$, 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

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

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** c 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)

$G_3 = [<\text{Sunny}, ?, ?, ?, ?, ?, ?>, <?, \text{Warm}, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>]$

$S_3 = [<\text{Sunny}, \text{Warm}, ?, \text{Strong}, \text{Warm}, \text{Same}>]$

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

$G_4 = [<\text{Sunny}, ?, ?, ?, ?, ?, ?>, <?, \text{Warm}, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>, <?, ?, ?, ?, ?, ?, ?>]$

$S_4 = [<\text{Sunny}, \text{Warm}, ?, \text{Strong}, ?, ?>]$

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 converge 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 store 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

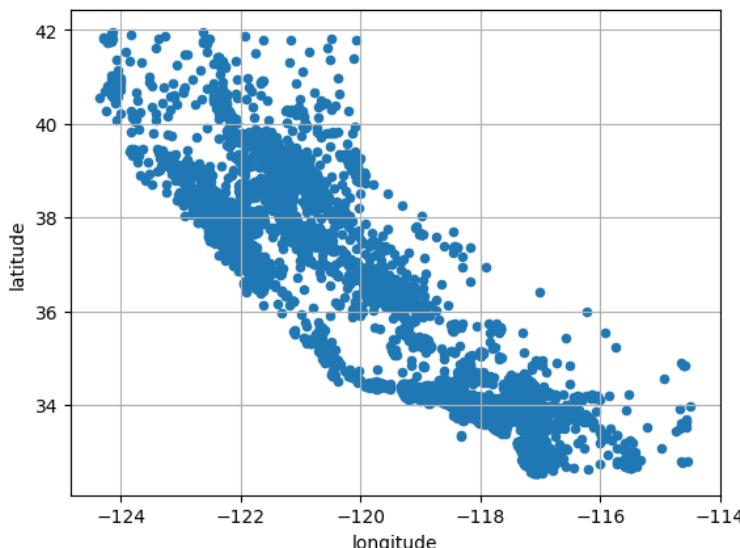
Exploratory data analysis is one of the basic and essential steps of a data science project. A data scientist involves almost 70% of his work in doing the EDA of the dataset. In this

article, we will discuss what is Exploratory Data Analysis (EDA) and the steps to perform EDA. Key aspects of EDA include:

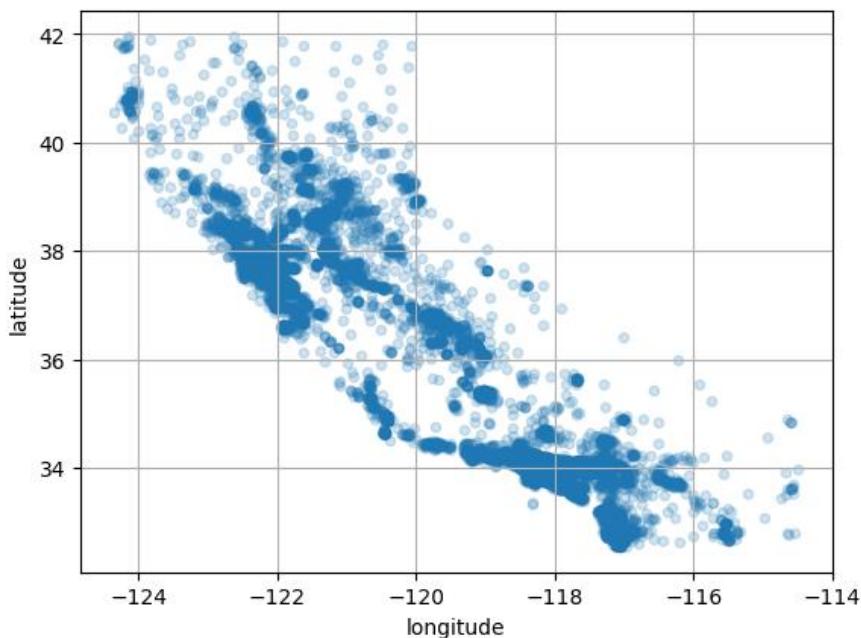
- Distribution of Data: Examining the distribution of data points to understand their range, central tendencies (mean, median), and dispersion (variance, standard deviation).
- Graphical Representations: Utilizing charts such as histograms, box plots, scatter plots, and bar charts to visualize relationships 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()
```

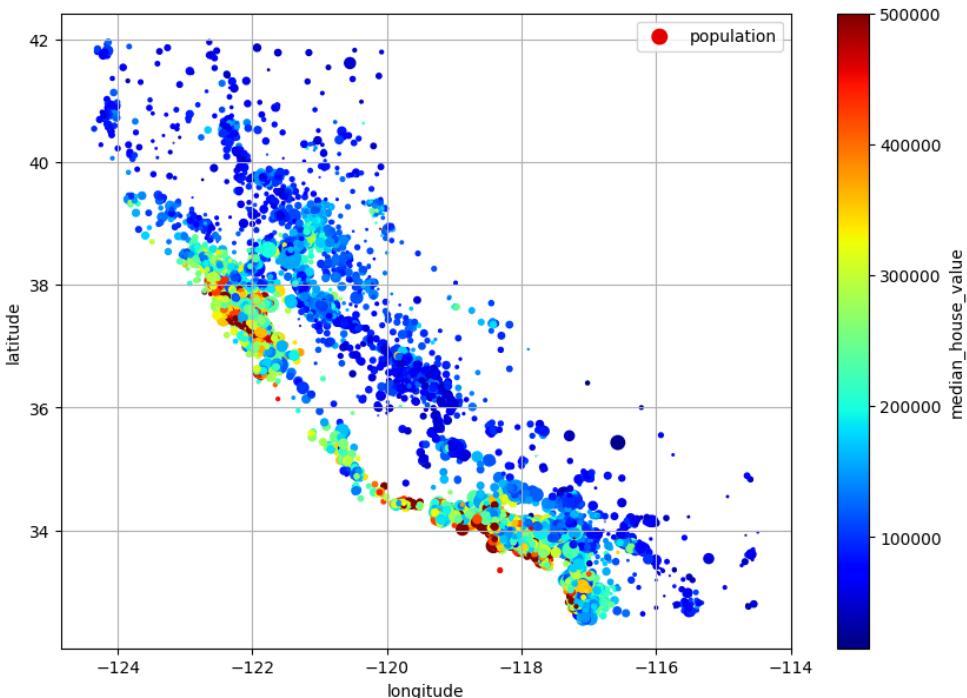


```
housing1.plot(kind="scatter",x="longitude",y="latitude",grid=True,alpha=0.2)
plt.show()
```



Population and expensive relationships

```
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
```

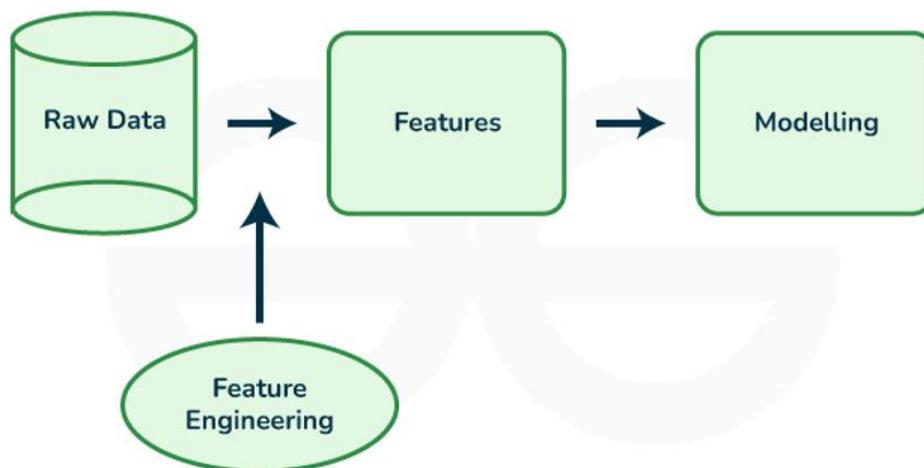
```
Name: median_house_value, dtype: float64
```

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()
```

Handling Text and Categorical data

Numerical data, as its name suggests, involves features that are only composed of numbers, such as integers or floating-point values. Categorical data are variables that contain label values rather than numeric values. The number of possible values is often limited to a fixed set. Categorical variables are often called nominal.

Some examples include:

- A “*pet*” variable with the values: “*dog*” and “*cat*”.
- A “*color*” variable with the values: “*red*”, “*green*”, and “*blue*”.
- A “*place*” variable with the values: “*first*”, “*second*”, and “*third*”.

A numerical variable can be converted to an ordinal variable by dividing the range of the numerical variable into bins and assigning values to each bin. For example, a numerical variable between 1 and 10 can be divided into an ordinal variable with 5 labels with an ordinal relationship: 1-2, 3-4, 5-6, 7-8, 9-10. This is called discretization.

- Nominal Variable (*Categorical*). Variable comprises a finite set of discrete values with no relationship between values.
- Ordinal Variable. Variable comprises a finite set of discrete values with a ranked ordering between values.

Some algorithms can work with categorical data directly. For example, a decision tree can be learned directly from categorical data with no data transform required (this depends on the specific implementation). Many machine learning algorithms cannot operate on label data directly. They require all input variables and output variables to be numeric.

Ordinal Encoding:

In ordinal encoding, each unique category value is assigned an integer value. This is called an ordinal encoding or an integer encoding and is easily reversible. Often, integer values starting at zero are used.

Eg. Python code to perform ordinal encoding on California housing dataset:

```
housing_cat=housing[['ocean_proximity']]
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder=OrdinalEncoder()
eh=ordinal_encoder.fit_transform(housing_cat)
print(eh)
print(ordinal_encoder.categories_)
```

One-Hot Encoding

For categorical variables where no ordinal relationship exists, the integer encoding may not be enough, at best, or misleading to the model at worst. Forcing an ordinal relationship via an ordinal encoding and allowing the model to assume a natural ordering between categories may result in poor performance or unexpected results (predictions halfway between categories).

In this case, a one-hot encoding can be applied to the ordinal representation. This is where the integer encoded variable is removed and one new binary variable is added for each unique integer value in the variable.

Eg. Python code to perform OneHot encoding on California housing dataset:

```
from sklearn.preprocessing import OneHotEncoder
ohe=OneHotEncoder()
oo=ohe.fit_transform(housing_cat)
print(oo)
```

Feature Scaling and Transformation

Oftentimes, we have datasets in which different columns have different units – like one column can be in kilograms, while another column can be in centimeters. Furthermore, we can have columns like income which can range from 20,000 to 100,000, and even more; while an age column which can range from 0 to 100(at the most). Thus, Income is about 1,000 times larger than age.

When we feed these features to the model as is, there is every chance that the income will influence the result more due to its larger value. But this doesn't necessarily mean it is more important as a predictor. So, to give importance to both Age, and Income, we need feature scaling.

MinMax Scaler

The MinMax scaler is one of the simplest scalers to understand. It just scales all the data between 0 and 1. The formula for calculating the scaled value is-

$$x_{\text{scaled}} = (x - x_{\text{min}}) / (x_{\text{max}} - x_{\text{min}})$$

Thus, a point to note is that it does so for every feature separately. Though (0, 1) is the default range, we can define our range of max and min values as well.

```
#Eg Python code for MinMax Scaling
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler(feature_range=(-1,1))
hnm=mms.fit_transform(housing_num)
```

Standard Scaler

Just like the MinMax Scaler, the Standard Scaler is another popular scaler that is very easy to understand and implement.

For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1(or the variance).

$$x_{\text{scaled}} = \frac{x - \text{mean}}{\text{std_dev}}$$

However, Standard Scaler assumes that the distribution of the variable is normal

```
#Eg Python code for MinMax Scaling
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
snm=ss.fit_transform(housing_num)
```

Custom Transformer

Consider this situation – Suppose you have your own Python function to transform the data. Sklearn also provides the ability to apply this transform to our dataset using what is called a FunctionTransformer. Let us take a simple example. I have a feature transformation techniques that involves taking (log to the base 2) of the values. In NumPy, there is a function called log2 which does that for us. Thus, we can now apply the FunctionTransformer:

```
from sklearn.preprocessing import FunctionTransformer
transformer = FunctionTransformer(np.log2, validate = True)
```

```
df_scaled[col_names] = transformer.transform(features.values)
df_scaled
```

Here is the output with log-base 2 applied on Age and Income:

	Income	Age	Department
0	13.872675	4.643856	HR
1	10.813781	4.169925	Legal
2	16.872675	5.392317	Marketing
3	13.287712	5.672425	Management

Transformation Pipelines

A machine learning pipeline is used to help automate machine learning workflows. They operate by enabling a sequence of data to be transformed and correlated together in a model that can be tested and evaluated to achieve an outcome, whether positive or negative.

Machine learning (ML) pipelines consist of several steps to train a model. Machine learning pipelines are iterative as every step is repeated to continuously improve the accuracy of the model and achieve a successful algorithm. To build better machine learning models, and get the most value from them, accessible, scalable and durable storage solutions are imperative, paving the way for on-premises object storage.

Need of ML pipelines:

1. The main objective of having a proper pipeline for any ML model is to exercise control over it. A well-organised pipeline makes the implementation more flexible.
2. The term ML model refers to the model that is created by the training process.
3. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer to be predicted), and it outputs an ML model that captures these patterns.
4. A model can have many dependencies and to store all the components to make sure all features available both offline and online for deployment, all the information is stored in a central repository.
5. A pipeline consists of a sequence of components which are a compilation of computations. Data is sent through these components and is manipulated with the help of computation.

Eg Python code for creating pipelines in ML – Numerical data
from sklearn.pipeline import Pipeline

```
nump=Pipeline([
    ("impute",SimpleImputer(strategy="median")),
    ("standardize",StandardScaler())
])
hnump=nump.fit_transform(housing_num)
hnump[:2].round(2)
```

Eg Python code for creating pipelines in ML- Categorical and Numerical data
from sklearn.compose import ColumnTransformer

```
from sklearn.pipeline import make_pipeline
na=["longitude","latitude","housing_median_age","total_rooms","total_bedrooms","population","households","median_income"]
ca=["ocean_proximity"]
catp=make_pipeline(
    SimpleImputer(strategy="most_frequent"),
    OneHotEncoder()
)
prep=ColumnTransformer([
    ("num",nump,na),
    ("cat",catp,ca)])
hp=prep.fit_transform(housing)
```

Select and Train a model

Linear Model:

```
from sklearn.linear_model import LinearRegression
lr=make_pipeline(prep,LinearRegression())
housing=strat_train_set.drop("median_house_value",axis=1)
housing_labels=strat_train_set["median_house_value"].copy()
lr.fit(housing,housing_labels)
hpred=lr.predict(housing)
```

Non-Linear Model:

```
from sklearn.metrics import mean_squared_error
lrmse=mean_squared_error(housing_labels,hpred,squared=False)
print(lrmse)
from sklearn.tree import DecisionTreeRegressor
treg=make_pipeline(prep,DecisionTreeRegressor())
treg.fit(housing,housing_labels)
hpred=treg.predict(housing)
from sklearn.metrics import mean_squared_error
```

```
lrmse=mean_squared_error(housing_labels,hpred,squared=False)
print(lrmse)
```

Cross Validation in Machine Learning

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance.

The main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model's generalization performance, i.e., its ability to perform well on new, unseen data.

Eg Python code to demonstrate cross validation

```
from sklearn.model_selection import cross_val_score
trmses=-
cross_val_score(treg,housing,housing_labels,scoring="neg_root_mean_squared_error",cv
=10)
from sklearn.ensemble import RandomForestRegressor
freg=make_pipeline(prep,RandomForestRegressor())
freg.fit(housing,housing_labels)
frmses=-
cross_val_score(freg,housing,housing_labels,scoring="neg_root_mean_squared_error",cv
=10)
```

Randomized and Grid Search for Hyperparameter optimization

Hyperparameters are the parameters that determine the behavior and performance of a machine-learning model. These parameters are not learned during training but are instead set prior to training. The process of finding the optimal values for these hyperparameters is known as hyperparameter optimization.

Grid search is a method for hyperparameter optimization that involves specifying a list of values for each hyperparameter that you want to optimize, and then training a model for each combination of these values. For example, if you want to optimize two hyperparameters, alpha and beta, with grid search, you would specify a list of values for alpha and a separate list of values for the beta. The grid search algorithm would then train a model using every combination of these values and evaluate the performance of each model. The optimal values for the hyperparameters are then chosen based on the performance of the models.

Eg Python code for GridSearch

```
from sklearn.model_selection import GridSearchCV
```

```
# Define the hyperparameters and their possible values
param_grid = {
    'alpha': [0.01, 0.1, 1.0, 10.0],
    'beta': [0.01, 0.1, 1.0, 10.0]
}
```

```
# Create a model
model = SomeModel()
```

```
# Use grid search to find the optimal hyperparameters
grid_search = GridSearchCV(model, param_grid)
grid_search.fit(X, y)
```

```
# Print the optimal values for the hyperparameters
print(grid_search.best_params_)
```

Randomized search is another method for hyperparameter optimization that can be more efficient than grid search in some cases. With randomized search, instead of specifying a list of values for each hyperparameter, you specify a distribution for each hyperparameter. The randomized search algorithm will then sample values for each hyperparameter from its corresponding distribution and train a model using the sampled values. This process is repeated a specified number of times, and the optimal values for the hyperparameters are chosen based on the performance of the models.

Eg Python code for Randomized Search

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
```

```
# Define the hyperparameters and their distributions
param_distributions = {
    'alpha': uniform(0.01, 10.0),
    'beta': uniform(0.01, 10.0)
}
```

```
# Create a model
model = SomeModel()
```

```
# Use randomized search to find the optimal hyperparameters
random_search = RandomizedSearchCV(model,
```

```
    param_distributions)
random_search.fit(X, y)
```

```
# Print the optimal values for the hyperparameters
print(random_search.best_params_)
```

Advantages of Randomized Search over Grid Search:

One advantage of RandomizedSearchCV over GridSearchCV is that RandomizedSearchCV can be more efficient if the search space is large since it only samples a subset of the possible combinations rather than evaluating them all. This can be especially useful if the model is computationally expensive to fit, or if the hyperparameters have continuous values rather than discrete ones.

Another advantage of RandomizedSearchCV is that it can be more robust to the risk of overfitting since it does not exhaustively search the entire search space. If the hyperparameter search space is very large and the model is relatively simple, it is possible that GridSearchCV could overfit to the training data by finding a set of hyperparameters that works well on the training set but not as well on unseen data. RandomizedSearchCV can help mitigate this risk by sampling randomly from the search space rather than evaluating every combination.

MODULE II:

Classification

The Classification algorithm is a Supervised Learning technique that is used to identify the category of new observations on the basis of training data. In Classification, a program learns from the given dataset or observations and then classifies new observation into a number of classes or groups. Such as, Yes or No, 0 or 1, Spam or Not Spam, cat or dog, etc. Classes can be called as targets/labels or categories.

Unlike regression, the output variable of Classification is a category, not a value, such as "Green or Blue", "fruit or animal", etc. Since the Classification algorithm is a Supervised learning technique, hence it takes labeled input data, which means it contains input with the corresponding output.

MNIST dataset

The MNIST database (Modified National Institute of Standards and Technology database) is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger NIST Special Database 3 (digits written by employees of the United States Census Bureau) and Special Database 1 (digits written by high school students) which contain monochrome images of handwritten digits. The digits have been size-normalized and centered in a fixed-size image. The original black and white (bilevel) images from NIST were size normalized to fit in a 20x20 pixel box while preserving their aspect ratio. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm. the images were centered in a 28x28 image by computing the center of mass of the pixels, and translating the image so as to position this point at the center of the 28x28 field.

Programming snippet to interact with MNIST dataset

```
from sklearn.datasets import fetch_openml
```

```

mnist=fetch_openml("mnist_784",as_frame=False )
#as_frame=False as we need to process image as numpy arrays

#extracting features and target
x,y=mnist.data,mnist.target

#split dataset into train and test
xtrain,xtest,ytrain,ytest=x[:60000],x[60000:],y[:60000],y[60000:]

#Code to display a digit
import matplotlib.pyplot as plt
def show_digit(img):
    imgdata=img.reshape(28,28)
    plt.imshow(imgdata,cmap="binary")

show_digit(x[1])

```

Training a Binary Classifier

In a binary classification task, the goal is to classify the input data into two mutually exclusive categories. The training data in such a situation is labeled in a binary format: true and false; positive and negative; 0 and 1; spam and not spam, etc. depending on the problem being tackled. For instance, we might want to detect whether a given image is a truck or a boat. Logistic Regression and Support Vector Machines algorithms are natively designed for binary classifications. However, other algorithms such as K-Nearest Neighbors and Decision Trees can also be used for binary classification.

Python code for a binary classifier for digits (5 or not-5)

```

ytrain5=(ytrain=='5')
ytest5=(ytest=='5')
from sklearn.linear_model import SGDClassifier
sg=SGDClassifier()
sg.fit(xtrain,ytrain5)
sg.predict([x[1]])

```

Performance Measures of Binary classifiers

Crossvalidation

Cross validation is a technique used in machine learning to evaluate the performance of a model on unseen data. It involves dividing the available data into multiple folds or subsets, using one of these folds as a validation set, and training the model on the remaining folds. This process is repeated multiple times, each time using a different fold as the validation set. Finally, the results from each validation step are averaged to produce a more robust estimate of the model's performance. The main purpose of cross validation is to prevent overfitting, which occurs when a model is trained too well on

the training data and performs poorly on new, unseen data. By evaluating the model on multiple validation sets, cross validation provides a more realistic estimate of the model's generalization performance, i.e., its ability to perform well on new, unseen data.

#Python code for cross-validation on digit binary classifier that computes accuracy of classification

```
from sklearn.model_selection import cross_val_score
cross_val_score(sg,xtrain,ytrain5,cv=3,scoring='accuracy')
```

#Python code for prediction with cross validation

```
from sklearn.model_selection import cross_val_predict
y_pred=cross_val_predict(sg,xtrain,ytrain5,cv=3)
```

Confusion Matrix

A confusion matrix is a matrix that summarizes the performance of a machine learning model on a set of test data. It is a means of displaying the number of accurate and inaccurate instances based on the model's predictions. It is often used to measure the performance of classification models, which aim to predict a categorical label for each input instance.

The matrix displays the number of instances produced by the model on the test data.

- True positives (TP): occur when the model accurately predicts a positive data point.
- True negatives (TN): occur when the model accurately predicts a negative data point.
- False positives (FP): occur when the model predicts a positive data point incorrectly.
- False negatives (FN): occur when the model mispredicts a negative data point.

Table for confusion matrix

Actual	Predicted	
	Non-5	5
Non-5	TN	FP
5	FN	TP

#Python code to generate Confusion matrix

