1. **Linear Regression**
   1. **Linear regression basic**
2. **Assumption**
3. Weak exogeneity.

the predictor variables *x* can be treated as fixed values, rather than [random variables](https://en.wikipedia.org/wiki/Random_variable).

1. Linearity.

The mean of the response variable is a [linear combination](https://en.wikipedia.org/wiki/Linear_combination) of the parameters (regression coefficients) and the predictor variables.

1. Constant variance (a.k.a. [**homoscedasticity**](https://en.wikipedia.org/wiki/Homoscedasticity)).

Different values of the response variable have the same [variance](https://en.wikipedia.org/wiki/Variance) in their errors, regardless of the values of the predictor variables

1. **Independence** of errors.

This assumes that the errors of the response variables are uncorrelated with each other.

1. **Lack of perfect multicollinearity** in the predictors.

For standard [least squares](https://en.wikipedia.org/wiki/Least_squares) estimation methods, the design matrix *X* must have full [column rank](https://en.wikipedia.org/wiki/Column_rank) *p*; otherwise, we have a condition known as perfect [multicollinearity](https://en.wikipedia.org/wiki/Multicollinearity) in the predictor variables

1. **Matrix representation**

Y = Hw + \epsilon

1. **Cost Function**

L(w) =\sum\_{i=1}^N (y - \hat y)^2

= \sum\_{i=1}^N (y - Hw)^2

1. **Analytical Solution**

grad L(w) = -2X^T(y-Hw) =0

w = (H^{T} H)^{-1} H^{T} y

1. **Analysis of analytical solution**
2. To have (H^{T} H)^{-1} invertible, the number of observations > the number of features.
3. Requires matrix inverse which is O(n^3), too computationally intensive.
4. That’s why we need to seek for numerical solution, like gradient descent
5. **Gradient descent algorithm**

Init **w**^{1} = **0**

while ||grad of L(w)||\_2 > \epsilon

For i= 1… D(loop of features)

partial w\_j = -2 \sum\_i^{N} H\_{ji}(y\_i - \hat y\_i (w^{t}))

w^{j+1}\_j = w^{j+1}\_j - \stepsize \* partial w\_j;

t= t+1;

* 1. **Performance Assessment/Model Selection**
  2. **Training/validation/testing data split**

1. Fit the model parameters using the training data
2. Select the model that minimize the error function on the validation data set
3. Use the error on the test set as a generalization assessment of the model
   1. **K fold Cross Validation**
4. Choose K empirically based on a tradeoff between accuracy and computation cost
5. Shuffle the data
6. Divide the data into k set, called data[1] data[2]…..data[k]
7. For(int i =0 ; i<=k; i++)

{ use data\_set[i] as validation set,

The rest data\_set as training set

Fit the model, get RSS\_i

}

1. Average RSS\_Aver(\lambda)
2. Repeat the same procedure 1-4 for models
3. Pick the model that gives the least average RSS(RSS\_Aver)
4. Use this model to train the entire data set.
   1. **Understanding bias and variance tradeoff**

Define f\_{\hat w}(x) as the fitted value average over all possible values of w, then

The mean square error

MSE(f\_{\hat w(train)} (x))

= E\_{training}((f\_{w(true)}(x) – f\_{\hat w}(x))^2)

= E\_{training}( ( (f\_{w(true)}(x) – f\_{\bar w}(x)) +

(f\_{\bar w}(x) – f\_{\hat w}(x) )^2)

= E((f -\bar f)^2) +2E((f-\bar f)(\bar - \hat f))

+ E(\bar f - \hat f)^2

E((f -\bar f)^2) = bias^2 f

E(\bar f - \hat f)^2 = var(\hat f)

2E((f-\bar f)(\bar - \hat f)) = 0

MSE(f\_{\hat w(train)} (x)) = bias^2 f + var(\hat f)

**1.3 Ridge and Lasso regression**

a. **Def**

Ridge uses two norm as penalty and add it into the cost function, \lambda \sum w\_i^2

