UNIT-V

Topic:

Reinforcement Learning: Overview, example: getting lost, State and Action Spaces, The Reward Function, Discounting, Action Selection, Policy, Markov decision processes, Q-learning, uses of Reinforcement learning

Applications of Machine Learning in various fields: Text classification, Image Classification, Speech Recognition.

What is Reinforcement Learning?

Reinforcement Learning is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions. For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.

In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.

Since there is no labeled data, so the agent is bound to learn by its experience only.

RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as game-playing, robotics, etc.

The agent learns with the process of hit and trial, and based on the experience, it learns to perform the task in a better way. Hence, we can say that "Reinforcement learning is a type of machine learning method where an intelligent agent (computer program) interacts with the environment and learns to act within that." How a Robotic dog learns the movement of his arms is an example of Reinforcement learning.

Terms used in Reinforcement Learning

Agent(): An entity that can perceive/explore the environment and act upon it.

Environment(): A situation in which an agent is present or surrounded by. In RL, we assume the stochastic environment, which means it is random in nature.

Action(): Actions are the moves taken by an agent within the environment.

State(): State is a situation returned by the environment after each action taken by the agent.

Reward(): A feedback returned to the agent from the environment to evaluate the action of the agent.

Policy(): Policy is a strategy applied by the agent for the next action based on the current state.

Value(): It is expected long-term retuned with the discount factor and opposite to the short-term reward.

Q-value(): It is mostly similar to the value, but it takes one additional parameter as a current action (a).

Example:

A common example used to explain reinforcement learning is the scenario of an agent trying to navigate through a maze or a grid world. Let's consider the example of a person getting lost in a grid-like city.

State Space: The state space represents all possible states the agent can be in. In this case, the state space could be represented by the agent's current location in the city. Each location in the city corresponds to a unique state. For instance, if the city is a 5x5 grid, there would be 25 possible states.

Action Space: The action space represents the set of actions that the agent can take in each state. In the case of our lost person, the action space could include moving in four directions: up, down, left, and right. Each action corresponds to a change in the agent's location.

Rewards: In reinforcement learning, rewards are used to guide the learning process. In this example, we can assign rewards to incentivize the agent to reach the destination or penalize it for taking wrong paths. For instance, we can give a positive reward when the agent reaches the destination and a negative reward for hitting a wall or moving in the wrong direction.

Learning Process: The agent would start in a random state (initial location) in the city. It would then take an action based on its current state and receive a reward based on the action taken. The agent's goal is to learn a policy (a mapping from states to actions) that maximizes the cumulative rewards over time.

The learning process typically involves using a reinforcement learning algorithm such as Q-learning or Deep Q-Networks (DQN). The agent explores the environment by taking actions and updates its policy based on the observed rewards. Over time, the agent learns the optimal policy to navigate through the city and reach the destination efficiently.

The agent's policy could be represented as a Q-table (in Q-learning) or a neural network (in DQN), which maps states to action values. The agent selects actions based on the highest action value (or using an exploration-exploitation strategy) to navigate through the city and eventually find the destination.

This example illustrates how reinforcement learning can be applied to solve the problem of navigating through a grid-like environment when someone is lost. The agent learns from trial and error to make decisions that lead to the desired outcome based on the observed rewards.

Reward function:

The Reward Function is an **incentive mechanism** that tells the agent what is correct and what is wrong using reward and punishment. The goal of agents in RL is to maximize the total rewards. Sometimes we need to sacrifice immediate rewards in order to maximize the total rewards.

Discount Factor:

The discount factor, γ, is a real value $\in [0, 1]$, cares for the rewards agent achieved in the past, present, and future. In different words, it relates the rewards to the time domain. Let’s explore the two following cases:

If $\gamma = 0$, the agent cares for his first reward only.

If $\gamma = 1$, the agent cares for all future rewards.

Generally, the designer should predefine the discount factor for the scenario episode. This might raise many stability problems and can be ended without achieving the desired goal. However, by exploring some parameters many problems can be solved with converged solutions.

Action Selection:

Action selection in reinforcement learning refers to the process of choosing an action for an agent to take based on its current policy or learned values. When faced with a particular state in an environment, the agent needs to decide which action to select to maximize its expected future rewards.

