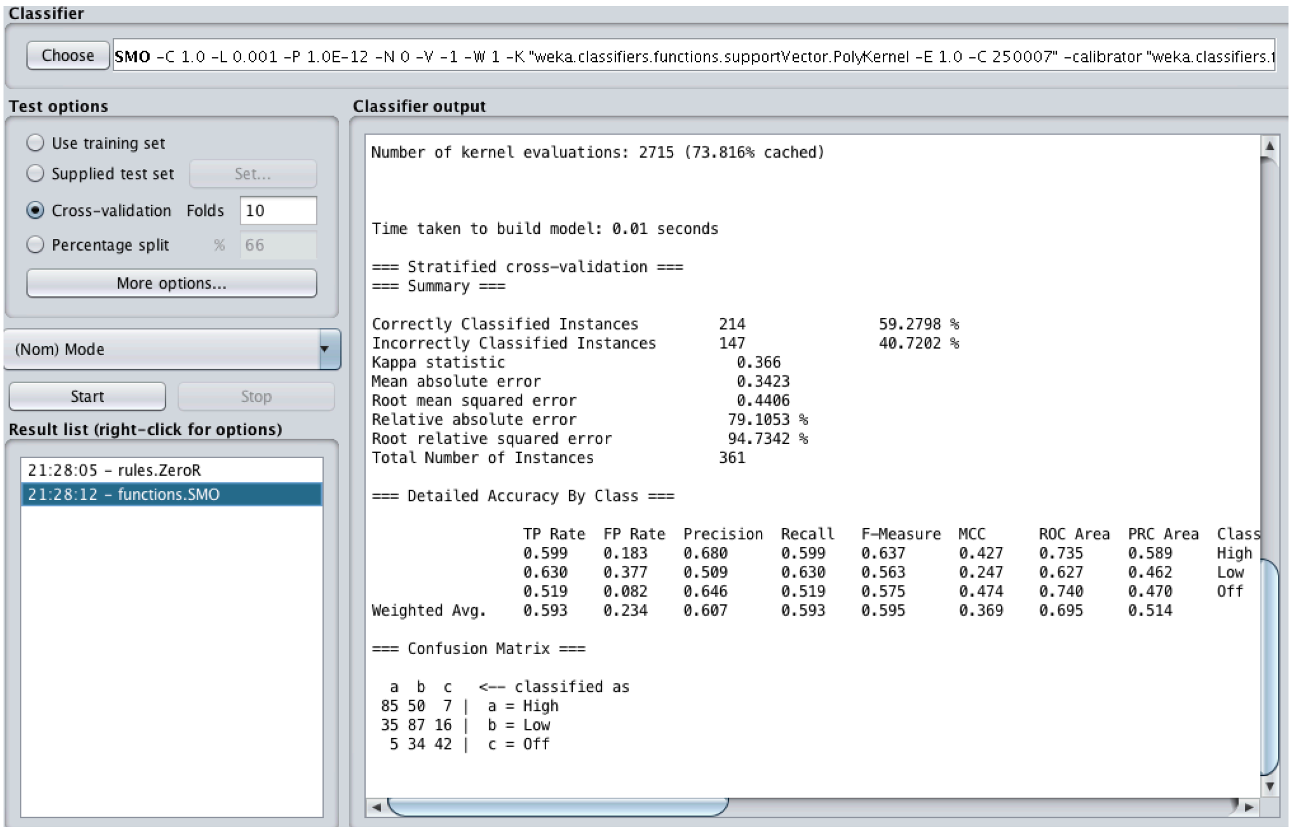
COMP7103 Data mining
Assignment2

Author: Li Xinwei (3035454949)

Question1 Classification

a) What is your **final model** and the corresponding evaluation result?

The final model I used was SVM, the accuracy was about 59% as shown below.



Classifier

Choose: SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.f

Test options

- Use training set
- Supplied test set
- [✓ Cross-validation] Folds 10
- Percentage split % 66

(Nom) Mode

Result list (right-click for options)

- 21:28:05 - rules.ZeroR
- 21:28:12 - functions.SMO

Classifier output

```
Number of kernel evaluations: 2715 (73.816% cached)



Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         214               59.2798 %
Incorrectly Classified Instances       147               40.7202 %
Kappa statistic                          0.366
Mean absolute error                      0.3423
Root mean squared error                  0.4406
Relative absolute error                 79.1053 %
Root relative squared error             94.7342 %
Total Number of Instances              361

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.599    0.183    0.680      0.599    0.637      0.427    0.735     0.589     High
                 0.630    0.377    0.509      0.630    0.563      0.247    0.627     0.462     Low
                 0.519    0.082    0.646      0.519    0.575      0.474    0.740     0.470     Off
Weighted Avg.    0.593    0.234    0.607      0.593    0.595      0.369    0.695     0.514

=== Confusion Matrix ===

  a  b  c   <-- classified as
 85 50  7 |  a = High
 35 87 16 |  b = Low
  5 34 42 |  c = Off
```

Graph1. SVM result

b) Briefly describe **what you have done** in finding the model, including but not limited to data preprocessing, attribute selection, parameter tuning, training and testing data construction, model building, and evaluation etc. You may omit some of the above mentioned items if you have not done that. You **should highlight and explain the choice** you have made in the process.

