Artificial Intelligence

For COMP4431/COMP4432/AMA4680/AMA564

Spring 2024

In the name of **Neuro Sama**

Prepared by Pony, B.Sc(Physics, AIDA(specializes in computer vision), PolyU)

Special Thanks to Prof. H.Li, BEng(Kyoto), MEng(Kyoto), PhD in CS(Tokyo)

Special Thanks to C.C.Aggarwal BEng(IIT), PhD(MIT)

1 General Denotation and Preface

1. T: Training dataset
2. (x,y)/(X,Y) : Input/Output data pairs

Human dreamed of simulating intelligence in computer to help them to deal with a lot of tasks automatically, which is called artificial intelligence, an abstracted definition can be given by:

Define an agent $\mathcal{P}$ which called percepts, which can get data or sense data from the environments, and define another agent $\mathcal{A}$ called actuator, which acts on the same environment or give response, we define the function $f : \mathcal{P} \to \mathcal{A}$ as an **Artificial Intelligence**, in other words, artificial intelligence is a function which learns from environment and response properly.

Determinism and Non-Determinism in AI:

Determinism is the philosophical stance that every event or state of affairs is determined by preceding events and the laws of nature. In the context of AI, a deterministic system is one where the same input will always produce the same output, given the same initial conditions. Deterministic approaches are often favored in scenarios where predictability and reproducibility are critical, such as in control systems or algorithmic decision-making.

On the other hand, non-determinism introduces an element of randomness or uncertainty into the system. Non-deterministic approaches, often rooted in probability theory, acknowledge that real-world environments are inherently uncertain and that perfect predictability is often unattainable. Probabilistic models, such as those used in Bayesian inference or stochastic optimization, embrace this uncertainty and provide a framework for reasoning about likelihoods and making decisions under incomplete information.

Think about it, when you can hold a water bottle firmly, do you believe that it is your brain processed all the inputs from your perceptions(smell, touch, sight, etc...) and produces correct signals to every muscles in your hand and arm, or you believe that the magnitude of each signals sample from a very complex probability distribution such that you can always create a correct combination of signals to let you hold water bottle firmly?(Assuming you cannot hold it firmly as a rare event due to uncertainty of probability).

The debate of whether Artificial Intelligence is mathematically deterministic by a function or probability is still on going, and we still do not have an explicit or clear solution to this question.

Artificial Intelligence (AI), Machine Learning (ML), and Deep Learning (DL) are often used interchangeably, but they represent distinct concepts within the broader field of computational intelligence. AI is the overarching discipline that aims to create systems capable of performing tasks that typically require human intelligence, such as reasoning, problem-solving, perception, and decision-making. It encompasses a wide range of techniques, from rule-based systems to advanced learning algorithms. Machine Learning, a subset of AI, focuses on developing algorithms that enable systems to learn from data and improve their performance on specific tasks without being explicitly programmed. ML techniques include supervised learning, unsupervised learning, and reinforcement learning, and they rely on statistical methods to identify patterns and make predictions. Deep Learning, a specialized branch of ML, uses artificial neural networks with multiple layers (hence "deep") to model complex relationships in data. DL excels at tasks involving large-scale data, such as image recognition, natural language processing, and speech recognition, by automatically learning hierarchical representations of data. In summary, AI is the broad goal of creating intelligent systems, ML is the approach to achieving this goal through data-driven learning, and DL is a powerful technique within ML that leverages deep neural networks for advanced pattern recognition and decision-making.

The Three Schools of Thought in Artificial Intelligence Artificial Intelligence (AI) is a multifaceted field that has evolved through various philosophical and methodological approaches. Among these, three key schools of thought have shaped the development of AI systems: logic and rule-based systems, connectivism, and reinforcement learning (RL). Each school offers a unique perspective on how intelligence can be modeled and implemented in machines.

1. Logic and Rule-Based Systems The logic and rule-based approach is rooted in the idea that intelligence can be achieved through formal reasoning and symbolic manipulation. This school of thought relies on predefined rules, logical structures, and symbolic representations of knowledge to solve problems. Early AI systems, such as expert systems, were built using this approach, where human expertise was encoded into a set of rules that the system could follow to make decisions. For example, a medical diagnosis system might use a series of "if-then" rules to infer diseases based on symptoms. While powerful for well-defined and structured problems, this approach struggles with ambiguity, uncertainty, and the complexity of real-world environments.

2. Connectivism Connectivism is inspired by the structure and function of biological neural networks, particularly the human brain. This school of thought emphasizes learning through the interaction of interconnected nodes, or neurons, that collectively process information. In AI, connectivism is the foundation of Artificial Neural Networks (ANNs) and Deep Learning (DL), where layers of neurons learn hierarchical representations of data. This approach excels at tasks involving large-scale, unstructured data, such as image recognition, natural language processing, and speech synthesis. Unlike rule-based systems, connectivist models learn directly from data, enabling them to capture complex patterns and relationships without explicit programming.

3. Reinforcement Learning (RL) Reinforcement Learning (RL) is grounded in the principles of behaviorism, where learning occurs through interaction with an environment and feedback in the form of rewards or penalties. In RL, an agent learns to take actions that maximize cumulative rewards over time by exploring the environment and refining its strategy through trial and error. This approach is particularly effective for dynamic and sequential decision-making tasks, such as game playing (e.g., AlphaGo), robotics, and autonomous systems. Unlike logic-based systems, RL does not rely on predefined rules, and unlike connectivism, it focuses on learning through interaction

rather than static data.

In recent 3 decades, with the skyrocketed computational power of computer, the "black box approach" soon replaced the traditional statistical/rule based method, which equips the computer to learning features itself rather than learning how to "create a system to fit the features we assigned to it", but that does not mean traditional techniques are not worth-learning, they still give you hints or insights on how a "neural network" is created from a bunch of 0 and 1.

This chapter will end with a famous quote said by John Von.Neumann:

"If people do not believe that mathematics is simple, it is only because they do not realize how complicated life is." John von Neumann

Indeed human is much more complicated than machines, we already created computer based on quantum principles after 3000 years of physics research, but human? We have no idea, until now.

2 Introduction

In this section, i will be giving you a brief and comprehensive introduction to the question:

What is machine learning?

We can divide the term into two words, "machine" and "learning", machine means computers, or processors, generally means every artificial objects that can be used to run some algorithms and compute the output.

Then what is "learning"? Herbert A.Simon once defined leaning as "a system's ability to improve its performance by executing a process", machine learning can be seen as **an extension of statistical inference**. Because real-world data are not always ideally interpretable and explainable, we might have no idea as to "what these data are inferring". So we certainly need some more techniques to "process" the data a bit to know what is the meaning behind these observed data. One of the very important distinguishable characteristics between statistical inference and machine learning is, statistical inference algorithm does not improve itself to interpretate the data better, but machine learning will.

