Machine Learning

Supervised learning	Unsupervised learning
(i) Algorithms are trained using labeled data.	(i) unlabeled data
(ii) More accuracy.	(ii) Less accuracy
(iii) Number of classes is known.	(iii) not known.
(iv) Uses offline analysis	(iv) Uses real time analysis of data.

Classification	Regression
(i) Output variable must be a discrete value.	(i) Output variable must be of continous nature or real value.
(ii) Classification tries to find the decision boundary which can divide the dataset into different classes.	(ii) Regression tries to find the best fit line, which can predict the output.
(iii) Types: Binary classifier, Multi-class classifier	Types: Linear regression, Non-linear regression.
Classification Algorithms:	**Regression Algorithms:**
* Logistic regression (not a regression)	• Simple linear regression
• K-nearest neighbors (KNN)	• Multiple linear regression
* Support Vector Machine (SVM)	• Polynomial regression
• Naive Bayes classification	• Support vector regression
* Decision tree classification	• Decision tree regression
* Random forest classification	• Random forest regression.

Overfitting & Underfitting: [Prerequisites: Variance, Bias]

* **Bias:** refers to how correct the model is.
  - Simple model with lot of mistakes ⟹ High bias.
  - Complecated model that does well on training data ⇒ low Bias.

* **Variance:** describes how much a prediction could potentially vary if one of the predictors changes slightly.
  - Simple model changes prediction slowly with predictor value ⇒ low variance high Bias.
  - Complecated model that changes prediction widely, even predictor value changes slightly : High variance Low Bias

Overfitting:

- When a model ~~is trained with so much data,~~ too complex relative to amount of noise, it starts learning from noise and results in **high variance**.
  - → Overfitted model doesnot categorise the data correctly.
  - → " " have high variance and low bias.
  - → Model is too complex and unrealistic, low generalization capacity.

Techniques to reduce overfitting:

1) Increase training data
2) Reduce model complexity
3) Regularization: Ridge & Lasso
4) Use of dropout for neural networks.
5) Using cross validation.

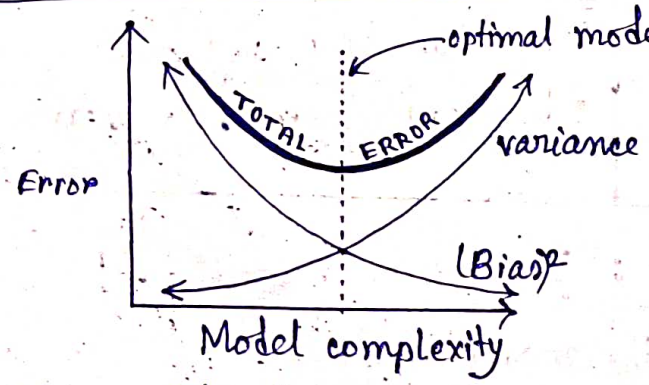
Underfitting:

If a statistical model can not capture underlying trend of the training data due to less data ~~and~~ but more features to build the model, hence have low generalization capacity, **high bias, low variance**

Techniques to reduce ~~overfitting~~ underfitting:

1) Increase model complexity
2) Remove noise from data
3) Increase number of features performing feature engineering.
4) Increase number of epochs (number of passes of training dataset)
5) Increase duration of training.

* Bias-variance tradeoff:



Total error of a model is composed of three terms: $(bias)^2$, variance, an irreducible error term.

- We modify the cost function to restrict the weights, which allows to trade exessive variance, potentially reducing overall error.

This trade comes in the form of regularization.

Linear Regression

A linear model that makes ~~by~~ prediction by simply computing a weighted sum of the input features + a constant bias term.

$$\hat{y} = w_0 + w_1 x_1 + w_2 x_2 + \cdots w_n x_n$$

$\hat{y}$ = the predicted value, x_i = i^{th} feature value

w_0 = bias term $w_1, w_2, \cdots w_n$ = feature weights.

