MACHINE LEARNING
23DS4PCMLG

Machine Learning

Sem	IV		
Course Code:	23DS4PCMLG	Total Contact Hours: 40 hours	
L-T-P:	3-0-1	Total Credits:	4

UNIT 1

- **Machine Learning Landscape:**
- Introduction, Types of Machine Learning, Challenges of Machine Learning, Testing and Validating. Supervised Learning
- **Decision Tree Learning:**
- Decision tree representation, Appropriate problems for decision tree learning, Basic decision tree learning algorithm, Issues in Decision tree learning, CART Training algorithm

UNIT 2

- **Support Vector Machines:**
- Linear SVM, Non Linear SVM, SVM Regression, Under the Hood.
- **Instance Based Learning:**
- Introduction, k-Nearest Neighbor learning

UNIT 3

- **Probabilistic Learning-Bayesian Learning:**
- Bayes Theorem and Concept Learning, Maximum Likelihood, Minimum Description Length Principle, Bayes Optimal Classifier, Gibbs Algorithm, Naïve Bayes Classifier, Bayesian Belief Network, EM Algorithm.

UNIT 4

- **Ensemble Learning and Random Forests:**
- Voting Classifiers, Bagging and Pasting, Random Patches and Random Subspaces, Random Forests, Boosting, Stacking

UNIT 5

- **Unsupervised Learning Techniques:**
- Clustering K-means, DBSCAN, Other Clustering Algorithms, Gaussian Mixtures Anomaly Detection, Selecting Clustering, Bayesian Gaussian Mixture Models, Other algorithms for anomaly and novelty detection
- **Reinforcement Learning:**
- Introduction, Learning Task,Q Learning, Markov Decision Process

Prescribed Text Book					
Sl. No.	Book Title	Authors	Edition	Publisher	Year
1.	Machine Learning	Tom M. Mitchell	First	McGraw Hill Education	2013
2	Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow	Aurelien Geron	Second	O'Reilly	2020

Reference Text Book					
Sl. No.	**Book Title**	**Authors**	**Edition**	**Publisher**	**Year**
1.	Introduction to Machine Learning with Python	Andreas C Muller & Sarah Guido	First	Shroff Publishers	2019
2.	Thoughtful Machine learning	Mathew Kirk	First	Shroff Publishers	2019

E-Book						
Sl. No.	**Book Title**	**Authors**	**Edition**	**Publisher**	**Year**	**URL**
1.	The Elements of Statistical Learning	Trevor Hastie, Robert Tibshirani, Jerome H. Friedman	Second	-	2009	https://web.stanford.edu/~hastie/Papers/ESLII.pdf
2.	Machine Learning in Action	Peter Harrington	First	Manning	2017	http://www2.ift.ulaval.ca/~chaib/IFT-4102-7025/public_html/Fichiers/Machine_Learning_in_Action.pdf

MOOC Course				
Sl. No.	**Course name**	**Course Offered By**	**Year**	**URL**
1.	Machine Learning	Coursera	--	https://www.coursera.org/learn/machine-learning
2.	Introduction to Machine learning	NPTEL	2016	https://swayam.gov.in/nd_noc20_cs29/preview

Course Outcomes

At the end of the course the student will be able to

CO1	Apply different learning algorithms for various complex problems
CO2	Analyze the learning techniques for given dataset
CO3	Design a model using machine learning to solve a problem.
CO4	Ability to conduct practical experiments to solve problems using appropriate machine learning techniques.

CO-PO mapping

	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
CO1	3											
CO2		2										
CO3			3									
CO4				3								

Proposed Assessment Plan (for 50 marks of CIE)

Tool	Remarks	Marks
Internals	2	25
QUIZ	1	5
Lab Component	CIE + 2 Lab Tests	25
Total		**50**

Laboratory plan

Lab Program	Unit#	Program Details
1	1	Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.
2	2	Develop a program to construct Support Vector Machine considering a Sample Dataset
3	2	Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions
4	3	Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets
5	3	Write a program to construct a Bayesian network considering training data. Use this model to make predictions.
6	3	Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.
7	4	Implement Boosting ensemble method on a given dataset.
8	4	Write a program to construct random forest for a sample training data. Display model accuracy using various metrics
9	5	Implement tic tac toe using reinforcement learning
10	5	Consider a sample application. Deploy machine learning model as a web service and make them available for the users to predict a given instance.

SEE Exam Question paper format

Unit-1	Mandatory	One Question to be asked for 20Marks
Unit-2	Mandatory	One Question to be asked for 20Marks
Unit-3	Internal Choice	Two Questions to be asked for 20Marks each
Unit-4	Internal Choice	Two Questions to be asked for 20Marks each
Unit-5	Mandatory	One Question to be asked for 20Marks

What is 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.
- —Arthur Samuel, *1959*
- **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.**
- **—Tom Mitchell, *1997***

Examples

- **A handwriting recognition learning problem:**
- Task **T**: recognizing and classifying handwritten words within images
- Performance measure ***P:*** percent of words correctly classified
- Training experience **E**: a database of handwritten words with given classifications

- **A robot driving learning problem:**
- Task **T**: driving on public four-lane highways using vision sensors
- Performance measure **P**: average distance travelled before an error (as judged by human overseer)
- Training experience **E**: a sequence of images and steering commands recorded while observing a human driver

Spam filter- an example

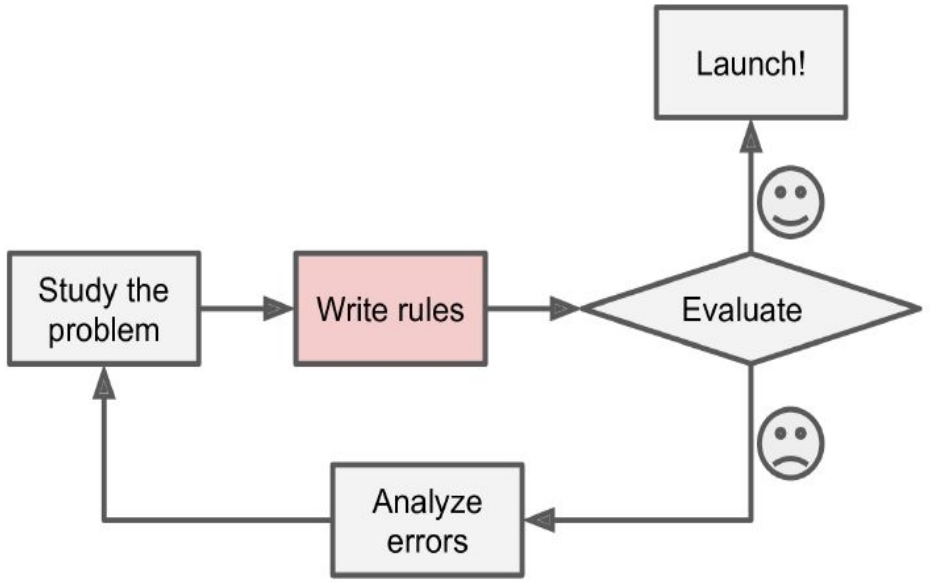
- is a Machine Learning program
- learn to flag spam: given examples of spam emails (e.g., flagged by users)
- Examples of regular(nonspam, also called "ham") emails.
- The examples that the system uses to learn are called the ***training set*.**
- **Each training example is called a *training instance* (or *sample*).**
- Task **T** is to flag spam for new emails
- Experience **E** is the *training data*, and the performance measure P needs to be defined
- for example, the ratio of correctly classified emails.
    - This particular performance measure is called *accuracy* and it is often used in classification tasks.

Why Use Machine Learning?

- **Writing a spam filter using traditional programming techniques**
- 1. **Look at what spam typically looks like**.
- words or phrases (such as “4U,” “credit card,” “free,” and “amazing”) tend to come up a lot in the subject.
- Perhaps you would also notice a few other patterns in the sender’s name, the email’s body, and so on.
- 2. **Write a detection algorithm** for each of the patterns noticed, and the program would flag emails as spam if a number of these patterns are detected.
- 3. **Test your program**, and repeat steps 1 and 2 until

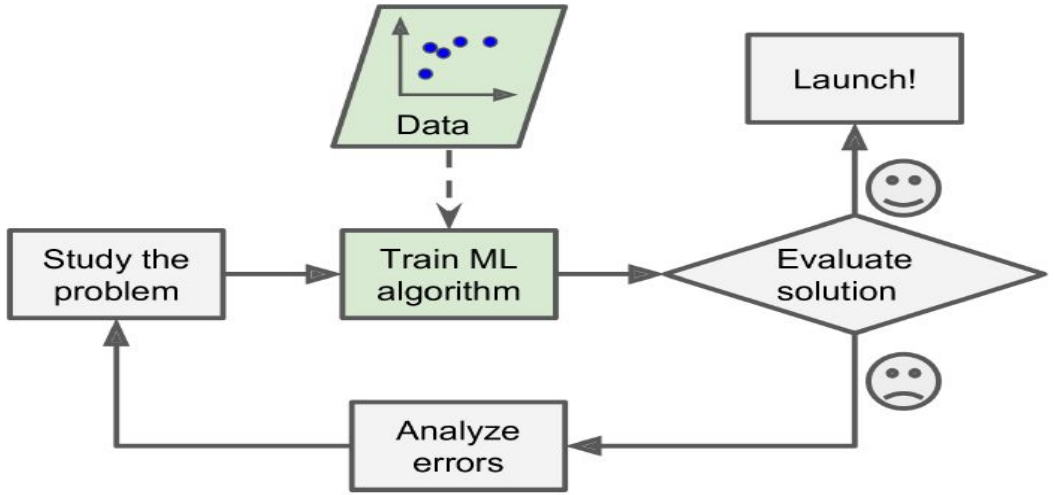
it is good enough.

- Since the problem is not trivial, your program will
- likely become a long list of complex rules
- —pretty hard to maintain.



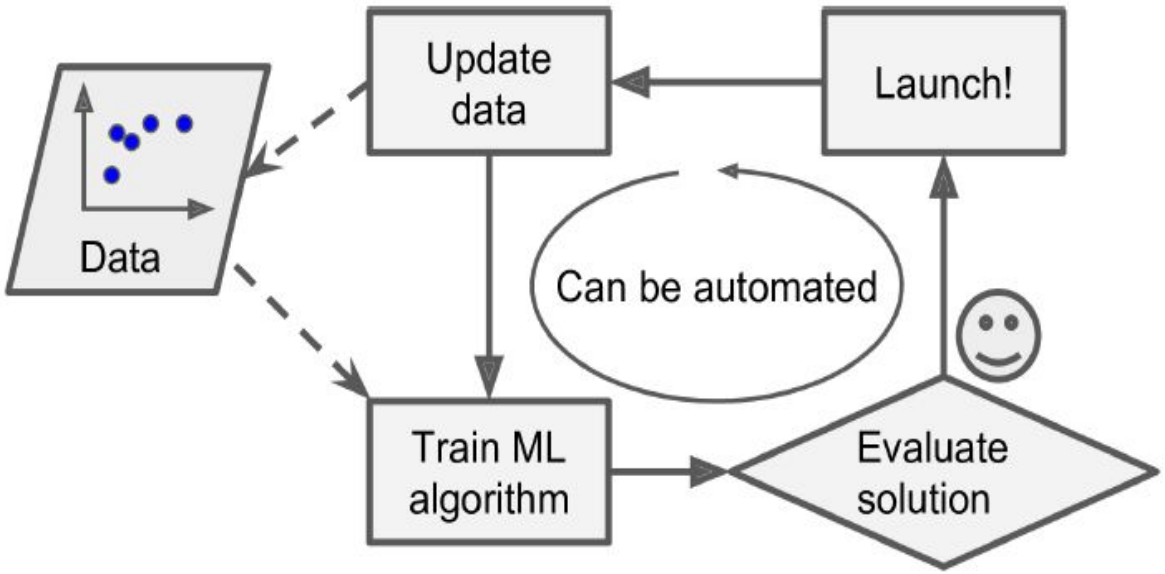
A spam filter based on Machine Learning techniques

- automatically learns which words and phrases are good predictors of spam by detecting persistent 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.



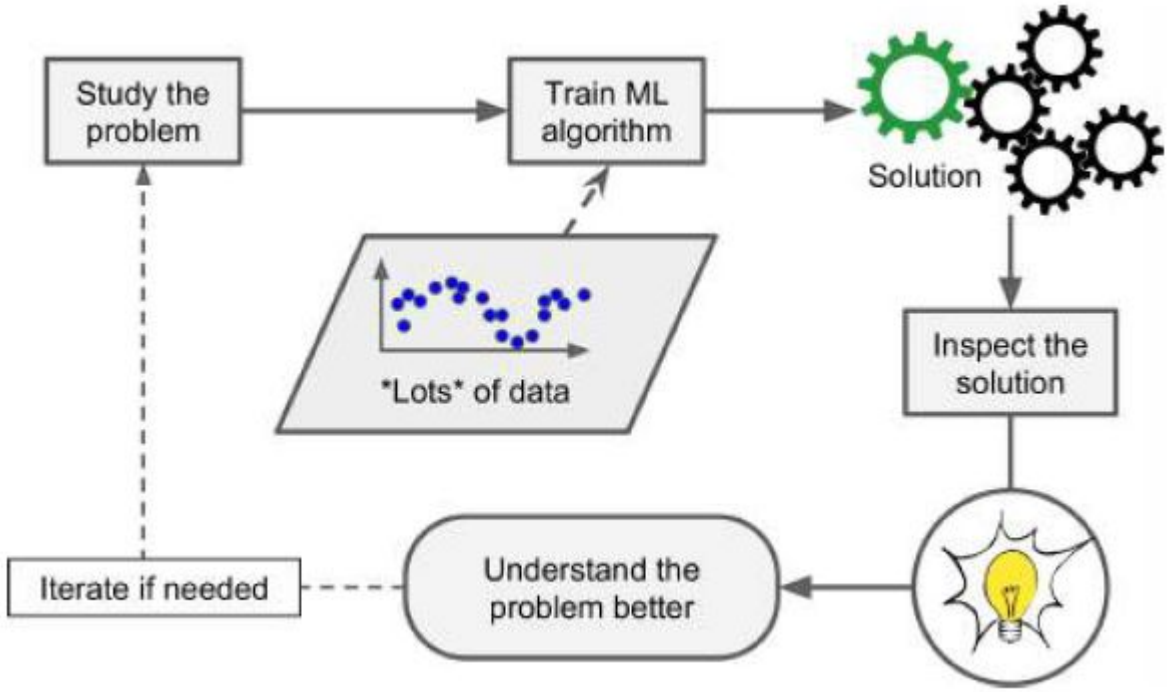
ML algorithm: Automatically adapting to change

- Spammers notice that all their emails containing “4U” are blocked : 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.
- A spam filter based on Machine Learning
- techniques automatically notices that
- “For U” has become unusually frequent
- in spam flagged by users, and it starts
- flagging them without your intervention.



