Machine Learning

1. Định nghĩa
   * Định nghĩa 1: **the field of study that gives computers the ability to learn without being explicitly programmed.**
   * Định nghĩa 2: **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.**
     + Ví dụ: chơi cờ checkers
       - E: kinh nghiệm chơi cờ checkers
       - T: nhiệm vụ chơi cờ checkers
       - P: xác suất máy tính thắng
2. Phân loại:
   * Thông thường thì mọi vấn đề được chia làm 2 loại:
     + Supervised Learning
     + Unsupervised Learning
3. Supervised Learning
   * Ở vấn đề này, chúng ta được biết trước **data set** và **correct output** với ý tưởng rằng có 1 mối quan hệ nào đó giữa **input** và **output**
   * Supervised Learning chia làm 2 loại:
     + Regression:
       - Dự đoán **continuous output**
       - Tức là gán cho đầu vào một vài **hàm số liên tục**
     + Classification:
       - Dự đoán **discrete output**
       - Tức là gán cho đầu vào những **giá trị rời rạc**
4. Unsupervised Learning
   * Ở vấn đề này, chúng ta **không biết được biết trước output** sẽ như thế nào
   * Mục đích của Unsupervised Learning là đưa ra các nhóm dữ liệu **theo 1 structure** (cấu trúc, kiểu nhóm) nào đó dựa trên **mối quan hệ giữa các thông tin** của dữ liệu
5. Model Representation
   * Kí hiệu thường dùng (Notation):
     + : input
     + : ouput
     + : số lượng **training example** trong **training set**
     + : training example (dòng thứ trong bảng training set, )
     + : tập xác định của
     + : tập xác định của
   * Quá trình xây dựng Model:  
     Diagram

     Description automatically generated
     + Quá trình này miêu tả Supervised Learning
     + Cho 1 Training Set để học 1 hàm số
     + : hàm số dự đoán đoán “đủ tốt” cho giá trị ( viết tắt của *hypthesis*)
6. Cost Function (aka Squared Error Function or Mean Squared Error):
   * Kí hiệu:
   * Dùng để đo độ chính xác của hàm
   * Xây dựng :  
     Chart, line chart

     Description automatically generated
     + Ý tưởng: chọn , sao cho gần với nhất có thể (dựa trên các training example )
     + Ta có:
     + Khoảng cách giữa “giá trị dự đoán” và “giá trị chênh lệch” là:
     + Tổng các khoảng cách đó: (tại sao lại có bình phương)
     + Trung bình của khoảng cách là:
     + Ta nhân thêm vào tổng trên để sau này khi đạo hàm xuống sẽ dễ dàng tính hơn, trong khi giá trị và cần tìm không thay đổi:
     + Đặt tên tổng trên là
   * Tóm lại ta cần đi tìm và sao cho hàm – Cost Function – có giá trị nhỏ nhất
7. Gradient Descent (Suy giảm độ dốc)
   * Là thuật toán để tìm giá trị nhỏ nhất của hàm số
   * Ý tưởng của thuật toán, chọn 1 điểm bất kí trên hàm số, tại đó, nhìn xung quanh, xem đâu là hướng dốc nhất, rồi đi theo hướng đó, lặp đi lặp lại hành động đó cho đến khi tới được điểm thấp nhất.
   * Hình ảnh minh hoạ:  
     Chart, surface chart

     Description automatically generated
   * Tuỳ vào vị trí điểm đầu (được chọn), thuật toán có thể cho các kết quả khác nhau (như hình)\
   * Thuật toán có thể dùng cho , nhưng đơn giản hoá thì thuật toán sau chỉ dùng cho 2 đầu vào và :
     + Lặp lại cho đến khi hội tụ: với
     + : **learning rate**, quyết định độ dài bước đi của thuật toán
       - Nếu như quá nhỏ thì thuật toán sẽ bị chậm
       - Nếu như quá to thì thuật toán có thể sẽ chạy qua điểm nhỏ nhất, không thể hội tụ và có thể phân kì
       - Với cố định, nhưng do độ dốc của hàm thay đổi nên bước đi của thuật toán sẽ ngày càng nhỏ, vậy nên không cần phải thay đổi trong quá trình thực hiện
     + và phải được cập nhật đồng thời  
       A picture containing text