There are different strategies for action selection in reinforcement learning, depending on the specific algorithm and approach being used. Here are a few common techniques:

Greedy Action Selection: In this approach, the agent selects the action with the highest estimated value (Q-value or action-value) for the current state. It chooses the action that it believes will lead to the highest immediate reward or long-term cumulative rewards. The greedy action selection exploits the current knowledge of the agent.

Epsilon-Greedy Action Selection: This strategy combines both exploration and exploitation. The agent selects the greedy action with a probability of (1-ε), where ε is the exploration rate, and it explores a random action with a probability of ε. Epsilon-greedy allows the agent to balance between exploiting the current knowledge (greedy action) and exploring new actions to gather more information about the environment.

Softmax Action Selection: The softmax action selection method assigns probabilities to each action based on their estimated values. It converts the action values into a probability distribution using the softmax function. Actions with higher estimated values have higher probabilities of being selected, but the agent still has a chance to explore suboptimal actions.

Upper Confidence Bound (UCB) Action Selection: UCB is often used in bandit problems, which are a simplified version of reinforcement learning. UCB assigns an exploration bonus to actions based on their uncertainty or lack of exploration. It balances exploration and exploitation by considering both the estimated value of an action and its confidence interval.

Thompson Sampling: Thompson Sampling is another strategy commonly used in bandit problems. It maintains a probability distribution over the true values of actions and samples from this distribution to select actions. It explores actions based on the uncertainty in their values, using a Bayesian approach.

Policy:

A policy defines the behavior of an agent, specifying how it selects actions given a particular state in the environment. It serves as a mapping from states to actions and guides the agent's decision-making process.

There are two main types of policies in reinforcement learning:

Deterministic Policy: A deterministic policy directly maps states to specific actions. For every state in the environment, the policy provides a single action to take. Mathematically, a deterministic policy can be represented as $\pi(s) = a$, where π denotes the policy, s represents a state, and a represents the action chosen for that state.

Stochastic Policy: A stochastic policy, on the other hand, defines a probability distribution over actions for each state. Instead of providing a single action, the policy provides the likelihood of selecting each action. This allows for random exploration and provides a way to handle uncertainty in decision-making. A stochastic policy can be represented as π(a|s) = P(A=a|S=s), indicating the probability of taking action a given state s.

Markov Decision Process:

The Agent-Environment Relationship

First let's look at some formal definitions :

Agent : Software programs that make intelligent decisions and they are the learners in RL. These agents interact with the environment by actions and receive rewards based on there actions.

Environment :It is the demonstration of the problem to be solved.Now, we can have a real-world environment or a simulated environment with which our agent will interact.

State : This is the position of the agents at a specific time-step in the environment.So,whenever an agent performs a action the environment gives the agent reward and a new state where the agent reached by performing the action.

The Markov Property state that :

"Future is Independent of the past given the present"

Mathematically we can express this statement as :

$$P[S_{t+1} \mid S_t] = P[S_{t+1} \mid S_1 , \ldots , S_t]$$

Markov Property

S[t] denotes the current state of the agent and s[t+1] denotes the next state. What this equation means is that the transition from state S[t] to S[t+1] is entirely independent of the past. So, the RHS of the Equation means the same as LHS if the system has a Markov Property. Intuitively meaning that our current state already captures the information of the past states.

State Transition Probability :

As we now know about transition probability we can define state Transition Probability as follows :

For Markov State from S[t] to S[t+1] i.e. any other successor state , the state transition probability is given by

$$\mathcal{P}_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$

State Transition Probability

We can formulate the State Transition probability into a State Transition probability matrix by :

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1n} \\ p_{21} & p_{22} & \cdots & p_{2n} \\ & \cdots & \cdots & \\ p_{n1} & p_{n2} & \cdots & p_{nn} \end{bmatrix}$$

