solution

May 19, 2020

1 Problem 1 - Machine Learning

1.0.1 Data Cleansing

We first read the Iris dataset from 'iris_data_for_cleansing.csv' and then perform the numeric data cleansing. To be precise, we replace the missing data with the mean of the +/- times the standard deviation δ.

We randomly select the value to add to the mean that is in the range of $[-2\delta, 2\delta]$.

By the way, we can use 'pandas' to read the csv file.

[32]:
```
import pandas as pd
import numpy as np
import random

df = pd.read_csv('iris_data_for_cleansing.csv', sep=',')

n, m = df.shape
for i in range(n):
    for j in range(m):
        if np.isnan(df.iloc[i, j]):
            label = int(df.iloc[i, 6])
            start = 50 * (label - 1)
            end = 50 * label
            mu = df.iloc[start:end].mean()[j]
            sigma = df.iloc[start:end].std(ddof=0)[j]
            df.iloc[i, j] = 4 * sigma * random.random() + mu - 2 * sigma

df.head(20)
```

[32]:
```
    sepal length  sepal width  petal length  petal width  New Feature 1  \
0            5.1     3.500000      1.400000          0.2       1.611281
1            4.9     3.000000      1.400000          0.2       1.295847
2            4.7     2.956786      1.300000          0.2       1.685578
3            4.6     3.100000      1.500000          0.2       1.546064
4            5.0     3.600000      1.400000          0.2       1.501464
5            5.4     3.900000      1.700000          0.4       1.325835
6            4.6     3.400000      1.400000          0.3       1.529386
7            5.0     3.400000      1.500000          0.2       1.331028
8            4.4     2.900000      1.400000          0.2       1.147873
```

```
9             4.9     3.100000      1.500000          0.1       1.401505
10            5.4     3.700000      1.500000          0.2       1.340574
11            4.8     3.400000      1.600000          0.2       1.259047
12            4.8     3.000000      1.526822          0.1       1.526365
13            4.3     3.000000      1.100000          0.1       1.314645
14            5.8     4.000000      1.200000          0.2       1.495580
15            5.7     4.400000      1.500000          0.4       1.403405
16            5.4     3.900000      1.300000          0.4       1.651607
17            5.1     3.500000      1.400000          0.3       1.181882
18            5.7     3.800000      1.700000          0.3       1.432863
19            5.1     3.800000      1.500000          0.3       1.193632

    New Feature 2  class
0        2.981148      1
1        2.210908      1
2        3.114562      1
3        2.714977      1
4        2.815603      1
5        2.235803      1
6        2.792032      1
7        2.278499      1
8        1.976229      1
9        2.434789      1
10       2.380203      1
11       2.296553      1
12       2.794599      1
13       2.262849      1
14       2.590192      1
15       2.429602      1
16       3.045064      1
17       2.088136      1
18       2.460808      1
19       2.108733      1
```

[33]:
```
df.tail(20)
```

[33]:
```
     sepal length  sepal width  petal length  petal width  New Feature 1  \
130      7.400000     2.800000      6.100000          1.9       1.327588
131      7.900000     3.800000      6.400000          2.0       1.542274
132      6.400000     3.518572      5.600000          2.2       1.150044
133      6.300000     2.800000      5.100000          1.5       1.412455
134      6.100000     2.600000      5.600000          1.4       1.401033
135      7.700000     3.000000      6.100000          2.3       1.211413
136      6.300000     3.400000      5.600000          2.4       1.269478
137      6.400000     3.100000      5.500000          1.8       1.412603
138      6.000000     3.000000      4.800000          1.8       1.488619
139      6.900000     3.100000      5.400000          2.1       1.270319
140      6.700000     3.100000      5.600000          2.4       1.491015
```

```
141        6.900000     3.100000      6.251812            2.3        1.469114
142        5.800000     2.700000      5.100000            1.9        1.273977
143        6.800000     3.200000      5.900000            2.3        1.371689
144        6.700000     3.300000      5.700000            2.5        1.409968
145        6.700000     3.000000      5.200000            2.3        1.491191
146        6.300000     2.500000      5.000000            1.9        1.259320
147        6.500000     3.000000      5.200000            2.0        1.419002
148        5.768616     3.400000      5.400000            2.3        1.131173
149        5.900000     3.000000      5.100000            1.8        1.228544

     New Feature 2  class
130       4.494880      3
131       4.989386      3
132       4.053020      3
133       4.662783      3
134       4.702212      3
135       4.165725      3
136       4.515550      3
137       4.593121      3
138       4.857393      3
139       4.378299      3
140       4.753491      3
141       4.710281      3
142       4.309591      3
143       4.660687      3
144       4.520793      3
145       4.936018      3
146       4.360127      3
147       4.583592      3
148       4.027495      3
149       4.334454      3
```