Lasso uses one norm as penalty and add it into the cost function, \lambda \sum abs(b\_i)

b. **Method**

1) Ridge regression: Gradient descent

**Y** = **Hw** + \epsilon

L(**w**)= \sum\_{i=1}^N (**y** - **Hw**)^2 + \lambda \sum w\_i^2

\grad Loss = -2 **H**^T(y-**Hw**)+ 2 \lambda **w**

Step update: for j\=0

w^(t+1) = w^(t) -stepsize (-2 H^T(y-H\*w) - 2 \lambda w)

if j =0,

w^(t+1) = w^(t) -stepsize (-2 H^T(y-H\*w) )

1. Lasso regression: coordinate descent

L(**w**)= \sum\_{i=1}^N (**y** - **Hw**)^2

L’(**w**)= \sum\_{i=1}^N (**y** - **Hw**)^2 + \lambda \sum\_{j=0} ^ D |w\_i|

Partial L(w) / w\_j

= -2 \sum\_{i=1} ^{N} H\_{ji}(y\_i - \sum\_{j=1}^D w\_j H\_{ji})

= -2 \sum\_{i=1} ^{N} H\_{ji}(y\_i - \sum\_{k not eq j} w\_k H\_{ki})

+ 2 w\_j \sum\_{i=1}^{N} H\_{ji} ^2

We let this equal to

-2 \rho\_j + 2 w\_j z\_j

For the penalty term

\lambda partial |w\_j|/ w\_j

= -lambda when w\_j < 0

[-lambda, lambda] when w\_j = 0

lambda when w\_j > 0

Partial L’(w) / w\_j

= -2 \rho\_j + 2 w\_j z\_j -lambda when w\_j < 0

[-2 \rho\_j -lambda, -2 \rho\_j + lambda] when w\_j = 0

-2 \rho\_j + 2 w\_j z\_j + lambda when w\_j > 0

So w\_j = \frac{\rho\_j + \lambda/2}{ z\_j} if \rho\_j < - \lambda/2

w\_j = 0

w\_j = \frac{\rho\_j - \lambda/2}{ z\_j} if \rho\_j > \lambda/2

1. **Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Ridge | Lasso | Comment |
| Model selection | No | Yes | By drawing the contour, parameter in lasso shrinks to zero |
| Has analytical, and unique solution | Yes | No | Derivative is not continuous in Lass |
| Stable | Yes | No | Rdige can deal better in collearity |

Visualizing ridge cost

Visualizing Lasso cost

1. **Usage**

We usually use Lasso to select parameters, then use ridge to find the optimal solution.

* 1. **Logistic regression**
  2. **Def**

For binary dependent variable, with parameter **w,**

the model states

P(y = +1| x, **w**) = \frac{e^{H(x) **w**}} {1 + e^{ H(x) **w**}}

Then P(y = -1| x, **w**)

= 1 – P(y = +1 | x, **w**)

= \frac{1} {1 + e^{ H(x) **w**}}

And a typically logistic regression task is to fit **w** to the data set (X, Y)

* 1. **Likelihood function and log likelihood function**

L(**w**) = \Pi^{i=1}^{N} P(y\_i | x\_i, **w**) (y\_i can be +1 or 0)

The solution w maximize the L(**w**)

Log L(**w**) = \sum\_i ^N (1 \_{y\_i =1} ln (P(y\_i=1 | x\_i, **w**))

+ 1 \_{y\_i =0} ln(P(y\_i=-1 | x\_i, **w**)) )

= \ sum\_i ^ N (y ln \hat y + (1-y) ln( 1- \hat y ) ))

* 1. Gradient descent solution

This solution minimize cost function which is the negative of the log

likelihood function

Loss(**w**) = - log Likelihood(**w**)

Init **w**^{(1)} = 0

While ||grad Log L(**w**^{(t)})|| > \epsilon

For j = 0…..D

partial[j] = \sum\_i ^N H\_{ij} (1\_{y\_i = +1} – P(y = +1 |

x\_i, w^{(t)}))

w\_j ^{(t+1)} = w\_j ^({t}) + stepsize \* partial[j]

t = t+1;

* 1. How to choose the step size

1. Picking step size requires a log of trials and error
2. Plot learning curve(cost function vs number of step)

Find the step size that is too small

Find the step size that is too large

Then fine tune the step size in between to find the optimal.

