Scaling vs normalization

May 20, 2022

[1]:
```
import pandas as pd
import numpy as np
from sklearn import datasets
wine = datasets.load_wine()
wine = pd.DataFrame(
    data=np.c_[wine['data'], wine['target']],
    columns=wine['feature_names'] + ['target']
)
```

[2]:
```
wine
```

[2]:
```
     alcohol  malic_acid   ash  alcalinity_of_ash  magnesium  total_phenols  \
0      14.23        1.71  2.43               15.6      127.0           2.80
1      13.20        1.78  2.14               11.2      100.0           2.65
2      13.16        2.36  2.67               18.6      101.0           2.80
3      14.37        1.95  2.50               16.8      113.0           3.85
4      13.24        2.59  2.87               21.0      118.0           2.80
..       …         …    …                 …          …              …
173    13.71        5.65  2.45               20.5       95.0           1.68
174    13.40        3.91  2.48               23.0      102.0           1.80
175    13.27        4.28  2.26               20.0      120.0           1.59
176    13.17        2.59  2.37               20.0      120.0           1.65
177    14.13        4.10  2.74               24.5       96.0           2.05

     flavanoids  nonflavanoid_phenols  proanthocyanins  color_intensity   hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          …                   …                …              …    …
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61
```

```
     od280/od315_of_diluted_wines  proline  target
0                            3.92   1065.0     0.0
1                            3.40   1050.0     0.0
2                            3.17   1185.0     0.0
3                            3.45   1480.0     0.0
4                            2.93    735.0     0.0
..                            …        …     …
173                          1.74    740.0     2.0
174                          1.56    750.0     2.0
175                          1.56    835.0     2.0
176                          1.62    840.0     2.0
177                          1.60    560.0     2.0

[178 rows x 14 columns]
```

[3]:
```
wine[['magnesium', 'total_phenols', 'color_intensity']].describe()
```

[3]:
```
        magnesium  total_phenols  color_intensity
count  178.000000     178.000000       178.000000
mean    99.741573       2.295112         5.058090
std     14.282484       0.625851         2.318286
min     70.000000       0.980000         1.280000
25%     88.000000       1.742500         3.220000
50%     98.000000       2.355000         4.690000
75%    107.000000       2.800000         6.200000
max    162.000000       3.880000        13.000000
```

As you can see all tree columns have different data distributions. So it may present a problem with distance based models like knn

[4]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   alcohol                       178 non-null    float64
 1   malic_acid                    178 non-null    float64
 2   ash                           178 non-null    float64
 3   alcalinity_of_ash             178 non-null    float64
 4   magnesium                     178 non-null    float64
 5   total_phenols                 178 non-null    float64
 6   flavanoids                    178 non-null    float64
 7   nonflavanoid_phenols          178 non-null    float64
 8   proanthocyanins               178 non-null    float64
 9   color_intensity               178 non-null    float64
 10  hue                           178 non-null    float64
```

```
 11   od280/od315_of_diluted_wines  178 non-null    float64
 12   proline                       178 non-null    float64
 13   target                        178 non-null    float64
dtypes: float64(14)
memory usage: 19.6 KB
```

1 Let's try a KNN

[5]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[6]:
```
knn_clf = KNeighborsClassifier()
```

[7]:
```
# Let's split data first
from sklearn.model_selection import train_test_split
y = wine["target"]
X = wine.drop(["target"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33,␣
 ↪random_state=42)
```

[8]:
```
model = knn_clf.fit(X_train, y_train)
```

[9]:
```
preds = model.predict(X_test)
```

[10]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, preds))
```

```
              precision    recall  f1-score   support

         0.0       0.85      0.85      0.85        20
         1.0       0.67      0.67      0.67        24
         2.0       0.47      0.47      0.47        15

    accuracy                           0.68        59
   macro avg       0.66      0.66      0.66        59
weighted avg       0.68      0.68      0.68        59
```

[11]:
```
# current data distribution is
y_test.value_counts() / sum(y_test.value_counts())

# this enables us to compare with random model
```

[11]:
```
1.0    0.406780
0.0    0.338983
2.0    0.254237
Name: target, dtype: float64
```

2 Let's do the same thing but with data normalization

[12]:
```
from sklearn.preprocessing import StandardScaler
# create the scaler
ss = StandardScaler()
```

[13]:
```
X_train_norm = ss.fit_transform(X_train)
X_test_norm = ss.fit_transform(X_test)
```

[14]:
```
pd.DataFrame(X_train_norm).describe()
# as can be seen now the values have average close to zero and std=1
```