```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(ytrain5,y_pred)
print(cm)
```

Precision

Precision is defined as the ratio of correctly classified positive samples (True Positive) to a total number of classified positive samples (either correctly or incorrectly).

$$\text{Precision} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Positive}}$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

precision helps us to visualize the reliability of the machine learning model in classifying the model as positive

Recall

The recall is calculated as the ratio between the numbers of Positive samples correctly classified as Positive to the total number of Positive samples. The recall measures the model's ability to detect positive samples. The higher the recall, the more positive samples detected.

$$\text{Recall} = \frac{\text{True Positive}}{\text{True Positive} + \text{False Negative}}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Unlike Precision, Recall is independent of the number of negative sample classifications. Further, if the model classifies all positive samples as positive, then Recall will be 1.

#Python code for precision and recall

```
from sklearn.metrics import precision_score,recall_score,f1_score
precision_score(ytrain5,ypred)
recall_score(ytrain5,ypred)
```

Difference between Precision and Recall in Machine Learning

Precision	Recall
It helps us to measure the ability to classify positive samples in the model.	It helps us to measure how many positive samples were correctly classified by the ML model.
While calculating the Precision of a model, we should consider both Positive as well as Negative samples that are classified.	While calculating the Recall of a model, we only need all positive samples while all negative samples will be neglected.
When a model classifies most of the positive samples correctly as well as many false-positive samples, then the model is said to be a high recall and low precision model.	When a model classifies a sample as Positive, but it can only classify a few positive samples, then the model is said to be high accuracy, high precision, and low recall model.
The precision of a machine learning model is dependent on both the negative and positive samples.	Recall of a machine learning model is dependent on positive samples and independent of negative samples.
In Precision, we should consider all positive samples that are classified as positive either correctly or incorrectly.	The recall cares about correctly classifying all positive samples. It does

not consider if any negative sample is classified as positive.

F1-score

Precision and recall offer a trade-off, i.e., one metric comes at the cost of another. More precision involves a harsher critic (classifier) that doubts even the actual positive samples from the dataset, thus reducing the recall score. On the other hand, more recall entails a lax critic that allows any sample that resembles a positive class to pass, which makes border-case negative samples classified as “positive,” thus reducing the precision. Ideally, we want to maximize both precision and recall metrics to obtain the perfect classifier.

The F1 score combines precision and recall using their harmonic mean, and maximizing the F1 score implies simultaneously maximizing both precision and recall. Thus, the F1 score has become the choice of researchers for evaluating their models in conjunction with accuracy.

The F1 score is calculated as the harmonic mean of the precision and recall scores, as shown below. It ranges from 0-100%, and a higher F1 score denotes a better quality classifier.

$$\begin{aligned} \text{F1 Score} &= \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}} \\ &= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned}$$

#API for f1-score

```
f1_score(ytrain5,ypred)
```

Precision-recall trade-off

The classification threshold is an important parameter when building and evaluating classification models. It can significantly impact the model's performance and the decisions made based on its predictions.

A typical default choice is to use a threshold of 0.5.

In the spam example, that would mean that any email with a predicted probability greater than 0.5 is classified as spam and put in a spam folder. Any email with a predicted probability of less than or equal to 0.5 is classified as legitimate.

Each metric has its limitations. Precision prioritizes “correctness” but may not account for the cost of missing positive cases. Recall emphasizes “completeness” but may result in falsely flagging some instances. Both types of errors can be expensive, depending on the specific use case. Since precision and recall measure different aspects of the model quality, this leads to the precision-recall trade-off. You must balance their importance and account for it when training and evaluating ML models.

To balance precision and recall, you should consider the costs of false positives and false negatives errors. This is highly custom and depends on the business context. You might make different choices when solving the same problem in different companies.

Optimize for recall

Say your task is to score the customers likely to buy a particular product. You then pass this list of high-potential customers to a call center team to contact them. You might have

thousands of customers registering on your website every week, and the call center cannot reach all of them. But they can easily reach a couple of hundred.

Every customer that buys the product will make an effort well worth it. In this scenario, the cost of false positives is low (just a quick call that does not result in a purchase), but the value of true positives is high (immediate revenue).

In this case, you'd likely optimize for recall. You want to make sure you reach all potential buyers. Your only limit is the number of people your call center can contact weekly. In this case, you can set a lower decision threshold. Your model might have low precision, but this is not a big deal as long as you reach your business goals and make a certain number of sales.

Optimize for precision

Let's say you are working for a food delivery company. Your team is developing a machine learning model to predict which orders might be delivered in under 20 minutes based on factors such as order size, restaurant location, time of day, and delivery distance.

You will use this prediction to display a "fast delivery" label next to a potential order.

In this case, optimizing for precision makes sense. False positives (orders predicted to be completed fast but actually delayed) can result in a loss of customer trust and ultimately lead to decreased sales. On the other hand, false negatives (orders predicted to take longer but completed in under 20 minutes) will likely have no consequences at all, as the customer would simply be pleasantly surprised by the fast delivery. Optimizing for precision typically means setting a higher classification threshold.

Balance precision and recall

consider a scenario where you are developing a model to detect fraudulent transactions in a banking system. In this case, the cost of false positives is high, as it can lead to blocking legitimate transactions and causing inconvenience to the customers. On the other hand, the cost of false negatives is also significant, as it can result in financial loss and lost trust due to fraudulent transactions.

In this case, you need to strike a balance between precision and recall. While you need to detect as many fraudulent transactions as possible (high recall), you must also ensure that you don't flag legitimate transactions as fraudulent too often (high precision). The threshold for detecting a fraudulent transaction must be set carefully, considering the costs associated with both types of errors.

Since the fraud cost can differ, you can also set different thresholds based on the transaction amounts. For example, you can set a lower decision threshold for high-volume transactions since they come with a higher potential financial loss. For smaller amounts (which are also more frequent), you can set the threshold higher to ensure you do not inconvenience customers too much.

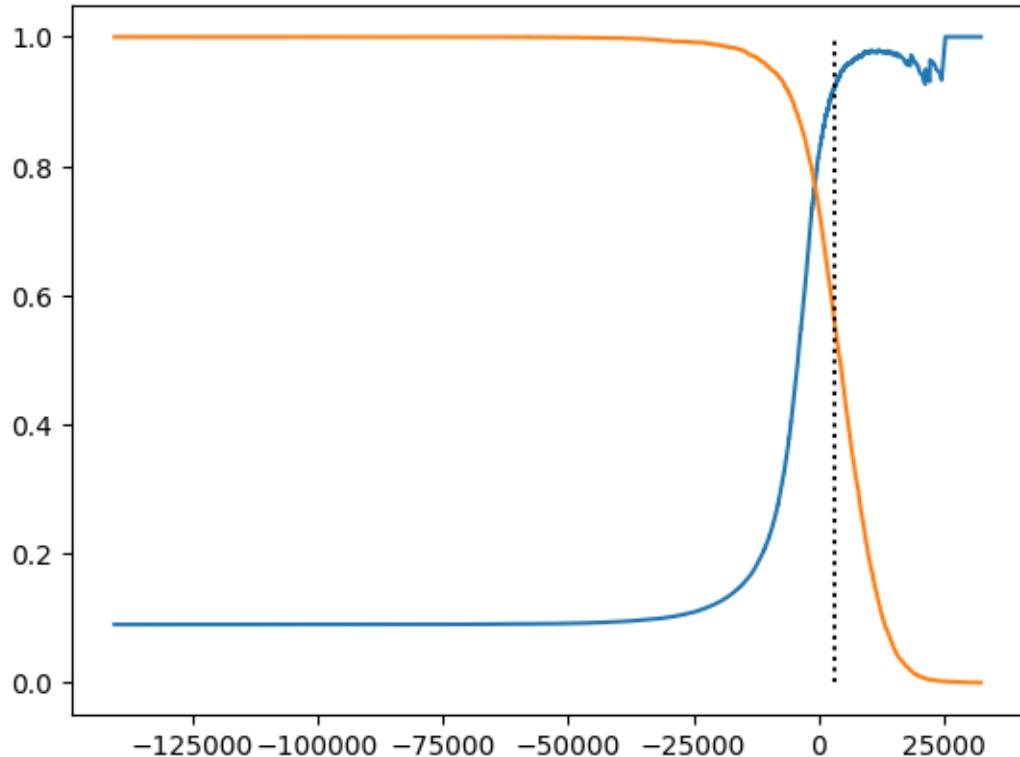
Precision-recall curve

One approach is the precision-recall curve. It shows the value pairs between precision and recall at different thresholds.

#Code for precision-recall curve

```
yscores=cross_val_predict(sg,xtrain,ytrain5,cv=3,
                           method='decision_function')
from sklearn.metrics import precision_recall_curve
```

```
p,r,t=precision_recall_curve(ytrain5,yscores)
plt.plot(t,p[:-1],label="T vs P")
plt.plot(t,r[:-1],label="T vs R")
plt.vlines(3000,0,1.0,"k","dotted",label="threshold line")
plt.show()
Output:
```



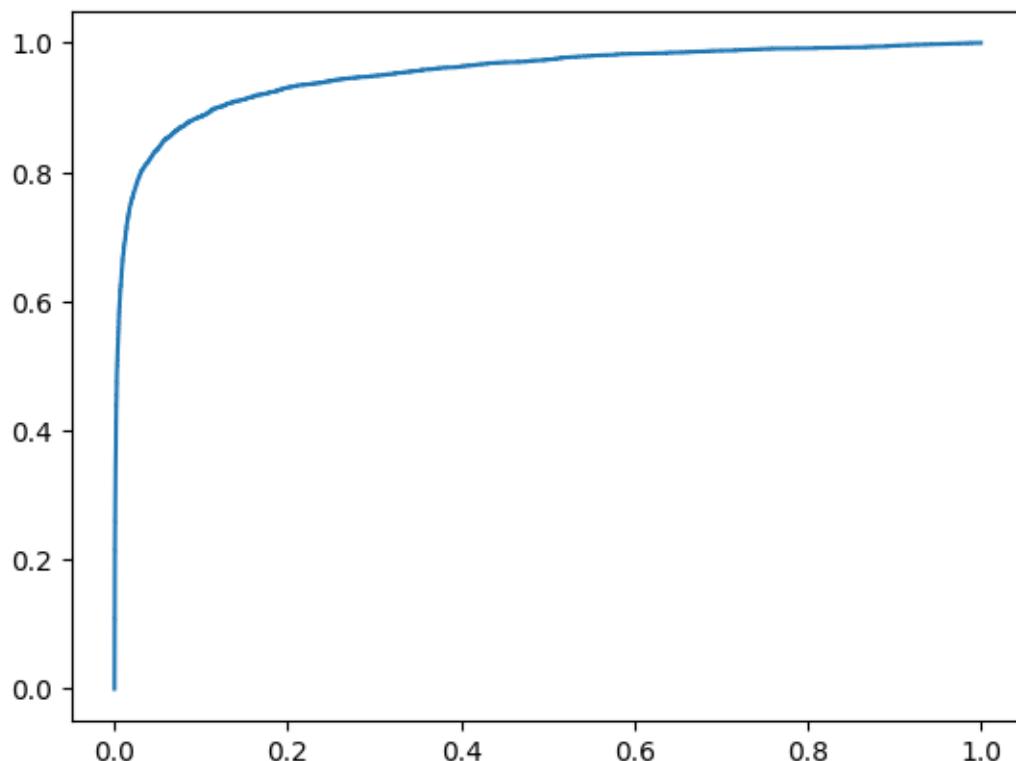
An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

#Python code for ROC curve

```
from sklearn.metrics import roc_curve
fpr,tpr,t=roc_curve(ytrain5,yscores)
plt.plot(fpr,tpr,label="FPR vs TPR")
plt.show()
Output:
```



AUC: Area Under the ROC Curve

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve (think integral calculus) from (0,0) to (1,1).

AUC provides an aggregate measure of performance across all possible classification thresholds. One way of interpreting AUC is as the probability that the model ranks a random positive example more highly than a random negative example. AUC is desirable for the following two reasons:

- AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.
- AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

#Python code for AUC

```
from sklearn.metrics import roc_auc_score
a=roc_auc_score(ytrain5,yscores)
print(a)
```

Multiclass classification

Binary classification are those tasks where examples are assigned exactly one of two classes. Multi-class classification is those tasks where examples are assigned exactly one of more than two classes.

- Binary Classification: Classification tasks with two classes.
- Multi-class Classification: Classification tasks with more than two classes.

Some algorithms are designed for binary classification problems. Examples include:

- Logistic Regression
- Perceptron

- Support Vector Machines

As such, they cannot be used for multi-class classification tasks, at least not directly. Instead, heuristic methods can be used to split a multi-class classification problem into multiple binary classification datasets and train a binary classification model each. Two examples of these heuristic methods include:

- One-vs-Rest (OvR)
- One-vs-One (OvO)

One-Vs-Rest for Multi-Class Classification

One-vs-rest (OvR for short, also referred to as One-vs-All or OvA) is a heuristic method for using binary classification algorithms for multi-class classification.

It involves splitting the multi-class dataset into multiple binary classification problems. A binary classifier is then trained on each binary classification problem and predictions are made using the model that is the most confident.

For example, given a multi-class classification problem with examples for each class ‘red,’ ‘blue,’ and ‘green’. This could be divided into three binary classification datasets as follows:

- Binary Classification Problem 1: red vs [blue, green]
- Binary Classification Problem 2: blue vs [red, green]
- Binary Classification Problem 3: green vs [red, blue]

A possible downside of this approach is that it requires one model to be created for each class. For example, three classes requires three models. This could be an issue for large datasets (e.g. millions of rows), slow models (e.g. neural networks), or very large numbers of classes (e.g. hundreds of classes).

#Python code for One-Vs-Rest

```
from sklearn.multiclass import OneVsRestClassifier
oc=OneVsRestClassifier(SVC())
oc.fit(xtrain[:2000],ytrain[:2000])
oc.predict([x[0]])
oc.decision_function([x[0]]).round(2)
```

One-Vs-One for Multi-Class Classification

One-vs-one (OvO for short) is another heuristic method for using binary classification algorithms for multi-class classification.

Like one-vs-rest, one-vs-one splits a multi-class classification dataset into binary classification problems. Unlike one-vs-rest that splits it into one binary dataset for each class, the one-vs-one approach splits the dataset into one dataset for each class versus every other class.

For example, consider a multi-class classification problem with four classes: ‘red,’ ‘blue,’ and ‘green,’ ‘yellow.’ This could be divided into six binary classification datasets as follows:

- Binary Classification Problem 1: red vs. blue
- Binary Classification Problem 2: red vs. green
- Binary Classification Problem 3: red vs. yellow
- Binary Classification Problem 4: blue vs. green
- Binary Classification Problem 5: blue vs. yellow
- Binary Classification Problem 6: green vs. yellow

The formula for calculating the number of binary datasets, and in turn, models, is as follows:

- $(\text{NumClasses} * (\text{NumClasses} - 1)) / 2$

Classically, this approach is suggested for support vector machines (SVM) and related kernel-based algorithms. This is believed because the performance of kernel methods does not scale in proportion to the size of the training dataset and using subsets of the training data may counter this effect.

#Python code for One-Vs-One approach

```
from sklearn.svm import SVC
sc=SVC()
sc.fit(xtrain[:2000],ytrain[:2000])
sc.predict([x[0]])
sds=sc.decision_function([x[0]])
sds.round(2)
classid=sds.argmax()
classid
```

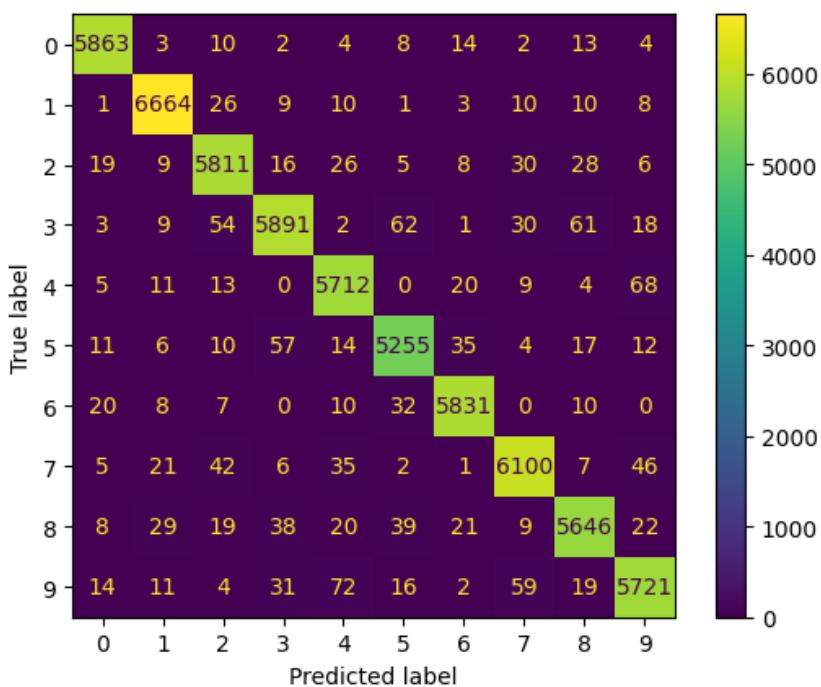
Error Analysis

Error analysis is the process to isolate, observe and diagnose erroneous ML predictions thereby helping understand pockets of high and low performance of the model. When it is said that “the model accuracy is 90%” it might not be uniform across subgroups of data and there might be some input conditions which the model fails more.

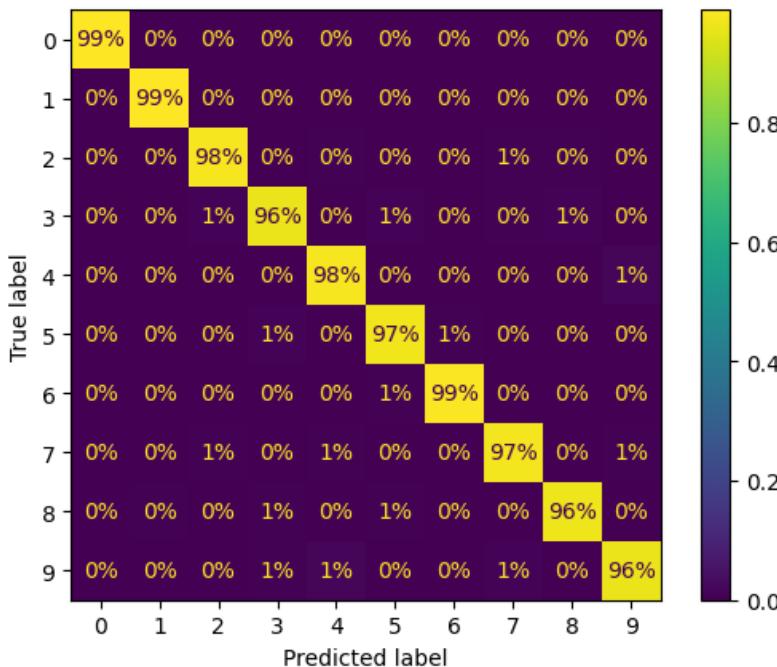
#Python code to display number of correct and wrong digit classifications

```
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import ConfusionMatrixDisplay
ypred=cross_val_predict(sc,xtrain,ytrain,cv=3)
ConfusionMatrixDisplay.from_predictions(ytrain,ypred)
```

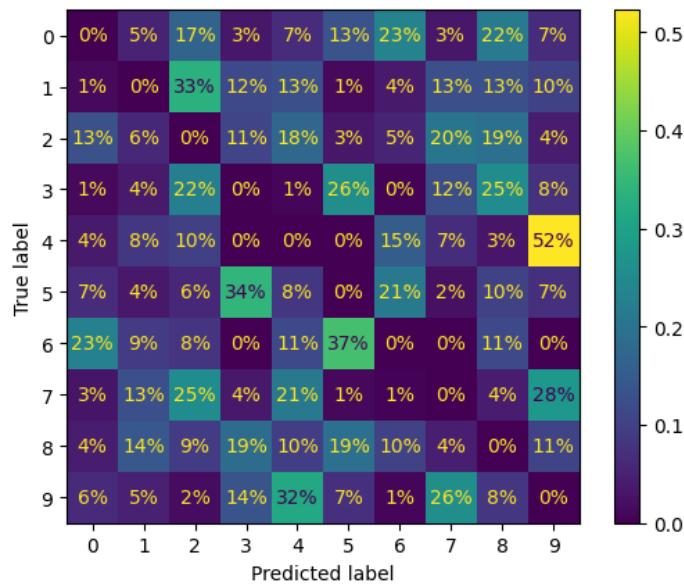
Output:

**#Python code to display normalized errors**

```
ConfusionMatrixDisplay.from_predictions(ytrain,ypred,normalize="true",values_format=".0%")
plt.show()
```

**#Python code to zoom-up the major error classifications**

```
sample_weight=(ypred!=ytrain)
ConfusionMatrixDisplay.from_predictions(ytrain,ypred,normalize="true",sample_weight=sample_weight,values_format=".0%")
plt.show()
```



Multilabel classification:

It is used when there are two or more classes and the data we want to classify may belong to none of the classes or all of them at the same time, e.g. to classify which traffic signs are contained on an image. In multi-label classification, the training set is composed of instances each associated with a set of labels, and the task is to predict the label sets of unseen instances through analyzing training instances with known label sets.

Difference between multi-class classification & multi-label classification is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related.

For example, multi-class classification makes the assumption that each sample is assigned to one and only one label: a fruit can be either an apple or a pear but not both at the same time. Whereas, an instance of multi-label classification can be that a text might be about any of religion, politics, finance or education at the same time or none of these.

Multioutput Algorithms

Multioutput algorithms are a type of machine learning approach designed for problems where the output consists of multiple variables, and each variable can belong to a different class or have a different range of values. In other words, multioutput problems involve predicting multiple dependent variables simultaneously.

Two main types of Multioutput Problems:

- Multioutput Classification: In multioutput classification, each instance is associated with a set of labels and the goal is to predict these labels simultaneously.
- Multioutput Regression: In multioutput regression, the task is to predict multiple continuous variables simultaneously.

MODULE-III

Linear Regression

Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between the dependent variable and one or more independent features by fitting a linear equation to observed data. Linear regression is not merely a predictive tool; it forms the basis for various advanced models. Techniques like regularization and support vector machines draw inspiration from linear regression, expanding its utility. Additionally, linear regression is a cornerstone in assumption testing, enabling researchers to validate key assumptions about the data.

Linear regression model prediction:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- \hat{y} is the predicted value.

n is the number of features.

x_i is the i^{th} feature value.

θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \dots, \theta_n$).

Linear Regression model prediction (vectorized form)

$$\hat{y} = h_{\theta}(\mathbf{x}) = \boldsymbol{\theta} \cdot \mathbf{x}$$

MSE cost function for a Linear Regression model

$$\text{MSE}(\mathbf{X}, h_{\theta}) = \frac{1}{m} \sum_{i=1}^m (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)})^2$$

Normal Equation

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

#Python code to predict Linear regression using Normal Equation

```
import numpy as np
n=100
x=2*np.random.randn(n,1)
y=4+3*x+np.random.randn(n,1)
from sklearn.preprocessing import add_dummy_feature
x1=add_dummy_feature(x)
theta=np.linalg.inv(x1.T@x1)@x1.T@y
xnew=np.array([[0],[2]])
xnew1=add_dummy_feature(xnew)
ypred=xnew1@theta
```

ypred

#Python code to perform Linear Regression based on the library function

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(x,y)
print(lr.intercept_, lr.coef_)
```

Gradient Descent (GD)

It is a widely used optimization algorithm in machine learning and deep learning that minimises the cost function of a neural network model during training. It works by iteratively adjusting the weights or parameters of the model in the direction of the negative gradient of the cost function until the minimum of the cost function is reached. The learning happens during the backpropagation while training the neural network-based model. There is a term known as Gradient Descent, which is used to optimize the weight and biases based on the cost function. The cost function evaluates the difference between the actual and predicted outputs.

