Ensemble Learning

Ensemble Learning in Machine Learning

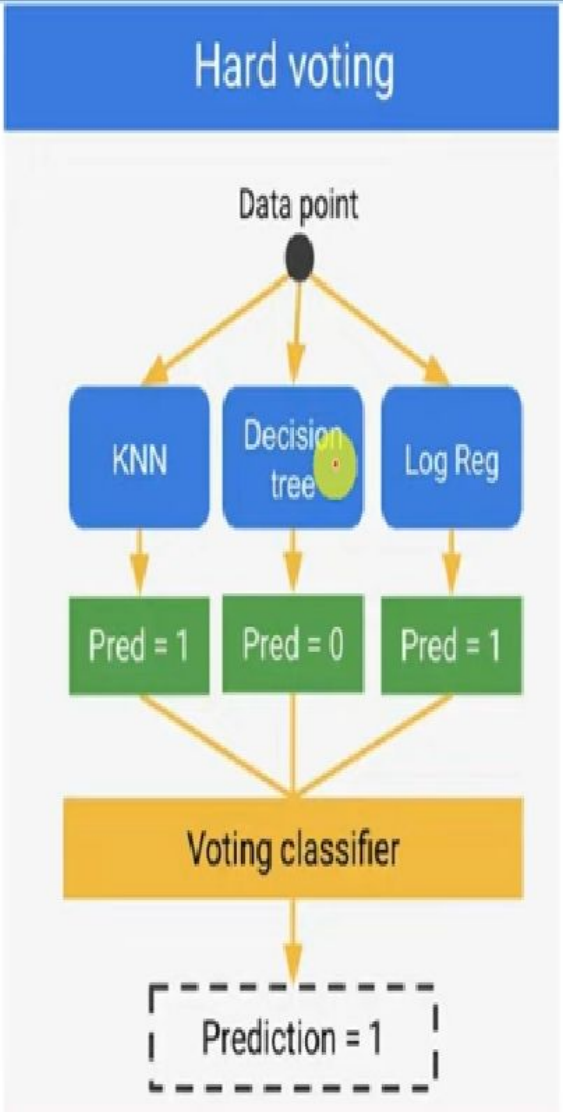
- Ensemble learning is a supervised learning technique used in machine learning to improve overall performance by combining the predictions from multiple models.

Ensemble Learning - Types of Ensemble Methods

- Voting (Averaging)
- Bootstrap aggregation (bagging)
- Random Forests
- Boosting
- Stacked Generalization (Blending)

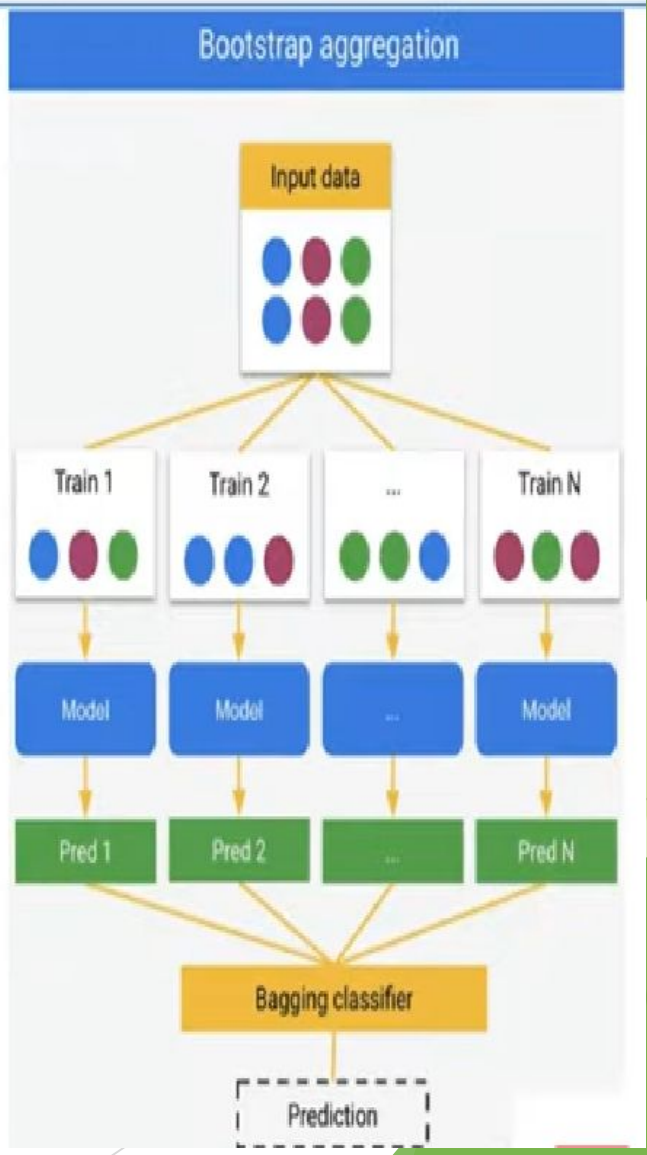
Ensemble Learning – Voting (Averaging)

- Voting is an ensemble machine learning algorithm that involves making a prediction that is the average (regression) or the sum (classification) of multiple machine learning models.



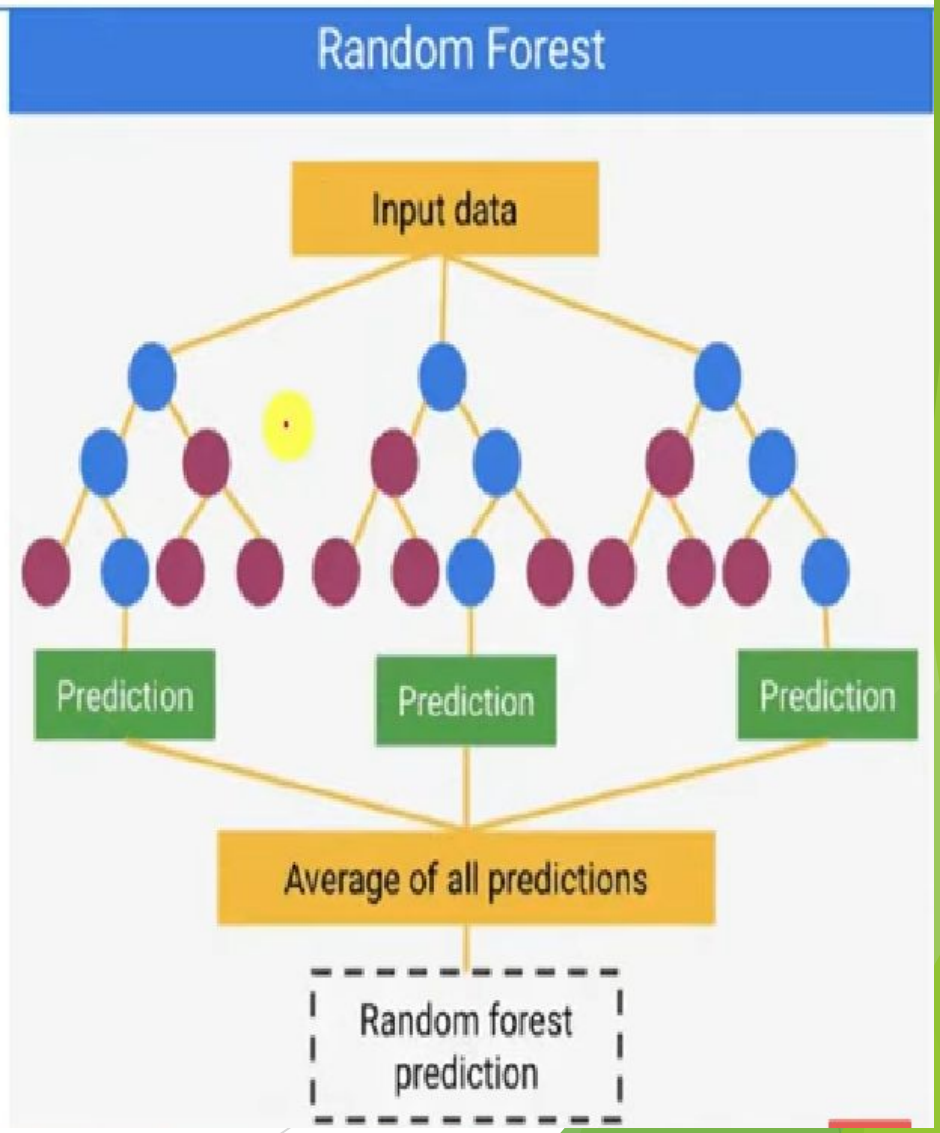
Ensemble Learning – Bootstrap aggregation (bagging)

- Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms like classification and regression.
- It decreases the variance and helps to avoid overfitting.
- It is usually applied to decision tree methods.
- Bagging is a special case of the model averaging approach.



Ensemble Learning – Random Forest

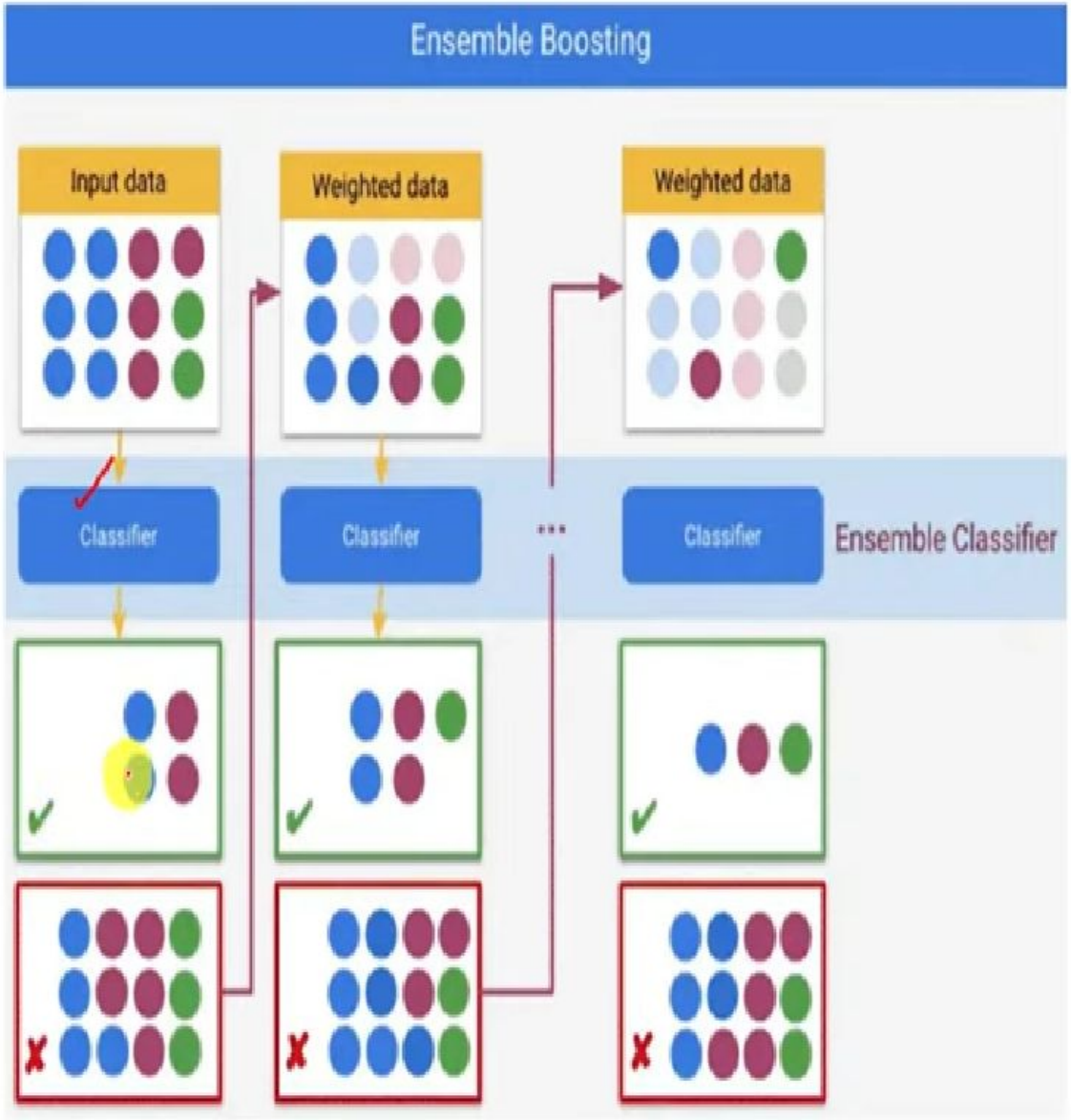
- Random forest is a commonly-used machine learning algorithm.
- A random forest is an ensemble learning method where multiple decision trees are constructed and then they are merged to get a more accurate prediction.



