MODULE 1

Introduction: The Machine Learning Landscape

What is Machine learning?

Machine Learning is the science and art of programming computers so they can learn from data.

- **Arthur Samuel, 1959:** Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.
- **Tom Mitchell, 1997:** 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.

Example: **Spam filter**

- It is a Machine Learning program that can learn to flag spam given examples of spam emails (flagged by users) and examples of regular emails.
- The examples that the system uses to learn are called the training set. Each training example is called a training instance (or sample).
- In this case, the task T is to flag spam for new emails, the experience E is the training data, and the performance measure P needs to be defined; for example, you can use 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?

Machine Learning is used 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

Example 1: Spam filtering

Consider a spam filter program which is written using traditional programming techniques (Figure 1-1):

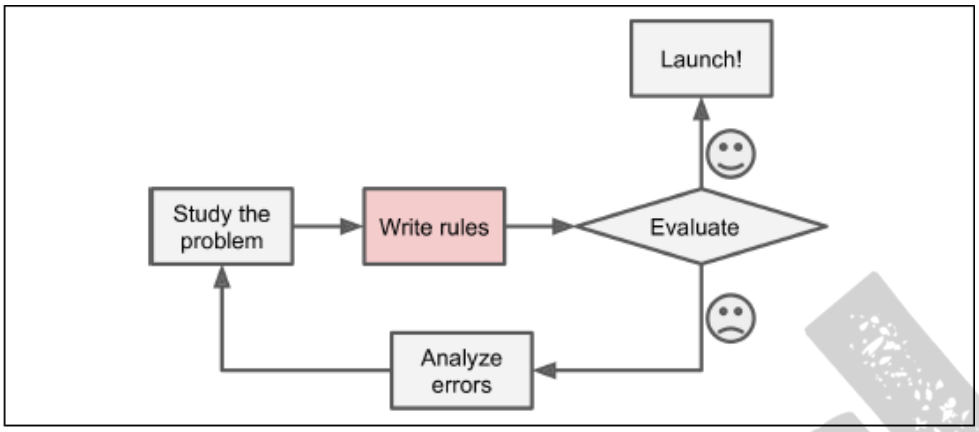


Figure 1-1. The traditional approach

1. First look at what spam typically looks like. Some words or phrases (such as "4U," "credit card," "free," and "amazing") tend to come up a lot in the subject are noticed. Few other patterns in the sender's name, the email's body, and so on are also noticed.
2. Then write a detection algorithm for each of the patterns that noticed, and the program would flag emails as spam if a number of these patterns are detected.
3. Next, test the program, and repeat steps 1 and 2 until it is good enough.

Since the problem is not trivial, the program will likely become a long list of complex rules and 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 unusually frequent patterns of words in the spam examples compared to the ham examples (Figure 1-2). The program is much shorter, easier to maintain, and most likely more accurate.

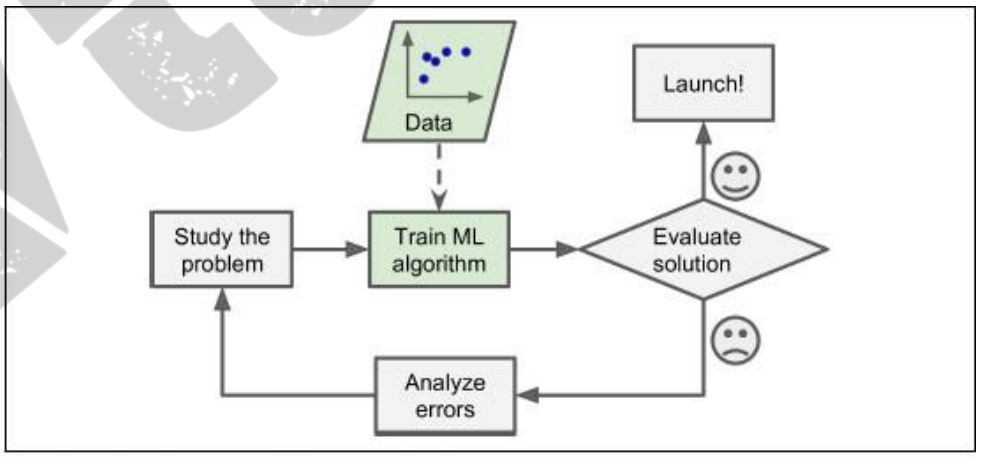


Figure 1-2. Machine Learning approach

A machine learning-based spam filter adapts dynamically to evolving spam tactics, recognizing patterns like "For U" without manual rule updates. This proactive approach saves time and effort, ensuring continuous effectiveness against emerging spam techniques.

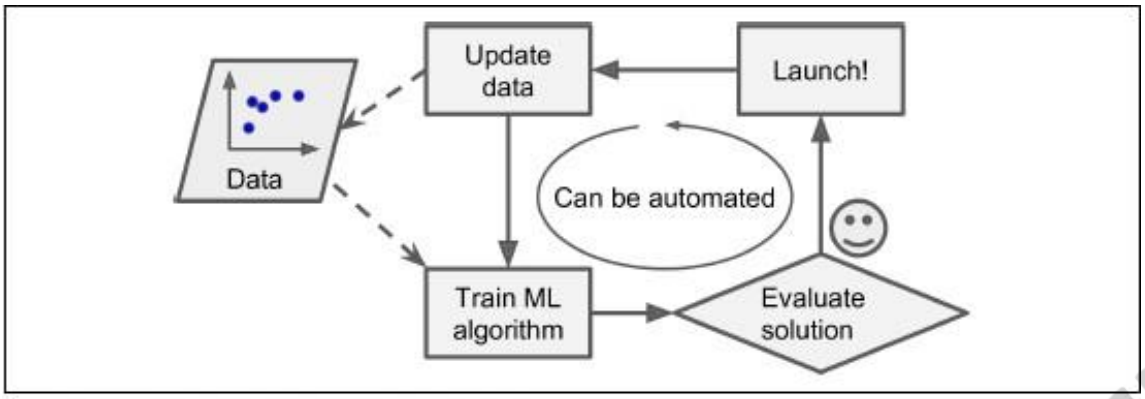


Figure 1-3. Automatically adapting to change

Another area where Machine Learning shines is for problems that either are too complex for traditional approaches or have no known algorithm.

Example 2: Speech recognition

- If one wants to start simple and write a program capable of distinguishing the words "one" and "two," one might notice that the word "two" starts with a high-pitch sound ("T"), so one could hardcode an algorithm that measures high-pitch sound intensity and use that to distinguish ones and twos.
- Obviously, this technique will not scale to thousands of words spoken by millions of very different people in noisy environments and in dozens of languages.
- The best solution is to write an algorithm that learns by itself, given many example recordings for each word.

Finally, Machine Learning can help humans learn (Figure 1-4): ML algorithms can be inspected to see what they have learned. Sometimes it will reveal unsuspected correlations or new trends, and thereby lead to a better understanding of the problem.

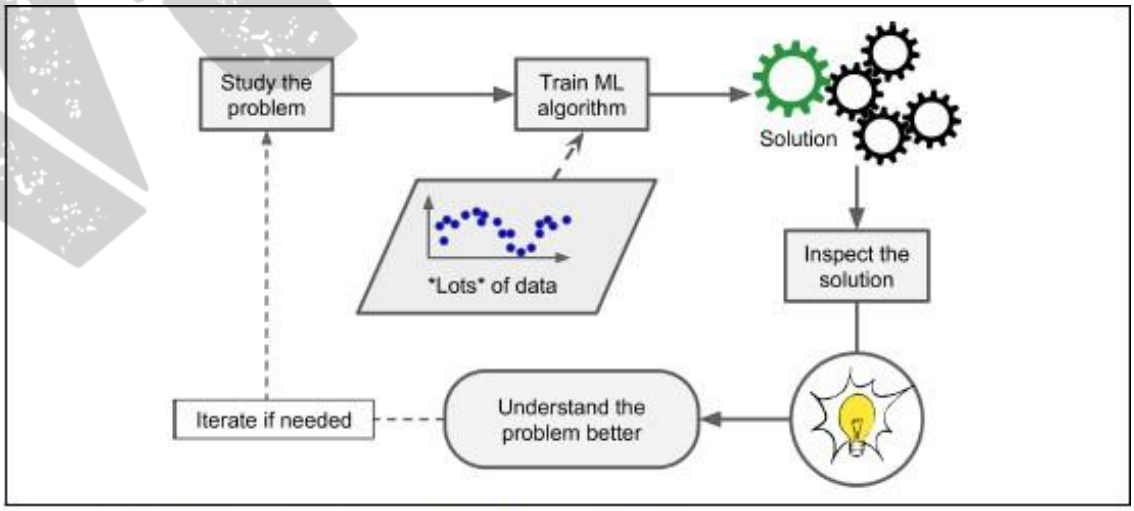


Figure 1-4. Machine Learning can help humans learn

Types of Machine learning

The different types of Machine Learning systems that it is useful to classify them in broad categories based on:

- Whether or not they are trained with human supervision
    1. Supervised learning
    2. Unsupervised learning
    3. Semi-supervised learning
    4. Reinforcement learning
- Whether or not they can learn incrementally on the fly
    1. Online learning
    2. 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
    1. Instance-based learning
    2. Model-based learning

1. 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. Semi-supervised learning
4. Reinforcement learning.