So you may have idea of how learning is related to data, the purpose of machine learning is to design an algorithm for the machine to learn the relationships in data, the relationship might be obvious, might not be(well finding hidden relationship inside the data is actually what Artificial Intelligence is doing), once the machine learns the relationship in data, it helps us to predict what is going to happen with some unseen inputs, or analyze the data.

2.1 Methods of machine learning

Although there does not exist a big theory that can explain everything in machine learning, we have concluded an effective path of building a machine learning algorithm that used to work in real-world:

(1) Acquire some training data
(2) Determine the set of possible learning models
(3) Determine the learning strategy for the models
(4) Implement the algorithm for learning
(5) Choose the optimal model with best performance
(6) use the learned optimal model to predict or analyze new data

2.2 Three types of classification categories

There are 2 main big task in machine learning, classification and regression, classification generally means "labeling data", regression means finding an exact formula to fit the relationship in the input data(like linear regression, logarithmic regression, quadratic,... etc) In this subsection we only discuss the task of classification.

2.2.1 Supervised Learning

As i mentioned previously, classification is about labeling data, when the labels of all input data are given, we call it as **Supervised learning**, its like some people are "supervising" the model to learn because each input data has a label. **The essence of supervised learning is to learn the relationship/statistical relationship/probability distribution** of the input and output data.

Lets build up a mathematical foundation for supervised learning ! We denote input training data pair as (x, y), x is called **features**, y is called **label**, why x is called features is because we believe why machine can do classification is it learns **some characteristics/patterns in the training data**, just like you rely on shapes, colors to classify an apple. Usually we should have more than 1 pair, assume we have nth pairs of training data, hence we can denote the set of training pairs as T:

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$$

We will use this notation in the upcoming section, so remember that ! In the previous example we said that we rely on "colors", "shape" to classify an apple, so it reveals that **we don't only rely on 1 feature to classify something very accurately**, usually we need multiple features to determine the label of an object, so to represent it mathematically, x is a d-dimensional vector assume we have d features.

$$\vec{x} = (x^{(1)}, x^{(2)}, ..., x^{(d)})$$

In fact the relationship between input and output should be viewed **probabilistically**, one assumption we made while doing machine learning is that there exists some unknown joint probability distribution between input X and output Y, our goal can simply be written as learning this probability distribution, and the training data set is **sampled** from this probability distribution.

However just like there are multiple groups of people in chinese martial arts, the supervised learning model can be either a probabilistic model $P(Y|X)$(Conditional probability) or a deterministic function $Y = f(x)$, it depends on the starting point of your algorithm. When we are doing prediction using the model, we simply write the output as $P(y|x)$ or $y = f(x)$.

Specifically, if we have nth input training data and the input data we used for prediction is x_{n+1}, then:

$$y_{n+1} = f(x_{n+1})$$

or

$$y_{n+1} = arg\max_{y} P(y|x_{n+1})$$

The meaning behind these two formulas are trivial and obvious.

2.3 Unsupervised Learning

Unsupervised learning is a type of machine learning where the algorithm is tasked with finding patterns, structures, or relationships in data without explicit guidance or labeled outcomes. Unlike supervised learning, which relies on labeled datasets to learn mappings between inputs and outputs, unsupervised learning works with raw, unlabeled data, making it particularly useful for exploratory analysis and discovering hidden insights. Common techniques include clustering, where data is grouped into meaningful categories (e.g., k-means clustering), and dimensionality reduction, which simplifies data while preserving its essential structure (e.g., Principal Component Analysis). Unsupervised learning is widely applied in areas such as customer segmentation, anomaly detection, and feature extraction, enabling systems to uncover intrinsic patterns and relationships that might not be immediately apparent.

2.4 Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning where an agent learns to make decisions by interacting with an environment to maximize cumulative rewards. Unlike supervised or unsupervised learning, RL does not rely on labeled data or predefined patterns. Instead, the agent explores the environment through trial and error, receiving feedback in the form of rewards or penalties based on its actions. Over time, the agent develops a strategy, or policy, to take actions that lead to the highest long-term rewards. RL is inspired by behavioral psychology and is particularly well-suited for dynamic and sequential decision-making tasks, such as game playing (e.g., AlphaGo), robotics, autonomous driving, and resource management. Its ability to learn optimal behaviors in complex, uncertain environments makes it a powerful tool for solving real-world problems where explicit supervision is impractical.

2.5 Model evaluation

As we mentioned, machine learning aims to create a model which have good predictive ability, to evaluate the performance of model, we use the loss function, assume a trained model $Y = f(x)$,the average loss of the model is:

$$R_{emp}(f) = \frac{1}{N} \sum_{i=1} L(y_i, f(x_i))$$

3 Computational Learning theory

3.1 Learnability

Around the year of 1984, computational learning theory came out, and it can be treated as a mathematical foundation theory of machine learning, the idea of "learnable machines" come from the following 3 conditions:

(1): The machines can provably learn whole classes of concepts. And these classes can be characterized.

(2): The classes of concepts are appropriate and nontrivial for general-purpose knowledge.

(3): The computational process by which the machines deduce the desired programs requires a polynomial number of steps.

One of the key note is that, human can learn new concept without being explicitly programmed to learn, so machine should also achieve that.

3.2 PAC Learning

PAC learning stands for "Probably approximately correct learning", this means the machine has to select the optimal function from the group of candidate function, the learner must can learn a concept from the data, the learner also have the highest probability of getting lowest generalization error. Lets begin introducing ML from the simplest possible case - Binary case.

In binary classification case, let $\mathcal{X}$ denoted as the sample space which contains the data, and c be a binary function which $c : \mathcal{X} \to \{0, 1\}$, if x is a positive sample, then $c(x_i) = 1$, else, $c(x_i) = 0$. Denote a set of hypothesis $H = \{h|h : \mathcal{X} : \{0, 1\}\}$, then, by using the training algorithm T, the hypothesis with the lowest generalization error is selected to be the relationship between the set $\mathcal{X}$ and $\{0, 1\}$, in other perspective, the selected h can be seen as an approximation to function c,

4 Perceptron

Perceptron is one of the earliest and simplest binary classification model, it is simply creating a hyperplane to separate the training data into two types, which the generalization of it is neural network(multilayer perceptron) and Support vector machine.

4.1 Perceptron model

Assume the input feature space $\mathcal{X} \in \mathbb{R}^n$, and the output space is $\mathcal{Y} = \{0, 1\}$, the following function is a perceptron from the input space to output space:

$$f(x) = sign(w \cdot x + b)$$

It is much like the equation of straight line in mathematics:

$$y = mx + c$$

You can generalize the concept of equation of straight line into perceptron, they are basically the same thing in the 2D case, however the parameter w is usually a d-dimensional vector in a

d-dimensional cases where we have d inputs.

