blossom

April 1, 2021

1 Blossoming Date Prediction

In this exercise we will predict the flowering date of cherry blossoms at Hirosaki Park in Hirosaki City, Aomori Prefecture, Japan using a Nueral Network by analyzing the temperature data and previous flowering information (from 1997 - 2019).

[1]:
```
import numpy as np
import pandas as pd
```

1.1 Step 1. Import Data

[2]:
```
data_blossom = pd.read_csv('hirosaki_temp_cherry_bloom.csv')
data_blossom.head(5)
```

[2]:
```
       date  temperature flower_status
0  1997/1/1          2.9           NaN
1  1997/1/2          2.2           NaN
2  1997/1/3         -1.6           NaN
3  1997/1/4          0.2           NaN
4  1997/1/5         -0.4           NaN
```

[3]:
```
df_blossom = pd.DataFrame(data_blossom)

# Adding 3 new cols: split date into year,month,day
dateList = df_blossom['date'].str.split('/', expand=True)
df_blossom['year'], df_blossom['month'], df_blossom['day'] = dateList[0],␣
↪dateList[1], dateList[2]
df_blossom.info()
df_blossom.head(5)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8400 entries, 0 to 8399
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   date           8400 non-null   object
 1   temperature    8400 non-null   float64
 2   flower_status  69 non-null     object
```

```
 3   year           8400 non-null   object
 4   month          8400 non-null   object
 5   day            8400 non-null   object
dtypes: float64(1), object(5)
memory usage: 393.9+ KB
```

[3]:

	date	temperature	flower_status	year	month	day
0	1997/1/1	2.9	NaN	1997	1	1
1	1997/1/2	2.2	NaN	1997	1	2
2	1997/1/3	-1.6	NaN	1997	1	3
3	1997/1/4	0.2	NaN	1997	1	4
4	1997/1/5	-0.4	NaN	1997	1	5

1.2 Step 2. Data Cleaning and Preprocessing

[4]:
```python
df_blossom['flower_status'].unique()
```

[4]:
```
array([nan, 'bloom', 'full', 'scatter'], dtype=object)
```

[5]:
```python
new_df_blossom = []

# 0:Before blooming, 1:Bloom, 2:Full bloom, 3:Scatter
# Note that we consider the status of flower to be 0 at the begining of the year
# and we fill in all NaN as below:
# NaN NaN NaN 1 NaN NaN 2 NaN NaN NaN 3 NaN NaN -->
# 0 0 0 1 1 1 2 2 2 2 3 3 3

for i in range(len(df_blossom)):
    if ((df_blossom['month'][i] == '1') & (df_blossom['day'][i] == '1')):
        status = 0
    elif(df_blossom['flower_status'][i] == 'bloom'):
        status = 1
    elif(df_blossom['flower_status'][i] == 'full'):
        status = 2
    elif(df_blossom['flower_status'][i] == 'scatter'):
        status = 3

    new_df_blossom.append({'year':df_blossom['year'][i],
                           'month':df_blossom['month'][i],
                           'day':df_blossom['day'][i],
                           'temperature':df_blossom['temperature'][i],
                           'flower_status':status})

new_df_blossom = pd.DataFrame(new_df_blossom)
print(new_df_blossom)
```

```
      year month day  temperature  flower_status
```

```
0     1997    1   1         2.9            0
1     1997    1   2         2.2            0
2     1997    1   3        -1.6            0
3     1997    1   4         0.2            0
4     1997    1   5        -0.4            0
…     …    …   ..        …            …
8395  2019   12  27        -0.2            3
8396  2019   12  28        -1.3            3
8397  2019   12  29        -0.6            3
8398  2019   12  30         1.8            3
8399  2019   12  31        -0.2            3

[8400 rows x 5 columns]
```

[6]:
```
new_df_blossom['flower_status'].unique()
```

[6]:
```
array([0, 1, 2, 3])
```

[7]:
```
new_df_blossom.head(5)
```

[7]:
```
   year month day  temperature  flower_status
0  1997     1   1          2.9              0
1  1997     1   2          2.2              0
2  1997     1   3         -1.6              0
3  1997     1   4          0.2              0
4  1997     1   5         -0.4              0
```

[8]:
```
new_df_blossom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8400 entries, 0 to 8399
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   year           8400 non-null   object
 1   month          8400 non-null   object
 2   day            8400 non-null   object
 3   temperature    8400 non-null   float64
 4   flower_status  8400 non-null   int64
dtypes: float64(1), int64(1), object(3)
memory usage: 328.2+ KB
```