State Transition Probability Matrix

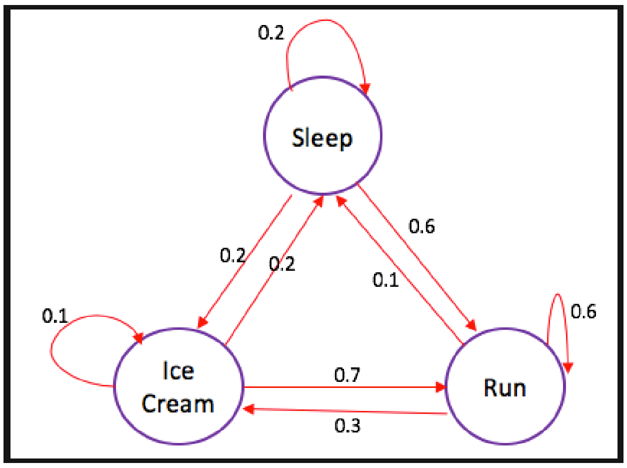
Each row in the matrix represents the probability from moving from our original or starting state to any successor state.Sum of each row is equal to 1.

Markov Process or Markov Chains

Markov Process is the memory less random process i.e. a sequence of a random state S[1],S[2],....S[n] with a Markov Property.So, it's basically a sequence of states with the Markov Property.It can be defined using a set of states(S) and transition probability matrix (P).The dynamics of the environment can be fully defined using the States(S) and Transition Probability matrix(P).

But what random process means ?

To answer this question let's look at a example:



Markov chain

The edges of the tree denote transition probability. From this chain let's take some sample. Now, suppose that we were sleeping and the according to the probability distribution there is a 0.6 chance that we will Run and 0.2 chance we sleep more and again 0.2 that we will eat ice-cream. Similarly, we can think of other sequences that we can sample from this chain.

Some samples from the chain :

Sleep — Run — Ice-cream — Sleep

Sleep — Ice-cream — Ice-cream — Run

In the above two sequences what we see is we get random set of States(S) (i.e. Sleep,Ice-cream,Sleep) every time we run the chain.Hope, it's now clear why Markov process is called random set of sequences.

Before going to Markov Reward process let's look at some important concepts that will help us in understand MRPs.

Reward and Returns

We can define Returns as :

$$G_t = r_{t+1} + r_{t+2} + \ldots.. + r_T$$

Returns (Total rewards from the environment)

r[t+1] is the reward received by the agent at time step t[0] while performing an action(a) to move from one state to another. Similarly, r[t+2] is the reward received by the agent at time step t[1] by performing an action to move to another state. And, r[T] is the reward received by the agent by at the final time step by performing an action to move to another state.

Markov Reward Process

Till now we have seen how Markov chain defined the dynamics of a environment using set of states(S) and Transition Probability Matrix(P).But, we know that Reinforcement Learning is all about goal to maximize the reward.So, let's **add reward** to our Markov Chain.This gives us **Markov Reward Process.**

Mathematically, we define Markov Reward Process as :

$$R_s = E[R_{t+1} \mid S_t]$$

Markov Reward Process

What this equation means is how much reward (Rs) we get from a particular state S[t]. This tells us the immediate reward from that particular state our agent is in. As we will see in the next story how we maximize these rewards from each state our agent is in. In simple terms, maximizing the cumulative reward we get from each state.

Q-Learning

What is Q-learning?

Q-learning is a machine learning approach that enables a model to iteratively learn and improve over time by taking the correct action. Q-learning is a type of reinforcement learning.

Q-learning provides a model-free approach to reinforcement learning. There is no model of the environment to guide the reinforcement learning process. The agent -- which is the AI component that acts in the environment -- iteratively learns and makes predictions about the environment on its own.

Q-learning also takes an off-policy approach to reinforcement learning. A Q-learning approach aims to determine the optimal action based on its current state. The Q-learning approach can accomplish this by either developing its own set of rules or deviating from the prescribed policy. Because Q-learning may deviate from the given policy, a defined policy is not needed.

How does Q-learning work?

Q-learning models operate in an iterative process that involves multiple components working together to help train a model. The iterative process involves the agent learning by exploring the environment and updating the model as the exploration continues.

The multiple components of Q-learning include the following:

Agents: The agent is the entity that acts and operates within an environment.

States: The state is a variable that identifies the current position in an environment of an agent.

Actions: The action is the agent's operation when it is in a specific state.

Rewards: A foundational concept within reinforcement learning is the concept of providing either a positive or a negative response for the agent's actions.