1. Logistic regression with penalty

Use L2 norm as penalty term

Then the new loss function

L^’ (w) = L(w) + \sum\_{i=j}^{D} \lambda w\_j^2

For gradient descent, we use

\partial L^’(**w**) = \partial L(**w**) + 2 \lambda **w**

* 1. **Batch gradient descent vs. stochastic gradient descent vs. mini batch gradient descent**

1. **Def**
2. Batch gradient

Batch gradient means using all the data point to calculate the gradient

cost = \sum\_{i=1} ^ {N} –loglikelihood

grad = \partial(cost)/\partial w

update all parameter based on gradient

1. Stochastic gradient descent

Stochastic gradient descent means using all the data point to calculate the gradient

For m = 1 : N

cost = – loglikelihood of the ith sample

grad = \partial(cost)/\partial w

update all parameter based on gradient

1. We divide N samples into G = N/k groups so that each group contains k data points

For n = 1 : G

cost = – \sum\_{(n-1)k} ^{nk} loglikelihood

grad = \partial(cost)/\partial w

update all parameter based on gradient

1. **Comparison**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Time per iteration | Convergence time for large data | Sensitivity to parameters | Smoothness |
| Batch Gradient | Slow for large data | Slower | Moderate | Smooth |
| Stochastic Gradient | Always fast | Faster | High | Very noisy |

1. Practical usage

Shuffle the data before running the stochastic gradient descent

1. **Decision Trees**

2.1 **Greedy algorithm**

1. **Greedy decision tree algorithm**
2. Start with an empty tree
3. Select a feature to split data

Pick a feature, then split data, for each split data set, predict \hat y which the class of the majority data.

1. Calculate the classification error for each feature split, choose the feature split which leads to lowest classification error
2. For each tree node, if all data in the nodes have same y value, or feature split has been done on all possible features then stop on this node otherwise go back to step 2 and keep splitting.
3. **Preventing overfitting: early stopping**

For each node, before we do further feature splitting, if one of the following is met, stop splitting

1. The depth of the tree is larger than the limit
2. Splitting does not improve any classification error
3. The number of data points contained in a node is too small.
4. **Preventing overfitting: Pruning**

Start from the bottom of the tree and traverse up, apply the following step for each of the decision node M(nodes that have children)

1. Compute the total cost function of the Tree C(T) = classification error + \lambda # of leaves(T)
2. Prune node M, compute cost function of the new smaller tree C(T’)
3. If C(T’) < C(T), prune.
   1. **Entropy algorithm**
4. **Entropy**

H = -\sum\_{k=1} ^{K} p\_k log\_2 p\_k

Where p\_1 + p\_2 + …+ p\_n =1

When each state has an equal probability, then p\_k =1/K

H(T) = -K 1/K log\_2(1/K) = log\_2 K

Compared to thermodynamics, the entropy is defined as

S = k\_B ln(W)

The difference are

1. K\_B factor, which is used in the microscopic regime
2. Log base, different log bases are essentially the same except changing a unit.

In binomial case, entropy reduces to

p\_1 = {# of y =1} / N

P\_0= {# of y=0} /N

H = -p\_1 log p\_1 – p\_0 log p\_0

1. **Feature splitting using entropy.**

If feature X takes x\_i values, and its corresponding samples size is N\_xi

1. Entropy before splitting

H(N) == -\sum\_{k=1} ^{K} p\_k log\_2 p\_= -p\_1 log p\_1 – p\_0 log p\_0

1. The entropy after splitting

H(N|X) = \sum\_xi \frac{N\_xi}{N} H(N\_xi)

1. The information gain is = -\Delta H

This is because entropy is a measure of diversity, and the higher entropy, the higher diversity, which means lower information.

1. Repeat step 2) 3) for all the features and choose the feature to split with the most information gain.
2. Cost function for pruning

Parameters:

T\_f: the number of leaves of tree T

t: the index of leaves

N\_t: the total sample size at this leaf

N\_tk: the total sample size where y = y\_k at leaf t

C(T) = \sum\_{t=1} ^{T\_f} N\_t H(t) + \lambda |T\_f|

* 1. **Difference between logistic regression and decision tree**

1. For logistic regression, the decision boundary is a line in 2d, plane in 3d, or hyperplane in high dimensions, which is always **linear**. For decision tree, the decision boundary is more complex.
2. **Clustering** 
   1. **K means{\displaystyle {\underset {\mathbf {S} }{\operatorname {arg\,min} }}\sum \_{i=1}^{k}\sum \_{\mathbf {x} \in S\_{i}}\left\|\mathbf {x} -{\boldsymbol {\mu }}\_{i}\right\|^{2}={\underset {\mathbf {S} }{\operatorname {arg\,min} }}\sum \_{i=1}^{k}|S\_{i}|\operatorname {Var} S\_{i}}aa**
3. **Definition**