1. **Supervised learning**: The training data is feed to the algorithm includes the desired solutions, called labels.

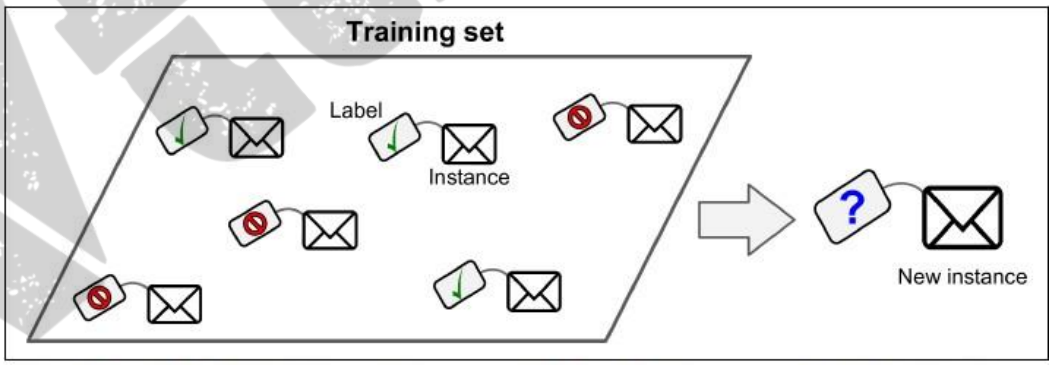


Figure 1-5. A labeled training set for supervised learning (e.g., spam classification)

A typical Supervised learning task is **Classification**.

- The spam filter is a good example of classification: it is trained with many example emails along with their class (spam or ham), and it must learn how to classify new emails.

Another Supervised learning task is **Regression**

- Here, task is to predict a target numeric value, such as the price of a car, given a set of features (mileage, age, brand, etc.) called predictors. This sort of task is called regression (Figure 1-6)
- To train the system, you need to give it many examples of cars, including both their predictors and their labels (i.e., their prices).

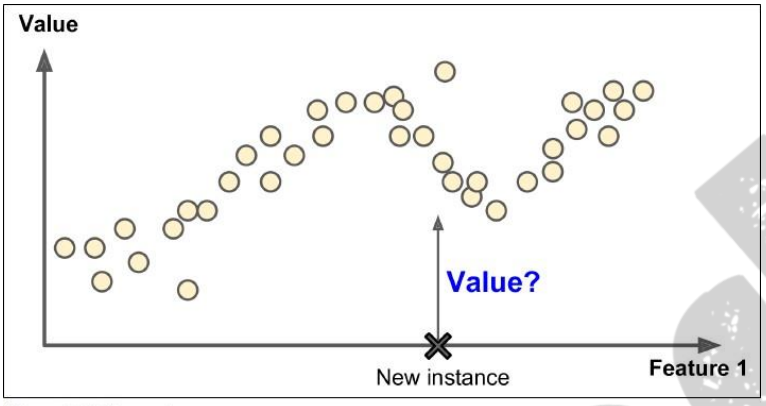


Figure 1-6. Regression

Here are some of the most important supervised learning algorithms

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Decision Trees and Random Forests
- Neural networks

2. **Unsupervised learning:** In unsupervised learning the training data is unlabeled (Figure 1-7). The system tries to learn without a teacher

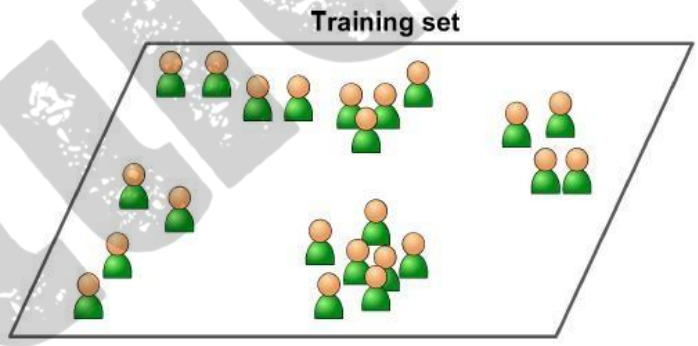


Figure 1-7. An unlabeled training set for unsupervised learning

Here are some of the most important unsupervised learning algorithms

- Clustering
- K-Means
- DBSCAN
- Hierarchical Cluster Analysis (HCA)

Anomaly detection and novelty detection

- One-class SVM
- Isolation Forest
- Eclat

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

For example:

- Data about blog's visitors. You may want to run a clustering algorithm to try to detect groups of similar visitors (Figure 1-8). the algorithm tells which group a visitor belongs to.
- It might notice that 40% of your visitors are males who love comic books and generally read your blog in the evening, while 20% are young sci-fi lovers who visit during the weekends, and so on.
- If you use a hierarchical clustering algorithm, it may also subdivide each group into smaller groups. This may help you target your posts for each group.

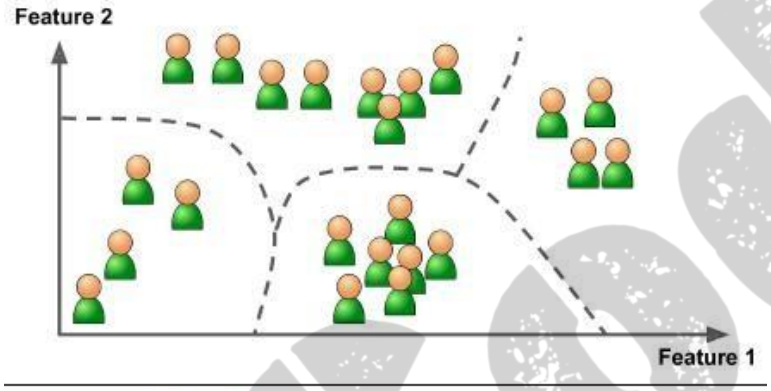


Figure 1-8. Clustering

3. **Semisupervised learning**: Some algorithms can deal with partially labeled training data, usually a lot of unlabeled data and a little bit of labeled data. This is called semisupervised learning (Figure 1-11).

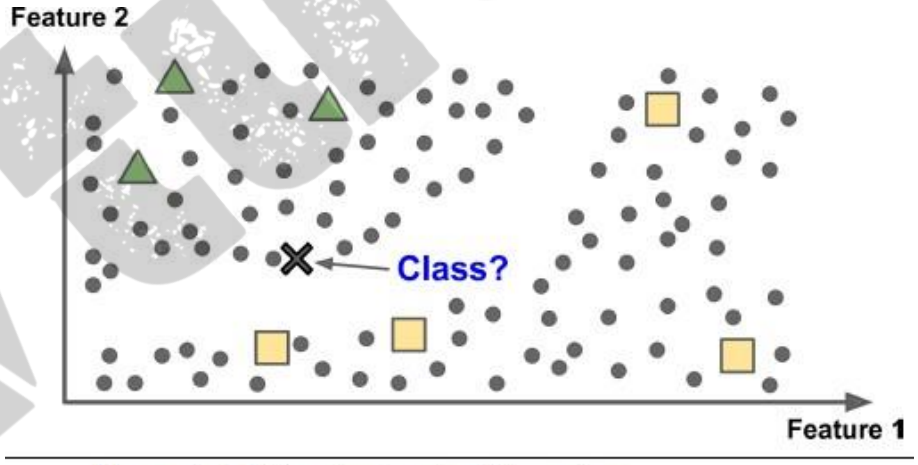


Figure 1-11. Semisupervised learning

Example: Google Photos

Once users upload all their 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 constitutes the unsupervised part of the algorithm (clustering). Now all the system needs are for users 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.

4. **Reinforcement Learning:** The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, as in Figure 1-12). 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.

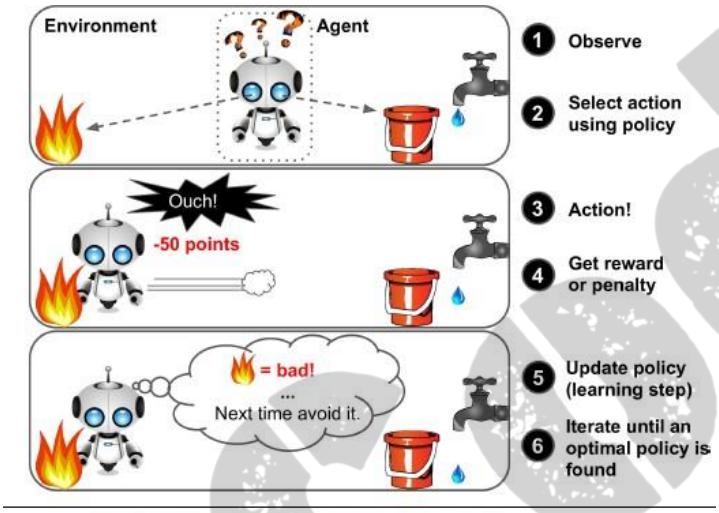


Figure 1-12. Reinforcement Learning

2. Machine Learning systems is classified based on whether or not the system can learn incrementally from a stream of incoming data.