Machine Learning can help humans learn

- **ML algorithms can be inspected to see what they have learned.**
- Ex: once the spam filter has been trained on enough spam, it can easily be inspected to reveal the list of words and combinations of words that it believes are the best predictors of spam.
- Sometimes this will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem.
- Applying ML techniques to dig into large amounts
- of data can help discover patterns that were
- not immediately apparent.
- This is called ***data mining*.**



Machine Learning is good 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.
- Complex problems for which there is no good solution at all using a traditional approach: the best Machine Learning techniques can find a solution.
- Fluctuating environments: a Machine Learning system can adapt to new data.
- Getting insights about complex problems and large amounts of data.

Examples of Applications

- Analysing images of products on a production line to automatically classify them
- This is image classification, typically performed using convolutional neural networks (CNNs).
- Detecting tumors in brain scans
- This is semantic segmentation, where each pixel in the image is classified (as we want to determine the exact location and shape of tumors), typically using CNNs.

- Automatically classifying news articles
- This is natural language processing (NLP), and more specifically text classification, which can be tackled using recurrent neural networks (RINNs), CNNs, or Transformers.
- Automatically flagging offensive comments on discussion forums
- This is also text classification, using the NLP tools.
- Summarizing long documents automatically
- This is a branch of NLP called text summarization, again using the same tools.

- Creating a chatbot or a personal assistant
- This involves many NLP components, including natural language understanding (NLU) and question-answering modules.
- Forecasting your company's revenue next year, based on many performance metrics
- This is a regression task (i.e., predicting values) that may be tackled using any regression model, such as a Linear Regression or Polynomial Regression model, a regression SVM, a regression Random Forest, or an artificial neural network.
- If you want to take into account sequences of past performance metrics, you may want to use RNNs, CNNs, or Transformers.
- Making your app react to voice commands
- This is speech recognition, which requires processing audio samples: since they are long and complex sequences, they are typically processed using RNNs, CNNs, or Transformers.

- Detecting credit card fraud
- This is anomaly detection.
- Segmenting clients based on their purchases so that you can design a different marketing strategy for each segment
- This is clustering.
- Representing a complex, high-dimensional dataset in a clear and insightful diagram
- This is data visualization, often involving dimensionality reduction techniques.

- Recommending a product that a client may be interested in, based on past purchases
- This is a recommender system.
- One approach is to feed past purchases (and other information about the client) to an artificial neural network, and get it to output the most likely next purchase.
- This neural net would typically be trained on past sequences of purchases across all clients.
- Building an intelligent bot for a game
- This is often tackled using Reinforcement Learning (RL), which is a branch of Machine Learning that trains agents (such as bots) to pick the actions that will maximize their rewards over time (e.g., a bot may get a reward every time the player loses some life points), within a given environment (such as the game).

Types of Machine Learning Systems

- Broad categories based on:
- Whether or not they are trained with human supervision (supervised, unsupervised, semisupervised, and Reinforcement Learning)
- Whether or not they can learn incrementally on the fly (online versus batch learning)
- Whether they work by simply comparing new data points to known data points, or instead detect patterns in the training data and build a predictive model, much like scientists do (instance-based versus model-based learning)

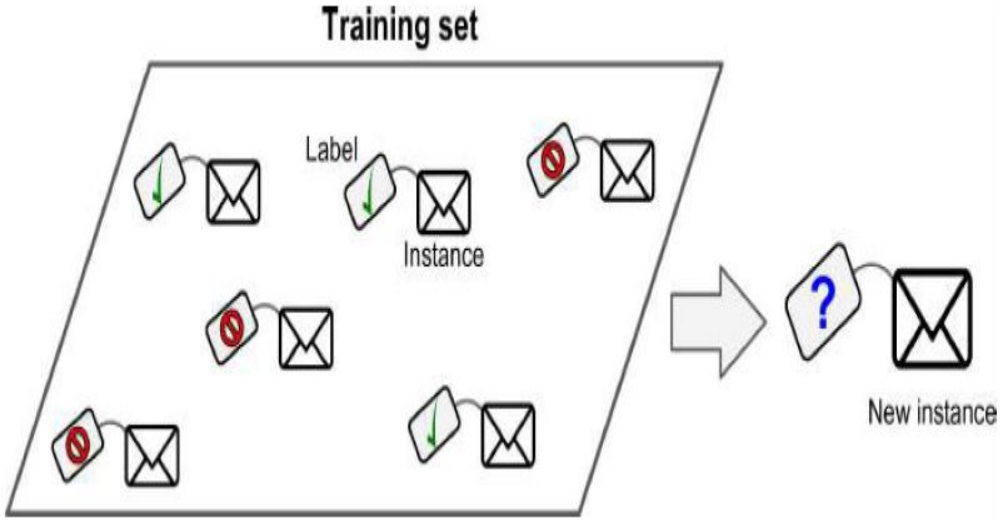
These criteria are not exclusive

- Machine Learning systems can be classified according to the amount and type of supervision they get during training.
- There are four major categories:

1. Supervised learning,
2. Unsupervised learning,
3. Semisupervised learning, and
4. Reinforcement Learning.

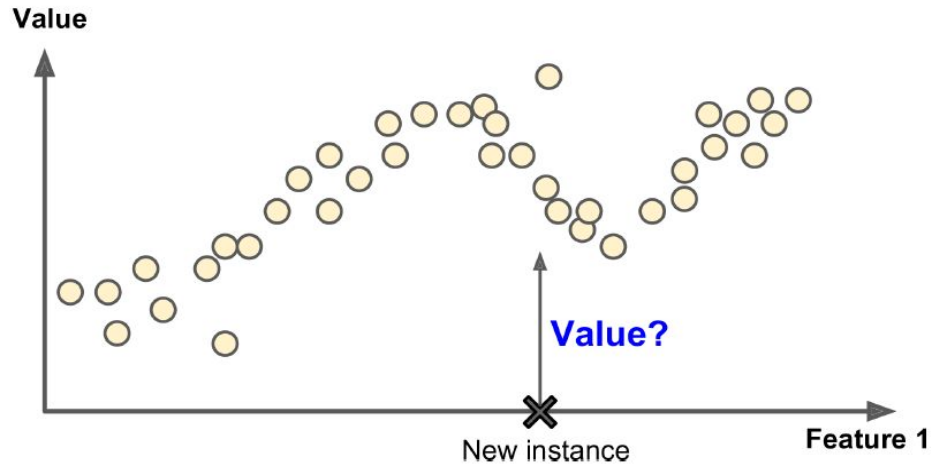
Supervised learning

- In *supervised learning*, the training data fed 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.



Example: predict a *target* numeric value- price of a car

- given a set of *features* (mileage, age, brand, etc.) called *predictors*.
- This sort of task is called *regression*.
- To train the system, give it many examples of cars, including both their predictors and their labels (i.e., their prices).
- In Machine Learning
- an *attribute* is a data type (e.g., "Mileage"),
- a *feature* generally means an attribute plus its value (e.g., "Mileage = 15,000").

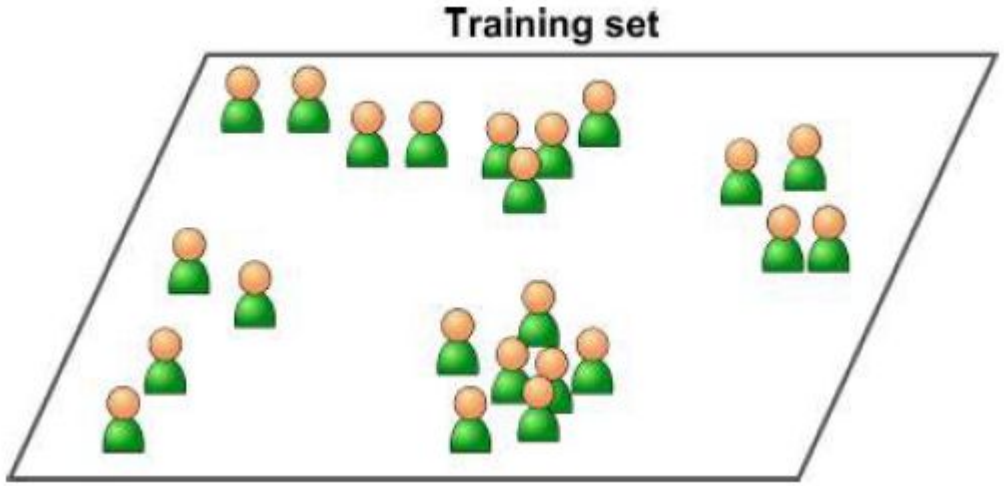


Some important supervised learning algorithms:

1. k-Nearest Neighbours
2. Linear Regression
3. Logistic Regression
4. Support Vector Machines (SVMs)
5. Decision Trees and Random Forests
6. Neural networks

Unsupervised learning

- In *unsupervised learning, the* training data is unlabelled.
- The system tries to learn without a labelled dataset.



- *An unlabelled training set for unsupervised learning*

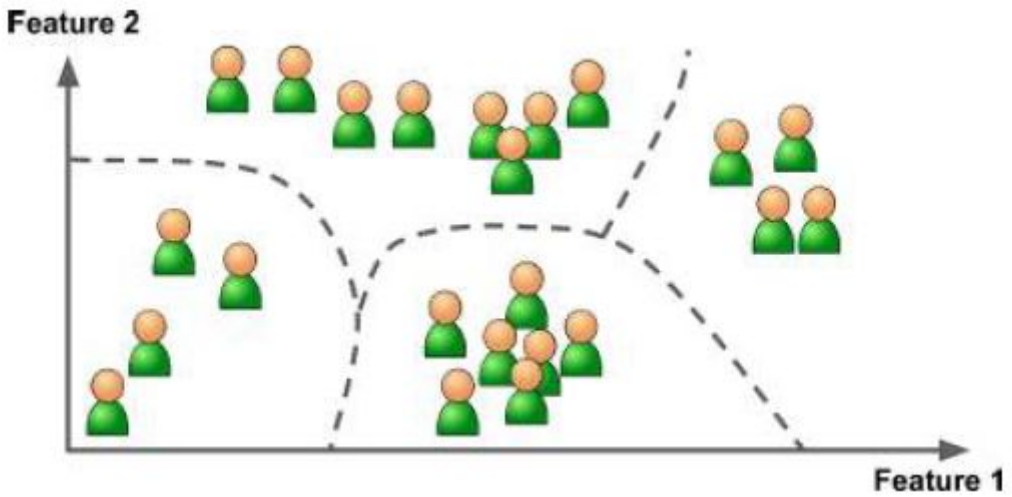
Some unsupervised learning algorithms:

- • Clustering
- —K-Means
- —DBSCAN
- —Hierarchical Cluster Analysis (HCA)
- • Anomaly detection and novelty detection
- —One-class SVM
- —Isolation Forest

- Visualization and dimensionality reduction
- —Principal Component Analysis (PCA)
- —Kernel PCA
- —Locally-Linear Embedding (LLE)
- —t-distributed Stochastic Neighbor Embedding (t-SNE)
- • Association rule learning
- —Apriori
- —Eclat

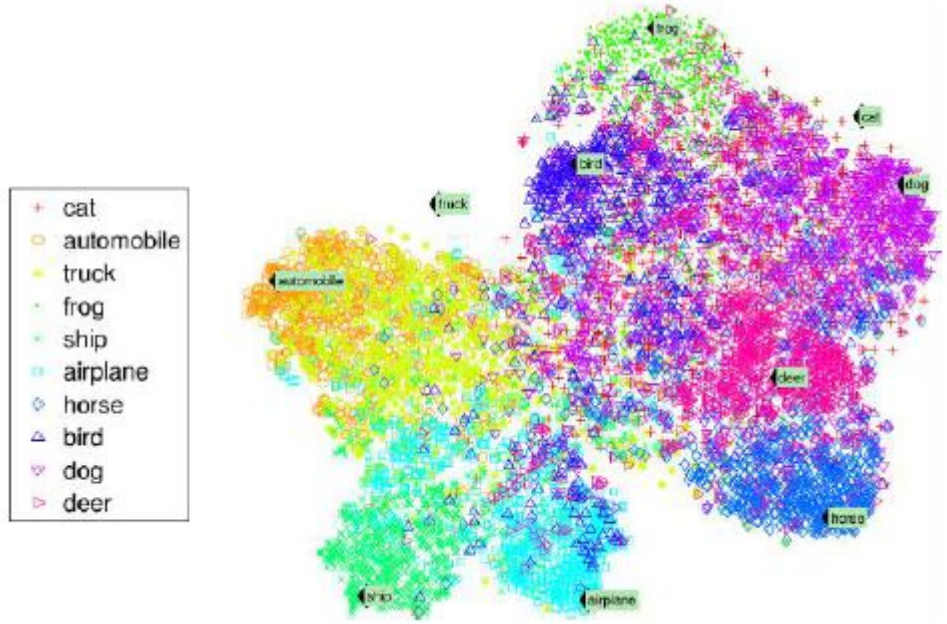
Example

- Huge data about a blog's visitors.
- Run a *clustering* algorithm to detect groups of similar visitors.
- You never tell the algorithm which group a visitor belongs to: it finds those connections without your help.
- For example,
  - it might notice that 40% of 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.
- *hierarchical clustering* algorithm: subdivide each group into smaller groups.
- This may help you target your posts for each group.



Visualization algorithms are unsupervised learning algorithms:

- Feed them a lot of complex and unlabelled data, and they output a 2D or 3D representation of your data that can easily be plotted.
- These algorithms try to preserve as much structure as they can
  - e.g., trying to keep separate clusters in the input space from overlapping in the visualization
- so you can understand how the data is organized and perhaps identify unsuspected patterns.



Example of a t-SNE visualization highlighting semantic clust

Tasks related to unsupervised learning

- Dimensionality reduction
- Anomaly detection
- Novelty detection
- Association rule learning

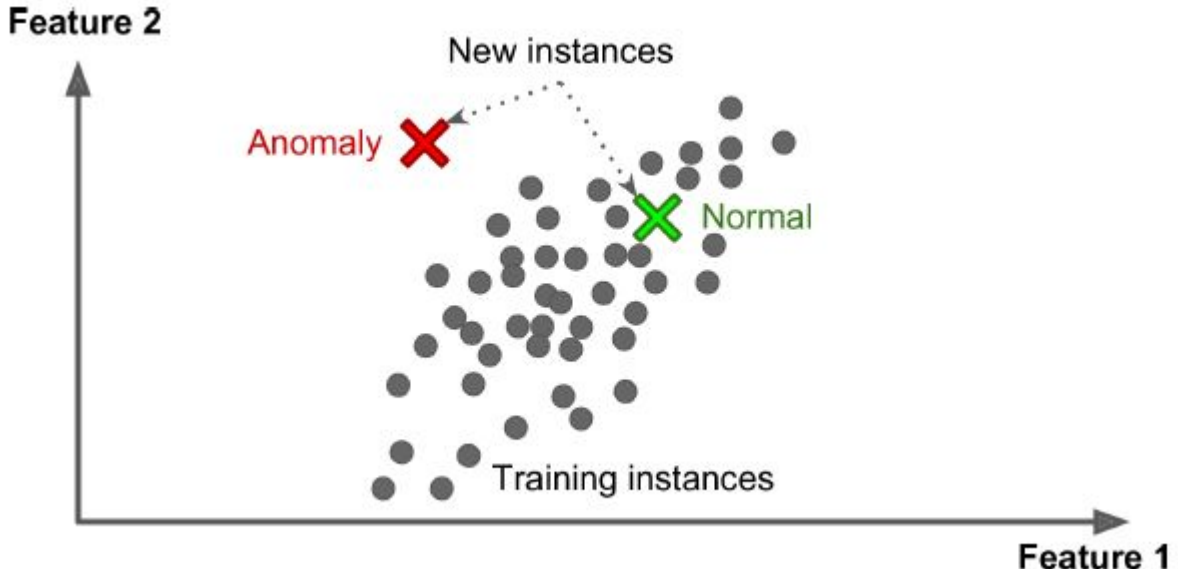
Dimensionality reduction

- Goal: simplify the data without losing too much information.
- One way to do this is to merge several correlated features into one.
- For example, a car's mileage may be very correlated with its age, so the dimensionality reduction algorithm will merge them into one feature that represents the car's wear and tear.
- This is called *feature extraction*.
- using a dimensionality reduction algorithm before you feed it to another Machine Learning algorithm.