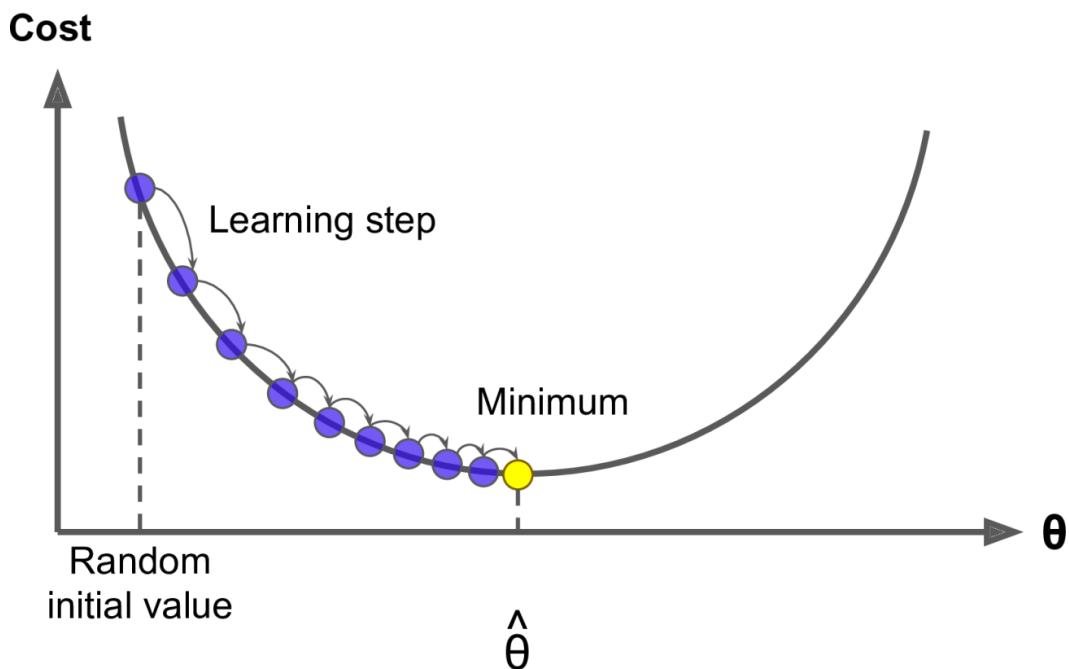
Gradient Descent is a fundamental optimization algorithm in machine learning used to minimize the cost or loss function during model training.

- It iteratively adjusts model parameters by moving in the direction of the steepest decrease in the cost function.
- The algorithm calculates gradients, representing the partial derivatives of the cost function concerning each parameter.

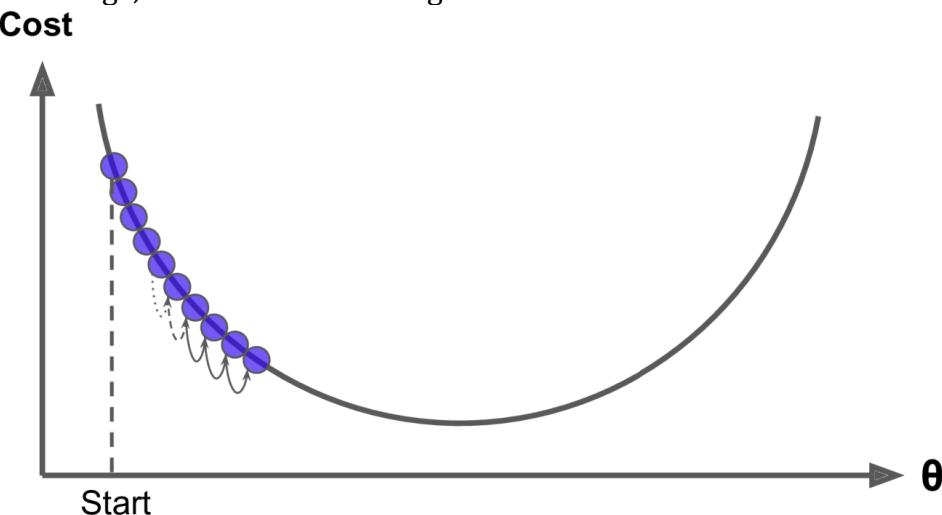
These gradients guide the updates, ensuring convergence towards the optimal parameter values that yield the lowest possible cost.

Gradient Descent is versatile and applicable to various machine learning models, including linear regression and neural networks.

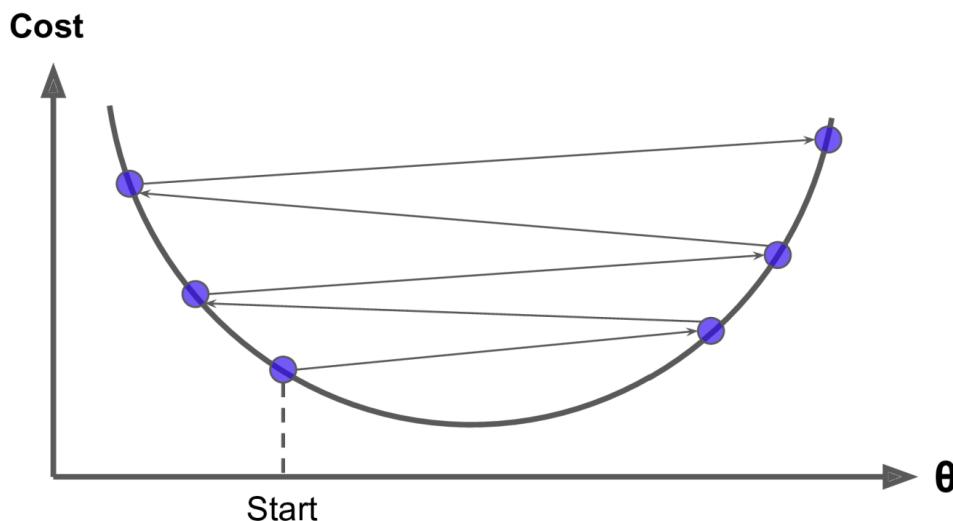
The path of Gradient descent is depicted in the following figure:

**Impact of Learning rate:**

If the learning rate is too small, then the algorithm will have to go through many iterations to converge, which will take a long time:

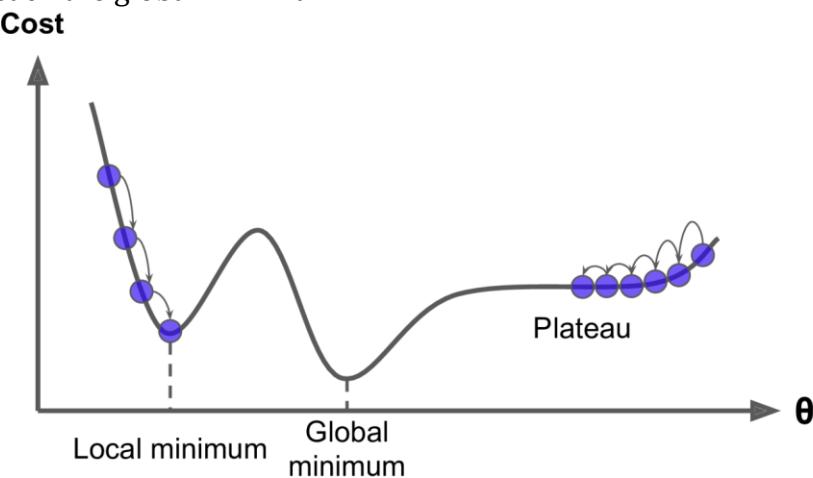


On the other hand, if the learning rate is too high, you might jump across the valley and end up on the other side, possibly even higher up than you were before. This might make the algorithm diverge, with larger and larger values, failing to find a good solution



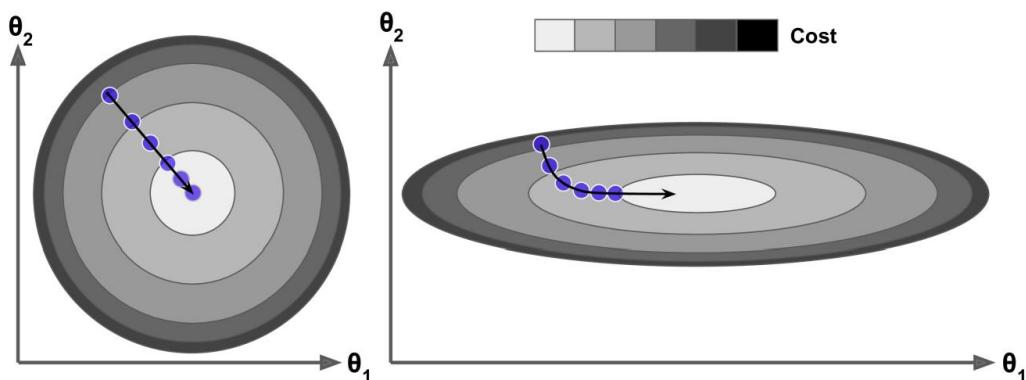
Challenges in Gradient Descent:

if the random initialization starts the algorithm on the left, then it will converge to a local minimum, which is not as good as the global minimum. If it starts on the right, then it will take a very long time to cross the plateau, and if you stop too early you will never reach the global minimum.



Impact of scaling on Gradient Descent:

Gradient Descent on a training set where features 1 and 2 have the same scale (on the left), and on a training set where feature 1 has much smaller values than feature 2 (on the right)



Batch Gradient Descent

In Batch Gradient Descent, all the training data is taken into consideration to take a single step. We take the average of the gradients of all the training examples and then use that mean gradient to update our parameters. So that's just one step of gradient descent in one epoch.

Cost function in Batch Gradient descent:

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\boldsymbol{\theta}) = \frac{2}{m} \sum_{i=1}^m (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

Gradient vector of the cost function

$$\nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\boldsymbol{\theta}) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\boldsymbol{\theta}) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})$$

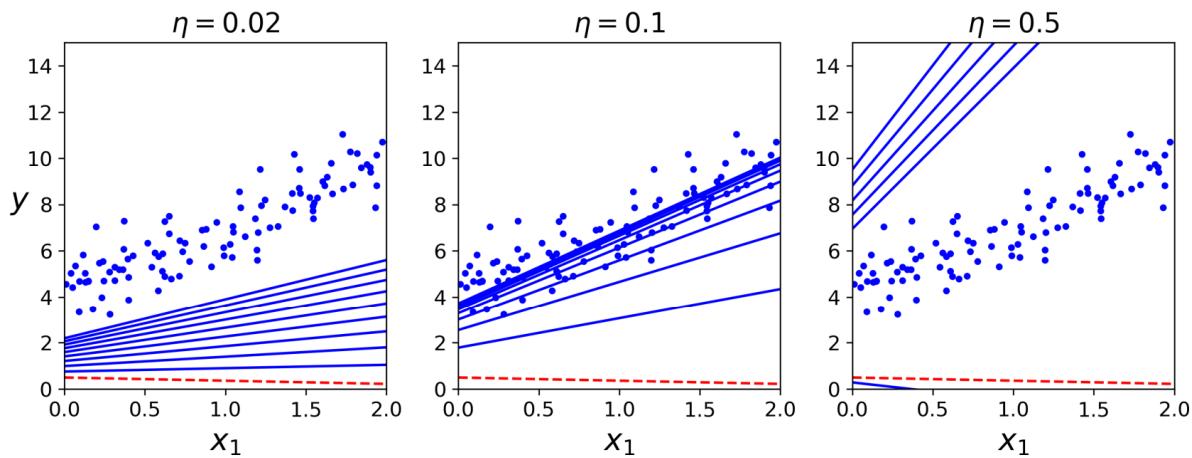
Gradient Descent step

$$\boldsymbol{\theta}^{(\text{next step})} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \text{MSE}(\boldsymbol{\theta})$$

#Python code to implement Full/Batch Gradient Descent

```
from sklearn.preprocessing import add_dummy_feature
eta=0.1
n=100
x=2*np.random.randn(n,1)
y=4+3*x+np.random.randn(n,1)
x1=add_dummy_feature(x)
m=len(x1)
theta=np.random.randn(2,1)
epochs=1000
for epoch in range(epochs):
    grad=2/m*x1.T@(x1@theta-y)
    theta=theta-eta*grad
theta
```

Full/Batch Gradient Descent with various learning rates:



On the left, the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time. In the middle, the learning rate looks pretty good: in just a few iterations, it has already converged to the solution. On the right, the learning rate is too high: the algorithm diverges.

Stochastic Gradient Descent

In Batch Gradient Descent we were considering all the examples for every step of Gradient Descent. But what if our dataset is very huge. Deep learning models crave for data. The more the data the more chances of a model to be good. Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one random sample at a time to take a single step. Also because the cost is so fluctuating, it will never reach the minima but it will keep dancing around it.

#Python code to implement Stochastic Gradient Descent

```
from sklearn.preprocessing import add_dummy_feature
eta=0.1
n=100
x=2*np.random.randn(n,1)
y=4+3*x+np.random.randn(n,1)
x1=add_dummy_feature(x)
m=len(x1)
theta=np.random.randn(2,1)
epochs=1000
m=len(x)
def learning_schedule(t):
    return 5/(50+t)

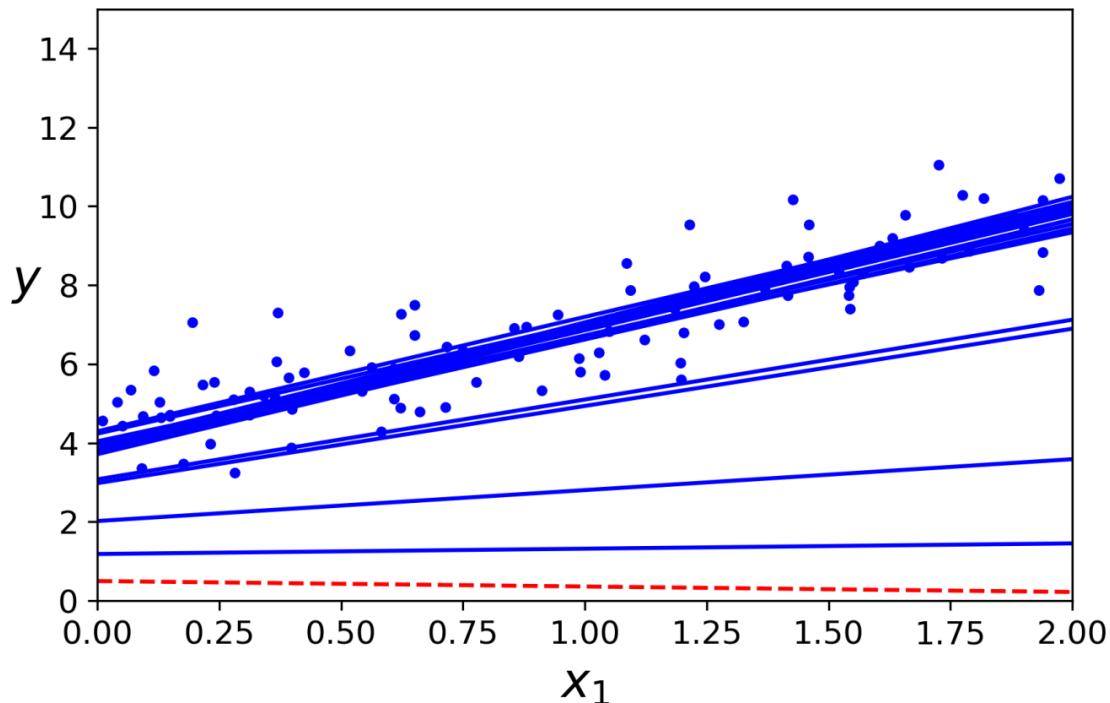
for epoch in range(epochs):
    for iteration in range(m):
        ri=np.random.randint(m)
        xi=x1[ri:ri+1]
        yi=y[ri:ri+1]
        grad=2/m*xi.T@(xi@theta-yi)
```

```

eta=learning_schedule(epoch*m+iteration)
theta=theta-eta*grad
theta

```

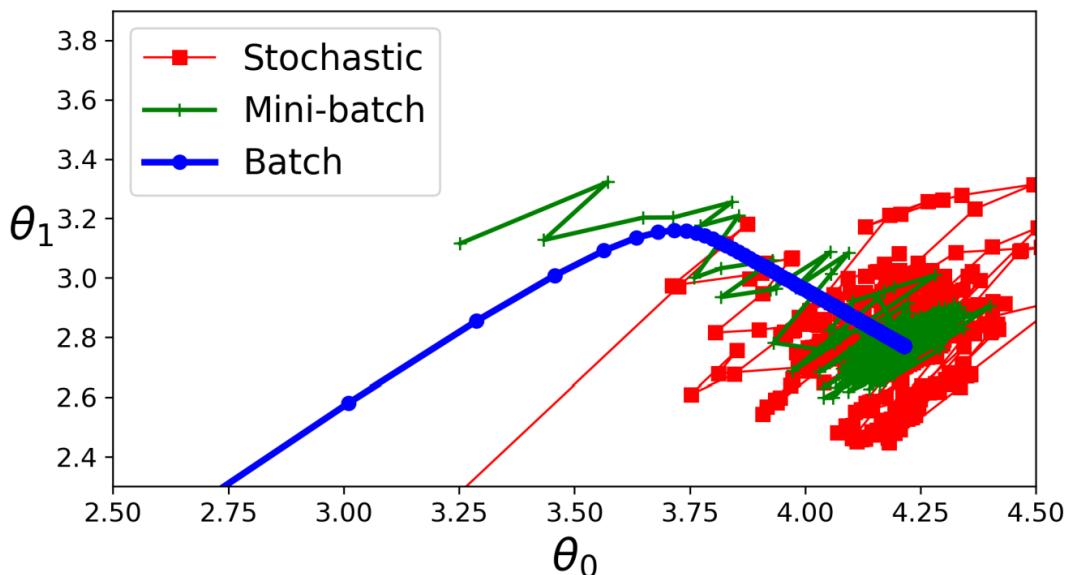
Performance of Stochastic Gradient Descent



Mini-batch Gradient Descent

We have seen the Batch Gradient Descent. We have also seen the Stochastic Gradient Descent. Batch Gradient Descent can be used for smoother curves. SGD can be used when the dataset is large. Batch Gradient Descent converges directly to minima. SGD converges faster for larger datasets. But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used. Neither we use all the dataset all at once nor we use the single example at a time. We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants.

Gradient Descent paths in parameter space



All algorithms end up near the minimum, but Batch GD's path actually stops at the minimum, while both Stochastic GD and Mini-batch GD continue to walk around. However, don't forget that Batch GD takes a lot of time to take each step, and Stochastic GD and Mini-batch GD would also reach the minimum if you used a good learning schedule.

Polynomial Regression

Polynomial regression is a type of regression analysis used in statistics and machine learning when the relationship between the independent variable (input) and the dependent variable (output) is not linear. While simple linear regression models the relationship as a straight line, polynomial regression allows for more flexibility by fitting a polynomial equation to the data. When the relationship between the variables is better represented by a curve rather than a straight line, polynomial regression can capture the non-linear patterns in the data.

#Python code for polynomial Regression

```
m=100
import numpy as np
x=6*np.random.randn(m,1)-3
y=0.5*x**2+x+2+np.random.randn(m,1)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
pf=PolynomialFeatures(degree=2,include_bias=False)
xpoly=pf.fit_transform(x)
lr=LinearRegression()
lr.fit(xpoly,y)
lr.intercept_,lr.coef_
```

Learning Curves

Learning curves are plots used to show a model's performance as the training set size increases. Another way it can be used is to show the model's performance over a defined period of time. We typically used them to diagnose algorithms that learn incrementally

from data. It works by evaluating a model on the training and validation datasets, then plotting the measured performance.

Finding the right degree of a polynomial is a challenge and learning curves help in resolving it. Learning curves are the plots of training and validation error as a function of the training iteration.

#Python code to illustrate the use of Learning curves with Linear Regression

```
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
tsize,tscore,vscore=learning_curve(LinearRegression(),x,y,
                                  train_sizes=np.linspace(0.01,1,40),
                                  cv=5,scoring="neg_root_mean_squared_error")
trainerr=-tscore.mean(axis=1)
validerr=-vscore.mean(axis=1)
import matplotlib.pyplot as plt
plt.plot(tsize,trainerr)
plt.plot(tsize,validerr)
plt.legend(["train","valid"])
plt.show()
```

#Python code to illustrate the use of Learning curves with Polynomial Regression

```
from sklearn.pipeline import make_pipeline
pr=make_pipeline(PolynomialFeatures(degree=2,include_bias=False),
                  LinearRegression())
tsize,tscore,vscore=learning_curve(pr,x,y,
                                  train_sizes=np.linspace(0.01,1,40),
                                  cv=5,scoring="neg_root_mean_squared_error")
trainerr=-tscore.mean(axis=1)
validerr=-vscore.mean(axis=1)
import matplotlib.pyplot as plt
plt.plot(tsize,trainerr)
plt.plot(tsize,validerr)
plt.legend(["train","valid"])
plt.show()
```

Regularized Linear Models

Regularization is one of the most important concepts of machine learning. It is a technique to prevent the model from overfitting by adding extra information to it. Regularization works by adding a penalty or complexity term to the complex mode

There are three types of regularization techniques, which are given below:

- Ridge Regression
- Lasso Regression
- ElasticNet Regression

Ridge Regression

- Ridge regression is one of the types of linear regression in which a small amount of bias is introduced so that we can get better long-term predictions.

