# Review1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| method | key properties to remember | prediction | learning | R function |
| linear regression | + try to learn the linear relationship between x and y for prediction  y = wx + b  salary = w\*degree + b  food\_price = w\*gas\_price + b | + for each x, we just compute  y = w\*x + b | + Gradient Descending/Ascending (minimize or maximize)  + The L2-cost function  J(w) = sum(j=1..n) { (w\*xj + b - yj)^2 }  + The L1-cost function  J(w) = sum(j=1..n) { |w\*xj + b - yj| } | M <- lm(y ~ x, data=D)  y <- predict(M, newdata=D); |
| logistic regression | + for classification (not for prediction)  + try to model the probability function (or discriminant function) | + for each input x, we have  y = ( p(face|x) ? "face" : "back" );  where  p(face|x) = 1/(1 + exp(w\_face\*x)) | + Gradient Descending/Ascending (minimize or maximize)  + The L2-cost function for binary case  J(w) = sum(j=1..n) {  yj\*log(h(x, w)) + (1-yj)\*log(1 - h(x, w)) }  in which h(x, w) = 1/(1 + exp(w\*x) ); | M <- multino(y ~ x, data=D);  y <- predict(M, newdata=D); |
| kmeans /clustering | + to divide the input into groups (clusters) without any output classes (unsupervised learning)  + we have to find k centers for labeling the new input | + assuming that we have k centers (c1, c2, .., ck) then for each input x we have the output label  y = { the nearest center of x }  = argmin(j=1..k){ dist(x, cj) } | + we use kmeans procedure to learn  + first we randomizes centers    + then we compute over the generated centers.  + if we have the labels, we will compute centers. | M <- kmeans(data, k, …)  M <- kmeans(data, centers, …) |
| EM | + for each x, we can assign k labels (sim(x, c1), sim(x, c2), .. sim(x, ck) )  + the labels should be normalized to show the contribution of vector x to each center  x -> (0.5, 0.5, 0, 0, 0) means x half-belongs to c1 and half-belongs to c2 and does not belong to other centers | + for each input x, we have to  y = (sim(x, c1), sim(x, c2), .. sim(x, ck) ) | + very similar to kmeans  + C0 = randomize(X);  + L0 = label(X, C0)  + L0 = normalize(L0);  + C1 = average(X, L0)  + L1 = label(X, C1)  + L1 = normalize(L1);  + C2 = average(X, L1)  + L2 = label(X, C2)  + L2 = normalize(L2);  … | \*\*\* ignore |
| svm | + using the linear plane y = w\*x + b to separate the sample (face from background)  + w is computed by support vectors w = sum(j) { aj\*yj\*xj }  + use kernel function K(x1, x2) or the transformation function phi(x) to learn non-linear dataset  + soft margin svm can be done with C parameters | + for linear  y = sign( w \* x + b)  + for nonlinear  y = sign( w \* phi(x) + b) | + SMO algorithm | M <- svm(x, y, type="C", kernel="polynomial", degree=3)  # this svm function will use the SMO algorithm to learn the svm model M  y <- predict(M, newdata=D); |
| decision tree | + fast  + relationship x->y  + homogeneity  + output class division | + follow from root leave of the tree to find the output class | + CART, C4.5, ID3 | M <- train(y ~ x, method="rpart", data=D)  y <- predict(M, newdata=D); |
| bagging | + combine weak classifier into one powerful classifier  + bootstrap the data to have many datasets and learn the weak classifiers on each dataset | + we have to apply the input x to all the weak classifiers to have y1=c1(x), .., yk=ck(x)  + we vote or we average the results  y = vote(y1, y2, .., yk)  y = avg(y1, y2, .., yk) | + generate (D1, D2, .., Dk) from D  + train (c1, c2, .., ck) from (D1, D2, .., Dk) | \*\*\* ignore |
| boosting | + combine weak classifiers  + we don’t remove samples but we use weights to emphasize them in training | y = a1\*h1(x) + a2\*h2(x) + .. ak\*hk(x) | + we learn (h1, h2, .., hk) with current weights then we date weights  + (a1, a2, .., ak) is updated by weights | M <- train(y ~ x, method="gbm", data=D)  y <- predict(M, newdata=D); |
| random forest | + combine many trees  + we select/remove both rows and columns so that the trees will have to use all columns | + same as bagging  + we apply all the tree for each input x, then we vote/average the result | + generate (D1, D2, .., Dk) from D with row/col selection  + train (c1, c2, .., ck) from (D1, D2, .., Dk) | M <- train(y ~ x, method="rf", data=D)  y <- predict(M, newdata=D); |