[9]:
```
new_df_blossom.flower_status.value_counts()
```

[9]:
```
3    5648
0    2537
1     127
2      88
```

```
Name: flower_status, dtype: int64
```

[10]:
```python
# Since most of our flower_status data is 0 or 3 we focus on the months when it␣
↪is more likely to have blossoms
# March, April, May
new_df_blossom = new_df_blossom.loc[(new_df_blossom['month'] == '3') |
                                    (new_df_blossom['month'] == '4') |
                                    (new_df_blossom['month'] == '5') ]
new_df_blossom.reset_index(drop=True, inplace=True)
```

1.3 Step 3. Split Data to Train and Test Sets

[11]:
```python
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

X = new_df_blossom.drop('flower_status', axis = 1)
y = new_df_blossom['flower_status']

X = X.astype('float64')
y = y.astype('int32')

features = MinMaxScaler().fit_transform(X)
X_train, X_test, y_train, y_test = train_test_split(features, y, test_size=0.3)
```

[12]:
```python
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2116 entries, 0 to 2115
Data columns (total 4 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   year         2116 non-null   float64
 1   month        2116 non-null   float64
 2   day          2116 non-null   float64
 3   temperature  2116 non-null   float64
dtypes: float64(4)
memory usage: 66.2 KB
```

[13]:
```python
y_test.value_counts()
```

[13]:
```
0    353
3    223
1     35
2     24
Name: flower_status, dtype: int64
```