1. **Batch learning:** In batch learning, the system is incapable of learning incrementally: it must be trained using all the available data. This will generally take a lot of time and computing resources, so it is typically done offline. First the system is trained, and then it is launched into production and runs without learning anymore; it just applies what it has learned. This is called offline learning (**Batch learning**).

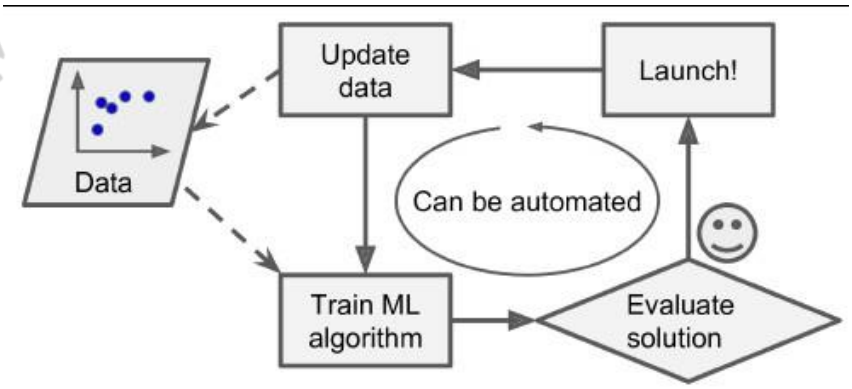


Figure 1-3. Automatically adapting to change

- If one desires a batch learning system to know about new data, one must train a new version of the system from scratch on the full dataset, 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 fairly easily, so even a batch learning system can adapt to change. Simply update the data and train a new version of the system from scratch as often as needed.
- This solution is simple and often works fine, but training using the full set of data can take many hours, so you would typically train a new system only every 24 hours or even just weekly. If your system needs to adapt to rapidly changing data (e.g., to predict stock prices), then you need a more reactive solution.

2. **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 (see Figure 1-13).

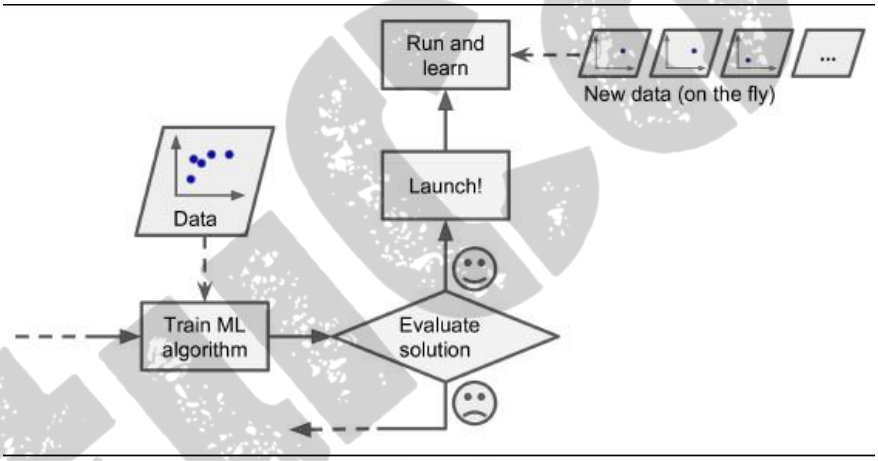


Figure 1-13. Online learning

- Online learning is great for systems that receive data as a continuous flow (e.g., stock prices) and need to adapt to change rapidly or autonomously. If it has had limited computing resources: once an online learning system has learned about new data instances, it does not need them anymore, so you can discard them is also a good option. This can save a huge amount of space.
- Online learning algorithms can also be used to train systems on huge datasets that cannot fit in one machine's main memory (this is called out-of-core learning). 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.
- One important parameter of online learning systems is how fast they should adapt to changing data: this is called the ***learning rate***. If you set a high learning rate, then your system will rapidly adapt to new data, but it will also tend to quickly forget the old data.

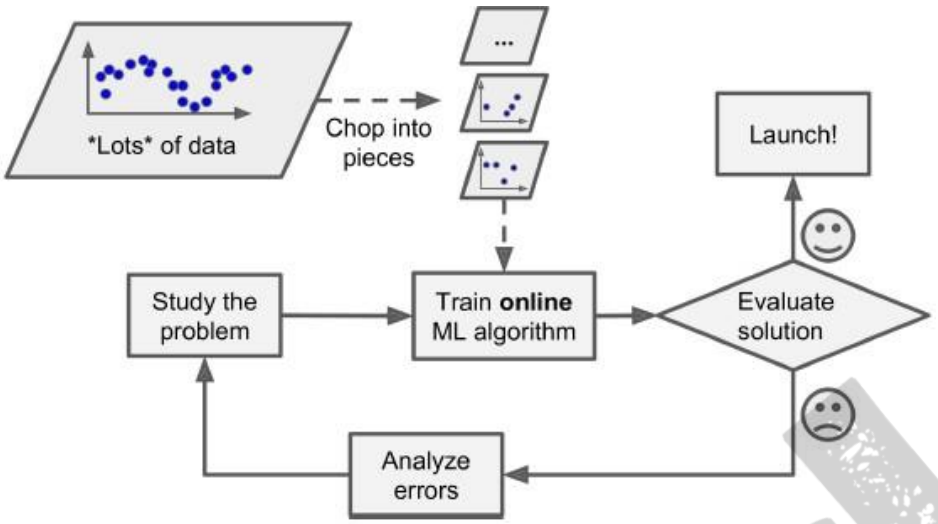


Figure 1-14. Using online learning to handle huge datasets

3. One more way Machine Learning systems is classified based by how they generalize.
There are two main approaches to generalization:

1. **Instance-based learning:** The system learns the examples, then generalizes to new cases by comparing them to the learned examples using a similarity measure.
   For example, in Figure 1-15 the new instance would be classified as a triangle because the majority of the most similar instances belong to that class.

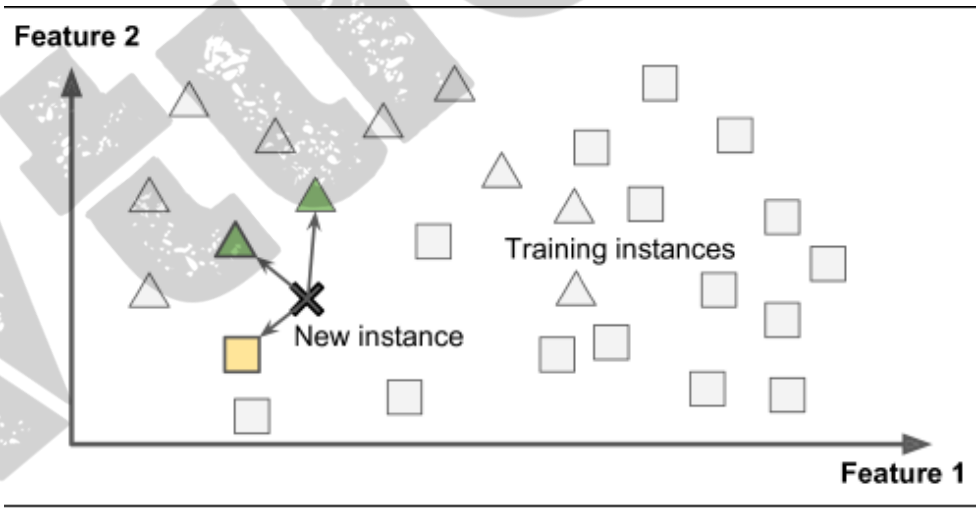


Figure 1-15. Instance-based learning

2. **Model-based learning:** Build a model of from a set of examples, then use that model to make predictions. This is called model-based learning.

