1 Testing

2 Foreword

Author : Anderson Chau

Disclaimer : The notes are written only for my understanding and memorization purpose after I have self-studied those online lecture notes.

3 Likelihood and Maximum Likelihood Estimation(MLE)

We have m sampling data, we attempt to establish a hypothesis $h(\theta)$ with parameter θ to model the data
The term Likelihood denotes the probabilty that particular value θ represents the sampling data.
We want to find the value of parameter(s) which best represent the data(*), the process is called MLE.

Let f be the pdf of random variable X, X_i is the value if sample i , then $f(X_i \mid \theta)$ is read as the chance of X_i happening if value of parmeter is θ)

Likelihood function : $L(\theta) = \prod_{i=1}^{m} f(X_i \mid \theta)$ (assumption here : sampling are independent process). We want to find θ that maximize L , i.e. (*)
Generally, we take log (monotonically increasing function) likelihood to convert multiplications to additions for easier handling and then find maxima (by partial derivatives = 0)

4 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 in inverse relationship, there is a tradeoff between recall and precision.

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

5 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

6 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.

7 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.

8 Linear Regression(MLE approach)

9 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.

10 Logistic Regression(MLE approach)

11 Softmax Regression(Multi-Class Logistic)

Softmax is used because it is differentiable 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 quite easy to understand : By referencing to previous hpothesis, we want to maximize the y=k associated probility if that data belongs to class k

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

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 :

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

12 Regularization in Linear Regression

13 BGD variation : Mini BGD/SGD

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

14 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

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

(I) Purpose : To map x 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 $K_i j$ is $\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$

Therefore, the process is : (1) compute $K_i j$ 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)

16 Generative vs Discriminative Learning Algorithm for classification, discussion

D : Learn the curve that separates the classes, e.g. Logistic Regression, SVM , ANN, CNN
G : Learn (all the parameters of) the model itself and just class of data. e.g. Naive Bayes, Gradient Discriminant Analysis, GAN
An Analogy of G : Learn both English and French , and guess whether the word Bonjour is French or English.
Another Example of G (*):
Let's say a model is trained with 1000 pictures :
(i) Dog without glasses : 1
(ii) Dog with glasses : 239
(iii) Human without glasses : 500
(iv) Human with glasses : 260
Assume we have a photo with a glasses. To classify a dog or human in the picture for a generative model : Since P (H & G) / P (G)= 260/261 > P(D & G) / P(G)= 1/261, the model infer that it is a human.
let x be feature, y be class
Put it in another way, in D (e.g. (multi-class) logistic regression) , we learn h which is p(y|x) and infer the class with largest p(y|x). i.e. we are finding $argmax_y p(y|x)$

In G, we are learning p(x|y) (by learning all p(x) for each possible classes of y) and p(y) (pdf of all classes of y). Let y be the class (Dog=1 vs Human=0) , x be feature(with glasses) , we learn p(x|y=1) (case (i)/(ii)) and p(x|y=0) (case (iii)/(iv))
Mathematically, D and G's relationship : $argmax_y p(y|x)[D] = argmax_y \frac{p(x|y)p(y)}{p(x)} = argmax_y p(x|y)p(y)[G] = argmax_y p(x\&y)$
(bayes rule, x is independent variable, bayes rule again, also see (*))

17 Naive Bayes Classifier

An example of Generative Learning algorithm Example usage , spam mail detection
Let x_i's be the all words in dictionary. y = 1 for spam mail, y = 0 for non-spam mail.
In training, we want to learn the parmaters : ϕ_y (p of spam mail) , $\phi_{j(y=1)}$ (p of j^{th} word appearing in spam mail), and $\phi_{j(y=0)}$
We have the following joint likelihood function

$$L(\phi_y, \phi_{j(y=0)}, \phi_{j(y=1)}) = \prod_{i=1}^{n} p(x^{(i)}, y^{(i)})$$

$$\phi_{j(y=1)} = \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y(i) = 1\}$$

$$\phi_{j(y=0)} = \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y(i) = 0\}$$

$$\phi_y = \sum_{i=1}^{n} 1\{y(i) = 1\}$$

Above is just simple counting

For Inference, how ?
A (non-)spam email having x's words has the probability :

$$p(x's|y) = p(x_1....x_{5000}|y) = p(x_1|y)p(x_2|y,x_1)p(x_3|y,x_2,x_1)....p(x_{5000}|y,x_2,x_1,....,x_{4999})$$

$$= p(x_1|y)p(x_2|y)p(x_3|y)....p(x_{5000}|y) = \prod_{i=1}^{n} p(x_i \mid y_i)$$