[14]:
```
                  0             1             2             3             4  \
count  1.190000e+02  1.190000e+02  1.190000e+02  1.190000e+02  1.190000e+02
mean   9.143013e-16 -9.358177e-16  2.928563e-15  1.026257e-16 -2.549315e-16
std    1.004228e+00  1.004228e+00  1.004228e+00  1.004228e+00  1.004228e+00
min   -2.287878e+00 -1.380891e+00 -3.788813e+00 -2.570844e+00 -2.043016e+00
25%   -8.027088e-01 -6.965013e-01 -5.132388e-01 -6.092008e-01 -8.512565e-01
50%   -1.297567e-02 -4.653499e-01 -4.529965e-02 -2.213963e-02 -1.891681e-01
75%    8.710539e-01  6.858750e-01  5.723800e-01  5.506030e-01  4.729203e-01
max    2.191205e+00  3.069907e+00  3.211557e+00  2.984759e+00  4.048198e+00

                  5             6             7             8             9  \
count  1.190000e+02  1.190000e+02  1.190000e+02  1.190000e+02  1.190000e+02
mean   2.287153e-15 -3.883448e-17 -4.688127e-16  6.260165e-16  5.392512e-16
std    1.004228e+00  1.004228e+00  1.004228e+00  1.004228e+00  1.004228e+00
min   -1.966736e+00 -1.648405e+00 -1.847646e+00 -2.028179e+00 -1.496884e+00
25%   -8.912050e-01 -8.559681e-01 -7.423755e-01 -5.965104e-01 -8.179582e-01
50%    3.284258e-02  1.235043e-01 -1.897403e-01 -4.987326e-02 -1.025798e-01
75%    8.129811e-01  8.421121e-01  6.786864e-01  6.008852e-01  5.193893e-01
max    2.426277e+00  3.017623e+00  2.336592e+00  3.455546e+00  2.631350e+00

                 10            11            12
count  1.190000e+02  1.190000e+02  1.190000e+02
mean   1.708251e-15  2.164468e-16 -1.016927e-16
std    1.004228e+00  1.004228e+00  1.004228e+00
min   -2.018458e+00 -1.809669e+00 -1.513264e+00
25%   -7.896365e-01 -1.050959e+00 -7.967631e-01
50%    1.545374e-02  2.545691e-01 -2.663237e-01
75%    6.934245e-01  7.877167e-01  7.162131e-01
max    3.193442e+00  1.922364e+00  2.629059e+00
```

[15]:
```
model = knn_clf.fit(X_train_norm, y_train)
preds = model.predict(X_test_norm)
from sklearn.metrics import classification_report
print(classification_report(y_test, preds))
```

```
# as can be seen in the results below, the results dramatically improved
```

```
              precision    recall  f1-score   support

         0.0       0.95      1.00      0.98        20
         1.0       1.00      0.88      0.93        24
         2.0       0.88      1.00      0.94        15

    accuracy                           0.95        59
   macro avg       0.94      0.96      0.95        59
weighted avg       0.95      0.95      0.95        59
```

3 Let's do the same thing but with min max scaler

[16]:
```
from sklearn.preprocessing import MinMaxScaler
```

[20]:
```
# create the scaler
minmax = MinMaxScaler()
X_train_mm = minmax.fit_transform(X_train)
X_test_mm = minmax.fit_transform(X_test)
```

[21]:
```
pd.DataFrame(X_train_mm).describe()
```

[21]:
```
                0           1           2           3           4           5  \
count  119.000000  119.000000  119.000000  119.000000  119.000000  119.000000
mean     0.510792    0.310257    0.541230    0.462748    0.335404    0.447696
std      0.224204    0.225629    0.143454    0.180760    0.164865    0.228597
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.331579    0.153768    0.467914    0.353093    0.195652    0.244828
50%      0.507895    0.205703    0.534759    0.458763    0.304348    0.455172
75%      0.705263    0.464358    0.622995    0.561856    0.413043    0.632759
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

                6           7           8           9          10          11  \
count  119.000000  119.000000  119.000000  119.000000  119.000000  119.000000
mean     0.353278    0.441573    0.369854    0.362597    0.387279    0.484902
std      0.215221    0.240003    0.183129    0.243259    0.192680    0.269083
min      0.000000    0.000000    0.000000    0.000000    0.000000    0.000000
25%      0.169831    0.264151    0.261076    0.164459    0.235772    0.203297
50%      0.379747    0.396226    0.360759    0.337748    0.390244    0.553114
75%      0.533755    0.603774    0.479430    0.488411    0.520325    0.695971
max      1.000000    1.000000    1.000000    1.000000    1.000000    1.000000

                12
count  119.000000
```

```
mean     0.365318
std      0.242431
min      0.000000
25%      0.172971
50%      0.301024
75%      0.538219
max      1.000000
```

[22]:

```python
model = knn_clf.fit(X_train_mm, y_train)
preds = model.predict(X_test_mm)
from sklearn.metrics import classification_report
print(classification_report(y_test, preds))

# as can be seen in the results below, the results dramatically improved
```

```
              precision    recall  f1-score   support

         0.0       0.91      1.00      0.95        20
         1.0       1.00      0.92      0.96        24
         2.0       1.00      1.00      1.00        15

    accuracy                           0.97        59
   macro avg       0.97      0.97      0.97        59
weighted avg       0.97      0.97      0.97        59
```

Again the results improved relative to the baseline, but they are equal or marginally better compared to standard normalization

[ ]: