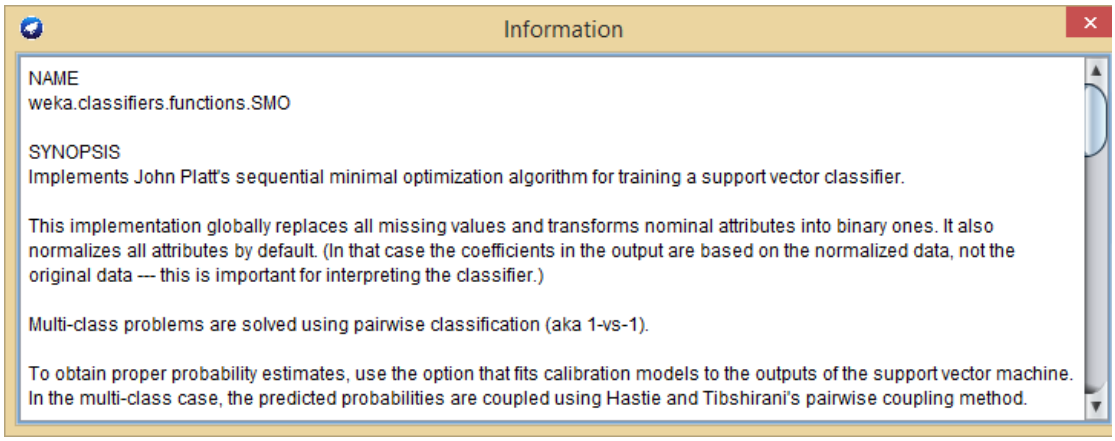
COMP7103 Assignment 2

Lee Sai Kit (3035109489)

Question 1 Classification

Final Model:

weka.classifiers.functions.SMO



Data Pre-processing

Firstly, I insert a line of column headers into CSV file for WEKA to identify the features.

```
Rainfall: Total rainfall of the day, can be "-" (none recorded), "Trace" (very small amount only), or numerical (measured in mm).
Sunshine: Total bright sunshine of the day, can be "-" (none recorded), or numerical (measured in hours).
Evaporation: Total evaporation of the day, measured in mm. Contains missing data, denoted as "N.A." OR "N/A".
```

Then, I replaced the '-' and 'Trace' data in the rainfall and sunshine field with 0, as they actually mean 0 or a value that is close to 0. I also changed the 'N.A.' or 'N/A' data in the Evaporation field with empty string to denote missing data.

```
[Data Pre-processing]
Remove non-numeric values in numeric field
```

```
with open('weather2016_1.csv', 'w', newline='') as outputfile:
    writer = csv.writer(outputfile, delimiter=',')
    with open('weather2016.csv') as inputfile:
        reader = csv.reader(inputfile)
        for row in reader:
            # replace the '-' and 'Trace' in Rainfall with 0
            if row[11] == '-' or row [11] == 'Trace':
                row[11] = '0'
            # replace the '-' in Sunshine with 0
            if row[13] == '-':
                row[13] = '0'

            writer.writerow(row)
```

```
with open('weather2016_2.csv', 'w', newline='') as outputfile:
    writer = csv.writer(outputfile, delimiter=',')
    with open('weather2016_1.csv') as inputfile:
        reader = csv.reader(inputfile)
        prev_row = None
        for row in reader:
            if row[15] == 'N.A.' or row[15] == 'N/A':
                row[15] = ''
            prev_row = row
            writer.writerow(row)
```

After that, I wrote code to calculate the net energy gain under each operation mode and insert the best mode at the end of the data file (as a new column).

```
[Data Preprocessing]
Insert label for the mode of operation
```

```
with open('weather2016_3.csv', 'w', newline='') as outputfile:
    writer = csv.writer(outputfile, delimiter=',')
    with open('weather2016_2.csv') as inputfile:
        reader = csv.reader(inputfile)
        first_row = True
        for row in reader:
            # Add the column header
            if first_row:
                first_row = False
                row.append('Label')
                writer.writerow(row)
                continue

            # Add the label according to e_high and e_low
            label = 'Off'
            e_high = float(row[14]) * 60 * 0.2 - 24 * 4
            e_low = float(row[14]) * 60 * 0.18 - 24 * 3 - float(row[13])
            if e_high > 0 and e_high > e_low:
                label = 'High'
            elif e_low > 0:
                label = 'Low'
            row.append(label)
            writer.writerow(row)
```