Episodes: An episode is when an agent can no longer take a new action and ends up terminating.

Q-values: The Q-value is the metric used to measure an action at a particular state.

Here are the two methods to determine the Q-value:

Temporal difference. The temporal difference formula calculates the Q-value by incorporating the value of the current state and action by comparing the differences with the previous state and action.

Bellman's equation. Mathematician Richard Bellman invented this equation in 1957 as a recursive formula for optimal decision-making. In the q-learning context, Bellman's equation is used to help calculate the value of a given state and assess its relative position. The state with the highest value is considered the optimal state.

Q-learning models work through trial-and-error experiences to learn the optimal behavior for a task. The Q-learning process involves modeling optimal behavior by learning an optimal action value function or q-function. This function represents the optimal long-term value of action a in state s and subsequently follows optimal behavior in every subsequent state.

Bellman's equation

$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * \max(Q(s',a')) - Q(s,a))$

The equation breaks down as follows:

Q(s, a) represents the expected reward for taking action a in state s.

The actual reward received for that action is referenced by r while s' refers to the next state.

The learning rate is α and γ is the discount factor.

The highest expected reward for all possible actions a' in state s' is represented by max(Q(s', a')).

What is a Q-table?

The Q-table includes columns and rows with lists of rewards for the best actions of each state in a specific environment. A Q-table helps an agent understand what actions are likely to lead to positive outcomes in different situations.

The table rows represent different situations the agent might encounter, and the columns represent the actions it can take. As the agent interacts with the environment and receives feedback in the form of rewards or penalties, the values in the Q-table are updated to reflect what the model has learned.

The purpose of reinforcement learning is to gradually improve performance through the Q-table to help choose actions. With more feedback, the Q-table becomes more accurate so the agent can make better decisions and achieve optimal results.

The Q-table is directly related to the concept of the Q-function. The Q-function is a mathematical equation that looks at the current state of the environment and the action under consideration as inputs. The Q-function then generates outputs along with expected future rewards for that action in the specific state. The Q-table allows the agent to look up the expected future reward for any given state-action pair to move toward an optimized state.

What is the Q-learning algorithm process?

The Q-learning algorithm process is an interactive method where the agent learns by exploring the environment and updating the Q-table based on the rewards received.

The steps involved in the Q-learning algorithm process include the following:

Q-table initialization: The first step is to create the Q-table as a place to track each action in each state and the associated progress.

Observation: The agent needs to observe the current state of the environment.

Action: The agent chooses to act in the environment. Upon completion of the action, the model observes if the action is beneficial in the environment.

Update: After the action has been taken, it's time to update the Q-table with the results.

Repeat: Repeat steps 2-4 until the model reaches a termination state for a desired objective.

What are the advantages of Q-learning?

The Q-learning approach to reinforcement learning can potentially be advantageous for several reasons, including the following:

Model-free: The model-free approach is the foundation of Q-learning and one of the biggest potential advantages for some uses. Rather than requiring prior knowledge about an environment, the Q-learning agent can learn about the environment as it trains. The model-free approach is particularly beneficial for scenarios where the underlying dynamics of an environment are difficult to model or completely unknown.

Off-policy optimization: The model can optimize to get the best possible result without being strictly tethered to a policy that might not enable the same degree of optimization.

Flexibility: The model-free, off-policy approach enables Q-learning flexibility to work across a variety of problems and environments.

Offline training: A Q-learning model can be deployed on pre-collected, offline data sets.

What are the disadvantages of Q-learning?

The Q-learning approach to reinforcement model machine learning also has some disadvantages, such as the following:

Exploration vs. exploitation tradeoff. It can be hard for a Q-learning model to find the right balance between trying new actions and sticking with what's already known. It's a dilemma that is commonly referred to as the exploration vs. exploitation tradeoff for reinforcement learning.

Curse of dimensionality: Q-learning can potentially face a machine learning risk known as the curse of dimensionality. The curse of dimensionality is a problem with high-dimensional data where the amount of data required to represent the distribution increases exponentially. This can lead to computational challenges and decreased accuracy.

Overestimation: A Q-learning model can sometimes be too optimistic and overestimate how good a particular action or strategy is.