1. run much faster
2. the data will take up less disk and memory space
3. in some cases it may also perform better.

Anomaly detection

- Example:
  a) detecting unusual credit card transactions to prevent fraud,
  b) catching manufacturing defects,
  c) automatically removing outliers from a dataset before feeding it to another learning algorithm.
- The system is shown mostly normal instances during training, so it learns to recognize them and when it sees a new instance it can tell whether it looks like a normal one or whether it is likely an anomaly.



Novelty detection

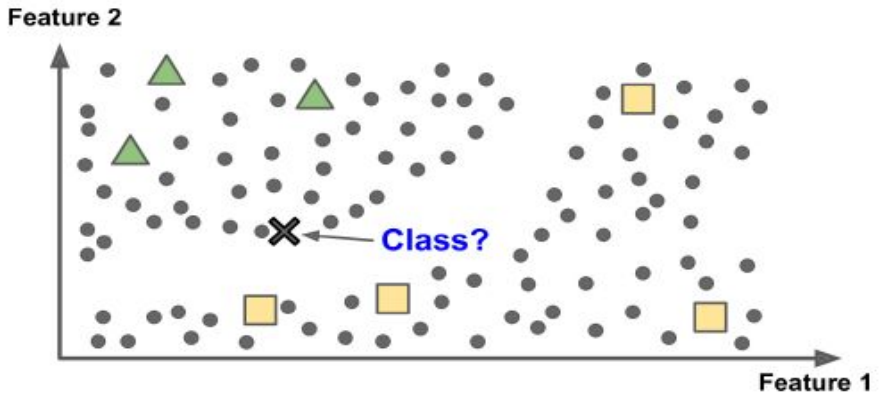
- Novelty detection algorithms expect to see only normal data during training
- It aims to detect new instances that look different from all the instances in the training set.
- EXAMPLE:
- Thousands of pictures of dogs
- 1% is Labrador
- Anomaly detection?
- Novelty detection?

Association rule learning

- Goal is to dig into large amounts of data and discover interesting relations between attributes.
- Example:
  - suppose you own a supermarket.
  - Running an association rule on your sales logs may reveal that people who purchase barbecue sauce and potato chips also tend to buy steak.
  - Thus, you may want to place these items close to each other.

Semisupervised learning

- Labelling data is time-consuming & costly, and often has plenty of unlabelled instances, and a few labelled instances.
- Algorithms that deal with data that's partially labelled are called SEMISUPERVISED LEARNING

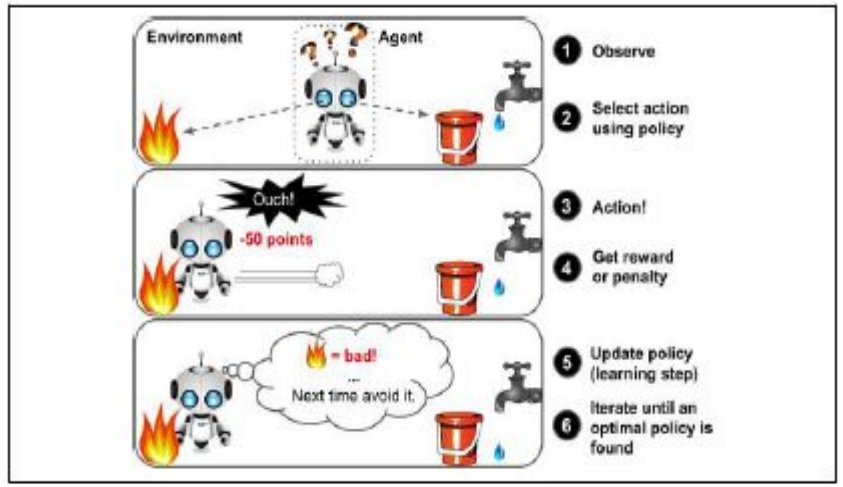


- Semisupervised Learning with two classes(triangles & squares):
  - the unlabelled examples (circles) help classify a new instance (cross mark) into the triangle class rather than the square class, even though it is closer to the labelled squares.

- Example: Google Photos
- 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).
- Now all the system needs is for you to tell it who these people are.
- Just one label per person, and it is able to name everyone in every photo, which is useful for searching photos.
- Most semi supervised learning algorithms are combinations of unsupervised and supervised algorithms.
- For example, *deep belief networks* (DBNs) are based on unsupervised components called *restricted Boltzmann machines* (RBMs) stacked on top of one another.

Reinforcement Learning

- The learning system called an *agent*
  - 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.



Example:

- Robots implement Reinforcement Learning algorithms to learn how to walk.
- DeepMind's AlphaGo program is also a good example of Reinforcement Learning:
- it made the headlines in May 2017 when it beat the world champion Ke Jie at the game of *Go*.
- It learned its winning policy by analyzing millions of games, and then playing many games against itself.
- Note that learning was turned off during the games against the champion; AlphaGo was just applying the policy it had learned.

- Classification of Machine Learning systems based on the system's incrementally learning capability from a stream of incoming data.

1. Batch Learning and
2. Online Learning

Batch learning

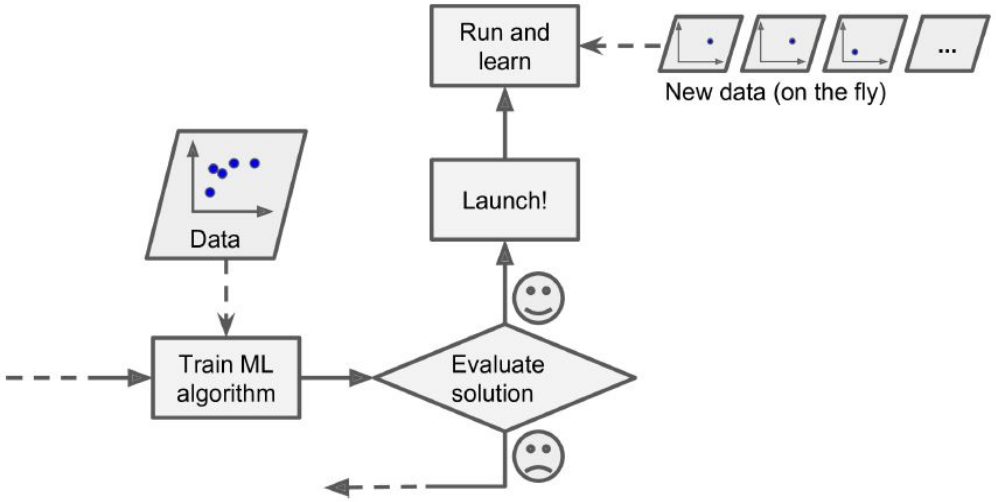
- The system is incapable of learning incrementally: must be trained using all the available data.
- Takes a lot of time and computing resources and done offline.
- **Offline learning**
  - The system is trained
  - it is launched into production and
  - runs without learning anymore (applies what it has learned).
- If you want a batch learning system to know about new data (such as a new type of spam), you need to train a new version of the system from scratch on the full dataset (not just the new data, but also the old data), then stop the old system and replace it with the new one.
- Fortunately, the whole process of training, evaluating, and launching a Machine Learning system can be automated, so even a batch learning system can adapt to change.

- Drawback:

1. training using the full set of data can take many hours,
    1. train a new system periodically (every 24 hours or weekly).
2. requires a lot of computing resources (CPU, memory space, disk space, disk I/O, network I/O, etc.).
    1. it will end up costing you a lot of money.
3. If the amount of data is huge, it is nearly impossible to use a batch learning algorithm.
4. No autonomously learning

Online learning

- Train the system incrementally by feeding it data instances sequentially, either individually or by small groups called *mini-batches*.
- Each learning step is fast and cheap, so the system can learn about new data on the fly, as it arrives

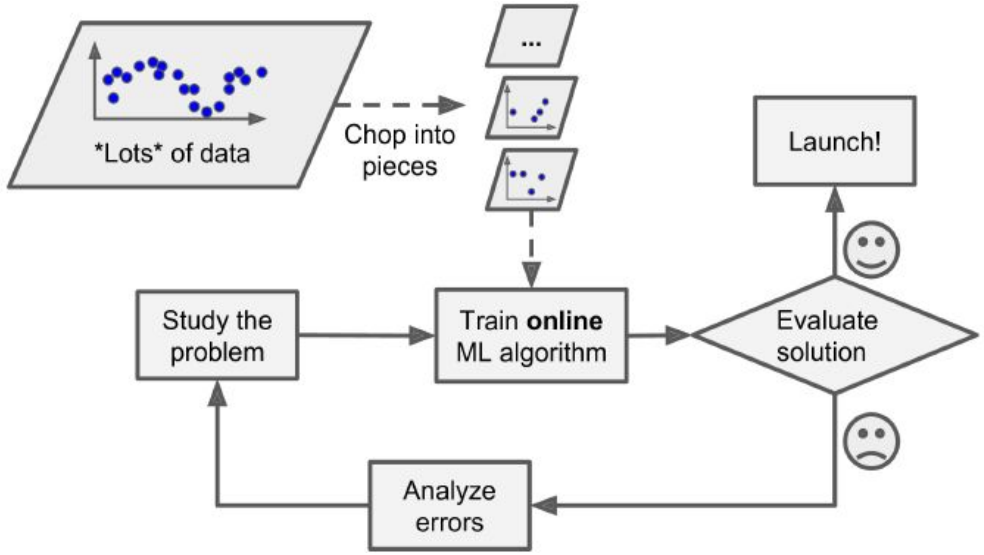


- Good for:

1. Systems that receive data as a continuous flow (e.g., stock prices)
2. Systems that need to adapt to change rapidly or autonomously.
3. Systems with limited computing resources

- Once learned about new data instances, it can discard them. This can save a huge amount of space.

- *Out-of-core* learning
- Used to train systems on huge datasets that cannot fit in one machine's main memory.
- The algorithm loads part of the data, runs a training step on that data, and repeats the process until it has run on all of the data
- usually done offline, *incremental learning*.



- Learning rate: how fast system should adapt to changing data.
- high learning rate
- system will rapidly adapt to new data
- but it will also tend to quickly forget the old data
- low learning rate
- system will learn more slowly
- but it will be less sensitive to noise in the new data or to outliers.

- Challenge with online learning
- if bad data is fed to the system, the system's performance will gradually decline.
- In a live system, users/clients will notice.
- For example, bad data could come from a malfunctioning sensor on a robot.
- To reduce this risk
- monitor system closely and promptly switch learning off and possibly revert to a previously working state if you detect a drop in performance.
- monitor the input data and react to abnormal data (e.g., using an anomaly detection algorithm).

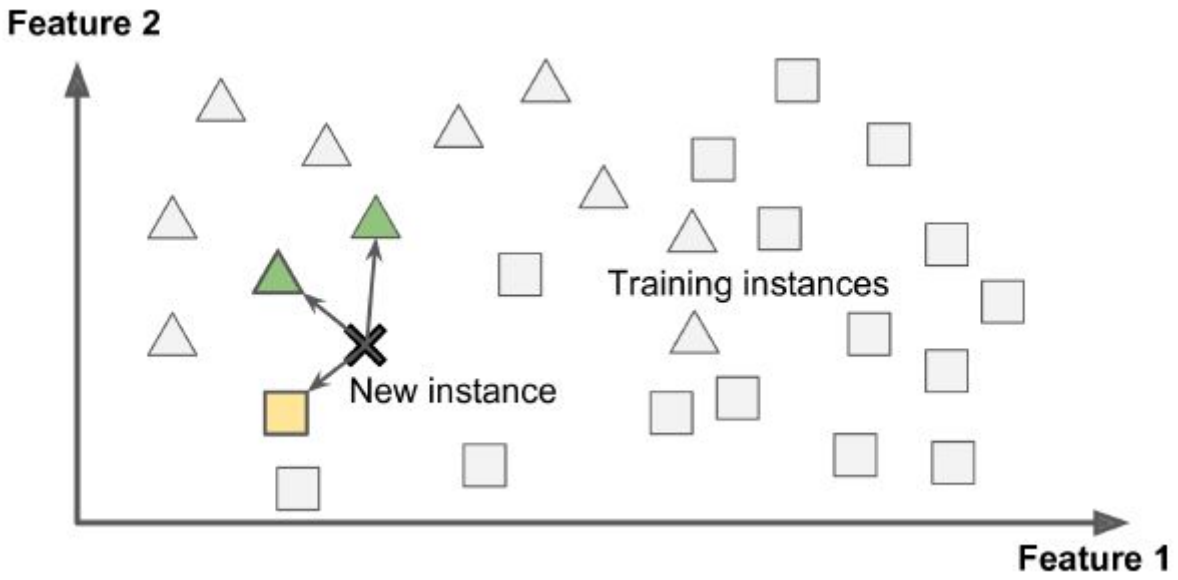
- Categorize Machine Learning systems based on how they *generalize*.
- Machine Learning tasks make predictions; the true goal is to perform well on new instances.
- There are two main approaches to generalization:

1. instance-based learning and
2. model-based learning.