- Simple linear regression : $\hat{y} = w_0 + w_1$
  prediction is made by only one predictor variable 'x'.
- Multiple linear regression : more than one predictor variable is present. $(x_1, x_2, \cdots x_n)$

Vectorized form : $\hat{y} = h_w(x) = w^T \cdot x$

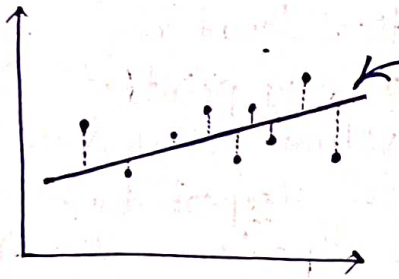
* w, x are column vectors or column matrices.

w = model's parameter vector

x = instance's feature vector.

$w^T \cdot x = [w_0 x_0 + w_1 x_1 + \cdots + w_n x_n]$, $x_0 = 1$,

w_0 = bias term.



The regression line.

(The unique line such that the sum of the squared vertical distances between the data points and the line is the smallest possible.)

→ Our goal is to find the line, i.e, find the weights w_1 to w_n and bias term w_0.

Average Loss function : $$MSE(X, h_w) = \frac{1}{m} \sum_{i=1}^{m} \left(w^T x^{(i)} - y^{(i)} \right)^2$$

MSE = mean square error.

The squared vertical distance

→ Target is to minimize the loss function.

- for simple linear regression, $\hat{y} = w_0 + w_1$,

we have the closed form solution : (taking partial derivatives of loss function w.r.t w_0 & w_1)

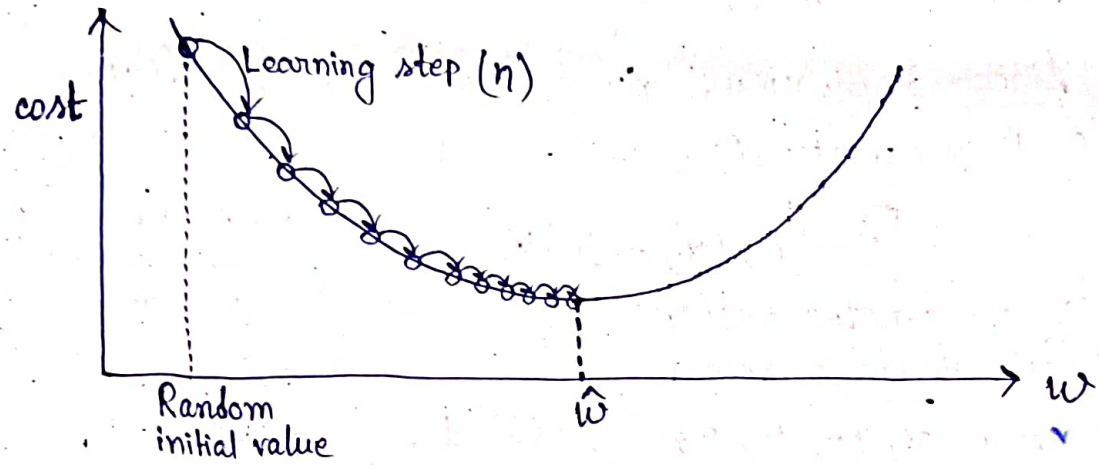
$$w_1 = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}, \quad w_0 = \frac{\sum y - w_1 \sum x}{n}$$

LMS (least minimum slope) Algorithm (delta rule)

LMS algo tunes w_0 & w_1 to produce the model that minimizes the avg loss function.

→ We start with a initial guess of w_0, w_1.
→ then update the values of w_0 & w_1
→ We use gradient descent method to update the values.

Gradient descent



The loss function is a convex function (quadratic) that has a single minima. We start at any point of this function space and update the values of w_0 & w_1 as: we take a step in the direction of steepest decrease of Loss function, i.e, towards the -ve slope (or gradient).

The learning step size (η) is proportional to the slope of the cost function so that the steps gradually get smaller as the parameters approach the minimum.

Batch gradient descent:

$MSE(X, h_w)$ is represented as $MSE(w)$

(i) Gradient vector of the cost function:

$$\nabla_w (MSE(w)) = \begin{bmatrix} \frac{\delta}{\delta w_0} MSE(w) \\ \frac{\delta}{\delta w_1} MSE(w) \\ \vdots \\ \frac{\delta}{\delta w_n} MSE(w) \end{bmatrix} = \frac{2}{m} X^T (Xw - y)$$

* y, w are column matrices.
* X = matrix of all features.
* η = learning step.

