Machine Learning (CS60050) – Weekly Report

Kaushal Banthia (19CS10039)

Week 10: 20th – 22nd October, 2021

Topics Covered:

- Discriminant Functions
- Perceptron Classifier
- Support Vector Machine (SVM)
- Kernel Machines
- Parametric Discrimination
- Artificial Neural Network
- Computing gradient: Backpropagation method
- ANN training

Summary (Topic Wise):

- Discriminant Functions

  - $Choose\ C_i\ if\ g_i(x) = \max_j g_j(x)$
  - For a Linear function
    - $g_i(x|w_i, w_{i0}) = w_i^T x + w_{i0} = \Sigma_{j=1}^{d} w_{ij} x_j + w_{i0}$
    - This is a simple model, linear in form has $O(d)$ in storage and time of computing $g(.)$
  - For Quadratic function
    - $g_i(x|W_i, w_i, w_{i0}) = x^T W_i x + w_i^T x + w_{i0}$
    - A more complex model, $O(d^2)$ storage and time of computing $g(.)$
  - For two classes, one discriminant function is enough, which is $g(x) = w^T x + w_0$
  - If $g(x) > 0$ assign C_1 else C_2.
  - The hyper-plane dividing classes is $g(x) = 0$
  - For extending to more than 2 classes, we can consider Pairwise separation and only those samples of the class that lie in the positive half.

- Perceptron Classifier

  - A linear classifier with a different perspective.
  - $y = sign(\sum_i w_i x_i + w_0) = sign(W^T X), where\ X\ and\ W\ are\ column\ vectors.$
  - If we are able to find a hyperplane separating data points of two classes, the classes are called linearly separable.
  - We normalize data by making Y = X (if X is in class 1), and -X (if X is in class 2). For correct classification, $W^T Y > 0\ \forall\ Y$.
  - Thus $J(W) = \sum_{Y\ misclassified} -W^T Y$. We obtain W which minimizes J(W).
  - Gradient descent method for iterative optimization:
    - To obtain W which minimizes J(W)
    - Start with an initial vector $W^{(0)}$
    - Compute the gradient vector $\nabla J(W^{(0)})$
    - Move closer to minimum by updating W
    - $W^{(i)} = W^{(i-1)} - \eta(i)\nabla J(W),\ where\ \eta\ is\ the\ learning\ rate$

- There could be other forms of the error function as J(W) is not continuous.
  - $J_q(W) = \sum_{Y\ misclassified}(W^T Y)^2$
  - $J_r(W) = \frac{1}{2}\sum_{Y\ misclassified\ (W^T Y \le b)} \frac{(W^T Y - b)^2}{||Y||^2}$
  - Thus, $\nabla J_r(W) = \sum_{W^T Y \le b} \frac{Y(W^T Y - b)}{||Y||^2}$
- Batch Relaxation with Margin
  - Initialize W to $W^{(0)}$
  - Iterate till convergence
  - Compute the set M of misclassified samples (with margin b), so that
    $$M = \{Y | W^T Y <= b\}$$
  - Compute gradient.
  - Update W as $W^{(i)} = W^{(i-1)} - \eta(i)\nabla J_r(W^{(i-1)})$
- Single Sample Relaxation with Margin
  - Initialize W to $W(0)$
  - Perform the update of W by considering samples one by one in every iteration.
  - Consider an i[th] sample Y_i at k[th] iteration.
  - If $(W^T Y_i <= b)$ update W as $W^{(k)} = W^{(k-1)} - \frac{\eta(k)(b - W^T Y_i)}{||Y_i||^2} Y_i$
  - Stop when there is very little change in updates at the end of an iteration.

- <u>Support Vector Machine (SVM)</u>
  - A linear discriminant classifier that uses Vapnik's principle of never solving a more complex problem as a first step before the actual problem.
    - Classification: Sufficient to compute class boundaries (where $P(C_1|x) = P(C_2|x)$) without computing class distributions $P(C_i|x)$, etc.
    - Outlier detection: Compute boundaries separating those x having low $P(x)$.
  - After training the weight vector can be written in terms of training samples lying in class boundaries.
  - We need to maximize the margin between the class. Thus, we need to minimize ||w||. This calls for an optimization problem.
  - SVM – Testing:
    - Check only the sign of discriminant value (Margin not enforced).
    - Only support vectors decide class boundaries. Other samples do not influence the classifier.
  - There might be non-separable cases, like the soft margin hyperplane. In such cases, we make use of slack variable, $\{s^t\}, t = 1,2,..N$. Also, projecting to higher dimensional space may make them linearly separable.

- <u>Kernel Machines</u>
  - Discriminant function $g(x) = w^T\phi(x) = \sum_t \alpha^t r^t \phi(x^t)^T \phi(x) = \sum_t \alpha^t r^t K(x^t, x)$
  - No need to compute with basis functions and also performing dot products with z.
  - Gram matrix: The matrix of kernel values K, where $K_{t,s} = K(x^t, x^s)$
  - Should be symmetric and positive semidefinite.
  - Vectorial kernel functions

- Polynomials of degree q => $K(x^t, x) = (x^T x^t + 1)^q$
- Radial basis functions => $K(x^t, x) = e^{\left[-\frac{\|x-x^t\|^2}{2s^2}\right]}$
- Mahalanobis kernel function => $K(x^t, x) = e^{\left[-(x-x^t)^T S^{-1}\frac{(x-x^t)}{2}\right]}$
- Distance function based => $K(x^t, x) = e^{\left[-\frac{D(x,x^t)}{2s^2}\right]}$
- Sigmoidal function: $K(x^t, x) = tanh(2x^T x^t + 1)$
- Kernels may be defined between a pair of objects flexibly without using any closed functional form. The same principle is applicable for designing SVM for classifying such objects.

- <u>Parametric Discrimination</u>

  - Let $P(C_1|x) = y$, hence $P(C_2|x) = 1 - y$ be two classes. Choose C_1 if $logit(y)(log(y/1-y)) > 0, else\ choose\ C_2$
  - For two normal classes sharing a common covariance matrix, the log odds linear.
  - $logit\big(P(C_1|x)\big) = w^T x + w_0$
  - $w = \Sigma^{-1}\ (\mu_1 - \mu_2)$
  - $w_0 = -\frac{(\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)}{2} + log\frac{P(C_1)}{P(C_2)}$
  - The inverse of logit is the logistic function, also called sigmoid function.
  - $P(C_1|x) = logit - 1(wTx + w0) = 1/(1 + exp(-(wTx + w0)))$
  - Two class classification using discriminant
    - Estimate parameters $\Sigma, \mu_1, and\ \mu_2$
    - Compute coefficients of g(x): w, and w_0
    - During testing, calculate g(x) and assign $C_1\ if\ g(x) > 0, else\ C_2$ OR calculate $y = \frac{1}{1+e^{-(w^T x + w0)}}$ and assign $C_1\ if\ y > 0.5\ else\ C_2$
  - Logistic discrimination of two classes
    - Ratio of class densities modeled: $P(x|C_1)/P(x|C_2)$
    - Assume log likelihood ratio is linear (true for normal density functions).
    - $log\left(\frac{P(x|C_1)}{P(x|C_2)}\right) = w^{Tx} + w_0' \rightarrow logit\big(P(C_1|x)\big) = log\left(\frac{P(C_1|x)}{P(C_2|x)}\right) = log\left(\frac{P(x|C_1)}{P(x|C_2)}\right) + log\left(\frac{P(C_1)}{P(C_2)}\right) = w\hat{}Tx + w_0\left(when\ w_0 = w_0' + log\left(\frac{P(C_1)}{P(C_2)}\right)\right)$
    - $y = P(C_1|x)) = \frac{1}{1+e^{-(w^T x + w_0)}}$

  - Learning weights of logit functions.
    - Data: $X = \{x^t, r^t\}, t = 1,2,..N$
    - $r^t = 1\ for\ C_1, and\ 0\ for\ C_2$
    - Let $y = P(r^t = 1|x) \sim Bernoulli(y)$
    - Directly modeling likelihood of class assignment instead of likelihood of data given classes as in the parametric approach.

$$l(w, w_0|X) = \prod_{t=1}^{N} (y^t)^{r^t} (1 - y^t)^{(1-r^t)}$$

    - To minimize $E = -log(l)$ (Maximization of log likelihood)

$$E = -\sum_t (r^t \log(y^t) + (1 - r^t) \log(1 - y^t))$$

- Use gradient descent technique to iterate on weights.