Ensemble Learning – 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 is added.

Ensemble Learning – Boosting

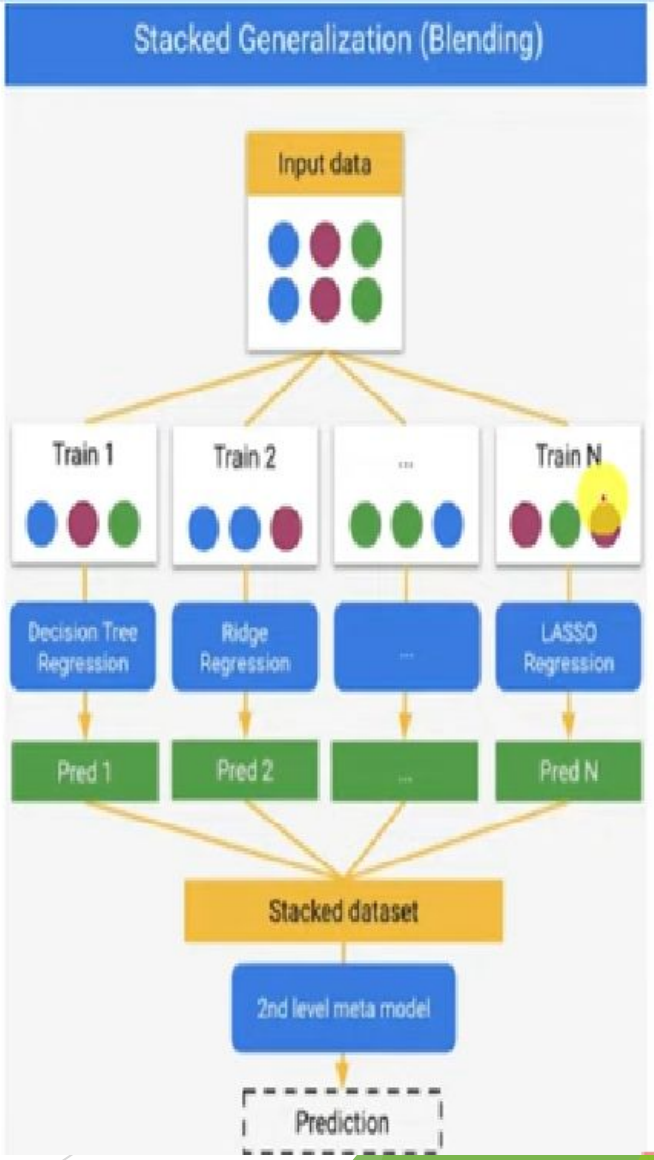


Ensemble Learning – Stacked Generalization (Blending)

- Stacking, Blending and and Stacked Generalization are all the same thing with different names. It is a kind of ensemble learning.
- In traditional ensemble learning, we have multiple classifiers trying to fit to a training set to approximate the target function.
- Since each classifier will have its own output, we will need to find a combining mechanism to combine the results.
- This can be through voting (majority wins), weighted voting (some classifier has more authority than the others), averaging the results, etc.

Ensemble Learning – Stacked Generalization (Blending)

- In stacking, the combining mechanism is that the output of the classifiers (Level 0 classifiers) will be used as training data for another classifier (Level 1 classifier) to approximate the same target function.
- Basically, you let the Level 1 classifier to figure out the combining mechanism.



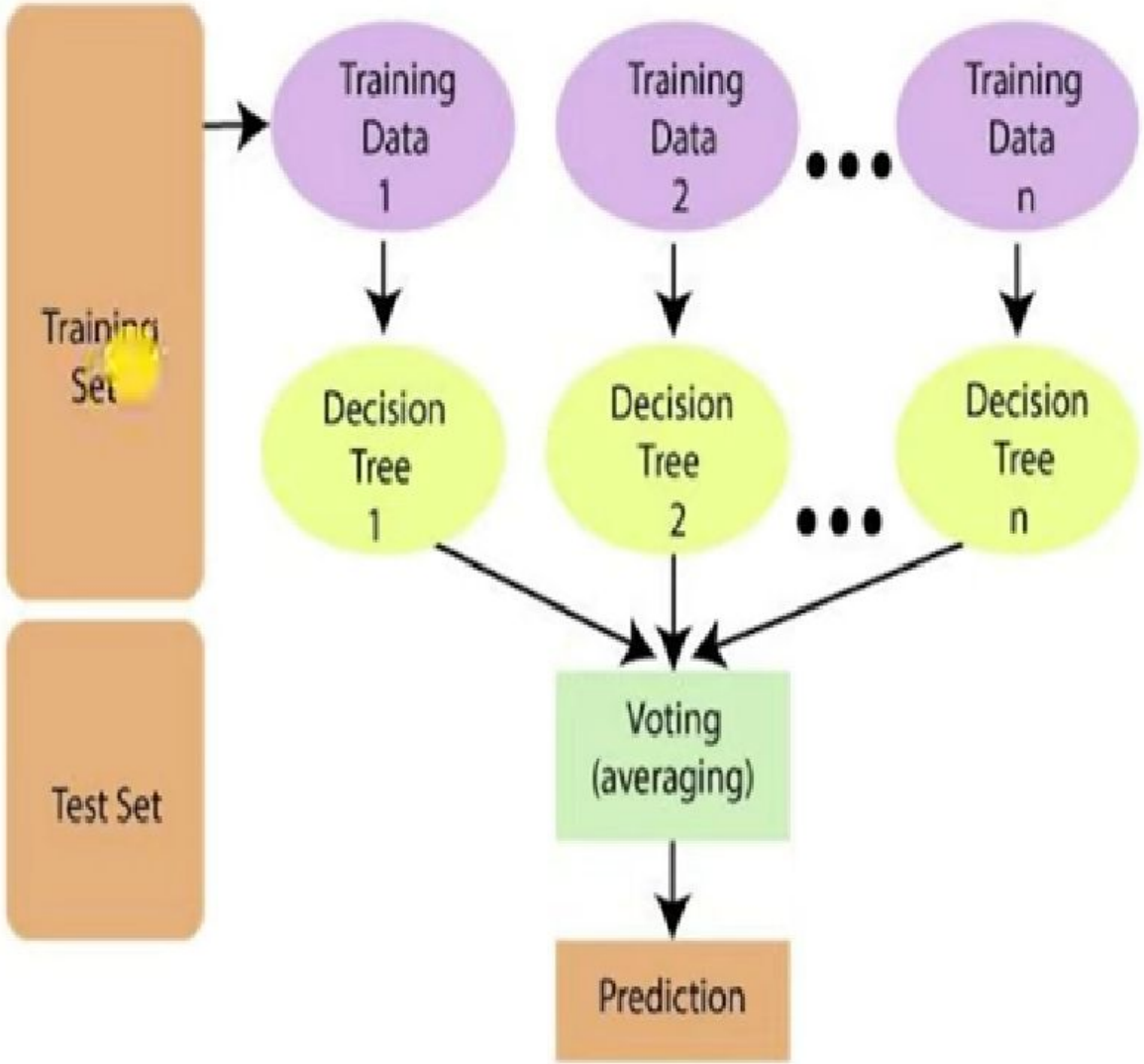
Random Forest Algorithm

- Random forest is a commonly-used machine learning algorithm.
- A random forest is an ensemble learning method where multiple decision trees are constructed and then they are merged to get a more accurate prediction.
- Random forest became popular because of its ease of use and flexibility in handling both classification and regression problems.