- Data preprocessing:

  Step1. Replace the Trace in the attribute 'Rainfall' to a small value: 0.01

  Step2. Fill the missing value with Python, using an existing function from pandas library, which is able to find the missing value automatically and fills it with the former value:

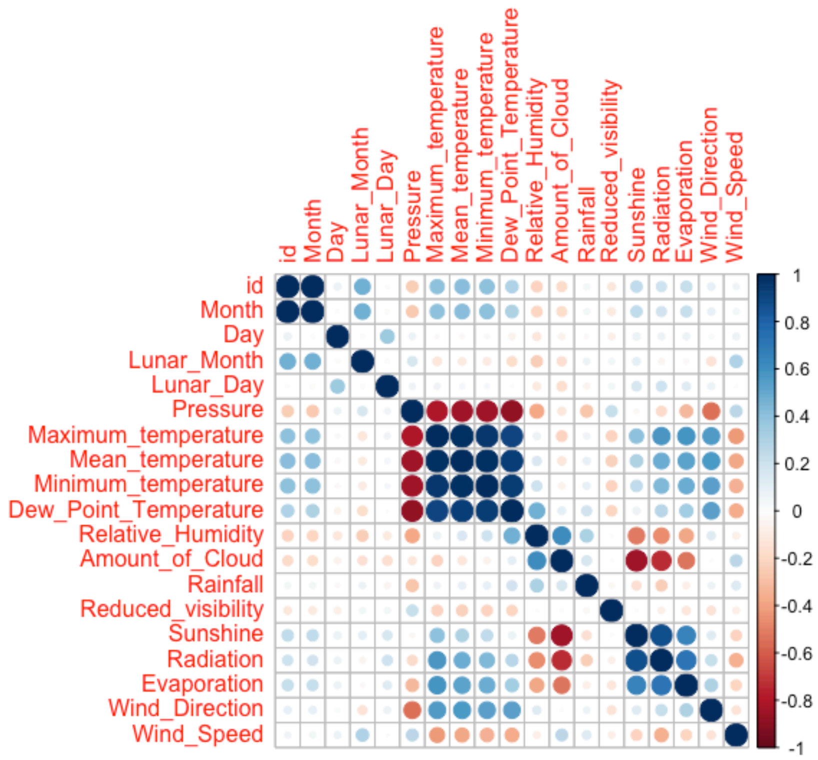
  pd.DataFrame.fillna

- Attribute selection:

  Step3. Output the preprocessed data to a new CSV file, using R to find the correlation matrix of each attributes. It is obvious that the correlations between these four kinds of temperature are high, hence, I leave mean temperature and remove the others from the dataset.

- Labeling

  Step4. Using the given function from the instruction to calculate the energy production of each mode, then label each row of data with the highest energy production mode.



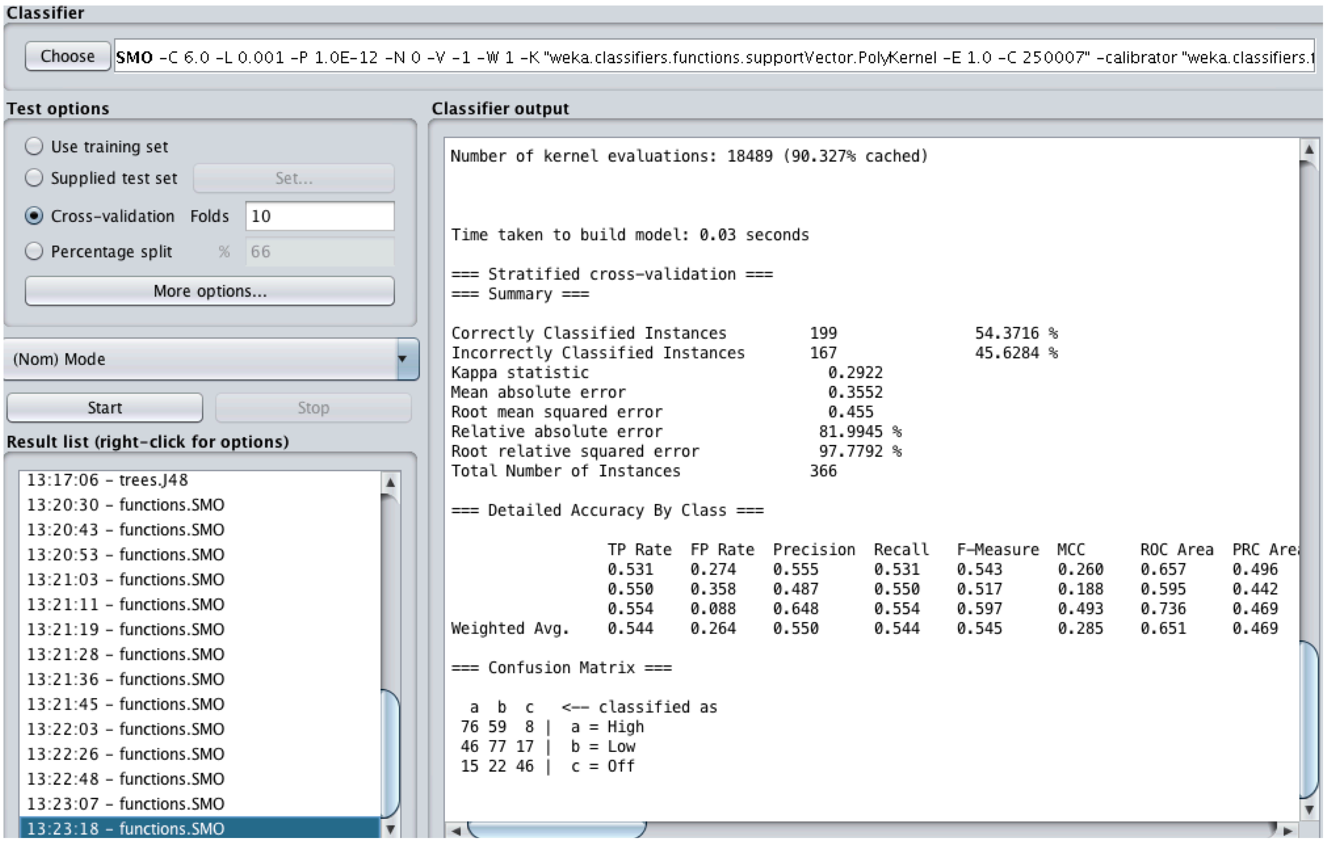
Graph2. Correlation matrix

- Training data generation

Using Python to merge data in precious five days (for the first 5 days, they used the data from corresponding previous days). I tried three ways of merging previous data to one row.

1) Using average data

As shown in the file: training_data.csv, for each row, which contains the average value of the previous days. However, from observation, the error between it and the original data was large. Then I tried to put it into Weka to train. The result was not good, even for the best result in SVM, which was about 54% accuracy. (c was 6.0)



```
SMO -C 6.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.f
```

Test options: Use training set [✓ Cross-validation Folds 10] Percentage split

Result list:

```
13:17:06 - trees.J48
13:20:30 - functions.SMO
13:20:43 - functions.SMO
13:20:53 - functions.SMO
13:21:03 - functions.SMO
13:21:11 - functions.SMO
13:21:19 - functions.SMO
13:21:28 - functions.SMO
13:21:36 - functions.SMO
13:21:45 - functions.SMO
13:22:03 - functions.SMO
13:22:26 - functions.SMO
13:22:48 - functions.SMO
13:23:07 - functions.SMO
13:23:18 - functions.SMO
```

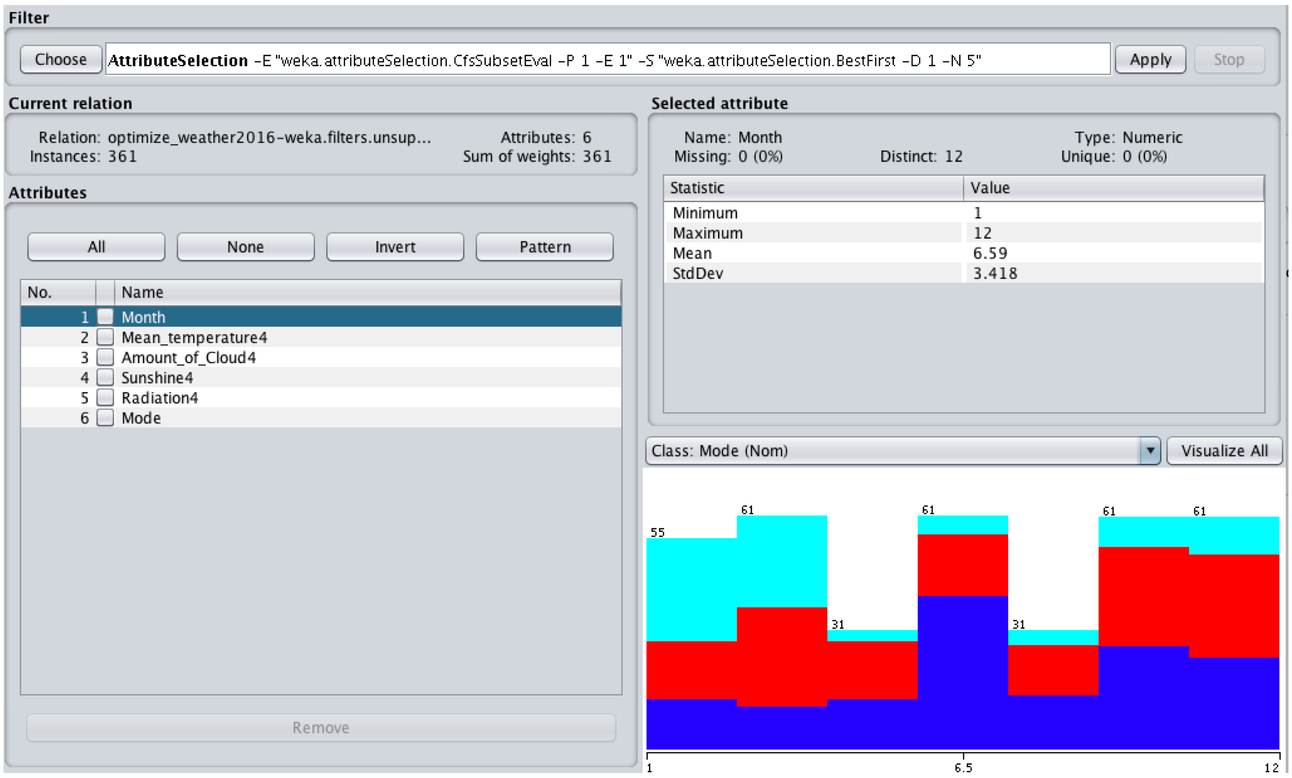
```
Number of kernel evaluations: 18489 (90.327% cached)



Time taken to build model: 0.03 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         199               54.3716 %
Incorrectly Classified Instances       167               45.6284 %
Kappa statistic                          0.2922
Mean absolute error                      0.3552
Root mean squared error                  0.455
Relative absolute error                 81.9945 %
Root relative squared error             97.7792 %
Total Number of Instances              366

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area
                 0.531    0.274    0.555      0.531    0.543      0.260    0.657     0.496
                 0.550    0.358    0.487      0.550    0.517      0.188    0.595     0.442
                 0.554    0.088    0.648      0.554    0.597      0.493    0.736     0.469
Weighted Avg.    0.544    0.264    0.550      0.544    0.545      0.285    0.651     0.469

=== Confusion Matrix ===

  a  b  c   <-- classified as
 76 59  8 |  a = High
 46 77 17 |  b = Low
 15 22 46 |  c = Off
```

Graph3. SVM result with training_data.csv

2) Combining data (Perform best!)

Then, I tried to combining the previous five days data into on row as different features. As shown in the file optimise_weather2016.csv.

In Weka, I used filter to select attributes at the very first time, the selected attributes was only the previous one day's data.