Instance-based learning

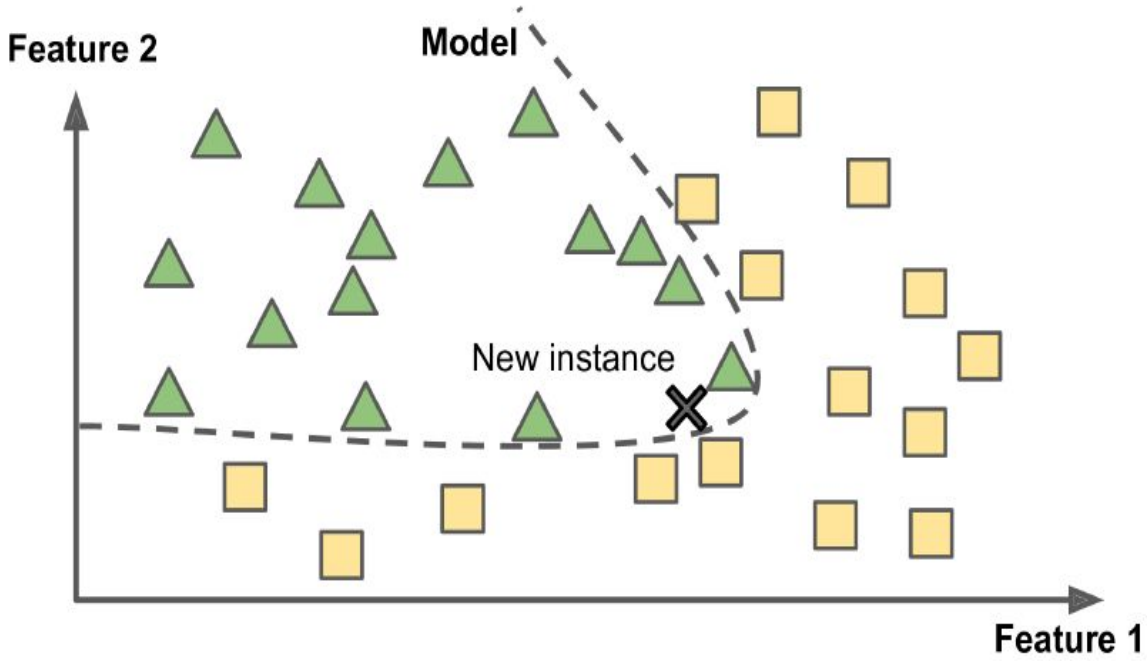
- Instead of just flagging emails that are identical to known spam emails, spam filter could be programmed to also flag emails that are very similar to known spam emails.
- This requires a *measure of similarity* between two emails.
- Example: similarity measure between two emails could be to count the number of words in common.
- The system would flag an email as spam if it has many words in common with a known spam email.
- This is called *instance-based learning*: the system learns the examples by heart, then generalizes to new cases by comparing them to the learned examples (or a subset of them), using a similarity measure.

For example, in Figure the new instance would be classified as a triangle because the majority of the most similar instance belong to that class.



Model-based learning

- Another way to generalize from a set of examples is to build a model, then use that model to make *predictions*.
- This is called *model-based learning*



Example: Does Money make people happy?

- In summary:
- • You studied the data.
- • You selected a model.
- • You trained it on the training data (i.e., the learning algorithm searched for the model parameter values that minimize a cost function).
- • Finally, you applied the model to make predictions on new cases (this is called *inference*), hoping that this model will generalize well.

Main Challenges of Machine Learning

- Main task

1. To select a learning algorithm
2. Train it on some data

- Two things that can go wrong are “bad algorithm” and “bad data.”

Bad Data

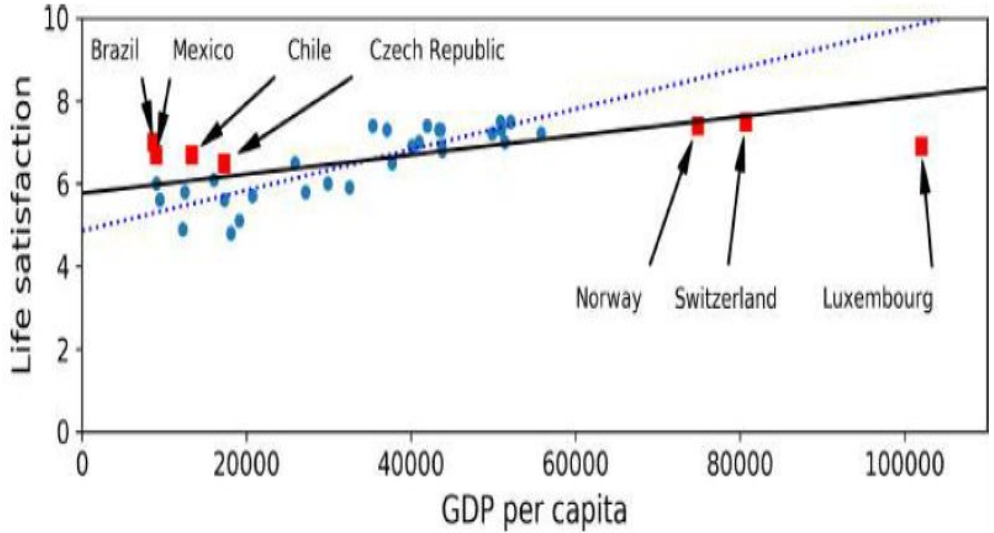
1. Insufficient Quantity of Training Data

- How can a toddler learn what an apple is?
- All it takes is for you to point to an apple and say “apple”.
- 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.

2. 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.
- For example, the set of countries for training a linear model for some instance may not be perfectly representative; a few countries missing.
- Figure shows what the data looks like
- when you add the missing countries.



- If you train a linear model on this data, you get the solid line, while the old model is represented by the dotted line.
- adding a few missing countries significantly alter the model, but it makes it clear that such a simple linear model is probably never going to work well.
- It seems that very rich countries are not happier than moderately rich countries, and conversely some poor countries seem happier than many rich countries.
- By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions, especially for very poor and very rich countries.
- 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*.

- By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions, especially for very poor and very rich countries.
- It is crucial to use a training set that is representative of the cases you want to generalize to.
- 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*.

3. Poor-Quality Data

- If your training data is full of errors, outliers, and noise
  - harder for the system to detect the underlying patterns, so system is less likely to perform well.
- Spend time cleaning up training data.
  - Most data scientists spend a significant part of their time doing just that.
- For example:
- If some instances are outliers: discard or try to fix the errors manually.
- If some instances are missing a few features (e.g., 5% of your customers did not specify their age), you must
  - decide whether you want to ignore this attribute,
  - ignore these instances,
  - fill in the missing values (e.g., with the median age), or
  - train one model with the feature and one model without it, and so on.

4. Irrelevant Features

- 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 (dimensionality reduction algorithms).
- Creating new features by gathering new data.

Bad Algorithms

5. Overfitting the Training Data

- Overgeneralizing is something that we humans do all too often, and unfortunately machines can fall into the same trap.
  - Example: you are visiting a foreign country and the taxi driver rips you off. You might be tempted to say that *all* taxi drivers in that country are thieves.
- This is called *overfitting*: it means that the
  - model performs well on the training data,
  - but it does not generalize well.

- Complex models (deep neural networks) can detect subtle patterns in the data, but if the training set is noisy, or if it is too small (sampling noise):
  - Then the model is likely to detect patterns in the noise itself which will not generalize to new instances.
- Example: say you feed a life satisfaction model many more attributes, including uninformative ones such as the country's name.
- A complex model may detect patterns like the fact that all countries in the training data with a *w* in their name have a life satisfaction greater than 7:
  - New Zealand (7.3), Norway (7.4), Sweden (7.2), and Switzerland (7.5).
- How confident are you that the W-satisfaction rule generalizes to Rwanda or Zimbabwe?
- Obviously this pattern occurred in the training data by pure chance, but the model has no way to tell whether a pattern is real or simply the result of noise in the data.

- Overfitting happens when the model is too complex relative to the amount and noisiness of the training data.
- The possible solutions are:

1. To simplify the model by selecting one with fewer parameters (e.g., a linear model rather than a high-degree polynomial model), by reducing the number of attributes in the training data or by constraining the model
2. To gather more training data
3. To reduce the noise in the training data (e.g., fix data errors and remove outliers)

- Constraining a model to make it simpler and reduce the risk of overfitting is called *regularization*.
- For example, the linear model we defined earlier has two parameters, $\theta 0$ and $\theta 1$.
- This gives the learning algorithm two *degrees of freedom* to adapt the model to the training data: it can tweak both the height ($\theta 0$) and the slope ($\theta 1$) of the line.
- If we forced $\theta 1 = 0$, the algorithm would have only one degree of freedom and would have a much harder time fitting the data properly: all it could do is move the line up or down to get as close as possible to the training instances, so it would end up around the mean.
- A very simple model indeed! If we allow the algorithm to modify $\theta 1$ but we force it to keep it small, then the learning algorithm will effectively have somewhere in between one and two degrees of freedom.
- It will produce a simpler model than with two degrees of freedom, but more complex than with just one.
- You want to find the right balance between fitting the training data perfectly and keeping the model simple enough to ensure that it will generalize well.

- Constraining a model to make it simpler and reduce the risk of overfitting is called *regularization*.
- The amount of regularization to apply during learning can be controlled by a *hyperparameter*.
- A hyperparameter is a parameter of a learning algorithm (not of the model).
- It is not affected by the learning algorithm ; it must be set prior to training and remains constant during training.
- If we set the regularization hyperparameter to a very large value
  - will get an almost flat model (a slope close to zero);
  - Overfitting can be avoided
  - but it will be less likely to find a good solution.
- Tuning hyperparameters is an important part of building a Machine Learning system.

6. Underfitting the Training Data

- *Underfitting* is the opposite of overfitting:
- It occurs when your model is too simple to learn the underlying structure of the data.
- The main options to fix this problem are:

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

In summary:

- Machine Learning makes machines get better at some task by learning from data, instead of having to explicitly code rules.
- Different types of ML systems: supervised, unsupervised, semi supervised batch or online, instance-based or model-based, and so on.
- In a ML project you gather data in a training set, and feed the training set to a learning algorithm.
- If the algorithm is model-based it tunes some parameters to fit the model to the training set (i.e., to make good predictions on the training set itself), and then it will be able to make good predictions on new cases as well.
- If the algorithm is instance-based, it just learns the examples by heart and generalizes to new instances by comparing them to the learned instances using a similarity measure.
- The system will not perform well if your training set is too small, or if the data is not representative, noisy, or polluted with irrelevant features.
- Lastly, model needs to be neither too simple (it will underfit) nor too complex (it will overfit).

Testing and Validating

- The only way to know the performance of a model on how well it will generalize is to actually try it out on new cases:

1. Put the model in production and monitor how well it performs. (Clients might complain)
2. Split data into two sets: the *training set* and the *test set:* train your model using the training set, and test it using the test set.
3. The error rate on new cases is called the *generalization error* (or *out-of-sample error*)
4. By evaluating model on the test set, you get an estimate of this error. This value tells you how well the model will perform on instances it has never seen before.
5. If the training error is low (i.e., your model makes few mistakes on the training set) but the generalization error is high, it means that your model is overfitting the training data.

- NOTE: It is common to use 80% of the data for training and *hold out* 20% for testing.

Hyperparameter Tuning and Model Selection

- Evaluating a model is simple: just use a test set.
- Suppose you are deciding between two models (say a linear model and a polynomial model): how can you decide?
  - Train both and compare how well they generalize using the test set.
- Now suppose that the linear model generalizes better, but you want to apply regularization to avoid overfitting.
- The question is: how do you choose the value of the regularization hyperparameter?
  - Train 100 different models using 100 different values for this hyperparameter.
- Suppose you find the best hyperparameter value that produces a model with the lowest generalization error, say just 5% error.
- In production unfortunately it does not perform as well as expected and produces 15% errors. What just happened?
- The problem is that you measured the generalization error multiple times on the test set, and you adapted the model and hyperparameters to produce the best model *for that particular set*.
- This means that the model is unlikely to perform as well on new data.

- Solution: *holdout validation;* hold out part of the training set to evaluate several candidate models and select the best one.
- The new heldout set is called the *validation set* (or *development set*, or *dev set*).
- More specifically
  - train multiple models with various hyperparameters on the reduced training set (i.e., the full training set minus the validation set)
  - select the model that performs best on the validation set.
- After this holdout validation process, train the best model on the full training set (including the validation set), and this gives you the final model.
- Lastly, evaluate this final model on the test set to get an estimate of the generalization error.

- If the validation set is too small, then model evaluations will be imprecise: may end up selecting a suboptimal model by mistake.
- If the validation set is too large, then the remaining training set will be much smaller than the full training set.
- Since the final model will be trained on the full training set, it is not ideal to compare candidate models trained on a much smaller training set.
    - It would be like selecting the fastest sprinter to participate in a marathon.
- Solution: perform repeated *cross-validation*, using many small validation sets.
- By averaging out all the evaluations of a model, we get a much more accurate measure of its performance.
- Drawback: the training time is multiplied by the number of validation sets.

Data Mismatch

- It is easy to get a large amount of data for training, but it is not perfectly representative of the data that will be used in production.
- Example: suppose you want to create a mobile app to take pictures of flowers and automatically determine their species.
- You can easily download millions of pictures of flowers on the web, but they won't be perfectly representative of the pictures that will actually be taken using the app on a mobile device.
- Perhaps you only have 10,000 representative pictures (i.e., actually taken with the app).
- Rule to remember: the validation set and the test must be as representative as possible of the data you expect to use in production
- So they should be composed exclusively of representative pictures:
  - shuffle them and put half in the validation set, and half in the test set.
- After training the model on the web pictures, if the performance of the model on the validation set is disappointing, you will not know whether this is because
  - model has overfit the training set, or
  - whether this is just due to the mismatch between the web pictures and the mobile app pictures.

- Solution:

1. Hold out part of the training pictures (from the web) in another set (calls the *train-dev set)*.
2. After the model is trained (on the training set, *not* on the train-dev set),
3. Evaluate it on the train-dev set:

- If it performs well, then the model is not overfitting the training set
- If performs poorly on the validation set, the problem must come from the data mismatch.
  - Tackle this problem by pre-processing the web images to make them look more like the pictures that will be taken by the mobile app.
- If the model performs poorly on the train-dev set,
  - then the model must have overfit the training set
    - try to simplify or regularize the model, get more training data and clean up the training data.

Decision Tree Learning

1. Decision tree representation
2. Appropriate problems for decision tree learning
3. Basic decision tree learning algorithm
4. Issues in Decision tree learning
5. CART Training algorithm

- Decision tree learning is a method for approximating discrete-valued target functions, in which a decision tree represents the learned function.
- Learned trees can also be re-represented as sets of if-then rules to improve human readability.
- These learning methods are type of inductive inference algorithms.
  - a type of reasoning that involves forming a general theory from specific observations. It is based on a generalization from a finite set of past observations, and extends the observed pattern or relation to other future instances.
- Decision tree learning has been applied to ***classification problems*.**
- Successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

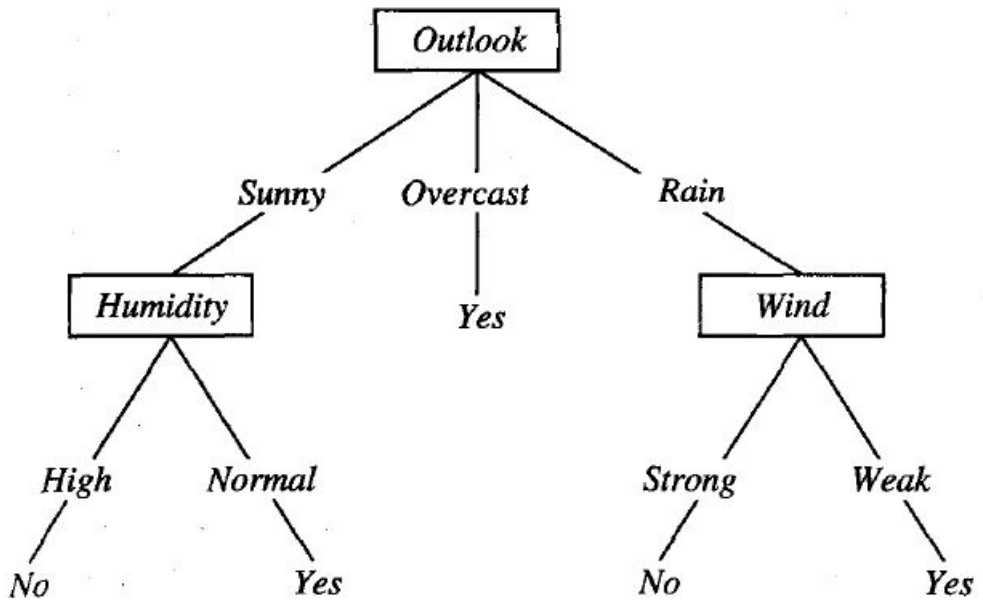
DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.

