1 Foreword

Author : Anderson Chau

Disclaimer : The notes are written only for my memorization purpose after I have studied online lecture notes and blogs.

2 Confusion Matrix

Predict True, Actual True : True Positive (TP)
Predict True, Actual False : False Positive (FP)
Predict False, Actual False : False Negative (FN)
Predict False, Actual True : True Negative (TN)

Accuracy = (TP+FN)/(TP+FP+FN+TN), performance of correct classification

Precision = TP / (TP+FP) (correctly classified as positive / Everything classified as positive), example usage : Cancer detection. (We don't want to initiate cancer treatment if the person is actually healthy).

Recall = TP / (TP + FN) (correctly classified as positive / Actually positive), FP is more expensive than TN . (e.g. Fraud detection).

Note : Mathematically, Precision and Recall are inverse relationship, there is a tradeoff between recall and precision.

F1 score = 2(P*R)/(P+R), a compromised metric

3 K-Nearest Neighbour

Description : Choose the *majority* class of nearest (e.g. Eclidean Distance) K data points and classify it.
How to Choose K(hyper-paramaeter) : General rule of thumb : sqrt(number of data)/2 or by searching and comparing different k's for highest prediction accuracy.
Normalization of data in preprocessing is a must

4 K-Means Clustering

Simple Description : Identify clusters by finding the centroid of data points

Algorithm :

1. Initialize μ_1,μ_2.... μ_k randomly (k is hyper-parameter)

2. Repeated until converge :

(i) $c^{(i)} := \arg\min_j ||x^{(i)} - \mu_j||^2, j \in [1 : k]$, (i.e. $c^{(i)}$ denote which μ the $x^{(i)}$ is linked to. Link each data point to nearest μ_j. If $x^{(i)}$ is nearest to μ_s, then $c^{(i)} = j$. Thus, k partitions are created.)

(ii) $\mu_j := \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\}x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}}$, (i.e. For each data point in each partition from (i) , find the new centroid and assign to μ_k

Proof of convergence of the algorithm : consider

$J(c, \mu) = \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$
Observation : J must be monotonically decreasing. It is because for step (i) It is adjusting $c^(i)$ to reduce J, for step (ii) we are adjusting μ_j to reduce J
J is non-convext, it may get to local minimum. To try several random initial values, and choose the lowest J.

5 Linear Regression(MSE approach)

Hypothesis :

$$h_\theta(x) = \sum_j \theta_j x_j = \theta^\top x$$

We want to minimize MSE (Mean Square Error)

$$J(\theta) = \frac{1}{2} \sum_i \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 = \frac{1}{2} \sum_i \left(\theta^\top x^{(i)} - y^{(i)} \right)^2$$

Gradient of J :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

Each θ_j is updated for each step by gradient descent algorithm.

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Practically :
(i) Learning rate α is hyperparameter and data dependent , larger, fewer steps to get to min. but may miss the minimum. (Monitor the loss curve, J value vs iteration).
(ii) Batch GD is slow, may be Mini-Batch GD or Stochastic GD.
(iii) If α is small but the loss oscillate , converged and stop learning.

6 Linear Regression(MLE approach)

7 Logistic Regression

$$P(y=1|x) = h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)} \equiv \sigma(\theta^\top x)$$
$$P(y=0|x) = 1 - P(y=1|x) = 1 - h_\theta(x)$$

Loss function is

$$J(\theta) = -\sum_i \left[y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) \right]$$

Again , BGD for following gradient of J :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x^{(i)} j \left(h\theta(x^{(i)}) - y^{(i)} \right)$$

Interpretation : For a particular sample : if h return 1/0 and y return 1/0 , the term is 0. if h return 1/0 and y return 0/1 , the term is positive infinity.

8 Logistic Regression(MLE approach)

9 Softmax Regression(Multi-Class Logistic)

k classes, n x k parameters , and the hypothesis is :

$$h_\theta(x) = \begin{bmatrix} P(y=1|x;\theta) \\ P(y=2|x;\theta) \\ \vdots \\ P(y=K|x;\theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^K \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \exp(\theta^{(2)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \tag{1}$$

Below Loss function is simple to understand : By referencing to previous hpothesis, we want to maximize the y=k associated probility.

$$J(\theta) = -\left[\sum_{i=1}^m \sum_{k=1}^K 1\{y^{(i)}=k\} \log \left(\frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)})} \right) \right]$$

Gradient of J is, we solve the problem by GD :

$$\nabla_{\theta^{(k)}} J(\theta) = -\sum_{i=1}^m \left[x^{(i)} \left(1\{y^{(i)}=k\} - P(y^{(i)}=k|x^{(i)};\theta) \right) \right]$$

Where :

$$P(y^{(i)}=k|x^{(i)};\theta) = \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{j=1}^K \exp(\theta^{(j)\top} x^{(i)})}$$

10 BGD variation : Mini BGD/SGD

BGD use all training data in a single step, which is exteremely costly.