Filter

AttributeSelection -E "weka.attributeSelection.CfsSubsetEval -P 1 -E 1" -S "weka.attributeSelection.BestFirst -D 1 -N 5"

Current relation

Relation: optimize_weather2016-weka.filters.unsup... Attributes: 6
Instances: 361 Sum of weights: 361

Attributes

No.	Name
1	Month
2	Mean_temperature4
3	Amount_of_Cloud4
4	Sunshine4
5	Radiation4
6	Mode

Selected attribute

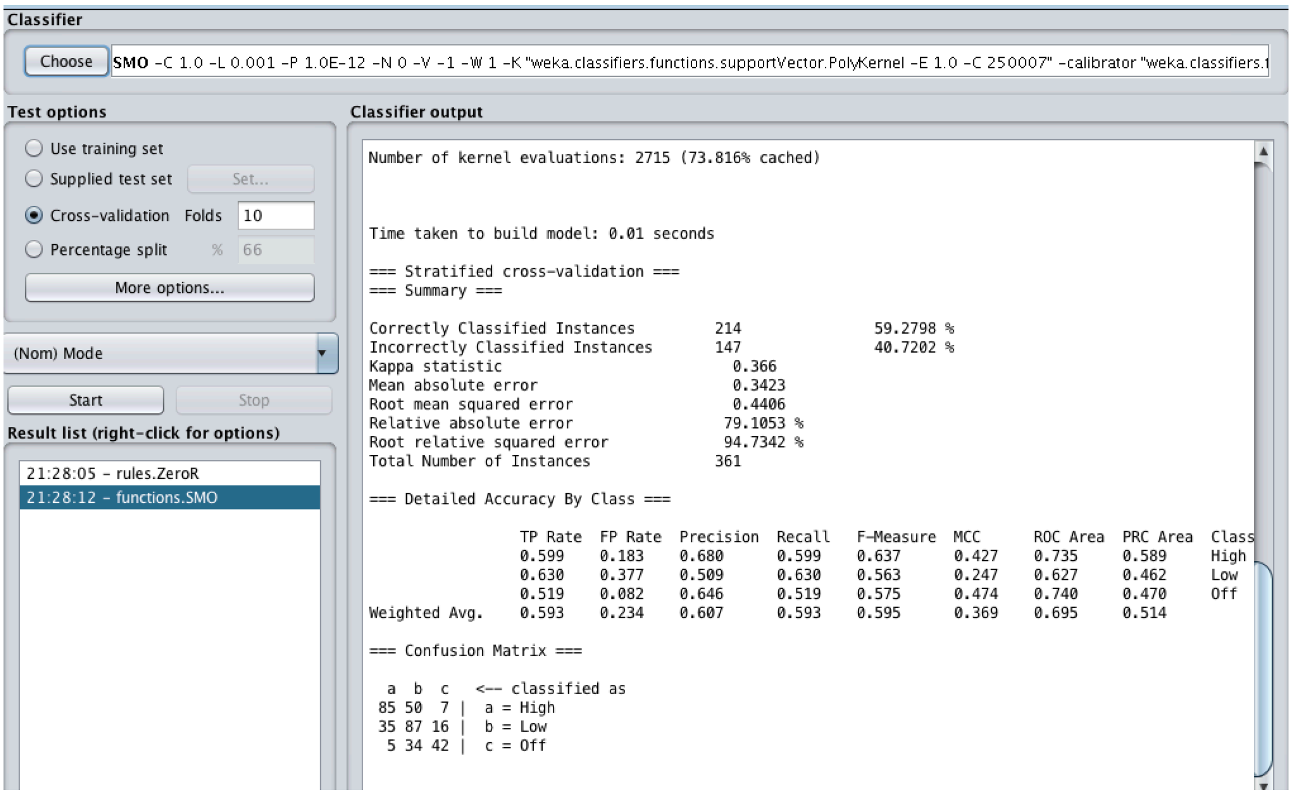
Name: Month Type: Numeric
Missing: 0 (0%) Distinct: 12 Unique: 0 (0%)

Statistic	Value
Minimum	1
Maximum	12
Mean	6.59
StdDev	3.418

Class: Mode (Nom)

Graph4. Attributes selected

Then I used J48 algorithm, ZeroR, Bayes, Logistic and SVM as I did before. Similarly, **SVM** performed the best and the result was about 59% which better than the former training_data.



Classifier

SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator "weka.classifiers.f

Test options

Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66

(Nom) Mode

Result list (right-click for options)

21:28:05 - rules.ZeroR
21:28:12 - functions.SMO

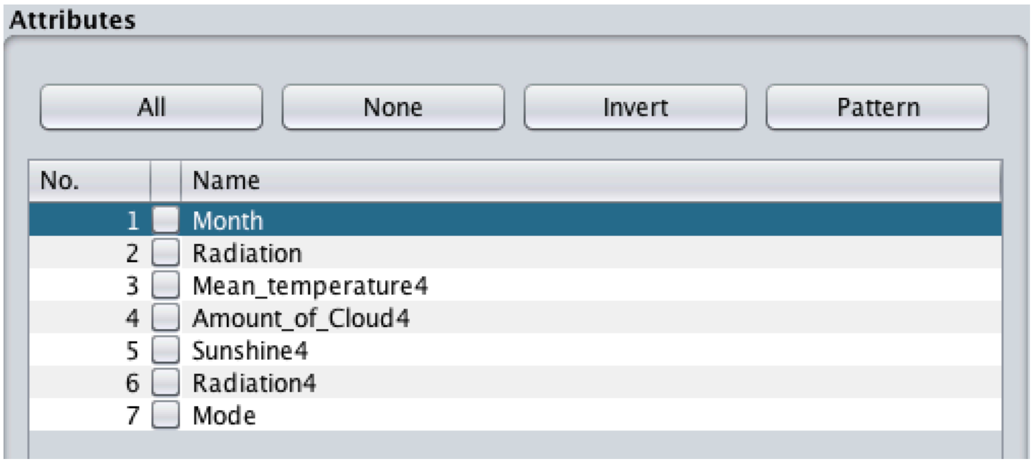
Classifier output

```
Number of kernel evaluations: 2715 (73.816% cached)



Time taken to build model: 0.01 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances         214               59.2798 %
Incorrectly Classified Instances       147               40.7202 %
Kappa statistic                          0.366
Mean absolute error                      0.3423
Root mean squared error                  0.4406
Relative absolute error                 79.1053 %
Root relative squared error             94.7342 %
Total Number of Instances              361

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 0.599    0.183    0.680      0.599    0.637      0.427    0.735     0.589     High
                 0.630    0.377    0.509      0.630    0.563      0.247    0.627     0.462     Low
                 0.519    0.082    0.646      0.519    0.575      0.474    0.740     0.470     Off
Weighted Avg.    0.593    0.234    0.607      0.593    0.595      0.369    0.695     0.514

=== Confusion Matrix ===

  a  b  c   <-- classified as
 85 50  7 |  a = High
 35 87 16 |  b = Low
  5 34 42 |  c = Off
```