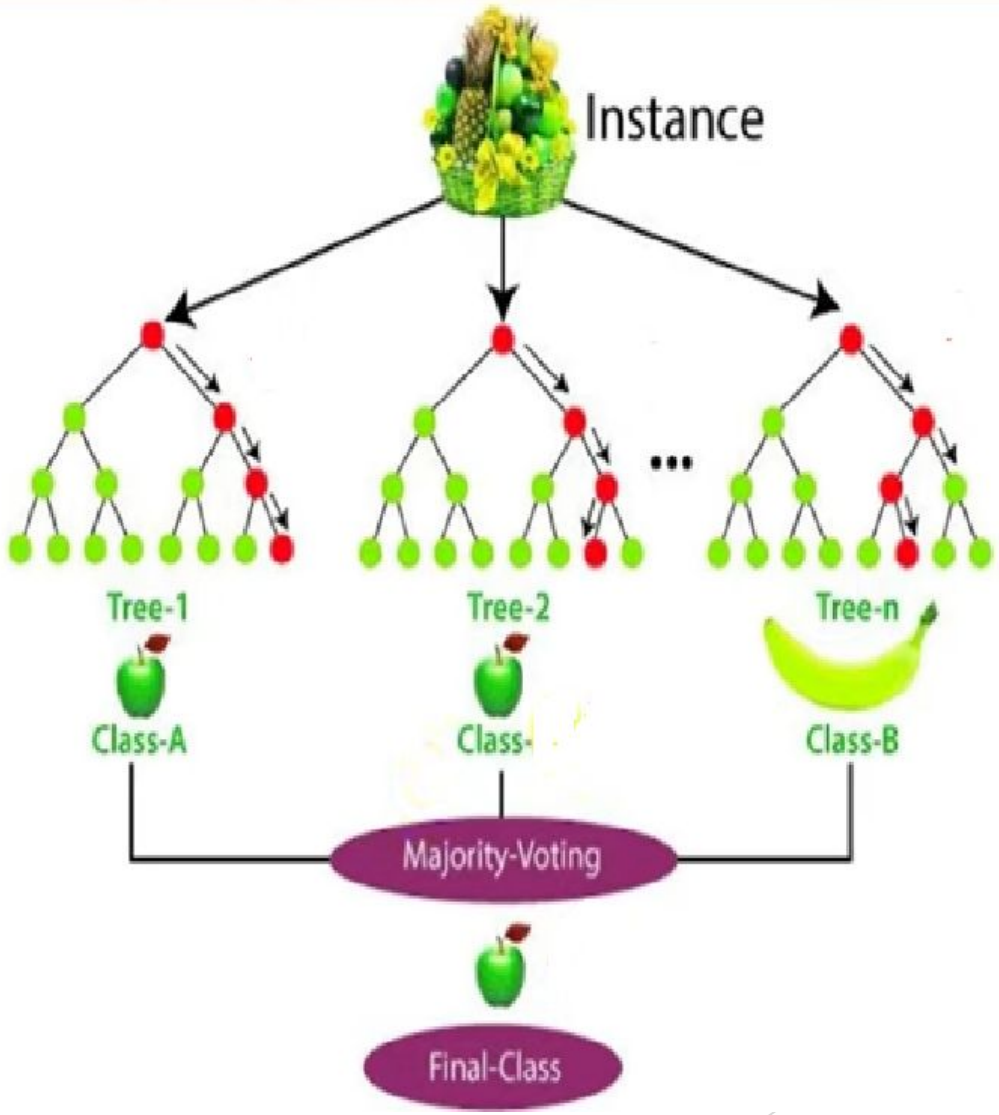
Random Forest Algorithm - Steps

1. Build random forests :
   a) If the number of examples in the training set is N, take a sample of n examples at random - but with replacement, from the original data. This sample will be the training set for generating the tree.
   b) If there are M input variables, m variables are selected at random out of the M and the best split on these m is used to split the node. The value of m is held constant during the generation of the various trees in the forest.
   c) Each tree is grown to the largest extent possible.
2. For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the **majority votes**.

Random Forest Algorithm - Steps



Random Forest Algorithm - Steps



Random Forest Algorithm - Strengths

1. It takes less training time as compared to other algorithms.
2. It predicts output with high accuracy, even for the large dataset it runs efficiently.
3. It can also maintain accuracy when a large proportion of data is missing.

Random Forest Algorithm - Weaknesses

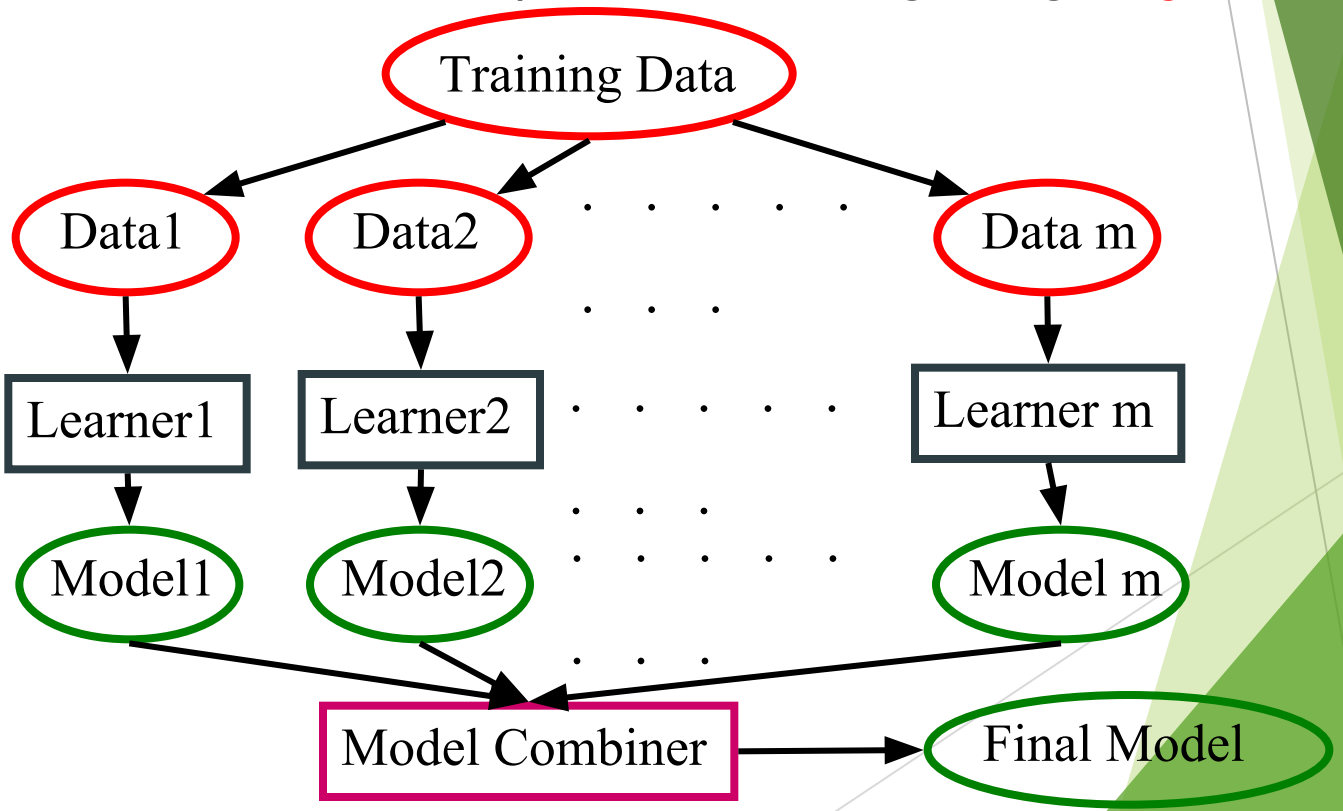
1. A weakness of random forest algorithms is that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.
2. The sizes of the models created by random forests may be very large. It may take hundreds of megabytes of memory and may be slow to evaluate.
3. Random forest models are black boxes that are very hard to interpret.

Ensemble Learning

- So far – learning methods that learn a single hypothesis, chosen from a hypothesis space that is used to make predictions.
- Ensemble learning ▫ select a collection (ensemble) of hypotheses and combine their predictions.
- Example 1 - generate 100 different decision trees from the same or different training set and have them vote on the best classification for a new example.
- Key motivation: reduce the error rate. Hope is that it will become much more unlikely that the ensemble will misclassify an example.

Learning Ensembles

- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.
- Combine decisions of multiple definitions, e.g. using weighted voting.



Value of Ensembles

- “No Free Lunch” Theorem
  - No single algorithm wins all the time!

- When combing multiple independent and diverse decisions each of which is at least more accurate than random guessing, random errors cancel each other out, correct decisions are reinforced.

Example: Weather Forecast

Reality							
1							
2							
3							
4							
5							
Combine							

Intuitions

- Majority vote
- Suppose we have 5 completely independent classifiers…
  - If accuracy is 70% for each
    - $(.7^5)+5(.7^4)(.3)+ 10\ (.7^3)(.3^2)$
    - **83.7% majority vote accuracy**
  - 101 such classifiers
    - **99.9% majority vote accuracy**

Note: Binomial Distribution: The probability of observing *x* heads in a sample of *n* independent coin tosses,
where in each toss the probability of heads is *p*, is

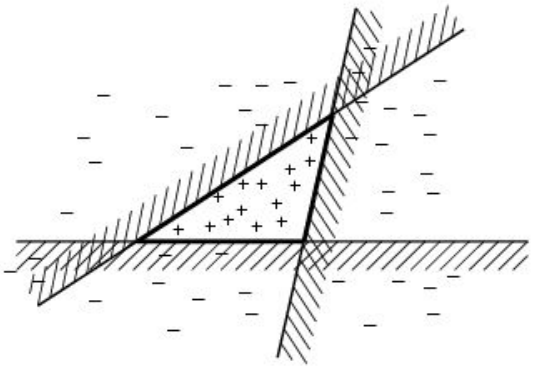
$$P(X = x|p, n) = \frac{n!}{r!(n-x)!}p^x(1-p)^{n-x}$$

Ensemble Learning

- Another way of thinking about ensemble learning:

- way of enlarging the hypothesis space, i.e., the ensemble itself is a hypothesis and the new hypothesis space is the set of all possible ensembles constructible form hypotheses of the original space.

Increasing power of ensemble learning:



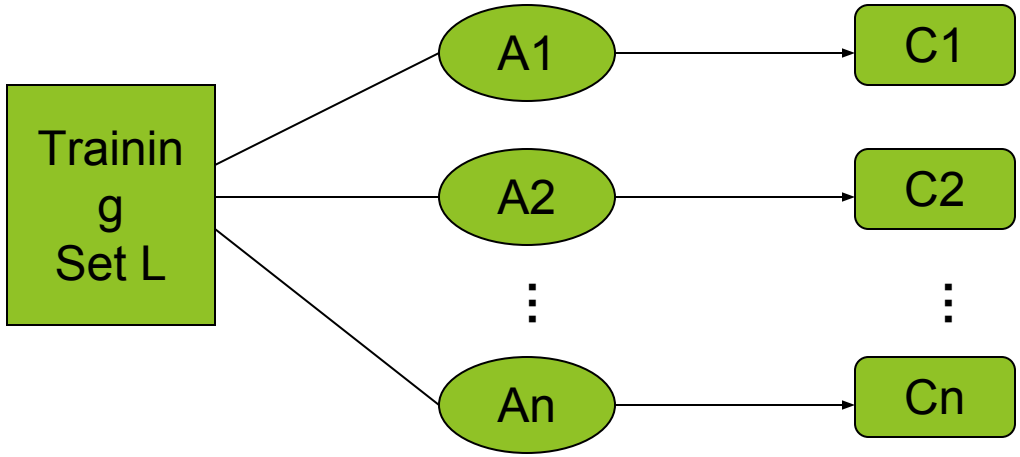
Three linear threshold hypothesis
(positive examples on the non-shaded side);
Ensemble classifies as positive any example classified positively be all three. The resulting triangular region hypothesis is not expressible in the original hypothesis space.