1.0.2 Generate two sets of features from the original 4 features

Let's generate two more features, the mean value and the standard deviation of the original 4 features, to extend every sample to in total 8 features.

[34]:
```
new_feature_3 = df.iloc[:, 0:4].mean(axis=1)
new_feature_4 = df.iloc[:, 0:4].std(axis=1, ddof=0)
df.insert(6, "New Feature 3", new_feature_3)
df.insert(7, "New Feature 4", new_feature_4)
df.head(20)
```

[34]:
```
    sepal length  sepal width  petal length  petal width  New Feature 1  \
0            5.1     3.500000      1.400000          0.2       1.611281
1            4.9     3.000000      1.400000          0.2       1.295847
2            4.7     2.956786      1.300000          0.2       1.685578
3            4.6     3.100000      1.500000          0.2       1.546064
```

```
4            5.0     3.600000      1.400000          0.2        1.501464
5            5.4     3.900000      1.700000          0.4        1.325835
6            4.6     3.400000      1.400000          0.3        1.529386
7            5.0     3.400000      1.500000          0.2        1.331028
8            4.4     2.900000      1.400000          0.2        1.147873
9            4.9     3.100000      1.500000          0.1        1.401505
10           5.4     3.700000      1.500000          0.2        1.340574
11           4.8     3.400000      1.600000          0.2        1.259047
12           4.8     3.000000      1.526822          0.1        1.526365
13           4.3     3.000000      1.100000          0.1        1.314645
14           5.8     4.000000      1.200000          0.2        1.495580
15           5.7     4.400000      1.500000          0.4        1.403405
16           5.4     3.900000      1.300000          0.4        1.651607
17           5.1     3.500000      1.400000          0.3        1.181882
18           5.7     3.800000      1.700000          0.3        1.432863
19           5.1     3.800000      1.500000          0.3        1.193632

    New Feature 2  New Feature 3  New Feature 4  class
0        2.981148       2.550000       1.887459      1
1        2.210908       2.375000       1.764051      1
2        3.114562       2.289196       1.703005      1
3        2.714977       2.350000       1.656050      1
4        2.815603       2.550000       1.867485      1
5        2.235803       2.850000       1.931968      1
6        2.792032       2.425000       1.676865      1
7        2.278499       2.525000       1.826712      1
8        1.976229       2.225000       1.578567      1
9        2.434789       2.400000       1.791647      1
10       2.380203       2.700000       1.998750      1
11       2.296553       2.500000       1.746425      1
12       2.794599       2.356705       1.743914      1
13       2.262849       2.125000       1.631525      1
14       2.590192       2.800000       2.222611      1
15       2.429602       3.000000       2.136586      1
16       3.045064       2.750000       1.998124      1
17       2.088136       2.575000       1.856576      1
18       2.460808       2.875000       2.052285      1
19       2.108733       2.675000       1.881987      1
```

1.0.3 Outlier Removal

We use confidence interval method here to remove outlier as below. Here we set $\alpha = 1.6$ and 9 outliers will be removed by this method.