1.4 Step 4. Our Model (Neural Network - MLPClassifier)

[14]:

```
from sklearn import neural_network

clf = neural_network.MLPClassifier(max_iter=1000,       # default:200
                                   activation="relu",   # default:"relu"
                                   solver="adam",       # default:"adam"
                                   alpha=0.0001,        # default:0.0001
                                   verbose=True,        # default:False
                                   early_stopping=False)# default:False
# fit our model
clf.fit(X_train, y_train)
```

```
Iteration 1, loss = 1.35057868
Iteration 2, loss = 1.23333781
Iteration 3, loss = 1.13863914
Iteration 4, loss = 1.06236393
Iteration 5, loss = 0.99978717
Iteration 6, loss = 0.94761075
Iteration 7, loss = 0.90471196
Iteration 8, loss = 0.86529212
Iteration 9, loss = 0.83154925
Iteration 10, loss = 0.80056069
Iteration 11, loss = 0.77185861
Iteration 12, loss = 0.74422368
Iteration 13, loss = 0.71869487
Iteration 14, loss = 0.69507559
Iteration 15, loss = 0.67318989
Iteration 16, loss = 0.65334377
Iteration 17, loss = 0.63448556
Iteration 18, loss = 0.61731953
Iteration 19, loss = 0.60148103
Iteration 20, loss = 0.58669883
Iteration 21, loss = 0.57307026
Iteration 22, loss = 0.56023860
Iteration 23, loss = 0.54823722
Iteration 24, loss = 0.53699665
Iteration 25, loss = 0.52619678
Iteration 26, loss = 0.51604581
Iteration 27, loss = 0.50643704
Iteration 28, loss = 0.49739774
Iteration 29, loss = 0.48857926
Iteration 30, loss = 0.48047863
Iteration 31, loss = 0.47267915
Iteration 32, loss = 0.46526493
Iteration 33, loss = 0.45795907
Iteration 34, loss = 0.45093560
Iteration 35, loss = 0.44416630
```

```
Iteration 36, loss = 0.43761744
Iteration 37, loss = 0.43148325
Iteration 38, loss = 0.42537475
Iteration 39, loss = 0.41957684
Iteration 40, loss = 0.41433772
Iteration 41, loss = 0.40888387
Iteration 42, loss = 0.40378620
Iteration 43, loss = 0.39893750
Iteration 44, loss = 0.39448232
Iteration 45, loss = 0.39016707
Iteration 46, loss = 0.38563883
Iteration 47, loss = 0.38164022
Iteration 48, loss = 0.37840905
Iteration 49, loss = 0.37416274
Iteration 50, loss = 0.37060410
Iteration 51, loss = 0.36703623
Iteration 52, loss = 0.36377904
Iteration 53, loss = 0.36059548
Iteration 54, loss = 0.35738626
Iteration 55, loss = 0.35441410
Iteration 56, loss = 0.35208596
Iteration 57, loss = 0.34920964
Iteration 58, loss = 0.34649062
Iteration 59, loss = 0.34392359
Iteration 60, loss = 0.34142734
Iteration 61, loss = 0.33919333
Iteration 62, loss = 0.33712987
Iteration 63, loss = 0.33468378
Iteration 64, loss = 0.33272340
Iteration 65, loss = 0.33092762
Iteration 66, loss = 0.32899942
Iteration 67, loss = 0.32749736
Iteration 68, loss = 0.32535795
Iteration 69, loss = 0.32354084
Iteration 70, loss = 0.32245138
Iteration 71, loss = 0.32028775
Iteration 72, loss = 0.31887320
Iteration 73, loss = 0.31721533
Iteration 74, loss = 0.31594014
Iteration 75, loss = 0.31430925
Iteration 76, loss = 0.31298109
Iteration 77, loss = 0.31164184
Iteration 78, loss = 0.30999768
Iteration 79, loss = 0.30906171
Iteration 80, loss = 0.30757814
Iteration 81, loss = 0.30660741
Iteration 82, loss = 0.30549943
Iteration 83, loss = 0.30404480
```

```
Iteration 84, loss = 0.30317094
Iteration 85, loss = 0.30192167
Iteration 86, loss = 0.30087608
Iteration 87, loss = 0.29979396
Iteration 88, loss = 0.29892597
Iteration 89, loss = 0.29823436
Iteration 90, loss = 0.29747700
Iteration 91, loss = 0.29637148
Iteration 92, loss = 0.29592393
Iteration 93, loss = 0.29460979
Iteration 94, loss = 0.29379030
Iteration 95, loss = 0.29278102
Iteration 96, loss = 0.29260102
Iteration 97, loss = 0.29149935
Iteration 98, loss = 0.29031509
Iteration 99, loss = 0.28967287
Iteration 100, loss = 0.28914070
Iteration 101, loss = 0.28898718
Iteration 102, loss = 0.28774790
Iteration 103, loss = 0.28667370
Iteration 104, loss = 0.28594184
Iteration 105, loss = 0.28538334
Iteration 106, loss = 0.28464930
Iteration 107, loss = 0.28420743
Iteration 108, loss = 0.28327552
Iteration 109, loss = 0.28256321
Iteration 110, loss = 0.28225177
Iteration 111, loss = 0.28150846
Iteration 112, loss = 0.28082726
Iteration 113, loss = 0.28048895
Iteration 114, loss = 0.27970414
Iteration 115, loss = 0.27917344
Iteration 116, loss = 0.27860041
Iteration 117, loss = 0.27840687
Iteration 118, loss = 0.27759007
Iteration 119, loss = 0.27723740
Iteration 120, loss = 0.27667802
Iteration 121, loss = 0.27617146
Iteration 122, loss = 0.27596060
Iteration 123, loss = 0.27504063
Iteration 124, loss = 0.27495938
Iteration 125, loss = 0.27434093
Iteration 126, loss = 0.27389281
Iteration 127, loss = 0.27344930
Iteration 128, loss = 0.27291245
Iteration 129, loss = 0.27257188
Iteration 130, loss = 0.27202817
Iteration 131, loss = 0.27174577
```

```
Iteration 132, loss = 0.27154853
Iteration 133, loss = 0.27097296
Iteration 134, loss = 0.27076438
Iteration 135, loss = 0.27007160
Iteration 136, loss = 0.26997451
Iteration 137, loss = 0.26931420
Iteration 138, loss = 0.26909872
Iteration 139, loss = 0.26871171
Iteration 140, loss = 0.26871003
Iteration 141, loss = 0.26821959