       Description automatically generated
8. Áp dụng Gradient Descent vào Linear Regression
   * Đầu tiên, ta xử lý đạo hàm từng phần của hàm
     + Đạo hàm theo :
     + Đạo hàm theo :
   * Thay đạo hàm vào thuật toán Gradient Descent:
     + Lặp lại cho đến khi hội tụ:
9. Multiple features (variables)
   * Linear regression với nhiều biến (variable) được gọi là **Multivaritate Linear Regression**
   * Notation:
     + : đầu vào thứ trong các training example
     + : giá trị thứ trong training example thứ
     + : số lượng training example
     + : số lượng biến
   * **Hypothesis Function** từ trở thành
   * Viết lại hàm dưới dạng ma trận:
     + Ta thêm 1 biến
     + Gọi:
     + Từ đó, ta viết được lại là:
   * **Cost Function** từ thành  
      nhưng 2 ở đây khác nhau, 1 là của 2 đầu vào, 1 là của đầu vào
     + Thay vì viết , thì ta có thể viết thành (với là )
   * Áp dụng thuật toán **Gradient Descent**:
     + Lặp lại cho đến khi hội tụ: với
     + Ta có:
     + **Lưu ý**:
10. Feature Scaling
    * Mục đích: tăng tốc độ cho thuật toán Gradient Descent
    * Lý do cần: giả sử biến nằm trong khoảng 0-2000, còn nằm trong khoảng 1-5, thì đồ thị của hàm này có dạng như dưới đây, và rất mất thời gian chạy.  
      Diagram

      Description automatically generated
    * Ý tưởng: đưa các giá trị về nằm trong khoảng từ đến (hoặc gần với , như hay
    * Bằng cách:
      + : giá trị trung bình của
      + : range của ,
11. Cách chọn learning rate ()
    * Có 2 cách chọn:
      + Debugging Gradient Descent
      + Automatic Convergence Test
    * Automatic Convergence Test
      + Chọn 1 giá trị đủ nhỏ (như ), nếu giá trị giữa 2 lần chạy thuật toán, tao thấy giảm bằng , thì ta nói hội tụ
      + **Chọn giá trị không dễ**
    * Debugging Gradient Descent:
      + Vẽ đồ thị với *số lần chạy* thuật toán Gradiant Descent (trục ) và *giá trị* của (trục )
      + Nếu quá lớn, đồ thị tăng giảm
      + Nếu quá nhỏ, đồ thị giảm chậm, hội tụ chậm tăng
      + **Nếu đủ nhỏ, đồ thị giảm với mỗi lần chạy**
12. Features và Polynominal Regression
    * Mục đích: cải thiện hàm hypothesis
    * Bằng cách:
      + Kết hợp biến (từ 2 biến chiều dài và chiều sâu thành diện tích chiều dài *chiều rộng*)
      + Đổi hàm (sang hàm bậc 2, hay bậc 3, hay căn)
      + Ví dụ:
        - Ta có:
        - Tao thêm feature dựa trên để có được hàm hoặc
        - Từ đó ta đặt (và ) để có
        - Khi đó, ta phải để ý đến feature scaling của các biến
          * Giả sử: trong khoảng thì sẽ trong khoảng và sẽ trong khoảng từ
13. Normal Function
    * Là giải pháp thay thế cho thuật toán Gradient Descent trong việc tìm để đạt giá trị nhỏ nhất.
    * Tại sao lại dùng Normal Function thay cho Gradient Descend?
      + Không phải đi tìm
      + Không phải lặp đi lặp lại thuật toán nhiều lần như Gradient Desent
      + Nhanh hơn Gradient Descent nếu như (features) nhỏ hơn khoảng
      + **Tuy nhiên** cũng có lúc Gradient Descent sẽ nhanh hơn (khi )
        - Gradient Descent chạy với thời gian là
        - Normal Function tốn thời gian chỉ riêng cho việc tính
    * Ý tưởng của Normal Fucntion:
      + Từ việc đạo hàm, cho đạo hàm bằng 0, giải ra để tìm cực trị:
        - Tìm điểm cực trị của hàm thì ta giải , tìm ra , thì tại đó hàm số đạt nhỏ nhất hoặc lớn nhất trong vùng lân cận
      + Áp dụng điều đó, ta muốn:
        - Lấy đạo hàm từng phần rồi đặt :
        - Rồi giải hệ tìm ra tìm vecto (vecto chiều)
      + Ta có công thức (không đi chứng minh):
        - Với ma trận và vecto được đưa ra giống như ví dụ dưới đây:  
          Diagram, schematic