- Ridge regression is a regularization technique, which is used to reduce the complexity of the model. It is also called as L2 regularization.
- In this technique, the cost function is altered by adding the penalty term to it. The amount of bias added to the model is called Ridge Regression penalty. We can calculate it by multiplying with the lambda to the squared weight of each individual feature.
- The equation for the cost function in ridge regression will be:

$$J(\theta) = \text{MSE}(\theta) + \frac{\alpha}{m} \sum_{i=1}^m \theta_i^2$$

Normal Equation with Ridge regression

$$\hat{\theta} = (x^T x + \alpha I)^{-1} x^T y$$

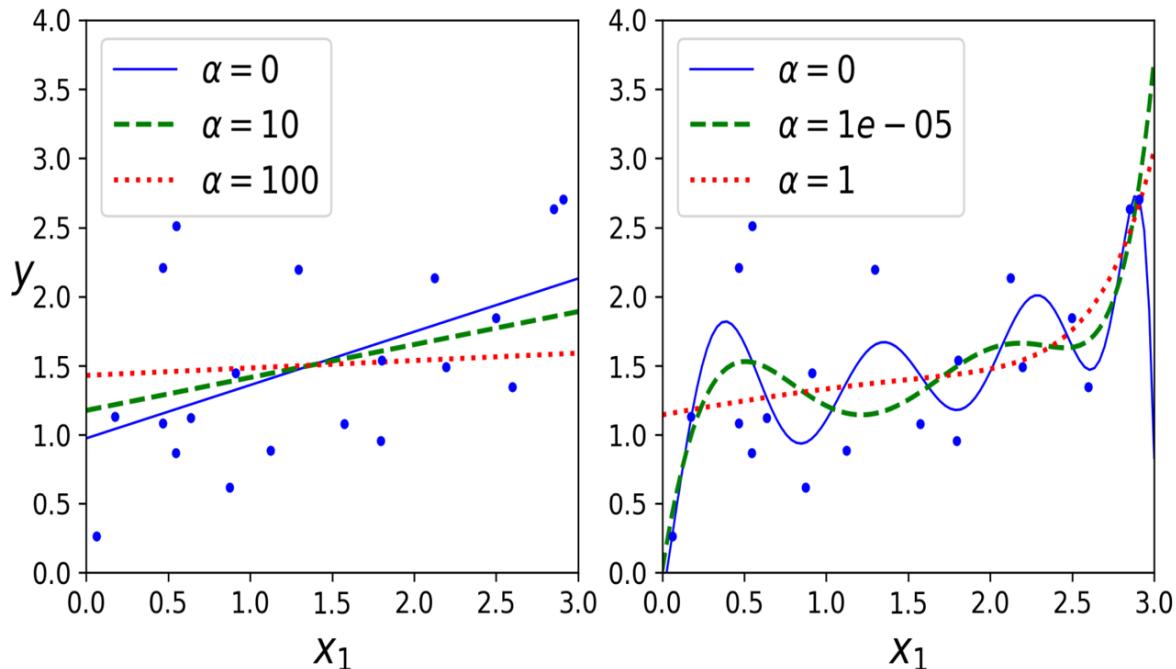
#Python code for Ridge Regression

```
from sklearn.linear_model import Ridge
ridge_reg = Ridge(alpha=1)
ridge_reg.fit(X, y)
```

OR

```
sgd_reg = SGDRegressor(penalty="l2")
```

Performance of Ridge regression for Linear and Non-linear cases:



Lasso Regression

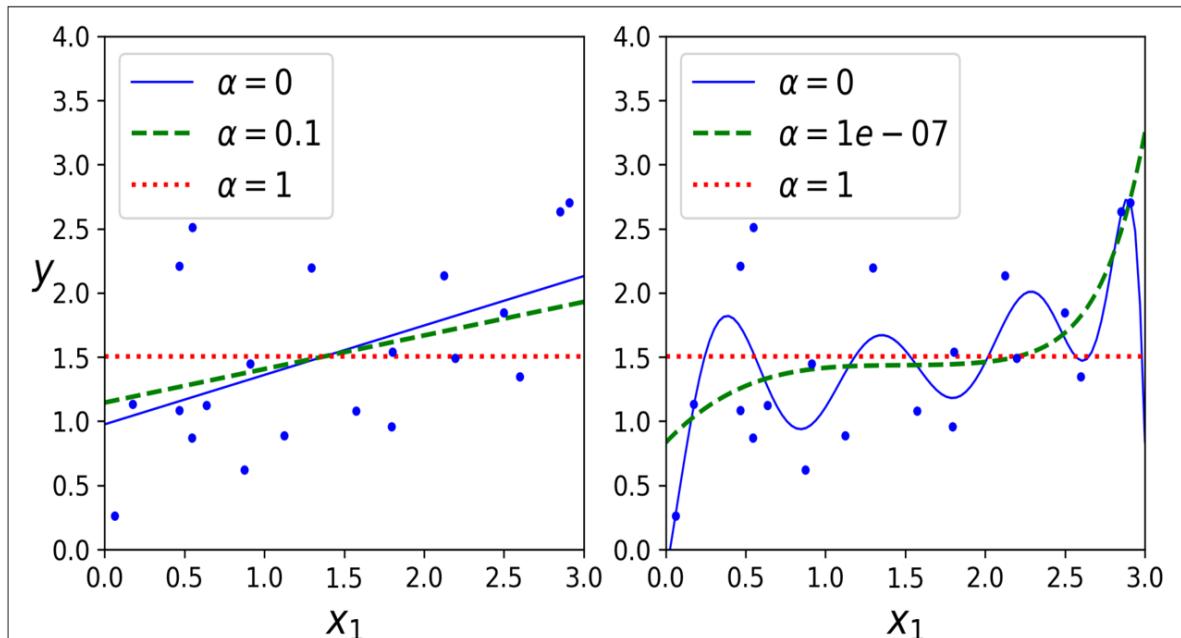
- Lasso regression is another regularization technique to reduce the complexity of the model. It stands for Least Absolute and Selection Operator.
- It is similar to the Ridge Regression except that the penalty term contains only the absolute weights instead of a square of weights.
- Since it takes absolute values, hence, it can shrink the slope to 0, whereas Ridge Regression can only shrink it near to 0.

- It is also called as L1 regularization. The equation for the cost function of Lasso regression will be:

$$J(\theta) = \text{MSE}(\theta) + 2\alpha \sum_{i=1}^m |\theta_i|$$

```
#Python code for Lasso Regression
from sklearn.linear_model import Lasso
lasso_reg = Lasso(alpha=0.1)
lasso_reg.fit(X, y)
```

Performance of Lasso Regression



Elastic Net Regression

Elastic Net Regression is a powerful machine learning algorithm that combines the features of both Lasso and Ridge Regression. It is a regularized regression technique that is used to deal with the problems of multicollinearity and overfitting, which are common in high-dimensional datasets. This algorithm works by adding a penalty term to the standard least-squares objective function

Cost function of Elastic net regression:

$$J(\theta) = \text{MSE}(\theta) + (1 - r) \cdot \frac{\alpha}{m} \sum_{i=1}^m \theta_i^2 + r \cdot 2\alpha \sum_{i=1}^m |\theta_i|$$

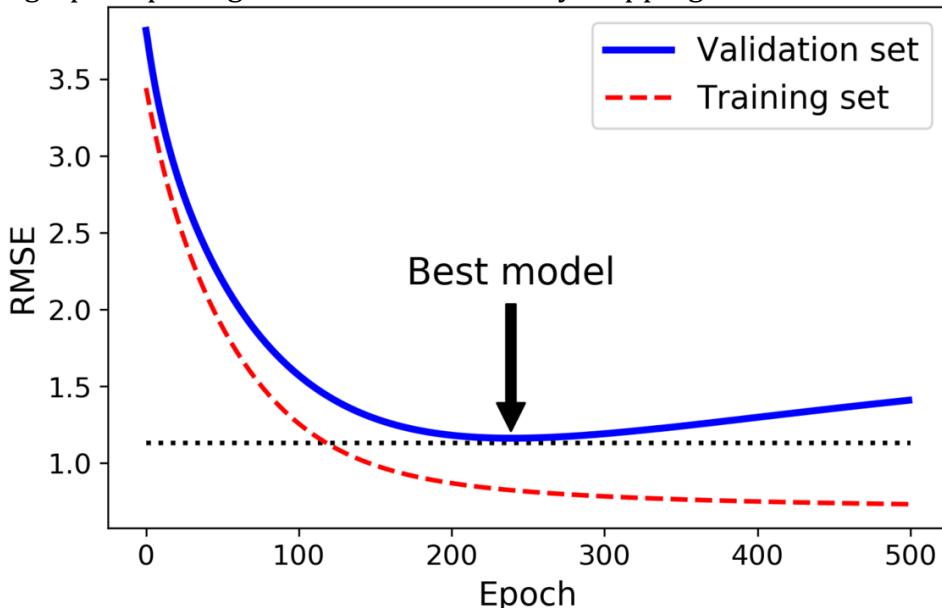
#Python code for Elastic net regression:

```
from sklearn.linear_model import ElasticNet
el_reg = ElasticNet(alpha=0.1, l1_ratio=0.5)
el_reg.fit(X, y)
```

Regularization by Early Stopping

In Regularization by Early Stopping, we stop training the model when the performance on the validation set is getting worse- increasing loss decreasing accuracy, or poorer scores of the scoring metric. By plotting the error on the training dataset and the validation dataset together, both the errors decrease with a number of iterations until the point where the model starts to overfit. After this point, the training error still decreases but the validation error increases. So, even if training is continued after this point, early stopping essentially returns the set of parameters that were used at this point and so is equivalent to stopping training at that point. So, the final parameters returned will enable the model to have low variance and better generalization. The model at the time the training is stopped will have a better generalization performance than the model with the least training errors.

The graph depicting the mechanism of early stopping:



Early stopping can be best used to prevent overfitting of the model, and saving resources. It would give best results if taken care of few things like – parameter tuning, preventing the model from overfitting, and ensuring that the model learns enough from the data.

Logistic Regression

Logistic regression is a supervised machine learning algorithm used for classification tasks where the goal is to predict the probability that an instance belongs to a given class or not. Logistic regression is used for binary classification where we use sigmoid function, that takes input as independent variables and produces a probability value between 0 and 1. Logistic regression predicts the output of a categorical dependent variable. Therefore, the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1. In Logistic regression, instead of fitting a regression line, we fit an “S” shaped logistic function, which predicts two maximum values (0 or 1).

Estimating probabilities in Logistic Regression:

$$\hat{p} = \sigma(\theta^T x)$$

$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

$$\hat{y} = \begin{cases} 0 & \text{if } \hat{p} < 0.5 \\ 1 & \text{if } \hat{p} \geq 0.5 \end{cases}$$

Cost function of a single training instance of Logistic Regression

$$c(\theta) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1 - \hat{p}) & \text{if } y = 0 \end{cases}$$

Logistic Regression cost function (log loss)

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right]$$

Decision Boundaries

The fundamental application of logistic regression is to determine a decision boundary for a binary classification problem. Although the baseline is to identify a binary decision boundary, the approach can be very well applied for scenarios with multiple classification classes or multi-class classification.

#Python code to compute decision boundary for iris dataset

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris
iris=load_iris(as_frame=True)
x=iris.data[['petal width (cm)']].values
y=iris.target_names[iris.target]=="virginica"
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.30)
lr=LogisticRegression()
lr.fit(xtrain,ytrain)
xnew=np.linspace(0,3,1000).reshape(-1,1)
yp=lr.predict_proba(xnew)
decisionboundary=xnew[yp[:,1]>=0.5][0,0]
print(decisionboundary)
```

Softmax regression

Softmax regression (or multinomial logistic regression) is a generalization of logistic regression to the case where we want to handle multiple classes in the target column. Softmax score for class k

$$s_k(\mathbf{x}) = \mathbf{x}^T \boldsymbol{\theta}^{(k)}$$

Softmax function

$$\hat{p}_k = \sigma(\mathbf{s}(\mathbf{x}))_k = \frac{\exp(s_k(\mathbf{x}))}{\sum_{j=1}^K \exp(s_j(\mathbf{x}))}$$

K is the number of classes.

$\mathbf{s}(\mathbf{x})$ is a vector containing the scores of each class for the instance \mathbf{x} .

$\sigma(\mathbf{s}(\mathbf{x}))_k$ is the estimated probability that the instance \mathbf{x} belongs to class k given the scores of each class for that instance.

Softmax Regression classifier prediction

$$\hat{y} = \operatorname{argmax}_k \sigma(\mathbf{s}(\mathbf{x}))_k = \operatorname{argmax}_k s_k(\mathbf{x}) = \operatorname{argmax}_k \left((\boldsymbol{\theta}^{(k)})^T \mathbf{x} \right)$$

The argmax operator returns the value of a variable that maximizes a function.

Cross entropy is frequently used to measure how well a set of estimated class probabilities match the target classes

Cross entropy cost function

$$J(\boldsymbol{\Theta}) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(\hat{p}_k^{(i)})$$

MODULE IV:

Decision Trees

A **decision tree** is a flowchart-like structure used to make decisions or predictions. It consists of nodes representing decisions or tests on attributes, branches representing the outcome of these decisions, and leaf nodes representing final outcomes or predictions. Each internal node corresponds to a test on an attribute, each branch corresponds to the result of the test, and each leaf node corresponds to a class label or a continuous value. Decision trees are versatile ML algorithms used for classification, regression and multi-output classification. Also suitable for any complex datasets.

Training and visualizing a decision tree

#Python code to construct a decision tree classifier for Iris dataset

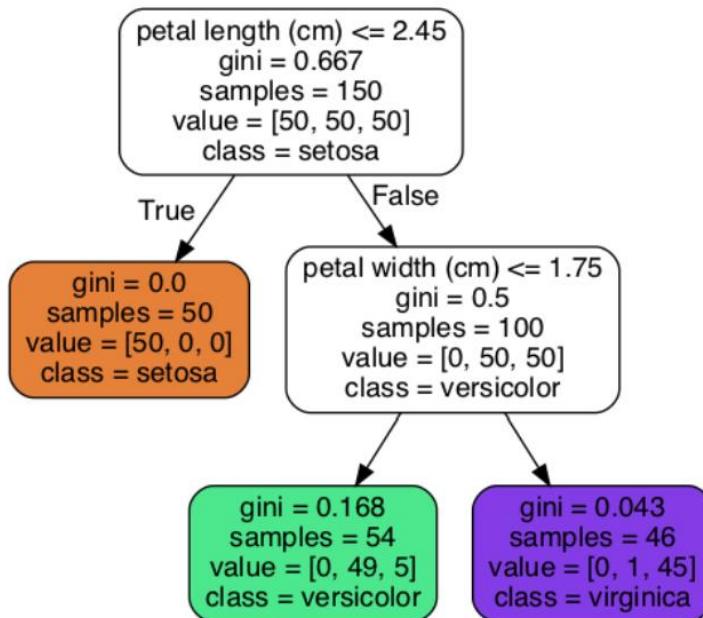
```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
iris=load_iris(as_frame=True)
x=iris.data[["petal length (cm)","petal width (cm)"]].values
y=iris.target
treeclf=DecisionTreeClassifier(max_depth=2,criterion="entropy")
treeclf.fit(x,y)
```

#Creating a graphics file for decision tree

```
from sklearn.tree import export_graphviz
export_graphviz(
    treeclf,
    out_file="o.dot",
    feature_names=["petal length (cm)","petal width (cm)"],
    class_names=iris.target_names,
    rounded=True,
    filled=True
)
```

```
from graphviz import Source
Source.from_file("o.dot")
```

#Visualization result of a Decision tree



#Prediction result of decision tree

- Traverse the tree to find leaf node instance
 - Return ratio of training instances of class k in this node
- treeclf.predict_proba([[5,1.5]]).round(3)

Gini Impurity

The equation for computing Gini Impurity is:

$$G_i = 1 - \sum_{k=1}^n p_{i,k}^2$$

Where n-number of classes,

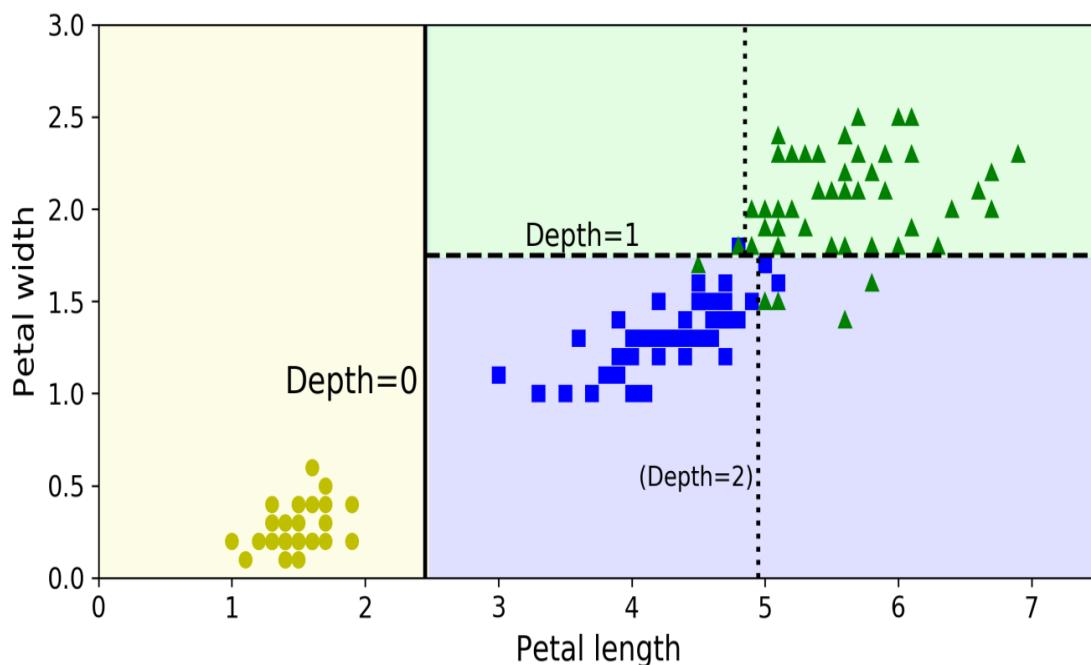
A node's gini attribute measures its Gini impurity. A node is pure if all training instances is pure and Gini impurity is 0.

Eg:

For the root node,

$$\begin{aligned} \text{Gini impurity} &= 1 - \left[\left(\frac{50}{150}\right)^2 + \left(\frac{50}{150}\right)^2 + \left(\frac{50}{150}\right)^2 \right] \\ &= 1 - [1/9 + 1/9 + 1/9] = 1 - 1/3 = 2/3 == 0.6667 \end{aligned}$$

Decision tree boundaries



The thick vertical line represents the decision boundary of the root node (depth 0): petal length = 2.45 cm. Since the left area is pure (only Iris-Setosa), it cannot be split any further. However, the right area is impure, so the depth-1 right node splits it at petal width = 1.75 cm (represented by the dashed line). Since `max_depth` was set to 2, the Decision Tree stops right there. However, if you set `max_depth` to 3, then the two depth-2 nodes would each add another decision boundary (represented by the dotted lines)

Whitebox and Blackbox Machine Learning models

A white box machine learning model (White Box) is one that allows humans to easily interpret how it was able to produce its output and draw its conclusions, thereby giving us insight into the algorithm's inner workings. White boxes are transparent in terms of:

- How they behave
- How they process data
- Which variables they give weight to
- How they generate their predictions

Examples of such models include linear trees, decision trees, and regression trees.

Black box machine learning models (Black Boxes), on the other hand, rank higher on innovation and accuracy, but lower on transparency and interpretability. Black Boxes produce output based on your input data set, but do not – and cannot – clarify how they came to those conclusions. So, while a user can observe the input variable and the output variable, everything in between related to the calculation and the process is not available. Even if it were, humans would not be able to understand it.

Black Boxes tend to model extremely complex scenarios with deep and non-linear interactions between the data. Some examples include:

- Deep-learning models

- Boosting models
- Random forest models

CART Training algorithm

CART(Classification And Regression Trees) is a variation of the decision tree algorithm. It can handle both classification and regression tasks. Scikit-Learn uses the Classification And Regression Tree (CART) algorithm to train Decision Trees (also called “growing” trees)

Classification and Regression Trees (CART) is a decision tree algorithm that is used for both classification and regression tasks. It is a supervised learning algorithm that learns from labelled data to predict unseen data.

- **Tree structure:** CART builds a tree-like structure consisting of nodes and branches. The nodes represent different decision points, and the branches represent the possible outcomes of those decisions. The leaf nodes in the tree contain a predicted class label or value for the target variable.
- **Splitting criteria:** CART uses a greedy approach to split the data at each node. It evaluates all possible splits and selects the one that best reduces the impurity of the resulting subsets. For classification tasks, CART uses Gini impurity as the splitting criterion. The lower the Gini impurity, the more pure the subset is. For regression tasks, CART uses residual reduction as the splitting criterion. The lower the residual reduction, the better the fit of the model to the data.
- **Pruning:** To prevent overfitting of the data, pruning is a technique used to remove the nodes that contribute little to the model accuracy. Cost complexity pruning and information gain pruning are two popular pruning techniques. Cost complexity pruning involves calculating the cost of each node and removing nodes that have a negative cost. Information gain pruning involves calculating the information gain of each node and removing nodes that have a low information gain.

CART algorithm Splits training dataset into two subsets using single feature k and threshold tk. It searches for pair (t,tk) that produces purest subsets weighted by their size. CART cost function for classification:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

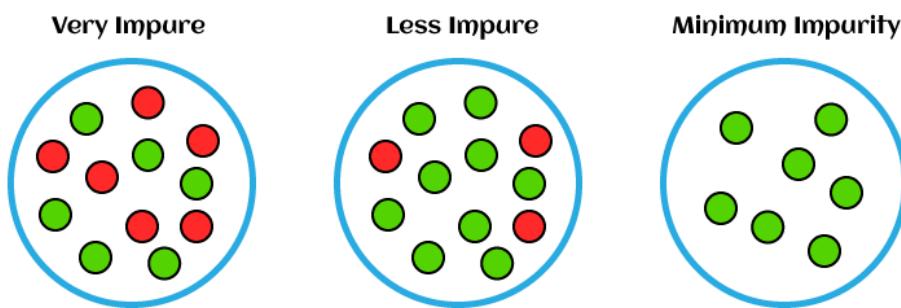
where $\begin{cases} G_{\text{left/right}} & \text{measures the impurity of the left/right subset,} \\ m_{\text{left/right}} & \text{is the number of instances in the left/right subset.} \end{cases}$

Entropy

- The word “entropy,” is hails from physics, and refers to an indicator of the disorder. The expected volume of “information,” “surprise,” or “uncertainty” associated with a

randomly chosen variable's potential outcomes is characterized as the entropy of the variable in information theory.

- Entropy is a quantifiable and measurable physical attribute and a scientific notion that is frequently associated with a circumstance of disorder, unpredictability, or uncertainty.
- From classical thermodynamics, where it was originally identified, through the macroscopic portrayal of existence in statistical physics, to the principles of information theory, the terminology, and notion are widely used in a variety of fields of study
- Entropy is defined as the randomness or measuring the disorder of the information being processed in Machine Learning. Further, in other words, we can say that **entropy is the machine learning metric that measures the unpredictability or impurity in the system.**



Formula to compute Entropy:

$$H_i = - \sum_{k=1}^n p_{i,k} \log_2 p_{i,k}$$

Entropy vs Gini Impurity

Gini Impurity	Entropy
It is the probability of misclassifying a randomly chosen element in a set.	While entropy measures the amount of uncertainty or randomness in a set.
The range of the Gini index is [0, 1], where 0 indicates perfect purity and 1 indicates maximum impurity.	The range of entropy is [0, log(c)], where c is the number of classes.
Gini index is a linear measure.	Entropy is a logarithmic measure.
It can be interpreted as the expected error rate in a classifier.	It can be interpreted as the average amount of information needed to specify the class of an instance.
It is sensitive to the distribution of classes in a set.	It is sensitive to the number of classes

Regularization hyperparameters of Decision tree

The few other hyperparameters that would restrict the structure of the decision tree are:

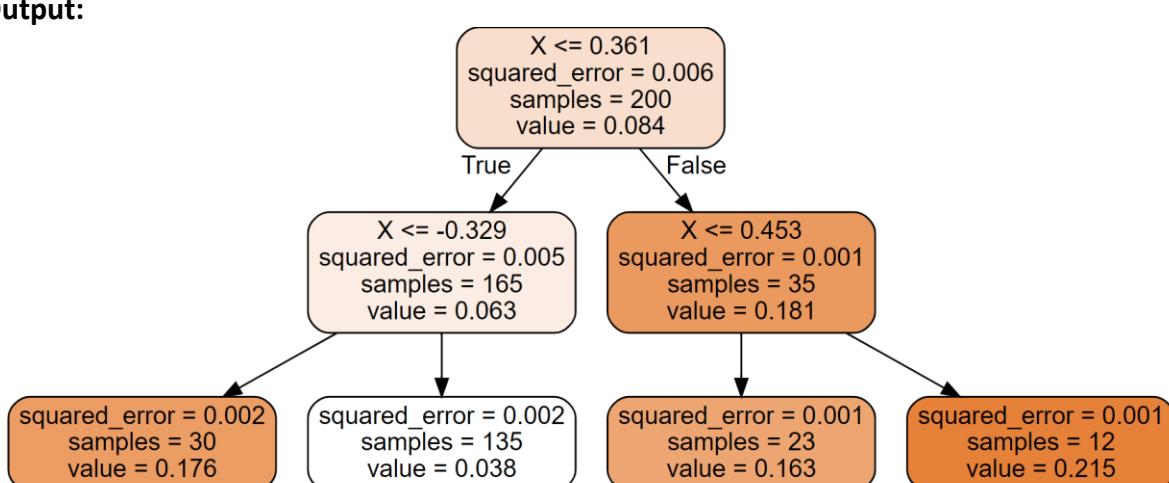
1. min_samples_split – Minimum number of samples a node must possess before splitting.
2. min_samples_leaf – Minimum number of samples a leaf node must possess.
3. min_weight_fraction_leaf – Minimum fraction of the sum total of weights required to be at a leaf node.
4. max_leaf_nodes – Maximum number of leaf nodes a decision tree can have.
5. max_features – Maximum number of features that are taken into the account for splitting each node.

Regression using Decision Trees

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
x=np.random.rand(200,1)-0.5
y=x**2+0.025*np.random.randn(200,1)
tr=DecisionTreeRegressor(max_depth=2)
tr.fit(x,y)
```

```
from sklearn.tree import export_graphviz
export_graphviz(
    tr,
    out_file="o.dot",
    feature_names=['X'],
    class_names=y,
    rounded=True,
    filled=True
)
from graphviz import Source
Source.from_file("o.dot")
```

