Preparing Data

Representation

- Data usually has many features
- Feature type
  - Numerical (has order and distance relation)
  - Categorical/symbolic
  - Continuous (quantitative)
    - Interval scale (zero is places arbitrarily)
      - Ex: temperature
    - Ratio Scale (zero has absolut position)
      - Ex: height, weight

Representation

- Discrete (qualitative)
  - Nominal scale (order-less scale)
    - Ex: product type (electronic, fashion, food)
  - Ordinal scale (ordered, discrete graduation)
    - Ex: rank
  - Periodic variable (has order but no distance)
    - Ex: days of the week, month or year
  - Static Data (does not change with time)
  - Dynamic Data (change with time)

Raw Data for Mining

- Usually large in quantities
- Problems:
  - missing,
  - wrong (miss-recorded)
- Needs transformation to be usable and useful for data mining application

Transformation

- Normalization
  - To a specific range such as [-1,1]
- Decimal Scaling
  - A typical scaling is [-1,1]
  - $V'(i) = V(i) / 10^k$

    for the smallest k such that $|V'(i)| < 1$
  - What is k if the largest value in the data set is 445 and the smallest value is -834 ?

Transformation

- Min-max normalization
  - V’(i) = (V(i) – min(V(i)) / (max(V(i))-min(V(i)))
  - Supose the data is [25,67,150, 200]
    - Min(V(i)) = 25
    - Max(V(i)) = 200
    - 25 → (25 – 25) / (200 – 25) = 0
    - 200 → (200 – 25) / (200 – 25) = 1

Transformation

- Standard Deviation Normalization
  - V’(i) = (V(i) – mean(V)) / sd(V)
  - Supose the data is [1, 2, 3]
    - Mean (V) = 2
    - Sd (V) = 1
    - Transformed data → [-1, 0, 1]

Data Smoothing

- Minor differences sometimes is not significant and will not degrade the performance and result of data mining
- F = { 0.93, 1.01, 1.001, 3.02, 4.98}
- $F_{smoothed}$ = {1.0, 1.0, 1.0, 3.0, 5.0}

Differences and Ratios

- Performances are often measured in differences or ratio rather then the output value.
- Instead of stating v(t+1) as the result, we can show the performance as v(t+1) – v(t) or v(t+1)/vt(t)
- Previous weight=65kg. Current weight=60kg. Performance = (60-65) kg = -5 kg or 60/65*100% = 92%.

Missing Data

- Replace with a single constant
- Replace using feature mean
- Replace using feature mean for the given class
- Replace using the possible combination of values
- X = {1, ?, 3, 4, 5}
  - ? = 0
  - ? = 3
  - ? = 1, 2, 3, 4, 5 → X_1, X_2, X_3, X_4, X_5

Time-dependent Data

- X = {t(1), t(2), t(3), …., t(n)}
- What is t(n+1) ?
- Or sometimes at some time lag, j, t(n+j)
- X= {t(0),t(1),t(2),t(3),t(4),t(5),t(6),t(7),t(8),t(9),t(10)}

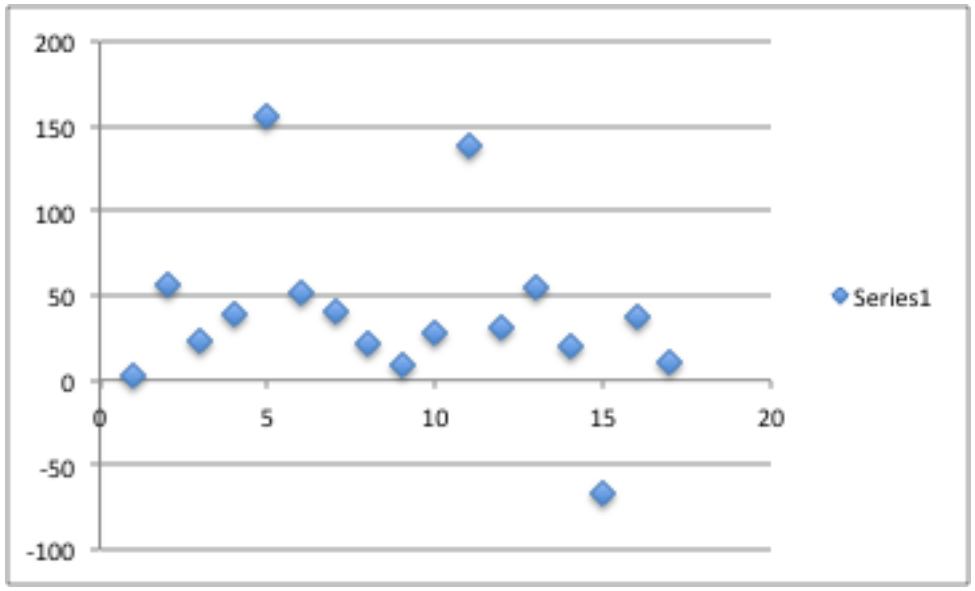
Time-dependent Data

Sample	Window					Next Value
	M1	M2	M3	M4	M5	
1	t(0)	t(1)	t(2)	t(3)	t(4)	t(5)
2	t(1)	t(2)	t(3)	t(4)	t(5)	t(6)
3	t(2)	t(3)	t(4)	t(5)	t(6)	t(7)
4	t(3)	t(4)	t(5)	t(6)	t(7)	t(8)
5	t(4)	t(5)	t(6)	t(7)	t(8)	t(9)
6	t(5)	t(6)	t(7)	t(8)	t(9)	t(10)

Sample	Window					Next Value
	M1	M2	M3	M4	M5	
1	t(0)	t(1)	t(2)	t(3)	t(4)	t(7)
2	t(1)	t(2)	t(3)	t(4)	t(5)	t(8)
3	t(2)	t(3)	t(4)	t(5)	t(6)	t(9)
4	t(3)	t(4)	t(5)	t(6)	t(7)	t(10)

Outlier Analysis

- Age = {3, 56, 23, 39, 156, 52, 41, 22, 9, 28, 139, 31, 55, 20, -67, 37, 11, 55, 45, 37}



Mean = 39.9
SD = 45.65
Threshold = Mean $\pm$ 2 * SD

Threshold = [-54.1, 131.2]

Using Python Pandas

- mydata.csv

```
BAD,LOAN,MORTDUE,VALUE,SALARY
1,1100,25860,39025,94.366666667
1,1300,70053,68400,121.83333333
1,1500,13500,16700,149.46666667
0,1700,97800,112000,93.333333333
1,1700,30548,40320,37.113613558
1,1800,48649,57037,77.1
1,1800,28502,43034,88.766029879
1,2000,32700,46740,216.93333333
1,2000,,62250,115.8,45123
0,2000,64536,87400,147.13333333
1,2300,28192,40150,54.6
```

Has extra column

Using Python Pandas

- Remove column name from mydata.csv and supply column names when opening the file in pandas

```
1,1100,25860,39025,94.366666667
1,1300,70053,68400,121.83333333
1,1500,13500,16700,149.46666667
0,1700,97800,112000,93.333333333
1,1700,30548,40320,37.113613558
1,1800,48649,57037,77.1
1,1800,28502,43034,88.766029879
1,2000,32700,46740,216.93333333
1,2000,,62250,115.8,45123
0,2000,64536,87400,147.13333333
1,2300,28192,40150,54.6
```

Using Python Pandas

- import pandas as pd
- Cols = ['BAD','LOAN','MORTDUE','VALUE', 'SALARY',‘DUMMY‘]
- data = pd.read_csv(“mydata.csv“, names=cols)
- data

	BAD	LOAN	MORTDUE	VALUE	SALARY	DUMMY
0	1	1100	25860.0	39025	94.366667	NaN
1	1	1300	70053.0	68400	121.833333	NaN
2	1	1500	13500.0	16700	149.466667	NaN
3	0	1700	97800.0	112000	93.333333	NaN
4	1	1700	30548.0	40320	37.113614	NaN
5	1	1800	48649.0	57037	77.100000	NaN
6	1	1800	28502.0	43034	88.766030	NaN
7	1	2000	32700.0	46740	216.933333	NaN
8	1	2000	NaN	62250	115.800000	45123.0
9	0	2000	64536.0	87400	147.133333	NaN
10	1	2300	28192.0	40150	54.600000	NaN

Using Python Pandas

- Some pandas operation
  - head(), tail(), head(*n*), tail(*n*), info(), shape, columns
  - Delete columns with name=DUMMY
  - data = data.drop('DUMMY', axis=1)

Using Python Pandas

- Some pandas operation
  - data.isnull()

	BAD	LOAN	MORTDUE	VALUE	SALARY
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	False	False
5	False	False	False	False	False
6	False	False	False	False	False
7	False	False	False	False	False
8	False	False	True	False	False
9	False	False	False	False	False
10	False	False	False	False	False

Using Python Pandas

- Some pandas operation
  - data = data.dropna()

	BAD	LOAN	MORTDUE	VALUE	SALARY
0	1	1100	25860.0	39025	94.366667
1	1	1300	70053.0	68400	121.833333
2	1	1500	13500.0	16700	149.466667
3	0	1700	97800.0	112000	93.333333
4	1	1700	30548.0	40320	37.113614
5	1	1800	48649.0	57037	77.100000
6	1	1800	28502.0	43034	88.766030
7	1	2000	32700.0	46740	216.933333
9	0	2000	64536.0	87400	147.133333
10	1	2300	28192.0	40150	54.600000

Drop rows with condition?
Replace column contents?

Using Python Pandas

- Some pandas operation
  - data.describe()

	BAD	LOAN	MORTDUE	VALUE	SALARY
count	11.000000	11.000000	10.000000	11.000000	11.000000
mean	0.818182	1745.454545	44034.000000	55732.363636	108.767846
std	0.404520	344.568241	25979.830891	26265.300738	49.965251
min	0.000000	1100.000000	13500.000000	16700.000000	37.113614
25%	1.000000	1600.000000	28269.500000	40235.000000	82.933015
50%	1.000000	1800.000000	31624.000000	46740.000000	94.366667
75%	1.000000	2000.000000	60564.250000	65325.000000	134.483333
max	1.000000	2300.000000	97800.000000	112000.000000	216.933333

- Save file
  - Data.to_csv(‘mynewdata.csv', index=None, header=True)