(ii) Gradient descent step: $\boxed{w^{(next\ step)} = w - \eta \nabla_w MSE(w)}$

** The calculation involves the full training dataset at each gradient descent step. So, it is terribly slow on very large training sets.

Calculation of partial derivative for a single feature :

$$\frac{\delta\, MSE(W)}{\delta\, w_n} = \frac{\delta}{\delta w_n}\left(\frac{1}{m}\sum_{i=1}^{m}\left(W^T x^{(i)} - y^{(i)}\right)^2\right)$$

$$= \frac{\delta}{\delta w_n}\left(\sum_{i=1}^{m}\left(W^T x^{(i)} - y^{(i)}\right)\right) \cdot \left\{\frac{2}{m}\sum_{i=1}^{m}\left(W^T x^{(i)} - y^{(i)}\right)\right\}$$

$$= x_n^{(i)} \cdot \left\{\frac{2}{m}\sum_{i=1}^{m}\left(W^T x^{(i)} - y^{(i)}\right)\right\}$$

$$= \boxed{\frac{2}{m}\sum_{i=1}^{m}\left(W^T x^{(i)} - y^{(i)}\right) \cdot x_n^{(i)}}$$

$$\boxed{w_n^{(\text{next step})} = w_n - \eta \cdot \frac{\delta\, MSE(W)}{\delta\, w_n}}$$

Notations :

$x^{(i)}$ = vector of all feature values of i^{th} instance.

X = a matrix containing all the feature values of all instances.

$x_n^{(i)}$ = n^{th} feature value of i^{th} instance.

Example:

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}_{m\times 1}, \quad X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_{p-1}^{(1)} \\ x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_{p-1}^{(2)} \\ \vdots & \vdots & \vdots & & \vdots \\ x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \cdots & x_{p-1}^{(m)} \end{bmatrix}_{m\times p}, \quad \hat{W} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_{p-1} \end{bmatrix}_{p\times}$$

$$\nabla_w (MSE(W)) = \frac{2}{m} \cdot X^T (XW - y)$$

$$\star\ X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

★ The Normal equation (Closed form soln.) :

$$\boxed{\hat{W} = (X^T \cdot X)^{-1}\, X^T \cdot y}$$

$\hat{W}$ = the value of W that minimizes cost function.

y = vector of target values.

X = matrix containing all feature values of all instances.

Stochastic Gradient Descent :

Stochastic Gradient descent picks a random instance in the training set at every step and computes the gradients based on that single instance, that makes the algorithm ~~makes~~ work ~~much~~ much faster and possible to train on huge training sets.

Instead of gently decreasing the cost function until it reaches the minima, the cost function will bounce up and down; decreasing only on average. Over time, it will end up very close to the minimum.

After reaching very close to minima, it still continues to bounce around and never settles down.

When the cost function is very irregular, this can help the algo to jump out of local minima and find the global minima.

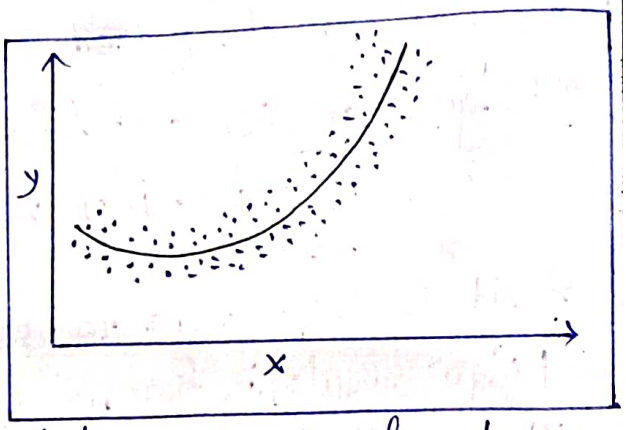
Mini batch gradient descent :

At each step, instead of computing the gradients based on full training set, or based on just one instance, it computes the gradients on small random sets of instances called mini-batches.