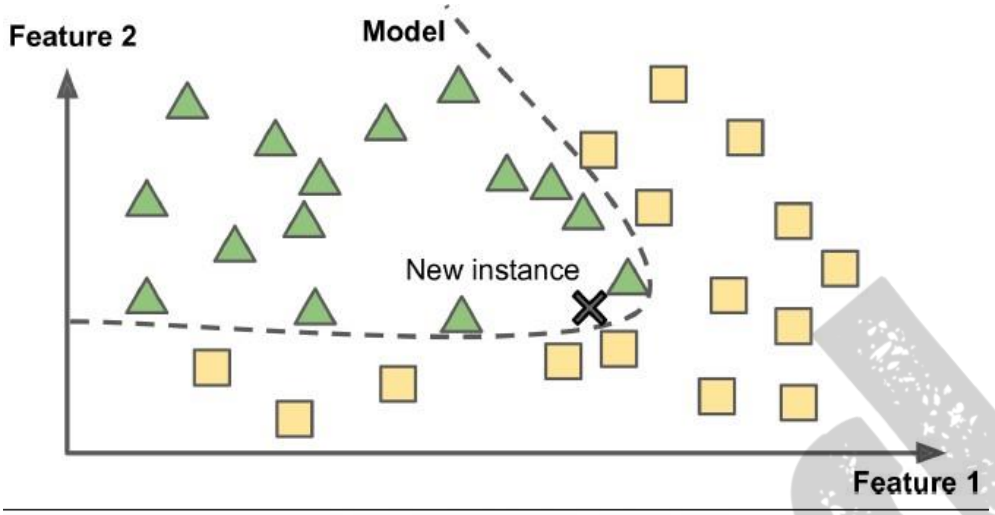


Figure 1-16. Model-based learning

Main challenges of Machine learning

In ML the main task is to select a learning algorithm and train it on some data, the two things that can go wrong are "bad algorithm" and "bad data. So, here are some challenges of ML

1. Insufficient Quantity of Training Data
2. Nonrepresentative Training Data
3. Poor-Quality Data
4. Irrelevant Features
5. Overfitting the Training Data
6. Underfitting the Training Data
7. Testing and Validating
8. Hyperparameter Tuning and Model Selection

1. **Insufficient Quantity of Training Data**: Machine Learning takes a lot of data for most ML algorithms to work properly. Even for very simple problems you typically need thousands of examples, and for complex problems such as image or speech recognition you may need millions of examples.
2. **Nonrepresentative Training Data**: In order to generalize well, it is crucial that training data is representative of the new cases one aims to generalize to. By using a nonrepresentative training set, we trained a model that is unlikely to make accurate predictions
3. **Poor-Quality Data**: If the training data is full of errors, outliers, and noise (e.g., due to poor-quality measurements), it will make it harder for the system to detect the

underlying patterns, so the system is less likely to perform well. It is often well worth the effort to spend time cleaning up the training data. The truth is, most data scientists spend a significant part of their time doing just that.

4. **Irrelevant Features**: The 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***.

   The feature engineering includes:
   - *Feature selection*: selecting the most useful features to train on among existing features.
   - *Feature extraction*: combining existing features to produce a more useful one
   - *Creating new features* by gathering new data.

5. **Overfitting the Training Data**: It means that the model performs well on the training data, but it does not generalize well. Complex models such as deep neural networks can detect subtle patterns in the data, but if the training set is noisy, or if it is too small then the model is likely to detect patterns in the noise itself. Obviously, these patterns will not generalize to new instances.
   Overfitting happens when the model is too complex relative to the amount and noisiness of the training data. The possible solutions are:
   - To simplify the model by selecting one with fewer parameters by reducing the number of attributes in the training data or by constraining the model
   - To gather more training data
   - 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.***

6. **Underfitting the Training Data:** It occurs when a model is too simple to learn the underlying structure of the data.
   Example: Attempting to fit a linear model to predict housing prices based solely on the number of bedrooms. While bedrooms may have some correlation with price, a linear model over simplifies the relationship between various features (such as square footage, location, amenities, etc.) and housing prices. As a result, the model would likely perform poorly in accurately predicting housing prices, both on the training data and unseen data, because it fails to capture the complexities of the housing market.
   The main options to fix this problem are:
   - Selecting a more powerful model, with more parameters
   - Feeding better features to the learning algorithm (feature engineering)
   - Reducing the constraints on the model (e.g., reducing the regularization hyperparameter)

7. **Testing and Validating**: Splitting the data into two sets: the training set and the test set. The model is trained using the training set, and it is tested using the test set. The error rate on new cases is called the generalization error, and by evaluating the model on the test set, an estimate of this error is obtained. This value indicates how well the model will perform on instances it has never seen before. If the training error is low but the generalization error is high, it means that the model is overfitting the training data.

8. **Hyperparameter Tuning and Model Selection:**
   - Suppose the linear model generalizes better, but regularization is desired to avoid overfitting. The challenge arises in selecting the appropriate regularization hyperparameter value. One approach is to train numerous models with varied hyperparameter values, selecting the one yielding the lowest generalization error. However, relying solely on test set performance for hyperparameter tuning may lead to reduced performance on new data due to overfitting to the test set.
   - A common solution to this problem is called holdout validation: part of the training set is simply held out to evaluate several candidate models and select the best one. The new holdout set is called the validation set. Multiple models with various hyperparameters are trained on the reduced training set and the model that performs best on the validation set is selected. After this holdout validation process, the best model is trained on the full training set, and this gives the final model. Lastly, this final model is evaluated on the test set to get an estimate of the generalization error.

Concept Learning and Learning Problems

Well- Posed Learning Problems

Definition: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

1. ***Checkers game:*** A computer program that learns to play ***checkers*** might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself.

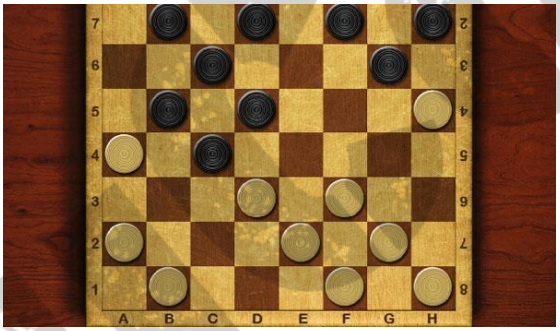


Fig: Checker game board

***A checkers learning problem*:**

- Task T: playing checkers
- Performance measure P: percent of games won against opponents
- Training experience E: playing practice games against itself

2. ***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
3. ***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

Designing a Leaning System

The basic design issues and approaches to machine learning are illustrated by designing a program to learn to play checkers, with the goal of entering it in the world checkers tournament

1. Choosing the Training Experience
2. Choosing the Target Function
3. Choosing a Representation for the Target Function
4. Choosing a Function Approximation Algorithm
    1. Estimating training values
    2. Adjusting the weights
5. The Final Design

1. Choosing the Training Experience

- The first design choice is to choose the type of training experience from which the system will learn.
- The type of training experience available can have a significant impact on success or failure of the learner.

There are three attributes which impact on success or failure of the learner

1. Whether the training experience provides ***direct or indirect feedback*** regarding the choices made by the performance system.
   For example, in checkers game:
   - In learning to play checkers, the system might learn from ***direct training examples*** consisting of ***individual checkers board states*** and ***the correct move for each***.
   - ***Indirect training examples*** consisting of the ***move sequences*** and ***final outcomes*** of various games played. The information about the correctness of specific moves early in the game must be inferred indirectly from the fact that the game was eventually won or lost.
   - Here the learner faces an additional problem of ***credit assignment***, or determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Credit assignment can be a particularly difficult problem because the game can be lost even when early moves are optimal, if these are followed later by poor moves.
   - Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

2. The degree to which the ***learner controls the sequence of training examples***

For example, in checkers game:

- The learner might depend on the ***teacher*** to select informative board states and to provide the correct move for each.

- Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move.
- The learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with ***no teacher present***.

3. How well it represents the ***distribution of examples*** over which the final system performance P must be measured
   - For example, in checkers game:
     In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament.
   - If its training experience E consists only of games played against itself, there is a danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
   - It is necessary to learn from a distribution of examples that is different from those on which the final system will be evaluated.

2. Choosing the Target Function

- The next design choice is to determine exactly what type of knowledge will be learned and how this will be used by the performance program.
- Let's consider a checkers-playing program that can generate the legal moves from any board state.
- The program needs only to learn how to choose the best move from among these legal moves.
- We must learn to choose among the legal moves, the most obvious choice for the type of information to be learned is a program, or function, that chooses the best move for any given board state.

1. Let ***ChooseMove*** be the target function and the notation is

   $$ChooseMove : B \rightarrow M$$

   which indicate that this function accepts as input any board from the set of legal board states B and produces as output some move from the set of legal moves M.

   ChooseMove is a choice for the target function in checkers example, but this function will turn out to be very difficult to learn given the kind of indirect training experience available to our system

2. An alternative target function is an ***evaluation function*** that assigns a ***numerical score*** to any given board state
   Let the target function V and the notation

   $$V : B \rightarrow R$$

which denote that V maps any legal board state from the set B to some real value. Intend for this target function V to assign higher scores to better board states. If the system can successfully learn such a target function V, then it can easily use it to select the best move from any current board position.

Let us define the target value V(b) for an arbitrary board state b in B, as follows:

- If b is a final board state that is won, then V(b) = 100
- If b is a final board state that is lost, then V(b) = -100
- If b is a final board state that is drawn, then V(b) = 0
- If b is a not a final state in the game, then V(b) = V(b'),

Where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game

3. Choosing a Representation for the Target Function

Let's choose a simple representation - for any given board state, the function ***c*** will be calculated as a linear combination of the following board features:

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

Thus, learning program will represent as a linear function of the form

$$\hat{V}(b) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

Where,

- w_0 through w_6 are numerical coefficients, or weights, to be chosen by the learning algorithm.
- Learned values for the weights w1 through w6 will determine the relative importance of the various board features in determining the value of the board
- The weight w_0 will provide an additive constant to the board value

4. Choosing a Function Approximation Algorithm

In order to learn the target function **f** we require a set of training examples, each describing a specific board state b and the training value $V_{train}(b)$ for b.

Each training example is an ordered pair of the form (b, $V_{train}(b)$).

