k_nearest_neighbor

May 4, 2023

[ ]:
```
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
import pickle
```

citation: Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

1 Data Processing

data from UCI Machine Learning Repositories: https://archive.ics.uci.edu/ml/datasets/SkillCraft1+Master+Table

source: Thompson JJ, Blair MR, Chen L, Henrey AJ (2013) Video Game Telemetry as a Critical Tool in the Study of Complex Skill Learning. PLoS ONE 8(9): e75129.

[ ]:
```
data = pd.read_csv('../data/SkillCraft1_Dataset.csv', na_values='?')
data = data.drop(['GameID'], axis=1)
# data = data[['LeagueIndex', 'Age', 'HoursPerWeek', 'TotalHours', 'APM']]

filtered_data = data[data[data.columns].notnull().all(1)] # filter out any row␣
 ↪that contains missing value
filtered_data
```

[ ]:
```
      LeagueIndex   Age  HoursPerWeek  TotalHours       APM  SelectByHotkeys  \
0               5  27.0          10.0      3000.0  143.7180         0.003515
1               5  23.0          10.0      5000.0  129.2322         0.003304
2               4  30.0          10.0       200.0   69.9612         0.001101
3               3  19.0          20.0       400.0  107.6016         0.001034
4               3  32.0          10.0       500.0  122.8908         0.001136
…             …    …             …          …       …                …
3335            4  20.0           8.0       400.0  158.1390         0.013829
3336            5  16.0          56.0      1500.0  186.1320         0.006951
3337            4  21.0           8.0       100.0  121.6992         0.002956
3338            3  20.0          28.0       400.0  134.2848         0.005424
3339            4  22.0           6.0       400.0   88.8246         0.000844

      AssignToHotkeys  UniqueHotkeys  MinimapAttacks  MinimapRightClicks  \
```

```
0            0.000220              7        0.000110            0.000392
1            0.000259              4        0.000294            0.000432
2            0.000336              4        0.000294            0.000461
3            0.000213              1        0.000053            0.000543
4            0.000327              2        0.000000            0.001329
…               …            …             …                 …
3335         0.000504              7        0.000217            0.000313
3336         0.000360              6        0.000083            0.000166
3337         0.000241              8        0.000055            0.000208
3338         0.000182              5        0.000000            0.000480
3339         0.000108              2        0.000000            0.000341

      NumberOfPACs  GapBetweenPACs  ActionLatency  ActionsInPAC  \
0         0.004849         32.6677        40.8673        4.7508
1         0.004307         32.9194        42.3454        4.8434
2         0.002926         44.6475        75.3548        4.0430
3         0.003783         29.2203        53.7352        4.9155
4         0.002368         22.6885        62.0813        9.3740
…            …             …              …             …
3335      0.003583         36.3990        66.2718        4.5097
3336      0.005414         22.8615        34.7417        4.9309
3337      0.003690         35.5833        57.9585        5.4154
3338      0.003205         18.2927        62.4615        6.0202
3339      0.003099         45.1512        63.4435        5.1913

      TotalMapExplored  WorkersMade  UniqueUnitsMade  ComplexUnitsMade  \
0                   28     0.001397                6               0.0
1                   22     0.001193                5               0.0
2                   22     0.000745                6               0.0
3                   19     0.000426                7               0.0
4                   15     0.001174                4               0.0
…                 …          …              …               …
3335                30     0.001035                7               0.0
3336                38     0.001343                7               0.0
3337                23     0.002014                7               0.0
3338                18     0.000934                5               0.0
3339                20     0.000476                8               0.0

      ComplexAbilitiesUsed
0                 0.000000
1                 0.000208
2                 0.000189
3                 0.000384
4                 0.000019
…                    …
3335              0.000287
3336              0.000388
```

```
3337              0.000000
3338              0.000000
3339              0.000054

[3338 rows x 19 columns]
```

2 Model Construction

[ ]:
```
predict = 'LeagueIndex'

x = np.array(filtered_data.drop([predict], axis=1))
y = np.array(filtered_data[predict])

print('Model Construction\n------------------')
best_acc = 0
for _ in range(10):
    for k in range(3, 14, 2):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

        model = KNeighborsClassifier(n_neighbors=k)

        model.fit(x_train, y_train)
        acc = model.score(x_test, y_test)
        if acc > best_acc:
            best_acc = acc
            with open('../model/8LeagueSkills_KNearestNeighborModel.pickle',␣
 ↪'wb') as f:
                pickle.dump(model, f)
                print(f'New most accurate model ({best_acc}) using {k}␣
 ↪neighbors is saved!')
```

```
Model Construction
------------------
New most accurate model (0.281437125748503) using 3 neighbors is saved!
New most accurate model (0.3592814371257485) using 5 neighbors is saved!
New most accurate model (0.38323353293413176) using 7 neighbors is saved!
New most accurate model (0.41317365269461076) using 13 neighbors is saved!
```