[52]:
```python
def confidence(data):
    n = data.shape[0]
    p = 6
    mark = [0] * n
    u_lower = np.zeros((p, ))
    u_upper = np.zeros((p, ))
    for i in range(p):
        mu = np.mean(data[:, i])
        low = 0
        upp = 0
        count = 0
        for j in range(n):
            if data[j, i] > mu:
                upp += data[j, i]
                count += 1
            else:
                low += data[j, i]
        low /= (n - count)
        upp /= count
        u_lower[i] = low
        u_upper[i] = upp
    alpha = 1.6
    for i in range(n):
        out = False
        for j in range(p):
            mu = np.mean(data[:, j])
            if data[i, j] < u_lower[j] - alpha * (mu - u_lower[j]):
                mark[i] |= 1 << j
            if data[i, j] > u_upper[j] + alpha * (u_upper[j] - mu):
                mark[i] |= 1 << j
    return mark

conf = confidence(df.iloc[:, 0:6].values)
mdf = df[list(map(lambda x: x == 0, conf))]
print(mdf.shape)
mdf.head(20)
```

```
(141, 9)
```

[52]:

```
    sepal length  sepal width  petal length  petal width  New Feature 1  \
0            5.1          3.5      1.400000          0.2       1.611281
1            4.9          3.0      1.400000          0.2       1.295847
3            4.6          3.1      1.500000          0.2       1.546064
4            5.0          3.6      1.400000          0.2       1.501464
5            5.4          3.9      1.700000          0.4       1.325835
6            4.6          3.4      1.400000          0.3       1.529386
7            5.0          3.4      1.500000          0.2       1.331028
8            4.4          2.9      1.400000          0.2       1.147873
9            4.9          3.1      1.500000          0.1       1.401505
10           5.4          3.7      1.500000          0.2       1.340574
```

```
11          4.8          3.4       1.600000          0.2          1.259047
12          4.8          3.0       1.526822          0.1          1.526365
13          4.3          3.0       1.100000          0.1          1.314645
14          5.8          4.0       1.200000          0.2          1.495580
16          5.4          3.9       1.300000          0.4          1.651607
17          5.1          3.5       1.400000          0.3          1.181882
18          5.7          3.8       1.700000          0.3          1.432863
19          5.1          3.8       1.500000          0.3          1.193632
20          5.4          3.4       1.700000          0.2          1.426801
21          5.1          3.7       1.500000          0.4          1.488398

    New Feature 2  New Feature 3  New Feature 4  class
0        2.981148       2.550000       1.887459      1
1        2.210908       2.375000       1.764051      1
3        2.714977       2.350000       1.656050      1
4        2.815603       2.550000       1.867485      1
5        2.235803       2.850000       1.931968      1
6        2.792032       2.425000       1.676865      1
7        2.278499       2.525000       1.826712      1
8        1.976229       2.225000       1.578567      1
9        2.434789       2.400000       1.791647      1
10       2.380203       2.700000       1.998750      1
11       2.296553       2.500000       1.746425      1
12       2.794599       2.356705       1.743914      1
13       2.262849       2.125000       1.631525      1
14       2.590192       2.800000       2.222611      1
16       3.045064       2.750000       1.998124      1
17       2.088136       2.575000       1.856576      1
18       2.460808       2.875000       2.052285      1
19       2.108733       2.675000       1.881987      1
20       2.550157       2.675000       1.938266      1
21       2.567217       2.675000       1.836267      1
```

1.0.4 Rank the 6 set of features to determine which are the two top features

We could use the sum of mean differences to rank all features, i.e., $\sum_i \sum_{j \neq i} \frac{|\mu_i - \mu_j|}{\sigma}$. Note for normalising the data, we use the ratio between the difference and the standard deviation of the feature here. And for calculating the sum of mean differences, we should first calculate the mean of the features of each class. This could be done via a nested loop.