Performance: A Q-learning model can take a long time to figure out the best method if there are several ways to approach a problem.

What are some examples of Q-learning?

Q-learning models can improve processes in various scenarios. Here are a few examples of Q-learning uses:

Energy management: Q-learning models help manage energy for different resources such as electricity, gas and water utilities. A 2022 report from IEEE provides a precise approach for integrating a Q-learning model for energy management.

Finance: A Q-learning-based training model can build models for decision-making assistance, such as determining optimal moments to buy or sell assets.

Gaming: Q-learning models can train gaming systems to achieve an expert level of proficiency in playing a wide range of games as the model learns the optimal strategy to advance.

Recommendation systems: Q-learning models can help optimize recommendation systems, such as advertising platforms. For example, an ad system that recommends products commonly bought together can be optimized based on what users select.

Robotics: Q-learning models can help train robots to execute various tasks, such as object manipulation, obstacle avoidance and transportation.

Self-driving cars: Autonomous vehicles use many different models, and Q-learning models help train models to make driving decisions, such as when to switch lanes or stop.

Supply chain management: The flow of goods and services as part of supply chain management can be improved with Q-learning models to help find the optimized path for products to market.

Applications Of Reinforcement Learning:

Reinforcement learning (RL) has a wide range of applications across various domains. Here are some common uses of reinforcement learning:

1. Game Playing: RL has been successful in game-playing scenarios. For instance, AlphaGo, developed by DeepMind, used RL techniques to defeat world champion Go players. RL has also been applied to games like chess, poker, and video games, where agents learn to make strategic decisions and optimize their gameplay.
2. Robotics: RL plays a vital role in robotics, enabling robots to learn complex tasks and adapt to dynamic environments. RL can be used to train robots to perform tasks such as object manipulation, locomotion, grasping, and navigation. By interacting with the environment and receiving rewards, robots can learn optimal policies for accomplishing tasks.
3. Autonomous Vehicles: Reinforcement learning can be employed to train autonomous vehicles, such as self-driving cars and drones. RL algorithms enable vehicles to learn how to navigate, make decisions, and respond to traffic conditions, while considering safety and efficiency.
4. Resource Management: RL can be utilized in optimizing resource allocation and management problems. For example, in energy management, RL can help balance the electricity grid, optimize power consumption, and allocate resources efficiently. RL can also be applied to network routing, inventory management, and supply chain optimization.
5. Healthcare: Reinforcement learning has applications in healthcare, including personalized treatment recommendation, optimizing drug dosage, and disease management. RL can help healthcare providers make decisions based on patient data, treatment outcomes, and medical guidelines.
6. Finance and Trading: RL algorithms have been employed in financial markets for tasks such as portfolio management, algorithmic trading, and risk assessment. RL can learn optimal trading strategies by considering market conditions, historical data, and financial indicators.
7. Natural Language Processing (NLP): RL can be utilized in NLP tasks such as dialogue systems, machine translation, and information retrieval. Agents can learn to generate responses, translate languages, or retrieve relevant information based on the interactions with users or the analysis of textual data.
8. Recommendation Systems: RL algorithms can optimize recommendation systems by learning to make personalized recommendations based on user feedback and preferences. Agents can learn to suggest products, movies, music, or content tailored to individual users.

These are just a few examples of the many areas where reinforcement learning is applied. RL's ability to learn from interactions and optimize decision-making in dynamic environments makes it a powerful technique for solving complex problems across various domains.

Applications of Machine Learning:

1. Text Classification

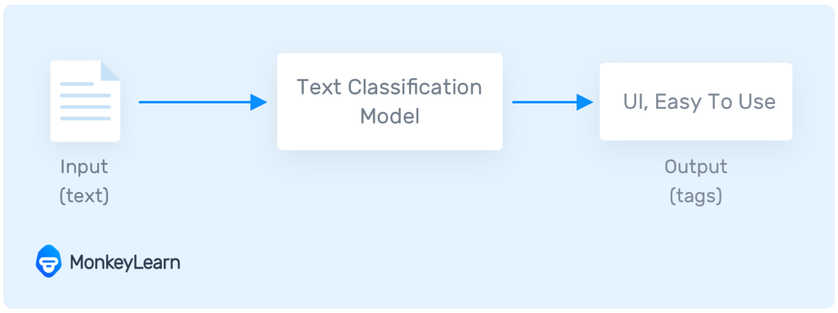
Text classification is a machine learning technique that assigns a set of predefined categories to open-ended text. Text classifiers can be used to organize, structure, and categorize pretty much any kind of text – from documents, medical studies and files, and all over the web.