Output:



CART Cost function for regression:

$$J(k, t_k) = \frac{m_{\text{left}}}{m} \text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m} \text{MSE}_{\text{right}}$$

Parametric and Non-parametric machine learning models

Machine learning models are widely classified into two types: parametric and nonparametric models. Models of the first category make specific hypotheses about the relationship between input and output data. These assumptions concern a fixed number of parameters and variables that impact the model's result. Furthermore, these assumptions are associated with a set of parameters that must be learned during the training process. Some examples of parametric models in neural networks include linear or polynomial regression, which are straightforward models that imply that the input and output have a linear or polynomial relation respectively.

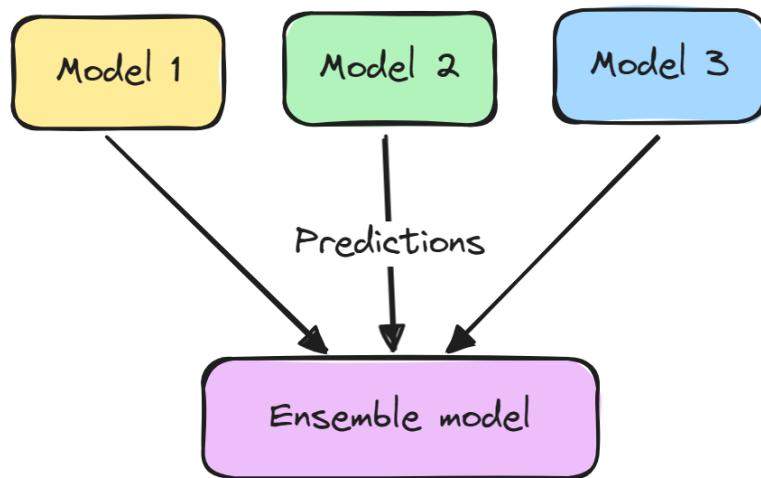
The second category includes non-parametric models. These models don't need to make assumptions about the relations between the input and output to generate an outcome and also don't require a certain number of parameters to be set and learned. Studies have shown that non-parametric perform better on large datasets and are more flexible. Common non-parametric algorithms are the random forests or decision trees that split the input into a smaller space based on the data features, generating the prediction based on the class.

Ensemble learning:

Ensemble learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

The underlying concept behind ensemble learning is to combine the outputs of diverse models to create a more precise prediction. By considering multiple perspectives and utilizing the strengths of different models, ensemble learning improves the overall performance of the learning system. This approach not only enhances accuracy but also provides resilience against uncertainties in the data. By effectively merging predictions from multiple models, ensemble learning has proven to be a powerful tool in various domains, offering more robust and reliable forecasts.

Eg:



Aggregating the predictions of group of predictors (regressors/classifiers) is called ensemble learning.

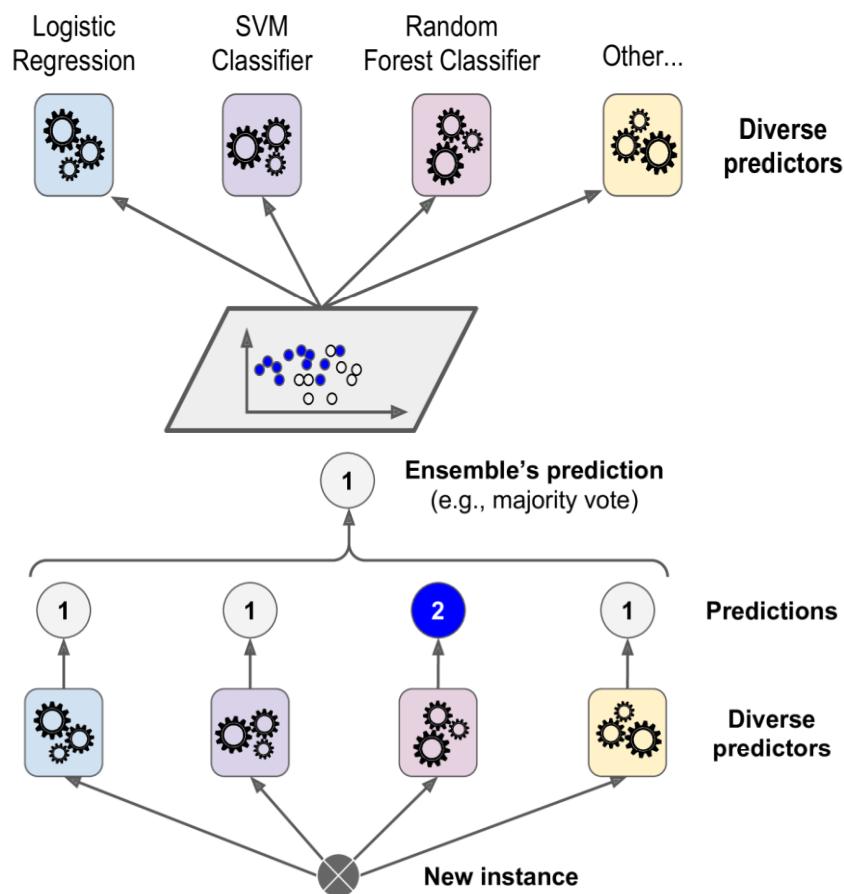
Voting Classifiers

A voting classifier is a machine learning model that gains experience by training on a collection of several models and forecasts an output (class) based on the class with the highest likelihood of becoming the output. To forecast the output class based on the largest majority of votes, it averages the results of each classifier provided into the voting classifier. The concept is to build a single model that learns from various models and predicts output based on their aggregate majority of votes for each output class, rather than building separate specialized models and determining the accuracy for each of them.

There are primarily **two** different types of voting classifiers:

- **Hard Voting:** In hard voting, the predicted output class is a class with the highest majority of votes. For example, let's say classifiers predicted the output classes as (Cat, Dog, Dog). As the classifiers predicted class "dog" a maximum number of times, we will proceed with Dog as our final prediction.
- **Soft Voting:** In this, the average probabilities of the classes determine which one will be the final prediction. For example, let's say the probabilities of the class being a "dog" is (0.30, 0.47, 0.53) and a "cat" is (0.20, 0.32, 0.40). So, the average for a class dog is 0.4333, and the cat is 0.3067, from this, we can confirm our final prediction to be a dog as it has the highest average probability.

Eg:



#Python code to demonstrate Hard voting classifiers

```

from sklearn.datasets import make_moons
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
x,y=make_moons(n_samples=500, noise=0.30)
xtrain,xtest,ytrain,ytest=train_test_split(x,y)
vcl=VotingClassifier(
    estimators=[
        ('lr',LogisticRegression()),
        ('rf',RandomForestClassifier()),
        ('svc',SVC()),
    ]
)
vcl.fit(xtrain,ytrain)
for name,clf in vcl.named_estimators_.items():
    print(name,clf.score(xtest,ytest))
vcl.score(xtest,ytest)

```

#Python code for softvoting classifiers

```
vcl.voting="soft"
vcl.named_estimators["svc"].probability=True
vcl.fit(xtrain,ytrain)
vcl.score(xtest,ytest)
```

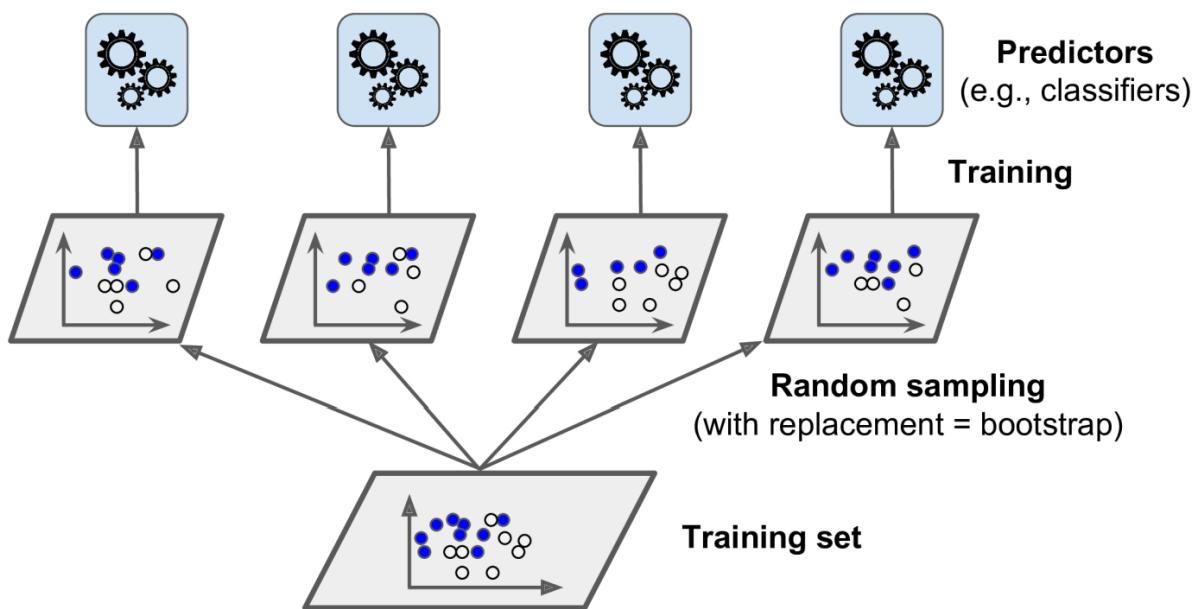
Bagging and Pasting

One way to get a diverse set of classifiers for ensemble learning is to use very different training algorithms. Another approach is to use the same training algorithm for every predictor and train them on different random subsets of the training set. When sampling is performed with replacement, this method is called bagging (short for bootstrap aggregating). When sampling is performed without replacement, it is called pasting.

Both bagging and pasting allow training instances to be sampled several times across multiple predictors, but only bagging allows training instances to be sampled several times for the same predictor.

Once all predictors are trained, the ensemble can make a prediction for a new instance by simply aggregating the predictions of all predictors. The aggregation function is typically the statistical mode for classification or the average for the regression. Predictors can all be trained in parallel, via different CPU cores. Similarly, predictions can be made in parallel. This is one of the reasons bagging and pasting scale very well.

Eg:



#Python code for Bagging/Pasting

```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
bag_clf = BaggingClassifier( base_estimator=DecisionTreeClassifier(), n_estimators=500,
max_samples=100, bootstrap=True, n_jobs=-1)
```

Bootstrapping introduces a bit more diversity in the subsets that each predictor is trained on, so bagging ends up with a slightly higher bias than pasting. But the extra diversity also means that the predictors end up being less correlated, so the ensemble's variance is reduced. Overall, bagging often results in better models.

Out-of-Bag Evaluation

With bagging, some instances may be sampled several times for any given predictor, while others may not be sampled at all. By default, *BaggingClassifier* samples m training instances with replacement (*bootstrap=True*), where m is the size of the training set. This means only about 63% of the training instances are sampled on average for each predictor. The remaining 37% of the training instances that are not sampled are called *out-of-bag (oob)* instances. Since a predictor never sees these instances during training, it can be evaluated on these instances, without the need for a separate validation set. We can evaluate the ensemble itself by averaging out the out-of-bag evaluations for each predictor.

In Scikit-Learn, we can set *oob_score=True* when creating a *BaggingClassifier* to request an automatic oob evaluation. The resulting evaluation score is available through the *oob_score_* variable.

#Python code for OOB Evaluation

```
bag_clf = BaggingClassifier( base_estimator=DecisionTreeClassifier(), n_estimators=500,
max_samples=100, bootstrap=True, n_jobs=-1, oob_score=True )
bag_clf.fit(X, y) print(bag_clf.oob_score_)
```

Random Patches and Random Subspaces

The *BaggingClassifier* class supports sampling the features as well. Sampling features is controlled by two hyperparameters: *max_features* and *bootstrap_features*. They work the same way as *max_samples* and *bootstrap*, but for feature sampling instead. Thus, each predictor will be trained on a random subset of the input features.

This is very useful when we are dealing with high-dimensional inputs. Sampling both training instances and features is called the *Random Patches method*. Keeping all the training instances but sampling features is called the *Random Subspaces method*.

Bias and variance in Machine Learning

Bias is simply defined as the inability of the model because of that there is some difference or error occurring between the model's predicted value and the actual value. These differences between actual or expected values and the predicted values are known as error or bias error or error due to bias.

Variance is the measure of spread in data from its mean position. In machine learning variance is the amount by which the performance of a predictive model changes when it is trained on different subsets of the training data. More specifically, variance is the variability of the model that how much it is sensitive to another subset of the training dataset. i.e. how much it can adjust on the new subset of the training dataset.

Different Combinations of Bias-Variance

There can be four combinations between bias and variance.

- **High Bias, Low Variance:** A model with high bias and low variance is said to be underfitting.
- **High Variance, Low Bias:** A model with high variance and low bias is said to be overfitting.
- **High-Bias, High-Variance:** A model has both high bias and high variance, which means that the model is not able to capture the underlying patterns in the data (high bias) and is also too sensitive to changes in the training data (high variance). As a result, the model will produce inconsistent and inaccurate predictions on average.
- **Low Bias, Low Variance:** A model that has low bias and low variance means that the model is able to capture the underlying patterns in the data (low bias) and is not too sensitive to changes in the training data (low variance). This is the ideal scenario for a machine learning model, as it is able to generalize well to new, unseen data and produce consistent and accurate predictions. But in practice, it's not possible.

Random Forests

Random Forest algorithm is a powerful tree learning technique in Machine Learning. It works by creating a number of Decision Trees during the training phase. Each tree is constructed using a random subset of the data set to measure a random subset of features in each partition. This randomness introduces variability among individual trees, reducing the risk of overfitting and improving overall prediction performance.

In prediction, the algorithm aggregates the results of all trees, either by voting (for classification tasks) or by averaging (for regression tasks). This collaborative decision-making process, supported by multiple trees with their insights, provides an example stable and precise results. Random forests are widely used for classification and regression functions, which are known for their ability to handle complex data, reduce overfitting, and provide reliable forecasts in different environments.

Random forest is an ensemble of Decision trees, trained with bagging method. The value of max_samples is set to 1.0. There is no need of pipelines in building classifiers/regressors. For splitting node, uses best feature among random subset of features.

#Python code to demonstrate Random Forest classifier

```
from sklearn.ensemble import RandomForestClassifier
rcl=RandomForestClassifier(n_estimators=1000,
                           max_leaf_nodes=16)
rcl.fit(xtrain,ytrain)
rcl.score(xtest,ytest)
```

Extra Tree classifier

Extremely Randomized Trees Classifier(Extra Trees Classifier) is a type of ensemble learning technique which aggregates the results of multiple de-correlated decision trees collected in a “forest” to output its classification result. In concept, it is very similar to a Random Forest Classifier and only differs from it in the manner of construction of the decision trees in the forest. Each Decision Tree in the Extra Trees Forest is constructed from the original training sample. Then, at each test node, Each tree is provided with a random sample of k features from the feature-set from which each decision tree must select the best feature to split the

data based on some mathematical criteria (typically the Gini Index). This random sample of features leads to the creation of multiple de-correlated decision trees.

Feature Importance

Features in machine learning, also known as variables or attributes, are individual measurable properties or characteristics of the phenomena being observed. They serve as the input to the model, and their quality and quantity can greatly influence the accuracy and efficiency of the model. Several techniques can be employed to calculate **feature importance in Random Forests**, each offering unique insights:

- **Built-in Feature Importance:** This method utilizes the model's internal calculations to measure feature importance, such as **Gini importance and mean decrease in accuracy**. Essentially, this method measures how much the **impurity** (or randomness) within a node of a decision tree decreases when a specific feature is used to split the data.
- **Permutation feature importance:** Permutation importance assesses the significance of each feature independently. By evaluating the impact of individual feature permutations on predictions, it calculates importance.
- **SHAP (SHapley Additive exPlanations) Values:** SHAP values delve deeper by explaining the contribution of each feature to individual predictions. This method offers a comprehensive understanding of feature importance across various data points.

#Python code to compute Feature Importance

```
from sklearn.datasets import load_iris
iris=load_iris(as_frame=True)
rcl=RandomForestClassifier(n_estimators=500)
rcl.fit(iris.data,iris.target)
for score,name in zip(rcl.feature_importances_,iris.data.columns):
    print(name,round(score,2))
```

Boosting

Boosting is an ensemble modeling technique that attempts to build a strong classifier from the number of weak classifiers. It is done by building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

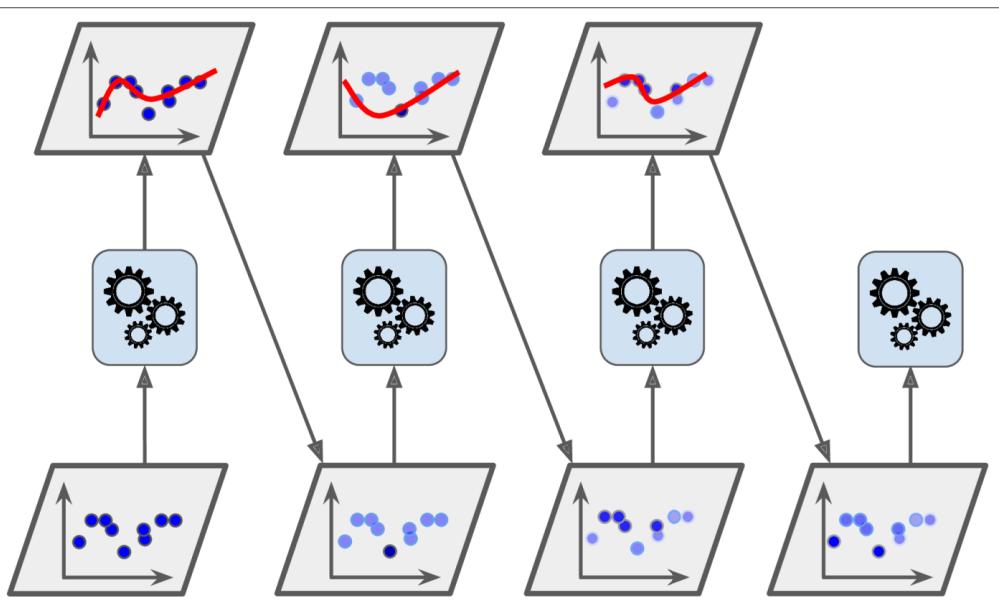
Advantages of Boosting

- Improved Accuracy – Boosting can improve the accuracy of the model by combining several weak models' accuracies and averaging them for regression or voting over them for classification to increase the accuracy of the final model.
- Robustness to Overfitting – Boosting can reduce the risk of overfitting by reweighting the inputs that are classified wrongly.
- Better handling of imbalanced data – Boosting can handle the imbalance data by focusing more on the data points that are misclassified

- Better Interpretability – Boosting can increase the interpretability of the model by breaking the model decision process into multiple processes.

Boosting methods

Adaboost – AdaBoost is a boosting algorithm that also works on the principle of the stagewise addition method where multiple weak learners are used for getting strong learners. The value of the alpha parameter, in this case, will be indirectly proportional to the error of the weak learner, Unlike Gradient Boosting in XGBoost, the alpha parameter calculated is related to the errors of the weak learner, here the value of the alpha parameter will be indirectly proportional to the error of the weak learner.



AdaBoost sequential training with instance weight updates

#Python code of Adaboost classifier

```
from sklearn.ensemble import AdaBoostClassifier
acl=AdaBoostClassifier(DecisionTreeClassifier(max_depth=1),
n_estimators=30,
learning_rate=0.5)
acl.fit(xtrain,ytrain)
```

Gradient Boosting – It is a boosting technique that builds a final model from the sum of several weak learning algorithms that were trained on the same dataset. It operates on the idea of stagewise addition. The first weak learner in the gradient boosting algorithm will not be trained on the dataset; instead, it will simply return the mean of the relevant column. The residual for the first weak learner algorithm's output will then be calculated and used as the output column or target column for the next weak learning algorithm that will be trained. The second weak learner will be trained using the same methodology, and the residuals will be computed and utilized as an output column once more for the third weak learner, and so on until we achieve zero residuals. The dataset for gradient boosting must be in the form of numerical or categorical data, and the loss function used to generate the residuals must be differential at all times.