          Description automatically generated with medium confidence
      + Ta **không cần** để ý tới features scaling với Normal Function
    * Trong trường hợp không thể tính :
      + Lý do 1: thừa feature, tồn tại feature có thể bị lược bỏ
        - Ví dụ: , , thì ta thấy , tức là điều này có thể khiến không tính được (kiến thức Linear Algebra)
      + Lý do 2: quá ít training examples , giải quyết:
        - Xoá bớt feature
        - Sử dụng **Regurazation** (sau sẽ học)
14. Classification
    * Vì đầu ra (output) của bài toàn không mà là các giá trị rời rạc như nên ta không thể áp dụng thuật toán Linear Regression đã học ở trên để áp dụng vào những bài toàn như này.
    * Thay vì dùng Linear Regression, ta sẽ sử dụng Logistic Regression, thuật toán mà trả về
    * Với , thì được gọi là **Negative Class**, còn được gọi là **Positive Class**
15. Hypothesis Representation
    * Để có được hàm thì ta sử dụng là hàm **Sigmoid**, hay còn gọi là **Logisstic Function** vì:
      + Đồ thị của hàm :  
        Chart, line chart

        Description automatically generated
      + Khi thì
      + Khi thì
    * Ta xây dựng hàm mới
    * Lúc này hàm sẽ trả về cho ta **xác suất** xuất hiện của .
16. Decision Boundary
    * Để có được đầu ra 0 và 1, ta quy ước:
      + (dấu bằng ở hay không quá ảnh hưởng đến bài toán)
    * Để thì (theo đồ thị ), từ đó bài toán trở thành:
    * Lúc này với ta sẽ thu được 1 phương trình, và phương trình này ta gọi là **Decision Boundary**.
      + Tuỳ vào biến số của phương trình mà ta có thể thu được rất nhiều hình dạng.
17. Cost Function của Logistic Regression
    * Nếu ta áp dụng Cost Function của Linear Regression vào trong Logistic Regression thì đồ thị sẽ có dạng **non-convex** (nhiều cực trị 🡪 Gradient Descent có thể không tìm thấy điểm cực tiểu nhỏ nhất), nên ta muốn đồ thị ở dạng **convex** (1 cực trị) để áp dụng Gradient Descent 1 cách hiệu quả.  
      **🡪 Xây dựng 1 Cost Function khác**
    * Ta có Cost Function của Linear Regression như sau:
      + Đặt
    * Với Logistic Regression ta đặt:
      + Đồ thị:
        - A picture containing diagram