Different types of ensemble learning

- Different learning algorithms
- Algorithms with different choice for parameters
- Data set with different features (e.g. random subspace)
- Data set = different subsets (e.g. bagging, boosting)

Different types of ensemble learning (1)

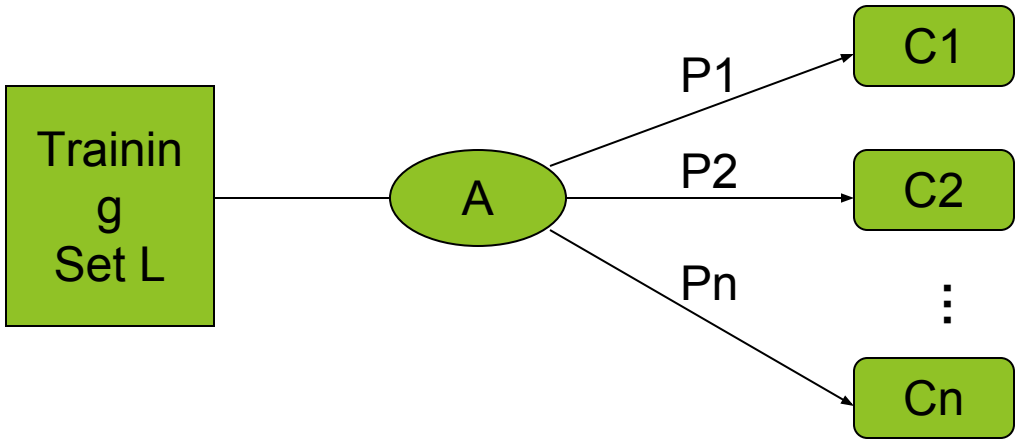
- Different algorithms, same set of training data



A: algorithm I; C: classifier

Different types of ensemble learning (2)

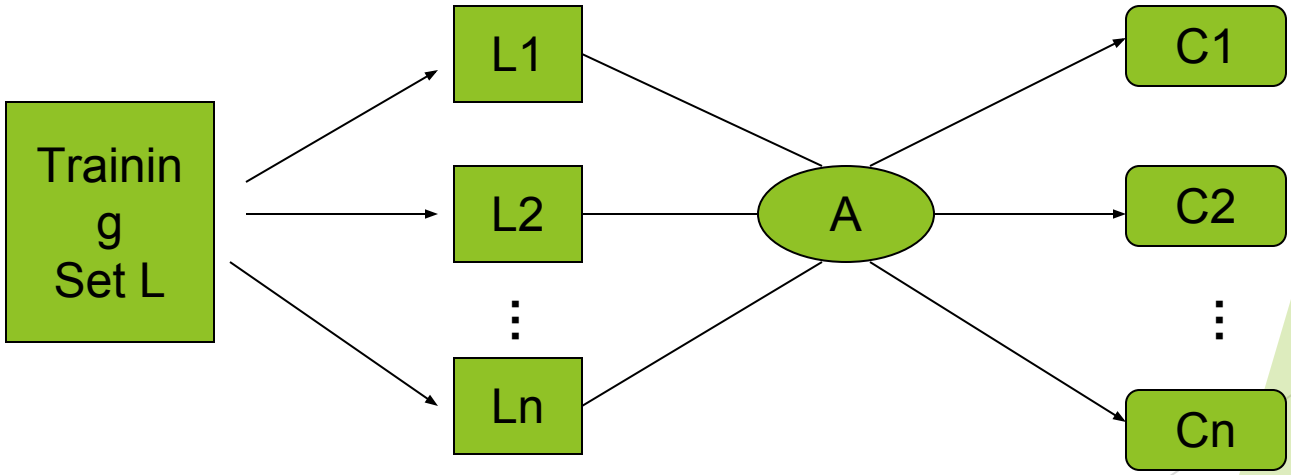
- Same algorithm, different parameter settings



P: parameters for the learning algorithm

Different types of ensemble learning (3)

- Same algorithm, different versions of data set, e.g.
  - Bagging: resample training data
  - Boosting: Reweight training data
  - Decorate: Add additional artificial training data
  - RandomSubSpace (random forest): random subsets of features



In WEKA, these are called ***meta-learners***, they take a learning algorithm as an argument (***base learner***) and create a new learning algorithm.

Combining an ensemble of classifiers (1)

- Voting:
  - Classifiers are combined in a static way
  - Each base-level classifier gives a (weighted) vote for its prediction
  - Plurality vote: each base-classifier predict a class
  - Class distribution vote: each predict a probability distribution
    - $p_C(x) = \Sigma_{C \in \mathcal{C}} p_C(x) \; / \; | \; \mathcal{C} \; |$

Combining an ensemble of classifiers (2)

- Stacking: a stack of classifiers
  - Classifiers are combined in a dynamically
  - A machine learning method is used to learn how to combine the prediction of the base-level classifiers.
  - Top level classifier is used to obtain the final prediction from the predictions of the base-level classifiers

Combining an ensemble of classifiers (3)

- Cascading:
  - Combine classifiers iteratively.
  - At each iteration, training data set is extended with the prediction obtained in the previous iteration

Bagging

- Create ensembles by *"bootstrap aggregation"*, i.e., repeatedly randomly resampling the training data (Brieman, 1996).

- Bootstrap: draw N items from X with replacement
- Each ***bootstrap sample*** will on average contain 63.2% of the unique training examples, the rest are replicates.

- Bagging
  - Train *M* learners on *M* bootstrap samples
  - Combine outputs by voting (e.g., majority vote)

- Decreases error by decreasing the variance in the results due to ***unstable learners***, algorithms (like decision trees and neural networks) whose output can change dramatically when the training data is slightly changed.

Bagging - Aggregate Bootstrapping

- Given a standard training set *D* of size *n*
- For i = 1 .. M
  - Draw a sample of size $n^* \leq n$ from *D* uniformly and with replacement
  - Learn classifier C_i
- Final classifier is a vote of $C_1 .. C_M$

Boosting

- Originally developed by computational learning theorists to guarantee performance improvements on fitting training data for a ***weak learner*** that only needs to generate a hypothesis with a training accuracy greater than 0.5 (Schapire, 1990).

- Revised to be a practical algorithm, AdaBoost, for building ensembles that empirically improves generalization performance (Freund & Shapire, 1996).

- Key Insights:
  - Instead of sampling (as in bagging) re-weigh examples!
  - Examples are given weights. At each iteration, a new hypothesis is learned (weak learner) and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.
  - Final classification based on weighted vote of weak classifiers

Construct Weak Classifiers

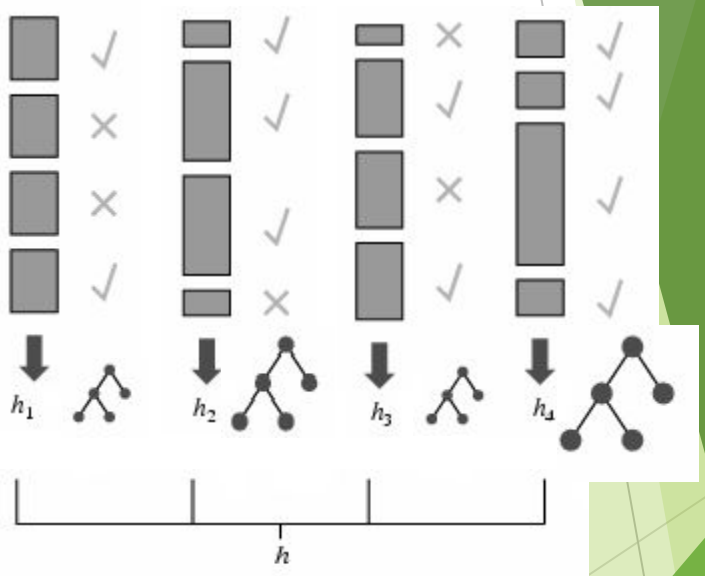
- Using Different Data Distribution
  - Start with uniform weighting
  - During each step of learning
    - Increase weights of the examples which are not correctly learned by the weak learner
    - Decrease weights of the examples which are correctly learned by the weak learner
- Idea
  - Focus on difficult examples which are not correctly classified in the previous steps
  - Intuitive justification: models should be experts that complement each other

Combine Weak Classifiers

- Weighted Voting
  - Construct strong classifier by weighted voting of the weak classifiers
- Idea
  - Better weak classifier gets a larger weight
  - Iteratively add weak classifiers
    - Increase accuracy of the combined classifier through minimization of a cost function

Adaptive Boosting

- Each rectangle corresponds to an example
- with weight proportional to its height.
- Crosses correspond to misclassified examples.
- Size of decision tree indicates the weight of that hypothesis in the final ensemble.



AdaBoost Pseudocode

TrainAdaBoost(D, BaseLearn)

For each example d_i in D let its weight $w_i = 1/|D|$

Let H be an empty set of hypotheses

For t from 1 to T do:

Learn a hypothesis, h_t, from the weighted examples: h_t=BaseLearn(D)

Add h_t to H

Calculate the error, ε_t, of the hypothesis h_t as the total sum weight of the examples that it classifies incorrectly.

If $\varepsilon_t > 0.5$ then exit loop, else continue.

Let $\beta_t = \varepsilon_t / (1 - \varepsilon_t)$

Multiply the weights of the examples that h_t classifies correctly by β_t

Rescale the weights of all of the examples so the total sum weight remains 1.

Return H

TestAdaBoost(*ex*, H)

Let each hypothesis, h_t, in H vote for *ex*'s classification with weight $\log(1/\beta_t)$

Return the class with the highest weighted vote total.

Experimental Results on Ensembles (Freund & Schapire, 1996; Quinlan, 1996)

- Ensembles have been used to improve generalization accuracy on a wide variety of problems.
- On average, Boosting provides a larger increase in accuracy than Bagging.
- Boosting on rare occasions can degrade accuracy.
- Bagging more consistently provides a modest improvement.
- Boosting is particularly subject to over-fitting when there is significant noise in the training data.
- Bagging is easily parallelized, Boosting is not.