- **Each node in the tree specifies a test of some attribute of the instance**
- **Each branch descending from that node corresponds to one of the possible values for this attribute.**

- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example.
- This process is then repeated for the subtree rooted at the new node.

Example tree representation



- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.
- Ex: the decision tree shown in Figure(prev slide)corresponds to the expression

$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee \quad (Outlook = Overcast)$$
$$\vee \quad (Outlook = Rain \wedge Wind = Weak)$$

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

- Decision tree learning is generally best suited to problems with the following characteristics:
- ***Instances are represented by attribute-value pairs.***
- Instances are described by a fixed set of attributes (***Temperature)*** and their values (e.g., ***Hot, Mild, Cold).***
- However, extensions to the basic algorithm allow handling real-valued attributes as well (e.g., representing ***Temperature*** numerically).

- ***The target function has discrete output values.***
- The decision tree assigns a Boolean classification (e.g., ***yes*** or ***no)***
- Extend to learning functions with more than two possible output values
- ***Disjunctive descriptions may be required.***
- decision trees naturally represent disjunctive expressions.

- ***The training data may contain errors.***
- Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.
- ***The training data may contain missing attribute values.***
- Decision tree methods can be used even when some training examples have unknown values
- (e.g., if the ***Humidity*** of the day is known for only some of the training examples).

THE BASIC DECISION TREE LEARNING ALGORITHM

- The algorithm employs a top-down, greedy search through the space of possible decision trees.
- The basic algorithm begins with the question "Which attribute should be tested at the root of the tree?"
- Each instance attribute is evaluated using a statistical test to determine how well it classifies the training examples.
- The best attribute is selected and used as the test at the root node of the tree.

- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node.
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.

- **Which Attribute Is the Best Classifier?**
- Select the attribute that is most useful for classifying examples.
- A statistical property, called ***information gain*** *is used,* measures how well a given attribute separates the training examples according to their target classification.
- In order to define information gain, we define a measure commonly used in information theory, called ***entropy,*** that characterizes the (im)purity of an arbitrary collection of examples.

Information Gain and Entropy

- Entropy is a measure of any sort of uncertainty that is present in data.
- It can be measured by using the formula

Entropy measures the impurity or uncertainty present in the data.

$$H(S) = -\sum_{i=1}^{N} P_i \log_2 P_i$$

where,

S = Set of all instances in the dataset

N = Number of distinct class values

P_i = Event probability

- Information gain(IG) indicates how much information a particular feature or a particular variable gives us about final outcomes.

IG indicates how much "information" a particular feature / variable gives us about the final outcome

$$Gain(A,S) = H(S) - \sum_{j=1}^{v} \frac{|Sj|}{|S|} . H(Sj) = H(S) - H(A,S)$$

where:

- H(S) - entropy of the whole dataset S

- | Sj | - number of instance with j value of an attribute A

- | S | - total number of instances in dataset S

- v - set of distinct values of an attribute A

- H(Sj) - entropy of subset of instances for attribute A

- H(A, S) - entropy of an attribute A

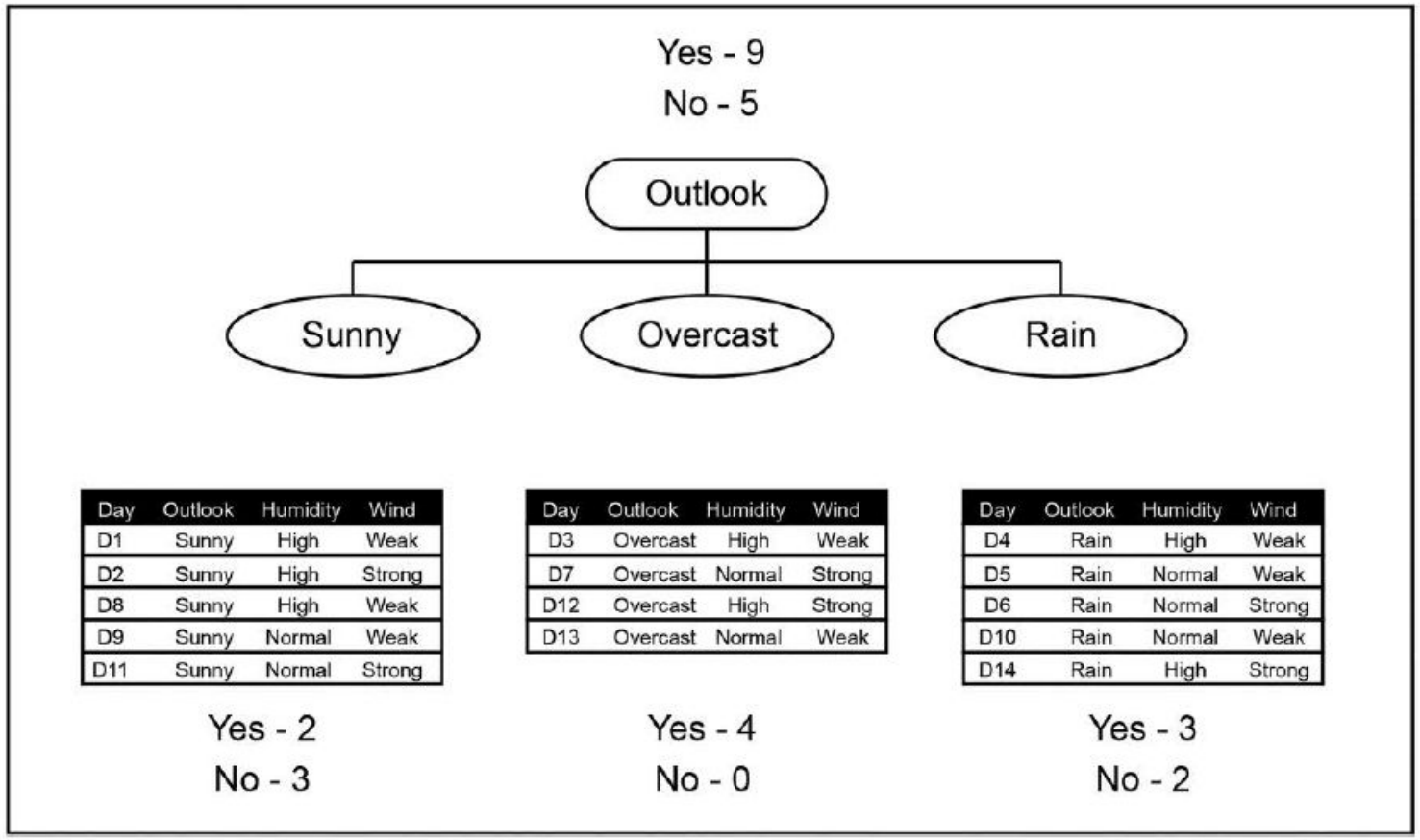
Example: predict whether match can be played or not, stating the weather conditions

- Predictor variables: outlook, humidity, and wind.
- Target variable is playing.
  - target variable is the variable we need to predict.
- Value of the target variable will decide, whether or not the game can be played.
  - The No prediction means that the weather condition is not good and therefore you cannot play the game, and
  - The Yes prediction means the game can be played,
  - The play has a value that is “yes” or “no”.
- To solve such a problem, make use of a decision tree.
- Consider an inverted tree and each branch of the tree denotes some decision.
- Each branch is known as a branch node, and at each branch, we need to decide in such a manner that you can get an outcome at the end of the branch.

Problem 1

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision tree for prediction.



Day	Outlook	Humidity	Wind
D1	Sunny	High	Weak
D2	Sunny	High	Strong
D8	Sunny	High	Weak
D9	Sunny	Normal	Weak
D11	Sunny	Normal	Strong

Day	Outlook	Humidity	Wind
D3	Overcast	High	Weak
D7	Overcast	Normal	Strong
D12	Overcast	High	Strong
D13	Overcast	Normal	Weak

Day	Outlook	Humidity	Wind
D4	Rain	High	Weak
D5	Rain	Normal	Weak
D6	Rain	Normal	Strong
D10	Rain	Normal	Weak
D14	Rain	High	Strong

- 14 observations, 9 observations result in a yes, meaning that out of 14 days, the match can be played on only 9 days
- Cluster dataset depending on the outlook
- The decision can be made as per outlook variable at the root node. So, the root node is the topmost node in a decision tree.
- The outlook node has three branches: sunny, overcast, and rainy.
- As sunny and rainy branches is mix of yes and no, will give impure output.
- The overcast variable, results in a 100% pure subject showing that the overcast variable will result in a definite and certain output. (perfect pure subset)
- This is the use of the entropy as a measure.
  - It calculates the impurity or the uncertainty.
  - Lesser the uncertainty or the entropy of a variable, the most significant is that variable

Note:

- Lesser the entropy of a particular variable, the most significant that variable will be
- Root node should have the most significant variable
- The overcast node is not a variable but the subset of outlook root node.
- Decision trees, information gain, and entropy, will help understand which variable will best split the dataset, or which variable is used to assign to the root node.

Find information gain and entropy

From the total of 14 instances we have

- 9 instances "yes"
- 5 instances "no"

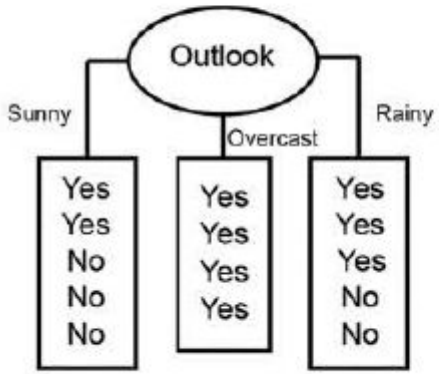
The entropy is :

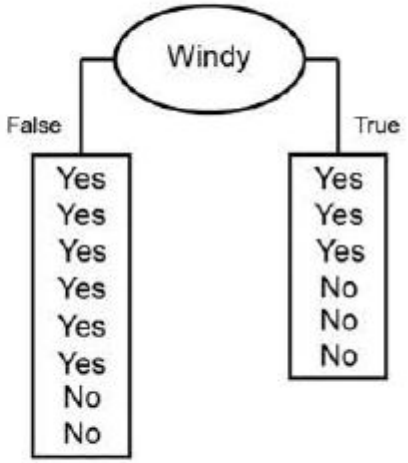
$$H(S) = -\sum_{i=1}^{N} P_i \log_2 P_i$$

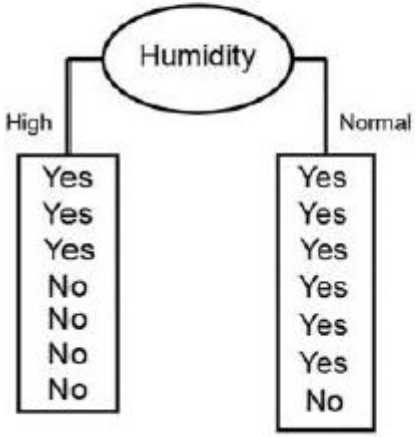
$$H(S) = -\frac{9}{14}\log_2\frac{9}{14} - \frac{5}{14}\log_2\frac{5}{14} = 0.940$$

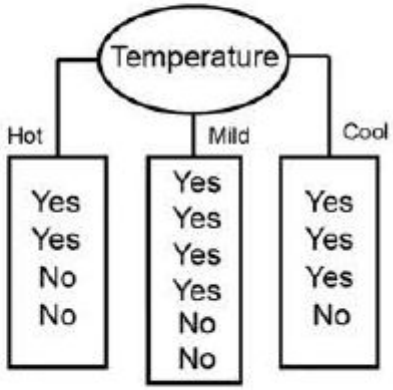
- This is the entropy and this is the uncertainty of the data present in the sample.

Selecting the root variable.









- The four variables are outlook, windy, humidity, or temperature.
- Can have any one of these variables as root node.
- How do you select the variable that best fits the root node?
  - Use information gain and entropy.
- Thus, the task is to find the information gain for each of these attributes, i.e., for outlook, windy, humidity, and temperature.
- The variable that results in the highest information gain must be chosen because it give most precise output information.

Information gain for the windy attribute.

- x

Information gain of attribute "Windy"

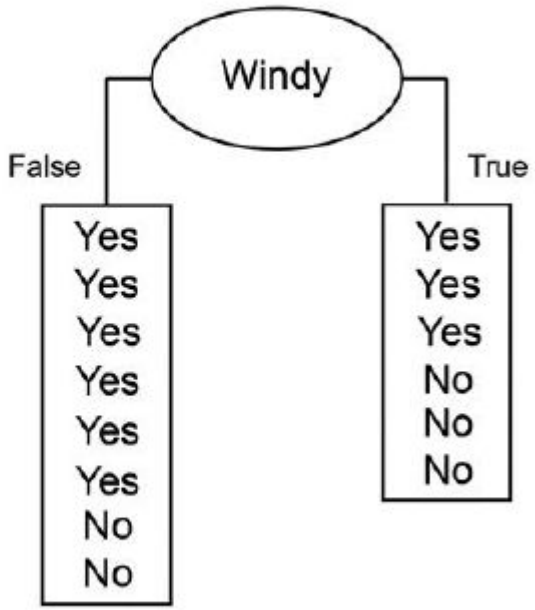
From the total of 14 instances we have :

- 6 instances of "ture"
- 8 instances of "ture"

$$Gain(A,S) = H(S) - \sum_{j=1}^{v} \frac{|Sj|}{|S|} - H(Sj)$$

$$Gain(A_{windy}, S) = 0.940 - \frac{8}{14} \cdot \left(-\left(\frac{6}{8} \cdot \log_2 \frac{6}{8} + \frac{2}{8} \cdot \log_2 \frac{2}{8}\right)\right) +$$

$$\frac{6}{14} \cdot \left(-\left(\frac{3}{6} \cdot \log_2 \frac{3}{6} + \frac{3}{6} \cdot \log_2 \frac{3}{6}\right)\right) = 0.048$$



Information gain for the outlook attribute.