Graph5.SVM result with optimisation

For SVM, I tried multiply kernel. PolyKernel performed well among the others.

3) Last try
At the end, I tried to use both average value and all the five days data as the training data, hence, I merged them into one same file: struggle_oneLestTime.csv.
The selected attributes were similar as before:



Attributes

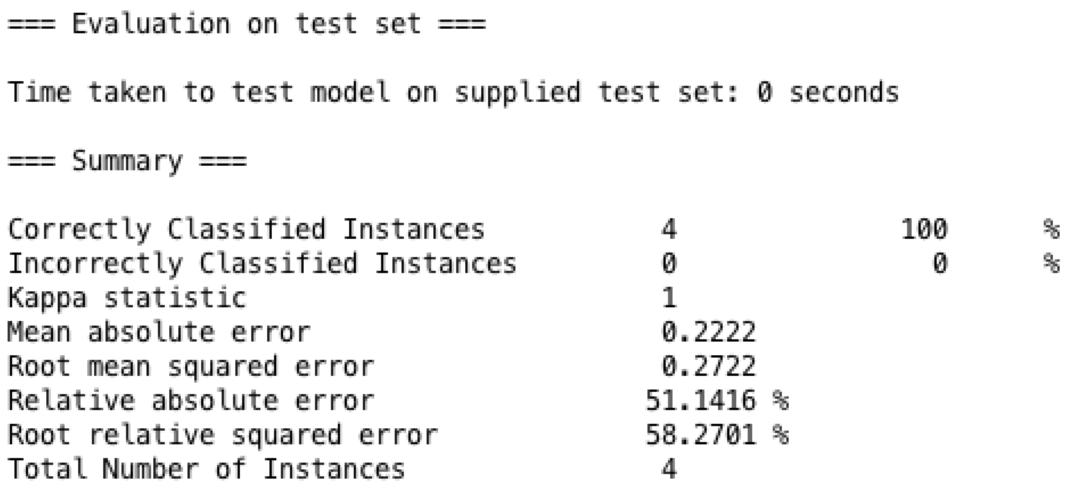
No.		Name
1	☐	Month
2	☐	Radiation
3	☐	Mean_temperature4
4	☐	Amount_of_Cloud4
5	☐	Sunshine4
6	☐	Radiation4
7	☐	Mode

Graph6. Attributes selected

However, the best prediction accuracy was slightly lower than above, which was about 57%.

b) Given a data set, **weather2017.csv** (available on Moodle), which contains the **weather data** of Hong Kong in the year of 2017. Predict the best mode of operation on the dates of 2017-02-10, 2017-05-30, 2017-08-28, and 2017-11-18 using the final model in part a. Show the result of predictions and the actual best mode of operations on these days. Comment on the predictability of your model based on the result.

The prediction accuracy was 100% with using the best performance model.
Since the test data amount was small, it is hard to tell the predictability of my model. However, according to last question and the prediction result from 2017 weather, the predictability of my model is decent.



```
=== Evaluation on test set ===

Time taken to test model on supplied test set: 0 seconds

=== Summary ===

Correctly Classified Instances           4              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.2222
Root mean squared error                  0.2722
Relative absolute error                 51.1416 %
Root relative squared error             58.2701 %
Total Number of Instances                4
```

Graph7. Test result with 2017 data

Date	Predicted mode	Actual mode
2017-02-10	Low	Low
2017-05-30	High	High
2017-08-28	Off	Off
2017-11-18	High	High

Table1. Prediction and actual result comparison

c) Using the final model in part a, **predict** the best mode of operation on the date of 2018- 01-01, using the weather data in **weather2017.csv**. **Show clearly** how a prediction can be manually generated from the model. If it is impossible to use the model manually, where tools or utilities (e.g., Weka) must be employed to generate the prediction, **explain** instead what these tools or utilities do to generate a prediction.

The process of predict the best mode of operation on the date of 2018-01-01 is as follow:

1. Generate training data from 2017

As I mentioned before, combining data from previous five days as a row and put it in to the new row of the date we are going to predicted.

2. Training the model by using Weka
3. Predict the mode in 2018-01-01

Generate the previous data from 2018-01-01 as the same format as the training data, which **would work**.

Explanation

Use weather2017 as training data to predict the best mode of operation on the date of 2018-01-01 is fine because the dataset of weather2017 contains the data of the last five days of 2017, which are exactly the previous five days of 2018. On the other hand, for the best performance mode, the previous one day's data was the most important to the model.

Hence, The model trained from 2017data would work in this case.