The sign function acts as a well-structured binary classifier where its definition is given by:

$$sign(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases}$$

4.2 Learning of perceptron

Definition 1. Given a dataset T with n pairs of points, where $y_i \; in\{+1, -1\}$, if there exists a hyper plane S $w \cdot x + b = 0$ such that it can precisely divide the positive instance points and negative instance points into both sides of hyperplane, then we call the dataset as a linearly separable dataset, otherwise, it is linearly inseparable.

The learning of perceptron aims to find a separating plane that can separate training set positive and negative instance points entirely and correctly, hence we need to determine the optimal choice of w and b, but how do we measure **How far is this model compared to the optimal one?**, hence, to quantify the distance, we need a loss function.

Consider each misclassified point, one of the very natural and precise definition of loss is "how far this misclassified point is compared to the perceptron plane?", hence we can employ a distance as its loss:

$$d = \frac{|w \cdot x_0 + b|}{||w||}$$

Secondly, for misclassified points:

$$-y_i(w \cdot x_i + b) > 0$$

This is because $w \cdot x + b > 0, y_i = -1$, when $w \cdot x + b < 0, y_i = +1$. Therefore for the misclassified point x_i, the distance from the point to the hyperplane S is :

$$-\frac{y_i(w \cdot x + b)}{||w||}$$

Assume the set of misclassified points in the hyperplane S is M, then the total distance can be represented using a sum:

$$\frac{-1}{||w||} \sum_{x_i \in M} y_i(w \cdot x_i + b)$$

Without considering the factor $\frac{-1}{||w||}$, the loss function of perceptron can be derived:

$$L(w, b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

Hence the perceptron learning problem is transformed into an optimization problem of minimalizing the loss function, the loss function quantifies the "loss" of this model, which helps us to transform into an optimization problem, hence the choice of loss function is crucial to AI.

Now lets consider the minimalization problem:

$$\min_{w,b} L(w,b) = -\sum_{x_i \in M} y_i(w \cdot x_i + b)$$

Using the gradient descent method, we can minimalize the loss function $L(w,b)$, at first the initial parameters w_0, b_0 are chosen arbitrarily, instead of updating parameters for all misclassified points, stochastic gradient descent randomly chooses 1 misclassified points and runs gradient descent.

Assume we have a set of misclassified points M, t he gradient of loss function $L(w,b)$ is given by:

$$\nabla_w L(w,b) = -\sum_{x_i \in M} y_i x_i$$

$$\nabla_b L(w,b) = -\sum_{x_i \in M} y_i$$

A misclassified point (x_i, y_i) in M is randomly selected to update w,b:

$$w \longleftarrow w + \alpha y_i x_i$$

$$b \longleftarrow b + \alpha y_i$$

The α is called step size or learning rate, and this constructs the primal form of our perceptron learning algorithm. This algorithm will keep looping until there are no misclassified points.

5 Support Vector Machine

Support vector machine can be seen as an extension of perceptron, which it is also doing binary classification, what SVM is solving is the "confidence" problem, in perceptron we are only making the misclassified points to 0, but we wont consider the "confidence" problem, imagine we have a perceptron which does not misclassify the points, however one of he possible situation is that the points are very close to the perceptron hyperplane, which if we add slight noise into the data, the data will be shifted a bit, if the distance between the points and hyperplane is small enough, it will misclassify the data points, hence SVM not only emphasize the correctness of labels, but also "how confident you are that this point is this class".

While the training data is linearly separable, the SVM is known as **hard-margin SVM**, when the training data is not linearly separable, the SVM is known as **soft-margin SVM**, the formalization of SVM when training data is linearly inseparable is more complex, hence lets take a look at the linearly separable case first

5.1 SVM in linearly separable case

In this case the given condition is same as we have in perceptron, we are dealing with a classical binary classification problem with label +1 or -1 and a training dataset T, the goal is to find a hyperplane which perfectly divides the data into 2 classes.

However, in perceptron, there are infinitely many solution of hyperplane S to divides the data into 2 classes, this is due to the axiom of completeness of real number system, if you move the hyperplane slightly, it is still a perceptron hyperplane, but in SVM, there is only one optimal solution of hyperplane as it added one more condition called **margin maximization**.

In a linearly separable case, it's decision function is same as perceptron:

$$f(x) = sign(w \cdot x + b)$$

To ensure maximum confidence of classification, we need to select the point which has shortest distance with the hyperplane, and then maximize it:

Definition 2. (Functional Margin) Define the function margin(distance) of point (x_i, y_i) with arbitrary label to the hyperplane as:

$$\hat{\gamma_i} = y_i(w \cdot x_i + b)$$

And:

$$\hat{\gamma} = \min_{1 \leq i \leq N} \gamma_i$$

Which it measures the confidence of classification of the SVM model.

In general, the distance between the point x_i and the hyperplane (w, b) is :

$$\gamma_i = y_i(\frac{w \cdot x_i}{||w||} + \frac{b}{||w||})$$

$$\gamma = \min_{1 \leq i \leq N} \gamma_i$$

It is called the **geometric margin**.

Hence we know that:

$$\gamma_i = \frac{\hat{\gamma_i}}{||w||}$$

$$\gamma = \frac{\hat{\gamma}}{||w||}$$

5.2 Maximum Margin

As our goal is to find the maximum margin, it is natural to convert the problem into a marginal maximazation problem, consider the following optimization problem:

$$\max_{w,b} \gamma = \max_{w,b} \frac{\hat{\gamma}}{||w||}$$

s.t.

$$y_i(\frac{w \cdot x_i}{||w||} + \frac{b}{||w||}) \geq \gamma$$

For simplicity, we usually take $\hat{\gamma} = 1$ because it is proportional to w and b, if $\hat{\gamma}$ change, w and b also change. Hence the optimization problem is equivalent to

$$\max_{w,b} \gamma = \max_{w,b} \frac{||w||^2}{2}$$

s.t.

$$y_i(\frac{w \cdot x_i}{||w||} + \frac{b}{||w||}) \geq 1$$

(1) :To solve this minmalization problem, we can use the Lagrange multiplier method, the lagrange function is given by:

$$\mathcal{L}(w, b, \alpha) = \frac{||w||^2}{2} - \sum_{i=1}^{N} \alpha_i y_i (w \cdot x_i + b) + \sum_{i=1}^{N} \alpha_i$$

where *alpha* is an Lagrange multiplier method.

