CS60050 Machine Learning - Weekly Report

Nisarg Upadhyaya (19CS30031)

Week 10: 20-22 October

1 Topics covered

- Discriminant Functions
- Perceptron Classifier
- Support Vector Machine
- Non-separable Case
- Kernel Machines
- Parametric Discrimination
- Logisitc Discrimination
- Artificial Neural Networks
- Back Propagation and ANN Training
- Improving Convergence

2 Summary

2.1 Discriminant Functions

For a set of discriminant functions in G, we assign class C_i if $g_i(x) = max_j g_j(x)$. We can have linear functions which take the form $g_i(x|w_i, w_{i0}) = w_i^T x + w_{i0}$ which is linear or the form $g_i(x|W_i, w_i, w_{i0}) = x^T W_i x + w_i^T x + w_{i0}$ which is quadratic. For two classes only a single discriminant function suffices. $g(x) = 0$ gives the hyper-plane dividing the two classes and we assign classes based on the sign taken by the function $g(x)$. For more than 2 classes we can do pairwise separation.

2.2 Perceptron Classifier

For weights vector W and input vector X it returns the outptut $y = sign(W^T X)$. Depending on the sign it is classified to one of the two classes. We want to find the optimum W which minimises the classsfication error. On one side of the hyperplane $W^T X = 0$ it takes positive values and on the other side negative. If a solution exists classes are called linearly separable. We normalize the data by making $Y = X$ (if X in class 1) and $Y = -X$ (if X in class 2). Thus for correct classification $W^T Y > 0 \forall Y$. One error function is $J(W) = \sum_{Y_{misclassified}} -W^T Y$ which we try to minimise. Using gradient descent we iteratively do $W^i = W^{i-1} - \eta(i) \nabla J(W)$ where η is the learning rate. There are other error functions which can also be used such as $J_q(W) = \sum_{Y_{misclassified}} (W^T Y)^2$ which is continuous unlike $J(W)$.

2.2.1 A more stringent criteria for separability

We consider misclassification when $W^T Y \leq b$ and the error function as $J_r(W) = \frac{1}{2} \sum_{Y_{misclassified}} \frac{(W^T Y - b)^2}{||Y||^2}$. In the batch relaxation with margin algorithm we do gradient descent till convergence and at each step while calculating $\nabla J_r(W)$ consider the complete set M of misclassified samples which have $W^T Y \leq b$. In the single sample relaxation algorithm we perform the update of W by considering samples one by one in every iteration.

2.3 Support Vector Machine

It is a linear discriminant classifier which follows $Vapnik's principle$. For classification it is sufficient to compute class boundaries. After training the weight vector can be written in terms of training samples lying in class boundaries. If r^t is the class (± 1) of x^t then we want $r^t(w^T x^t + w_0) \geq 1$ for all t. The 1 appears because of the margin between the classes. Thus we have a constrained optimisation problem which is to minimise $\frac{||w||^2}{2}$ subject to $r^t(w^T x^t + w_0) \geq 1$ for all t. This can be posed as lagrangian problem $L_p = \frac{||w||^2}{2} - \sum_{t=1}^{N} \alpha^t [r^t(w^T x^t + w_0) - 1]$ which needs to be minimized w.r.t w and w_0 and maximized w.r.t. lagrange multipliers. We get $w = \sum_t \alpha^t r^t x^t$ and $\sum_t \alpha^t r^t = 0$. A dual problem is obtained and solved using quadratic optimisation technique. Samples with positive α^t are support vectors.

2.3.1 Non-separable case

Sometimes classes may not be linearly separable. We make use of slack variables s^t for each sample. The new constraint becomes $r^t(w^T x^t + w_0) \geq 1 - s^t$. The soft error is $\sum_t s^t$. We minimise $\frac{||w||^2}{2} + C \sum_t s^t$ subject to $r^t(w^T x^t + w_0) \geq 1$ for all t where C is penalty factor. The same optimisation technique discussed before is used to solve this. Another approach can be to make them linearly separable by projecting to a higher dimensional space.