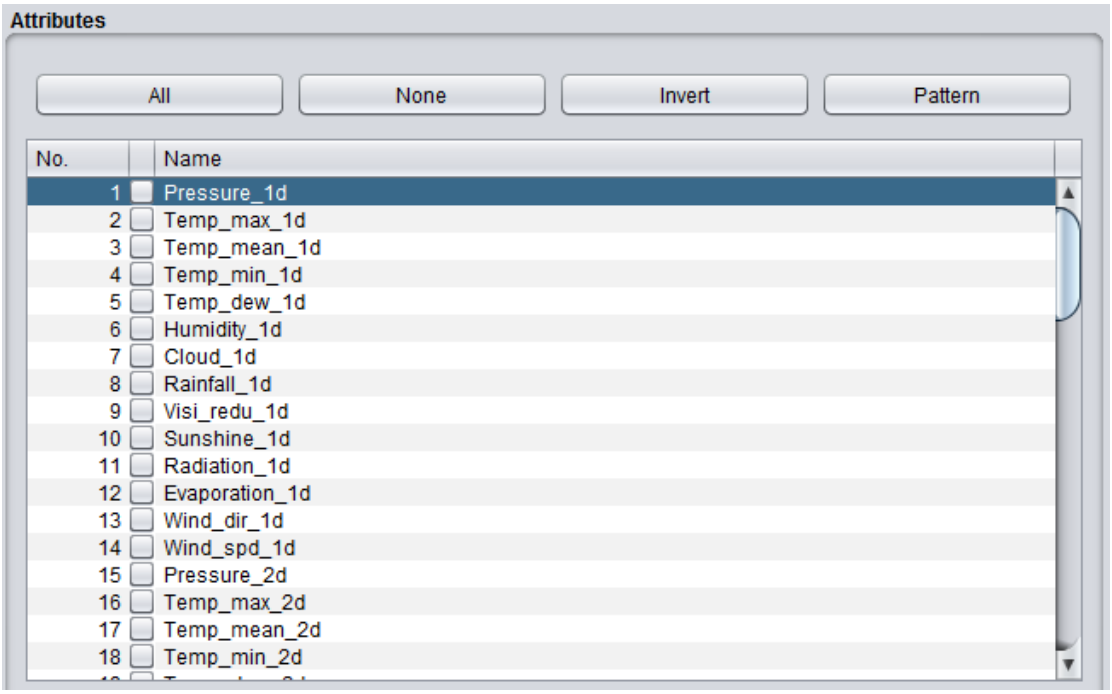
```
Month,Day,L_month,L_day,Pressure,Temp_max,Temp_mean,Temp_min,Temp_dew,Humidity,Cloud,Rainfall,
Visi_redu,Sunshine,Radiation,Evaporation,Wind_dir,Wind_spd,Label
1,1,11,22,1025.9,19.9,18.3,16.8,13.1,72,49,0,12,9.3,15.24,3.1,80,26.3,High
1,2,11,23,1022,21.7,18.9,17.2,15.6,81,83,0.3,20,0.6,7.77,2,50,15.3,Low
1,3,11,24,1019.7,20.3,19.3,18,18.4,95,91,5.6,3,0,4.39,0.6,40,14.2,Off
```

At last, I removed the weather data of each day and insert the weather data of the previous five days instead. The first five days in 2016 would not be in this data set as the weather data of their previous five days are not provided. By using this, I could train classification models by providing the best mode on a day and the weather data on previous five days.

```
num_of_prev_days = 5

with open('weather2017_4.csv', 'w', newline='') as outputfile:
    writer = csv.writer(outputfile, delimiter=',')
    with open('weather2017_3.csv') as inputfile:
        reader = csv.reader(inputfile)
        first_row = True
        headers = gen_n_days_headers(num_of_prev_days)
        prev_rows = [None] * num_of_prev_days

        for row in reader:
            # Add the column header
            if first_row:
                first_row = False
                row = row[0:2] + headers + [row[-1]]
                writer.writerow(row)
                continue

            # generate the output row
            if prev_rows[0] != None:
                output_row = row[0:2] + gen_prev_cols(prev_rows) + [row[-1]]
                writer.writerow(output_row)

            # Update prev_rows
            prev_rows.pop(0)
            prev_rows.append(row)
```

Attribute Selection

In WEKA, I removed the month, day, lunar month and lunar day as they are not weather data, they are not likely to affect the radiation.



Model Selection

Several models (with default configuration) have been tried in WEKA with 10-fold cross validation. Below are the results:

Model	Accuracy	Precision	Recall	F-Measure
SMO	55.4017%	0.555	0.554	0.554
Logistic	48.7535%	0.490	0.488	0.489
SimpleLogistic	50.6925%	0.509	0.507	0.507
MultilayerPerceptron	51.2465%	0.512	0.512	0.512
IBk	41.2742%	0.410	0.413	0.411
KStar	43.2133%	0.428	0.432	0.429
LWL	54.5706%	0.556	0.546	0.544
DecisionTable	51.2465%	0.529	0.512	0.504
JRip	47.6454%	0.472	0.476	0.463
OneR	47.6454%	0.478	0.476	0.477
PART	44.5983%	0.443	0.446	0.444
J48	45.7064%	0.454	0.457	0.455
HoeffdingTree	50.6925%	0.507	0.507	0.503
LMT	50.6925%	0.509	0.507	0.507
RandomForest	51.8006%	0.523	0.518	0.519
RandomTree	40.7202%	0.409	0.407	0.408
REPTree	45.9834%	0.456	0.460	0.454

After evaluating the results above, it was found that SMO has the best performance in all the indicators, so it is chosen as the classification model.

Parameter Tuning

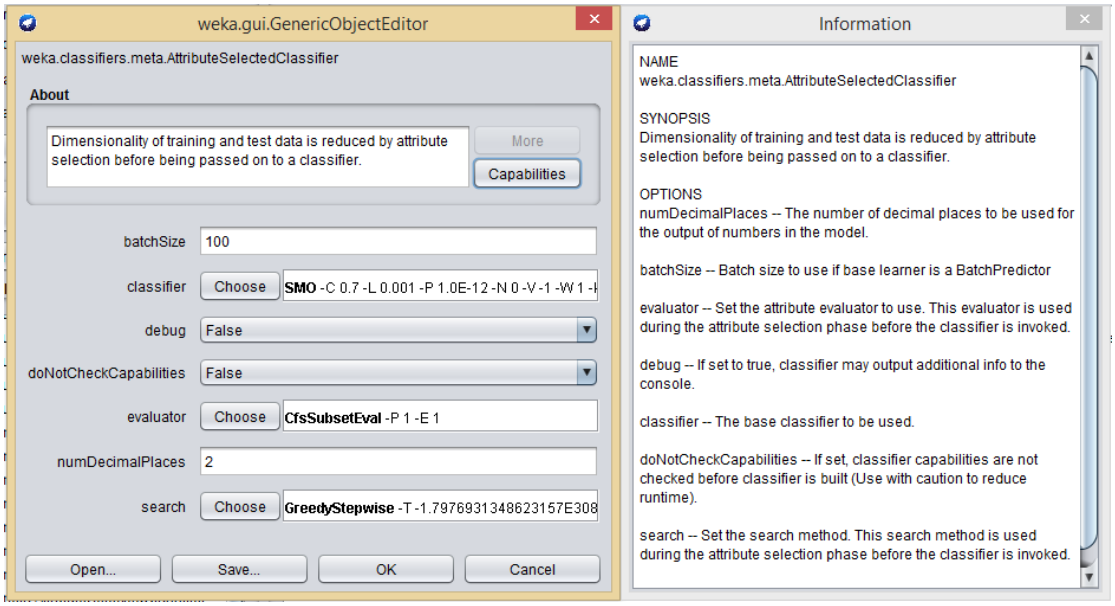
By varying the complexity parameter C of the SMO classifier, accuracy could be improved.

c	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
accuracy	55.40%	55.12%	54.29%	54.29%	55.12%	55.40%	55.96%	54.57%
c	0.9	1.0	1.1	1.2	1.3	1.4	1.5	1.6
accuracy	54.85%	55.40%	55.12%	54.29%	55.40%	55.69%	55.68%	55.68%

From the table above, it was shown that c=0.7 gave the highest accuracy so it was picked in the final model.

Further Attribute Selection (Final Tuning)

To further increase the accuracy, I applied the weka.meta.AttributeSelectedClassifier in WEKA to automatically select attribute and reduce dimensionality.



Several pairs of evaluator and search method have been tried and it was found that the pair of CfsSubsetEval and GreedyStepwise gave the best performance. (accuracy=58.4488%)

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

Correctly Classified Instances         211               58.4488 %
Incorrectly Classified Instances       150               41.5512 %
Kappa statistic                          0.3545
Mean absolute error                      0.3472
Root mean squared error                  0.4461
Relative absolute error                 80.2435 %
Root relative squared error             95.9283 %
Total Number of Instances              361

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.641    0.215    0.659      0.641    0.650      0.428    0.723     0.581     High
               0.565    0.341    0.506      0.565    0.534      0.220    0.607     0.451     Low
               0.519    0.096    0.609      0.519    0.560      0.448    0.742     0.456     Off
Weighted Avg.  0.584    0.236    0.590      0.584    0.586      0.353    0.683     0.503
```

Prediction

The model built from the previous steps was applied on the dates of 2017-02-10, 2017-05-30, 2017-08-28, and 2017-11-18 to predict the best mode of operation.

```
=== Predictions on test set ===

    inst#     actual  predicted error prediction
        1      2:Low     1:High   +    0.667
        2     1:High     1:High        0.667
        3      3:Off      2:Low   +    0.667
        4      2:Low     1:High   +    0.667
```

The prediction result was not satisfactory. 3 out of the 4 predictions were not correct.

Some possible reasons for the inaccuracy were:

- not enough instances in the testing sample to get a meaningful accuracy rate
- normalization issue in the testing sample
- the model overfit to the training data

Prediction for 2018-1-1

To generate a prediction, firstly the weather data of the given data should be normalized. Then, a probability would be calculated by each of the binary classifier (High-Low, High-Off, Low-Off). The probabilities would be compared to choose the final classification.

```
Classifier for classes: High, Low

BinarySMO

Machine linear: showing attribute weights, not support vectors.

       -0.2231 * (normalized) Cloud_4d
 +      0.9281 * (normalized) Temp_max_5d
 +     -0.8784 * (normalized) Temp_min_5d
 +      1.408  * (normalized) Cloud_5d
 +     -1.6964 * (normalized) Sunshine_5d
 +     -0.4444 * (normalized) Radiation_5d
 +      0.1806

Number of kernel evaluations: 4999 (68.5% cached)
```

```
Classifier for classes: High, Off

BinarySMO

Machine linear: showing attribute weights, not support vectors.

       -0.1098 * (normalized) Cloud_4d
 +     -0.5339 * (normalized) Temp_max_5d
 +     -1.3251 * (normalized) Temp_min_5d
 +      0.823  * (normalized) Cloud_5d
 +     -1.2268 * (normalized) Sunshine_5d
 +     -1.4701 * (normalized) Radiation_5d
 +      1.367

Number of kernel evaluations: 2291 (62.571% cached)
```

```
Classifier for classes: Low, Off

BinarySMO

Machine linear: showing attribute weights, not support vectors.

        0.5223 * (normalized) Cloud_4d
 +     -1.813  * (normalized) Temp_max_5d
 +     -1.1533 * (normalized) Temp_min_5d
 +      0.7547 * (normalized) Cloud_5d
 +     -0.3051 * (normalized) Sunshine_5d
 +     -1.5874 * (normalized) Radiation_5d
 +      0.942

Number of kernel evaluations: 2120 (69.342% cached)
```

Since the above work could be ambiguous and tedious if done manually, it was preferred to use WEKA to do the work and generate the prediction result.