According to lagrange duality, the dual problem of the primal problem is a min-max problem:

$$\max_{\alpha} \min_{w,b} L(w, b, \alpha)$$

To obtain the optimal solution for the minimalization problem, we can use the conventional lagrange multiplier method:

$$\frac{\partial L(w, b\alpha)}{\partial w} = w - \sum_{i=1}^{N} \alpha_i y_i x_i = 0$$

$$\frac{\partial L(w, b, \alpha)}{\partial b} = -\sum_{i=1}^{N} \alpha_i y_i = 0$$

Hence we have:

$$w = \sum_{i} \alpha_i y_i x_i$$

and

$$\sum_{i} \alpha_i y_i = 0$$

By substituting the equation back into $L(w, b, \alpha)$, we have:

$$\min_{w,b} L(w, b, \alpha) = \frac{-1}{2} \sum_{i} \sum_{j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i} \alpha_i$$

(2) The maximum of the minimization problem Now we have to find such α that $\min_{w,b} L(w, b, \alpha)$ attains the maximum, recall the dual problem:

$$\max_{\alpha} \frac{-1}{2} \sum_{i} \sum_{j} \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_{i} \alpha_i$$

s.t

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Where it is equivalent to :

$$\min_\alpha \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_i \alpha_i$$

s.t

$$\sum_i \alpha_i y_i = 0$$

$$\alpha_i \geq 0$$

Corollary 1. Suppose $\alpha^* = (\alpha_1^*, \alpha_2^*, ..., \alpha_l^*)^T$ is the solution of the minimalization problem mentioned above, there exists a solution w^*, b^*:

$$w^* = \sum_i \alpha_i^* y_i x_i$$

$$b^* = y_j - \sum_i \alpha^* y_i (x_i \cdot x_j)$$

Definition 3. Support vector is the boundary vector where it across the point which is nearest to the hyperplane, it is denoted as:

$$w^* \cdot x_i + b^* = \pm 1$$

Exercise 1. Suppose we have 3 points, $x_1 = (3,3)^T, x_2 = (4,3)^T, x_3 = (1,1)^T$ while x_1, x_2 are positive instances, x_3 is negative instance, use the above algorithm to find its separating plane.

Recall the minimalization problem we built up:

$$\min_\alpha \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_i \alpha_i$$

which equals to $= \frac{1}{2}(18\alpha_1^2 + 25\alpha_2^2 + 2\alpha_3^2 + 42\alpha_1\alpha_2 - 12\alpha_1\alpha_2 - 14\alpha_2\alpha_3 - \alpha_1 - \alpha_2 - \alpha_3)$
s.t.
$\alpha_1 + \alpha_2 - \alpha_3 = 0$

$$\alpha_i \geq 0$$

Suppose the objective function J is:

$$J(\alpha_1, \alpha_2) = \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j (x_i \cdot x_j) + \sum_i \alpha_i$$

$$= 4\alpha_1^2 + \frac{13}{2}\alpha_2^2 + 10\alpha_1\alpha_2 - 2\alpha_1 - 2\alpha_2$$

The partial derivative gives us:

$$\frac{\partial J}{\partial \alpha_1} = 8\alpha_1 + 10\alpha_2 - 2 = 0$$

$$\frac{\partial J}{\partial \alpha_2} = 13\alpha_2 + 10\alpha_1 - 2 = 0$$

$$\frac{\partial J^2}{\partial \alpha_1^2} = 8$$

$$\frac{\partial J^2}{\partial \alpha_2^2} = 13$$

$$\frac{\partial J^2}{\partial \alpha_1 \alpha_2} = 10$$

By second derivative test, the determinant of Hessian matrix is 4, in order to make objective function reach relative minimum, we need $\frac{\partial J}{\partial \alpha_1} > 0$, as the minimum of α_2 is 0, let $\alpha_2 = 0$, and solve for α_1, hence $\alpha_1 = \frac{1}{4}$, $\alpha_3 = \frac{1}{4}$

By the formula above, we have $w^* = (\frac{1}{2}, \frac{1}{2})^T$, $b^* = -2$ The separating hyperplane is:

$$\frac{x^{(1)}}{2} + \frac{x^{(2)}}{2} - 2 = 0$$

And the corresponding decision function is:

$$f(x) = sign(\frac{x^{(1)}}{2} + \frac{x^{(2)}}{2} - 2)$$

5.3 SVM for linearly inseparable case

Definition 4. If the training dataset T is linearly inseparable, it does not satisfy the following condition:

$$y_i(w \cdot x_i + b) - 1 \geq 0$$

6 K-NN classifier

6.1 Introduction

The k-NN algorithm is proposed as the simplest and most intuitive classification and regression method, where we want to classify the type of input feature vector.

Algorithm: k-NN Suppose we have the training dataset: T = $\{(x_1, y_1), (x_2, y_2), ...(x_N, y_N)\}$ where $y_i \in \{c_1, c_2, ..., c_k\}$ is the class instance of each pair of training data.

(1) : According to the metrics given, the k points closest to x are found in the training set T, and the neighborhood of x that contains these k points will be denoted $N_k(x)$;

(2) : Determine y with given $N_k(x)$, we usually use majority voting, which is to find the class that appeared most frequently in the neighborhood:

$$y = arg\max_{c_j} \sum_{x_i \in N_k(x)} I(y_i = c_j)$$

i = 1,2,3..., N:j = 1,2,...K I is the indicator function which its output is given by:

$$I = \begin{cases} 1, & y_i = c_j \\ 0, & \text{otherwise} \end{cases}$$

The k-NN will not update itself, hence it does not process learning.

6.2 Distance Metrics

Assume a n-dimension feature vector space for the input vector, that is, $\mathbf{R}^n$, the distance we usually use is **Euclidean Distance**, but there are also other distance metrics; Minkowski distance, as a generalization of euclidean distance, is defined as:

$$\mathcal{L}_p(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p)^{\frac{1}{p}}$$

when p=2, it is euclidean distance:

$$\mathcal{L}_2(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^2)^{\frac{1}{2}}$$

when p=1, it is manhattan distance:

$$\mathcal{L}_1(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|)$$

6.3 Probabilistic View of k-NN

The probability of misclassifying the classes can be represented as:

$$P(Y \neq f(X)) = 1 - P(Y = f(X))$$

6.4 Selection of k

The choice of the value of k has a significant impact to the performance of the k-NN classifier model, once a smaller k value is chosen, the size of neighborhood $N_k(x)$ will be decreased, and the selected data points will have a **more similar property** to the center, however, if the data points appeared to be noisy, then a smaller k value can apparently be disadvantageous to the classification because of the increased sensitivity to distance metrics

7 Naive Bayes Classifier

7.1 Basics of Naive Bayes Classifier

Suppose the input space $\mathcal{X} \subseteq \mathbf{R^n}$ be a set of n-dimension vectors and the output space is a class label collection $\mathcal{Y} = \{c_1, c_2, ...c_k\}$, we assume there is a joint probability distribution between the space $\mathcal{X}$ and $\mathcal{Y}$, hence we will have the training dataset T:

$$T = \{(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\}$$

The naive Bayes classifier learns the prior probability and the conditional probability, because we are maximizing the conditional probability of correct label with the condition of knowing the input, hence we have:

$$P(X = x|Y = c_k) = P(X^1 = x^1, X^2 = x^2, ..., X^n = x^n|Y = c_k) = \prod_{j=1}^{n} P(X^j = x^j|Y = c_k)$$