#Python code for Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor
gbrt = GradientBoostingRegressor(max_depth=2, n_estimators=3, learning_rate=1.0)
gbrt.fit(X, y)
```

Histogram based Gradient Boosting

Histogram Gradient Boosting (HGB) is an optimized version of Gradient Boosting that enhances speed and efficiency. Here's a detailed breakdown of how it works:

Step-by-Step Working of Histogram Gradient Boosting**Feature Binning:**

- Continuous features are divided into discrete bins (histograms). For example, a feature with continuous values might be divided into 256 bins.
- This binning process reduces the number of unique feature values, which simplifies and accelerates computations.

Gradient Calculation:

- Compute gradients based on the binned features. The gradient indicates how much and in which direction we need to adjust predictions to reduce the error.

Histogram Construction:

- For each feature, build a histogram of gradient values. Each bin in the histogram represents the sum of gradients for data points that fall into that bin.
- This step transforms the gradient computation into a simpler histogram update process.

Tree Building:

- Decision trees are constructed using these histograms. The algorithm evaluates split points by considering the bins rather than individual data points, which speeds up the process.
- Trees are added iteratively, each aiming to reduce the residual errors of the current ensemble model.

Model Update:

- After adding a new tree, the model is updated, and the process repeats. Each iteration seeks to improve the model by reducing the errors incrementally.

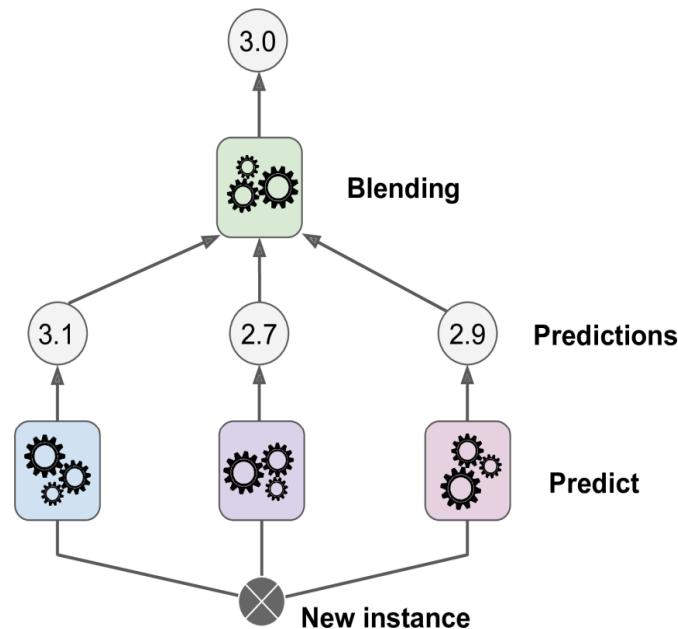
Stacking

Stacking is a way to ensemble multiple classifications or regression model. There are many ways to ensemble models, the widely known models are **Bagging** or **Boosting**. Bagging allows multiple similar models with high variance are averaged to decrease variance. Boosting builds multiple incremental models to decrease the bias, while keeping variance small.

Stacking (sometimes called *Stacked Generalization*) is a different paradigm. The point of stacking is to explore a space of different models for the same problem. The idea is that you can attack a learning problem with different types of models which are capable to learn some part of the problem, but not the whole space of the problem. So, you can build multiple different learners and you use them to build an intermediate prediction, one prediction for each learned model. Then you add a new model which learns from the intermediate predictions the same target.

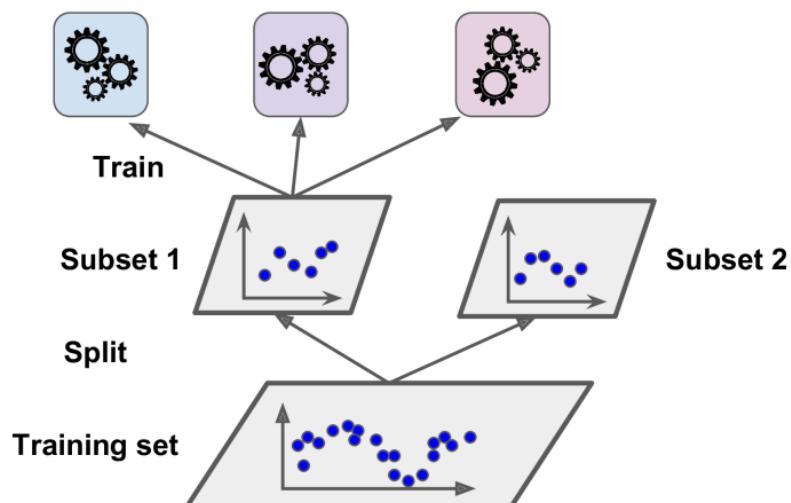
This final model is said to be stacked on the top of the others, hence the name. Thus, you might improve your overall performance, and often you end up with a model which is better than any individual intermediate model. Notice however, that it does not give you any guarantee, as is often the case with any machine learning technique.

Eg:

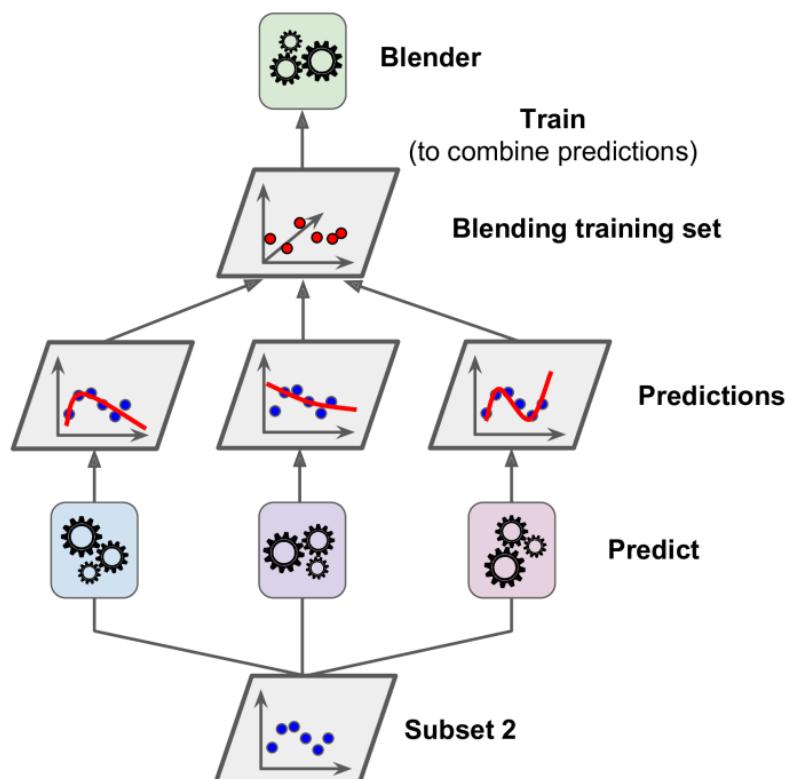


Aggregating predictions using a blending predictor

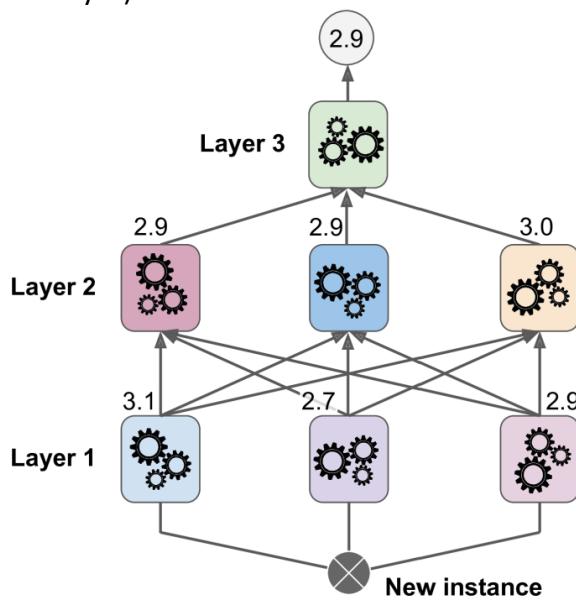
To train the blender, a common approach is to use a hold-out set. First, the training set is split in two subsets. The first subset is used to train the predictors in the first layer



The blender is trained on this new training set, so it learns to predict the target value given the first layer's predictions.



It is actually possible to train several different blenders this way (e.g., one using Linear Regression, another using Random Forest Regression, and so on): we get a whole layer of blenders. The trick is to split the training set into three subsets: the first one is used to train the first layer, the second one is used to create the training set used to train the second layer (using predictions made by the predictors of the first layer), and the third one is used to create the training set to train the third layer (using predictions made by the predictors of the second layer)



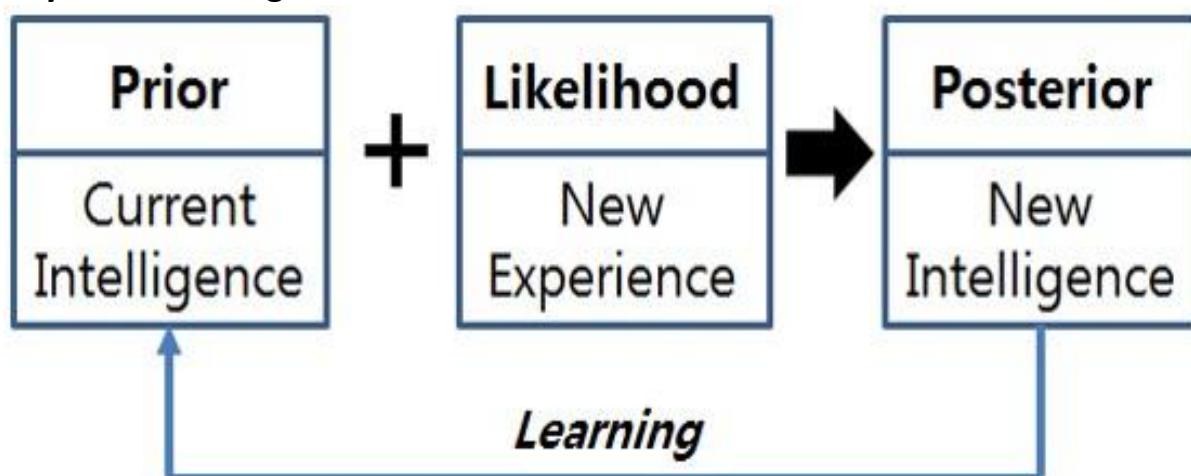
Predictions in a multilayer stacking ensemble

MODULE V:

Bayesian Learning

Bayesian Machine Learning (BML) encompasses a suite of techniques and algorithms that leverage Bayesian principles to model uncertainty in data. These methods are not just theoretical constructs; they are practical tools that have transformed the way machines learn from data.

Bayesian Learning Model:



Features of Bayesian learning

- ❖ Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct
- ❖ Prior knowledge can be combined with observed data to determine the final probability of a hypothesis
- ❖ Accommodate hypothesis that make probabilistic predictions
- ❖ Classify based on combining predictions of multiple hypothesis
- ❖ Provides optimal decision making

Bayes theorem in Machine Learning

Bayes' theorem is fundamental in machine learning, especially in the context of Bayesian inference. It provides a way to update our beliefs about a hypothesis based on new evidence. Formula:

$$P(h | D) = \frac{P(D | h) P(h)}{P(D)}$$

Where,

$P(h)$ is prior probability of hypothesis h

$P(D)$ is prior probability of training data D

$P(h|D)$ is posterior probability of h given D

$P(D|h)$ is posterior probability of D given h

$P(h|D)$ increases with increase in $P(D|h)$ and $P(h)$

$P(h|D)$ decreases with increase in $P(D)$

MAP Hypothesis

Maximum a Posteriori (MAP) estimation is a statistical technique used to estimate the probability distribution of a dataset by incorporating prior knowledge or experience. It is an extension of the maximum likelihood estimation (MLE) method, which estimates parameters of a statistical model by maximizing the likelihood function, without considering any prior distribution of the parameters.

In contrast, MAP estimation takes into account the prior distribution of the parameters, which reflects any existing beliefs or information about the parameters before observing the current data. This prior knowledge is combined with the likelihood of the observed data to produce the posterior distribution, which represents the updated beliefs about the parameters after taking the data into account.

MAP working:

$$\begin{aligned} h_{MAP} &\equiv \operatorname{argmax}_{h \in H} P(h|D) \\ &= \operatorname{argmax}_{h \in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \operatorname{argmax}_{h \in H} P(D|h)P(h) \end{aligned}$$

Second step comes from the application of Bayes theorem and third step is the consequence of independence of $P(D)$ on h .

Maximum likelihood hypothesis

In machine learning, the likelihood is a measure of the data observations up to which it can tell us the results or the target variables value for particular data points. In simple words, as the name suggests, the likelihood is a function that tells us how likely the specific data point suits the existing data distribution. maximum likelihood represents that we are maximizing the likelihood function, called the **Maximization of the Likelihood Function**. It is given by:

$$h_{ML} \equiv \operatorname{argmax}_{h \in H} P(D|h)$$

Problem:

Sample Space for events A and B

<i>A holds</i>	T	T	F	F	T	F	T
<i>B holds</i>	T	F	T	F	T	F	F

$$P(A) = 4/7 \quad P(B) = 3/7 \quad P(B|A) = 2/4 \quad P(A|B) = 2/3$$

Is Bayes Theorem correct?

Solution:

As per Bayes theorem :

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} = \frac{\frac{2}{4} * \frac{4}{7}}{\frac{3}{7}} = 2/3$$

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} = \frac{\frac{2}{4} * \frac{3}{7}}{\frac{4}{7}} = 2/4$$

Hence Bayes theorem is verified as correct for this example

Problem:

$$P(\text{cancer}) = .008, \quad P(\neg\text{cancer}) = .992$$

$$P(\oplus|\text{cancer}) = .98, \quad P(\ominus|\text{cancer}) = .02$$

$$P(\oplus|\neg\text{cancer}) = .03, \quad P(\ominus|\neg\text{cancer}) = .97$$

Given,

we now observe a new patient for whom the lab test returns a positive result. Should we diagnose the patient as having cancer or not?

Solution:

$$\begin{aligned} P(\text{cancer}|\oplus) &= P(\oplus|\text{cancer}) * P(\text{cancer}) \\ &= 0.98 * 0.008 = 0.0078 \end{aligned}$$

$$\begin{aligned} P(\neg\text{cancer}|\oplus) &= P(\oplus|\neg\text{cancer}) * P(\neg\text{cancer}) \\ &= 0.03 * 0.992 = 0.0298 \end{aligned}$$

Normalize the probabilities so that sum of probabilities=1

$$P(\text{cancer}|\oplus) = \frac{0.0078}{0.0078 + 0.0298} = 0.21$$

$$P(\neg\text{cancer}|\oplus) = \frac{0.0298}{0.0078 + 0.0298} = 0.79$$

Hence, it should be diagnosed as cancer is negative despite of a positive lab report.

BRUTE-FORCE MAP LEARNING algorithm

1. For each hypothesis h in H , calculate the posterior probability

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

2. Output the hypothesis h_{MAP} with the highest posterior probability

$$h_{MAP} = \operatorname{argmax}_{h \in H} P(h|D)$$

Brute force MAP learning algorithm consumes significant computational resources as all the hypothesis in the hypothesis set is to be checked.

Assumptions of Brute-Force MAP:

1. The training data D is noise free
2. The target concept c is contained in the hypothesis space H
3. We have no a priori reason to believe that any hypothesis is more probable than any other.

Derivations:

$$P(h) = \frac{1}{|H|} \quad \text{for all } h \text{ in } H$$

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \text{ in } D \\ 0 & \text{otherwise} \end{cases}$$

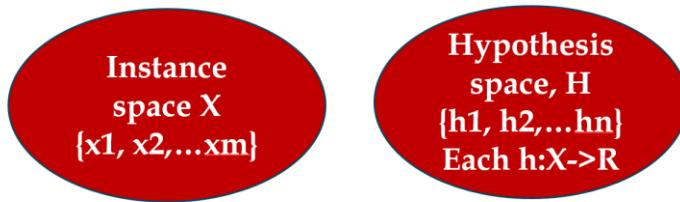
$$P(h|D) = \frac{0 \cdot P(h)}{P(D)} = 0 \text{ if } h \text{ is inconsistent with } D$$

$$\begin{aligned}
 P(h|D) &= \frac{1 \cdot \frac{1}{|H|}}{P(D)} \\
 &= \frac{1 \cdot \frac{1}{|H|}}{\frac{|VS_{H,D}|}{|H|}} \\
 &= \frac{1}{|VS_{H,D}|} \text{ if } h \text{ is consistent with } D
 \end{aligned}$$

Maximum Likelihood and Least Squared Error hypothesis

Any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis

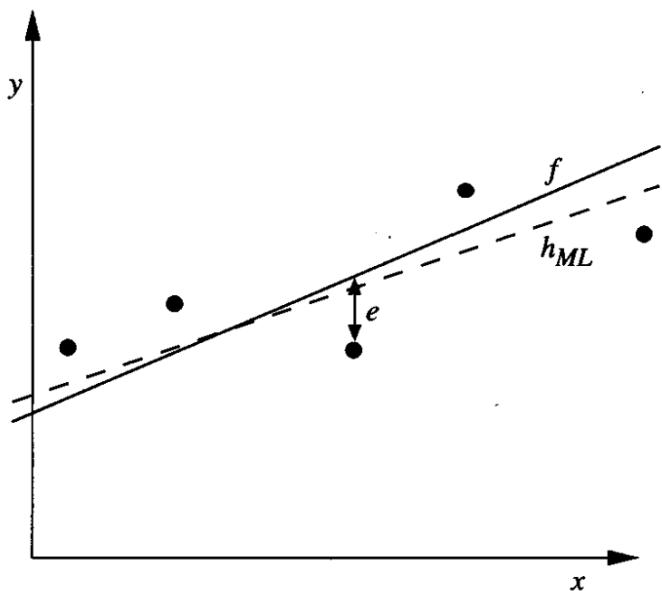
Consider:



Training instance
 $\langle x_i, d_i \rangle$
 $d_i = f(x_i) + e_i$
 $f(x_i)$ -noise free target
 e_i -random variable for noise

It is assumed that noise distribution is normal.

Learning of a real valued function has an example graph as depicted below:



Since continuous values are used, probability density function is used.

Probability density function:

$$p(x_0) \equiv \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

Derivation of h_{ML} :

$$h_{ML} = \operatorname{argmax}_{h \in H} p(D|h)$$

Probability density is the product of probability of individual instances

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m p(d_i|h)$$

The probability is computed as normal distribution

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2}$$

Applying log transformation:

$$= \operatorname{argmax}_{h \in H} \sum_{i=1}^m \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

Taking out independent term

$$= \operatorname{argmax}_{h \in H} \sum_{i=1}^m -\frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

Maximum of negatives is same as minimum of positives

$$= \operatorname{argmin}_{h \in H} \sum_{i=1}^m \frac{1}{2\sigma^2} (d_i - h(x_i))^2$$

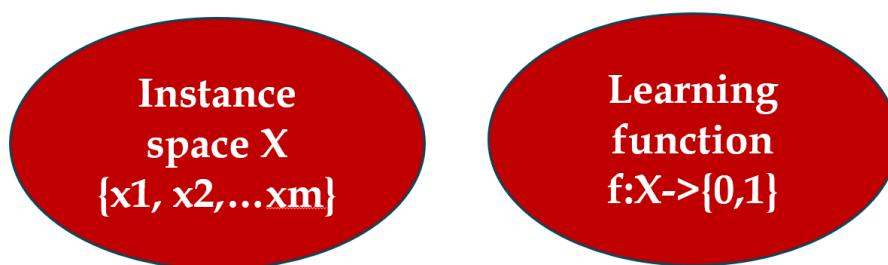
Taking out independent term:

$$h_{ML} = \operatorname{argmin}_{h \in H} \sum_{i=1}^m (d_i - h(x_i))^2$$

Thus, Any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis

Maximum Likelihood for predicting probability (Discrete case)

Consider:



Need to find $f':X \rightarrow [0, 1]$ such that $f'(x) = P(f(x) = 1)$

Derivations:

$$P(D|h) = \prod_{i=1}^m P(x_i, d_i | h)$$

Applying product theorem:

$$= \prod_{i=1}^m P(d_i | h, x_i) P(x_i)$$

$$P(d_i | h, x_i) = \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases}$$

Apply for all m samples:

$$= h(x_i)^{d_i} (1 - h(x_i))^{1-d_i}$$

Substitution:

$$= \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

$$h_{ML} = \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i)$$

Taking out independent term:

$$= \operatorname{argmax}_{h \in H} \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i}$$

Taking log transformation

$$h_{ML} = \operatorname{argmax}_{h \in H} \sum_{i=1}^m d_i \ln h(x_i) + (1 - d_i) \ln(1 - h(x_i))$$

Minimum length description Principle

Minimum Description Length (MDL) is a model selection principle where the shortest description of the data is the best model. MDL methods learn through a data compression perspective and are sometimes described as mathematical applications of Occam's razor. The MDL principle can be extended to other forms of inductive inference and learning, for example to estimation and sequential prediction, without explicitly identifying a single model of the data.

MDL applies in machine learning when algorithms (machines) generate descriptions. Learning occurs when an algorithm generates a shorter description of the same data set.

Deriving MDL:

$$h_{MAP} = \operatorname{argmax}_{h \in H} P(D|h)P(h)$$

$$= \operatorname{argmax}_{h \in H} \log_2 P(D|h) + \log_2 P(h)$$

$$= \operatorname{argmin}_{h \in H} -\log_2 P(D|h) - \log_2 P(h)$$

$$h_{MAP} = \operatorname{argmin}_{h \in H} -\log_2 P(D|h) - \log_2 P(h)$$

$$= \operatorname{argmin}_h L_{C_H}(h) + L_{C_{D|h}}(D|h)$$

Minimum Description Length principle: Choose h_{MDL} where

$$h_{MDL} = \operatorname{argmin}_{h \in H} L_{C_1}(h) + L_{C_2}(D|h)$$

Bayes optimal classifier

Bayes optimal classifier answers the question: what is the most probable classification of the new instance given the training data.

$$\operatorname{argmax}_{v_j \in V} \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

Problem:

$$V = \{\oplus, \ominus\}$$

$$P(h_1|D) = .4, \quad P(\ominus|h_1) = 0, \quad P(\oplus|h_1) = 1$$

$$P(h_2|D) = .3, \quad P(\ominus|h_2) = 1, \quad P(\oplus|h_2) = 0$$

$$P(h_3|D) = .3, \quad P(\ominus|h_3) = 1, \quad P(\oplus|h_3) = 0$$

Apply Bayes optimal classifier and find what is the classification result

Solution:

Computing $P(V)$

$$\begin{aligned} P(\oplus) &= [P(\oplus|h_1).P(h_1|D) + P(\oplus|h_2).P(h_2|D) + P(\oplus|h_3).P(h_3|D)] \\ &= [0.4+0+0]=0.4 \end{aligned}$$

$$\begin{aligned} P(\ominus) &= [P(\ominus|h_1).P(h_1|D) + P(\ominus|h_2).P(h_2|D) + P(\ominus|h_3).P(h_3|D)] \\ &= [0+0.3+0.3]=0.6 \end{aligned}$$

Hence, taking argmax, bayes optimal classifier results in negative class

Gibbs Algorithm:

1. Choose a hypothesis h from H at random, according to posterior probability distribution over H