Question2

a) List all the 4-element subsequences contained in the data sequence of Sensor S1.

```
<{B, D}, {C}, {A}>
<{B, D}, {C}, {B}>
<{B, D}, {A, B}>
<{B, D}, {B}, {A}>
<{B, D}, {A}, {A}>
<{B}, {C}, {A, B}>
<{B}, {C}, {A}, {A}>
<{B}, {C}, {A}, {B}>
<{D}, {C}, {A, B}>
<{D}, {C}, {A}, {A}>
<{D}, {C}, {A}, {B}>
<{C}, {A, B}, {A}>
```

b) Find all frequent subsequences with support ≥ 60% given the sequence database shown in Table 2 using GSP algorithm as discussed in Lecture Notes Chapter 7, p.65. Show the result of Candidate Generation, Candidate Pruning, and Support Counting, as well as the frequent sequences found after each pass.

Step1. Candidate generation

- candidate 1-subsequence

```
[['A']], [['B']], [['C']], [[‘D']]
```

- candidate 2-subsequence

```
[[['A', 'B']],
 [['A', 'C']],
 [['A', 'D']],
 [['B', 'C']],
 [['B', 'D']],
 [['C', 'D']],
[['A'], ['a']],
[['a'], ['B']],
[['a'], ['C']],
[['A'], ['D']],
[['B'], ['A']],
[['B'], ['B']],
[['B'], ['C']],
[['B'], ['D']],
[['C'], ['A']],
[['C'], ['B']],
[['C'], ['C']],
[['C'], ['D']],
[['D'], ['A']],
[['D'], ['B']],
[['D'], ['C']],
[['D'], ['D']]]
```

- candidate 3-subsequence

```
[[['A', 'B'], ['C'], ['A']],
 [['A', 'B'], ['C'], ['B']],
 [['B'], ['A', 'B'], ['C']],
[['B'], ['C'], ['A'], ['C']],
 [['B'], ['C'], ['A', 'B']],
[['B'], ['C'], ['B'], ['C']],
 [['C'], ['A', 'B'], ['C']],
[['C'], ['B'], ['C'], ['A']],
[['C'], ['B'], ['C'], ['B']]]
```

- candidate 4-subsequence

```
[[['B'], ['C'], ['A', 'B'], ['C']]]
```

Step2. Candidate pruning

- candidate 2-subsequence pruning

```
[[['A', 'B']],
 [['A', 'C']],
 [['A', 'D']],
 [['B', 'C']],
 [['B', 'D']],
 [['C', 'D']],
[['A'], ['A']],
[['A'], ['B']],
[['A'], ['C']],
[['A'], ['D']],
[['B'], ['A']],
[['B'], ['B']],
[['B'], ['C']],
[['B'], ['D']],
[['C'], ['A']],
[['C'], ['B']],
[['C'], ['C']],
[['C'], ['D']],
[['D'], ['A']],
[['D'], ['B']],
[['D'], ['C']],
[['D'], [‘D']]]
```

- candidate 3-subsequence pruning

```
[[['A'], ['A'], ['A']],
 [['A'], ['A'], ['C']],
  [['A'], ['A', 'C']],
 [['A'], ['C'], ['A']],
 [['A'], ['C'], ['C']],
  [['A', 'B'], ['A']],
  [['A', 'B'], ['C']],
  [['A', 'C'], ['A']],
  [['A', 'C'], ['C']],
 [['B'], ['A'], ['A']],
 [['B'], ['A'], ['C']],
  [['B'], ['A', 'B']],
  [['B'], ['A', 'C']],
 [['B'], ['B'], ['A']],
 [['B'], ['B'], ['B']],
 [['B'], ['B'], ['C']],
 [['B'], ['C'], ['A']],
 [['B'], ['C'], ['B']],
 [['B'], ['C'], ['C']],
 [['C'], ['A'], ['A']],
 [['C'], ['A'], ['C']],
  [['C'], ['A', 'B']],
[['C'], ['A', 'C']],
[['C'], ['B'], ['A']],
[['C'], ['B'], ['B']],
[['C'], ['B'], ['C']],
[['C'], ['C'], ['A']],
[['C'], ['C'], ['B']],
[['C'], ['C'], ['C']],
[['D'], ['A'], [‘A’]]]
```

- candidate 3-subsequence pruning

```
[[['B'], ['C'], ['A', 'B']], [['C'], ['A', 'B'], [‘C’]]]
```

- candidate 4-subsequence pruning

```
[[['B'], ['C'], ['A', 'B']], [['C'], ['A', 'B'], [‘C']]]
```

Step3. Supporting counting

- 2 - subsequence counting

```
[([['A', 'B']], 5),
 ([['A', 'C']], 3),
([['A'], ['A']], 3),
([['A'], ['C']], 3),
([['B'], ['A']], 4),
([['B'], ['B']], 3),
([['B'], ['C']], 5),
([['C'], ['A']], 5),
([['C'], ['B']], 5),
([['C'], ['C']], 3),
([['D'], ['A']], 3)]
```

- 3 - subsequence counting

```
[([['A', 'B'], ['C']], 3),
 ([['B'], ['A', 'B']], 3),
([['B'], ['C'], ['A']], 3),
([['B'], ['C'], ['B']], 3),
([['C'], ['A'], ['C']], 3),
 ([['C'], ['A', 'B']], 5),
([['C'], ['B'], ['C']], 3)]
```

- 4 - subsequence counting

```
[([['B'], ['C'], ['A', 'B']], 3), ([['C'], ['A', 'B'], ['C']], 3)]
```

Step4. All frequent sequences found in each pass

- First pass

```
[[['A', 'B']],
 [['A', 'C']],
 [['A', 'D']],
[['B', 'C']],
[['B', 'D']],
[['C', 'D']],
[['A'], ['A']],
[['A'], ['B']],
[['A'], ['C']],
[['A'], ['D']],
[['B'], ['A']],
[['B'], ['B']],
[['B'], ['C']],
[['B'], ['D']],
[['C'], ['A']],
[['C'], ['B']],
[['C'], ['C']],
[['C'], ['D']],
[['D'], ['A']],
[['D'], ['B']],
[['D'], ['C']],
[['D'], [‘D']]]
```