Because we assume that the training data is **sampled** from the joint probability distribution $P(X, Y)$, and also each time of sampling is independent, hence they sampling are independent events, and according to the basic probability rules of event independence:

$$P(\bigcup_i A_i) = \sum_i P(A_i)$$

In Naive Bayes classification, the given condition is the input data, hence we can calculate the conditional probability $P(Y = c_k|X = x)$, the class with the maximum probability will be determined as the class output of x, According to the Bayes rule, we have:

$$P(Y = c_k|X = x) = \frac{P(X = x|Y = c_k)P(Y = c_k)}{\sum_k P(X = x|Y = c_k)P(Y = c_k)}$$

$$P(Y = c_k|X = x) = \frac{P(Y = c_k)\prod_{j=1}^{n} P(X^j = x^j|Y = c_k)}{\sum_k P(Y = c_k)\prod_{j=1}^{n} P(X^j = x^j|Y = c_k)}$$

We take the class with maximum probability as the output of the classifier, hence we transform the naive bayes into a classifier:

$$y = f(x) = arg\max_{c_k} \frac{P(Y = c_k)\prod_{j=1}^{n} P(X^j = x^j|Y = c_k)}{\sum_k P(Y = c_k)\prod_{j=1}^{n} P(X^j = x^j|Y = c_k)}$$

Note that in the denominator, we are taking the summation, the nature of determining maximum is comparison, whenever there are common factors, they can be ignored, hence it can be ignored when we are considering the class which attains the maximum probability:

$$y = f(x) = arg\max_{c_k} P(X = x|Y = c_k)\prod_{j=1}^{n} P(X^j = x^j|Y = c_k)$$

Since we have a probabilistic decision function, we can transform the learning problem into a loss function minimalization problem: Assume the loss function is a 0-1 loss function: L(Y, f(X)) =

$$\begin{cases} 1, & Y \neq f(X) \\ 0, & Y = f(X) \end{cases}$$

where f is the decision function we just derived, according to the big principle - Risk minimalization principle, the expected risk of the model is:

$$R_{exp}(f) = \mathbb{E}[L(Y, f(X)]$$

The probability $P(x, y) = P(x)P(y|x)$ Hence the conditional expectation is:

$$R_{exp}(f) = E_X \sum_{k=1}^{K} P(c_k|X)[L(c_k, f(X)]$$

The reason we can write c_k instead of Y is because it is doing prediction, once the output of decision function not equal to the c_k, it has loss.

To minimize the expected risk, we are actually minimizing the **expectation of probability of misclassifying classes**, hence we directly have:

$$f(x) = arg \min_{y \in \mathcal{Y}} P(y = c_k|X = x)$$

Or equivalently:

$$f(x) = arg \max_{c_k} P(c_k|X = x)$$

Definition 5. (Naive Bayes Algorithm) $T = (x_1, y_1), (x_2, y_2)...., (x_N, y_N)$, where x_i is a nth dimensional vector, the jth feature of ith sample is denoted by $x_i^{(j)}$, the output of this algorithm of the class of an input test instance x. The sample space is :

$$x_i^{(j)} \in \{a_{j1}, a_{j2}, ..., a_{js}\}$$

$$y_i \in \{c_1, c_2, ..., c_K\}$$

(1) Calculate the prior and conditional probability

$$P(Y = c_k) = \frac{\sum_i I(y_i = c_k)}{N}$$

$$P(X_j = a_{jl}|Y = c_k) = \frac{\sum_i I(x_i^{(j)} = a_{jl}, y_i = c_k)}{\sum_i I(y_i = c_k)}$$

(2) By using Bayes rule, compute:

$$P(X = x|Y = c_k) \prod_{j=1}^{n} P(X^j = x^j|Y = c_k) \quad \forall k \in [1, 2, ..K]$$

(3) Take maximum and determine the output class:

$$y = arg \max_{c_k} P(X = x|Y = c_k) \prod_{j=1}^{n} P(X^j = x^j|Y = c_k)$$

This is done by maximum likelihood estimation.

In the above mentioned maximum likelihood estimation, envision a very special case when all the training data belongs to the same class, although it is obvious that the model is overbiased and not accurate, however the posterior probability of other classes will become 0 because the nominator becomes 0, to prevent this accident we can use a commonly used technique called *Laplaciansmoothing*, the smoothed posterior probability is:

$$\frac{\sum_i I(x_i^{(j)} = a_{jl}, y_i = c_k) + \lambda}{\sum_i I(y_i = c_k) + S_j + \lambda}$$

The effect of λ to the whole probability is very small, plus we only consider the class with maximum probability.

8 Maximum Entropy Method

8.1 Logistic regression model

Definition 6. Suppose X is a random variable, X obeys a logistic distribution if it has the following distribution and probability density function:

$$F(x) = P(X \leq x) = \frac{1}{1 + e^{-(x-\mu)/\gamma}}$$

$$\rho(x) = F'(x) = \frac{e^{-(x-\mu)/\gamma}}{\gamma(1 + e^{-(x-\mu)/\gamma})^2}$$

Definition 7. (**Binomial Logistic Regression**) The binomial logistic regression model is a conditional probability distribution as follows:

$$P(Y = 1|x) = \frac{\exp(w \cdot x + b)}{1 + \exp(w \cdot x + b)}$$

$$P(Y = 0|x) = \frac{1}{1 + \exp(w \cdot x + b)}$$

For multi-label classification, we have multi-nomial logistic regression model, which is:

$$P(Y = k|X) = \frac{\exp(w_k \cdot x)}{1 + \sum_{i=1}^{N-1} \exp\{w_i \cdot x\}}$$

Rearranging the equations, we can get:

$$w \cdot x = \log\left(\frac{P(Y = 1|X)}{1 - P(Y = 1|X)}\right)$$

where:

$$P(Y = 1|X) = \frac{\exp(w \cdot x)}{1 + \exp(w \cdot x)}$$

If the $w \cdot x$ close to positive infinity, the closer the probability to 1, if $w \cdot x$ close to negative infinity, the closer the probability to 0.

8.2 Parameter estimation by maximum likelihood

Given the training dataset $T = \{(x_1, y_1), ..., (x_N, y_N)\}$ where $x_i \in \mathbb{R}, y_i \in \{0, 1\}$, to learn the parameter, we can apply maximum likelihood estimation to obtain logistic regression model, assume:

$$P(Y = 1|x) = p(x), \quad P(Y = 0|x) = 1 - p(x)$$

For binary classification task, the likelihood function is same as bernoulli distribution:

$$L = \prod_{i=1}^{N} p^{y_i}(x_i)[1 - p(x_i)]^{1-y_i}$$

The log likelihood is given by :

$$\mathcal{L} = \log(L) = \sum_{i=1}^{N} \log\big(p^{y_i}(x_i)[1 - p(x_i)]^{1-y_i}\big)$$

$$= \sum_{i=1}^{N} y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))$$

$$= \sum_{i=1}^{N} y_i \log \frac{p(x_i)}{1 - p(x_i)} + \log(1 - p(x_i))$$

In this way, this problem becomes an optimization problems with log-likelihood function as the objective function, the commonly used method to optimize are the gradient descent method and Quasi-Newton method.