- x

Information Gain of Attribute "Outlook"

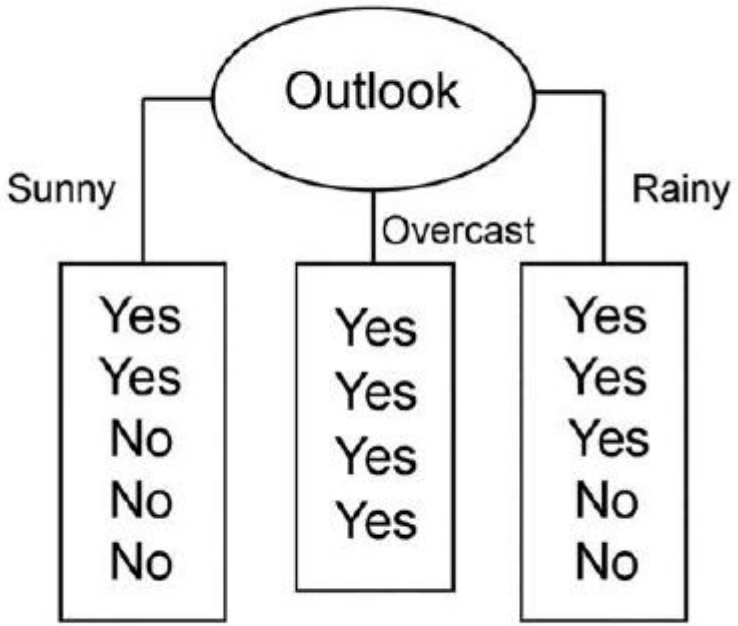
From the total of 14 instances we have :

- 5 instances "Sunny"
- 4 instances "Overcast"
- 5 instances "Rainy"

$$Gain(A_{outlook}, S) = 0.940 - \frac{5}{14}\cdot\left(-\left(\frac{2}{5}.\log_2\frac{2}{5}+\frac{3}{5}.\log_2\frac{3}{5}\right)\right) +$$

$$\frac{4}{14}\cdot\left(-\left(\frac{4}{4}.\log_2\frac{4}{4}\right)\right) +$$

$$\frac{5}{14}\cdot\left(-\left(\frac{3}{5}.\log_2\frac{3}{5}+\frac{2}{5}.\log_2\frac{2}{5}\right)\right) = 0.247$$



Information gain for the humidity attribute.

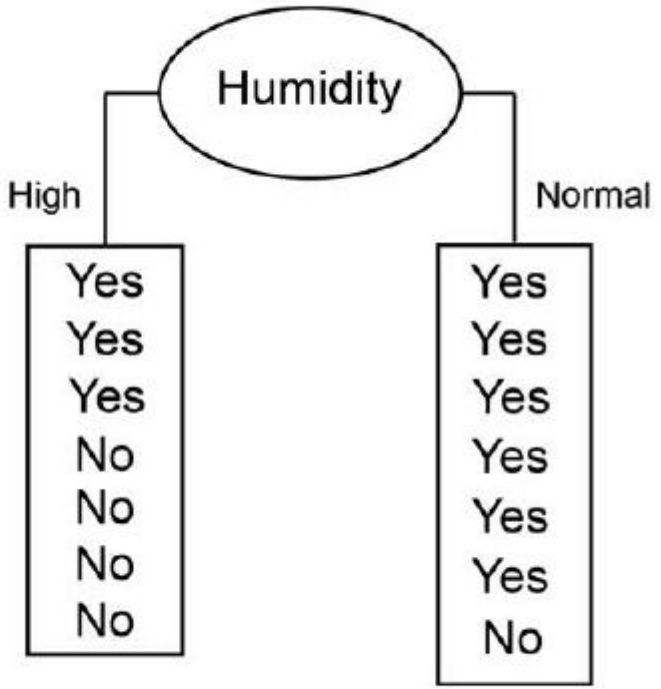
Information Gain of Attribute "Humidity"

From the total of 14 instances we have :

- 7 instances "High"
- 7 instances "Normal"

$$Gain(A_{Humidity}, S) = 0.940 - \frac{7}{14}\cdot\left(-\left(\frac{3}{7}\cdot\log_2\frac{3}{7}+\frac{4}{7}\cdot\log_2\frac{4}{7}\right)\right) +$$

$$\frac{7}{14}\cdot\left(-\left(\frac{6}{7}\cdot\log_2\frac{6}{7}+\frac{1}{7}\cdot\log_2\frac{1}{7}\right)\right) = 0.151$$



Information gain for the temperature attribute.

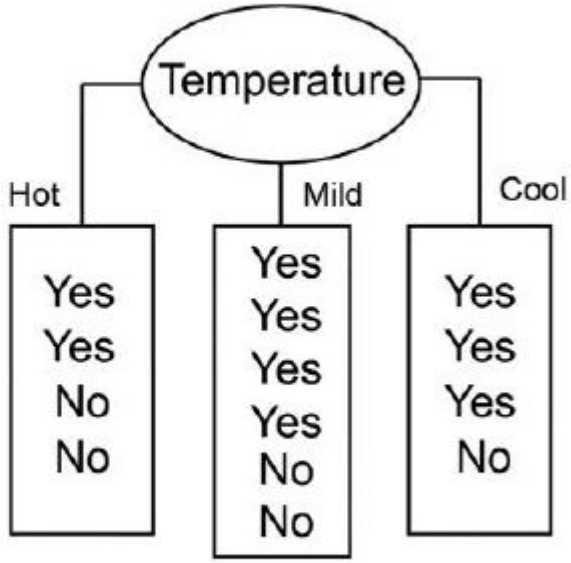
- x

Information Gain of Attribute "Temperature"

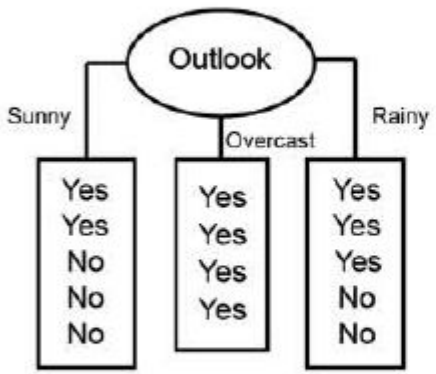
From the total of 14 instances we have :

- 4 instances "Hot"
- 6 instances "Mild"
- 4 instances "Cool"

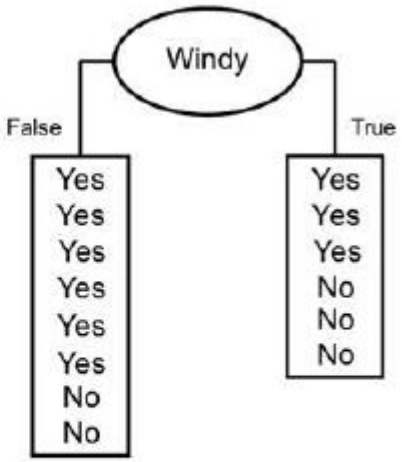
$$Gain(A_{Temperature}, S) = 0.940 - \frac{4}{14}\cdot\left(-\left(\frac{2}{4}\cdot\log_2\frac{2}{4}+\frac{2}{4}\cdot\log_2\frac{2}{4}\right)\right) + \frac{6}{14}\cdot\left(-\left(\frac{4}{6}\cdot\log_2\frac{4}{6}+\frac{2}{6}\cdot\log_2\frac{2}{6}\right)\right) + \frac{4}{14}\cdot\left(-\left(\frac{3}{4}\cdot\log_2\frac{3}{4}+\frac{1}{4}\cdot\log_2\frac{1}{4}\right)\right) = 0.029$$



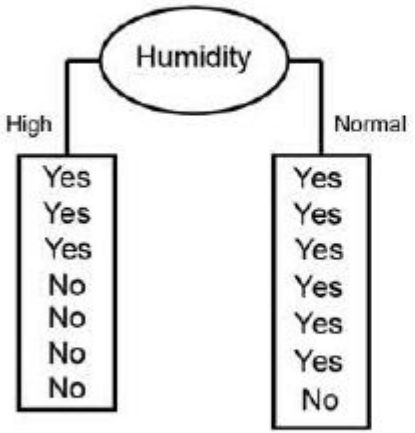
Information gain for all attributes.



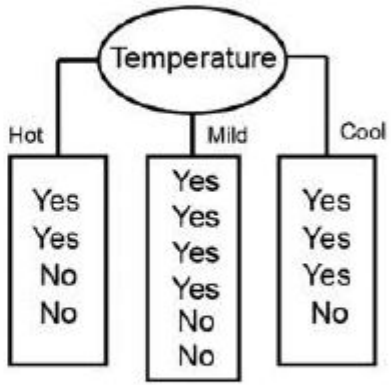
Gain = 0.247



Gain = 0.048



Gain = 0.151

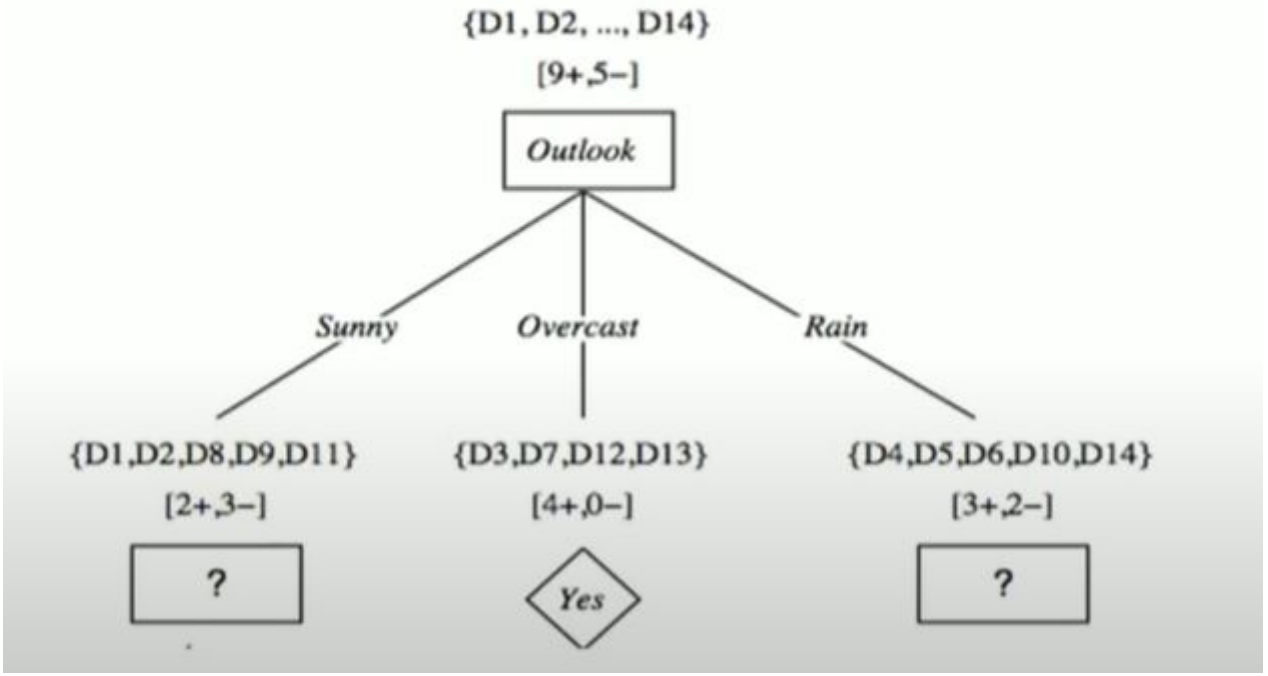


Gain = 0.029

Tree

$$Gain(S, Outlook) = 0.2464,\ Gain(S, Temp) = 0.0289$$

$$Gain(S, Humidity) = 0.1516,\ Gain(S, Wind) = 0.0478$$



Day	Temp	Humidity	Wind	Play Tennis
D1	Hot	High	Weak	No
D2	Hot	High	Strong	No
D8	Mild	High	Weak	No
D9	Cool	Normal	Weak	Yes
D11	Mild	Normal	Strong	Yes

Attribute: Temp

$Values\ (Temp) = Hot, Mild, Cool$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S_{Sunny}) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Hot} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Hot}) = 0.0$

$S_{Mild} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Mild}) = 1.0$

$S_{Cool} \leftarrow [1+, 0-]$ $\quad Entropy(S_{Cool}) = 0.0$

$$Gain\ (S_{Sunny}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Temp)$$

$$= Entropy(S) - \frac{2}{5}Entropy(S_{Hot}) - \frac{2}{5}Entropy(S_{Mild})$$

$$- \frac{1}{5}Entropy(S_{Cool})$$

$$Gain(S_{sunny}, Temp) = 0.97 - \frac{2}{5}0.0 - \frac{2}{5}1 - \frac{1}{5}0.0 = 0.570$$

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{high} \leftarrow [0+, 3-]$ $\quad Entropy(S_{High}) = 0.0$

$S_{Normal} \leftarrow [2+, 0-]$ $\quad Entropy(S_{Normal}) = 0.0$

$$Gain\ (S_{Sunny}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Sunny}, Humidity) = Entropy(S) - \frac{3}{5}Entropy(S_{High}) - \frac{2}{5}Entropy(S_{Normal})$$

$$Gain(S_{sunny}, Humidity) = 0.97 - \frac{3}{5}\,0.0 - \frac{2}{5}0.0 = 0.97$$

Attribute: Wind

$Values\ (Wind) = Strong, Weak$

$S_{Sunny} = [2+, 3-]$ $\quad Entropy(S) = -\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5} = 0.97$