11 Loss function in Classification(Binary) Problem - General treatment

General Hypothesis : $h_\theta(x) = x^T\theta$

Adjustment for binary classification :

$$\text{sign}(h_\theta(x)) = \text{sign}(\theta^T x) = \text{sign}(t) = \begin{cases} 1 & \text{if } t > 0 \\ 0 & \text{if } t = 0 \\ -1 & \text{if } t < 0 \end{cases}$$

Measure of confidence : $h_\theta(x) = x^T\theta$ gives larger value, more confident

Margin ($yx^T\theta$) : (i) if $h_\theta(x)$ classify correctly, margin is positive, otherwise negative.
(ii) Therefore our objective is to maximize the margin (we want both correct classification and be confident)

Consider the following loss function :

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \phi\left(y^{(i)}\theta^T x^{(i)}\right)$$

We want penalize wrong classfication and encourage correct one , we design ϕ as $\phi(z) \to 0$ as $z \to \infty$, while $\phi(z) \to \infty$ as $z \to -\infty$ where $z = yx^T\theta$, and examples are :

logistic loss : $\phi_{\text{logistic}}(z) = \log(1 + e^{-z})$,used in logistic regression

hinge loss : $\phi_{\text{hinge}}(z) = [1 - z]_+ = \max 1 - z, 0$, used in SVM

Exponential loss $\phi_{\text{exp}}(z) = e^{-z}$, used in boosting

12 Kernel Mapping (Special case demo by Linear Regression + Polymoninal Kernel)

(I) Purpose : To x map from lower higher dimension. Useful when data are non-linearly separable(Transform to a curve)
(II) Computation complexity does not necessarily increase proportionately.
(III) Example : a mapping function $\varphi : R \to R^4$, $x \to [1, x, x^2, x^3]$, and h is $\theta^T x$ having $\theta = [\theta_1, \theta_2, \theta_3, \theta_4]$

(IV) Terms : x is called attribute, $x \to [1, x, x^2, x^3]$ called feature, φ feature map, $\varphi : R^1 \to R^4$ in this case. d =1 p = 4
(IV) Another Example : a mapping function $\varphi : R^3 \to R^{1000}$, $x \to [1, x_1, x_1^2, x_1^3, x_1x_2, x_1x_2^2....]$ (*) ,let d = 3 , p =1000. If we exhaust all possibilities, then p = 1 + d + d^2 + d^3 (**)
Recall GD stepping :

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - h_\theta(x^{(i)}))x^{(i)}$$

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T x^{(i)})x^{(i)}$$

Putting kernel mapping to the equation :

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)}))\phi(x^{(i)})$$

We pause here to evaluate the cost of computing each of update (Curse of Demensionality...), considering (**). If we just use the kernel direction, we suffer the curse of demensionality : Suppose d (data dimension) = 1000, then by using the mapping in (**) we have p = 10^9. $\theta^T \phi(x^{(i)})$ need O(p) (dot product) , and O(np) for summing up all data in each step.
Going back to BGD.

$$\theta := \theta + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)}))\phi(x^{(i)})$$

, assuming $\theta = \sum_{i=1}^{n} \beta_i \phi(x^{(i)})$ (*) at some point, with initialization $\theta = 0 = \beta$
It becomes

$$\theta := \sum_{i=1}^{n} \beta_i \phi(x^{(i)}) + \alpha \sum_{i=1}^{n} (y^{(i)} - \theta^T \phi(x^{(i)}))\phi(x^{(i)})$$

Rearranging :

$$\theta := \sum_{i=1}^{n} (\beta_i + \alpha(y^{(i)} - \theta^T \phi(x^{(i)})))\phi(x^{(i)})$$

Therefore it is equivalent to updating β_i (instead of θ_i) by

$$\beta_i := \beta_i + \alpha(y^{(i)} - \theta^T \phi(x^{(i)}))$$

by (*) above

$$\beta_i := \beta_i + \alpha \left(y^{(i)} - \sum_{j=1}^{n} \beta_j \phi(x^{(j)})^T \phi(x^{(i)}) \right)$$

Computing of LHS is fast because : (1) we can pre-compute $\phi(x^{(j)})^T\phi(x^{(i)})$ for all i,j, and (2) $\phi(x^{(j)})^T\phi(x^{(i)})$ can be represented by $< x^{(i)}, x^{(j)} >$:
$\langle\phi(x), \phi(z)\rangle = 1+\sum_{i=1}^{d} x_i z_i+\sum_{i,j\in\{1,\ldots,d\}} x_i x_j z_i z_j+\sum_{i,j,k\in\{1,\ldots,d\}} x_i x_j x_k z_i z_j z_k =$
$1 + \langle x, z\rangle + \langle x, z\rangle^2 + \langle x, z\rangle^3$ (**)