First by bayes rule(can be proved by induction), then by naives bayes assumption. e.g. $p(x_{2087}|y) = p(x_(2087)|y, x_{39831})$

Finally compare $p(y=1|x's)$ and $p(y=0|x's)$ to determine whether it is a spam mail or not:
By bayes rule,

$$p(y=1|x's) = p(x's|y=1)p(y=1)/p(x's)$$

$$= \frac{\prod_{j=1}^{d} p(x_j \mid y=1)p(y=1)}{\prod_{j=1}^{d} p(x_j \mid y=1)p(y=1) + \prod_{j=1}^{d} p(x_j \mid y=0)p(y=0)}$$

$$= \frac{\prod_{j=1}^{d} \phi_{j(y=1)}\phi_y}{\prod_{j=1}^{d} \phi_{j(y=1)}\phi_y + \prod_{j=1}^{d} \phi_{j(y=0)}(1-\phi_y)}$$

Practically : (1) Remove common words in preprocessing. e.g. the , of (stop words) (2) Instead of labeling all words in dictionary, we build only from trained data.

Laplace Smoothing – Handling unseen word
Problem : both classes give zero in p(x—y)
To solve : Treat that new word to have appaeared in all classes once, good thing is that it won't change the relative p :

$$P(j \mid y=1) = \frac{1 + \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\}}{2 + \sum_{i=1}^{n} 1\{y^{(i)} = 1\}}$$

$$P(j \mid y=0) = \frac{1 + \sum_{i=1}^{n} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\}}{2 + \sum_{i=1}^{n} 1\{y^{(i)} = 0\}}$$

18 Bernoulli event model

19 Gradient Discriminant Analysis

20 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

21 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 gives highly confident result.
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

22 Bagging and Random Forest

Review of Decision Tree : Greedy algorithm , Split on latest Information gain, Entropy/Gini Coefficient
Bagging = Bootstrapping + Aggregration
Boostrapping = Resample with replacement, to generate different sample set of "same population"
Aggregration = perform averaging / voting with different Trees from different boostraping samples
Aim to reduce variance
Ramdom Forest : Bagging + randomly remove features in build indivudal trees

23 Adaboost

Main Idea : Boosting transform weak learner to strong classifier, by increasing the weight of wrongly classified samples to force the classifier to do well on those samples. It is kind of ensembling (by attaching different weights to different classifiers).

Let $\phi_\tau(x^{(i)}$ be a weak learner (e.g. Decision Stump).

$$h_\theta(x) = \text{sign}\left(\sum_{j=1}^{n} \theta_j \phi_j(x)\right)$$

is the hypothesis of boosting.
Loss function (also check the general loss function discussion above) :

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

By Coordinate descent (choose a coordinate in θ and compute

$$\theta_j = \arg\min_{\theta_j} J(\theta)$$

Specifically, the boosting algorithm performs coordinate descent on the exponential loss for classification problems. The objective:

Coordinate descent algorithm:

1. Choose a coordinate $j \in \{1, \ldots, N\}$ 2. Update θ_j: $\theta_j = \arg\min_{\theta_j} J(\theta)$ Leave θ_k unchanged for all $k \neq j$ Iterate until convergence

Derivation of the coordinate update for coordinate j:

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

The objective function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \exp\left(-y^{(i)} \sum_{j=1}^{N} \theta_j \phi_j(x^{(i)}) \right)$$

Property of exp:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} w_i \exp\left(-y^{(i)} \theta_j \phi_j(x^{(i)}) \right)$$

Define $w_i = \exp\left(-y^{(i)} \sum_{k \neq j} \theta_k \phi_k(x^{(i)}) \right)$