[54]:
```python
def concat_help(data, index):
    feature = np.zeros((50, 3))
    feature[:, 0] = data[0:50, index]
    feature[:, 1] = data[50:100, index]
    feature[:, 2] = data[100:150, index]
    return feature
```

```
def mean_diff(feature):
    mean = []
    q, p = feature.shape
    for i in range(p):
        mean.append(0.0)
        for j in range(q):
            mean[i] += feature[j, i]
        mean[i] /= q
    ret = 0
    for i in range(p):
        for j in range(p):
            ret += abs(mean[i] - mean[j])
    all_feature = feature.flatten()
    ret /= np.std(all_feature)
    return ret

num_feature = 6
feature_rank = []
for i in range(num_feature):
    feature = concat_help(df.values, i)
    feature_rank.append(mean_diff(feature))

feature_rank = list(enumerate(feature_rank))
for i in range(num_feature):
    for j in range(num_feature - 1 - i):
        if feature_rank[j][1] < feature_rank[j + 1][1]:
            swap = feature_rank[j]
            feature_rank[j] = feature_rank[j + 1]
            feature_rank[j + 1] = swap

print(feature_rank)
```

```
[(3, 9.37033004779957), (2, 9.30025963227339), (5, 9.25543980476204), (0,
7.621854590965482), (1, 5.941202057306683), (4, 1.0917235118411082)]
```

Thus, the rank order is 'petal width', 'petal length', 'new feature 2', 'sepal length', 'sepal width', 'new feature 1' (from the best to the worst, respectively).

1.0.5 Reduce the dimensionality to two features using PCA

Let's use the 'sklearn.decomposition.PCA' tool to help us perform the dimension reduction.

[73]:
```
from sklearn.decomposition import PCA

X = df.values[:, 0:6]
Y = df.values[:, 8].astype(int)
print(X[:20])
print(Y[:20])
```

```
[[5.1        3.5        1.4        0.2        1.6112812  2.98114776]
 [4.9        3.         1.4        0.2        1.29584692 2.21090849]
 [4.7        2.95678575 1.3        0.2        1.68557819 3.11456249]
 [4.6        3.1        1.5        0.2        1.54606365 2.71497659]
 [5.         3.6        1.4        0.2        1.50146433 2.81560308]
 [5.4        3.9        1.7        0.4        1.32583479 2.23580337]
 [4.6        3.4        1.4        0.3        1.52938595 2.79203244]
 [5.         3.4        1.5        0.2        1.33102778 2.2784994 ]
 [4.4        2.9        1.4        0.2        1.14787253 1.97622922]
 [4.9        3.1        1.5        0.1        1.4015047  2.4347891 ]
 [5.4        3.7        1.5        0.2        1.34057351 2.38020317]
 [4.8        3.4        1.6        0.2        1.25904682 2.2965532 ]
 [4.8        3.         1.52682198 0.1        1.52636464 2.7945989 ]
 [4.3        3.         1.1        0.1        1.31464473 2.26284901]
 [5.8        4.         1.2        0.2        1.49557971 2.5901917 ]
 [5.7        4.4        1.5        0.4        1.403405   2.42960237]
 [5.4        3.9        1.3        0.4        1.65160673 3.04506388]
 [5.1        3.5        1.4        0.3        1.18188168 2.0881361 ]
 [5.7        3.8        1.7        0.3        1.43286265 2.46080847]
 [5.1        3.8        1.5        0.3        1.19363174 2.10873309]]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

[74]:
```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
print(X_pca[:20])
```

