PSEUDOCODES

Machine Learning

Varnika Goel

40821121

LINEAR REGRESSION

Input: x- independent variable, y-dependent variable, w- weights (coefficients), b- intercept, n=number of features in x, alpha-learning rate, iteration- number of iterations

Equation for linear regression- w0+w1x1+w2x2+…………wnxn

w=weights

w0=bias

Output:

function linearregression(x,y)

    weight,bias=random points equal to the total features

    for i in range(iteration)

        predict value for y

        ypred= weight*x

    loss/error
    error= y-ypred
    sum_error=sum(error)

    new_weight=update the weights by subtracting the weight with product of alpha, sum_error and 1/n

    new_bias=update the bias by subtracting the bias with product of alpha, sum_error and 1/n

repeat until error is minimized

return ypred

LOGISTIC REGRESSION

Input: x- independent variable, y-dependent variable, w- weights (coefficients), b- intercept, n=number of features in x, alpha-learning rate, iteration- number of iterations

Equation for logistic regression- $z=1/e^{-y}+1$, where y is linear regression, w0+w1x1+w2x2+…………wnxn

w=weights, w0=bias

Output:

function logisticregression(x,y)

weight,bias=random points equal to the total features

for i in range(iteration)

predict value for y

ypred= weight*x

loss/error
error= y-ypred
sum_error=sum(error)

new_weight=update the weights by subtracting the weight with product of alpha, sum_error and 1/n

new_bias=update the bias by subtracting the bias with product of alpha, sum_error and 1/n

(repeat until error is minimized)

$z=1/e^{-ypred}+1$

(if z<0.5

then ypred=0

else

ypred=1)

return z

NAÏVE BAYES

Input: dataset

Output:

split the dataset
xtrain= features of training dataset
ytrain= class of training dataset

xtest=features of test dataset

take out unique values of y= uniy

for i in uniy:

> calculate the individual probabilities of each feature according to test's features wrt to each by dividing the count of particular instance by the total occurrences of that feature in that class
>
> calculate the total probability of particular class by adding the above individual probabilities and the probability of that class wrt to the all the class (i.e. class/ length of dataset)

print the probability of the class having maximum total probability

DECISION TREES

Input: x-features y-labels

Output:

```
function tree(x,y)
     if y are all the same:
        return first label as leaf node
    if x==0
       return most common y
    attribute=select attribute on which splitting is to be done(x,y)
    split data into two subparts
    select the sub-attributes for both right and left side
    recursively repeat to form sub nodes
    return tree
```

RANDOM FORESTS

Input: n- number of subsets, xtrain- features of training dataset, ytrain- class of training dataset, xtest- features of test dataset, dt- subset decision trees

Output:

```
function random_forest(xtrain,ytrain,xtest, dt, n):

        xtrain_sub=n subsets of xtrain

        ytrain=n subsets of y train

store decision trees in a location
dt_loc=[]

train n subsets trees
for i in range n
        m=dt()
        fit xtrain_sub[i],ytrain_sub[i] in m

        put m values in dt_loc

for m in dt:
        predict values for xtest

return the average of prediction
```

BAGGING

Input: n- number of subsets, xtrain- features of training dataset, ytrain- class of training dataset, xtest- features of test dataset, model- subset model

Output:

```
function bagging(xtrain,ytrain,xtest, model, n):

        xtrain_sub=n subsets of xtrain

        ytrain=n subsets of y train

store model in a list
models=[]

train n subsets
for i in range n
        m=model()
        fit xtrain_sub[i],ytrain_sub[i] in m

        put m values in list(models)

for m in model:
        predict values for xtest

return the average of prediction
```

KNN

Input: dataset

Output:

split the dataset into xtrain, xtest,ytrain,ytest

k=choose number of clusters

calculate the distance between each xtest data point and each xtrain point

for i in xtest:

for j in xtrain:

function distance(i,j)

    calculate the Euclidian distance between i and j by subtracting the sum of squares and square rooting the result

store all distances in a list, say d=[]

sort the distances in ascending order

for a in d

    top=select top k distances

    y=determine most common label among ‘top’ (label count=max)

return y

K MEANS

Imput: k-number of clusters

Output:

select k random points and assume them to be centroids of k clusters

centroid=[point 1, point 2, point 3….point k]

function distance(data point, centroid)

calculate the Euclidian distance between the data point and each cluster by subtracting the sum of squares and square rooting the result

find closest centroid where distance is minimum

for i in cluster:

calculate mean of all data points by summing i/total i in that dataset

assign the mean value as new centroids

repeat until centroids do not change i.e if new centroids==centroid,

return clusters and centroids

HIERARCHICAL CLUSTERING

Input: dataset

Output:

function hierarchicalclustering(dataset)

    consider all data points as individual clusters

    distance=[]

    compute Euclidian distance between the points by subtracting the sum of squares and square rooting them

    cluster=[], put all clusters in list

    while len(cluster)>1

        find closed clusters and merge them i.e. cluster1+cluster2

        repeat till you get one cluster

    delete the previous clusters from list(cluster) & update new cluster values

    delete the previous clusters from list(distance) & update new distance values

return cluster