Iteration 142, loss = 0.26802680
Iteration 143, loss = 0.26762364
Iteration 144, loss = 0.26726192
Iteration 145, loss = 0.26648736
Iteration 146, loss = 0.26699309
Iteration 147, loss = 0.26679889
Iteration 148, loss = 0.26588542
Iteration 149, loss = 0.26577671
Iteration 150, loss = 0.26531277
Iteration 151, loss = 0.26507547
Iteration 152, loss = 0.26456350
Iteration 153, loss = 0.26441380
Iteration 154, loss = 0.26401346
Iteration 155, loss = 0.26386877
Iteration 156, loss = 0.26362078
Iteration 157, loss = 0.26308367
Iteration 158, loss = 0.26306322
Iteration 159, loss = 0.26279917
Iteration 160, loss = 0.26248707
Iteration 161, loss = 0.26236074
Iteration 162, loss = 0.26236431
Iteration 163, loss = 0.26179413
Iteration 164, loss = 0.26144835
Iteration 165, loss = 0.26123946
Iteration 166, loss = 0.26110398
Iteration 167, loss = 0.26077662
Iteration 168, loss = 0.26113779
Iteration 169, loss = 0.26066910
Iteration 170, loss = 0.26028032
Iteration 171, loss = 0.26004058
Iteration 172, loss = 0.26020418
Iteration 173, loss = 0.25951505
Iteration 174, loss = 0.25958358
Iteration 175, loss = 0.25937823
Iteration 176, loss = 0.25949028
Iteration 177, loss = 0.25904999
Iteration 178, loss = 0.25863301
Iteration 179, loss = 0.25870568
```

```
Iteration 180, loss = 0.25820895
Iteration 181, loss = 0.25831166
Iteration 182, loss = 0.25794988
Iteration 183, loss = 0.25767753
Iteration 184, loss = 0.25752898
Iteration 185, loss = 0.25717327
Iteration 186, loss = 0.25711718
Iteration 187, loss = 0.25711412
Iteration 188, loss = 0.25685955
Iteration 189, loss = 0.25695052
Iteration 190, loss = 0.25673291
Iteration 191, loss = 0.25638609
Iteration 192, loss = 0.25644189
Iteration 193, loss = 0.25612303
Iteration 194, loss = 0.25603065
Iteration 195, loss = 0.25577389
Iteration 196, loss = 0.25576712
Iteration 197, loss = 0.25553610
Iteration 198, loss = 0.25534724
Iteration 199, loss = 0.25541673
Iteration 200, loss = 0.25557314
Iteration 201, loss = 0.25473233
Iteration 202, loss = 0.25454367
Iteration 203, loss = 0.25453497
Iteration 204, loss = 0.25442911
Iteration 205, loss = 0.25447439
Iteration 206, loss = 0.25473734
Iteration 207, loss = 0.25471565
Iteration 208, loss = 0.25417620
Iteration 209, loss = 0.25389529
Iteration 210, loss = 0.25395329
Iteration 211, loss = 0.25365723
Iteration 212, loss = 0.25362052
Iteration 213, loss = 0.25347657
Iteration 214, loss = 0.25331244
Iteration 215, loss = 0.25343286
Iteration 216, loss = 0.25322050
Iteration 217, loss = 0.25323017
Iteration 218, loss = 0.25296200
Iteration 219, loss = 0.25296733
Iteration 220, loss = 0.25255653
Iteration 221, loss = 0.25242551
Iteration 222, loss = 0.25237616
Iteration 223, loss = 0.25212881
Iteration 224, loss = 0.25216444
Iteration 225, loss = 0.25223600
Iteration 226, loss = 0.25222748
Iteration 227, loss = 0.25189783
```

```
Iteration 228, loss = 0.25231510
Iteration 229, loss = 0.25164133
Iteration 230, loss = 0.25183892
Iteration 231, loss = 0.25138467
Iteration 232, loss = 0.25175620
Iteration 233, loss = 0.25164405
Iteration 234, loss = 0.25131352
Iteration 235, loss = 0.25123772
Iteration 236, loss = 0.25134337
Iteration 237, loss = 0.25122486
Iteration 238, loss = 0.25143313
Iteration 239, loss = 0.25115766
Iteration 240, loss = 0.25086389
Iteration 241, loss = 0.25087488
Iteration 242, loss = 0.25079806
Iteration 243, loss = 0.25086179
Iteration 244, loss = 0.25061839
Iteration 245, loss = 0.25057116
Iteration 246, loss = 0.25055215
Iteration 247, loss = 0.25018746
Iteration 248, loss = 0.25038352
Iteration 249, loss = 0.25057323
Iteration 250, loss = 0.25048211
Iteration 251, loss = 0.24987045
Iteration 252, loss = 0.25024844
Iteration 253, loss = 0.24988540
Iteration 254, loss = 0.24989082
Iteration 255, loss = 0.24996536
Iteration 256, loss = 0.25043904
Iteration 257, loss = 0.25041149
Iteration 258, loss = 0.25013648
Iteration 259, loss = 0.24998335
Iteration 260, loss = 0.24930259
Iteration 261, loss = 0.24944699
Iteration 262, loss = 0.24991756
Iteration 263, loss = 0.24952486
Iteration 264, loss = 0.24953532
Iteration 265, loss = 0.24924048
Iteration 266, loss = 0.24919683
Iteration 267, loss = 0.24936531
Iteration 268, loss = 0.24935665
Iteration 269, loss = 0.24996325
Iteration 270, loss = 0.24850983
Iteration 271, loss = 0.24929849
Iteration 272, loss = 0.24879606
Iteration 273, loss = 0.24904276
Iteration 274, loss = 0.24867559
Iteration 275, loss = 0.24854848
```

```
Iteration 276, loss = 0.24860837
Iteration 277, loss = 0.24825079
Iteration 278, loss = 0.24845594
Iteration 279, loss = 0.24832549
Iteration 280, loss = 0.24832675
Iteration 281, loss = 0.24830462
Iteration 282, loss = 0.24883532
Iteration 283, loss = 0.24805637
Iteration 284, loss = 0.24842766
Iteration 285, loss = 0.24860333
Iteration 286, loss = 0.24797082
Iteration 287, loss = 0.24814348
Iteration 288, loss = 0.24795995
Iteration 289, loss = 0.24795085
Iteration 290, loss = 0.24839881
Iteration 291, loss = 0.24805941
Iteration 292, loss = 0.24864926
Iteration 293, loss = 0.24818708
Iteration 294, loss = 0.24785723
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs.
Stopping.
```

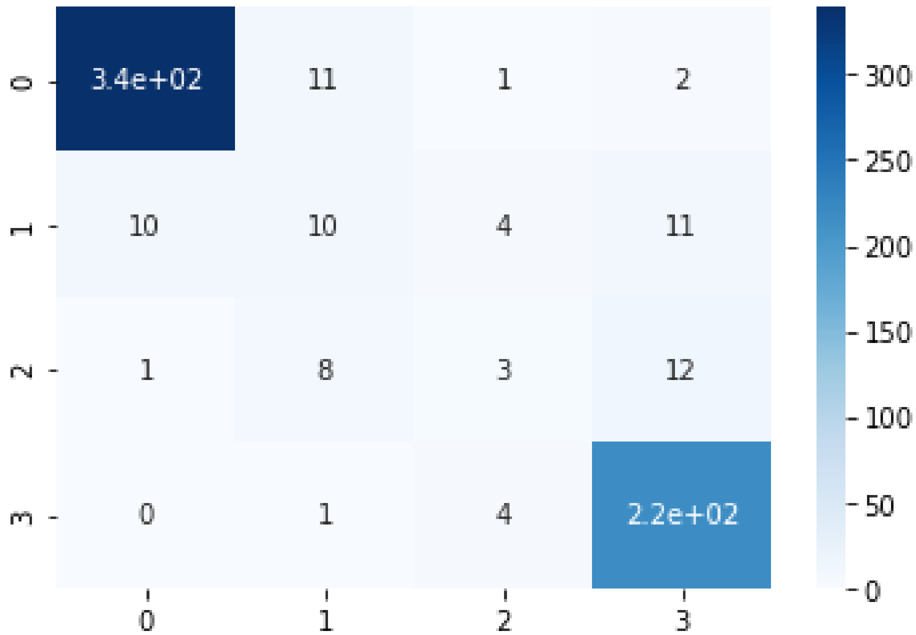
[14]:
```
MLPClassifier(max_iter=1000, verbose=True)
```

1.5 Step 5. Plot the Test Results

[15]:
```python
predict = clf.predict(X_test)
```

[16]:
```python
from sklearn.metrics import confusion_matrix, classification_report,␣
↪accuracy_score
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, predict)
sns.heatmap(cm, annot=True, cmap='Blues')
plt.figure(figsize=(16,6))
plt.show()
```




```
<Figure size 1152x432 with 0 Axes>
```

[17]:

```python
classReport = classification_report(y_test, predict)
print(classReport)

score = accuracy_score(y_test, predict)
print('accuracy :', '{:.5f}'.format(score))
```

```
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       353
           1       0.33      0.29      0.31        35
           2       0.25      0.12      0.17        24
           3       0.90      0.98      0.94       223

    accuracy                           0.90       635
   macro avg       0.61      0.59      0.59       635
weighted avg       0.88      0.90      0.89       635

accuracy : 0.89764
```

[ ]:

```python
```