* All the three gradient descents end up near the minimum, but batch gradient descent's path actually stops at the minimum, while both Stochastic and mini-batch gradient descent continue to walk around the minima.

Polynomial Regression:

A straight line will never fit this data property.



Adds higher degrees to the feature:

* If there are more than one feature (n features), and degree 'n', total features (including higher order terms and combinations of them) = $(n+d)!/(n!\cdot d!)$

for one feature and degree = 2, $\hat{y} = w_0 + w_1 x + w_2 x^2$

for two features (x_1 & x_2) and degree = 2,

$$\hat{y} = w_0 + w_1 x_1^2 + w_2 x_1 + w_3 x_2^2 + w_4 x_2 + w_5 x_1 x_2$$

* Not only the higher order of the features are added, but the combinations are also added.

$(2+2)!/(2!\cdot 2!) = 6$ (including power 0)

Regularization:

(1) L1 regularization or Lasso regularization:

→ adds a penalty to the loss function.

Lasso regularization cost function:

$$J(w) = MSE(w) + \alpha \sum_{i=1}^{n} |w_i|$$

$\sum_{i=1}^{n} |w_i|$ = sum of absolute values of weights,

α = regularization parameter.

* Lasso regularization tends to eliminate the weights of the least important features. So, it automatically performs the feature selection and outputs a sparse model.

* To avoid gradient descent from bouncing around the optimum at the end while using Lasso, the learning step is needed to be reduced gradually so that the steps get smaller and smaller near the optimum and so it will converge.

(2) L2 regularization or Ridge regularization :

→ Adds sum of squared values of weights as penalty.

(i) Ridge regression cost function :

$$J(w) = MSE(w) + \alpha \frac{1}{2} \sum_{i=1}^{n} w_i^2$$

it tries to keep the model weights as small as possible.

* It is important to scale the data before performing Ridge regularization, as it is sensitive to the scale of the input features.

Ridge regression closed form solution :

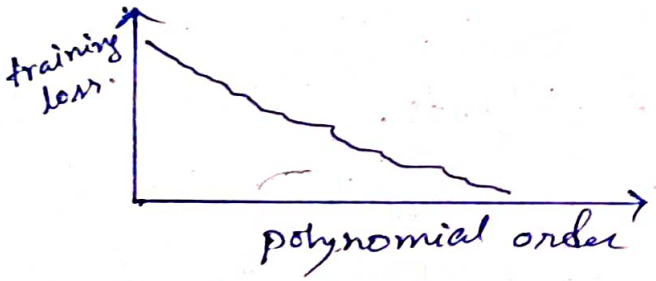
$$\hat{w} = (X^T X + \alpha I)^{-1} X^T y$$

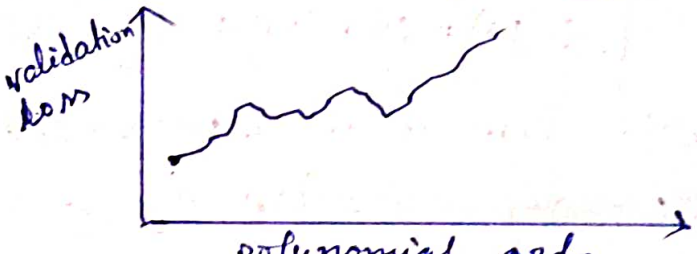
, I = identity matrix except with a '0' at top left, corresponding to bias-term.

* Elastic Net : it is a middle ground between ridge and Lasso regression.

$$J(w) = MSE(w) + r\alpha \sum_{i=1}^{n} |w_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} w_i^2$$

(ii) Validation : Use of a second dataset to compute loss. Validation loss increases ~~monotonically~~ with the order of polynomial, i.e, model complexity.





