Pre-processing

הגשת עבודה 1- שניר בן יוסף 307908699

1) נירמולים

מהו נרמול נתונים ?

נרמול נתונים הוא תהליך מקדים שיש לבצע לנתונים כדי שכל הנתונים יהיו באותו קנה מידה, כאשר יש לנו נתונים שהם לא באותו קנה מידה זה עלול להוביל לביצוע מודלים באיכות נמוכה.

מהי סטנדרטיזציה של נתונים?

סטנדטיצזיה היא תהליך שינוי קנה מידה של הנתונים כל שיהיו להם תכונות גאוסיות, זאת אומרת שיהיה להם μ=0 ו - σ=1 כאשר μ זה ממוצע וσ זה סטיית התקן מהממוצע.

ההבדלים ביניהם?

סטנדרטיזציה ונורמליזציה נוצרים כדי להשיג מטרה דומה , שהיא לבנות נתונים עם טווחים דומים זה לזה, והמתכנת משתמש בהם באופן נרחב בניתוח הנתונים, סטנדרטיזציה היא חיסור הממוצע ואז חלוקה לפי סטיית התקן, ונורמליזציה היא תהליך חלוקת הווקטור לאורכו וזה הופך את הנתונים שלנו להיות בין 0 ל1.

– Min-Max Normalization

סוג זה של נרמול, משתמש בטרנספורמציה לינארית, נמצא את הערך הגדול ביותר ואת הערך הקטן ביותר ואז נשנה כל ערך בתא לפי הנוסחא הבאה :
כאשר X_i הוא הערך המקורי, ו- X'_i הוא הערך החדש.

$$X'_i = \frac{X_i - \min(x)}{\max(x) - \min(x)}$$

-Z-score normalization

סוג זה של נרמול מתבסס על הממוצע וסטיית התקן, ע"י הנוסחה:

כאשר Z_i הערך המנורמל , X_i הערך המקורי μ זה הממוצע וσ הוא סטיית התקן

$$Z_i = \frac{X_i - \mu}{\sigma}$$

– Decimal Scaling

סוג זה של נרמול עושה העברה של הנקודה העשרונית כדי לנרמל בטכניקה הזאת נחלק כל ערך של הנתונים במספר המקסימלי המוחלט של הנתונים לדוגמא: המספר הגדול ביותר שלנו הוא 54 אז הערך המקסימלי יהיה 100. נשתמש בנוסחה :
כאשר j הוא מספר האיברים במספר הגדול ביותר, במקרה שלנו j=2 . U_i הערך הסופי, v_i הערך בתא המוחלף.

$$U_i = \frac{v_i}{10^j}$$

Min-Max-Func:

[203]:
```
def minmax(data):
    maxnum=max(data)
    minnum=min(data)
    for i in range(len(data)):
        if (maxnum-minnum)!=0:
         data[i]=(data[i]-minnum)/(maxnum-minnum)
    return data

minmax([23,123,534,657,346,48,99,556])
```

[203]:
```
[0.0,
 0.15772870662460567,
 0.805993690851735,
 1.0,
 0.5094637223974764,
 0.03943217665615142,
 0.11987381703470032,
 0.8406940063091483]
```

z-score Func:

[204]:
```
import statistics

def zscore(data):
    std=0
    mean= statistics.mean(data)
    for i in range(len(data)):
        std+=(data[i]-mean)**2

    std=(std/len(data))**0.5
    for i in range(len(data)):
        data[i]=((data[i]-mean)/std)
    return data

zscore([10,15,20,35])
```

[204]:
```
[-1.0690449676496976, -0.5345224838248488, 0.0, 1.6035674514745462]
```

Decimal Scaling Func:

```
[205]: def decimalscaling(data):
           maxnum=0
           for i in range(len(data)):
            if maxnum< abs(data[i]):
               maxnum=abs(data[i])
           j=len(str(abs(maxnum)))
           for i in range(len(data)):
               data[i]=data[i]/10**j
           return data
       decimalscaling([56,34,12,-567,34,23])
```

```
[205]: [0.056, 0.034, 0.012, -0.567, 0.034, 0.023]
```

יש לחקור ולמצוא ספריות פייתון שיודעות לבצע דיסקרטיזציה בעזרת דליים ולבצע בעזרתן את אותן פעולות

Min-Max – using skleran:

```
[206]: from sklearn.preprocessing import minmax_scale
       data=[23,123,534,657,346,48,99,556]
       minmax=minmax_scale(data)
       print(minmax)
```

```
[0.         0.15772871 0.80599369 1.         0.50946372 0.03943218
 0.11987382 0.84069401]
```

Z-score – using scipy:

```
[207]: from scipy import stats
       data=[10,15,20,35]
       stats.zscore(data)
```

```
[207]: array([-1.06904497, -0.53452248, 0.  , 1.60356745])
```

Decimal Scaling – using skleran:

```
[220]: from sklearn.preprocessing import MaxAbsScaler
       data = [[ 12, 13,  26],
             [ 123,  233,  54],
             [ 677,  33, 45]]
       transformer = MaxAbsScaler().fit(data)
       transformer.transform(data)
```

```
[220]: array([[0.01772526, 0.05579399, 0.48148148],
             [0.1816839 , 1.        , 1.        ],
              [1.      , 0.1416309 , 0.83333333]])
```

2) דיסקרטיזציה לא מונחית של ערכים רציפים

מהי דיסקרטיזציה?

דיסקרטיזציה היא שיטה לעיבוד נתונים המשמשת למזעור ההשפעות של טעיות תצפית קטנות, ערכי הנתונים (ללא שינוי) מחולקים ל"מרווחים" קטנים המכונים פחים/דליים ואז הם מוחלפים בערך כללי, שמחושב לאותו פח/דלי, זה משפיע על ההחלטה של נתוני הקלט ועשוי גם להפחית ת הסיכוי להתאמת יתר.

Equal-frequency discretization - מהי דיסקרטיזציה של עומק שווה?

בדרך זו אנחנו יוצרים פחים/דלים המכילים את אותו מספר איבריים אבל מסודר (עולה או יורד)

Equal-frequency discretization func:

[209]:
```
def equalfreq(data,numbin):
    data.sort()
    size=len(data)
    nfd=int(size/numbin)
    bins=[]
    for i in range(0,numbin):
     arr=[]
     for j in range(i*nfd,(i+1)*nfd):
         if j >=size:
            break
         arr=arr+[data[j]]
     bins=bins+[arr]
    return bins
data=[1,99,88,9,23,34,45,56,67,78,12,32]
equalfreq(data,3)
```

[209]: [[1, 9, 12, 23], [32, 34, 45, 56], [67, 78, 88, 99]]

Equal-width discretization - מהי דיסקרטיזציה של רוחב שווה ?

זאת שיטה שמחלקים את האיברים לפחים/דליים לפי המספר העשרוני

Equal- width discretization func:

```
[210]: def equiwidth(data, numbin):
           size = len(data)
           w = int(round((max(data) - min(data)) / numbin))
           minnum = min(data)
           arr = []
           for i in range(0, numbin + 1):
               arr = arr + [minnum + w * i]
           bins=[]

           for i in range(0, numbin):
               temp = []
               for j in data:
                   if j > arr[i] and j < arr[i+1]:
                       temp += [j]
               bins += [temp]
           return bins

       data=[1,959,88,9,23,2334,45,56,617,78,12,32]
       equiwidth(data,3)
```

```
[210]: [[88, 9, 23, 45, 56, 617, 78, 12, 32], [959], [2334]]
```

Equal-frequency discretization - Using pandas

```
[211]: import pandas as pd data =
       pd.DataFrame({'a':
       [4,8,9,15,21,24,25,26,27,28,29,34]})
       pd.qcut(data.a,4,retbins=True)
```

```
[211]:Name: a, dtype:
       category
       Categories (4, interval[float64]): [(3.999, 13.5] < (13.5,
       24.5] < (24.5,27.25] < (27.25, 34.0]], array([ 4. , 13.5 ,
       24.5 , 27.25, 34. ]))
```

Equal-width discretization - Using pandas

```
[212]: import pandas as pd data = pd.DataFrame({'a':
       [42,9,12,15,21,24,25,26,27,281,98,334]})
       pd.cut(data.a,bins=4,retbins=True)
```

```
[212]:Name: a,dtype: category
Categories (4, interval[float64]): [(8.675, 90.25] < (90.25, 171.5]
< (171.5,
      252.75] < (252.75, 334.0]], array([
       8.675, 90.25 , 171.5 , 252.75 , 334.
           ]))
```

3) החלקה

הסבירו מהי החלקת נתונים ?

החלקת נתונים היא טכניקה סטטיסטית להוצאת מידע שהוא "רעש" זאת אומרת מידע שהוא חורג בצורה גבוהה מהנורמל ואז נוכל לקבל תבנית מדויקת יותר .

הסבירו מהו ממוצע נע ואיך הוא עוזר בהחלקת נתונים?

ממוצע נע הוא טכניקה להבנה כללית של טרנדים בנתונים, ממוצע של כל תת קבוצה של מספרים.
הממוצע הנע שימושי לחיזוי מגמות לטווח ארוך.