$S_{Strong} \leftarrow [1+, 1-]$ $\quad Entropy(S_{Strong}) = 1.0$

$S_{Weak} \leftarrow [1+, 2-]$ $\quad Entropy(S_{Weak}) = -\frac{1}{3}log_2\frac{1}{3} - \frac{2}{3}log_2\frac{2}{3} = 0.9183$

$$Gain\ (S_{Sunny}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

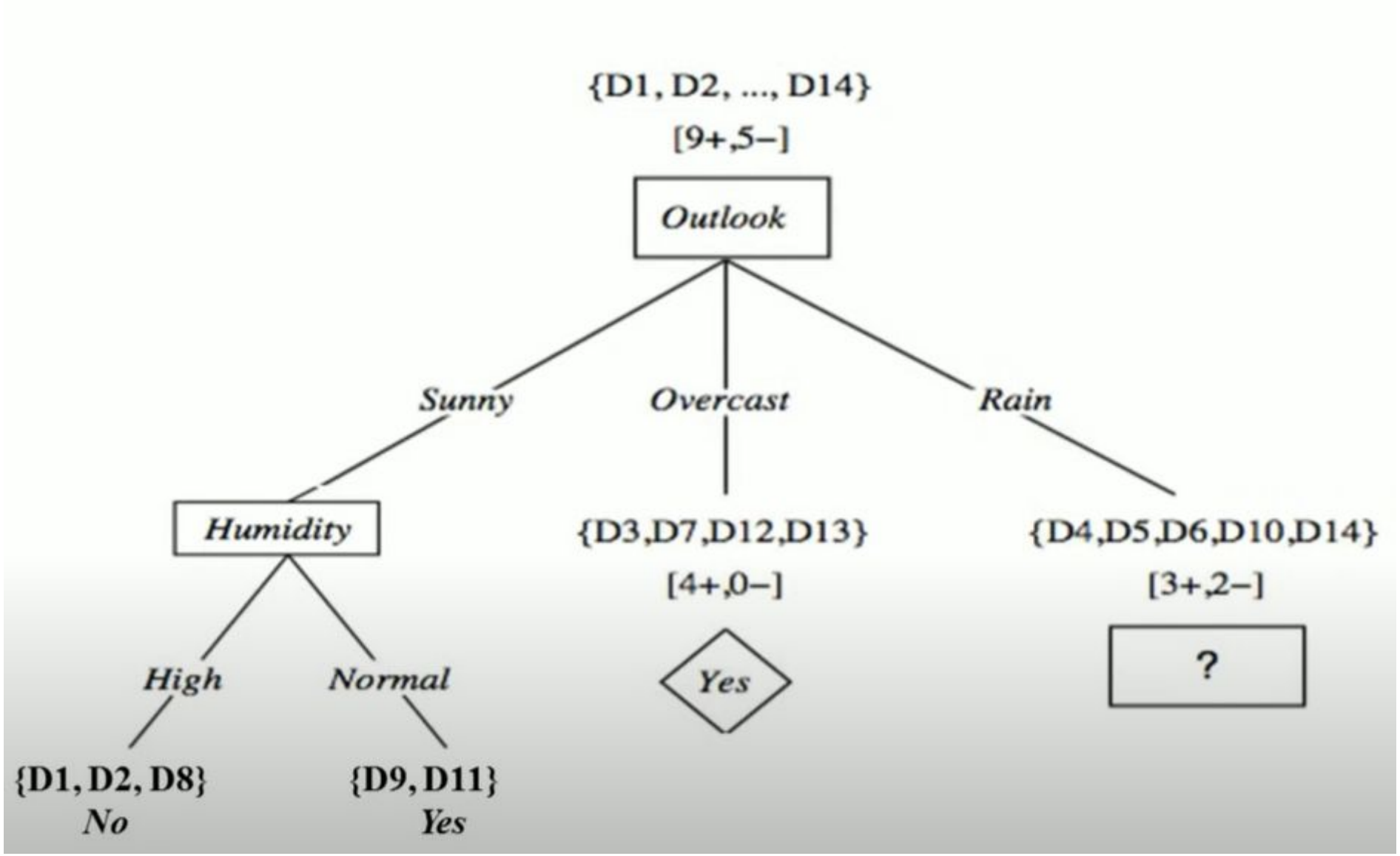
$$Gain(S_{Sunny}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{sunny}, Wind) = 0.97 - \frac{2}{5}1.0 - \frac{3}{5}0.918 = 0.0192$$

$$Gain(S_{sunny}, Temp) = 0.570$$

$$Gain(S_{sunny}, Humidity) = 0.97$$

$$Gain(S_{sunny}, Wind) = 0.0192$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
{D3,D7,D12,D13}
[4+,0−]
{D4,D5,D6,D10,D14}
[3+,2−]
High
Normal
Yes
?
{D1, D2, D8}
No
{D9, D11}
Yes

Day	Temp	Humidity	Wind	Play Tennis
D4	Mild	High	Weak	Yes
D5	Cool	Normal	Weak	Yes
D6	Cool	Normal	Strong	No
D10	Mild	Normal	Weak	Yes
D14	Mild	High	Strong	No

Attribute: Temp

$$Values\,(Temp) = Hot, Mild, Cool$$

$$S_{Rain} = [3+, 2-] \qquad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$$

$$S_{Hot} \leftarrow [0+, 0-] \qquad Entropy(S_{Hot}) = 0.0$$

$$S_{Mild} \leftarrow [2+, 1-] \qquad Entropy(S_{Mild}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$$

$$S_{Cool} \leftarrow [1+, 1-] \qquad Entropy(S_{Cool}) = 1.0$$

$$Gain\,(S_{Rain}, Temp) = Entropy(S) - \sum_{v \in \{Hot, Mild, Cool\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Temp)$$

$$= Entropy(S) - \frac{0}{5}Entropy(S_{Hot}) - \frac{3}{5}Entropy(S_{Mild})$$

$$- \frac{2}{5}Entropy(S_{Cool})$$

$$Gain(S_{Rain}, Temp) = 0.97 - \frac{0}{5}0.0 - \frac{3}{5}0.918 - \frac{2}{5}1.0 = 0.0192$$

Attribute: Humidity

$Values\ (Humidity) = High, Normal$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{High} \leftarrow [1+, 1-]$ $\quad Entropy(S_{High}) = 1.0$

$S_{Normal} \leftarrow [2+, 1-]$ $\quad Entropy(S_{Normal}) = -\frac{2}{3}log_2\frac{2}{3} - \frac{1}{3}log_2\frac{1}{3} = 0.9183$

$$Gain\ (S_{Rain}, Humidity) = Entropy(S) - \sum_{v \in \{High, Normal\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S_{Rain}, Humidity) = Entropy(S) - \frac{2}{5}Entropy(S_{High}) - \frac{3}{5}Entropy(S_{Normal})$$

$$Gain(S_{Rain}, Humidity) = 0.97 - \frac{2}{5}\ 1.0 - \frac{3}{5}0.918 = 0.0192$$

Attribute: Wind

$Values\ (wind) = Strong, Weak$

$S_{Rain} = [3+, 2-]$ $\quad Entropy(S_{Sunny}) = -\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5} = 0.97$

$S_{Strong} \leftarrow [0+, 2-]$ $\quad Entropy(S_{Strong}) = 0.0$

$S_{Weak} \leftarrow [3+, 0-]$ $\quad Entropy(S_{weak}) = 0.0$

$$Gain\ (S_{Rain}, Wind) = Entropy(S) - \sum_{v \in \{Strong, Weak\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

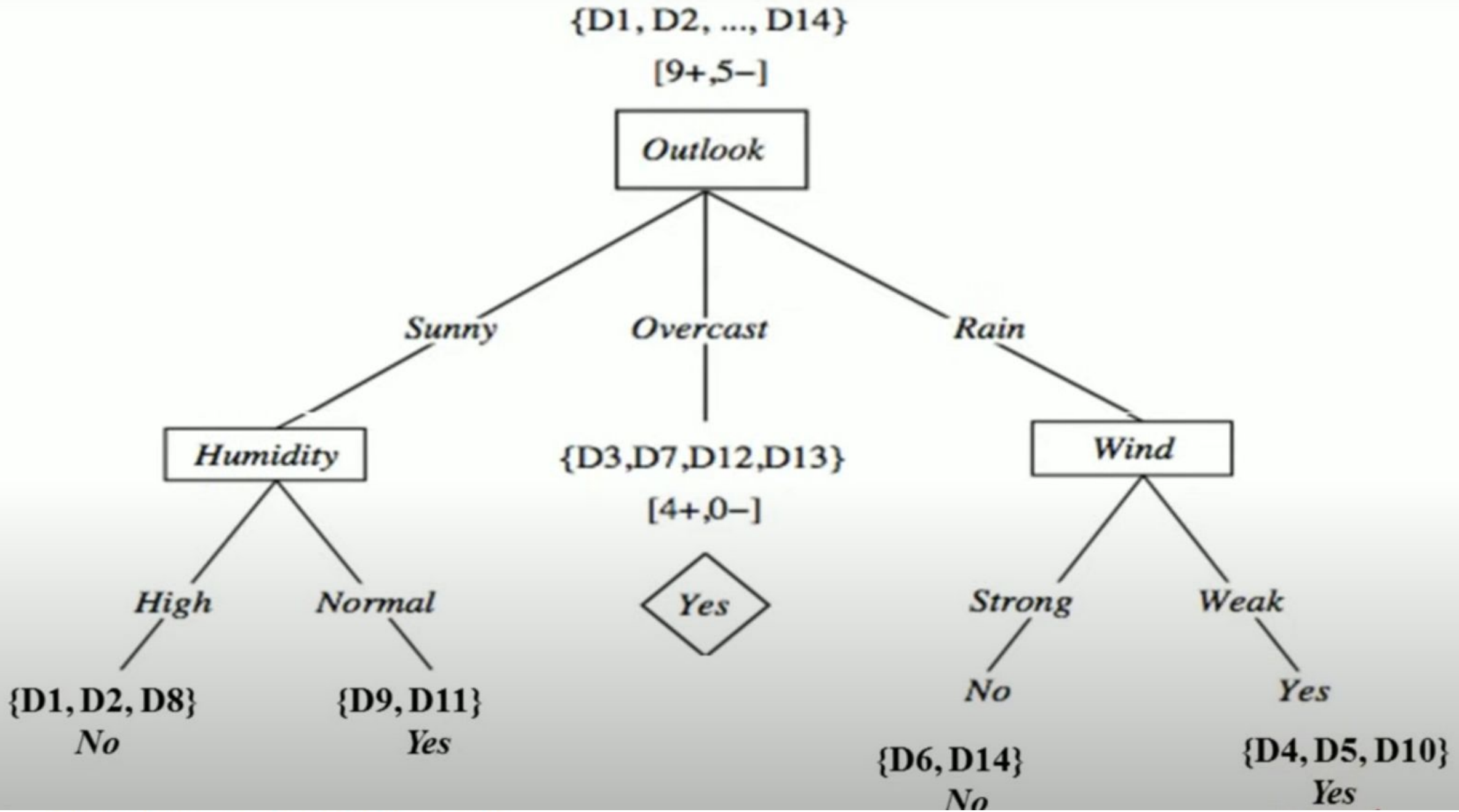
$$Gain(S_{Rain}, Wind) = Entropy(S) - \frac{2}{5}Entropy(S_{Strong}) - \frac{3}{5}Entropy(S_{Weak})$$

$$Gain(S_{Rain}, Wind) = 0.97 - \frac{2}{5}\ 0.0 - \frac{3}{5}0.0 = 0.97$$

$$Gain(S_{Rain}, Temp) = 0.0192$$

$$Gain(S_{Rain}, Humidity) = 0.0192$$

$$Gain(S_{Rain}, Wind) = 0.97$$


{D1, D2, ..., D14}
[9+,5−]
Outlook
Sunny
Overcast
Rain
Humidity
{D3,D7,D12,D13}
[4+,0−]
Wind
High
Normal
Yes
Strong
Weak
{D1, D2, D8}
No
{D9, D11}
Yes
No
Yes
{D6, D14}
No
{D4, D5, D10}
Yes

Problem 2: Draw a decision tree for the following training examples

Instance	a1	a2	a3	Classification
1	True	Hot	High	No
2	True	Hot	High	No
3	False	Hot	High	Yes
4	False	Cool	Normal	Yes
5	False	Cool	Normal	Yes
6	True	Cool	High	No
7	True	Hot	High	No
8	True	Hot	Normal	Yes
9	False	Cool	Normal	Yes
10	False	Cool	High	Yes

Problem 3: Draw a decision tree for the following data

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

ID3(*Examples*, *Target_attribute*, *Attributes*)

Examples are the training examples. Target_attribute is the attribute whose value is to be predicted by the tree. Attributes is a list of other attributes that may be tested by the learned decision tree. Returns a decision tree that correctly classifies the given Examples.

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
    - $A \leftarrow$ the attribute from *Attributes* that best* classifies *Examples*
    - The decision attribute for *Root* $\leftarrow A$
    - For each possible value, v_i, of A,
        - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
        - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for A
        - If $Examples_{v_i}$ is empty
            - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
            - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {A}))
- End
- Return *Root*

Day	Outlook	Temp	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

- ID3 is an recursive algorithm.
- Three parameters:
  - Examples are training examples: 14 cases
  - Target_Attribute: Play Tennis
  - Attributes: Outlook, temperature, humidity, wind

ISSUES IN DECISION TREE LEARNING

1. Avoiding Overfitting the Data
    1. REDUCED ERROR PRUNING
    2. RULE POST-PRUNING
2. Incorporating Continuous-Valued Attributes
3. Alternative Measures for Selecting Attributes
4. Handling Training Examples with Missing Attribute Values
5. Handling Attributes with Differing Costs

Avoiding Overfitting the Data

- The ID3 algorithm grows each branch of the tree to perfectly classify the training examples.
  - It can lead to difficulties when
    a. there is noise in the data
    b. the number of training examples is too small to produce a representative sample of the true target function.
- This can produce trees that ***overfit*** the training examples.

Overfitting definition:

- **Given a hypothesis space H,**

❖ **a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$,**

❖ **such that h has a smaller error than h' over the training examples,**

❖ **but h' has a smaller error than h over the entire distribution of instances.**

- How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?
- Can occur when the training examples contain random errors or noise.
- Consider the effect of adding the following positive training example, incorrectly labelled as negative
- ***(Outlook** = **Sunny, Temperature** = **Hot, Humidity** = **Normal, Wind** = **Strong, PlayTennis** = **No)***

- The result is that ID3 will output a decision tree (h) that is more complex than the original tree from (h').
- h will fit the collection of training examples perfectly, whereas h' will not.
- However, given that the new decision node is simply a consequence of fitting the noisy training example, we expect h to outperform h' over subsequent data drawn from the same instance distribution.
- Overfitting decreases the accuracy of learned decision trees

Approaches to avoiding overfitting in decision tree learning

- These can be grouped into two classes:

1. approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
2. approaches that allow the tree to overfit the data, and then post-prune the tree.

Criterion used to determine the correct final tree size.

1. Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree referred to as a training and validation set approach.

- the available data are separated into two sets of examples:

a. a training set used to form the learned hypothesis

b. a validation set used to evaluate the accuracy of this hypothesis over subsequent data and to evaluate the impact of pruning this hypothesis

- Even though the learner may be misled by random errors, the validation set is unlikely to exhibit the same random fluctuations.
- The validation provides a safety check against overfitting the spurious characteristics of the training set

2. Use all the available data for training
   - apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set.
   - For example, chi-square test to estimate whether further expanding a node is likely to improve performance.
3. Use an explicit measure of the complexity for encoding the training examples and the decision tree, halting the growth of the tree when this encoding size is minimized.
   - This approach is based on a heuristic called the Minimum Description Length principle.

- **REDUCED ERROR PRUNING**
- Consider each of the decision nodes in the tree to be candidates for pruning.
- Pruning a decision node consists of
  a. removing the subtree rooted at that node,
  b. making it a leaf node, and
  c. assigning it the most common classification of the training examples affiliated with that node.
- Nodes are removed only if the resulting pruned tree performs as the original over the validation set.
- Any leaf node added due to coincidental regularities in the training set is pruned because these coincidences are unlikely to occur in the validation set.
- Nodes are pruned iteratively: always choose a node whose removal increases the decision tree accuracy over the validation set.
- Pruning of nodes continues until it decreases the accuracy of the tree over the validation set.