```
[[-2.69279431  0.22350646]
 [-3.01148793 -0.13023135]
 [-2.81520177 -0.43860646]
 [-2.8497756  -0.36063467]
 [-2.79526385  0.25109784]
 [-2.5968124   0.7861758 ]
 [-2.88871406 -0.16401747]
 [-2.9027288   0.1939663 ]
 [-3.25786838 -0.50389706]
 [-2.88881953 -0.10122151]
 [-2.75382945  0.65787345]
 [-2.88264228  0.04118618]
 [-2.75835509 -0.30322731]
 [-3.46278165 -0.51099742]
 [-2.80291868  1.1378658 ]
 [-2.6201266   1.32659276]
 [-2.61116822  0.69140697]
 [-2.99417761  0.36484321]
 [-2.43914694  0.89227474]
 [-2.92888623  0.55901697]]
```

So, we can see that the dimension has already been reduced to 2 as above.

1.0.6 Expectation Maximization with Bayes

Let's use the 'sklearn.mixture.BayesianGaussianMixture' tool to help us to train and predict the model for iris dataset.

[75]:
```
from sklearn.mixture import BayesianGaussianMixture

em_bayes = BayesianGaussianMixture(n_components=3)
em_pred = em_bayes.fit_predict(X_pca)
print(em_pred)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 2 2 2 0 2
 2 2 2 2 2 2 2 2 2 0 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
```

Note since we don't give the labels to the classifier for training. So all $3! = 6$ permutations of the labels is valid for counting the total accuracy. Thus, we calculate all of them and output the maximum one as the accuracy.

[76]:
```
import itertools

acc = 0
for perm in itertools.permutations([1, 2, 3]):
    count = 0
    for i in range(len(em_pred)):
        if perm[em_pred[i]] == Y[i]:
            count += 1
    if count >= acc:
        acc = count
print("Accuracy: {}".format(acc / 150.0))
```

```
Accuracy: 0.98
```

So the accuracy of EM algorithm with Bayes classifier is about 98%.

1.0.7 Fisher Linear Discriminant (LDA)

Again, let's use the 'sklearn.discriminant_analysis.LinearDiscriminantAnalysis' tool to help us to train and predict the model for iris dataset. Note that we use a 60%/40% partition to split the train and the test data.

[85]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

X_train = np.concatenate((X_pca[0:30], X_pca[50:80], X_pca[100:130]))
Y_train = np.concatenate((Y[0:30], Y[50:80], Y[100:130]))
X_test = np.concatenate((X_pca[30:50], X_pca[80:100], X_pca[130:150]))
Y_test = np.concatenate((Y[30:50], Y[80:100], Y[130:150]))

lda = LinearDiscriminantAnalysis()
```

```
lda.fit(X_train, Y_train)
lda_pred = lda.predict(X_test)

from sklearn.metrics import accuracy_score

print("Accuracy: {}".format(accuracy_score(Y_test, lda_pred)))
```

```
Accuracy: 1.0
```

So the accuracy of LDA is about 100% for this dataset.

1.0.8 Neural Network Method

Let's use the 'sklearn.neural_network.MLPClassifier' tool to help us to train and predict the model for iris dataset.

[86]:
```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier()
nn.fit(X_train, Y_train)
nn_pred = nn.predict(X_test)

print("Accuracy: {}".format(accuracy_score(Y_test, nn_pred)))
```

```
Accuracy: 0.9833333333333333

/usr/local/lib/python3.7/site-
packages/sklearn/neural_network/multilayer_perceptron.py:566:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  % self.max_iter, ConvergenceWarning)
```

So the accuracy of neural nework method is about 98.3% for this dataset.

1.0.9 Support Vector Machine

Let's use the 'sklearn.svm.SVC' tool to help us to train and predict the model for iris dataset.

[89]:
```
from sklearn.svm import SVC

svm = SVC(gamma='scale')
svm.fit(X_train, Y_train)
svm_pred = svm.predict(X_test)

print("Accuracy: {}".format(accuracy_score(Y_test, svm_pred)))
```

```
Accuracy: 1.0
```

So the accuracy of support vector machine is about 100% for this dataset.