➢ Gradient descent technique

- $y = sigmoid(a) \rightarrow \frac{dy}{da} = y.(1-y)$
- $\frac{\partial E}{\partial w_j} = -\sum_t (\frac{r^t}{y^t} - \frac{1-r^t}{1-y^t}) y^t(1-y^t) x_j^t = -\sum_t (r^t - y^t)\, x_j^t$
- $\frac{\partial E}{\partial w_0} = -\sum_t (r^t - y^t)$
- Update of weights at i[th] iteration as

$$w_j^{(i)} = w_j^{(i-1)} - \eta \frac{\partial E}{\partial w_j}$$

➢ Algorithm (Learning Weights)

1. Assume initial w, and w0.
2. Compute $y = sigmoid(w^T x + w_0)$
3. Compute gradients.
4. Update $w\ and\ w_0$
5. Continue steps 2 to 4 till convergence.

- <u>Artificial Neural Network</u>

➢ A network of perceptrons with a vector input and vector or scalar as output.
➢ It is a Feed-forward Network (no feedback or loop in the network).
➢ Multilayered feed-forward Network

- Layer-wise processing: i[th] layer takes input from (i-1)[th] layer and forwards its output to the input of next layer.

➢ Mathematical description of the model

- Let j[th] neuron of i[th] layer be $ne_j^{(i)}$
- Its corresponding weights are $W\_j\text{^}(i) = (w_{j1}^{(i)}, w_{j2}^{(i)}, \ldots., w_{jn\_(i-1)}^{(i)})$
- Bias: w_{j0}^i
- n_(i-1): Dimension of input to the neuron
- n_i: Dimension of output at i[th] layer
- Output of j[th] neuron in i[th] layer: $y_j^{(i)} = f\left(W_j^{(i)^T} X^{(i-1)} + W_{j0}^{(i)}\right)$.
- This output would serve as the input for the next layer.

➢ Optimization Problem

- Given $\{(X_i, O_i)\}, i = 1,2, \ldots, N$, find W such that it produces O_i given input X_i for all i.
- Minimize: $J_{n(W)} = \frac{1}{N}\sum_{i=1}^{N} \left|\left|O_i - F(X_i; W)\right|\right|^2$
- This can be done via stochastic gradient descent.

- <u>Computing gradient: Backpropagation method</u>

➢ For multi-layered feed forward network apply chain rule.

- From output to toward input.
- From output layer to toward input layer.
- Compute partial derivatives of weights at (i-1)[th] layer from the i[th] layer

- ANN training

  - Initialize $W(0)$
  - For each training sample (x_i, o_i) do
    - Compute functional values of each neuron in the forward pass.
    - Update weights of each link starting from the output layer using back propagation.
    - Continue till it converges.
  - Improving convergence by other methods:
    - Momentum: Gradients may change abruptly in consecutive iteration. To avoid we may use running average of weight updates to be added with the gradient. $\Delta w_i^{(t)} = \alpha \Delta w_i^{(t-1)} - \eta \frac{\partial E^{(t)}}{\partial w_i}$
    - Adaptive learning rate: We increase learning rate at constant steps if error decreases, else decrease it geometrically. $\Delta\eta = +a \; if \; E^{t+T} < E^t; else - b\eta$

Concepts Challenging to Comprehend: None yet.

Interesting and Exciting Concepts: Support Vector Machine (SVM) and Computing gradient: Backpropagation method

Concepts not understood: None yet.

A novel idea: Instead of applying momentum or Adaptive Learning Rate separately, we can apply them together. This would lead to an even better convergence. This would prevent oscillation (smoothen out the curve) and thus help in convergence.

We could work something on the lines of $\Delta w_i^{(t)} = \alpha \Delta w_i^{(t-1)} - \Delta\eta \frac{\partial E^{(t)}}{\partial w_i}$; where $\Delta\eta$ changes according to the gradient, thus helping converge better.