- Second pass

```
[[['A'], ['A'], ['A']],
 [['A'], ['A'], ['C']],
 [['A'], ['A', 'B']],
 [['A'], ['A', 'C']],
 [['A'], ['C'], ['A']],
 [['A'], ['C'], ['B']],
 [['A'], ['C'], ['C']],
 [['A', 'B'], ['A']],
 [['A', 'B'], ['B']],
 [['A', 'B'], ['C']],
 [['A', 'C'], ['A']],
 [['A', 'C'], ['B']],
 [['A', 'C'], ['C']],
 [['B'], ['A'], ['A']],
 [['B'], ['A'], ['C']],
 [['B'], ['A', 'B']],
 [['B'], ['A', 'C']],
 [['B'], ['B'], ['A']],
 [['B'], ['B'], ['B']],
 [['B'], ['B'], ['C']],
 [['B'], ['C'], ['A']],
 [['B'], ['C'], ['B']],
 [['B'], ['C'], ['C']],
 [['C'], ['A'], ['A']],
 [['C'], ['A'], ['C']],
 [['C'], ['A', 'B']],
 [['C'], ['A', 'C']],
[['C'], ['B'], ['A']],
[['C'], ['B'], ['B']],
[['C'], ['B'], ['C']],
[['C'], ['C'], ['A']],
[['C'], ['C'], ['B']],
[['C'], ['C'], ['C']],
[['D'], ['A'], ['A']],
[['D'], ['A'], ['C']],
 [['D'], ['A', 'B']],
 [['D'], ['A', 'C']]]
```

- Third pass

```
[[['A', 'B'], ['C'], ['A']],
 [['A', 'B'], ['C'], ['B']],
 [['B'], ['A', 'B'], ['C']],
[['B'], ['C'], ['A'], ['C']],
 [['B'], ['C'], ['A', 'B']],
[['B'], ['C'], ['B'], ['C']],
 [['C'], ['A', 'B'], ['C']],
[['C'], ['B'], ['C'], ['A']],
[['C'], ['B'], ['C'], ['B']]]
```

- Forth pass

```
[[['B'], ['C'], ['A', 'B'], ['C']]]
```

Reference (code I wrote for data preprocessing)

1. Preprocessing code

```
import pandas as pd
import csv

train = pd.read_csv("weather2016.csv", encoding='UTF-8')

# fill missing value
train.fillna(method='pad', inplace=True)
train.to_csv("no_missing_data.csv", encoding='utf-8', index=False)

# drop the redundance attributes according to correlation matrix plot in
R
train.drop('Maximum_temperature',  axis=1, inplace=True)
train.drop('Minimum_temperature',  axis=1, inplace=True)
train.drop('Dew_Point_Temperature',  axis=1, inplace=True)


# add the previous days average to new data frame
def appendnewfeature(index, train):
    new_row = []
```

```python
pressure_sum = 0
temperature_sum = 0
humidity_sum = 0
cloud_sum = 0
rainfall_sum = 0
visibility_sum = 0
sunshine_sum = 0
radiation_sum = 0
evaporation_sum = 0
wind_sum = 0
speed_sum = 0

new_row.append(train.iloc[index].id)
new_row.append(train.iloc[index].Month)
new_row.append(train.iloc[index].Day)
new_row.append(train.iloc[index].Lunar_Month)
new_row.append(train.iloc[index].Lunar_Day)

if index<5:
    if index == 0:
        pressure_sum = train.iloc[0].Pressure
        temperature_sum = train.iloc[0].Mean_temperature
        humidity_sum = train.iloc[0].Relative_Humidity
        cloud_sum = train.iloc[0].Amount_of_Cloud
        rainfall_sum = train.iloc[0].Rainfall
        visibility_sum = train.iloc[0].Reduced_visibility
        sunshine_sum = train.iloc[0].Sunshine
        radiation_sum = train.iloc[0].Radiation
        evaporation_sum = train.iloc[0].Evaporation
        wind_sum = train.iloc[0].Wind_Direction
        speed_sum = train.iloc[0].Wind_Speed

        pressure_new = pressure_sum
        temperature_new = temperature_sum
        humidity_new = humidity_sum
        cloud_new = cloud_sum
        rainfall_new = rainfall_sum
        visibility_new = visibility_sum
        sunshine_new = sunshine_sum
        radiation_new = radiation_sum
        evaporation_new = evaporation_sum
        wind_new = wind_sum
        speed_new = speed_sum

    else:
        for j in range(0, index):
            pressure_sum += train.iloc[j].Pressure
            temperature_sum += train.iloc[j].Mean_temperature
            humidity_sum += train.iloc[j].Relative_Humidity
            cloud_sum += train.iloc[j].Amount_of_Cloud
            rainfall_sum += train.iloc[j].Rainfall
            visibility_sum += train.iloc[j].Reduced_visibility
```

```
                sunshine_sum += train.iloc[j].Sunshine
                radiation_sum += train.iloc[j].Radiation
                evaporation_sum += train.iloc[j].Evaporation
                wind_sum += train.iloc[j].Wind_Direction
                speed_sum += train.iloc[j].Wind_Speed
                j += 1

            pressure_new = pressure_sum/(index)
            temperature_new = temperature_sum/(index)
            humidity_new = humidity_sum/(index)
            cloud_new = cloud_sum/(index)
            rainfall_new = rainfall_sum/(index)
            visibility_new = visibility_sum/(index)
            sunshine_new = sunshine_sum/(index)
            radiation_new = radiation_sum/(index)
            evaporation_new = evaporation_sum/(index)
            wind_new = wind_sum/(index)
            speed_new = speed_sum/(index)

    else:
        for i in range(1, 6):
            pressure_sum += train.iloc[index - i].Pressure
            temperature_sum += train.iloc[index - i].Mean_temperature
            humidity_sum += train.iloc[index - i].Relative_Humidity
            cloud_sum += train.iloc[index - i].Amount_of_Cloud
            rainfall_sum += train.iloc[index - i].Rainfall
            visibility_sum += train.iloc[index - i].Reduced_visibility
            sunshine_sum += train.iloc[index - i].Sunshine
            radiation_sum += train.iloc[index - i].Radiation
            evaporation_sum += train.iloc[index - i].Evaporation
            wind_sum += train.iloc[index - i].Wind_Direction
            speed_sum += train.iloc[index - i].Wind_Speed

        pressure_new = pressure_sum / 5
        temperature_new = temperature_sum / 5
        humidity_new = humidity_sum / 5
        cloud_new = cloud_sum / 5
        rainfall_new = rainfall_sum / 5
        visibility_new = visibility_sum / 5
        sunshine_new = sunshine_sum / 5
        radiation_new = radiation_sum / 5
        evaporation_new = evaporation_sum / 5
        wind_new = wind_sum / 5
        speed_new = speed_sum / 5

    new_row.extend([round(pressure_new,2), round(temperature_new,2),
round(humidity_new,2), round(cloud_new,2), round(rainfall_new,2),
round(visibility_new,2),
                    round(sunshine_new,2), round(radiation_new,2),
round(evaporation_new,2), round(wind_new,2), round(speed_new,2)])

    return new_row
```

```
# Add label to original data
def generatelabel(train):
    mode = []

    for i in range(366):
        e_high = train.iloc[i].Radiation * 0.2 * 60 - 24 * 4
        e_low = train.iloc[i].Radiation * 0.18 * 60 - 24 * 3 -
train.iloc[i].Sunshine
        e_off = 0

        # print(e_high)
        # print(e_low)
        # print(e_off, '\n')

        if(max(e_high, e_low, e_off) == e_high):
            mode.append('High')
        elif(max(e_high, e_low, e_off) == e_low):
            mode.append('Low')
        else:
            mode.append('Off')
    s = pd.Series(mode)
    train['Mode'] = s.values
    train.to_csv("withMode.csv", encoding='utf-8', index=False)


writer=csv.writer(open("new_weather2016.csv",'w'))
attributes = ['id', 'Month','Day', 'Lunar_Month', 'Lunar_Day',
'Pressure', 'Mean_temperature', 'Relative_Humidity',
              'Amount_of_Cloud','Rainfall', 'Reduced_visibility',
'Sunshine', 'Radiation', 'Evaporation', 'Wind_Direction',
              'Wind_Speed']
writer.writerow(attributes)

for i in range(366):
    new_row = appendnewfeature(i, train)
    writer.writerow(new_row)

generatelabel(train)

print('Finish')
```