Define K where K is n x n (n is the number of training samples) matrix, with $K(x, z) = \langle\phi(x), \phi(z)\rangle$, where K_ij is $\langle\phi(x^{(i)}), \phi(x^{(j)})\rangle$

Therefore, the process is : (1) compute K_ij using (**) , for all $i, j \in \{1, \ldots, n\}$. Set $\beta :=$ 0 ,
(2) Loop

$$\forall i \in \{1, \ldots, n\}, \quad \beta_i := \beta_i + \alpha\left(y^{(i)} - \sum_{j=1}^{n} \beta_j K(x^{(i)}, x^{(j)})\right)$$

in vectorized notation:

$$\beta := \beta + \alpha(\tilde{y} - K\beta)$$

When doing inference :

$$\theta^T\phi(x) = \sum_{i=1}^{n} \beta_i\phi(x^{(i)})^T\phi(x) = \sum_{i=1}^{n} \beta_i K(x^{(i)}, x)$$

In practice, we do computation using K (at O(d) cost) instead of directly from $\phi(x)$ is much faster. Further, We only need to know K but "just only need to know" the existence of $\phi(x)$. There is no need to be able to write down $\phi(x)$.
Consider the Kernel applied to bitmap : number of bits as d. (Great reduction!)
Intuitively, K represents similarity matrix, i.e. K is small if $\phi(x^{(j)})^T\phi(x^{(i)})$ is small
Example : Gaussian Kernel, it can support inifinitely dimensional space of mapping.

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right)$$

Mercer Theorem : For K to be a valid Kernel iff K is PSD.
Application : To SVM, perceptron, linear regression, and other learning algorithms represented only in inner product $\langle x, z\rangle$, then Apply K(x,z)

13 Generative vs Discriminative Learning Algorithm

14 Entropy

Definition :

$$H(x) = -\sum_{i=1}^{n} p(x_i)\log_2 p(x_i)$$

where $x_1, x_2...$ are all possible events of random variable(distribution) and $p(x_1), p(x_2)...$ are the probabilities of the respective events.

Connection with uncertainty (High Entropy , High Uncertainty) :

Entropy measure the uncertainity of a distribution. Consider a random variable distribution : X=1 at 0.33 , X = 2 at 0.33, X=3 at 0.33, and another : X = 1 at 0.98 , X = 2 at 0.01, X =3 at 0.01, we say the former distribution has higher uncertainity (more difficult to guess its value).

Connection with amount of information in a *message* (not distribution) :

Average number of bits (yes/no answers) NEEDED TO PROVIDE to tell x in a message. Therefore High Entropy. Higher Uncertainty , Higher Amount of Information.

Connection with Decision Tree splitting :

Remember that DT is greedy algorithm : It is to find the split that have greatest reduction in uncertainty (Information Gain) of the distribution (after splitting). We have a certain distribution

15 SVM, Support Vector Machine

Assumptions for illustration : data in binary classes only, linearly separable (if not, then apply Kernel)
Main Idea in Training : We construct a separating hyperplane, the plane has largest distance to all data point.
How ?
Let $y \in \{-1, 1\}$
Define classifier $h_{w,b}(x) = g(w^T x + b)$, where $w^T x + b$ is the formula of hyperplane, w is the normal vector to hyperplane.
where g : $g(z) = 1$ if $z \geq 0, g(z) = -1$ if $z < 0$
where w = $[\theta_1 ... \theta_n]^T$
Define functional margin (FM): $\hat{\gamma}^{(i)} = y^{(i)}(w^T x + b)$.
If FM is positive, it classify correctly. Negative, classify incorrectly.
If the magnitude is large , classifier is confident.
Define Geometric Margin (GM), $\gamma_i = \frac{\hat{\gamma^i}}{\|w\|}$
Further define smallest distance from hyperplane to data points : $\gamma = \min_{i=1,...,m} \gamma^{(i)}$
Thus, our training objective is to maximize this smallest distance.

$$\max_{\gamma, \mathbf{w}, b} \quad \gamma$$

$$\text{s.t.} \quad y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq \gamma, \quad i = 1, \ldots, m$$

$$\|\mathbf{w}\| = 1$$

The last condition ensure FM =>GM
Why use GM instead of FM in training ? We can always scale w and b to achieve greater magnitude in FM, therefore FM is meaningless for training.
Rearranging :

$$\max_{\gamma,\mathbf{w},b} \quad \hat{\gamma}/ \quad \|\mathbf{w}\|$$

$$\text{s.t.} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \ldots, m$$

In order to make it a convex optimization problem (another subject to study ! F!)
(1) we restrict the value of $\hat{\gamma} = 1$, by scaling w and b (can do single w/b for all $\hat{\gamma}$?)
(2) and rewriting $\frac{\hat{\gamma}}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|}$, we get w in from nominator to denominator

$$\min_{\gamma,\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

$$\text{s.t.} \quad y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad i = 1, \ldots, m$$

Apply cvx opt. library to solve the above problem