2.4 Kernel machines

We have the discriminant function as $g(x) = \sum_t \alpha^t r^t K(x^t, x)$, where K is the kernel function. Some vectorial kernel functions are polynomials of degree q such as $K(x^t, x) = (x^T x^t + 1)^q$. The radial basis function $K(x^t, x) = exp[\frac{-||x-x^t||^2}{2s^2}]$, the *Mahalanobis* kernel function and the sigmoidal function $K(x^t, x) = tanh(2x^T x^t + 1)$. Kernels may be defined between a pair of objects flexibly.

2.5 Parametric discrimination

For two classes let $P(C_1|x) = y$ and $P(C_2|x) = 1 - y$. We have $logit(y) = log(\frac{y}{1-y})$. Choose C_1 if $logit(y) > 0$. $logit(P(C_1|x)) = w^T x + w_0$, the inverse of this is the logistic function also called sigmoid function $P(C_1|X) = \frac{1}{1+exp(-(w^T x + w_0))}$.

2.6 Logisitc discrimination

The ratio of class densities is modelled as $\frac{P(x|C_1)}{P(x|C_2)}$. We assume that the log likelihood ratio is linear. This is true for normal density functions. Let $X = \{x^t, r^t\}$ be the data. $r^t = 1$ for C_1 and 0 for C_2. Assume $P(r^t = 1|x) \sim$Bernoulli(y). We need to minimise $E = -\sum_t (r^t log(y^t) + (1 - r^t) log(1 - y^t))$. We make use of gradient descent. $w_j^i = w_j^{i-1} - \eta \frac{\partial E}{\partial w_j}$. So we start by assuming initial w and w_0. Compute $y = sigmoid(w^T x + w_0)$ and the gradients. Update w and w_0 and repeat till convergence.

2.7 Artificial Neural Network

It is a network of perceptrons which takes as input a vector and gives a vector/scalar as output. There is no loop in the network. Layer wise processing takes place where each layer takes as input the output of the layer before it and forward it to the layer after it.

2.7.1 Mathematical description of the model

We have the j^{th} neuron of i^{th} layer as $ne_j^{(i)}$. Its corresponding weights are $W_j^{(i)}$ and bias $w_{j0}^{(i)}$. We have n_{i-1} as input dimension to this neuron and output dimension as n_i. The output of the neuron is as follows $y_j^{(i)} = f({W_j^{(i)}}^T X^{(i-1)} + w_{j0}^{(i)})$. This serves as input for the next layer. Let all the parameters of the different layers together be represented as W. Given $\{(X_i, O_i)\}, i = 1, 2, ..., N$ find W such that it minimises $J_n(W) = \frac{1}{N} \sum_{i=1}^N ||O_i - F(X_i; W)||^2$ where $F(X_i; W)$ is the output of the neural network on input X_i and parameters set to W. This can be solved using stochastic gradient descent using the same gradient descent procedure as discussed before.

2.7.2 Back Propagation

For multilayered feed forward network we apply chain rule and compute partial derivatives for $(i-1)^{th}$ layer using i^{th} layer. Thus we propagate derivatives from output layers towards input layers. Thus for training the ANN initialise $W^{(0)}$. Then for each training sample (x_i, o_i) do

1. Compute functional values of each neuron in the forward pass.
2. Update weights of each link starting from the output layer using back propagation.
3. Continue till it converges.

2.7.3 Improving convergence

1. Gradients may change abruptly in consecutive iteration. To avoid we may use running average of weight updates to be added with the gradient.
2. We increase learning rate at constant steps if error decreases, else decrease it geometrically.

3 Challenging concepts

The solution to dual optimisation problems and logistic discrimination

4 Interesting concepts

Artificial Neural Networks

5 Concepts not understood

None

6 Ideas

While training neural networks, first-time weights are assigned randomly. It may so happen that the network converges to a local minima with these weights. Just like having an adaptive learning rate it will also be good to have an adaptive weight initialisation technique at hand. It would be good to some some optimal initial weights which might be known through experience and try different random seeds and see which gives the best results.