Random Forest

- Leo Breiman, *Random Forests*, Machine Learning, 45, 5-32, 2001
- Motivation: reduce error correlation between classifiers
- Main idea: build a larger number of un-pruned decision trees
- Key: using a random selection of features to split on at each node

How Random Forest Work

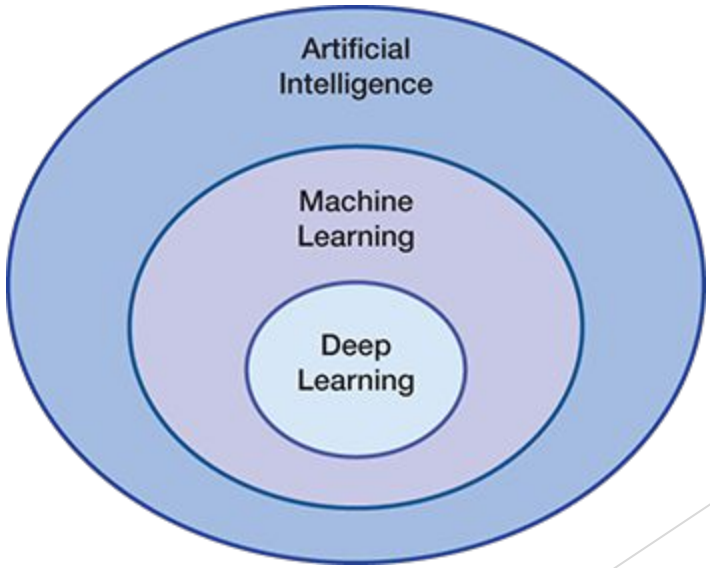
- Each tree is grown on a bootstrap sample of the training set of **N** cases.
- A number **m** is specified much smaller than the total number of variables **M** (e.g. m = sqrt(M)).
- At each node, m variables are selected at random out of the M.
- The split used is the best split on these **m** variables.
- Final classification is done by majority vote across trees.

Advantages of random forest

- Error rates compare favorably to Adaboost
- More robust with respect to noise.
- More efficient on large data
- Provides an estimation of the importance of features in determining classification
- More info at: http://stat-www.berkeley.edu/users/breiman/RandomForests/cc_home.htm

Deep Learning

- Artificial intelligence is the capability of a machine to imitate intelligent human behavior (Figure).
- Machine learning (ML) is a branch of AI that gives computers the ability to "learn" — often from data — without being explicitly programmed.
- Deep learning is a subfield of ML that uses algorithms called artificial neural networks (ANNs), which are inspired by the structure and function of the brain and are capable of self-learning.
- ANNs are trained to "learn" models and patterns rather than being explicitly told how to solve a problem



Overview of Deep Learning

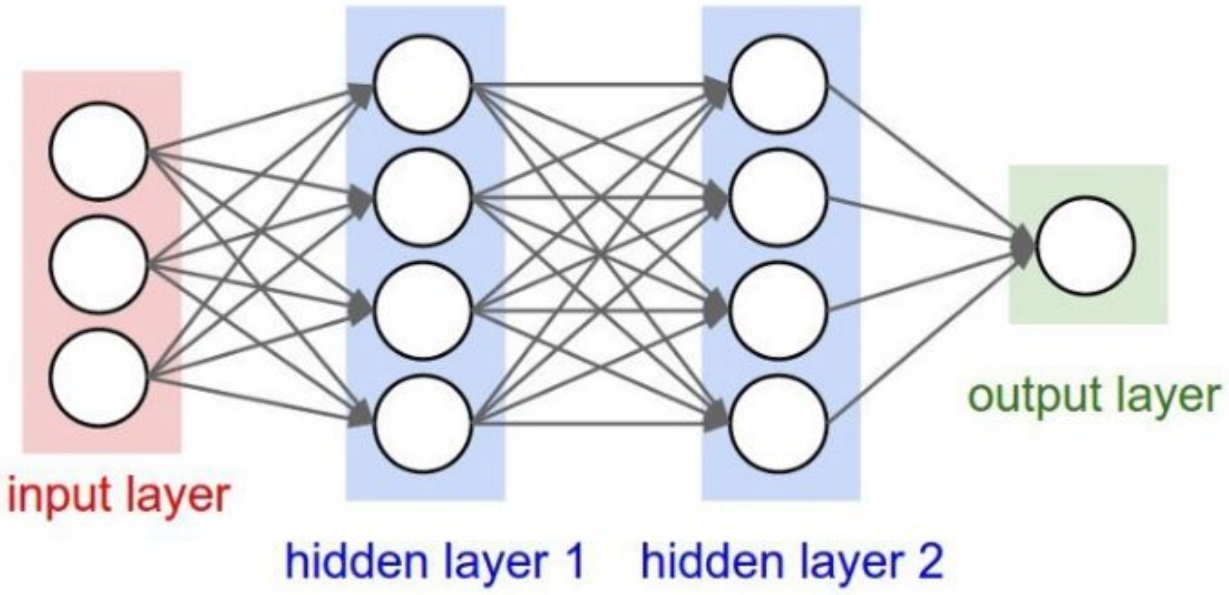
Building the Intuition

Generally speaking, deep learning is a machine learning method that takes in an input X, and uses it to predict an output of Y. As an example, given the stock prices of the past week as input, my deep learning algorithm will try to predict the stock price of the next day.

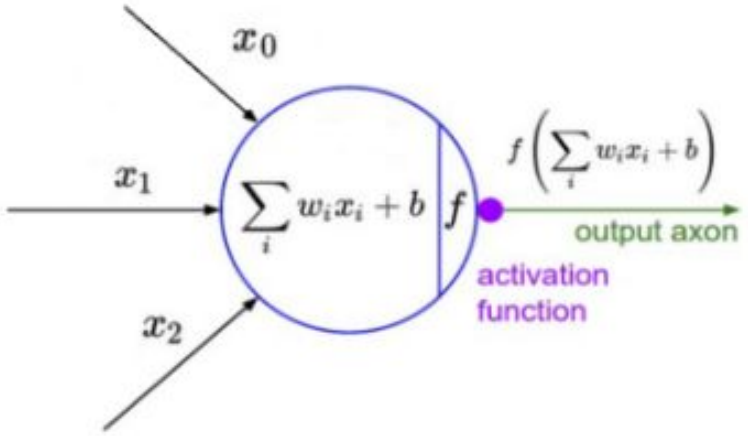
Given a large dataset of input and output pairs, a deep learning algorithm will try to minimize the difference between its prediction and expected output. By doing this, it tries to learn the association/pattern between given inputs and outputs — this in turn allows a deep learning model to generalize to inputs that it hasn't seen before.

How Do Deep Learning algorithms "learn"?

Deep Learning Algorithms use something called a neural network to find associations between a set of inputs and outputs. The basic structure is seen below:



A neural network is composed of input, hidden, and output layers — all of which are composed of "nodes". Input layers take in a numerical representation of data (e.g. images with pixel specs), output layers output predictions, while hidden layers are correlated with most of the computation.



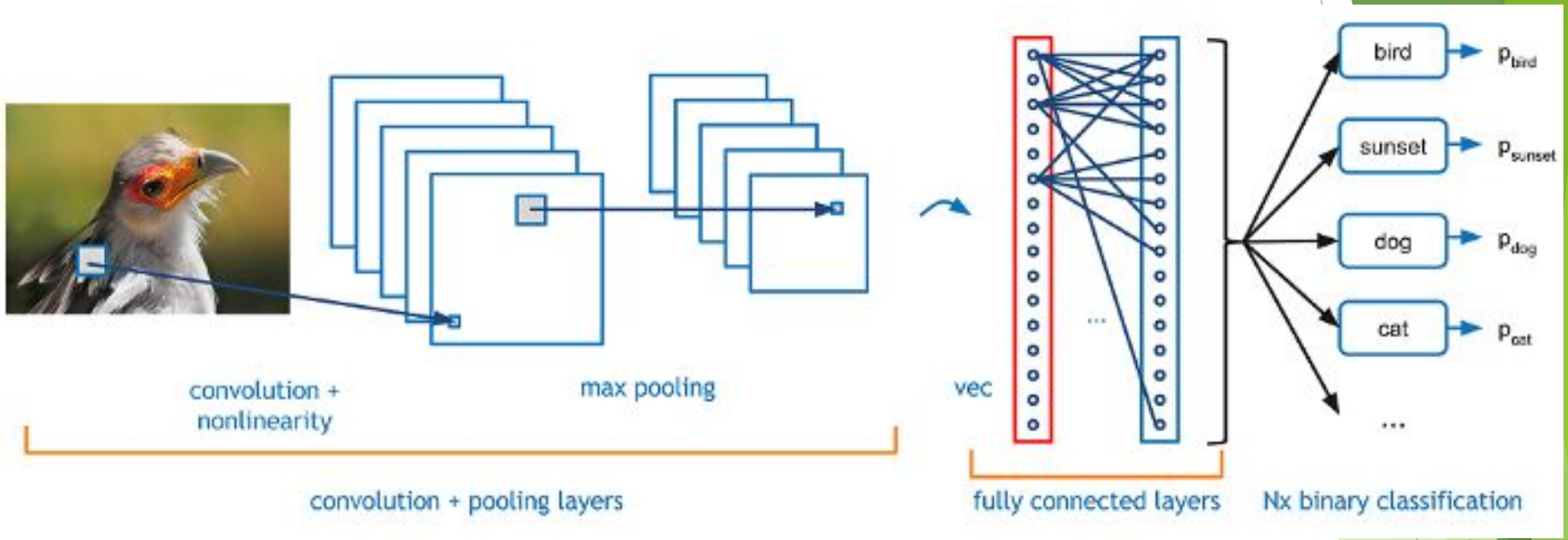
After the neural network passes its inputs all the way to its outputs, the network evaluates how good its prediction was (relative to the expected output) through something called a loss function. As an example, the "Mean Squared Error" loss function is shown below.

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Y hat represents the prediction, while Y represents the expected output. A mean is used if batches of inputs and outputs are used simultaneously (n represents sample count)

The goal of my network is ultimately to minimize this loss by adjusting the weights and biases of the network. In using something called "back propagation" through gradient descent, the network backtracks through all its layers to update the weights and biases of every node in the opposite direction of the loss function — in other words, every iteration of back propagation should result in a smaller loss function than before.

Without going into the proof, the continuous updates of the weights and biases of the network ultimately turns it into a precise function approximator — one that models the relationship between inputs and expected outputs.


bird
sunset
dog
cat
...
p_{bird}
p_{sunset}
p_{dog}
p_{cat}
convolution + nonlinearity
max pooling
vec
convolution + pooling layers
fully connected layers
Nx binary classification

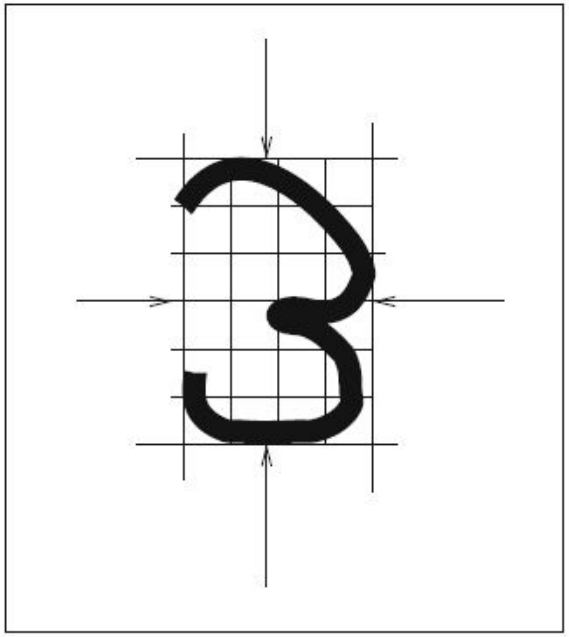
Machine Learning	Deep Learning
Works on small amount of Dataset for accuracy.	Works on Large amount of Dataset.
Dependent on Low-end Machine.	Heavily dependent on High-end Machine.
Divides the tasks into sub-tasks, solves them individually and finally combine the results.	Solves problem end to end.
Takes less time to train.	Takes longer time to train.
Testing time may increase.	Less time to test the data.