For instance, the following training example describes a board state b in which black has won the game (note $x_2 = 0$ indicates that red has no remaining pieces) and for which the target function value $V_{train}(b)$ is therefore +100.

$$((x_1=3, x_2=0, x_3=1, x_4=0, x_5=0, x_6=0), +100)$$

Function Approximation Procedure

1. Derive training examples from the indirect training experience available to the learner
2. Adjusts the weights w_i to best fit these training examples

1. Estimating training values

   A simple approach for estimating training values for intermediate board states is to assign the training value of $V_{train}(b)$ for any intermediate board state b to be $\hat{V}$ (Successor(b))

   Where ,
   - $\hat{V}$ is the learner's current approximation to V
   - Successor(b) denotes the next board state following b for which it is again the program's turn to move

   Rule for estimating training values

   $$\mathbf{V_{train}(b) \leftarrow \hat{V}\ (Successor(b))}$$

2. Adjusting the weights

   Specify the learning algorithm for choosing the weights w_i to best fit the set of training examples {(b, $V_{train}(b)$)}

   A first step is to define what we mean by the bestfit to the training data.

   One common approach is to define the best hypothesis, or set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis.

$$E \equiv \sum_{\langle b, V_{train}(b)\rangle \in \text{training examples}} (V_{train}(b) - \hat{V}(b))^2$$

Several algorithms are known for finding weights of a linear function that minimize E. One such algorithm is called the ***least mean squares, or LMS training rule***. For each observed training example it adjusts the weights a small amount in the direction that reduces the error on this training example

LMS weight update rule :- For each training example (b, $V_{train}(b)$)

Use the current weights to calculate $\hat{V}$ (b)

For each weight w_i, update it as

$$w_i \leftarrow w_i + \eta\,(V_{train}(b) - \hat{V}(b))\,x_i$$

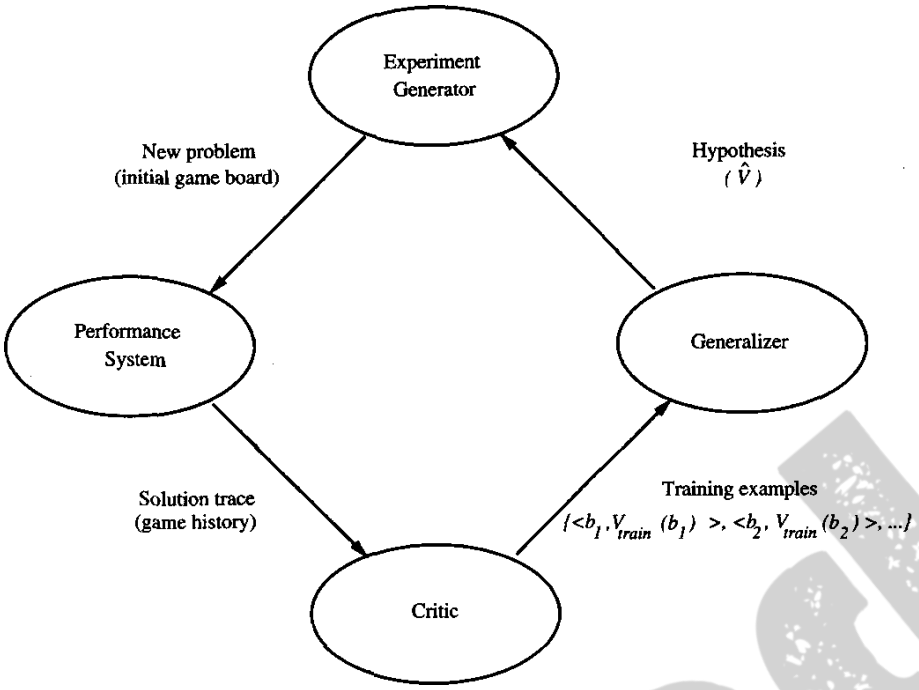
Here η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Working of weight update rule

- When the error ($V_{train}(b) - \hat{V}(b)$) is zero, no weights are changed.
- When ($V_{train}(b) - \hat{V}(b)$) is positive (i.e., when $\hat{V}$ (b) is too low), then each weight is increased in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}$ (b), reducing the error.
- If the value of some feature x_i is zero, then its weight is not altered regardless of the error, so that the only weights updated are those whose features actually occur on the training example board.

5. The Final Design

The final design of checkers learning system can be described by four distinct program modules that represent the central components in many learning systems



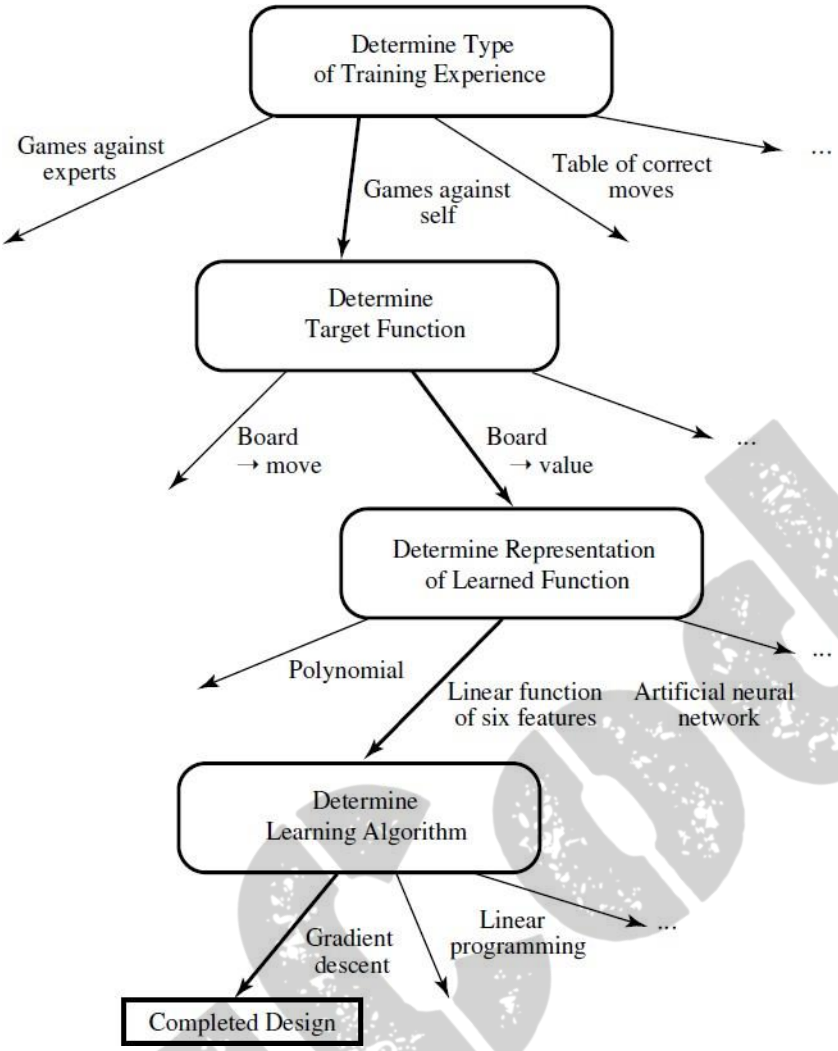
1. **The Performance System** is the module that must solve the given performance task by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output.

2. **The Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function

3. **The Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples.

4. **The Experiment Generator** takes as input the current hypothesis and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system.

The sequence of design choices made for the checkers program is summarized in below figure


Determine Type of Training Experience
Games against experts
Games against self
Table of correct moves
...
Determine Target Function
Board → move
Board → value
...
Determine Representation of Learned Function
Polynomial
Linear function of six features
Artificial neural network
...
Determine Learning Algorithm
Gradient descent
Linear programming
...
Completed Design

PERSPECTIVES AND ISSUES IN MACHINE LEARNING

Issues in Machine Learning

The field of machine learning is concerned with answering questions such as the following

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

CONCEPT LEARNING

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

Definition: Concept learning **-** Inferring a Boolean-valued function from training examples of its input and output

A CONCEPT LEARNING TASK

Consider the example task of learning the target concept "Days on which ***Aldo*** enjoys his favorite water sport"

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

Table: Positive and negative training examples for the target concept ***EnjoySport.***

The task is to learn to predict the value of ***EnjoySport*** for an arbitrary day, based on the values of its other attributes?

What hypothesis representation is provided to the learner?

- Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- Let each hypothesis be a vector of six constraints, specifying the values of the six attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*.

For each attribute, the hypothesis will either

- Indicate by a "?' that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a "Φ" that no value is acceptable

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

The hypothesis that ***PERSON*** enjoys his favorite sport only on cold days with high humidity is represented by the expression

(?, Cold, High, ?, ?, ?)

The most general hypothesis-that every day is a positive example-is represented by

(?, ?, ?, ?, ?, ?)

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

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

- The set of items over which the concept is defined is called the ***set of instances***, which is denoted by X.

Example: X is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

- The concept or function to be learned is called the ***target concept***, which is denoted by c.

c can be any Boolean valued function defined over the instances X

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

Example: The target concept corresponds to the value of the attribute ***EnjoySport*** (i.e., c(x) = 1 if ***EnjoySport*** = Yes, and c(x) = 0 if ***EnjoySport*** = No).