8.3 Maximum Entropy Principle

The maximum entropy principle is an important criterion for probabilistic model learning, it states that among all combinations of parameters, the model with the maximum entropy performs the best among all models, assume we have a discrete random variable X, with a probability mass function P(X), the classical entropy is defined as:

$$H(P) = -\sum_{x} P(x) \log P(x)$$

where it satisfies the following inequalities:

$$0 \leq H(P) \leq \log |X|$$

Assume we have a training dataset T, we can already determine the joint distribution and its marginal distribution:

$$P(X = x, Y = y) = \frac{v(X = x, Y = y)}{N} \quad \textit{(Counting principal of probability)}$$

$$P(X = x) = \frac{v(X = x)}{N}$$

Which the v denotes the frequency of data in the training dataset We also need a deterministic function $f(x, y)$ to describe some certain fact between input X and output Y, since we are doing classification, its natural to have:

$$f(x,y) = \begin{cases} 1, & \text{x and y satisfy a certain fact} \\ 0, & \text{otherwise} \end{cases}$$

The expectation of the feature function f(x,y) is denoted as $E_P(f)$:

$$E_P(f) = \int_{(x,y)} P(x,y) f(x,y)$$

Since in the maximum entropy model we are given training data, which are input X, we can express using the conditional probability:

$$P(x,y) = P(x)P(y|x)$$

This is one of the important assumption of maximum entropy model.

Hence the expectation of the model:

$$E_p(f) = \int_{(x,y)} P(x)P(y|x)f(x,y)$$

And the equality constructed the first constraint of this model:

$$\int_{(x,y)} P(x,y)f(x,y) = \int_{(x,y)} P(x)P(y|x)f(x,y)$$

The classical definition of conditional entropy is:

$$H(P) = -\sum_{(x,y)} P(x)P(y|x)\ln P(y|x)$$

Proof. The conditional entropy is given by:

$$\begin{aligned} &\sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \\ -&\sum_{x\in\mathcal{X}} p(x)\cdot\sum_{y\in\mathcal{Y}} p(y|x)\ln p(y|x) \\ &-\sum_{x,y} p(x)p(y|x)\ln p(y|x) \end{aligned}$$

□

Hence we can transform it into a constrained optimization problem:

$$\max_{P\in\mathcal{P}} H(P)$$

subject to:

$$\int_{(x,y)} P(x,y)f(x,y)dxdy = \int_{(x,y)} P(x)P(y|x)f(x,y)dxdy$$

8.3.1 Learning of maximum-entropy model

The canonical definition of the constrained maximization problem is given by:

Definition 8. (**Maximum Entropy learning problem**)

$$\max_{P \in \mathcal{P}} H(P) = -\sum_{x,y} P(x)P(y|x) \ln P(y|x)$$

subject to the following constraint:

$$E_{\mathrm{p}}(f_i) = E_P(f_i) \implies \sum_{(x,y)} P(x,y) f(x,y) = \sum_{(x,y)} P(x)P(y|x) f(x,y)$$

$$\sum_{y} P(y|x) = 1$$

One of the most intuitive way to solve this problem is to use Lagrange multiplier method, hence, we shall introduce a new Lagrange function with the Lagrange multiplier:

$$\mathcal{L}(P,w) = -H(P) + w_0(1 - \sum_{y} P(y|x)) + \sum_{i=1}^{n} w_i(E_{\mathrm{p}}(f_i) - E_P(f_i))$$

$$= -\sum_{x,y} P(x)P(y|x) \ln P(y|x) + w_0(1 - \sum_{y} P(y|x)) + \sum_{i=1}^{n} w_i(\sum_{(x,y)} P(x,y) f(x,y) - \sum_{(x,y)} P(x)P(y|x) f(x,y))$$

Hence we get a dual optimization problem:

$$\min_{P \in \mathcal{P}} \max_{w} \mathcal{L}(P,w) := \max_{w} \min_{P \in \mathcal{P}} \mathcal{L}(P,w)$$

8.3.2 Dual problem optimization

First, we need to solve the minimization problem $\min_{P\in\mathcal{P}} \mathcal{L}(P, w)$, where $\mathcal{L}$ is a function of w, denoted as:

$$\Psi(w) = \min_{P\in\mathcal{P}} \mathcal{L}(P, w)$$

Recall the langrange function:

$$-\sum_{x,y} P(x)P(y|x)\ln P(y|x) + w_0(1 - \sum_{y} P(y|x)) + \sum_{i=1}^{n} w_i(\sum_{(x,y)} P(x,y)f(x,y) - \sum_{(x,y)} P(x)P(y|x)f(x,y)) \tag{1}$$

By the principle of Lagrange multiplier, we shall take the partial derivative to the objective function against $P(y|x)$. hence;

$$\begin{aligned}\frac{\partial H}{\partial P(y|x)} &= -\sum_{x,y} P(x)\cdot(P(y|x)\frac{\partial}{\partial P(y|x)}\ln P(y|x) + \frac{\partial}{\partial P(y|x)}P(y|x)) \\ &= -\sum_{x,y} P(x)(\ln P(y|x) + 1)\end{aligned}$$

For second term:

$$\frac{\partial}{\partial P(y|x)} w_0(1 - \sum_{y} P(y|x)) = -\sum_{y} w_0$$

For the third term:

$$= -\sum_{x,y}(P(x)\sum_{i=1}^{n} w_i f_i(x,y))$$

Hence the total partial derivative is:

$$\frac{\partial \mathcal{L}}{\partial P(y|x)} = \sum_{x,y} P(x)(\ln P(y|x) + 1 - w_0 - \sum_{i=1}^{n} w_i f_i(x,y))$$

Since the partial derivative is 0, we can have, by solving the equation, we can have:

$$P(y|x) = \frac{\exp\{(\sum_{i=1}^{n} w_i f_i(x,y)\})}{\exp\{(1-w_0)\}}$$

$$P_w(y|x) = \frac{1}{Z_w(x)}\exp\left\{(\sum_{i=1}^{n} w_i f_i(x,y))\right\}$$

where:

$$Z_w(x) = \sum_{y}\exp\left\{(\sum_{i=1}^{n} w_i f_i(x,y)\right\}$$

After that, we can solve the maximization problem outside the dual problem:

$$\max_{w} \Psi(w)$$

We can apply the same technique to find such w which implies it is the maximum.