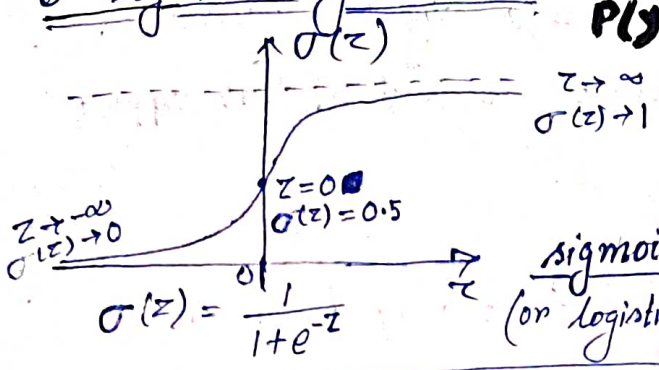
(iii) Cross validation : Every data pt gets to be in validation set exactly once and gets to be in training set (K-1) times.

→ This significantly reduces bias and variance as we are using most of the data for fitting and in validation set.

→ K = 5 to 10 is generally preferred.

Logistic Regression

$P(y|x) = h(x)^y \cdot (1-h(x))^{1-y}$



$\sigma(z) = \frac{1}{1+e^{-z}}$ — sigmoid func. (or logistic func)

$$\sigma'(z) = \frac{1}{(1+e^{-z})^2} \cdot e^{-z} = \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right)$$

$$\boxed{h_w(x) = \sigma(W^T x) = \frac{1}{1+e^{-w^T x}}} \qquad \boxed{\sigma'(z) = \sigma(z).(1-\sigma(z))}$$

★ Probability refers to the chance that a particular outcome occurs based on the values of parameters in a model.

★ Likelihood refers to how well a sample provides support for particular values of a parameter in a model.

- Logistic regression model estimates the probability $\boxed{\hat{p} = h_w(x)}$ that an instance x belongs to the +ve class

- Cost function for a single training instance:

$$c(w) = \begin{cases} -\log(h_w(x)) & \text{if } y = 1 \\ -\log(1-h_w(x)) & \text{if } y = 0 \end{cases}$$

[There are two classes: $y=1$ and $y=0$]

- Cost function over the whole training set is the avg cost over all training instances.

- Logistic regression cost function (log loss)

$$J(w) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)} \log(\hat{p}^{(i)}) + (1-y^{(i)}) \log(1-\hat{p}^{(i)})\right]$$

There is no known closed form solution to compute 'w'.
The cost function is convex. so, gradient descent method finds global minimum.

$$\frac{\partial}{\partial w_n} J(w) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma(w^T x^{(i)}) - y^{(i)}\right) x_n^{(i)}$$

Softmax regression: (Multinomial logistic regression)

This is generalization of logistic regression, that support multiple classes.

→ for a given instance (x):

- Softmax regression model first computes a score $s_k(x)$ for each class k.
- Then estimates the probability of each class by applying the softmax function (or normalized exponential) to the scores.
- Softmax score for class k:

$$\boxed{s_k(x) = x^T w^{(k)}}$$

→ Each class has its own dedicated parameter vector $w^{(k)}$.

→ All these vectors are stored as rows in a parameter matrix W:

$$W = \begin{bmatrix} (w^{(1)})^T \\ (w^{(2)})^T \\ \vdots \\ (w^{(n)})^T \end{bmatrix}$$

- The probability that the instance x belongs to class k: (Softmax function)

$$\boxed{\hat{p}_k = \sigma(s(x))_k = \frac{e^{s_k(x)}}{\sum_{j=1}^{K} e^{s_j(x)}}}$$

- k = number of classes
- $s(x)$ = vector containing the scores of each class for instance x
- $\sigma(s(x))_k$ = estimated probability that x belongs to class k, given score of each class for x.

→ Softmax regression predicts the class with the highest estimated probability.

- Softmax regression classifier prediction:

$$\hat{y} = \underset{k}{\operatorname{argmax}}\ \sigma(s(x))_k = \underset{k}{\operatorname{argmax}}\ s_k(x) = \underset{k}{\operatorname{argmax}}\ \left((w^{(k)})^T x\right)$$

- 'argmax' operator returns the value of a variable that maximizes a func.

Cross entropy cost function:

It penalizes the model when it estimates a low probability for a target class. It is used to measure how well a set of estimated class probalities matches the target classes.

$$J(W) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(p_k^{(i)}\right)$$

- $y_k^{(i)}$ is the target probability that the i^{th} instance belongs to class k. In general, it is either 0 or 1.
- for $k=2$, the cost function is equivalent to logistic regression's cost func.