To optimize coordinate θ_j (***):

$$\theta_j = \arg\min_{\theta_j} \sum_{i=1}^{m} w_i \exp\left(-y^{(i)} \theta_j \phi_j(x^{(i)}) \right)$$

Define the weights (*) :

$$w_i = \exp\left(-y^{(i)} \sum_{k \neq j} \theta_k \phi_k(x^{(i)}) \right)$$

Important Note : by definition of (*), updating θ_j corresponds to updating w, therefore we say Adaboost is updating weights of samples

By definition of (***) , it has the meaning of assigning weight w_i to sample x_i, and we are finding θ_j to do the best for classification (i.e. minimize the loss)

Optimizing coordinate θ_j corresponds to minimizing:

$$\sum_{i=1}^{m} w_i \exp\left(-y^{(i)} \theta_j \phi_j(x^{(i)}) \right)$$

Define:

$$w_i^+ := \sum_{i:y^{(i)}\phi_j(x^{(i)})=1} w_i$$

$$w_i^- := \sum_{i:y^{(i)}\phi_j(x^{(i)})=-1} w_i$$

****

Following is the whole algorithm :
****

For each iteration $t = 1, 2, \ldots$:

(i) Define weights (**)

$$w^{(i)} = \exp\left(-y^{(i)} \sum_{\tau=1}^{t-1} \theta_\tau \phi_\tau(x^{(i)})\right)$$

and distribution, which is the weight (which is uniform initially) attached to each sample, we are tuning this :

$$p^{(i)} = \frac{w^{(i)}}{\sum_{j=1}^m w^{(j)}}$$

(ii) Construct a weak hypothesis $\phi_t : R^n \to \{-1, 1\}$ from the distribution

$$p = \left(p^{(1)}, \ldots, p^{(m)}\right)$$

on the training set.

(iii) Compute

$$W_t^+ = \sum_{i:y^{(i)}\phi_t(x^{(i)})=1} w^{(i)}$$

and

$$W_t^- = \sum_{i:y^{(i)}\phi_t(x^{(i)})=-1} w^{(i)}$$

and set

$$\theta_t = \frac{1}{2} \log \frac{W_t^+}{W_t^-}.$$

Final note here : when Proof of Boosting convergence and Discussion of weak learners : Omitted

24 Gradient Boosting - Regression

1. Fit F(x) to model data
2. Fit another $h_1(x)$ to fit the residue in 1. : $(x_1, y_1 - F(x_1), (x_2, y_2 - F(x_2), (x_3, y_3 - F(x_3)....)$. We got F+$h_1$
3. Goes on to get $F + h_1 + h_2 + h_3...$

4. If we consider the loss function as $L(y, F(x)) = (y - F(x))^2/2$, then we get
$\frac{\partial J}{\partial F(x_i)} = \frac{\partial}{\partial F(x_i)} \sum_i L(y_i, F(x_i)) = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$
We are actually doing Gradient descent!
5. Note here we are treating the model $F(x_i)$ as random variable, i.e. we are updating F instead of x_i. And data dimension = number of samples
6. MSE may be susceptible by outliners. Adopt absolute loss L(y,F)=$|y - F|$ or Huber loss:

$$L_\delta = \begin{cases} \frac{1}{2}(y - \hat{y})^2 & if\ |(y - \hat{y})| < \delta \\ \delta((y - \hat{y}) - \frac{1}{2}\delta) & otherwise \end{cases}$$

7. Choose proper learning reate (Omitted)

25 Gradient Boosting - Classification

As an Illustration , English character recognition : A-Z :
1. Generate feature vectors x with k dimension, n samples , e.g. 1^{st} = length of longest straight line, 2^{nd} = number of pixels ....
2. Define 26 models $F_A, F_B...F_Z$, and corresponding $P_A(x)....P_Z(x)$ by softmax , where

$$P_A(x) = \frac{e^{F_A(x)}}{\sum_{c=A}^{Z} e^{F_c(x)}}$$

3. Define 26 "true" distributions (*) $Y_c(x_i)$, where x_i is i^{th} sammple. If y_5 is G , then $Y_G(x_5) = 1$ and $Y_{c=/=G}(x_5) = 0$. Notes : there are n x 26 values for it
4. Here, we have n x 26 variables to optimize : $F_A(x_1), F_A(x_2)...., F_A(x_{n-1}), F_Z(x_n)$.
5. Use KL divergence as loss function, by comparing currently adjusted F and (*).