Given a set of observations(x1, x2, …, xn), where each observation is a —dimensional real vector, k-means clustering aims to partition the n observations into k(<=n ) sets S={S1, S2,…, Sk} so as to minimize the within cluster sum of squares. Formally, the objective is to find:

argmin\_{S} \sum\_{i=1}{k} \sum{x in S\_i}|| x- ui||^2

= argmin\_{S} \sum\_{i=1}{k} |S\_i| Var S\_i

1. **Algorithm**
2. Give the initial guess of k means m1,…,mk
3. Assign each observation to the cluster whose mean has the least squared Euclidean distance.
4. Calculate the new means to be the centroids of the observations in the new clusters.
5. m\_i^{(t+1)} = \frac{1}{S\_i^{(t)} } \sum\_{s\_j in S\_i^{(t)}} x\_j 3)
6. **Time Complexity**

O(nkdi), where n is the number of d dimensional vectors, k is the number of clusters and i is the number of iterations need till convergence.

1)11dddddddddddddddddcdd

* 1. **Gaussian Mixture**

1. **Idea and Definition**
2. In K means clustering, one sample point solely belongs to one cluster. In other words, we assign a sample point to a cluster with probability 1. In Mixture model, we assign sample point i to a cluster k with the probability r\_{ik}, with

\sum\_{k} r\_{ik} = 1

The r\_{ik} also follows the fact

\sum\_{i} \sum\_{k} r\_{ik} = \sum\_{i} 1 = N

By changing the order of summation

\sum\_{i} \sum\_{k} r\_{ik} = \sum\_{k} \sum\_{i} r\_{ik}

Let weight of cluster: w\_k k be \sum\_{i} r\_{ik} /N

=\sum\_{k} weight of cluster k \* N = N

So \sum\_{k} w\_k = 1

We can also interpret w\_k as a prior distribution of a sample point being assigned to cluster k.

1. And for each cluster k, we define the probability of having a sample point i at x\_i use a normal distribution N(x\_i | u\_k, \Sigma\_k)

Diagram:

1. The r\_{ik} \pi\_k and N(x\_i | u\_k, \Sigma\_k) are connected with Bayesian rule

P(A|B) = \frac{P(A) P(B|A)}{ P(B)}

= \frac{P(A) P(B|A)}{ \sum\_c P(C) |P(B|C) }

P(X\_i = x\_i and X\_i in cluster k)

= P(X\_i in cluster k) P(X\_i=x\_i given X\_i in cluster k)

= P(X\_i in cluster k |X\_i = x\_i) P (X\_i= xi)

So P(X\_i in cluster k |X\_i = x\_i)

P(X\_i in cluster k) P(X\_i=x\_i given X\_i in cluster k)/(X\_i= xi)

Namely,

r\_{ik}

= \frac{\pi\_k N(x\_i | u\_k, \Sigma\_k }{\sum\_j \pi\_j N(x\_i | u\_j, \Sigma\_j)

1. Our goal is the find u\_k, \Sigma\_k, w\_k
2. **Cost function and Minimization**

For a given point x\_i, the likelihood function is

p(x\_i) = \sum\_k \pi\_k N(x\_i | u\_k, \Sigma\_k)

The likelihood function for the whole sample is

\Pi\_{i=1}^{N} p(x\_i) = \Pi\_{i=1}^{N} \sum\_k \pi\_k N(x\_i | u\_k, \Sigma\_k)

The goal is to minimize the negative of Log Likelihood

L = - \Sum\_{i=1}^{N} ln( \sum\_k \pi\_k N(x\_i | u\_k, \Sigma\_k))

1. Take the derivative with respect to u\_k

dL/d u\_k = \sum\_i

( \frac{\pi\_k N(x\_i | u\_k, \Sigma\_k }{\sum\_j \pi\_j N(x\_i | u\_j, \Sigma\_j }