          Description automatically generated
        - Chart

          Description automatically generated with medium confidence
      + Từ đồ thị ta thấy:
        - trong cả 2 trường hợp 🡪 Giá trị dự đoán và Giá trị thật giống nhau, dự đoán đúng thì
        - 🡪 Giá trị thực là 0 nhưng lại dư đoán là 1, tức là dự đoán sai thì
        - Tương tự, ngược lại thì
    * Ta rút gọn (thay các giá trị vào sẽ thấy tương đương)
    * Chốt lại ta được Cost Funcion của Logistic Regression như sau:
      + Dạng Vector: với
18. Gradient Descent với Cost Function của Logistic Regression:
    * Ta có đạo hàm từng phần (các bước đạo hàm có thể tìm thấy trên GG)
    * Ta thấy kết quả này giống với của Linear Regression. Nên thuật toán Gradient Descent giống hệt với Linear Regression (khác )
    * Dạng Vecto:
19. Multiclass Classification: One-vs-All
    * Đầu ra output lúc này không phải nữa mà là
    * Ta chia vấn đề ra thành vấn đề nhỏ, mỗi vấn đề này chỉ có 2 đầu ra là “chính nó” với “những giá trị còn lại”. Bằng cách áp dụng **Binary Logistic Regression** với mỗi vấn đề này rồi chọn hàm với giá trị cao nhất để dự đoán.
      + …
    * Ví dụ minh hoạ:  
      Diagram

      Description automatically generated
20. Overfitting Problem
    * Là vấn đề mà hàm không phù hợp, bao quát, miêu tả tốt các training example và kết quả dự đoán đầu ra không hợp lý.
    * Hình ảnh minh hoạ:  
      Chart, line chart

      Description automatically generated
    * Có 2 loại Overfitting:
      + **Underfit**: hàm miêu tả dữ liệu kém
        - Nguyên nhân: quá đơn giản hoặc quá ít feature
      + **Overfit**: hàm miêu tả tốt các dữ liệu cho trước nhưng dự đoán dữ liệu mới kém
        - Nguyên nhân: hàm quá phức tạp nên tạo nhiều đường cong không cần thiết, không liên quan tới dữ liệu
    * Cách giải quyết:
      + Giảm số lượng feature:
        - Chọn 1 cách thủ công các feature
        - Dùng thuật toán **Model Selection**
      + Regularization:
        - Giữ nguyên số lượng feature, nhưng giảm độ lớn của các parameter
        - Hoạt động tốt khi chúng ta có những feature hữu dụng
21. Regularization với Linear Regression
    * Cost Function:
      + Bằng cách cộng thêm vào , ta sẽ có được đầu ra nhỏ hơn,   
        🡪 sẽ đơn giản hơn 🡪 Không bị Overfit nữa
      + Ta sẽ được Cost Function mới như sau:
      + được gọi là **Regularization Parameter**
      + Trong trường hợp:
        - : giá trị đầu ra không đổi, không khắc phục được Overfit
        - quá lớn: giá trị đầu ra quá nhỏ, dẫn tới Underfit
    * Gradient Descent:
      + Thuật toán trở thành:
        - Repeat:
        - Rút gọn
        - Nhận xét: ta luôn có với , nhỏ và , lớn 🡪 giảm giá trị của trước 1 ít rồi mới cập nhật.
    * Normal Function:
      + Phương trình tính ma trận trở thành:
        - với là ma trận
      + Bằng cách thêm vào phương trình thì ma trận luôn khả nghịch
22. Regularization với Logistic Regression
    * Cost Function:
    * Gradient Descent:
      + Repeat:
      + Giống với khi dùng Linear Regression
23. Non-Linear Hypothese
    * Non-Linear Classification: trong trường hợp chúng ta có quá nhiều feature mà ta lại thấy các giá trị phân bố không đều   
      🡪 Decision Boundary của chúng ta xấu   
      🡪 phải dùng đến các giá trị bình phương   
      🡪 số lượng feature lúc này trở lên cực lớn   
      🡪 tính toán lâu và không hợp lý
24. Neural Networks
    * Hình ảnh minh hoạ:
      + Text, letter