26 Gradient Boosting - Ranking

27 Regularized Linear Regression

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

1. To reduce overfitting : we add lambda which aims to minimize the magnitude θ. The larger Lambda (Hyperparameter)gives smaller θ.
2. Note here lambda is also divided by training size (m). When m increase, the regularization effect is reduced. One explanation is : the purpose of R is to reduce overfitting, we should put more emphais on the added data iteslef.

28 Bias Variance Analysis

Let $\hat{\theta}_n$ be estimators(or say sampling data parameters),θ^* be true population parameters. The following shows that MSE comprise of Bias and Variance.

$$\begin{aligned}
\text{MSE}(\hat{\theta}_n) &= \mathbb{E}\left[\|\hat{\theta}_n - \theta^*\|^2\right] \\
&= \mathbb{E}\left[\|\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n] + \mathbb{E}[\hat{\theta}_n] - \theta^*\|^2\right] \\
&= \mathbb{E}\left[\|\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n]\|^2 + \|\mathbb{E}[\hat{\theta}_n] - \theta^*\|^2 + 2(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^T(\mathbb{E}[\hat{\theta}_n] - \theta^*)\right] \\
&= \mathbb{E}\left[\|\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n]\|^2\right] + \|\mathbb{E}[\hat{\theta}_n] - \theta^*\|^2 \\
&= \text{tr}\left(\mathbb{E}\left[(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])(\hat{\theta}_n - \mathbb{E}[\hat{\theta}_n])^T\right]\right) + \|\mathbb{E}[\hat{\theta}_n] - \theta^*\|^2 \\
&= \text{tr}\left(\text{Var}(\hat{\theta}_n)\right) + \|\text{Bias}(\hat{\theta}_n)\|^2.
\end{aligned}$$

BV tradeoff : decrease B will increase V. Increase model complexity.
Intuition to the tradeoff. If the model is too simple, we are missing some features to model the data. When we increase the model complexity, the model capture the noise in training data which may not appear in validation data. Therefore the model fail to generalize (overfitting).
We have to find a sweetspot which balance B and V.

29 PCA, Principal Component Analysis

We thnk that N dimensoinal data have certain "directions" - called principal axis (PA). We project those data points with reference to these axis/planes.
Generally, (i) we could intepret the meaning of the axis by comparing (data in differente extreme of the axis), or (ii) by mapping data to the axis - dimensionality reduction.
(Stanford cs168 notes' explanation is awesome ...).
0. x_i's are mapped to v_k's (eigenvector), Hence, N th dmension is mapped to kth dimension , where generally N is greater than k
1. The data keep the most variance (Retain the variability of data : Minimize information loss, minimize reconstruction error, remember the triangle) if it is projected to PA(remember animation online). Or say, MAXIMIZE the projection length

$$\sum_{i=1}^{n}\sum_{j=1}^{k}\langle x_i, v_j\rangle^2$$

where v is the PA and x is the data point
Equivalently:

$$\arg\max_{v:\|v\|=1}\frac{1}{n}\sum_{i=1}^{n}\langle x_i, v\rangle^2$$

Since :

$$v^\top X^\top X v = (Xv)^\top (Xv) = \sum_{i=1}^{n}\langle x_i, v\rangle^2$$

Also, by Linear Algebra

$$(Xv)^\top (Xv) = vX^\top Xv$$

, $X^\top X$ is called corvariance matrix and is Positive Semi Definite and can be written in $QDQ^\top$, where D is diagonal Matrix and Q is orthogonal Matrix ($Q^\top Q = I$),
Let λ_1 be the lartest value in D
To maximize $v_1^\top A v_1$, since

$$\begin{aligned} v_1^\top A v_1 &= v_1^\top Q D Q^\top v_1 \\ &= e_1^\top Q^\top Q D Q^\top Q e_1 \\ &= e_1^\top D e_1 \\ &= \lambda_1 \end{aligned}$$

For above shows that, the largest value of $v_1^\top A v_1$ is λ_1(the largest eigenvalue) . And when $v_1 = Qe_1$, $v_1^\top A v_1$ attain largest value.