Text classification is one of the fundamental tasks in natural language processing with broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection.

Here's an example of how it works:

"The user interface is quite straightforward and easy to use."

A text classifier can take this phrase as an input, analyze its content, and then automatically assign relevant tags, such as UI and Easy To Use.



Why is Text Classification Important?

It's estimated that around 80% of all information is unstructured, with text being one of the most common types of unstructured data. Because of the messy nature of text, analyzing, understanding, organizing, and sorting through text data is hard and time-consuming, so most companies fail to use it to its full potential.

This is where text classification with machine learning comes in. Using text classifiers, companies can automatically structure all manner of relevant text, from emails, legal documents, social media, chatbots, surveys, and more in a fast and cost-effective way. This allows companies to save time analyzing text data, automate business processes, and make data-driven business decisions.

Why use machine learning text classification? Some of the top reasons:

Scalability

Manually analyzing and organizing is slow and much less accurate.. Machine learning can automatically analyze millions of surveys, comments, emails, etc., at a fraction of the cost, often in just a few minutes. Text classification tools are scalable to any business needs, large or small.

Real-time analysis

There are critical situations that companies need to identify as soon as possible and take immediate action (e.g., PR crises on social media). Machine learning text classification can follow your brand mentions constantly and in real time, so you'll identify critical information and be able to take action right away.

Consistent criteria

Human annotators make mistakes when classifying text data due to distractions, fatigue, and boredom, and human subjectivity creates inconsistent criteria. Machine learning, on the other hand, applies the same lens and criteria to all data and results. Once a text classification model is properly trained it performs with unsurpassed accuracy.

How Does Text Classification Work?

Automatic text classification applies machine learning, natural language processing (NLP), and other AI-guided techniques to automatically classify text in a faster, more cost-effective, and more accurate manner.

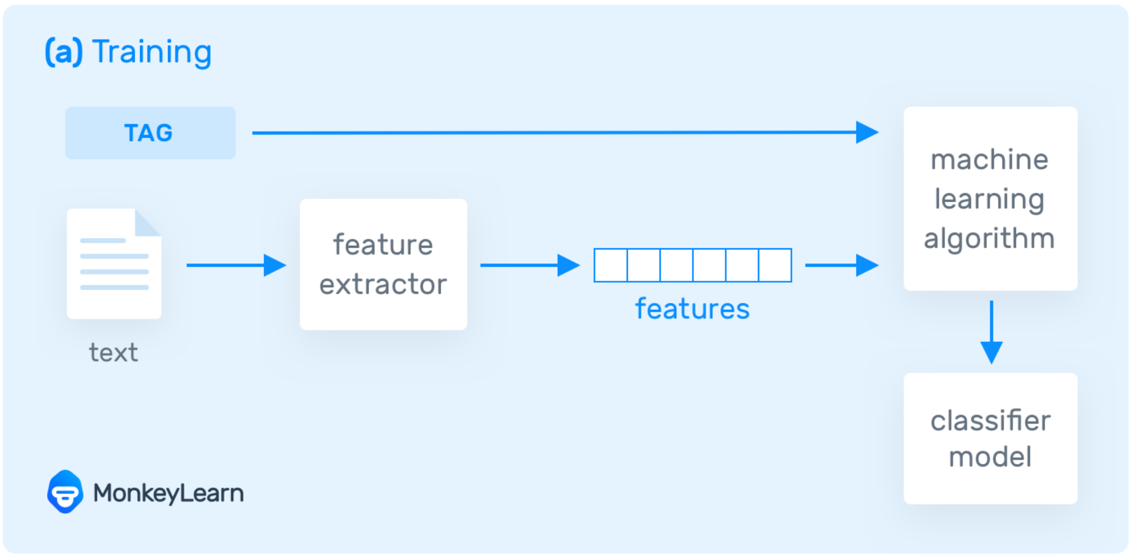
Machine learning based systems

Instead of relying on manually crafted rules, machine learning text classification learns to make classifications based on past observations. By using pre-labeled examples as training data, machine learning algorithms can learn the different associations between pieces of text, and that a particular output (i.e., tags) is expected for a particular input (i.e., text). A “tag” is the pre-determined classification or category that any given text could fall into.