2. Use h to predict the classification of the next instance
3. Less computationally complex
4. misclassification error for the Gibbs algorithm is at most twice the expected error of the Bayes optimal classifier

Naïve Bayes Classifier

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

Derivations:

$$\begin{aligned}
 v_{MAP} &= \operatorname{argmax}_{v_j \in V} P(v_j | a_1, a_2 \dots a_n) \\
 &= \operatorname{argmax}_{v_j \in V} \frac{P(a_1, a_2 \dots a_n | v_j) P(v_j)}{P(a_1, a_2 \dots a_n)} \\
 &= \operatorname{argmax}_{v_j \in V} P(a_1, a_2 \dots a_n | v_j) P(v_j)
 \end{aligned}$$

Assumption : feature attributes are independent of each other

Naive Bayes classifier:

$$v_{NB} = \operatorname{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i | v_j)$$

Problem:**Given the dataset:**

	Outlook	Temperature	Humidity	Windy	Play Golf
0	Rainy	Hot	High	False	No
1	Rainy	Hot	High	True	No
2	Overcast	Hot	High	False	Yes
3	Sunny	Mild	High	False	Yes
4	Sunny	Cool	Normal	False	Yes
5	Sunny	Cool	Normal	True	No
6	Overcast	Cool	Normal	True	Yes
7	Rainy	Mild	High	False	No
8	Rainy	Cool	Normal	False	Yes
9	Sunny	Mild	Normal	False	Yes
10	Rainy	Mild	Normal	True	Yes
11	Overcast	Mild	High	True	Yes
12	Overcast	Hot	Normal	False	Yes
13	Sunny	Mild	High	True	No

$$P(\text{PlayTennis} = \text{yes}) = 9/14$$

$$P(\text{PlayTennis} = \text{no}) = 5/14$$

$$P(\text{Wind} = \text{strong} | \text{PlayTennis} = \text{yes}) = 3/9$$

$$P(\text{Wind} = \text{strong} | \text{PlayTennis} = \text{no}) = 3/5$$

Need of m-estimate:

Conditional probabilities can be estimated directly as relative frequencies:

$$P(a_i | v_j) = \frac{n_c}{n}$$

where n is the total number of training instances with class v_j , and n_c is the number of instances with attribute a_i and class v_i . However,

Problem: this provides a poor estimate if n_c is very small.

Extreme case: if $n_c = 0$, then the whole posterior will be zero.

Solution: use the ***m*-estimate** of probabilities:

$$P(a_i|v_j) = \frac{n_c + mp}{n + m}$$

p : prior estimate of the probability

m : equivalent sample size (constant)

In the absence of other information, assume a **uniform prior**:

$$p = \frac{1}{k}$$

where k is the number of values that the attribute a_i can take.

Bayesian Belief Network

Bayesian belief network is key computer technology for dealing with probabilistic events and to solve a problem which has uncertainty. We can define a Bayesian network as: "A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph."

It is also called a Bayes network, belief network, decision network, or Bayesian model. Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.

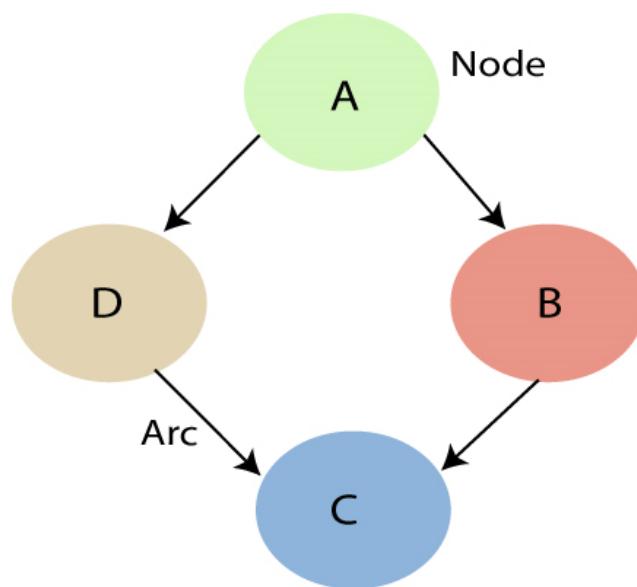
Real world applications are probabilistic in nature, and to represent the relationship between multiple events, we need a Bayesian network. It can also be used in various tasks including prediction, anomaly detection, diagnostics, automated insight, reasoning, time series prediction, and decision making under uncertainty.

Bayesian Network can be used for building models from data and experts opinions, and it consists of two parts:

- Directed Acyclic Graph
- Table of conditional probabilities.

The generalized form of Bayesian network that represents and solve decision problems under uncertain knowledge is known as an Influence diagram.

A Bayesian network graph is made up of nodes and Arcs (directed links), where:



- Each node corresponds to the random variables, and a variable can be continuous or discrete.
- Arc or directed arrows represent the causal relationship or conditional probabilities between random variables. These directed links or arrows connect the pair of nodes in the graph.

The Bayesian network has mainly two components:

- Causal Component
- Actual numbers

Each node in the Bayesian network has condition probability distribution $P(X_i | \text{Parent}(X_i))$, which determines the effect of the parent on that node.

Example:

Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbors David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm. Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

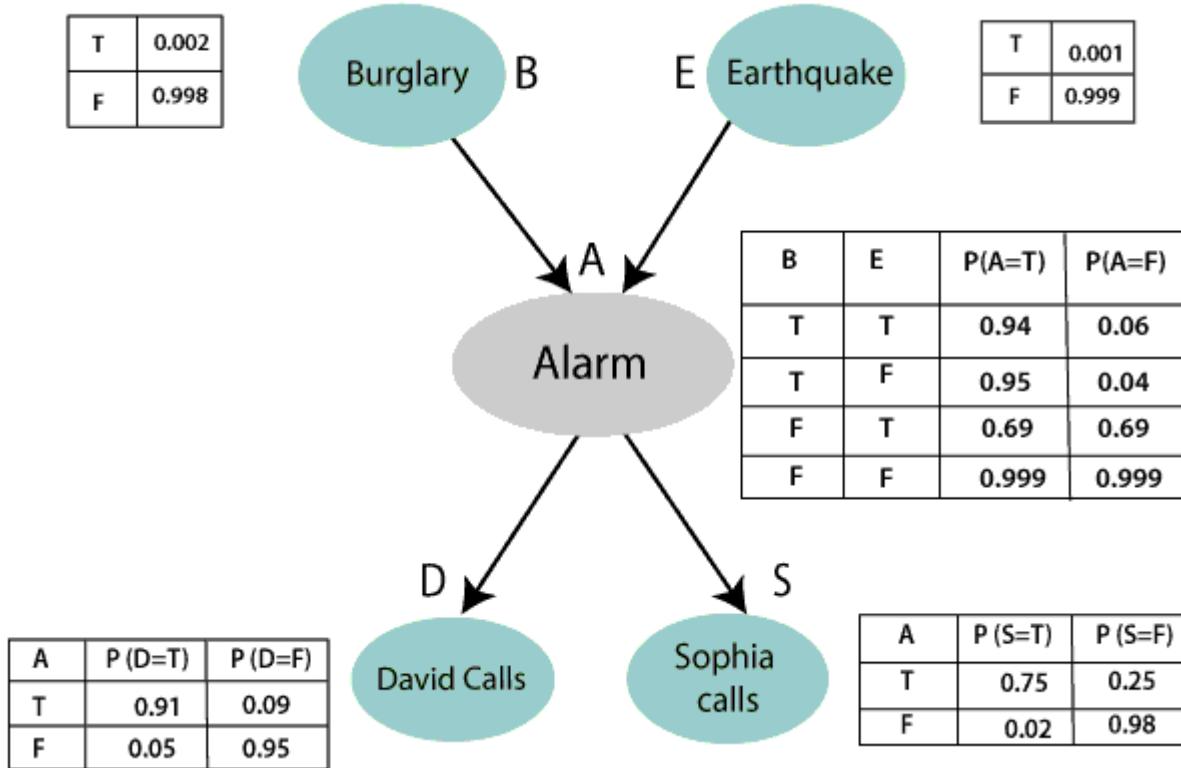
Solution:

List of all events occurring in this network:

- Burglary (B)
- Earthquake(E)
- Alarm(A)
- David Calls(D)
- Sophia calls(S)

$$\begin{aligned} P[D, S, A, B, E] &= P[D | S, A, B, E] \cdot P[S, A, B, E] \\ &= P[D | S, A, B, E] \cdot P[S | A, B, E] \cdot P[A, B, E] \end{aligned}$$

$$\begin{aligned}
 &= P[D | A] \cdot P[S | A, B, E] \cdot P[A, B, E] \\
 &= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B, E] \\
 &= P[D | A] \cdot P[S | A] \cdot P[A | B, E] \cdot P[B | E] \cdot P[E]
 \end{aligned}$$



$P(B = \text{True}) = 0.002$, which is the probability of burglary.

$P(B = \text{False}) = 0.998$, which is the probability of no burglary.

$P(E = \text{True}) = 0.001$, which is the probability of a minor earthquake

$P(E = \text{False}) = 0.999$, Which is the probability that an earthquake not occurred.

We can provide the conditional probabilities as per the below tables:

The Conditional probability of Alarm A depends on Burglar and earthquake:

B	E	P(A=True)	P(A=False)
True	True	0.94	0.06
True	False	0.95	0.04
False	True	0.31	0.69
False	False	0.001	0.999

Conditional probability table for David Calls:

The Conditional probability of David that he will call depends on the probability of Alarm.

A	P(D=True)	P(D=False)

True	0.91	0.09
False	0.05	0.95

Conditional probability table for Sophia Calls:

The Conditional probability of Sophia that she calls is depending on its Parent Node "Alarm."

A	P(S= True)	P(S= False)
True	0.75	0.25
False	0.02	0.98

From the formula of joint distribution, we can write the problem statement in the form of probability distribution:

$$\begin{aligned}
 P(S, D, A, \neg B, \neg E) &= P(S|A) * P(D|A) * P(A|\neg B \wedge \neg E) * P(\neg B) * P(\neg E) \\
 &= 0.75 * 0.91 * 0.001 * 0.998 * 0.999 \\
 &= 0.00068045.
 \end{aligned}$$

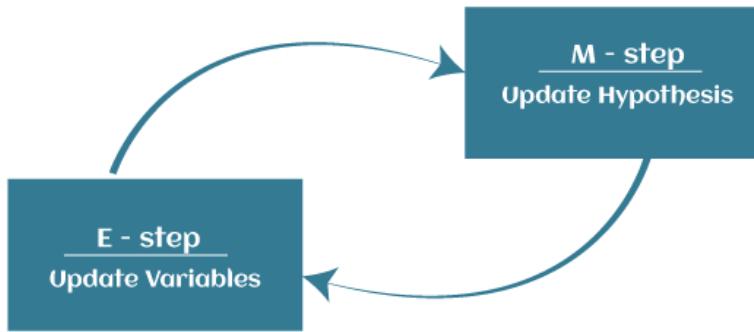
Hence, a Bayesian network can answer any query about the domain by using Joint distribution.

EM Algorithm in Machine Learning

The EM algorithm is considered a latent variable model to find the local maximum likelihood parameters of a statistical model. The EM (Expectation-Maximization) algorithm is one of the most commonly used terms in machine learning to obtain maximum likelihood estimates of variables that are sometimes observable and sometimes not. However, it is also applicable to unobserved data or sometimes called latent. It has various real-world applications in statistics, including obtaining the mode of the posterior marginal distribution of parameters in machine learning and data mining applications.

The Expectation-Maximization (EM) algorithm is defined as the combination of various unsupervised machine learning algorithms, which is used to determine the local maximum likelihood estimates (MLE) or maximum a posteriori estimates (MAP) for unobservable variables in statistical models. Further, it is a technique to find maximum likelihood estimation when the latent variables are present. It is also referred to as the latent variable model.

The EM algorithm is the combination of various unsupervised ML algorithms, such as the k-means clustering algorithm. Being an iterative approach, it consists of two modes. In the first mode, we estimate the missing or latent variables. Hence it is referred to as the Expectation/estimation step (E-step). Further, the other mode is used to optimize the parameters of the models so that it can explain the data more clearly. The second mode is known as the maximization-step or M-step.

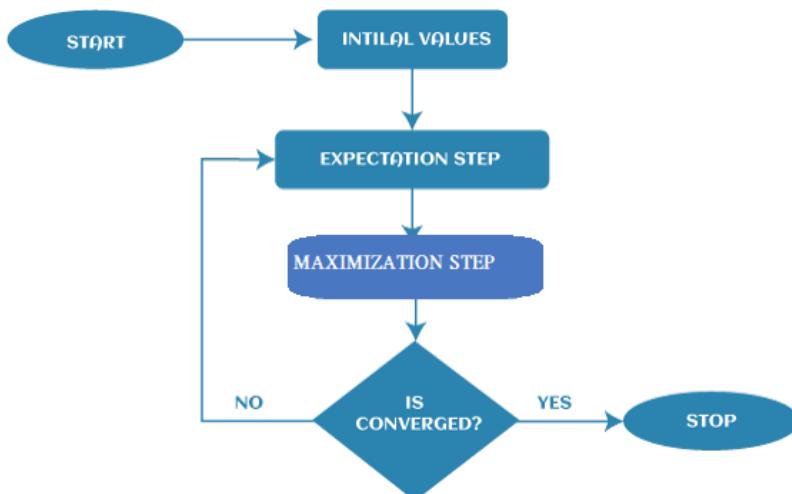


- Expectation step (E - step): It involves the estimation (guess) of all missing values in the dataset so that after completing this step, there should not be any missing value.
- Maximization step (M - step): This step involves the use of estimated data in the E-step and updating the parameters.
- Repeat E-step and M-step until the convergence of the values occurs.

The primary goal of the EM algorithm is to use the available observed data of the dataset to estimate the missing data of the latent variables and then use that data to update the values of the parameters in the M-step.

Steps in EM Algorithm

The EM algorithm is completed mainly in 4 steps, which include Initialization Step, Expectation Step, Maximization Step, and convergence Step. These steps are explained as follows:



Applications of EM algorithm

The primary aim of the EM algorithm is to estimate the missing data in the latent variables through observed data in datasets. The EM algorithm or latent variable model has a broad range of real-life applications in machine learning. These are as follows:

- The EM algorithm is applicable in data clustering in machine learning.
- It is often used in computer vision and NLP (Natural language processing).

- It is used to estimate the value of the parameter in mixed models such as the Gaussian Mixture Model and quantitative genetics.

Additional Problems:

1. Apply candidate elimination for following dataset: Candidate Elimination Algorithm):

Example	Citations	Size	InLibrary	Price	Editions	Buy
1	Some	Small	No	Affordable	One	No
2	Many	Big	No	Expensive	Many	Yes
3	Many	Medium	No	Expensive	Few	Yes
4	Many	Small	No	Affordable	Many	Yes

Solution:

S0: (0, 0, 0, 0, 0) Most Specific Boundary

G0: (?, ?, ?, ?, ?, ?) Most Generic Boundary

The first example is negative, the hypothesis at the specific boundary is consistent, hence we retain it, and the hypothesis at the generic boundary is inconsistent hence we write all consistent hypotheses by removing one "?" at a time.

S1: (0, 0, 0, 0, 0)

G1: (Many, ?, ?, ?, ?) (?, Big, ?, ?, ?) (?, Medium, ?, ?, ?) (?, ?, Exp, ?) (?, ?, ?, One) (?, ?, ?, Few)

The second example is positive, the hypothesis at the specific boundary is inconsistent, hence we extend the specific boundary, and the consistent hypothesis at the generic boundary is retained and inconsistent hypotheses are removed from the generic boundary.

S2: (Many, Big, No, Exp, Many)

G2: (Many, ?, ?, ?, ?) (?, Big, ?, ?, ?) (?, ?, Exp, ?) (?, ?, ?, Many)

The third example is positive, the hypothesis at the specific boundary is inconsistent, hence we extend the specific boundary, and the consistent hypothesis at the generic boundary is retained and inconsistent hypotheses are removed from the generic boundary.

S3: (Many, ?, No, Exp, ?)

G3: (Many, ?, ?, ?, ?) (?, ?, Exp, ?)

The fourth example is positive, the hypothesis at the specific boundary is inconsistent, hence we extend the specific boundary, and the consistent hypothesis at the generic boundary is retained and inconsistent hypotheses are removed from the generic boundary.

S4: (Many, ?, No, ?, ?)

G4: (Many, ?, ?, ?, ?)

Learned Version Space by Candidate Elimination Algorithm for given data set is:

(Many, ?, No, ?, ?) (Many, ?, ?, ?, ?)

2. Problem on Version Space – Candidate Elimination. Learning the concept of "Japanese Economy Car"

Features: (Country of Origin, Manufacturer, Color, Decade, Type)

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive

Solution:

Step 1: Positive Example: (Japan, Honda, Blue, 1980, Economy)

$$G = \{ (?, ?, ?, ?, ?) \}$$

$$S = \{ (\text{Japan}, \text{Honda}, \text{Blue}, 1980, \text{Economy}) \}$$

Step 2: Negative Example: (Japan, Toyota, Green, 1970, Sports)

G =	{ (?, Honda, ?, ?, ?), (?, ?, Blue, ?, ?), (?, ?, ?, 1980, ?), (?, ?, ?, ?, Economy) }
S =	{ (\text{Japan}, \text{Honda}, \text{Blue}, 1980, \text{Economy}) }

Step 3: Positive Example: (Japan, Toyota, Blue, 1990, Economy)

G =	{ (?, ?, Blue, ?, ?, ?), (?, ?, ?, ?, Economy) }
S =	{ (\text{Japan}, ?, \text{Blue}, ?, \text{Economy}) }

Step 4: Negative Example: (USA, Chrysler, Red, 1980, Economy)

G =	{ (?, ?, Blue, ?, ?, ?), (?, ?, ?, ?, Economy) }
S =	{ (\text{Japan}, ?, \text{Blue}, ?, \text{Economy}) }

Step 5: Positive Example: (Japan, Honda, White, 1980, Economy)

$$G = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

$$S = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

3. In Orange County, 51% of the adults are males. (It doesn't take too much advanced mathematics to deduce that the other 49% are females.) One adult is randomly selected for a survey involving credit card usage.

a) Find the prior probability that the selected person is a male.

b) It is later learned that the selected survey subject was smoking a cigar. Also, 9.5% of males smoke cigars, whereas 1.7% of females smoke cigars (based on data from the Substance Abuse and Mental Health Services Administration). Use this additional information to find the probability that the selected subject is a male.

Solution

Let's use the following notation:

$$\begin{array}{ll} M = \text{male} & \bar{M} = \text{female (or not male)} \\ C = \text{cigar smoker} & \bar{C} = \text{not a cigar smoker.} \end{array}$$

- a. Before using the information given in part b, we know only that 51% of the adults in Orange County are males, so the probability of randomly selecting an adult and getting a male is given by $P(M) = 0.51$.
- b. Based on the additional given information, we have the following:

$$P(M) = 0.51 \quad \text{because 51\% of the adults are males}$$

$$P(\bar{M}) = 0.49 \quad \text{because 49\% of the adults are females (not males)}$$

$$P(C|M) = 0.095 \quad \text{because 9.5\% of the males smoke cigars (That is, the probability of getting someone who smokes cigars, given that the person is a male, is 0.095.)}$$

$$P(C|\bar{M}) = 0.017. \quad \text{because 1.7\% of the females smoke cigars (That is, the probability of getting someone who smokes cigars, given that the person is a female, is 0.017.)}$$

Let's now apply Bayes' theorem by using the preceding formula with M in place of A, and C in place of B. We get the following result:

$$\begin{aligned} P(M|C) &= \frac{P(M) \cdot P(C|M)}{[P(M) \cdot P(C|M)] + [P(\bar{M}) \cdot P(C|\bar{M})]} \\ &= \frac{0.51 \cdot 0.095}{[0.51 \cdot 0.095] + [0.49 \cdot 0.017]} \\ &= 0.85329341 \\ &= 0.853 \text{ (rounded)} \end{aligned}$$

4. Apply Naïve Bayes Classifier for the following dataset:

Sl. No.	Color	Legs	Height	Smelly	Species
1	White	3	Short	Yes	M
2	Green	2	Tall	No	M
3	Green	3	Short	Yes	M
4	White	3	Short	Yes	M
5	Green	2	Short	No	H
6	White	2	Tall	No	H
7	White	2	Tall	No	H
8	White	2	Short	Yes	H

Solution:

Using the above data, we have to identify the species of an entity with the following attributes.

$$X = \{\text{Color=Green, Legs=2, Height=Tall, Smelly=No}\}$$

To predict the class label for the above attribute set, we will first calculate the probability of the species being M or H in total.

$$P(\text{Species}=M) = 4/8 = 0.5$$

$$P(\text{Species}=H) = 4/8 = 0.5$$

Next, we will calculate the conditional probability of each attribute value for each class label.

$$P(\text{Color=White}/\text{Species}=M) = 2/4 = 0.5$$

$$P(\text{Color=White}/\text{Species}=H) = 3/4 = 0.75$$

$$P(\text{Color=Green}/\text{Species}=M) = 2/4 = 0.5$$

$$P(\text{Color=Green}/\text{Species}=H) = 1/4 = 0.25$$

$$P(\text{Legs}=2/\text{Species}=M) = 1/4 = 0.25$$

$$P(\text{Legs}=2/\text{Species}=H) = 4/4 = 1$$

$$P(\text{Legs}=3/\text{Species}=M) = 3/4 = 0.75$$

$$P(\text{Legs}=3/\text{Species}=H) = 0/4 = 0$$

$$P(\text{Height=Tall}/\text{Species}=M) = 3/4 = 0.75$$

$$P(\text{Height=Tall}/\text{Species}=H) = 2/4 = 0.5$$

$$P(\text{Height=Short}/\text{Species}=M) = 1/4 = 0.25$$

$$P(\text{Height=Short}/\text{Species}=H) = 2/4 = 0.5$$

$$P(\text{Smelly=Yes}/\text{Species}=M) = 3/4 = 0.75$$

$$P(\text{Smelly=Yes}/\text{Species}=H) = 1/4 = 0.25$$

$$P(\text{Smelly=No}/\text{Species}=M) = 1/4 = 0.25$$

$$P(\text{Smelly=No}/\text{Species}=H) = 3/4 = 0.75$$

We can tabulate the above calculations in the tables for better visualization.

The conditional probability table for the Color attribute is as follows.

Color	M	H
White	0.5	0.75
Green	0.5	0.25

Conditional Probabilities for Color Attribute

The conditional probability table for the Legs attribute is as follows.

Legs	M	H
2	0.25	1
3	0.75	0

Conditional Probabilities for Legs Attribute

The conditional probability table for the Height attribute is as follows.

Height	M	H
Tall	0.75	0.5
Short	0.25	0.5

Conditional Probabilities for Height Attribute

The conditional probability table for the Smelly attribute is as follows.

Smelly	M	H
Yes	0.75	0.25
No	0.25	0.75

Conditional Probabilities for Smelly Attribute

Now that we have calculated the conditional probabilities, we will use them to calculate the probability of the new attribute set belonging to a single class.

Let us consider $X = \{\text{Color}=\text{Green}, \text{Legs}=2, \text{Height}=\text{Tall}, \text{Smelly}=\text{No}\}$.

Then, the probability of X belonging to Species M will be as follows.

$$\begin{aligned} P(M/X) &= P(\text{Species}=M) * P(\text{Color}=\text{Green}/\text{Species}=M) * P(\text{Legs}=2/\text{Species}=M) * P(\text{Height}=\text{Tall}/\text{Species}=M) * P(\text{Smelly}=\text{No}/\text{Species}=M) \\ &= 0.5 * 0.5 * 0.25 * 0.75 * 0.25 \\ &= 0.0117 \end{aligned}$$

Similarly, the probability of X belonging to Species H will be calculated as follows.

$$\begin{aligned} P(H/X) &= P(\text{Species}=H) * P(\text{Color}=\text{Green}/\text{Species}=H) * P(\text{Legs}=2/\text{Species}=H) * P(\text{Height}=\text{Tall}/\text{Species}=H) * P(\text{Smelly}=\text{No}/\text{Species}=H) \\ &= 0.5 * 0.25 * 1 * 0.5 * 0.75 \\ &= 0.0468 \end{aligned}$$

So, the probability of X belonging to Species M is 0.0117 and that to Species H is 0.0468. Hence, we will assign the entity X with attributes $\{\text{Color}=\text{Green}, \text{Legs}=2, \text{Height}=\text{Tall}, \text{Smelly}=\text{No}\}$ to species H.

5. Build a decision tree using ID3 algorithm for the given training data in the table (Buy Computer data), and predict the class of the following new example: age<=30, income=medium, student=yes, credit-rating=fair

age	income	student	Credit rating	Buys computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31...40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31...40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31...40	medium	no	excellent	yes

31...40	high	yes	fair	yes
>40	medium	no	excellent	no

Solution:

The information gain is this mutual information minus the entropy:

The mutual information of the two classes,

$$\text{Entropy}(S) = E(9,5) = -9/14 \log_2(9/14) - 5/14 \log_2(5/14) = 0.94$$

Now Consider the Age attribute

For Age, we have three values age ≤ 30 (2 yes and 3 no), age $31..40$ (4 yes and 0 no), and age >40 (3 yes and 2 no)

$$\begin{aligned} \text{Entropy(age)} &= 5/14 (-2/5 \log_2(2/5) - 3/5 \log_2(3/5)) + 4/14 (0) + 5/14 (-3/5 \log_2(3/5) - 2/5 \log_2(2/5)) \\ &= 5/14(0.9709) + 0 + 5/14(0.9709) = 0.6935 \end{aligned}$$

$$\text{Gain(age)} = 0.94 - 0.6935 = 0.2465$$

Next, consider Income Attribute

For Income, we have three values income_{high} (2 yes and 2 no), income_{medium} (4 yes and 2 no), and income_{low} (3 yes 1 no)

$$\begin{aligned} \text{Entropy(income)} &= 4/14(-2/4 \log_2(2/4) - 2/4 \log_2(2/4)) + 6/14 (-4/6 \log_2(4/6) - 2/6 \log_2(2/6)) + 4/14 (-3/4 \log_2(3/4) - 1/4 \log_2(1/4)) \\ &= 4/14(1) + 6/14(0.918) + 4/14(0.811) \\ &= 0.285714 + 0.393428 + 0.231714 = 0.9108 \end{aligned}$$

$$\text{Gain(income)} = 0.94 - 0.9108 = 0.0292$$

Next, consider Student Attribute

For Student, we have two values student_{yes} (6 yes and 1 no) and student_{no} (3 yes 4 no)

$$\begin{aligned} \text{Entropy(student)} &= 7/14(-6/7 \log_2(6/7) - 1/7 \log_2(1/7)) + 7/14(-3/7 \log_2(3/7) - 4/7 \log_2(4/7)) \\ &= 7/14(0.5916) + 7/14(0.9852) \\ &= 0.2958 + 0.4926 = 0.7884 \end{aligned}$$

$$\text{Gain(student)} = 0.94 - 0.7884 = 0.1516$$

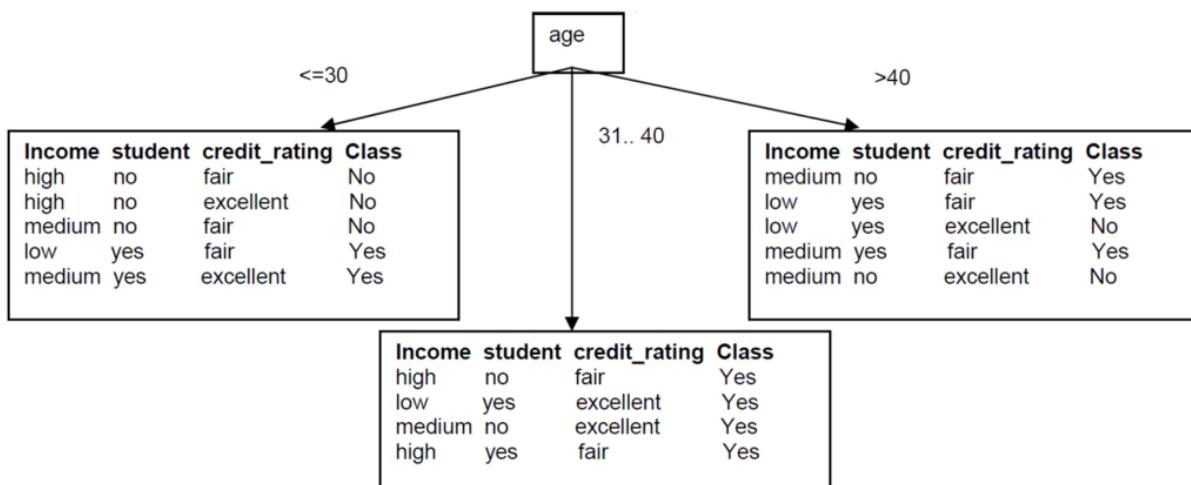
Finally, consider Credit_Rating Attribute

For Credit_Rating we have two values credit_rating_{fair} (6 yes and 2 no) and credit_rating_{excellent} (3 yes 3 no)

$$\begin{aligned} \text{Entropy(credit_rating)} &= 8/14(-6/8 \log_2(6/8) - 2/8 \log_2(2/8)) + 6/14(-3/6 \log_2(3/6) - 3/6 \log_2(3/6)) \\ &= 8/14(0.8112) + 6/14(1) \\ &= 0.4635 + 0.4285 = 0.8920 \end{aligned}$$

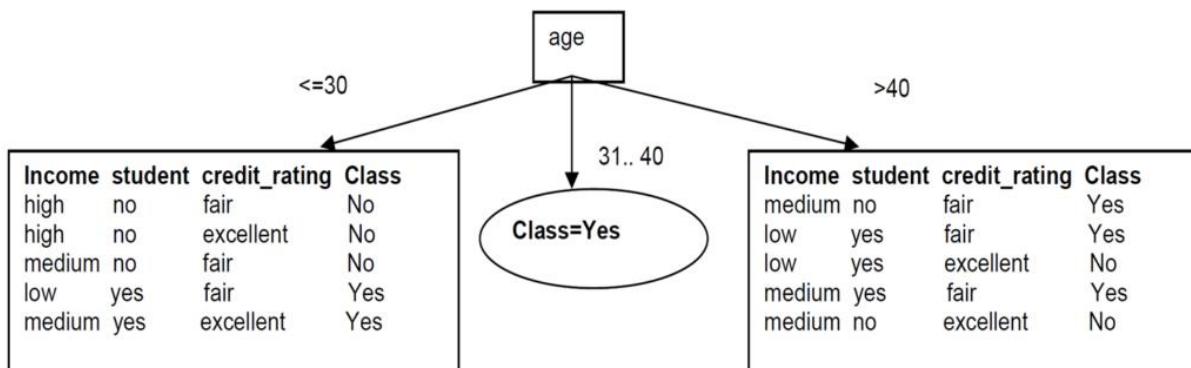
$$\text{Gain(credit_rating)} = 0.94 - 0.8920 = 0.479$$

Since Age has the highest Information Gain we start splitting the dataset using the age attribute.



Decision Tree after step 1

Since all records under the branch age31..40 are all of the class, Yes, we can replace the leaf with Class=Yes



Decision Tree after step 1_1

Now build the decision tree for the left subtree

The same process of splitting has to happen for the two remaining branches.

Income student credit_rating Class

high	no	fair	No
high	no	excellent	No
medium	no	fair	No
low	yes	fair	Yes
medium	yes	excellent	Yes

Left sub-branch

For branch age ≤ 30 we still have attributes income, student, and credit_rating. Which one should be used to split the partition?

The mutual information is $E(S_{age \leq 30}) = E(2,3) = -2/5 \log_2(2/5) - 3/5 \log_2(3/5) = 0.97$

For Income, we have three values income_{high} (0 yes and 2 no), income_{medium} (1 yes and 1 no) and income_{low} (1 yes and 0 no)

$$\text{Entropy(income)} = 2/5(0) + 2/5(-1/2\log_2(1/2)-1/2\log_2(1/2)) + 1/5(0) = 2/5(1) = 0.4$$

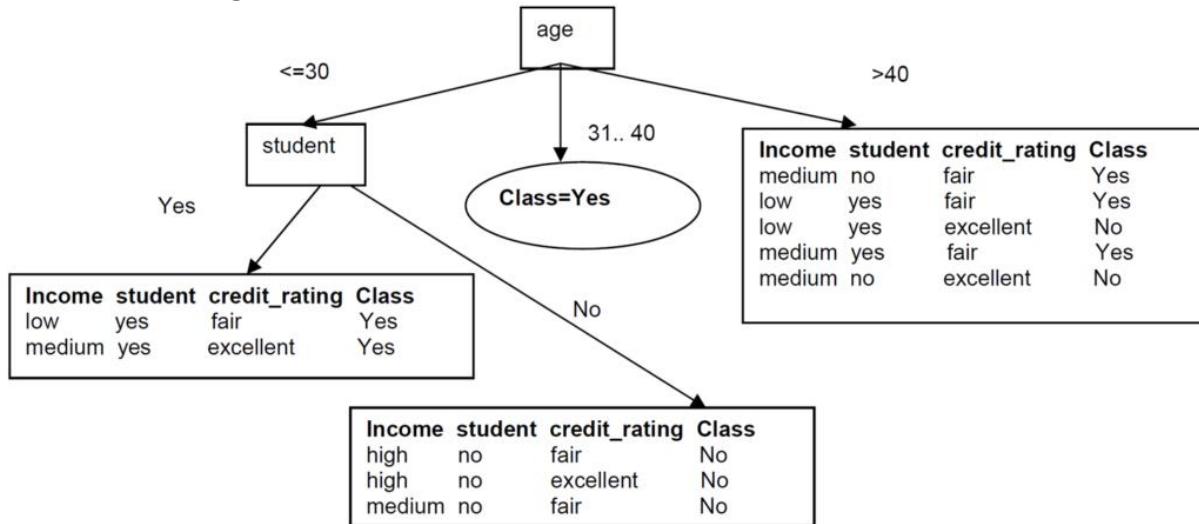
$$\text{Gain(income)} = 0.97 - 0.4 = 0.57$$

For Student, we have two values student_{yes} (2 yes and 0 no) and student_{no} (0 yes 3 no)

$$\text{Entropy}(\text{student}) = 2/5(0) + 3/5(0) = 0$$

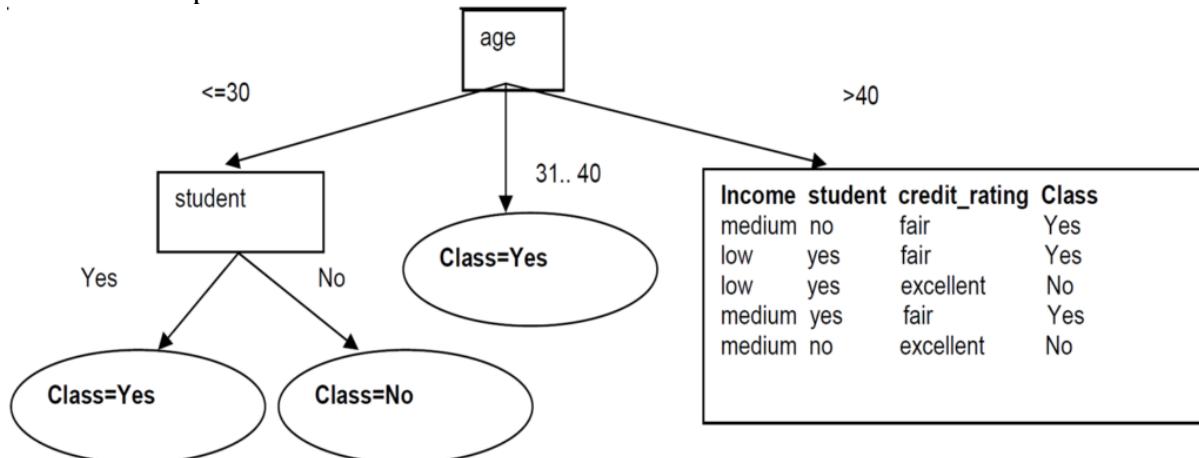
$$\text{Gain}(\text{student}) = 0.97 - 0 = 0.97$$

We can then safely split on attribute student without checking the other attributes since the information gain is maximized.



Decision Tree after step 2

Since these two new branches are from distinct classes, we make them into leaf nodes with their respective class as label:



Decision Tree after step 2_2

Now build the decision tree for right left subtree

Income	student	credit_rating	Class
medium	no	fair	Yes
low	yes	fair	Yes
low	yes	excellent	No
medium	yes	fair	Yes
medium	no	excellent	No

Right sub-branch

The mutual information is $\text{Entropy}(S_{\text{age}>40}) = I(3,2) = -3/5 \log_2(3/5) - 2/5 \log_2(2/5) = 0.97$
 For Income, we have two values $\text{income}_{\text{medium}}$ (2 yes and 1 no) and $\text{income}_{\text{low}}$ (1 yes and 1 no)

$$\text{Entropy}(\text{income}) = 3/5(-2/3\log_2(2/3)-1/3\log_2(1/3)) + 2/5(-1/2\log_2(1/2)-1/2\log_2(1/2))$$

$$= 3/5(0.9182) + 2/5(1) = 0.55 + 0.4 = 0.95$$

$$\text{Gain}(\text{income}) = 0.97 - 0.95 = 0.02$$

For Student, we have two values $\text{student}_{\text{yes}}$ (2 yes and 1 no) and $\text{student}_{\text{no}}$ (1 yes and 1 no)

$$\text{Entropy}(\text{student}) = 3/5(-2/3\log_2(2/3)-1/3\log_2(1/3)) + 2/5(-1/2\log_2(1/2)-1/2\log_2(1/2)) = 0.95$$

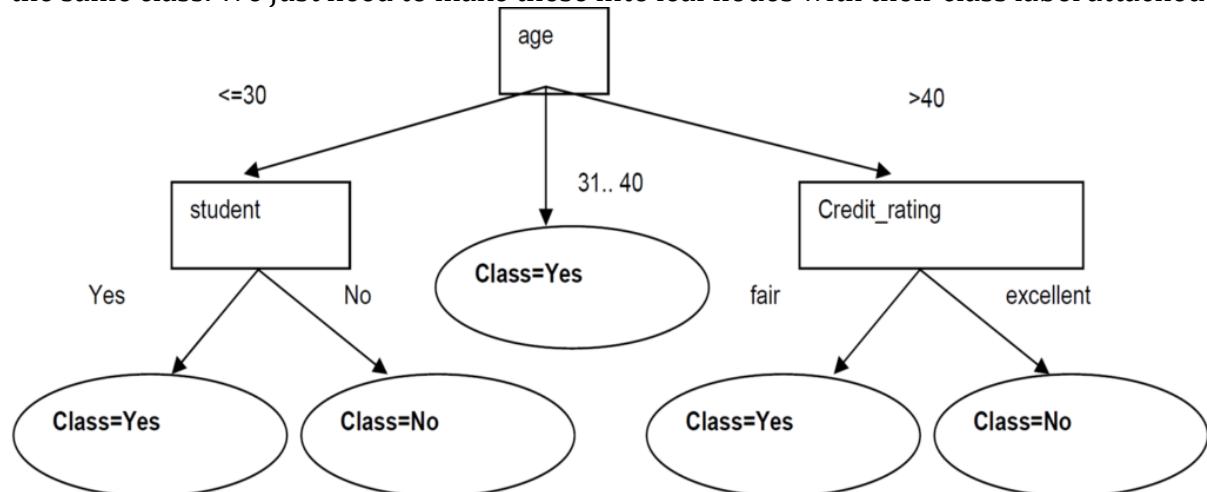
$$\text{Gain}(\text{student}) = 0.97 - 0.95 = 0.02$$

For Credit_Rating, we have two values $\text{credit_rating}_{\text{fair}}$ (3 yes and 0 no) and $\text{credit_rating}_{\text{excellent}}$ (0 yes and 2 no)

$$\text{Entropy}(\text{credit_rating}) = 0$$

$$\text{Gain}(\text{credit_rating}) = 0.97 - 0 = 0.97$$

We then split based on credit_rating. These splits give partitions each with records from the same class. We just need to make these into leaf nodes with their class label attached:



Decision Tree for Buys Computer

New example: age<=30, income=medium, student=yes, credit-rating=fair

Follow branch(age<=30) then student=yes we predict Class=yes

Buy_computer = yes

Model Paper:

Q.1	(a)	What is Machine Learning? Explain the applications of Machine Learning						04M																																																							
	(b)	Discuss the any four main challenges of machine learning						08M																																																							
	(c)	Consider the "Japanese Economy Car" concept and instance given in Table 1, Illustrate the hypothesis using Candidate Elimination Learning algorithm.						08M																																																							
<table border="1"> <thead> <tr> <th>Origin</th><th>Manufacturer</th><th>Color</th><th>Decade</th><th>Type</th><th>Example Type</th><th></th><th></th><th></th></tr> </thead> <tbody> <tr> <td>Japan</td><td>Honda</td><td>Blue</td><td>1980</td><td>Economy</td><td>Positive</td><td></td><td></td><td></td></tr> <tr> <td>Japan</td><td>Toyota</td><td>Green</td><td>1970</td><td>Sports</td><td>Negative</td><td></td><td></td><td></td></tr> <tr> <td>Japan</td><td>Toyota</td><td>Blue</td><td>1990</td><td>Economy</td><td>Positive</td><td></td><td></td><td></td></tr> <tr> <td>USA</td><td>Chrysler</td><td>Red</td><td>1980</td><td>Economy</td><td>Negative</td><td></td><td></td><td></td></tr> <tr> <td>Japan</td><td>Honda</td><td>White</td><td>1980</td><td>Economy</td><td>Positive</td><td></td><td></td><td></td></tr> </tbody> </table>	Origin							Manufacturer	Color	Decade	Type	Example Type				Japan	Honda	Blue	1980	Economy	Positive				Japan	Toyota	Green	1970	Sports	Negative				Japan	Toyota	Blue	1990	Economy	Positive				USA	Chrysler	Red	1980	Economy	Negative				Japan	Honda	White	1980	Economy	Positive						
Origin	Manufacturer	Color	Decade	Type	Example Type																																																										
Japan	Honda	Blue	1980	Economy	Positive																																																										
Japan	Toyota	Green	1970	Sports	Negative																																																										
Japan	Toyota	Blue	1990	Economy	Positive																																																										
USA	Chrysler	Red	1980	Economy	Negative																																																										
Japan	Honda	White	1980	Economy	Positive																																																										
Q.2	(a)	Explain Find-S algorithm ad show its working by taking the enjoy sport concept and training instances given in Table 2.							10M																																																						
		Exampl e	Sky	AirTem p	Humidit y	Wind	Wate r	Foreca st																																																							
		1	Sunn y	Warm	Normal	Stron g	War m	Same																																																							
		2	Sunn y	Warm	High	Stron g	War m	Same																																																							
		3	Rain y	Cold	High	Stron g	War m	Change																																																							
		4	Sunn y	Warm	High	Stron g	Cool	Change																																																							
	(b)	Discuss the features of an unbiased Learner.							06M																																																						
	(c)	State the following problems with respect to Tasks, Performance, and Experience: i) A Checkers learning problem ii) A Robot driving learning problem.							04M																																																						
Q.3	(a)	In context to prepare the data for Machine Learning algorithms, Write a note on (i) Data Cleaning (ii) Handling text and categorical attributes iii) Feature scaling							10M																																																						
	(b)	With the code snippets show how Grid Search and Randomized Search helps in Fine-Tuning a model.							10M																																																						
Q.4	(a)	Using code snippets, outline the concepts involved in i) Measuring accuracy using Cross-Validation. ii) Confusion Matrix. iii) Precision and Recall.							10M																																																						
	(b)	With the code snippet explain how Multilabels classification different from multiclassMultioutput classification?							10M																																																						
Q.5	(a)	what is gradient descent algorithm and discuss its various types.							10M																																																						

	(b)	In Regularized Linear Models illustrate the three different methods to constrain the weights.	10M
Q.6	(a)	With respect to Nonlinear SVM Classification, explain Polynomial Kernel Gaussian and RBF Kernel along with code snippet.	10M
	(b)	Show that how SVMs make predictions using Quadratic Programming and KernelizedSVM.	10 M
Q.7	(a)	With an example dataset examine how Decision Trees are used in making predictions.	10M
	(b)	Explain The CART Training Algorithm.	06M
	(c)	Identify the features of Regression and Instability w.r.t decision trees.	04M
Q.8	(a)	In context to Ensemble methods determine the concepts of i) Bagging and Pasting. Voting Classifiers.	10M
	(b)	Examine the following boosting methods along with code snippets. i) AdaBoost ii) Gradient Boosting	10M
Q.9	(a)	Write Bayes theorem. Identify the relationship between Bayes theorem and the problem of concept learning?	10M
	(b)	Show that how Maximum Likelihood Hypothesis is helpful for predicting probabilities.	10M
Q.10	(a)	Construct Naïve Bayes Classifier with an Example.	10M
	(b)	Derive the EM Algorithm in detail.	10M