        Description automatically generated
    * Mô tả Model:
      + Định nghĩa:
        - Đầu vào (Input): **Dendrites** –
        - Đầu ra (Output): **Axons** –
        - : **bias unit**, luôn
        - Chúng ta dùng hàm giống trong Classification, gọi là Sigmoid (Logistic) **Activation** Function
        - : **parameter** hoặc **weight**
      + Biểu diễn đơn giản:
        - Mỗi 1 gọi là 1 layer
        - : **Input Layer**
        - : **Hidden Layer**, có thể có nhiều hơn 1
        - : **Output Layer**
        - hay : **Node**
        - : **Activation Unit** của **unit**  trong **layer**
        - : ma trận của **weight controling function** từ layer đến
      + Giá trị của mỗi Activation Node:
      + Chiều của ma trận sẽ bằng
        - : số lượng node của Layer
        - : số lượng node của Layer
        - xuất hiện vì ta lấy thêm bias unit ở Layer 🡪 ta chỉ lấy bias unit ở đầu vào, không lấy thêm bias unit ở đầu ra
    * Vectorized Implementation
      + Ta đặt:
        - Ví dụ (lấy từ trên):
      + Vecto , Vecto
      + Ta đặt
      + Giá trị cuối cùng:
25. Các bước giải quyết bài toán One-vs-All:
    * Ta có Input và Ouput (cho trước)
    * Ta đi tìm (tuỳ bài toán, nếu có dùng Neural Network thì là )
      + Bằng cách vẽ hàm đi qua các Output (càng gần càng tốt, nhưng không nên gần quá, dẫn tới Overfitting)
      + Để tìm được các tham số trong hàm , ta đi tối thiểu hàm để tìm
    * Sau khi đã có được hàm , với input và tham số tìm được. Ta đi tính các giá trị của tương ứng rồi đem ra so sánh với output . Nếu có xác suất giống cao thì tức là hàm này dùng để dự đoán các đầu vào (input) mới được.
26. Cost Function & Backpropagation
    * Cost Function
      + Một số định nghĩa cần biết:
        - : tổng số lượng Layer của Network
        - : số lượng node của Layer
        - : số lượng đầu ra (số lượng giá trị của output )
      + Với 🡪 🡪 bài toàn Classification:
        - Ta có Cost Function như sau:
      + Với 🡪 nhiều output 🡪 bài toàn Multiclass Classification:
        - Nếu như ta dùng Neural Network cho dạng toán này thì Cost Function sẽ như sau:
        - Trong đó:
          * Tổng và là tổng của các Logistic Regresion Cost với mỗi node ở output layer
          * Tổng , và là tổng của tất cả các trong cả Network
    * Backpropagation
      + Ta cần đi tính đạo hàm từng phần của :  để có thể tối tiểu hàm   
        🡪 Sử dụng thuật toán Backpropagation
      + Thuật toán Backpropagation:
        - Cho: Training set
        - Đặt: ma trận toàn 0
        - Với mỗi training set :
          * Đặt
          * Sử dụng Forward Propagation để tính với   
            A picture containing diagram

            Description automatically generated
          * Sử dụng để tính ( error value)
          * Tính  bằng   
            trong đó
          * Ta gán các giá trị vào ma trận   
            Vectorization:
          * Sau đó ta cập nhật ma trận :
        - Cuối cùng ta được
    * Để tiện tính toán ta có thể đổi từ ma trận sang vecto để truyền vào như là tham số cho các hàm tối thiểu hoá như trong Octave. Rồi sau khi tối thiểu thì ta sẽ đưa vecto kết quả về ma trận kích thước ban đầu.
      + Từ ma trận sang vecto (**Octave**):  
        thetaVector = [ Theta1(:); Theta2(:); Theta3(:); ]  
        deltaVector = [ D1(:); D2(:); D3(:) ]
      + Từ vecto sang ma trận (**Octave**):