\Sigma^{-1}(x\_i – u\_k))

We found that the term

\frac{\pi\_k N(x\_i | u\_k, \Sigma\_k }{\sum\_j \pi\_j N(x\_i | u\_j, \Sigma\_j }

is exactly r\_{ik}

Let the derivative equal to zero, we have

u\_k = \frac{1}{N\_k} \sum\_i r\_ik x\_i (N\_k = \sum\_i r\_{ik})

1. Taking the derivative with respect to \Sigma\_k gives

\Sigma\_k = 1/N\_k \sum\_i r\_{ik} (x\_i – u\_k)(x\_i – u\_k)^T

1. Taking the derivative with respect to \pi\_k gives

\pi\_k = \frac{N\_k}{N}

We see u\_k, \Sigma\_k, w\_k, r\_{ik} are mutually dependent, therefore we need to solve this iteratively

1. **Algorithm**
2. initialize cluster prior assignment \pi\_k = P(z\_i = k)
3. Given an observation x\_i from cluster k, calculate P(x\_i | z =k , u\_k, \Sigma\_k) = N(x\_i | u\_k, \Sigma\_k)
4. E step

Given an observation x\_i, calculate r\_ik

r\_{ik}

1. M step

N\_k = \sum\_i^N r\_{ik}

\hat u\_k = \sum\_i^N frac{ r\_{ik}\}{N\_k} x\_i

\hat \Sigma\_k = \sum\_i^N frac{ r\_{ik}\}{N\_k}

(x\_i - \hat u\_k) (x\_i - \hat u\_k)^{T}

\pi\_k = \frac{N\_k} {N}

1. **Connection to K means**

In order to easily see how Gaussian mixture clustering relates to K means, we need to introduce another latent variable Z and consider the log likelihood function of the complete data set (X, Z).

We discussed the probability to assign a sample point to cluster k as \pi\_k, now we assign \pi\_k to an indicator random variable Z\_{ik} = (z1, z2, …,z\_K)

Z1 = (1,0,0,0…..0), Zk=(0,0,0,1(kth element),…,0)

Where z\_k = 1 when a sample point is assigned to k cluster

= 0 otherwise

And p(Z =Z\_k )= \pi\_k = \Pi\_k \pi\_k^{z\_k}

We rewrite the likelihood function given X and Z

L = \Pi\_{i=1}^N \Pi\_{k=1}^K \pi\_k^{z\_ik} N(x\_i| u\_k, \Sigma\_k)^{z\_ik}

If we let \pi\_k = 1/K, and the covariance matrix = \sigma^2 I

Log L = \sum\_i \sum\_k (log \pi\_k -1/2 1/\sigma^2 ||x\_i-u\_k||^2)

This reduces to the K means cost function

Now let \sigma ->0

r\_ik

=\frac{\pi\_k N(x\_i | u\_k, \Sigma\_k }{\sum\_k \pi\_k N(x\_i | u\_k, \Sigma\_k )

When \sigma goes to zero, the exponential term that decays the slowest survives, and the term that decays the slowest is the one that minimize ||x\_i-u\_k ||. Let u\* be the u\_k that minimize ||x\_i- uk||, then

r\_ik = 1 for k =k\*

0 otherwise

This reduces to the k means where a sample point i is solely assigned to a cluster k.

1. **Support Vector Machine**

**4.1 Support Vector Machine: Basic**

1. **Motivation: From logistic regression cost function to support vector machine cost function**
2. **Logistic Cost function:**

For y = 1

logistic regression SVM

For y = 0

logistic regression SVM

1. **Support Vector Machine Cost function:**

Equivalently:

Hypothesis

1. When C is large,

Geometric intuition for large C: Large Margin.

