Weekly report of lessons

Name: Mayank Kumar
Roll No: 19CS30029
The week: 18-10-2021 to 24-10-2021

The topics covered:

Linear Discrimination: Discriminant Functions, Perceptron Classifier, Augmented Input and Linear Form, Linearly Separable Classes, Error function, Gradient descent method for iterative optimization, Other error functions, More stringent criteria of linear separability, Batch Relaxation Algorithm, Single sample relaxation with margin, Support Vector Machine: Two class problem, Optimization problem, Convex quadratic optimization problem, Solution, SVM Testing; Non-separable case, Slack variable to define soft margin, Constraints with soft margin hyperplanes, Projecting to higher dimensional space, Kernel machines, Vectorial kernel functions, Kernels, Parametric two class classification using discriminant, Logistic discrimination of two classes, Learning weights of logit functions, Algorithm (Learning weights)

ANN: Perceptron modelling a neuron, Artificial Neural Network, Feed-forward Network, Multilayered Feed-forward Network, Mathematical description, Input Output Relation, Optimization Problem, Chain rule of computing gradient of a single neuron, Computing gradient: Back propagation method, Back propagation: Concept, Back propagation: Delta Rule, ANN Training, Improving convergence

Summary topic wise:

Linear Discrimination

Discriminant Functions: Linear function: $g_i(x|w_i, w_{i0}) = \sum_{j=1}^{d} w_{ij}x_j + w_{i0}$

Quadratic Function: $g_i(x|W_i, w_i, w_{i0}) = x^T W_i x + w_i^T x + w_{i0}$
Perceptron Classifier: It is a linear classifier with a different perspective. $y = f(z) = sign(z)$
Augmented Input and Linear Form: W is computed by minimizing classification error for given $\{y_i, X_i\}$.
$z = \Sigma_i w_i x_i + w_0 \Rightarrow z = W^T X$ $\quad o = f(z) = sign(z)$

Linearly Separable Classes: If a solution to find a hyperplane separating data points of two classes exists, then the classes are called linearly separable.

Error function: Error function (Perceptron Criterion) $J(W) = \sum_{Y\ misclassified} - W^T Y$.
Gradient descent method for iterative optimization: Start with initial $W^{(0)}$ and update $W^{(i)}$ iteratively by computing $\nabla J(W^{(0)})$. $W^{(i)} = W^{(i-1)} - \eta(i)\nabla J(W)$, where ηis a positive scale factor (learning rate).

Other error functions: $J_q(W) = \sum_{Y\ misclassified} (W^T Y)^2$, $J_r(W) = \sum_{Y\ misclassified} \frac{(W^T Y - b)^2}{||Y||^2}$, $W^T Y \le b$
More stringent criteria of linear separability: The margin of separation between two linearly separable data points of classes is maximized using linear support vector machines.
Batch Relaxation Algorithm: W is initialized with $W^{(0)}$ and iterated till convergence. M is computed with margin b such that M = $\{Y|W^T Y <= b\}$, where M is the set of misclassified samples. Gradient $\nabla J_r(W) = \sum_{W^T Y \le b} \frac{Y(W^T Y - b)}{||Y||^2}$ and W is updated as $W^{(i)} = W^{(i-1)} - \eta \nabla J_r(W^{(i-1)})$

Single sample relaxation with margin: W is initialized with W(0) and is updated by considering samples one by one in every iteration. For sample Y_i at iteration k, $W^{(k)} = W^{(k-1)} + \eta(k)\frac{b - W^T Y_i}{||Y_i||^2} Y_i$ if $W^T Y_i <= b$, and the algorithm stops when change in update at the end of an iteration is very small.

Support Vector Machine (SVM): It is a linear discriminant classifier that uses Vapnik's principle and computes class boundaries without computing class distributions.
Two class problem: For X=$\{x^t, r^t\}$, t = 1 to N, x^t in R^d, r^t in {+1, -1}, w and w_0 are computed such that, $r^t(w^T x^t + w_0) \ge 1\ \forall\, t$

Optimization problem: Constrained Problem: To minimize $||w||^2/2$ subject to $r^t(w^T x^t + w_0) \ge 1\ \forall\, t$

Unconstrained Problem: $L_p = \frac{1}{2}||w||^2 - \sum_{t=1}^{N} \alpha^t [r^t(w^T x^t + w_0) - 1]$, where α^t is the Lagrange multipliers.

Convex quadratic optimization problem: Dual Problem: To be maximized wrt to α^t (>0) subject to zero gradients of L_p wrt w and w_0. From this, we get $w = \Sigma_t \alpha^t r^t x^t$ and $\Sigma_t \alpha^t r^t = 0$

Dual optimization problem: To maximize L_d wrt α^t, $L_d = -\frac{1}{2}\Sigma_t\Sigma_s \alpha^t\alpha^s r^t r^s (x^t)^T x^s + \Sigma_t \alpha^t$

Solution: On applying quadratic optimization technique, we get $w_0 = r^t - w^T x^t$

SVM Testing: Here, margin is not enforced and only support vectors decide class boundaries.

Non-separable case: If classes are not linearly separable, use slack variable, $\{s^t\}$, t = 1 to N

Slack variable to define soft margin: For soft margin hyperplane, to maximize margin, minimize $||w||^2$ subject to $r^t(w^T x^t + w_0) \geq 1 - s^t \,\forall t$

Constraints with soft margin hyperplanes: Slack variable is used to define constraints.