Few Instructive Applications of Machine Learning

Character Recognition

The Task To describe each character in a way that facilitates its recognition by a computer is not easy. To get the idea, just try to explain, in plain English, how to distinguish digit "5" from digit "6," or what constitutes the differencebetween a hand-written "t" and "l." You will be surprised how difficult this is. And to convert the plain-English explanation to a code that a computer can understand is harder still.

Fig. 8.1 A simple way to convert the image of a hand-written character into a vector of 64 continuous attributes, each giving the mean intensity of the corresponding field



a 6−by−4 matrix of numeric attributes, each giving mean intensity of a field

Examples and Attribute Vectors To begin with, the engineer has to define the attributes to describe the examples. Figure 8.1 illustrates one possibility. The first step identifies a rectangular region in which the digit is located; the second divides this region into $6 \times 4 = 64$ equally sized fields, each characterized by a continuous attribute whose value gives the field's mean intensity. More ink means the field reflects less light, and thus results in a lower value of the attribute, and, conversely, the value is maximized if the field contains no ink at all. Of course, this is just the principle. There is no reason why there should be only 64 attributes. Indeed, the higher the level of detail the program is expected to discern, the smaller the size of the fields it needs to rely on, and thus the greater the number of attributes that will be used to describe the examples.

The first thing to be considered is that, in the attribute-vector obtained by the mechanism from Fig. 8.1, only a small percentage of the attributes (if any) are likely to be irrelevant or redundant, which means that this is not an issue to worry about (unless the number of attributes is increased way beyond those shown in Fig. 8.1). Another aspect guiding our choice of an appropriate machine-learning paradigm is the fact that we do not know whether the classes are linearly separable, and therefore hesitate to use linear classifiers. Finally, the classifiers need not be capable of offering explanations. If the intention is to read and convert to a text editor a long text, the user will hardly care to know the exact reason why a concrete character was classified as a "P" and not as a "D."

Based on these observations, the simple and easy-to-implement *k*-NN classifier looks like a good candidate. A cautious engineer will perhaps be concerned about the computational costs incurred in a domain with hundreds of thousands of training and testing examples. But as long as the number of attributes is moderate, these costs are unlikely to be prohibitive. From the practical point of view, a reasonable limit on what constitutes "prohibitive costs" will be determined by the computations associated with the isolation of the individual characters and the conversion of their images to attribute vectors. As long as these are comparable with the costs of classification (and they usually are), the classifier is affordable.

The Number of Classes In a domain where all characters are capitalized, the induced classifier is to discern 10 digits and 26 letters, which amounts to 36 classes. If both lowercase and uppercase letters are allowed, the total increases to $10 + 2 \times 26 = 62$ classes, and to these we may have to add special characters such as "?," "!," "$," and so on. This relatively high number of classes is not without consequences, and these deserve our attention.

The Classifier Should Be Allowed to Reject an Example To decipher a person's handwriting is far from easy. Certain letters are so ambiguous as to make the reader shrug his shoulders in despair. Yet the *k*-NN classifier from Chap. 3 is undeterred: it always finds a nearest neighbor, and then simply returns its class, no matter how arbitrary this class is.

Practical experience shows this circumstance to be harmful because the costs of getting the wrong class can be greater than the costs of not knowing the class at all. Thus in an automated reader of postal codes, an incorrectly read digit can result in the letter being sent to a wrong destination, which, in turn, may cause great delay in delivery. On the other hand, if the classifier does not give *any* answer, a human employee will have to do the reading. The costs of manual processing may then be lower than the costs of getting the wrong address.

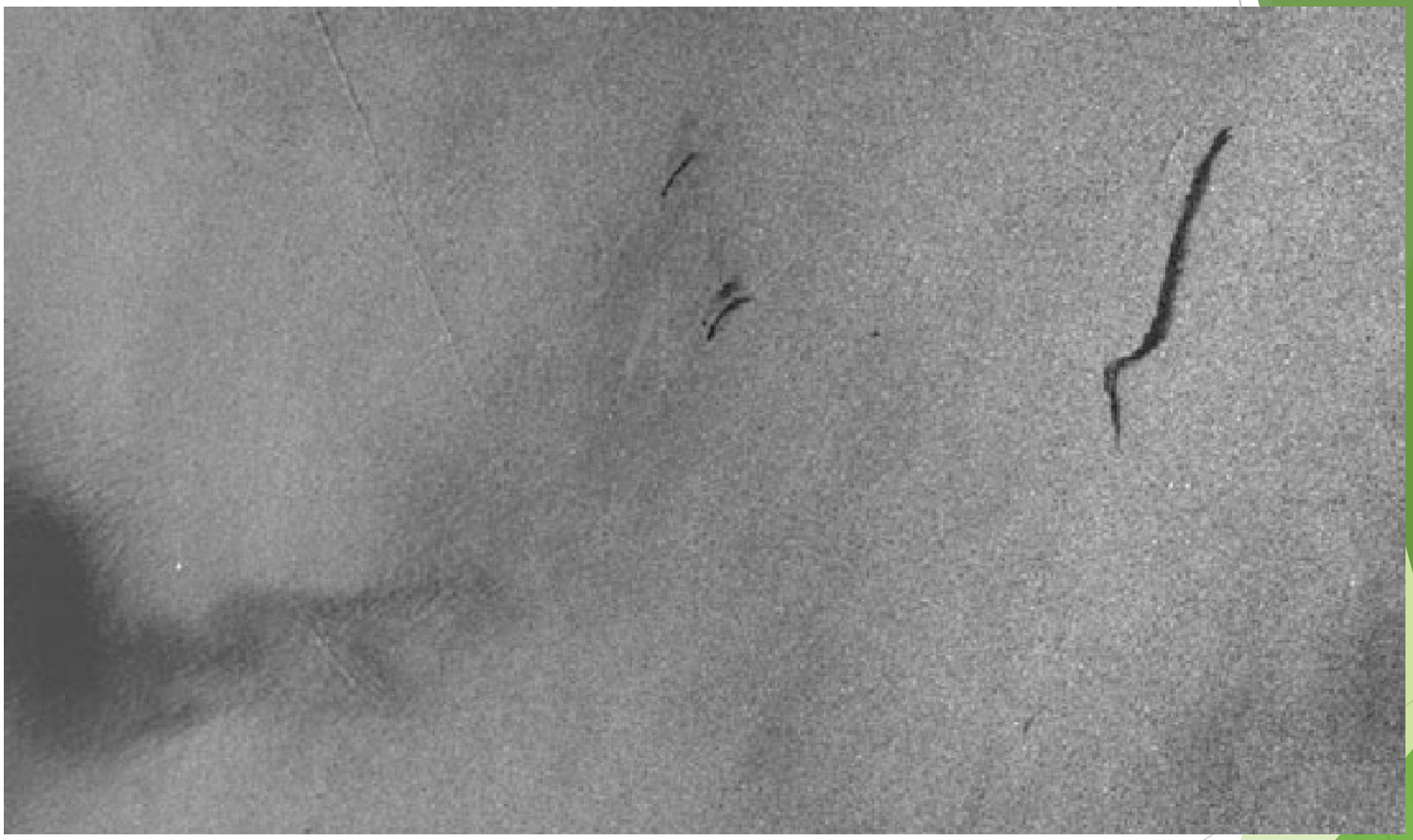
Error Rate Versus Rejection Rate A classifier that rejects ambiguous examples will surely reduce its error rate; on the other hand, excessive reluctance to classify will not be beneficial, either. What if *all* examples are rejected? The error rate then drops to zero—and yet the user will question the tool's practical utility.

The lesson is, the engineer needs to consider the trade-off between the rejection rate and the error rate. True enough, increasing the former is likely to reduce the latter; but overdoing it may render the classifier useless.

Oil-Spill Recognition

Figure 8.2 shows a radar image of the sea surface as taken by a satellite-born device. Against the grayish background, the reader can see several dark regions of the most varied characteristics: small or large, sharp or barely discernible, and of all possible shapes. What interests us here primarily is the sharp and elongated object in the vicinity of the upper-right corner: an oil spill, illegally dumped by a tanker's captain who has chosen to get rid of the residues in its bilges in the open sea rather than doing so in a specially designed terminal as required by the law. As such, this particular oil spoil is of great interest to the Coast Guard.

For all relevant sea-surface areas (close to major ports, for example), satellites take many of such "snapshots" and send them down to collaborating ground stations. Here, experts pore over these images in search for signs of illegal oil spills. Whenever they detect one, an airplane is dispatched and verifies the suspicion by a spectrometer (which is unavailable to the satellite), collects evidence, and perhaps even identifies the perpetrator.



The picture shown in Fig. 8.2 has been selected out of many, the main criterion being its rare clarity. Indeed, the oil spill it contains is so different from the other dark regions that even an untrained eye will easily recognize it as such. Even so, the reader will find it difficult to specify the oil-spill's distinguishing features in a manner that can be used in a computer program. In the case of more realistic objects, the task will be even more difficult. At any rate, to hard-code the oil-spill recognition ability is quite a challenge.

Again, machine learning got its chance, the intention being to let the machine develop the requisite skills automatically, by induction from training examples. The general scenario of the adventure can be summarized into the following steps.

1. collect a set of radar images containing oil spills;
2. use some image-processing software capable of identifying, in these images, the dark regions of interest;
3. ask an expert to label the oil spills as positive examples, and the other dark regions (so-called "look-alikes") as negative examples;
4. describe all examples by attribute vectors, and let a machine-learning program induce the requisite classifier from the training set thus obtained.

Attributes and Class Labels State-of-the-art image-processing techniques easily discover dark regions in an image. For these to be used by the machine-learning tool, we need to describe them by attributes that are hoped to capture those aspects that distinguish spills from "look-alikes." Preferably, their values should be unaffected by the given object's size and orientation.

The attributes that have been used in this project include the region's mean intensity, average edge-gradient (which quantifies the sharpness of the edges), the ratio between the lengths of the object's minor-axis and major-axis, variance of the background intensity, variance of the edge gradient, and so on. All in all, more than forty such attributes were selected in a rather ad hoc manner. Which of them would really be useful was hard to tell because experts were unable to reach consensus about the attributes' respective relevance and redundancy. The final choice was left to the machine-learning software.