- Instances for which c(x) = 1 are called ***positive examples***, or members of the target concept.
- Instances for which c(x) = 0 are called ***negative examples***, or non-members of the target concept.
- The ordered pair (x, c(x)) to describe the training example consisting of the instance x and its target ***concept value c(x).***
- ***D*** to denote the set of available training examples

- The symbol ***H*** to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis ***h*** in ***H*** represents a Boolean-valued function defined over ***X***

$$h: X \rightarrow \{O, 1\}$$

The goal of the learner is to find a hypothesis h such that h(x) = c(x) for all x in X.

- Given:
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values Sunny, Cloudy, and Rainy),
    - *AirTemp* (with values Warm and Cold),
    - *Humidity* (with values Normal and High),
    - *Wind* (with values Strong and Weak),
    - *Water* (with values Warm and Cool),
    - *Forecast* (with values Same and Change).
  - Hypotheses *H*: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Φ" (no value is acceptable), or a specific value.
  - Target concept *c*: ***EnjoySport*** : X → {0, 1}
  - Training examples *D*: Positive and negative examples of the target function
- Determine:
  - A hypothesis h in H such that *h(x) = c(x)* for all *x* in X.

Table: The ***EnjoySport*** concept learning task.

The inductive learning hypothesis

Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

CONCEPT LEARNING AS SEARCH

- Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- The goal of this search is to find the hypothesis that best fits the training examples.

Example:
Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp, Humidity, Wind, Water, Forecast* each have two possible values, the instance space X contains exactly

3.2.2.2.2.2 = 96 distinct instances

5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.

Every hypothesis containing one or more "Φ" symbols represents the empty set of instances; that is, it classifies every instance as negative.

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

h_1 = (Sunny, ?, ?, Strong, ?, ?)

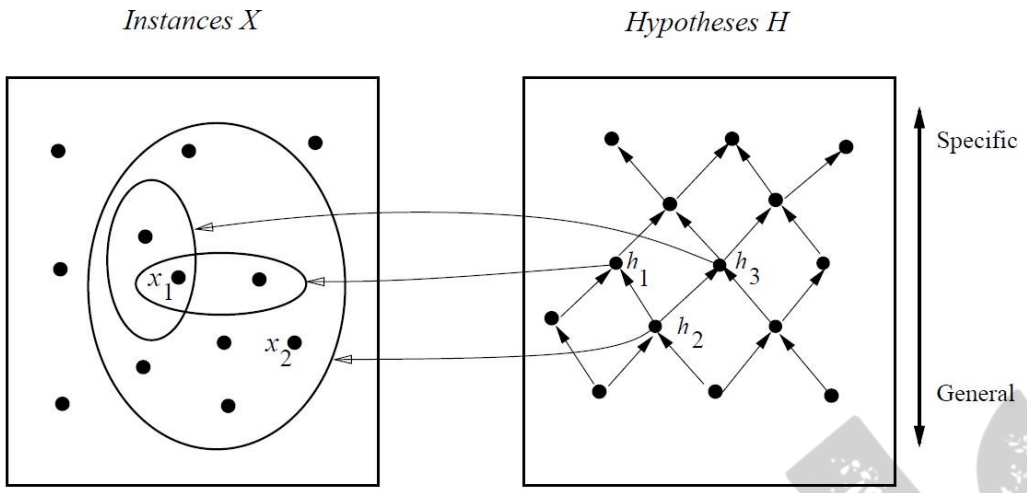
h_2 = (Sunny, ?, ?, ?, ?, ?)

- Consider the sets of instances that are classified positive by h_1 and by h_2.
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_1 will also be classified positive by h_2. Therefore, h_2 is more general than h_1.

Given hypotheses h_j and h_k, h_j is more-general-than or- equal do h_k if and only if any instance that satisfies h_k also satisfies h_i

Definition: Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is ***more general-than-or-equal-to*** h_k (written $h_j \geq h_k$) if and only if

$$(\forall\ x \in X\ )\ [(h_k\ (x) = 1) \rightarrow (h_j\ (x) = 1)]$$



x_1 = <Sunny, Warm, High, Strong, Cool, Same>
x_2 = <Sunny, Warm, High, Light, Warm, Same>

h_1 = <Sunny, ?, ?, Strong, ?, ?>
h_2 = <Sunny, ?, ?, ?, ?, ?>
h_3 = <Sunny, ?, ?, ?, Cool, ?>

- In the figure, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X-the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the more - general -than relation, with the arrow pointing toward the less general hypothesis.
- Note the subset of instances characterized by h_2 subsumes the subset characterized by h_l , hence h_2 is more - general– than h_1

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

FIND-S Algorithm

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

To illustrate this algorithm, assume the learner is given the sequence of training examples from the ***EnjoySport*** task

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

- The first step of FIND-S is to initialize h to the most specific hypothesis in H

  h - (Ø, Ø, Ø, Ø, Ø, Ø)

- Consider the first training example

  x_1 = <Sunny Warm Normal Strong Warm Same>, +

  Observing the first training example, it is clear that hypothesis *h* is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

  h_1 = <Sunny Warm Normal Strong Warm Same>

- Consider the second training example

  x_2 = <Sunny, Warm, High, Strong, Warm, Same>, +

  The second training example forces the algorithm to further generalize h, this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example

  h_2 = <Sunny Warm ? Strong Warm Same>

- Consider the third training example

  x3 = <Rainy, Cold, High, Strong, Warm, Change>, -

  Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example.

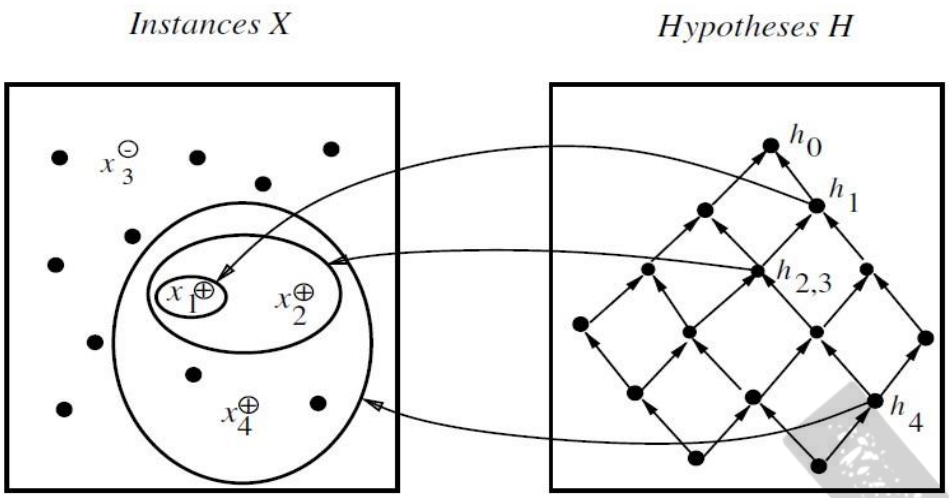
  h_3 = < Sunny Warm ? Strong Warm Same>

- Consider the fourth training example

  x_4 = <Sunny Warm High Strong Cool Change>, +

  The fourth example leads to a further generalization of h

  h_4 = < Sunny Warm ? Strong ? ? >



x_1 = <Sunny Warm Normal Strong Warm Same>, +
x_2 = <Sunny Warm High Strong Warm Same>, +
x_3 = <Rainy Cold High Strong Warm Change>, -
x_4 = <Sunny Warm High Strong Cool Change>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <Sunny Warm Normal Strong Warm Sam
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

The key property of the FIND-S algorithm

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of the set of all ***hypotheses consistent with the training examples***

Representation

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

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

Note difference between definitions of *consistent* and *satisfies*

- An example *x* is said to ***satisfy*** hypothesis ***h*** when ***h(x)*** = 1, regardless of whether *x* is a positive or negative example of the target concept.
- An example *x* is said to ***consistent*** with hypothesis ***h*** iff ***h(x)*** = ***c(x)***

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

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

The LIST-THEN-ELIMINATION algorithm

The LIST-THEN-ELIMINATE algorithm first initializes the version space to contain all hypotheses in H and then eliminates any hypothesis found inconsistent with any training example.