- $0 < s^t < 1$, x^t correctly classified; $s^t > 1$, x^t misclassified
- Soft Error = $\Sigma_t s^t$

Projecting to higher dimensional space: It may make them linearly separable.

$z = \varphi(x) \qquad g(z) = w^T z \qquad g(x) = \Sigma_j w_j \varphi_j(x)$, where $z_j = \varphi_j(x), j = 1 \text{ to } k$

Kernel machines: Discriminant function $g(x) = w^T\varphi(x) = \Sigma_t \alpha^t r^t K(x^t, x)$

The matrix of kernel values K is called Gram matrix, where $K_{t,s} = K(x^t, x^s)$

Vectorial kernel functions: Polynomials of degree q, Radial basis functions, Mahalanobis kernel functions, Distance function based, Sigmoid function

Kernels: It may be defined between a pair of objects flexibly without using any closed functional form.

Parametric two class classification using discriminant: Parameters Σ, μ_1, and μ_2 are estimated to compute coefficients of g(x): *w and w_0*.

Logistic discrimination of two classes: $y = P(C_1|x) = \frac{1}{1+e^{-(w^Tx+w_0)}}$

Learning weights of logit functions: For $y = P(r^t=1|x)$, minimize E using gradient descent to iterate on weights, where $E = -\Sigma_t(r^t \log y^t + (1-r^t)\log(1-y^t))$

Algorithm (Learning weights): Assume initial w and w_0 and compute y = sigmoid(w^Tx+w_0), compute gradients, and update w and w_0. Continue these computations till convergence.

ANN

Perceptron modelling a neuron: $Input \;->\; z = \Sigma_i w_i x_i + w_0,\; f(z) \;->\; output$, w_0 is bias

Artificial Neural Network: It is a network of perceptrons that describe input/output relations

Feed-forward Network: Signals travel in one direction only, and there is no feed back or loop

Multilayered Feed-forward Network: Every unit in a layer is connected with all units in the previous layer, and the ith layer takes input from (i-1)th layer and forwards its output to the input of (i+1)th layer.

Mathematical description: $ne_j^{(i)}$ -> jth neuron of ith layer, weights $W_j^{(i)} = (w_{j1}^{(i)}, w_{j2}^{(i)}, ..., w_{jn\_(i-1)}^{(i)})$, bias $w_{j0}^{(i)}$

n_(i-1): dimension of input to neuron, n_i: dimension of output at ith layer

Output of jth neuron in ith layer: $y_j^{(i)} = f(W_j^{(i)^T} X^{(i-1)} + w_{j0}^{(i)})$

Input Output Relation: In ith layer, $Y^{(i)} = f(W^{(i)}X^{(i-1)} + b^{(i)})$

Optimization Problem: Given X_i and O_i, i = 1 to N, to minimize $J_n(W) = \frac{1}{N}\sum_{i=1}^{N} ||O_i - F(X_i; W)||^2$; X_i is input that produces output O_i. Gradient descent procedure: Start with initial W_0 and update W iteratively.

$W_i = W_{i-1} + \eta(i)\,\Sigma_k(O_k - F(X_k; W_{i-1}))\,\nabla F(X_k; W_{i-1})$

Stochastic gradient procedure: $W_i = W_{i-1} + \eta(i)\,(O_k - F(X_k; W_{i-1}))\,\nabla F(X_k; W_{i-1})$

Chain rule of computing gradient of a single neuron: $\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial o}\frac{\partial o}{\partial z}\frac{\partial z}{\partial w_i} \qquad \nabla(W) = (\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n})$

Computing gradient: Back propagation method: Chain rule is applied from output to input and from output layer to input layer, and partial derivatives of weights at (i-1)th layer from ith layer are computed.

Back propagation: Concept: $\Delta w_{31}^{(2)} = \frac{\partial E}{\partial w_{31}^{(2)}} = \frac{\partial E}{\partial o}\left(\frac{\partial o}{\partial y_1^{(3)}}\frac{\partial y_1^{(3)}}{\partial y_1^{(2)}} + \frac{\partial o}{\partial y_2^{(3)}}\frac{\partial y_2^{(3)}}{\partial y_1^{(2)}}\right)\frac{\partial y_1^{(2)}}{\partial w_{31}^{(2)}} \qquad \frac{\partial y_1^{(2)}}{\partial w_{31}^{(2)}} = f'(z_1^{(2)})\,y_4^{(1)}$

Back propagation: Delta Rule: $\delta_1^{(2)} = \delta_{11}^{(3)}w_{11}^{(3)} + \delta_{12}^{(3)}\delta_{12}^{(3)}$

ANN Training: Initialize $W^{(0)}$ and for each training sample (x_i, o_i), functional values of each neuron is computed in forward pass, weights of each link starting from the output layer is updated using back propagation, and the process continues till convergence.

Improving convergence: To avoid abrupt change in gradient, we use momentum: $\Delta w_i = \alpha \Delta w_i^{(t-1)} - \eta \frac{\partial E^{(t)}}{\partial w_i}$

Adaptive learning rate is used according to change in error.

Concepts challenging to comprehend:

None

Interesting and exciting concepts:

I found Kernel Functions and Perceptrons somewhat interesting.

Concepts not understood:

None

Any novel idea of yours out of the lessons:

In the kernels, we work in the space of φ(x) and the dimensionality d of this new space may be much larger than the dataset size N, and as we use the N×N matrix XX^T, it might get computationally expensive. So, to solve this issue, we can use principal component analysis to reduce its linear dimensionality in the φ(x) space and compute the projected new k-dimensional values. This will optimize the algorithm and make it efficient.