When the *k*-NN classifier was applied to examples described by attributes that "survived" this decision-tree-based elimination process, the classification performance turned out to be acceptable. The oil-spill problem was thus solved with a very simple machine-learning technique—which is good: having a choice, the engineer is always well advised to give preference to the simpler tool.

The decision to use the *k*-NN classifier was also driven by another important consideration. Since this is typical of many realistic applications, let us discuss it in some detail.

Sleep Classification

Throughout the night, we go through different sleep stages such as deep, shallow, or rapid-eye movements (REM, this is when we dream). To identify these stages in a concrete sleeping subject, advanced instrumentation is used: an electrooculogram to record eye movements, an electromyogram to record muscle contractions, and contact electrodes attached to the scalp to record the brain's neural signals. Based on the readings of all these instruments, a medical expert can decide what stage the sleeping subject is in at any given moment, and can even draw a *hypnogram* such as the one shown in Fig. 8.3.

Note that the deepest sleep stage occurs here only three times during the 8-h sleep, and that it usually does not last long. Note also the `move` stage; this occurs, for instance, when the subject turns from one side to another, moves the head or an arm, and the like.

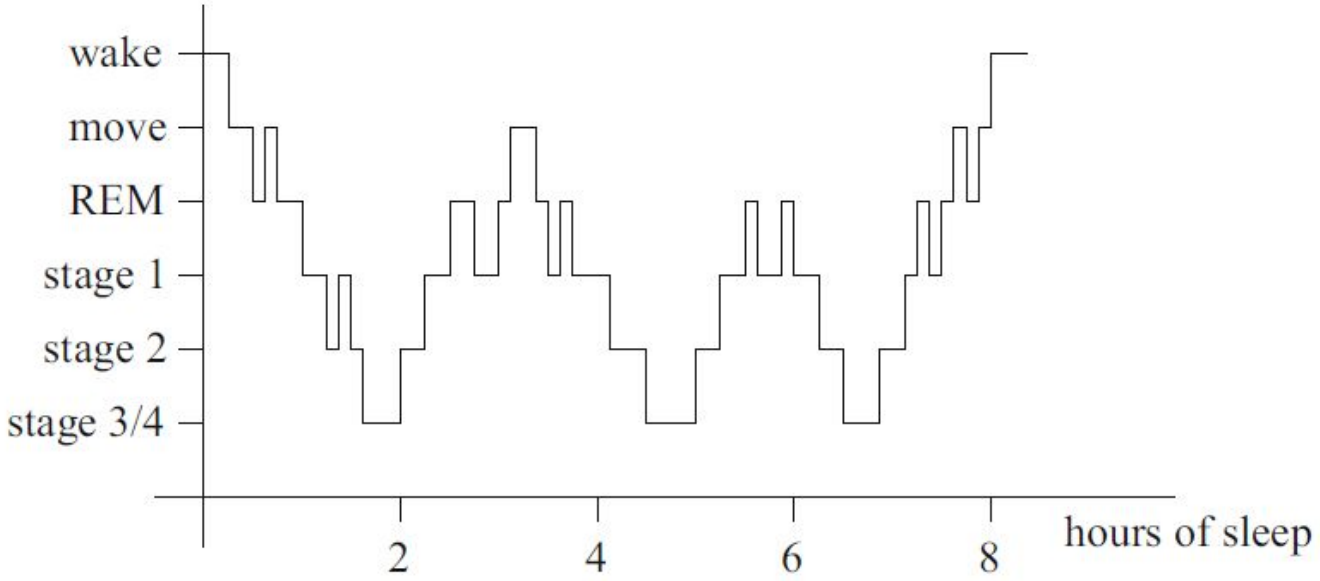


Fig. 8.3 An example hypnogram that records the sleep stages experienced by a subject during an 8-h sleep

Why Is It Important Medical practice needs to be able to recognize specific sleep stages. A case in point is the so-called *sudden infant death syndrome* (SIDS): an infant dies without any apparent cause. A newborn suspected of being in danger has to be watched, in a hospital, so that resuscitation can be started immediately. Fortunately, SIDS is known almost always to occur during the REM stage. This means that it is not necessary to watch the patient all the time, but only during this period of increased risk. For instance, a device capable of recognizing the onset of the REM stage might alert the nurse that more attention might be needed during the next five of so minutes.

The hypnogram, in turn, is a useful diagnostic tool because the distribution of the sleep stages during the night may indicate specific neural disorders such as epilepsy.

Why Machine Learning To draw the hypnogram manually is a slow and tedious undertaking, easily taking 3–5 h of a highly qualified expert's time. Moreover, the expert is not always available. This is why efforts have been made to develop a computer program capable of identifying the individual sleep stages based on observed data, and, hopefully, even to draw the hypnogram.

To be able to help, the computer needs a description of the individual stages. Such description, however, is difficult to obtain. Medical experts rely on skills obtained in the course of long training, and they use features and indications that are too subjective to be converted to a computer program.

This motivated an attempt to induce the classifier from pre-classified data. Specifically, the data-acquisition process divided the 8-h sleep period into 30-s samples, each of them treated as a separate training example. All in all, a few hours' sleep thus provided hundreds of training examples. Diverse instruments than provide data to describe each 30-s sample.

Attributes and Classes Again, the first task was to remove attributes suspected of being irrelevant or redundant. In the previous application, oil-spill recognition, this removal was motivated by the intention to increase the performance of the *k*-NN classifier (which is known to be sensitive to their presence). In sleep classification, another reason comes to the fore: the physician wants to minimize the number of measurement devices attached to the sleeping subject. Not only does their presence make the subject feel uncomfortable; they also disturb the sleep, and thus interfere with the results of the sleep analysis.

The Classifier and Its Performance The classifier employed in this particular case combined decision trees with a neural network in a manner whose details are unimportant of our needs here. Suffice it to say that the classifier's accuracy on independent data indeed achieved those 70–80% observed in human experts, which means that the natural performance limits have been reached. It is perhaps worth noting that plain decision trees were a few percent weaker than that.

This said, it is important to understand that classification accuracy does not give the full picture of the classifier's behavior (similarly as in the OCR domain from Sect. 8.1). For one thing, the classifier correctly recognized some of the seven classes most of the time, while failing on others. Particularly disappointing was its treatment of the REM state. Here, classification accuracy was in the range 90–95%, which, at first sight, looked good enough. However, closer inspection of the training data revealed that only less than 10% of all examples belonged to the REM class; this means that a comparable classification accuracy could be achieved by a classifier that says, "there is not a single REM example"—and yet this is hardly what the engineers hoped to achieve.

Text Classification

Suppose you have a vast collection of text documents, and you need to decide which of them is relevant to a specific topic of interest. If there are really a great many of them, there is no way you can do so manually. However, taking inspiration for some of the above-described applications, you choose a manageable subset of these documents, read them, assign the class labels to them (positive versus negative), and induce a classifier that will then be used to classify the rest.

Attributes A common way to describe a document is by the frequency of the words it contains. Each attribute represents one word, and its value represents the frequency with which it appears in the text. For instance, in a document that is 983 words long, the term "classifier" may appear five times, in which case its frequency is $5/983 = 0.005$.

Since the vocabulary typically contains tens of thousands of words, the attribute vectors will be very long. And of course, since only a small subset of the vocabulary finds its way into the document, most attribute values will be zero. This being somewhat unwieldy, simple applications prefer to work with only a subset of the whole vocabulary, say, 1000 most common words.

Class Labels The class labels for the training examples may be difficult to determine. Even if we want only something very easy such as to decide, for each document, if it deals with `computer science`, whether an example clearly belongs into this class may not be easy to decide because some documents are more relevant to this category than others. For instance, a very clear-cut case will be a scientific paper dealing with algorithm analysis; on the other hand, a less relevant document will only mention in passing that some algorithms are more expensive than others; and yet another document will be an article from a popular magazine.

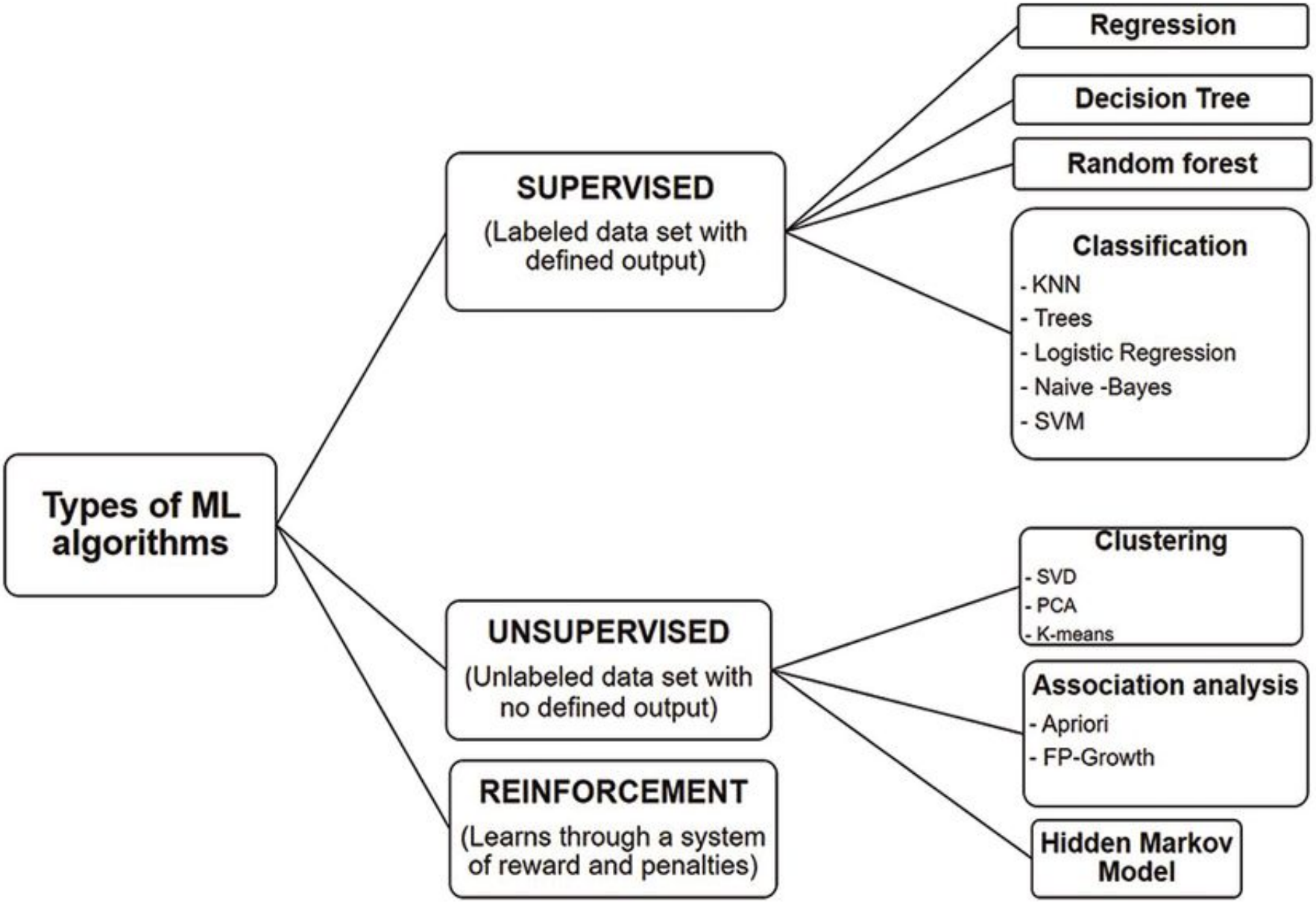
One possibility to deal with this circumstance is to "grade" the class labels. For instance, if the document is most certainly relevant, the class is 2; if it is only somewhat relevant, the class is 1; and only if it is totally irrelevant, the class is 0. The user can then decide what level of relevance is needed for the given application.