```
=== Predictions on test set ===

    inst#     actual  predicted error prediction
        1        1:?     1:High       0.667
```

The best mode for 2018-1-1 was predicted to be High by the model.

Question 2 Association Analysis

Sensor	Timestamp	Events
S1	1	B, D
	2	C
	3	A, B
	4	A
S2	1	A, C
	2	A, B
	3	C

Sensor	Timestamp	Events
S3	1	B
	2	C, D
	3	A, B
S4	1	B
	2	C
	3	A, B, C
	4	C

Sensor	Timestamp	Events
S5	1	C
	2	A, B
	3	D
	4	A, C, D

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

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

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

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

Generalized Sequential Pattern

Candidate Generation & Support Count for 1-sequence

Sequence	<{A}>	<{B}>	<{C}>	<{D}>
Support	5	5	5	3

No candidate sequences are pruned as all of them have support >= 60%.

Candidate Generation for 2-sequence

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

(repeated sequences are omitted in the table)

Candidate Pruning for 2-sequence

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

(sequences that contain infrequent sub-sequence are pruned)

Support Count for 2-sequence

	<A, A>	<AB>	<AC>	<A,C>	<B,A>	<B,B>	<B,C>	<C,A>	<C,B>	<C,C>	<D,A>
Count	3	5	3	3	4	3	5	5	5	3	3

Candidate Generation & Pruning for 3-sequence

	<A, A>	<AB>	<AC>	<A,C>	<B,A>	<B,B>	<B,C>	<C,A>	<C,B>	<C,C>	<D,A>
<A, A>	~~<A,A,A>~~	~~<A, AB>~~	~~<A, AC>~~	~~<A,A, C>~~							
<AB>	~~<AB,A>~~			<AB, C>		~~<AB ,B>~~					
<AC>	~~<AC,A>~~			~~<AC, C>~~					~~<AC, B>~~		
<A,C>								~~<A,C, A>~~	~~<A,C, B>~~	~~<A,C, C>~~	
<B,A>	~~<B,A,A>~~	<B, AB>	~~<B, AC>~~	~~<B,A, C>~~							
<B,B>					~~<B,B, A>~~	~~<B,B, B>~~	~~<B,B, C>~~				
<B,C>								<B,C, A>	<B,C, B>	~~<B,C, C>~~	
<C,A>	~~<C,A,A>~~	<C, AB>	~~<C, AC>~~	<C,A, C>							
<C,B>					~~<C,B, A>~~	~~<C,B, B>~~	<C,B, C>				
<C,C>								~~<C,C, A>~~	~~<C,C, B>~~	~~<C,C, C>~~	
<D,A>	~~<D,A,A>~~	~~<D, AB>~~	~~<D, AC>~~	~~<D,A, C>~~							

(repeated sequences are omitted; sequences that contain infrequent subsequences are stroke through)

Support Count for 3-sequence

	<AB,C>	<B,AB>	<B,C,A>	<B,C,B>	<C,AB>	<C,A,C>	<C,B,C>
count	3	3	3	3	5	3	3

Candidate Generation & Pruning for 4-sequence

	<AB,C>	<B,AB>	<B,C,A>	<B,C,B>	<C,AB>	<C,A,C>	<C,B,C>
<AB,C>			~~<AB,C,A>~~	~~<AB,C,B>~~			
<B,AB>	~~<B,AB,C>~~						
<B,C,A>					<B,C,AB>	~~<B,C,A,C>~~	
<B,C,B>							~~<B,C,B,C>~~
<C,AB>	<C,AB,C>						
<C,A,C>							
<C,B,C>			~~<C,B,C,A>~~	~~<C,B,C,B>~~			

(repeated sequences are omitted; sequences that contain infrequent subsequences are stroke through)

Support Count for 4-sequence

	<B,C,AB>	<C,AB,C>
Count	3	4