---

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

The LIST-THEN-ELIMINATE Algorithm

- List-Then-Eliminate works in principle, so long as version space is finite.
- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

A More Compact Representation for Version Spaces

The version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

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

$$G \equiv \{g \in H \mid Consistent\ (g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

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

$$S \equiv \{s \in H \mid Consistent\ (s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

Theorem: Version Space representation theorem

Theorem: Let X be an arbitrary set of instances and Let H be a set of Boolean-valued hypotheses defined over X. Let c: X $\rightarrow$ {O, 1} be an arbitrary target concept defined over X, and let D be an arbitrary set of training examples {(x, c(x))). For all X, H, c, and D such that S and G are well defined,

$$VS_{H,D} = \{ h \in H \mid (\exists s \in S)\ (\exists g \in G)\ (g \geq_g h \geq_g s) \}$$

To Prove:

1. Every h satisfying the right hand side of the above expression is in $\boldsymbol{VS_{H, D}}$

2. Every member of $\boldsymbol{VS_{H, D}}$ satisfies the right-hand side of the expression

Sketch of proof:

1. let g, h, s be arbitrary members of G, H, S respectively with $g \geq_g h \geq_g s$
   - By the definition of ***S,*** **s** must be satisfied by all positive examples in D. Because $h \geq_g s$, h must also be satisfied by all positive examples in D.
   - By the definition of ***G,*** g cannot be satisfied by any negative example in D, and because $g \geq_g h$ h cannot be satisfied by any negative example in D. Because h is satisfied by all positive examples in D and by no negative examples in D, h is consistent with D, and therefore h is a member of $VS_{H,D}$.
2. It can be proven by assuming some h in $VS_{H,D}$,that does not satisfy the right-hand side of the expression, then showing that this leads to an inconsistency

CANDIDATE-ELIMINATION Learning Algorithm

The CANDIDATE-ELIMINTION algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

- If d is a positive example
  - Remove from G any hypothesis inconsistent with d
  - For each hypothesis s in S that is not consistent with d
    - Remove s from S
    - Add to S all minimal generalizations h of s such that
      - h is consistent with d, and some member of G is more general than h
    - Remove from S any hypothesis that is more general than another hypothesis in S

- If d is a negative example
  - Remove from S any hypothesis inconsistent with d
  - For each hypothesis g in G that is not consistent with d
    - Remove g from G
    - Add to G all minimal specializations h of g such that
      - h is consistent with d, and some member of S is more specific than h
    - Remove from G any hypothesis that is less general than another hypothesis in G

CANDIDATE- ELIMINTION algorithm using version spaces

An Illustrative Example

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

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set of all hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

$G_0 \langle ?, ?, ?, ?, ?, ? \rangle$

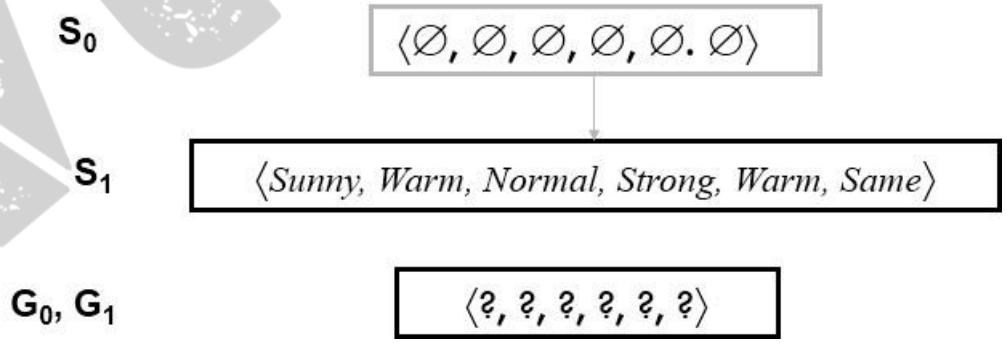
Initializing the S boundary set to contain the most specific (least general) hypothesis

$S_0 \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$

- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
- No update of the G boundary is needed in response to this training example because G_o correctly covers this example

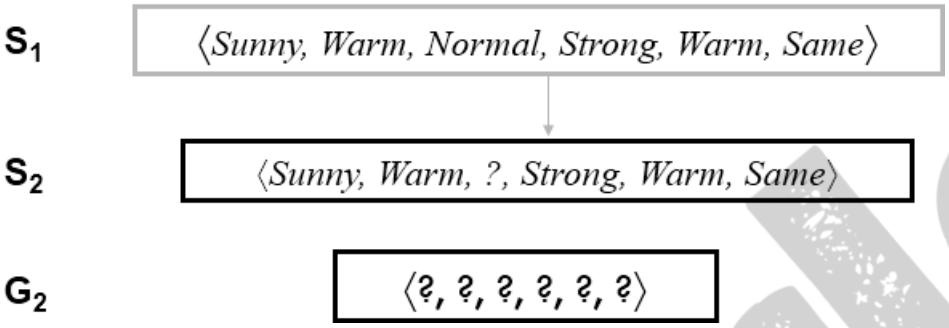
For training example d,

⟨Sunny, Warm, Normal, Strong, Warm, Same ⟩ +



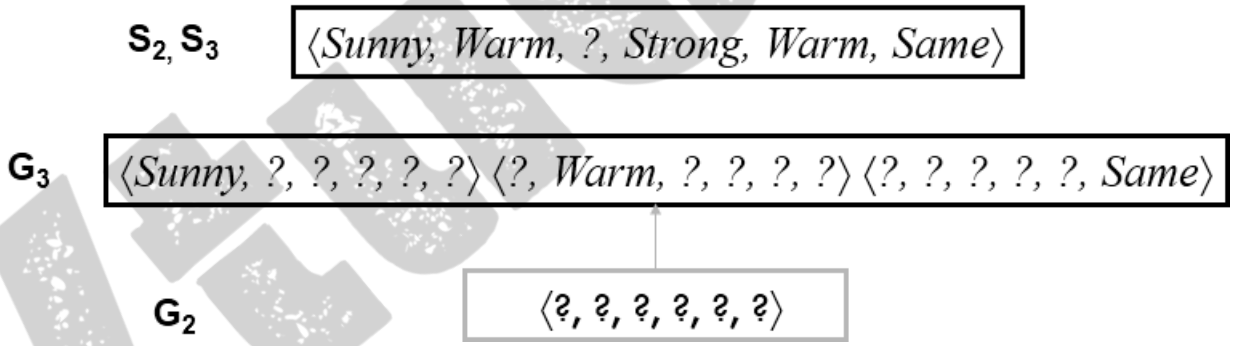
- When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

For training example d, ⟨Sunny, Warm, High, Strong, Warm, Same⟩ +



- Consider the third training example. This negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –



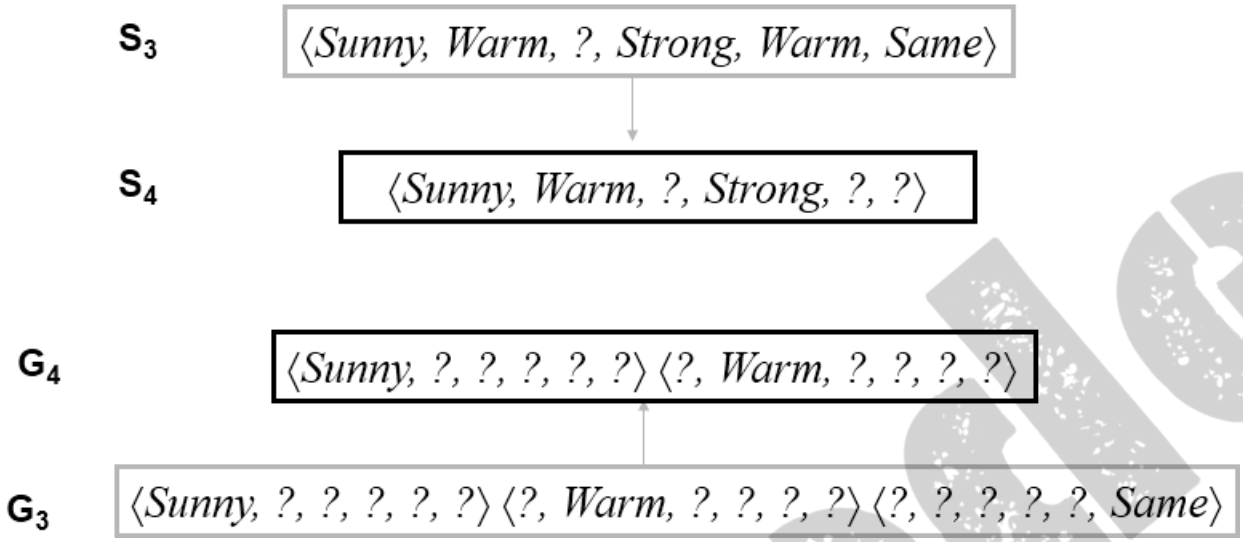
Given that there are six attributes that could be specified to specialize G_2, why are there only three new hypotheses in G_3?

For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G_2 that correctly labels the new example as a negative example, but it is not included in G_3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

- Consider the fourth training example.