2 Problem 2 - Game Theory

2.0.1 Implement a method that uses conditional statements to play against a user of your code.

A conditional AI with 'aiSkill=2' should be implemented for this problem. We first modify the 'checkWin2' function in 'checkWin_Incomplete.py' file to provide the functionality of check whether a board state is terminated or not and return the winner of the state.
If the function returns 2, it means that AI wins.
If the function returns 1, it means that player wins.
If the function returns 0, it means that it's tie.
If the function returns 'None', it means that the state is not terminated yet.
See below code for more details.

[90]:
```python
def checkWin2(place):
    if place[1] == place[0] and place[0] == place[2] and place[1] != 0:
        return place[1]
    if place[0] == place[3] and place[0] == place[6] and place[0] != 0:
        return place[0]
    if place[0] == place[4] and place[0] == place[8] and place[0] != 0:
        return place[0]
    if place[1] == place[4] and place[1] == place[7] and place[1] != 0:
        return place[1]
    if place[2] == place[4] and place[2] == place[6] and place[2] != 0:
        return place[2]
    if place[2] == place[5] and place[2] == place[8] and place[2] != 0:
        return place[2]
    if place[3] == place[4] and place[3] == place[5] and place[3] != 0:
        return place[3]
    if place[6] == place[7] and place[8] == place[6] and place[6] != 0:
        return place[6]
    if list(filter(lambda x: x == 0, place)) == []:
        return 0
    return None
```

Then we could perform the conditional AI move based on above 'checkWin2' function as below.
Note that there are 4 level's conditions in this AI.

1. If we find that AI could win in next step, we simply take that move.
2. If we find that AI can't win in one step, but the player can win in next step, we must block that move.
3. If it is neither of the above case, we take the move of placing at the center.
4. Otherwise, we can simply make a random move.

[91]:
```python
def checkWinPos(place):
    # if AI can win in one step, take that move.
    if checkWin2(place) is not None:
        return None
    G = place.copy()
    for i in range(len(G)):
        if G[i] == 0:
            G[i] = 2
            if checkWin2(G) == 2:
                return i
            G[i] = 0
    # if player can win in next step, block that move.
    for i in range(len(G)):
        if G[i] == 0:
            G[i] = 1
            if checkWin2(G) == 1:
                return i
            G[i] = 0
    # if the center is empty, take this move
    if G[4] == 0:
        return 4
    # otherwise, take a random move
    while True:
        i = random.randint(0, 8)
        if G[i] == 0:
            return i
```

2.0.2 MiniMax

Next, we implement a Minimax search AI with 'aiSkill=3'.

The 'decisionMaker' function simply takes the result action of the 'minimax_search' recursive search function. In the implementation below, we use a 'depth' parameter to denote the depth in the search tree. When the depth is an even number, it's the AI's move (thus a maximum move). And when the depth is an odd number, it's the player's move (thus a minimum move).

And, again, we could use 'checkWin2' function to check whether a state is terminated or not. However, here, we also need to check the tie state, i.e., there is no empty cell in the board.

See below code for more details.

[92]:
```
def minimax_search(G, depth):
    win = checkWin2(G)
    if win == 1:
        # player wins.
        return None, -1
    elif win == 2:
        # AI wins.
        return None, 1
    elif win == 0:
        return None, 0

    # True for min, False for max
    min_or_max = bool(depth & 1)
    best = None
    # intial utility -2/2 is enough for this problem
    util = 2 if min_or_max else -2
    for i in range(len(G)):
        # a valid cell to place
        if G[i] == 0:
            # current player is 2 - (depth & 1).
            G[i] = 2 - (depth & 1)
            act, u = minimax_search(G, depth + 1)
            # min search
            if min_or_max and u < util:
                util = u
                best = i
            # max search
            elif (not min_or_max) and u > util:
                util = u
                best = i
            G[i] = 0
    return best, util

def decisionMaker(boardState,minimax,depth):
    action, util = minimax_search(boardState, 0)
    return action
```