To compute the projection (Power Iteration / SVD)

Power Iteration Given matrix A = $X^\top X$: • Select random unit vector u0 • For $i = 1, 2, \ldots$, set $u_i = A^i u_0$. If

$$\frac{u_i}{\|u_i\|} \approx \frac{u_{i-1}}{\|u_{i-1}\|},$$

then return

$$\frac{u_i}{\|u_i\|}.$$

i.e. keep multiplying A to u until converge. (Analysis omitted, SVD method omitted)

30 SVD, Singular Value Decomposition

Definition : Any Matrix A (m x n) can be factorized as USV^T, where
U is m x m Orthogonal Matrix , Column of U called left singular vectors
S is m x n Diagonal Matrix, entries of S called Orthogonal Matrix
V is n x n Orthogonal Matrix. Column of V (Row of V^T) called right singular vectors of A

Rank of Matrix : Minimum number of linear independent row or column in a Matrix
Rank Factorization: Let a matrix has Rank k , then A can be factorized as USV^T
where U is m x k and V^T is k x n
Low Rank Approximation : remove small K , therefore A is approximated to $A_k = U_k S_k V^T$
Used in compression, fine tune LLM, De-noising, Matrix completion.
Relationship between SVD and PCA (TODO CS168 notes)

Facts to Remember $A = USV^T$
(i) U are columns of (left) singular vectors, which are orthogonal (unit and perpendicular) to each other
(ii) V^T are rows of (right) singular vectors , which are orthogonal
(iii) Singular vectors : Similar to principal axis in PCA, which meaning may (not) be interpretable
(iv) S is diagonal, with decreasing constants
(v) Singular vectors may(not) be interpretable.

31 Example SVD Application : Recommender System

Input Data :
Rows of A : audience 1,2....
Columns of A : Movie 1,2,3....
Element of A : Score given by auidence to each movie ....
After SVD and low-rank Approximation :
Row of U , singular vectors of audience N, which represent the taste of auidence
Columns of V^T , singular vectors of Movie N, which represent the some qualities of the movie.
By using vector search (e.g. cosine), we can find the similarity of taste of audience, and recommend movie to him. (e.g. by filling missing values)

32 SVD Application : Word Embedding, Establishing word similarity as vectors

Input Data :
A_{ij} : number of occurrance word i and word j in the same document
After SVD and low-rank Approximation, each row of U represent the vector representation of each word
Then we can do vector search(Cosine) for word similarity .e.g. MongoDB

33 EM Algorithm

34 Reinforcement learning - Discrete State

Markov decision processes (MDP)
Definitions :
S : set of states.
A : set of actions
P_{sa} : denote the set of probability of action a on state s to change to another state.
γ between [0-1) as discount factor, to punish (decrease value function when time pass)

35 Reinforcement learning - Continuous State

36 MAP (Maximum a Posterior) vs MLE (Maximum Likelihood Estimation)

37 IDP, Indepent Component Analysis

38 Hidden Markov Model

39 Apriori

40 Recommender System

41 Anomaly Detection

42 Perceptron

43 KL Divergence

$$KL(P||Q) = \sum_x P(x) \log(\frac{P(x)}{Q(x)})$$

1. Measure the difference betwwen 2 distributions : You think it is Q, but actually it is P
2. The above is for discrete distribution
3. KLD is assymetric
4. In ML , measure the Information loss if Q is used instead of P. In other words, It measures the information loss (Entropy increased) if Q is used to approximate the true distribution (P).

44 Cross Entropy

$$H(p, q) = -\sum_x p(x) \log q(x).$$

1. It measures the average number of bits required to identify an event from one probability distribution, p, using the optimal code for another probability distribution, q.
2. The large the difference in bit numbers : we consider it as much difference.
2. As loss function for classification , e.g. logistic regression , CNN classifier.

45 Cross Validation