1. **Concept**

Given a dataset(\hat x\_1, y\_1), (\hat x\_2, y\_2)

Based on the cost function

We use a hyperplane w^T \* x + b =0, given this plane, we need

if y=1, \hat w \* \hat x + b >= 1

if y=-1, \hat w \* \hat x + b <= -1

Graph

and the distance between the two hyperplanes are 2/||w||, so maximize the distance is the same as minimize ||w||. Therefore our problem becomes

min ½ |||w||^2

st y^(i)\*( w^T x^(i) + b) >=1

1. **Method**
2. Lagrangian Duality
3. SMO, Coordinate Descent
4. **Pros and cons**

Pros

Firstly it has a regularization parameter, which makes the user think about avoiding over-fitting. Secondly it uses the kernel trick, so you can build in expert knowledge about the problem via engineering the kernel. Thirdly an SVM is defined by a convex optimization problem (no local minima) for which there are efficient methods (e.g. SMO). Lastly, it is an approximation to a bound on the test error rate, and there is a substantial body of theory behind it which suggests it should be a good idea.

Cons

The disadvantages are that the theory only really covers the determination of the parameters for a given value of the regularisation and kernel parameters and choice of kernel.

* 1. **Support Vector Machine with Kernels.**

1. **Motivation of Kernels**

Suppose we have a one dimensional case

We want to make a boundary decision that separate 1 and 0. It is quite straightforward to see that it is impossible to have one boundary decision. However, what we can do is to make this one dimensional case into two dimension. We add a second coordinate which is x2 = x1^2, then this case becomes linear separable, namely we can draw a straight line that separates 1s and 0s.

1. **Optimization**
   1. **Gradient Descent.**

See gradient descent solution in regression section

* 1. **Newton Method**

1. Newton Method Principles

Based on Taylor’s expansion if we are at x\_0, we try to find \delta x so that x\_0 + \delta X is closer to the stationary point.

f(x\_0 + \delta x ) = f(x\_0) + f^{‘}(x\_0) \delta x + f^{‘’}(x\_0) (\delta x) ^ 2

take the derivative

d f(x\_0 + \delta x)/d x = f^{‘}(x\_0) + f^{‘’}(x\_0) \delta x

therefore

\delta x = - \frac{f^{‘}(x\_0)}{f^{‘’} (x\_0)}

X^{(t+1)} = X^{t} - \frac{f^{‘}(x\_0)}{f^{‘’} (x\_0)}

1. Matrix Forms

x^{(t+1)} = x^{t} - H^{-1}(f(x^{t})) \grad f(x^{t}) where H is the Hessian matrix

1. Connection with Gradient descent

The newton method can be reduce to gradient descent method by taking Hessian matrix as Identity matrix

1. Pros

Since it utilizes the second order derivative, it converges much faster than gradient descent.

For quadratic function, the equation from the Taylor expansion is exact, therefore the stationary point can be found using only one step

1. Cons

Need to evaluate the inverse of the Hessian Matrix, so it is computationally expensive

* 1. **Other optimization method**

1. Quasi Newton

Newton method requires the inverse of the Hessian matrix, which is usually not easy to solve. So we need to find an approximation of the Hessian. Similar to the way we solve for gradient, we can use finite difference method, in which the gradient is

grad f(x) = \frac{f(x+\delta x) – f(x)} {\delta x},

this is only exact when \delta x approaches zero. For 2nd order derivative, we can write

f^{‘’}(x) = \frac{f^{‘}(x+\delta)- f^{‘}(x)}{\delta}

again this is only exact when \delta is zero. Based on this idea we

replace the Hessian Matrix with an approximation that satisfies the following approximation

\grad f(x + \delta x) = \grad f(x) + B\delta x

This is quasi newton method

Various Quasi Newton methods exist with different choice of B

1. Levenburg Marquadt

This Method adds a scaled Identity matrix uI to the Hessian, for large u and small Hessian, the method is equivalent to gradient descent with step size 1/u.

1. **Principle Component Analysis**
2. **Geometry Intuition**

Imagine we have a two dimensional plane with axis x1 and x2 perpendicular to each other. On this plane we have a data set(x1\_i, x2\_i), if most of the data lies on the 45 degree angle between the x1 and x2 axis. If we do a coordinate transformation by rotating the x1 and x2 axis by 45 degree counter clockwise, we get new axis z1 and z2. We see our data mainly lies on z1 axis. This means we reduce a two dimensional data to one dimension.