Cross entropy gradient vector for class k:

$$\nabla_{w^{(k)}} J(W) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^{(i)} - y_k^{(i)}\right) x^{(i)}$$

Now, the gradient vector for every class can be computed & then use gradient descent to find the parameter matrix 'W', that minimizes the cost function.

Decision Tree

Gini impurity: $G_i = 1 - \sum_{k=1}^{n} P_{i,k}^2$

$P_{i,k}$ = the ratio of class k instances among the training instances in the i^{th} node.

① CART cost function for classification:

CART = classification and regression tree.

$$J(k, t_k) = \frac{m_{left}}{m} G_{left} + \frac{m_{right}}{m} \cdot G_{right}$$

Training set is split into two subsets using a single feature 'k' and threshold 't_k', m = number of instances.

CART algo searches for pair (k, t_k) that produces purest subsets.

Once the CART algo successfully searches for the sets, it splits the subsets using same logic recursively untill it reaches the maximum depth, or it cannot find a split that will reduce impurity.

② Entropy: A sets entropy is 0 when it contains instances of only one class.

Entropy of i^{th} node: $H_i = -\sum_{k=1}^{n} P_{i,k} \cdot \log_2(P_{i,k})$, $P_{i,k} \neq 0$

③ Information Gain(feature) = Entropy(dataset) − Entropy(feature)

④ Gini impurity Vs Entropy:

most of the times, they lead to similar trees.

Gini impurity tends to isolate the most frequent class in its own branch of the tree.

Entropy tends to produce slightly balanced tree.

⑤ CART cost function for regression:

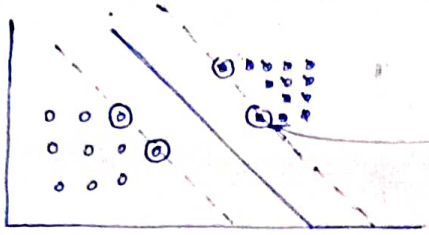
$$J(k, t_k) = \frac{m_{left}}{m} MSE_{left} + \frac{m_{right}}{m} MSE_{right}$$

$$MSE_{node} = \sum_{i \in node} (\hat{y}_{node} - y^{(i)})^2$$

$$\hat{y}_{node} = \frac{1}{m_{node}} \cdot \sum_{i \in node} y^{(i)}$$

Support Vector Machine

→ capable of linear, non-linear classification, regression, and outlier detection.



support vectors : instances located on the edge of the street.

● Hard and soft margine classification :

In hard margine classification, all instances must be off the street to one side for a class. It is sensitive to outliers.

Soft margine classification finds a good balance between keeping the street as large as possible and limiting the margine violations.

● Kernel trick : The kernel trick makes possible to get same result. even if many polynomial features are added with very high degree polynomials. So, there is no combinatorial explosion of the number of features.

● Similarity features :

Similarity function measures how much each instance resembles a particular landmark.

Gaussian Radial Bias function :

$$\phi_\gamma(x, l) = \exp\left(-\gamma \|x - l\|^2\right)$$

l = landmark
x = value of instance.

decision func: $W^T x + b = w_1 x_1 + \cdots w_n x_n + b$

prediction $\hat{y} = 0$ if $W^T x + b < 0$

1 if $W^T x + b \geq 0$

⑩ Hard margine SVM classifier objective:

$$\underset{w,b}{\text{minimize}} \quad \frac{1}{2} w^T w$$

subject to $t^{(i)}(w^T x^{(i)} + b) \geq 1$ for $1, 2, \cdots m$.

⑩ Soft margine SVM classifier objective:

$$\underset{w,b,\zeta}{\text{minimize}} \quad \frac{1}{2} w^T w + C \sum_{i=1}^{m} \zeta^{(i)}$$

subject to $t^{(i)}(w^T x^{(i)} + b) \geq 1 - \zeta^{(i)}$

ζ = slack variable ≥ 0

$\zeta^{(i)}$ measures how much the i^{th} instance is allowed to violate the margine.