2. Optimising code

```
import pandas as pd
import csv

train = pd.read_csv("weather2016.csv", encoding='UTF-8')

# fill missing value
train.fillna(method='pad', inplace=True)
train.to_csv("no_missing_data.csv", encoding='utf-8', index=False)
```

```python
# drop the redundance attributes according to correlation matrix plot in
R
train.drop('Maximum_temperature',  axis=1, inplace=True)
train.drop('Minimum_temperature',  axis=1, inplace=True)
train.drop('Dew_Point_Temperature',  axis=1, inplace=True)


# Append five days of rows
def appendnewfeature(index, train):
    new_row = []
    new_row.append(train.iloc[index].id)
    new_row.append(train.iloc[index].Month)
    new_row.append(train.iloc[index].Day)
    new_row.append(train.iloc[index].Lunar_Month)
    new_row.append(train.iloc[index].Lunar_Day)

    i = 5
    while i > 0:
        list = []
        list.extend([train.iloc[index - i].Pressure, train.iloc[index -
i].Mean_temperature, train.iloc[index - i].Relative_Humidity,
                    train.iloc[index - i].Amount_of_Cloud,
train.iloc[index - i].Rainfall, train.iloc[index -
i].Reduced_visibility,
                    train.iloc[index - i].Sunshine, train.iloc[index -
i].Radiation, train.iloc[index - i].Evaporation,
                    train.iloc[index - i].Wind_Direction,
train.iloc[index - i].Wind_Speed])
        new_row.extend(list)
        i -= 1

    # print(new_row)
    return new_row

writer=csv.writer(open("optimize_weather2016.csv",'w'))
attributes = ['id', 'Month','Day', 'Lunar_Month', 'Lunar_Day']

for i in range(5):
    attributes.append("Pressure%d" % i)
    attributes.append("Mean_temperature%d" % i)
    attributes.append("Relative_Humidity%d" % i)
    attributes.append("Amount_of_Cloud%d" % i)
    attributes.append("Rainfall%d" % i)
    attributes.append("Reduced_visibility%d" % i)
    attributes.append("Sunshine%d" % i)
    attributes.append("Radiation%d" % i)
    attributes.append("Evaporation%d" % i)
    attributes.append("Wind_Direction%d" % i)
    attributes.append("Wind_Speed%d" % i)

writer.writerow(attributes)
```

```python
for i in range(366):
    if i < 5:
        new_row = []
        new_row.append(train.iloc[i].id)
        new_row.append(train.iloc[i].Month)
        new_row.append(train.iloc[i].Day)
        new_row.append(train.iloc[i].Lunar_Month)
        new_row.append(train.iloc[i].Lunar_Day)
    else:
        new_row = appendnewfeature(i, train)

    writer.writerow(new_row)

print('Finish')
```