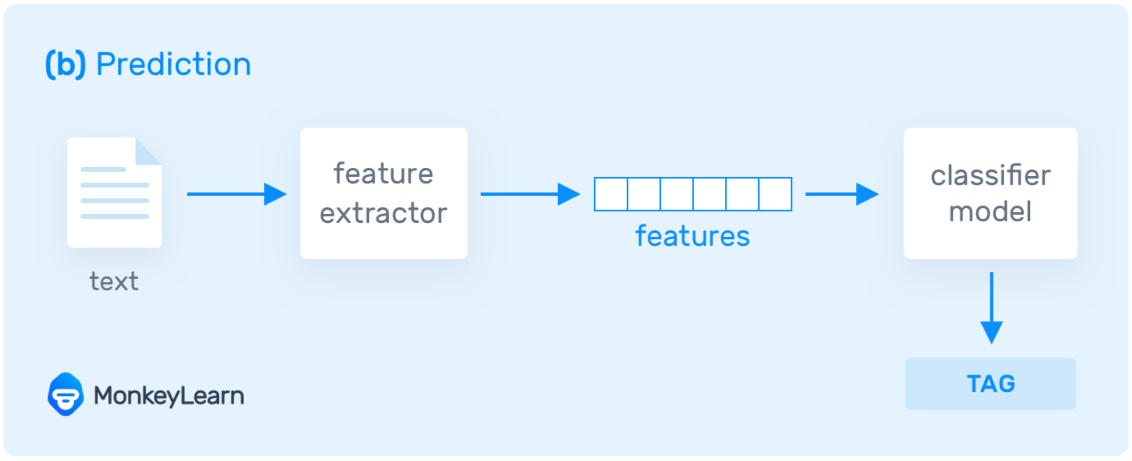
The first step towards training a machine learning NLP classifier is feature extraction: a method is used to transform each text into a numerical representation in the form of a vector. One of the most frequently used approaches is bag of words, where a vector represents the frequency of a word in a predefined dictionary of words.

For example, if we have defined our dictionary to have the following words {This, is, the, not, awesome, bad, basketball}, and we wanted to vectorize the text “This is awesome,” we would have the following vector representation of that text: (1, 1, 0, 0, 1, 0, 0).

Then, the machine learning algorithm is fed with training data that consists of pairs of feature sets (vectors for each text example) and tags (e.g. sports, politics) to produce a classification model:



Once it's trained with enough training samples, the machine learning model can begin to make accurate predictions. The same feature extractor is used to transform unseen text to feature sets, which can be fed into the classification model to get predictions on tags (e.g., sports, politics):



Text classification with machine learning is usually much more accurate than human-crafted rule systems, especially on complex NLP classification tasks. Also, classifiers with machine learning are easier to maintain and you can always tag new examples to learn new tasks.

Machine Learning Text Classification Algorithms

Some of the most popular text classification algorithms include the Naive Bayes family of algorithms, support vector machines (SVM), and deep learning.

Naive Bayes

The Naive Bayes family of statistical algorithms are some of the most used algorithms in text classification and text analysis, overall.

One of the members of that family is Multinomial Naive Bayes (MNB) with a huge advantage, that you can get really good results even when your dataset isn't very large (~ a couple of thousand tagged samples) and computational resources are scarce.

Naive Bayes is based on Bayes's Theorem, which helps us compute the conditional probabilities of the occurrence of two events, based on the probabilities of the occurrence of each individual event. So we're calculating the probability of each tag for a given text, and then outputting the tag with the highest probability.

The probability of A, if B is true, is equal to the probability of B, if A is true, times the probability of A being true, divided by the probability of B being true.