8.3.3 Maximum Likelihood Estimation

By the definition of maximum likelihood estimation, the log likelihood function of the conditional probability distribution:

$$L_p(P_w) = \ln\left(\prod_{x,y} P(y|x)^{P(x,y)}\right) = \sum_{x,y} P(x,y) \ln P(y|x)$$

Assume the conditional probability distribution $P(Y|X)$ is the maximum entropy model as shown in previous subsection:

$$P_w(y|x) = \frac{1}{Z_w(x)} \exp\left\{(\sum_{i=1}^{n} w_i f_i(x,y))\right\}$$

where:

$$Z_w(x) = \sum_{y} \exp\left\{(\sum_{i=1}^{n} w_i f_i(x,y))\right\}$$

The expanded $L_p(P_w)$:

$$L_p(P_w) = \sum_{x,y} P(x,y) \sum_{i=1}^{n} w_i f_i(x,y) - \sum_{x} P(x) \ln Z_w(x)$$

Recall the Lagrange function:

$$\mathcal{L} = -\sum_{x,y} P(x)P(y|x) \ln P(y|x) + w_0(1 - \sum_{y} P(y|x)) + \sum_{i=1}^{n} w_i (\sum_{(x,y)} P(x,y) f(x,y) - \sum_{(x,y)} P(x)P(y|x) f(x,y))$$

Since we have:

$$\sum_{y} P(y|x) = 1$$

It becomes:

$$\mathcal{L} = -\sum_{x,y} P(x)P(y|x) \ln P(y|x) + \sum_{i=1}^{n} w_i (\sum_{(x,y)} P(x,y) f(x,y) - \sum_{(x,y)} P(x)P(y|x) f(x,y))$$

After doing some substitution, we can get:

$$\mathcal{L} = \sum_{x,y} P(x,y) \sum_{i=1}^{n} w_i f_i(x,y) - \sum_{x} P(x) \ln Z_w(x) = L_p(P_w)$$

Hence the **dual function optimization and maximum likelihood estimation are fundamentally equival**

9 Introduction to Various optimization methods

9.1 Gradient Descent method

Gradient descent is an intuitive and simple minimalization method, consider an objective function $f(x)$ which we wants to find its minimum(either relative or absolute). Note that the direction which the function f decreases most rapidly at a point $x_0$is :

$$-\nabla f(x_0)$$

Hence by using iterative method, for kth iteration suppose we have point x_k, the selected x_{k+1} is :

$$x_{k+1} = x_k - \nabla f(x_k)$$

By looping the iteration we can minimalize the function f.

9.2 Newton's method

Newton method and quasi-Newton method are also commonly used to solve optimization problem, it is popular of its fast convergence compared to gradient-descent.

Consider an unconstrained optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

Assuming the function $f(x)$ is a C^2 function, assume the kth iteration value of this function is $x^{(k)}$, hence by taylor series expansion:

$$f(x) = f(x^{(k)}) + \nabla_k^T (x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T H(x^{(k)})(x - x^{(k)})$$

where ∇ is the gradient of f(x) on $x^{(k)}$, and $H(x^{(k)})$ is the Hessian matrix of second-order partial derivatives:

$$H(x) = [\frac{\partial^2 f}{\partial x_i \partial x_j}]_{n \times n}$$

at the point $x^{(k)}$, in order to determine is the point $x^{(k)}$ at the minimal value, we need a second-derivative test method. If $\frac{\partial f}{\partial x}(x^{(k)}) = 0$ and $H(x^{(k)}) > 0$, then the point $x^{(k)}$ is a relative minimum.

The very first necessary condition for newton method is:

$$\nabla f = 0$$

Assume in (k+1) iteration, its gradient =0, we have:

$$\nabla f(x) = \nabla f(x^k) + H_k(x - x^k) = 0$$

Therefore we have:

$$x^{(k+1)} = x^{(k)} - H_k^{-1} \nabla_k$$

Algorithm of Newton's method:

Input : the objective function f(x), gradient $\nabla f(x)$, the Hessian matrix.

Output : a, which is the point obtaining smallest value of f(x)

(1) Take an initial value x_0 and set k=0
(2) Calculate gradient $\nabla f(x)$
(3) If $||\nabla|| < \varepsilon$, stop the calculation and the approximate point x is obtained.
(4) If not, calculate the Hessian matrix
(5) Hence, set $x^{(k+1)} = x^{(k)} - H_k^{-1}\nabla f(x_k)$
(6) Set k=k+1

The update can further be simplified to:

$$x_{n+1} = x_n - [\mathcal{H}f(x_n)]^{-1}\nabla f(x_n)$$

9.3 Quasi-Newton Method

When n becomes very large, it is computationally not feasible to calculate the Hessian matrix and the inverse of Hessian matrix, hence quasi-newton method was invented to solve this problem, instead of directly calculating, it **approximates** the Hessian matrix and its inverse, hence it is computationally cheaper than conventional Newton's method.

Suppose $\nabla f(x_{k+1}) = g_{k+1}$, we have:

$$g_{k+1} - g_k = H_k(x_{k+1} - x_k)$$

Let $y_k = g_{k+1} - g_k, \delta_k = x_{k+1} - x_k$, we have:

$$y_k = H_k\delta_k$$

which is known as **quasi-Newton conditions** Denote:

$$p_k = -H_k^{-1}\nabla f(x_k)$$

As the direction of gradient flow, by the previous newton method update, we have:

$$x_{k+1} = x_k + \lambda p_k = x_k - \lambda H_k^{-1} g_k$$

Hence the Taylor expansion can be approximated:

$$f(x) \sim f(x_k) - \lambda g_k^T H_k^{-1} g_k$$

In Quasi-Newton method, we take G_k as an approximation of the inverse Hessian H_k^{-1} and requires G_k to satisfies the same newton method condition, which is :

$$g_{k+1} - g_k = G_k(x_{k+1} - x_k)$$

9.4 Lagrange duality

In constrained optimization problem, Lagrange duality is often used to transform the original problem into a dual problem, and the solution of the original problem is obtained by solving the dual problem, this method is used in many statistical learning methods such as maximum entropy model and support vector machine.

Consider the following general constrained optimization problem:

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$s.t. \quad c_i(x) \leq 0, i = 1, 2, ..., k$$

$$h_j(x) = 0, j = 1, 2, ..., l$$

Firstly, the generalized Lagrange function is introduced by using lagrange multiplier:

$$L(x, \alpha, \beta) = f(x) + \sum_{i=1}^{k} a_i c_i(x) + \sum_{j=1}^{l} \beta_j h_j(x)$$

Here we have $x = \vec{x} = (x^{(1)}), ..., x^{(n)})^T \in \mathbb{R}^n, \alpha_i, \beta_j$ are Lagrange multipliers. This can be claimed as the **original problem**.