⑪ Kernelized SVM

→ Second degree polynomial mapping:

$$\phi(x) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}\, x_1 x_2 \\ x_2^2 \end{pmatrix}$$

for two 2D vectors a, b.

⑪ Kernel trick for 2nd degree polynomial mapping:

$$\phi(a)^T \phi(b) = (a^T b)^2$$

$\boxed{K(a,b) = (a^T b)^2}$ is 2nd degree polynomial kernel.

★ Kernel is a function capable of computing the dot product $\phi(a)^T \phi(b)$ based only on the original vectors a, b without computing the transformation ϕ.

Common kernels:

Linear: $K(a,b) = a^T b$

polynomial: $K(a,b) = (\gamma a^T b + r)^d$

Gaussian RBF: $K(a,b) = \exp(-\gamma \|a-b\|^2)$

sigmoid: $K(a,b) = \tanh(\gamma a^T b + r)$

⑪ Making prediction with a kernelized SVM :

$$h_{\hat{w},b}\left(\phi(x^{(n)})\right) = \hat{w}^T \phi(x^{(n)}) + \hat{b}$$

$$= \left(\sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \phi(x^{(i)})\right)^T \phi(x^{(n)}) + \hat{b}$$

$\star \rightarrow$

$$= \sum_{i=1}^{m} \hat{\alpha}^{(i)} t^{(i)} \left(\phi(x^{(i)})^T \phi(x^{(n)})\right) + \hat{b}$$

$$= \sum_{\substack{i=1 \\ \hat{\alpha}^{(i)} > 0}}^{m} \hat{\alpha}^{(i)} t^{(i)} k(x^{(i)}, x^{(n)}) + \hat{b}$$

Using kernel trick to compute bias term $\hat{b}$:

$$\hat{b} = \frac{1}{n_s} \sum_{\substack{i=1 \\ \alpha^{(i)} > 0}}^{m} \left(t^{(i)} - \hat{w}^T \phi(x^{(i)})\right)$$

$$= \frac{1}{n_s} \sum_{i=1}^{m} \left(t^{(i)} - \left(\sum_{j=1}^{m} \hat{\alpha}^{(j)} t^{(j)} \phi(x^{(j)})\right)^T \phi(x^{(i)})\right)$$

$$= \frac{1}{n_s} \sum_{i=1}^{m} \left(t^{(i)} - \sum_{j=1}^{m} \alpha^{(j)} t^{(j)} k(x^{(i)}, x^{(j)})\right)$$

* If you are starting to get a headache, it's perfectly normal; it's an unfortunate side effect of the kernel trick.

⋆ Linear classifier SVM cost function :

$$J(w,b) = \frac{1}{2} w^T w + C \sum_{i=1}^{m} \max\left(0, 1 - t^{(i)}(w^T x^{(i)} + b)\right)$$

Dual form of the linear SVM objective:

$$\underset{\alpha}{\text{minimize}} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}x^{(i)T}x^{(j)} - \sum_{i=1}^{m}\alpha^{(i)}$$

subject to $\alpha^{(i)} \geq 0$

after we find α,

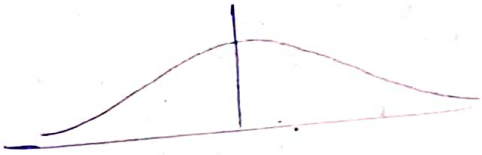
the dual solution to the primal solution:

$$\hat{w} = \sum_{i=1}^{m}\hat{\alpha}^{(i)}t^{(i)}x^{(i)}$$

$$\hat{b} = \frac{1}{n_s}\sum_{i=1}^{m}\left(t^{(i)} - (w^T x)^{(i)}\right)$$

Normal distribution / Gaussian distribution: (standard)

$$f_z(z) = \frac{1}{\sqrt{2\pi}} \cdot e^{-z^2/2} \qquad z \in R$$



Binomial distribution:

$$f(k, n, p) = {}^{n}C_k \, p^k (1-p)^{n-k} = Pr(X = k)$$

Maximum likelihood estimation:

generalised gaussian distribution with mean='μ' & SD='σ',

$$N(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

the parameters μ and σ are represented together as

$$\theta = \{\mu, \sigma\}$$

for a dataset of given size 'n',

$$L = P(x_1, x_2, \cdots, x_n | \theta) = \prod_{i=1}^{n} P(x_i | \theta)$$

$$\hat{\theta}_{MLE} = \underset{\theta}{\text{argmax}} \prod_{i=1}^{n} P(x_i|\theta) \Rightarrow \frac{\delta}{\delta\theta}\prod_{i=1}^{n} P(x_i|\theta) = 0$$

maximizing $\prod_{i=1}^{n} P(x_i|\theta)$ means maximizing its logarithm. [$\underset{x}{\text{argmax}}\, f(x) = \underset{x}{\text{argmax}} \log(f(x))$]

$$\Rightarrow \frac{\delta}{\delta\theta}\log\left(\prod_{i=1}^{n} P(x_i|\theta)\right) = 0 \Rightarrow \frac{\delta}{\delta\theta}\left(\sum_{i=1}^{n}\log P(x_i|\theta)\right) = 0$$

now, $\sum_{i=1}^{n} \frac{\partial}{\partial \theta} \cdot \log\left(P(x_i|\theta)\right) = 0$

$\therefore \sum_{i=1}^{n} \nabla_\mu \left(\log \left(\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \right) \right)$

$\sum_{i=1}^{n} \nabla_\mu \left(-\frac{1}{2} \log(\sigma\sqrt{2\pi}) \right)$

$= \sum_{i=1}^{n} \nabla_\mu \left(-\frac{1}{2} \log(2\pi\sigma^2) - \frac{(x_i-\mu)^2}{2\sigma^2} \right)$

$= \frac{1}{2\sigma^2} \sum_{i=1}^{n} \nabla_\mu (x_i-\mu)^2$

$= \frac{1}{\sigma^2} \sum_{i=1}^{n} (x_i-\mu)$

$\frac{1}{\sigma^2} \sum_{i=1}^{n} (x_i-\mu) = 0$

$\Rightarrow \boxed{\hat{\mu}_{MLE} = \frac{1}{n} \sum_{i}^{n} x_i}$

Again, $\sum_{i=1}^{n} \nabla_\sigma \left(-\frac{1}{2} \log(2\pi\sigma^2) - \frac{(x_i-\mu)^2}{2\sigma^2} \right) = 0$

$\Rightarrow -\frac{n}{\sigma} + \frac{1}{\sigma^3} \sum_{i}^{n} (x_i-\mu)^2 = 0$

$\Rightarrow \hat{\sigma}_{MLE} = \left(\frac{1}{n} \sum_{i}^{n} (x_i-\mu)^2 \right)^{1/2}$

Example:

Suppose the weights of randomly selected college students are normally distributed with unknown μ and σ.

Random sample: 115 122 130 127 149 160, 152 138 146 180

Q: Identify the likelihood function find maximum likelihood estimator $\hat{\mu}$.

$$L = f(x_1, x_2, \cdots, x_n|\theta) = \prod_{i=1}^{n} f(x_i|\theta)$$

(joint density func)

- log likelihood func. $\log L = \sum_{i=1}^{n} f(x_i|\theta)$

- Probability distribution functions

① Binomial distribution $\Rightarrow$ $B(n,p) = \boxed{{}^nC_k p^k q^{n-k}}$, $[q = 1-p]$

mean $= np$

variance $= npq$

② Bernoulli distribution $\Rightarrow$ $Bern(p) = \boxed{p^k (1-p)^{1-k}}$

mean $= p$, variance $= pq$

③ Poisson Distribution $\Rightarrow$ $P_o(\lambda) = \boxed{\frac{\lambda^k e^{-\lambda}}{k!}}$

mean $=$ variance $= \lambda$

④ Normal / Gaussian distribution $\Rightarrow$ $N(\mu,\sigma) = \boxed{\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}}$

mean $= \mu$, variance $= \sigma^2$

- Standard normal: $\mu = 0$, $\sigma = 1$

scaling: $z = (x-\mu)/\sigma$ $\boxed{f_z(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}}$