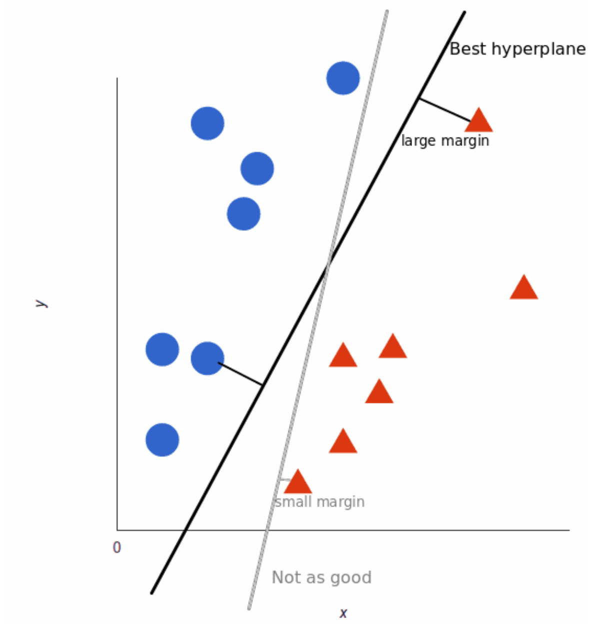
This means that any vector that represents a text will have to contain information about the probabilities of the appearance of certain words within the texts of a given category, so that the algorithm can compute the likelihood of that text belonging to the category.

Support Vector Machines

Support Vector Machines (SVM) is another powerful text classification machine learning algorithm, becauseike Naive Bayes, SVM doesn't need much training data to start providing accurate results. SVM does, however, require more computational resources than Naive Bayes, but the results are even faster and more accurate.

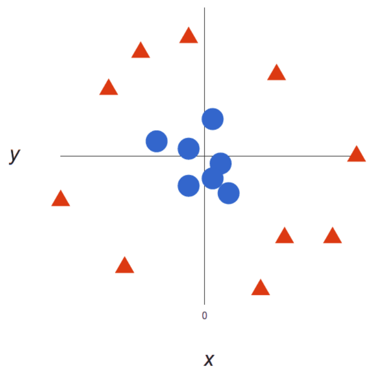
In short, SVM draws a line or "hyperplane" that divides a space into two subspaces. One subspace contains vectors (tags) that belong to a group, and another subspace contains vectors that do not belong to that group.

The optimal hyperplane is the one with the largest distance between each tag. In two dimensions it looks like this:



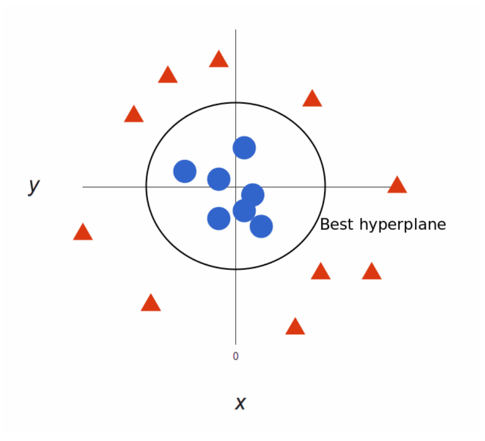
Those vectors are representations of your training texts, and a group is a tag you have tagged your texts with.

As data gets more complex, it may not be possible to classify vectors/tags into only two categories. So, it looks like this:



But that's the great thing about SVM algorithms – they're "multi-dimensional." So, the more complex the data, the more accurate the results will be. Imagine the above in three dimensions, with an added Z-axis, to create a circle.

Mapped back to two dimensions the ideal hyperplane looks like this:



2. Image Recognition

Image Recognition is one of the reasons behind the boom one could have experienced in the field of Deep Learning. The task which started from classification between cats and dog images has now evolved up to the level of Face Recognition and real-world use cases based on that like employee attendance tracking.

Also, image recognition has helped revolutionized the healthcare industry by employing smart systems in disease recognition and diagnosis methodologies.

3. Speech Recognition

Speech Recognition based smart systems like Alexa and Siri have certainly come across and used to communicate with them. In the backend, these systems are based basically on Speech Recognition systems. These systems are designed such that they can convert voice instructions into text.

One more application of the Speech recognition that we can encounter in our day-to-day life is that of performing Google searches just by speaking to it.