Definition 9. (Karush-Kuhn-Tucker Conditions) Assume an objective function $f : \mathbb{R}^n \to \mathbb{R}$, constraints $g_i : \mathbb{R}^n \to \mathbb{R}$, $h_j : \mathbb{R}^n \to \mathbb{R}$, Consider the following non-linear optimization problem:

$$\min_{x} f(x)$$

Subject to:
$g_i(x) \leq 0 \wedge h_j(x) = 0$
where $g_i(x)$ denotes the inequality constraint, $h_j(x)$ denotes the equality constraint The KKT conditions states that the necessary conditions for the above minimalization problems to be solvable is:

$\exists \lambda \geq 0, \mu_i \geq 0, v_j \geq 0$ such that :

$$\lambda + \sum_i \mu_i + \sum_j v_j > 0$$

$$\lambda \nabla f(x) + \sum_i \mu_i \nabla g_i(x) + \sum_j v_j \nabla h_j(x) = 0 \quad \textit{(Langrange Multiplier Condition)}$$

$$\mu_i g_i(x) = 0 \quad \forall i \in [1, m]$$

Consider a function of x:

$$\theta_P(x) = \max_{\alpha, \beta : \alpha_i \geq 0} L(x, \alpha, \beta)$$

Assume x is given, if x violates any constraint of the problem, that is:

$$\exists i \in \mathbb{R} \implies c_i(x) \geq 0 \vee \exists j \in \mathbb{R} \implies h_j(x) \neq 0$$

Then we have:

$$\theta_P(x) = \max_{\alpha,\beta:\alpha_i \geq 0} L(x, \alpha, \beta) \to +\infty$$

This is because if $c_i(x) > 0$, we are equivalently maximizing the $a_i c_i(x)$ where it is ¿0, it will make this term infinitely large, hence $\to +\infty$ Or in another case we have $h_j(x) = 0$, it will similarly leads to $\beta_j \to \infty$

Therefore, a more precise description of the original problem should be :

$$\min_x \theta_P(x) = \min_x \max_{\alpha,\beta:\alpha_i \geq 0} L(x, \alpha, \beta)$$

It is called the minmax problem of generalized Lagrange function, we use this notation because it avoided to case where the function $\theta_P(x) \to +\infty$,

9.4.1 Dual problem

Define:

$$\theta_D(\alpha, \beta) = \min_x L(x, \alpha, \beta)$$

Consider maximizing the function $\theta_D(\alpha, \beta)$, we have:

$$\max_{\alpha,\beta:\alpha_i \geq 0} \theta_D(\alpha, \beta) = \max_{\alpha,\beta:\alpha_i \geq 0} \min_x L(x, \alpha, \beta)$$

10 Delving into Era of Deep learning

10.1 Introduction to deep learning

Welcome to the world of Deep learning !

I believe you are familiar with the word "learning" in the previous machine learning sections, now lets take a moment to know what it means to be "deep", and what's the difference between "deep learning" and "machine learning".

First of all, you might noticed, the abovementioned machine learning techniques do not perform well in highly-complex data, especially in non-linear case, why? Because we all our deterministic functions are linear function, of course *a linear combination of linear function cannot approximate non-linear data well enough*, hence **deep learning** is invented, by incorporating non-linear computational nodes into the functions, we can approximate the non-linear relationship well enough, and it is crucial to achieve the goal of Artificial Intelligence.

The essence of deep learning is called "neural network", one can view it as a very large composite function which output the prediction with given an input, or you can view it as a computational graph which is a **hierarchical structure of computational nodes**, the most basic unit in neural network is a "neuron", which is no different to a mathematical function, output something after received the input.

The artificial neural network is a bionic product, it copies the structure of the human brain, however nowadays the human neural network still outperforms the artificial neural network we made in the computer, due to the lack of understanding on the brain function and specific structure, we still do not know how to fully copy a brain structure in computer, but we can at least approximate it in computer, and experiment shows that it outperforms traditional machine learning techniques a lot, so we seldom see people use machine learning techniques like SVM nowadays.

Recall the definition of perceptron we introduced before:

$$\hat{y} = f(X) = sign(W \cdot X^T) = sign(\sum_{j=1} w_j x_j)$$

It can be claimed as the most basic type of neural network.

10.1.1 Use of Bias and leanring algorithm of perceptron

If you envision the hyperplane in 2D case, it is simply a straight line across the origin, however, this may worsen the predictive ability of the perceptron because we cannot assume all training data are so perfect that a line across origin can classify them perfectly, hence we need a bias b to play as the y-intercept in the line equation $y = mx + c$:

$$\hat{y} = sign(W \cdot X^T + b) = sign(\sum_{j} w_j x_j + b)$$

There are few learning algorithms for perceptron, the most commonly used algorithm is the gradient descent method we introduced before, let us introduce a loss function first:

$$L_i = max\{-y_i(W \cdot X_i^T), 0\}$$

As introduced before, gradient descent method finds the direction which objective function decreases most rapidly, first let us consider the matrix notation of partial derivative:

$$\frac{\partial L_i}{\partial W} = [\frac{\partial L_i}{\partial w_1}, \frac{\partial L_i}{\partial w_2}, ..., \frac{\partial L_i}{\partial w_d}]^T$$

If the perceptron did correct classification on certain instance, then the loss will be 0, as well as its partial derivative, otherwise:

$$\frac{\partial L_i}{\partial W} = -y_i X_i^T$$

Which it can be derived very easily. We use matrix calculus notation is because the parameter w in perceptron can be viewed as a column vector, it is usually called "weight" because it is the weight in the weighted sum of perceptron output, the update of weight matrix can simply be written as:

$$W \longleftarrow W - \alpha \frac{\partial L_i}{\partial W}$$

Then we derived the learning algorithm for the simplest type of neural network !

10.1.2 Basics components of Neural network

1. Activation function The activation function is the output function of a node, the choice of activation is crucial in the part of designing neural network, in early ages the activations are usually *sigmoid, tanh, sign*, let us consider the perceptron with a generic activation ψ:

$$\hat{y} = \psi(W \cdot X^T)$$

In perceptron, $\psi = sign$, lets take a look on other used activations in the early ages of neural network development:

$$\psi(x) = sign(x) \quad \textit{(sign function)}$$
$$\psi(x) = \frac{1}{1 + e^- x} \quad \textit{(sigmoid function)}$$
$$\psi(x) = \frac{e^{2x} - 1}{e^{2v} + 1} \quad \textit{(tanh function)}$$

To precisely describe a neuron, we usually divide the single output into two activation values, which is **pre-activation** and **post-activation**:

$$a_h = W \cdot X^T \quad \textit{(Pre-activation)}$$
$$h = \psi(a_h) \quad \textit{(Post-activation)}$$

Another special type of activation is called "softmax", where it is commonly used in the output node of a neural network

10.2 Convolutional Neural Network

11 References

[1] O. Stein, Basic Concepts of Nonlinear Optimization. Springer, 2021. doi: 10.1007/978-3-662-69741-2.