Theta1 = reshape(thetaVector(1:110),10,11)  
Theta2 = reshape(thetaVector(111:220),10,11)  
Theta3 = reshape(thetaVector(221:231),1,11)

* + - Tổng kết lại các bước:  
      Graphical user interface, text, application

      Description automatically generated
  + Gradient Checking
    - Thuật toán này dùng để đảm bảo thuật toán Backpropagation tính toán theo kế hoạch.
    - Thuật toán này dự trên các tính xấp xỉ của đạo hàm:
    - Với ma trận , với mỗi ta có:  
      * Với là đủ nhỏ (có thể nhỏ hơn, hoặc lớn hơn) để đảm bảo thuật toán hoạt động.
    - Implement (Octave):  
      epsilon = 1e-4;  
      for i = 1:n,  
       thetaPlus = theta;  
       thetaPlus(i) += epsilon;  
       thetaMinus = theta;  
       thetaMinus(i) epsilon;  
       gradApprox(i) = (J(thetaPlus) - J(thetaMinus))/(2\*epsilon)  
      end;
    - Ta chạy kiểm tra gradApprox deltaVector, nếu đúng thì ta tắt thuật toán Gradient Checking đi để chạy Backpropagation (vì thuật toán Backpropagation tính toán đạo hàm của nhanh hơn Gradient Checking)
    - Không hoạt động với Dropout (chạy Gradient Checking, rồi mới Dropout)

1. Random Initialization (Khởi tạo ngẫu nhiên)
   * Lý do chúng ta cần khởi tạo ngẫu nhiên:
     + Thường thì nếu khởi tạo ma trận ma trận 0 thì thuật toán vẫn hoạt động bình thường nhưng điều đó không đúng nếu như chúng ta dùng Neural Network
     + Vì nếu như ma trận thì có thể dẫn tới việc tất cả các Node trong Hidden Layer đều giống nhau   
       🡪 khi dùng Backpropagation cũng giống nhau  
       🡪 đạo hàm của từng phần của bằng nhau  
       🡪 1 hidden layer có các Node giống hệt nhau (chúng ta không muốn điều này)
   * Khởi tạo ngẫu nhiên bằng cách:
     + Chúng ta gán cho mỗi giá trị 1 giá trị ngẫu nhiên trong khoảng   
       Text

       Description automatically generated
     + Lý do :
       - Rand(): khởi tạo giả trị trong khoản
       - Nên ta để được khoảng
2. Các bước chạy Neural Network
   * Bước 1: chọn kiến trúc (chọn số lượng hidden layer và số lượng node mỗi hidden layer)
     + Số lượng Input node: số lượng feature
     + Số lượng Output node: số lượng class
     + Số lượng node mỗi hidden layer: càng nhiều càng tốt nhưng phải cân bằng với thời gian tính toán (cost of computation)
     + Mặc định 1 hidden layer, nếu có nhiều hơn 1 hidden layer thì nên để số lượng node mối layer bằng nhau
   * Bước 2: training
     + Khởi tạo ngẫu nhiên
     + Implement Forward Propagation để tính với mỗi
     + Implement cost function
     + Implement Backpropagation để tính đạo hàm từng phần của
     + Dùng Gradient Checking để kiểm tra tính đúng đắn của Backpropagation. (nhớ tắt Gradient Checking sau khi sử dụng)
     + Sử dụng Gradient Descent hoặc các hàm tối thiểu được build in để tối thiểu hàm nhằm tính
   * Lưu ý:
     + Các implement forward and back propagation trong Octave:
       - for i = 1:m,  
          Perform forward propagation and backpropagation using example (x(i),y(i))  
          (Get activations a(l) and delta terms d(l) for l = 2,...,L
     + Chúng ta muốn vì điều đó đồng nghĩa với việc ta tối thiểu hoá thành công
     + là hàm non-convex nên chúng ta có thể sẽ tìm thấy điểm cực tiểu lân cận thay vì giá trị nhỏ nhất.