1. **Definition**

Suppose we have a data set (X,y) where x is the feature variable. It is an mxn matrix where m is the data size, and n is the dimension of the features

x11 x12  …. x1n

x21 x22  …. x2n

…

xm1 xm2  …. xmn

We call the feature vector associated with the ith data x(i), we consider a coordinate transformation:

Z = X W

m x n m x n n x n

z11 z12  …. z1n x11 x12  …. x1n w11 w12  …. w1n

z21 z22  …. z2n = x21 x22  …. x2n  w22 w22  …. w2n

… …

zm1 zm2  …. zmn xm1 xm2  …. xmn wn1 wn2  …. wnn

The goal is to reduce the dimension of the feature to d, still have a good representation of the data.

When Z has only d (d<n) dimension, then

Z = X W

m x d m x n n x d

z11 z11  …. z1d x11 x12  …. x1n w11 w12  …. w1d

z21 z22  …. z2d = x21 x22  …. x2n  w21 w22  …. w2d

… …

zm1 zm2  …. zmd xm1 xm2  …. Xmn wn1 wn2  …. wnd

The problem is how to choose d dimensions out of n.

We define the error function as

\sum\_i ^{m} = || xi – zi||22

=|| X – W WT x||22 (This is a little involved)

W = argmax || X – X W WT||22

= argmax tr(WT X^TX W)

Let w\_1, w\_2,…w\_n be the column vectors of matrix W, then

tr(WT X^TX W)

= w\_1^T X^TX w\_1 + w\_2^T X^TX w\_2 +… + w\_d^T X^TX w\_d

= \sum w\_i^T w\_i \lamdba\_i (\lambda\_i is the ith eigenvalues of X^T X)

If w\_i are the eigenvector corresponding to \lambda\_i

Then the maximum value of the trace is achieved when we take w\_1 to w\_d as the eigenvectors associated with the first d maximum eigenvalues.

**7. Ada Boosting**

**a. Def**

The boosting classifier is a linear combination of weak classifiers, and the goal is to find the linear coefficients of each classifier. Imagine, we have m-1 weak classifier

and we would like to add one more weak classifier

We define the error function is

Let

E =

Take the derivative with respect to /alpha

=

**8. K nearest Neighbors**

**a. Def**

Given a data set and a new sample, find k dataset that are closed to the new sample.

**b. Algorithm using brute force search**

1) Initialize a priority queue using the distance between the first k samples and the new sample.

2) For the rest of the N-k sample, insert the distance between the sample and the new sample to the priority queue.

3) After looping all the samples, the data set that remains in the priority queue are the k nearest neighbors.

Complexity N \*log k, log k term comes as we implement the priority queue as a heap.

**c. Algorithm using KD tree**

**KD tree**

Every non-leaf node can be thought of as implicitly generating a splitting [hyperplane](https://en.wikipedia.org/wiki/Hyperplane) that divides the space into two parts, known as [half-spaces](https://en.wikipedia.org/wiki/Half-space_(geometry)). Points to the left of this hyperplane are represented by the left subtree of that node and points to the right of the hyperplane are represented by the right subtree.

**Choice to construct KD tree:**

1. Choose the kth dimension to split
2. Choose the value to split(usually median)
3. When to stop(hyperspace contains less than 2 nodes)

**Query:**

1. Starting with the root node, the algorithm moves down the tree recursively, in the same way that it would if the search point were being inserted (i.e. it goes left or right depending on whether the point is lesser than or greater than the current node in the split dimension).
2. Once the algorithm reaches a leaf node, it saves that node point as the "current best"
3. The algorithm unwinds the recursion of the tree, performing the following steps at each node:

(1)If the current node is closer than the current best, then it becomes the current best.

(2)The algorithm checks whether there could be any points on the other side of the splitting plane that are closer to the search point than the current best. In concept, this is done by intersecting the splitting [hyperplane](https://en.wikipedia.org/wiki/Hyperplane) with a [hypersphere](https://en.wikipedia.org/wiki/Hypersphere) around the search point that has a radius equal to the current nearest distance. Since the hyperplanes are all axis-aligned this is implemented as a simple comparison to see whether the distance between the splitting coordinate of the search point and current node is lesser than the distance (overall coordinates) from the search point to the current best.

* + 1. If the hypersphere crosses the plane, there could be nearer points on the other side of the plane, so the algorithm must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
    2. If the hypersphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.

1. When the algorithm finishes this process for the root node, then the search is complete.