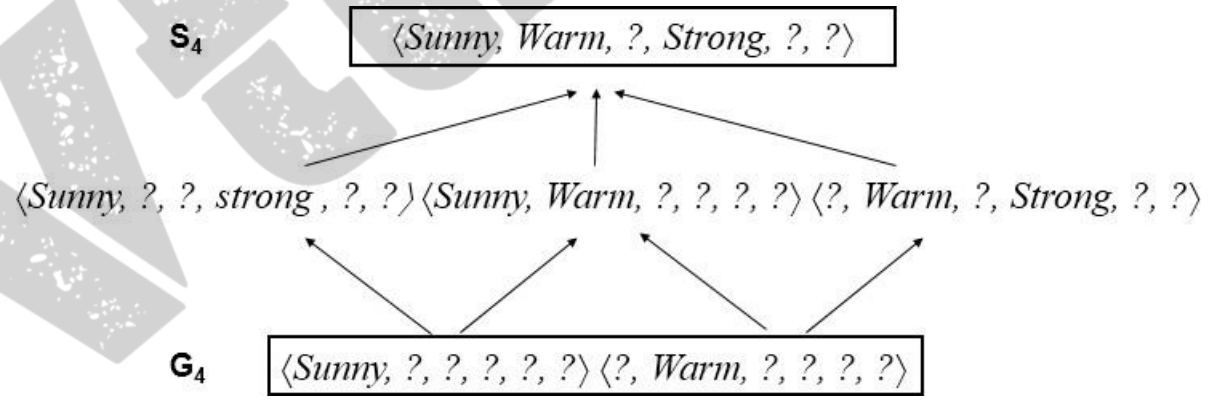
For training example d,

⟨Sunny, Warm, High, Strong, Cool Change ⟩ +



- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

After processing these four examples, the boundary sets S_4 and G_4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



INDUCTIVE BIAS

The fundamental questions for inductive inference

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

These fundamental questions are examined in the context of the CANDIDATE-ELIMINTION algorithm

A Biased Hypothesis Space

- Suppose the target concept is not contained in the hypothesis space *H*, then obvious solution is to enrich the hypothesis space to include every possible hypothesis.
  - Consider the ***EnjoySport*** example in which the hypothesis space is restricted to include only conjunctions of attribute values. Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as

    "Sky = Sunny or Sky = Cloudy."
  - The following three training examples of disjunctive hypothesis, the algorithm would find that there are zero hypotheses in the version space

⟨Sunny Warm Normal Strong Cool Change⟩	Y
⟨Cloudy Warm Normal Strong Cool Change⟩	Y
⟨Rainy Warm Normal Strong Cool Change⟩	N

  - If Candidate Elimination algorithm is applied, then it end up with empty Version Space. After first two training example

    S= ⟨? Warm Normal Strong Cool Change⟩

  - This new hypothesis is overly general and it incorrectly covers the third negative training example! So H does not include the appropriate c.
  - In this case, a more expressive hypothesis space is required.

An Unbiased Learner

- The solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space capable of representing every teachable concept that is representing every possible subset of the instances X.
- The set of all subsets of a set X is called the power set of X

  - In the ***EnjoySport*** learning task the size of the instance space X of days described by the six attributes is 96 instances.
  - Thus, there are 2^{96} distinct target concepts that could be defined over this instance space and learner might be called upon to learn.
  - The conjunctive hypothesis space is able to represent only 973 of these - a biased hypothesis space indeed

  - Let us reformulate the ***EnjoySport*** learning task in an unbiased way by defining a new hypothesis space H' that can represent every subset of instances
  - The target concept "Sky = Sunny or Sky = Cloudy" could then be described as

(Sunny, ?, ?, ?, ?, ?) v (Cloudy, ?, ?, ?, ?, ?)

The Futility of Bias-Free Learning

Inductive learning requires some form of prior assumptions, or inductive bias

Definition:

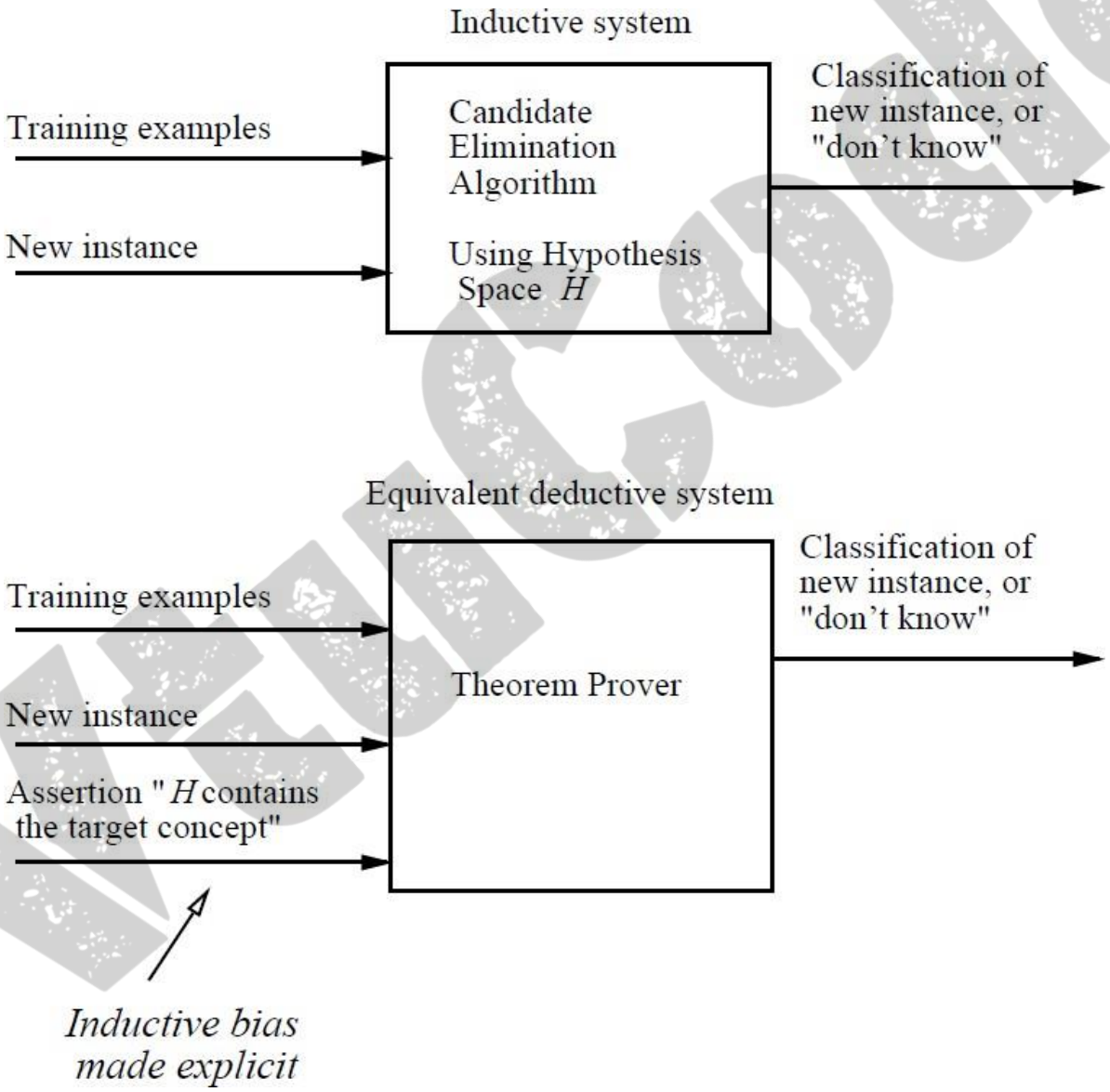
Consider a concept learning algorithm L for the set of instances X.

- Let c be an arbitrary concept defined over X
- Let $D_c = \{(x , c(x))\}$ be an arbitrary set of training examples of c.
- Let $L (x_i , D_c)$ denote the classification assigned to the instance x_i by L after training on the data D_c.
- The inductive bias of L is any minimal set of assertions B such that for any target concept c and corresponding training examples D_c

  - $(\forall \langle x_i \in X) [(B \wedge D_c \wedge x_i) \vdash L (x_i, D_c)]$

The below figure explains

- Modelling inductive systems by equivalent deductive systems.
- The input-output behavior of the CANDIDATE-ELIMINATION algorithm using a hypothesis space H is identical to that of a deductive theorem prover utilizing the assertion "H contains the target concept." This assertion is therefore called the inductive bias of the CANDIDATE-ELIMINATION algorithm.
- Characterizing inductive systems by their inductive bias allows modelling them by their equivalent deductive systems. This provides a way to compare inductive systems according to their policies for generalizing beyond the observed training data.



QUESTION BANK

1. What is Machine learning?
2. Why Use Machine learning? Explain with the example
3. Explain how you would categorize Machine Learning systems based on the amount and type of supervision they receive during training. Provide examples for each category.
4. Demonstrate how you would categorize Machine Learning systems based on their ability to learn incrementally from a stream of incoming data. Provide examples for each type.
5. Demonstrate how Machine Learning systems are classified based on their generalization capabilities. Provide examples for each classification.
6. Illustrate the main challenges of ML that can arise when selecting a learning algorithm and training it on data, specifically focusing on "bad algorithm" and "bad data." Provide examples to support the explanation.
7. Explain the concepts of overfitting and underfitting in the context of training data in Machine Learning
8. Illustrate some of the basic design issues and approaches to machine learning.
9. What are the issues in Machine Learning?
10. Discuss the concept learning with example.
11. Explain the General-to-Specific Ordering of Hypotheses
12. Explain the FIND-S algorithm with example.
13. Write the candidate elimination algorithm and illustrate with example
14. Write the final version space for the below mentioned training example using candidate elimination algorithm

Example – 1:

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

Example – 2:

Size	Color	Shape	Class
Big	Red	Circle	No
Small	Red	Triangle	No
Small	Red	Circle	Yes
Big	Blue	Circle	No
Small	Blue	Circle	Yes