Simple Moving Average- הממוצע לאורך התקופה והטווח שצוין

[213]:
```
def sma(data):
    size= len(data)
    result=data[0]
    for i in range(1,size):
        result+=data[i]
    return result/size

sma([1,2,3,4])
```

[213]: 2.5

Weighted Moving Average- שיטה זאת נותנת יותר חשיבות על הנתונים האחרונים ופחות על נתונים ישנים זה נעשה ע"י הכפלת מחיר כל תא ע"י איזשהו גורם משוקלל

[300]:
```
def wma(data,period=4):
    x=0
    j=1
    sump=sum(list(range(1,period+1)))
    for i in range(period-1,-1,-1):
        x=x+(data[i]*j)
        j+=1
    x=x/sump
    return x
wma([1,2,3,4])
```

[300]: 2.0

Exponential Moving Average- הוא סוג של שיטת ממוצע נע שמציב

משקל ומשמעות גדולים יותר על נקודות הנתונים האחרונות, השיטה מגיבה טוב יותר לשינויים בנתונים האחרונים

[279]:
```
def ema(data,period,pricetoday,yesterday):
    smares=sma(data)
    wm=2/(period+1)
    return pricetoday*wm+smares*(1-wm) #EMA= Pricetoday*weighting
 multiplier+ema(yasterday)*(1-weighting multiplier)

ema([1,2,3,4],4,3)
```

[279]: 2.7

Binning Methods for Data Smoothing- הסבר

שיטה זו משמשת להחלקת נתונים או טיפול בנתונים רועשים, הנתונים ממוינים ואז מחלקים את הנתונים לדליים לפי השיטה שנבחרה

Smoothing by bin means- בשיטה זו כל ערך בדלי מוחלף בערך הממוצע של הדלי

[216]:
```
def smoothinbymean(data,bins):
    data.sort()
    size=len(data)
    ingroup=size/bins
    arr =[]
    arr1=[]
    for i in range(bins):
        arr1=arr1+[data[:int(ingroup)]]
        arrmean=sum(arr1[i])/len(arr1[i])
        for k in range(len(arr1[i])):
            arr1[i][k]=arrmean
        data=data[int(ingroup):]

    return arr1

smoothinbymean([1,2,3,4,5,6,7,8,9,10,11,12],3) #enter data and number of bins
```

[216]: [[2.5, 2.5, 2.5, 2.5], [6.5, 6.5, 6.5, 6.5], [10.5, 10.5, 10.5, 10.5]]

Smoothing by bin boundary-

בשיטה זו הערכים המינימלים והמקסימלים בדלי הם הגבולות ואז כל ערך בדלי מוחלף בערך הקרוב לו יותר

[217]:
```
def smoothinbybound(data,bins):
    data.sort()
    size=len(data)
    ingroup=size/bins
    arr1=[]
    for i in range(bins):
        arr1=arr1+[data[:int(ingroup)]]
        for k in range(1,len(arr1[i])):
            if ((arr1[i][k]-arr1[i][0]) < (arr1[i][len(arr1[i])-1]-arr1[i][k])):
                    arr1[i][k]=arr1[i][0]
            else:
                arr1[i][k]=arr1[i][len(arr1[i])-1]
        data=data[int(ingroup):]

    return arr1

smoothinbybound([1,3,9,14,25,36,47,48,59,110,121,250],3) #enter data and number of bins
```

[217]: [[1, 1, 14, 14], [25, 25, 48, 48], [59, 59, 59, 250]]

יש לחקור ולמצוא ספריות פייתון שיודעות לבצע דיסקרטיזציה בעזרת דליים ולבצע בעזרתן את אותן פעולות

Simple Moving Average- using pandas

[262]:
```
df=pd.DataFrame({'data':[1,2,3,4]})
df
df.rolling(window=4).mean()
```

[262]:
```
data   0
 NaN
1   NaN
2   NaN
3   2.5
```

Weight Moving Average- using pandas

[303]:
```
df=pd.DataFrame({'data':[1,2,3,4]})      weights      =
np.arange(1,5)    df.rolling(window=4).apply(lambda    nums:
np.dot(nums, weights)/weights.sum(),␣ ↪raw=True)
```

[303]:
```
data   0
 NaN
1   NaN
2   NaN
3   3.0
```

Exponential Moving Average- using pandas

[293]:
```
import pandas as pd
df=pd.DataFrame({'data':[1,2,5,6,7,8,9,3,4]})
df
df.ewm(span=9,adjust=False,min_periods=2).mean()
```

[293]:
```
        data
0        NaN
1        1.2000002 1.960000 3
         2.768000 4 3.614400 5
         4.491520 6 5.393216 7
         4.914573
8 4.731658
```

Smoothing by bin means- using scipy

```
[312]: import scipy
values =
[1,2,3,4,5,6,7,8,9,10,1
1,12]
stats.binned_statistic([1,2,3,4,5,6,7,8,9,10,11,12], values,
'sum', bins=4)
```

```
[312]: BinnedStatisticResult(statistic=array([ 6., 15., 24., 33.]),
bin_edges=array([
      1. , 3.75, 6.5 , 9.25, 12. ]), binnumber=array([1, 1, 1, 2, 2,
      2, 3, 3, 3, 4, 4, 4], dtype=int64))
```

Smoothing by bin boundaries-

*לא הצלחתי למצוא פונקציה שמבצעת את זה, חיפשתי בscipy ובpandas וגם בskleran

חשבתי שאולי זה יתאים :

scipy.stats.binned_statistic

אבל זה לא התאים לי