Typical Observations The great number of attributes makes the induction computationally expensive. Thus in the case of decision trees, where one has to calculate the information gain separately for each attribute, this means that tens of thousands of attributes need to be investigated for each of the tests, and these calculations can take a lot of time, especially if the number of examples is high. Upon some thought, the reader will agree that Bayesian classifiers and neural networks are quite expensive, too.

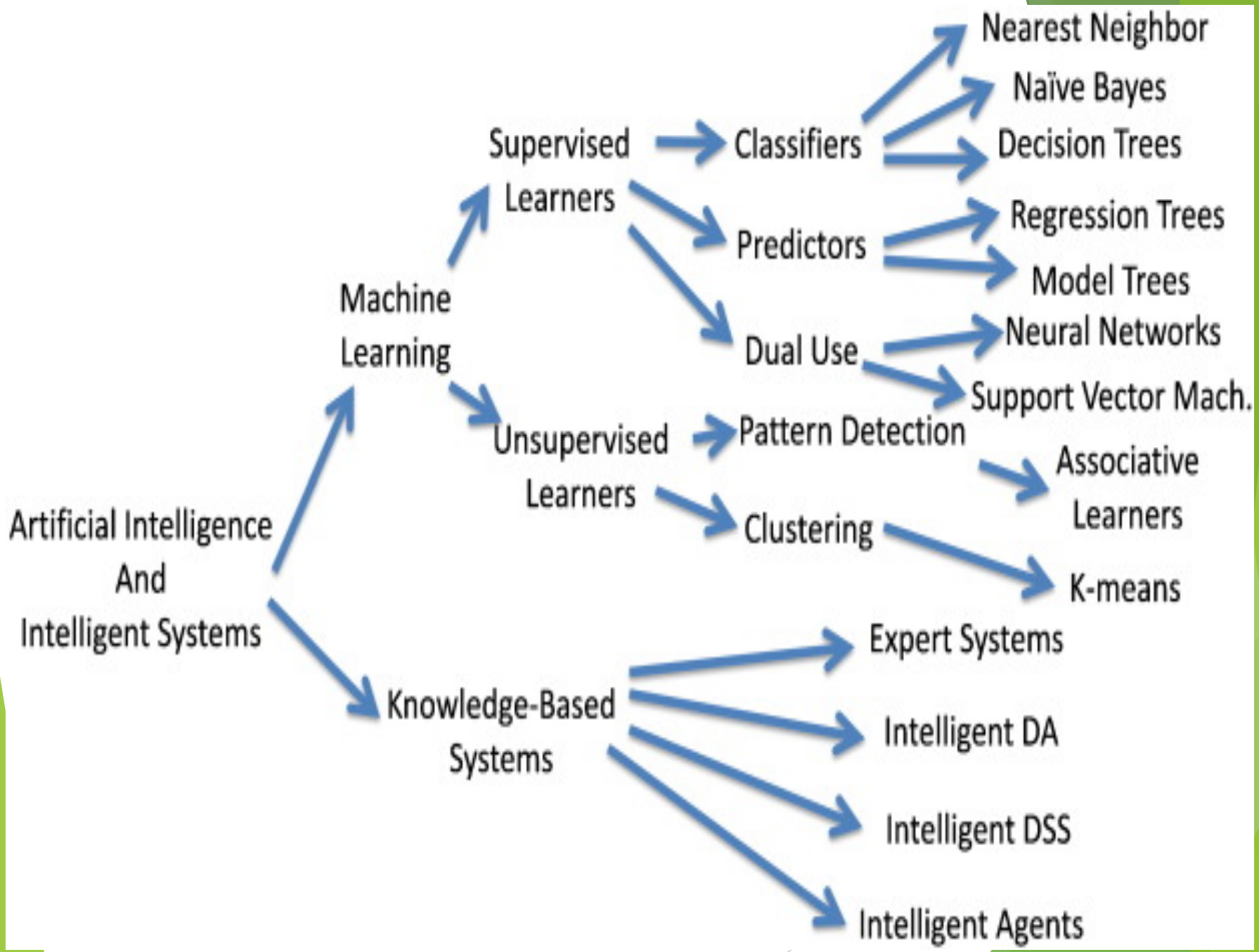
One can suggest that induction will be cheaper in the case of the k-NN classifier or a linear classifier. Here, however, another difficulty comes to the fore: for each given topic, the vast majority of the attributes will be irrelevant, a circumstance that reduces the utility of both of these paradigms.

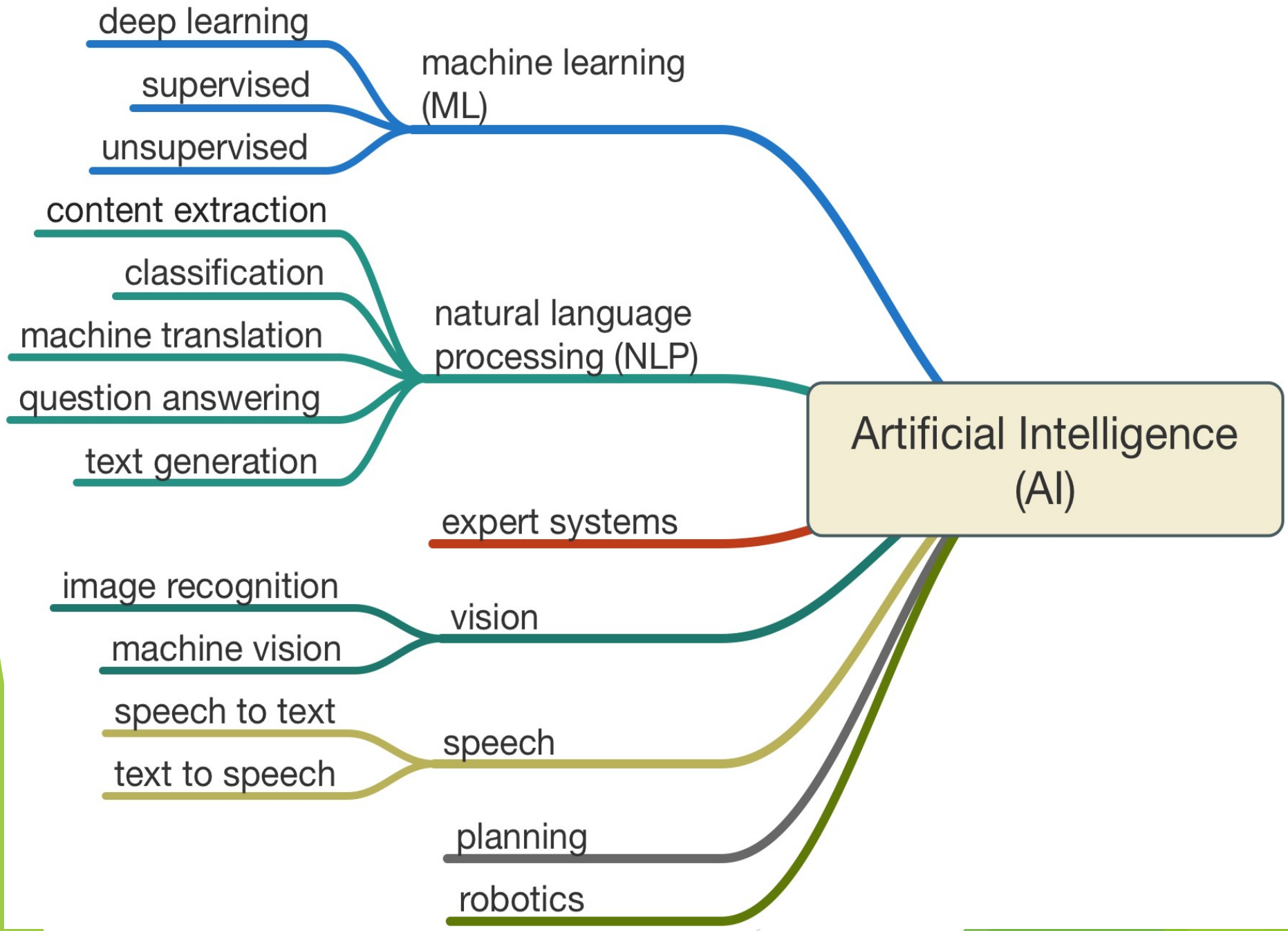
Applications :

1. **Automatic Text Generation** – Corpus of text is learned and from this model new text is generated, word-by-word or character-by-character. Then this model is capable of learning how to spell, punctuate, form sentences, or it may even capture the style.
2. **Healthcare** – Helps in diagnosing various diseases and treating it.
3. Automatic Machine Translation – Certain words, sentences or phrases in one language is transformed into another language (Deep Learning is achieving top results in the areas of text, images).
4. **Image Recognition** – Recognizes and identifies peoples and objects in images as well as to understand content and context. This area is already being used in Gaming, Retail, Tourism, etc.
5. **Predicting Earthquakes** – Teaches a computer to perform viscoelastic computations which are used in predicting earthquakes.



Abbreviations: KNN: k-nearest neighbour; SVM: Support Vector Machine; SVD: Singular Value Decomposition; PCA: Principal Component Analysis; FP: Frequent pattern


Artificial Intelligence
And
Intelligent Systems
Machine
Learning
Supervised
Learners
Classifiers
Nearest Neighbor
Naïve Bayes
Decision Trees
Predictors
Regression Trees
Model Trees
Dual Use
Neural Networks
Support Vector Mach.
Unsupervised
Learners
Pattern Detection
Associative
Learners
Clustering
K-means
Knowledge-Based
Systems
Expert Systems
Intelligent DA
Intelligent DSS
Intelligent Agents


Artificial Intelligence (AI)
machine learning (ML)
deep learning
supervised
unsupervised
natural language processing (NLP)
content extraction
classification
machine translation
question answering
text generation
expert systems
vision
image recognition
machine vision
speech
speech to text
text to speech
planning
robotics