- As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- Drawback:
- When data is limited, withholding part of it for the validation set reduces even further the number of examples available for training

- **RULE POST-PRUNING**
- Method for finding high-accuracy hypotheses
- Rule post-pruning involves the following steps:

1. Infer the decision tree from the training set: grow the tree until the training data is fit and allowing overfitting to occur.
2. Convert the learned tree into an equivalent set of rules by creating one rule for each path from the root node to a leaf node.
3. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
4. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

- Consider the decision tree in the example.
- In rule post-pruning, one rule is generated for each leaf node in the tree.
- **Each attribute test along the path from the root to the leaf becomes a rule antecedent (precondition) and the classification at the leaf node becomes the rule consequent (post condition).**
- For example,
- IF (Outlook = Sunny) $\wedge$ (Humidity = High)
- THEN PlayTennis = No
- Next, each rule is pruned by removing any antecedent, or precondition, whose removal does not worsen its estimated accuracy.
- For example, rule post-pruning would consider removing the preconditions
- (Outlook = Sunny) and (Humidity = High).
- No pruning step is performed if it reduces the estimated rule accuracy.

Why convert the decision tree to rules before pruning? There are three main advantages.

1. Converting to rules allows distinguishing among the different contexts in which a decision node is used.

- Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.

2. Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves.

- bookkeeping issues is avoided such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.

3. Converting to rules improves readability.

- Rules are often easier for people to understand.

Incorporating Continuous-Valued Attributes

- ID3 is restricted to attributes that take on a discrete set of values.
  1. The target attribute
  2. the attributes tested in the decision nodes
- To incorporate continuous-valued decision attributes into the learned tree, second restriction can easily be removed
- by dynamically defining new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals for an attribute A that is continuous-valued,
- the algorithm can dynamically create a new Boolean attribute A, that is true if A < c and false otherwise

- Suppose: to include the continuous-valued attribute
- ***Temperature*** in describing the training example days in the learning task

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

- What threshold-based boolean attribute should be defined based on Temperature?

- pick a threshold, ***c,*** that produces the greatest information gain.

a) Sort the examples according to the continuous attribute **A,**

b) then identify adjacent examples that differ in their target classification,

c) a set of candidate thresholds midway between the corresponding values of ***A is generated.***

- The value of ***c*** that maximizes information gain must always lie at such a boundary.
- Candidate thresholds can then be evaluated by computing the information gain associated with each.
- In example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes:
- (48 + 60)/2, and (80 + 90)/2.
- The information gain can then be computed for each of the candidate attributes, Temperature $_{>54}$ and Temperature $_{>85}$, and the best can be selected (Temperature $_{>54}$)
- This dynamically created boolean attribute can then compete with the other discrete-valued candidate attributes available for growing the decision tree.

Alternative Measures for Selecting Attributes

- There is a natural bias in the information gain measure that favors attributes with many values over those with few values.
- Select decision attributes based on some measure other than information gain.
- Alternative measure: Gain ratio
- The gain ratio measure, penalizes attributes such as Date by incorporating a term, called split information,
- Split information is sensitive to how broadly and uniformly the attribute splits the data:

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- where S_i through S, are the ***c*** subsets of examples resulting from partitioning S by the c-valued attribute A.
- Split information is actually the entropy of S with respect to the values of attribute A.

- The Gain Ratio measure is defined in terms of the earlier Gain measure, as well as Split information, as follows

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

Handling Training Examples with Missing Attribute Values

- In certain cases, the available data may be missing values for some attributes.
- Example: in a medical domain to predict patient outcome based on various laboratory tests, the lab test Blood-Test-Result is available only for a subset of the patients.
- In such cases, it is common to estimate the missing attribute value based on other examples for which this attribute has a known value.

- Strategy for dealing with the missing attribute value:

1. Assign it the is most common value among training examples A.
2. More complex procedure is to assign a probability to each of the possible values of *A*.

Handling Attributes with Differing Costs

- In some learning tasks the instance attributes may have associated costs.
- Example: in learning to classify medical diseases we might describe patients in terms of attributes such **as** Temperature, BiopsyResult, Pulse, BloodTestResults, etc.
- These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort.
- In such tasks,

i. prefer decision trees that use low-cost attributes where possible,

ii. Rely on high-cost attributes only when needed to produce reliable classifications.

- ID3 can be modified to take into account attribute costs by introducing a cost term into the attribute selection measure.
- Example: we might divide the ***Gain*** by the cost of the attribute, so that lower-cost attributes would be preferred.
- Such cost-sensitive measures do not guarantee finding an optimal cost-sensitive decision tree
  - BUT they do bias the search in favor of low-cost attributes.

- Example: An approach was described and applied to a robot perception task
  - the robot must learn to classify different objects according to how they can be grasped by the robot's manipulator.
- The attributes correspond to different sensor readings obtained by a movable sonar on the robot.
- Attribute cost is measured by the number of seconds required to obtain the attribute value by positioning and operating the sonar.
- Conclusion: more efficient recognition strategies are learned, without sacrificing classification accuracy, by replacing the information gain attribute selection measure by the following measure

$$\frac{Gain^2(S, A)}{Cost(A)}$$

- Another approach describes application to learning medical diagnosis rules.
- Here the attributes are different symptoms and laboratory tests with differing costs.

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^{w}}$$

- where $w \in [0, 11]$ is a constant that determines the relative importance of cost versus information gain.

CART (Classification And Regression Tree)Algorithm

- CART algorithm, produces only *binary trees*: nonleaf nodes always have two children (i.e., questions only have yes/no answers).
- However, other algorithms such as ID3 can produce Decision Trees with nodes that have more than two children.
- The algorithm first splits the training set in two subsets using a single feature k and a threshold t_k
- How does it choose k and t_k?
- It searches for the pair (k, t_k) that produces the purest subsets (weighted by their size).
- The cost function that the algorithm tries to minimize is given by

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

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

- Once it has successfully split the training set in two, it splits the subsets using the same logic, then the sub-subsets and so on, recursively.
- It stops recursing once it reaches the maximum depth, or if it cannot find a split that will reduce impurity
- The CART algorithm is a *greedy algorithm*:
- It greedily searches for an optimum split at the top level, then repeats the process at each level.
- It does not check whether or not the split will lead to the lowest possible impurity several levels down.
- A greedy algorithm often produces a reasonably good solution, but it is not guaranteed to be the optimal solution.

- The CART used Gini index for classifying the decision points for the given dataset and applied splitting rule that improves the performance of CART classifier algorithm which results in an optimal tree.

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

- where p_i is the probability of an object being classified to a particular class.

- CART(Classification And Regression Trees) is a variation of the decision tree algorithm.
- It can handle both classification and regression tasks.
- The term CART serves as a generic term for the following categories of decision trees:

1. Classification Trees: The tree is used to determine which “class” the target variable is most likely to fall into when it is continuous.
2. Regression trees: These are used to predict a continuous variable’s value.

CART Algorithm

- 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 uses Gini Impurity to split the dataset into a decision tree .
- It does that by searching for the best homogeneity for the sub nodes, with the help of the Gini index criterion.

Gini index/Gini impurity

- The Gini index is a metric for the classification tasks in CART.
- It stores the sum of squared probabilities of each class.
- It computes the degree of probability of a specific variable that is wrongly being classified when chosen randomly and a variation of the Gini coefficient.
- It works on categorical variables, provides outcomes either "successful" or "failure" and hence conducts binary splitting only.

- The degree of the Gini index varies from 0 to 1,

❑ 0 depicts that all the elements are allied to a certain class, or only one class exists there.

❑ 1 signifies that all the elements are randomly distributed across various classes, and

❑ A value of 0.5 denotes the elements are uniformly distributed into some classes.

- Mathematically, we can write Gini Impurity as follows:

$$Gini = 1 - \sum_{i=1}^{n} (p_i)^2$$

- where p_i is the probability of an object being classified to a particular class.

- **Advantages of CART**

1. Results are simplistic.
2. Classification and regression trees are Nonparametric and Nonlinear.
3. Classification and regression trees implicitly perform feature selection.
4. Outliers have no meaningful effect on CART.
5. It requires minimal supervision and produces easy-to-understand models.

- **Limitations of CART**

1. Overfitting.
2. High Variance.
3. low bias.
4. the tree structure may be unstable.

- **Applications of the CART algorithm**

1. For quick Data insights.
2. In Blood Donors Classification.
3. For environmental and ecological data.
4. In the financial sectors.

Problem Solving for CART Algorithm

1. Calculate the Gini index for the data set. The target attribute job offer has 7 instances of Yes and 3 instances of No.
2. Compute Gini index for each of the attribute and each of the subset in the attribute.
3. Choose the best splitting subset which has minimum Gini index for an attribute
4. Compute ΔGini or best splitting subset of that attribute.

Construct a decision tree using CART Algorithm

CGPA	Inter active	Practical Knowledge	Comm Skills	Job Offer
>=9	Yes	Very good	Good	Yes
>=8	No	Good	Moderate	Yes
>=9	No	Average	Poor	No
<8	No	Average	Good	No
>=8	Yes	Good	Moderate	Yes
>=9	Yes	Good	Moderate	Yes
<8	Yes	Good	Poor	No
>=9	No	Very good	Good	Yes
>=8	Yes	Good	Good	Yes
>=8	Yes	Average	Good	Yes

$$\text{Gini\_Index}(T) = 1 - \sum_{i=1}^{m} P_i^2$$

$$\text{Gini\_Index}(T, A) = \frac{|S_1|}{|T|}\text{Gini}(S_1) + \frac{|S_2|}{|T|}\text{Gini}(S_2)$$

REMAINING 8 EXAMPLES (problem continued)

CGPA	Inter active	Practical Knowledge	Comm Skills	Job Offer
>=9	Yes	Very good	Good	Yes
>=8	No	Good	Moderate	Yes
>=9	No	Average	Poor	No
>=8	Yes	Good	Moderate	Yes
>=9	Yes	Good	Moderate	Yes
>=9	No	Very good	Good	Yes
>=8	Yes	Good	Good	Yes
>=8	Yes	Average	Good	Yes

Interactiveness	Job Offer = Yes	Job Offer = No
Yes	5	0
No	2	1

$$\text{Gini\_Index}(T, \text{Interactiveness} \in \{\text{Yes}\}) = 1 - \left(\frac{5}{5}\right)^2 - \left(\frac{0}{5}\right)^2$$
$$= 1 - 1 = 0$$

$$\text{Gini\_Index}(T, \text{Interactiveness} \in \{\text{No}\}) = 1 - \left(\frac{2}{3}\right)^2 - \left(\frac{1}{3}\right)^2$$
$$= 1 - 0.44 - 0.111 = 0.449$$

$$\text{Gini\_Index}(T, \text{Interactiveness} \in \{\text{Yes, No}\}) = \left(\frac{7}{8}\right) \times 0 + \left(\frac{1}{8}\right) \times 0.449$$
$$= 0.056$$

$$\Delta\text{Gini}(\text{Interactiveness}) =$$
$$\text{Gini}(T) - \text{Gini}(T, \text{Interactiveness})$$
$$= 0.2184 - 0.056 = 0.1624$$

'Gini_Index for Subsets of Practical Knowledge

Subsets		Gini_Index
(Very Good, Good)	Average	0.125
(Very Good, Average)	Good	0.1875
(Good, Average)	Very Good	0.2085

$$\Delta\text{Gini}(\text{Practical Knowledge}) = \text{Gini}(T) - \text{Gini}(T, \text{Practical Knowledge})$$
$$= 0.2184 - 0.125$$
$$= 0.0934$$

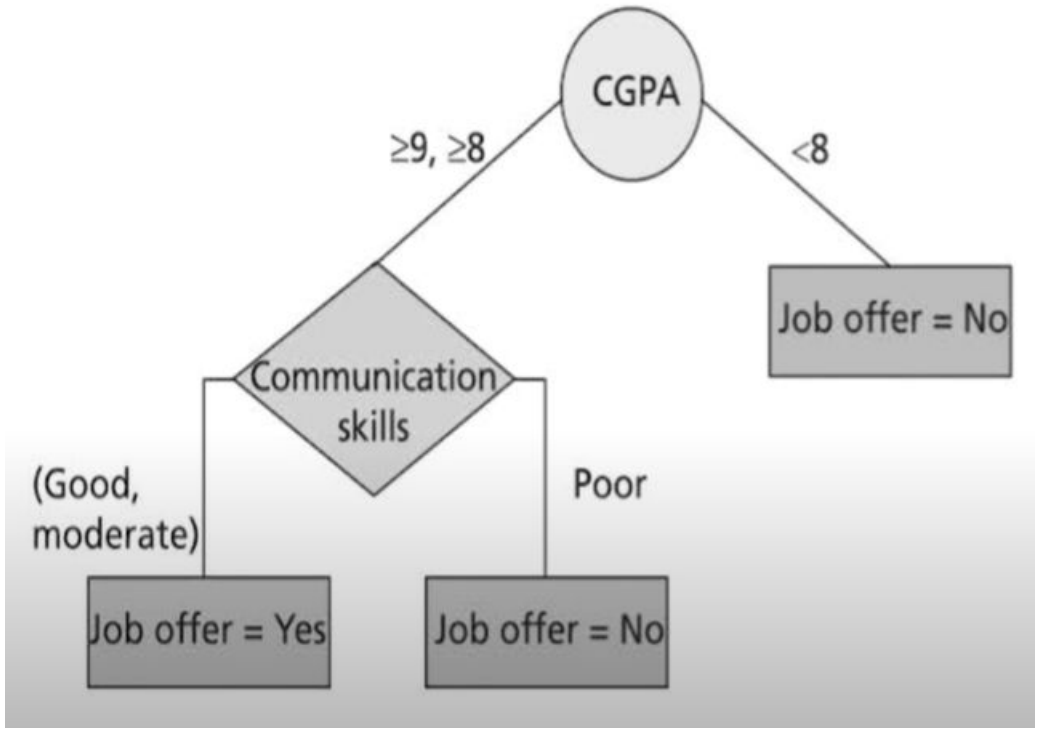
Gini_Index for Subsets of Communication Skills

Subsets		Gini_Index
(Good, Moderate)	Poor	0
(Good, Poor)	Moderate	0.2
(Moderate, Poor)	Good	0.1875

$$\Delta\text{Gini}(\text{Communication Skills}) = \text{Gini}(T) - \text{Gini}(T, \text{Communication Skills})$$
$$= 0.2184 - 0 = 0.2184$$

Gini_Index and ΔGini Values for All Attributes

Attribute	Gini_Index	ΔGini
Interactiveness	0.056	0.1624
Practical knowledge	0.125	0.0934
Communication Skills	0	0.2184


CGPA
≥9, ≥8
<8
Job offer = No
Communication skills
(Good, moderate)
Poor
Job offer = Yes
Job offer = No

Problem, 2

Weekend	Weather	Parents	Money	Decision
W1	Sunny	Yes	Rich	Cinema
W2	Sunny	No	Rich	Tennis
W3	Windy	Yes	Rich	Cinema
W4	Rainy	Yes	Poor	Cinema
W5	Rainy	No	Rich	Stay In
W6	Rainy	Yes	Poor	Cinema
W7	Windy	No	Poor	Cinema
W8	Windy	No	Rich	Shopping
W9	Windy	Yes	Rich	Cinema
W10	Sunny	No	Rich	Tennis


B.M.S. COLLEGE OF ENGINEERING
BANGALORE - 560 019