3 Prediction

[ ]:
```
model = open('../model/8LeagueSkills_KNearestNeighborModel.pickle', 'rb')
model: KNeighborsClassifier = pickle.load(model)
```

[ ]:
```
print('\nPredition\n---------')
rank: dict = {1: 'Bronze', 2: 'Silver', 3: 'Gold', 4: 'Platinum', 5: 'Diamond',␣
 ↪6: 'Master', 7: 'GrandMaster', 8: 'Professional'}
```

```
predictions = model.predict(x_test)

for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction]}, Actual: {rank[y_test[i]]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i]]}')
    else:
        break
```

```
Predition
---------
Prediction: Master, Actual: Diamond
Prediction: Master, Actual: Diamond
Prediction: Master, Actual: Master
Prediction: Gold, Actual: Platinum
Prediction: Silver, Actual: Silver
Prediction: Master, Actual: Master
Prediction: Master, Actual: Master
Prediction: Bronze, Actual: Silver
Prediction: Gold, Actual: Gold
Prediction: Silver, Actual: Silver
```

3.1 4 Leagues Categorization

[ ]:
```
fourLeague_data = filtered_data.copy()

rank: dict = {1: 'Bronze-Silver', 2: 'Gold-Platinum', 3: 'Diamond-Master', 4:␣
 ↪'GrandMaster-Professional'}
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 2
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 2
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 3
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 4
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 4
fourLeague_data.loc[fourLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 4

predict = 'LeagueIndex'

x = np.array(fourLeague_data.drop([predict], axis=1))
y = np.array(fourLeague_data[predict])

print('Model Construction\n------------------')
best_acc = 0
```

```
for _ in range(10):
    for k in range(3, 14, 2):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

        model = KNeighborsClassifier(n_neighbors=k)

        model.fit(x_train, y_train)
        acc = model.score(x_test, y_test)
        if acc > best_acc:
            best_acc = acc
            with open('../model/4LeagueSkills_KNearestNeighborModel.pickle',␣
 ↪'wb') as f:
                pickle.dump(model, f)
                print(f'New most accurate model ({best_acc}) using {k}␣
 ↪neighbors is saved!')

print('\nPredition\n---------')
predictions = model.predict(x_test)

for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction]}, Actual: {rank[y_test[i]]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i]]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.47305389221556887) using 3 neighbors is saved!
New most accurate model (0.47604790419161674) using 5 neighbors is saved!
New most accurate model (0.48502994011976047) using 7 neighbors is saved!
New most accurate model (0.5089820359281437) using 9 neighbors is saved!
New most accurate model (0.5688622754491018) using 13 neighbors is saved!

Predition
---------
Prediction: Diamond-Master, Actual: Diamond-Master
Prediction: Gold-Platinum, Actual: Gold-Platinum
Prediction: GrandMaster-Professional, Actual: GrandMaster-Professional
Prediction: Gold-Platinum, Actual: Gold-Platinum
Prediction: Bronze-Silver, Actual: Bronze-Silver
Prediction: Gold-Platinum, Actual: Gold-Platinum
Prediction: Diamond-Master, Actual: GrandMaster-Professional
Prediction: Bronze-Silver, Actual: Bronze-Silver
Prediction: Gold-Platinum, Actual: Bronze-Silver
```

```
Prediction: Gold-Platinum, Actual: Gold-Platinum
```

3.2 3 Leagues Categorization

[ ]:
```
threeLeague_data = filtered_data.copy()

rank: dict = {1: 'Bronze-Silver-Gold', 2: 'Platinum-Diamond-Master', 3:␣
 ↪'GrandMaster-Professional'}
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 1
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 2
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 3
threeLeague_data.loc[threeLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 3

predict = 'LeagueIndex'

x = np.array(threeLeague_data.drop([predict], axis=1))
y = np.array(threeLeague_data[predict])

print('Model Construction\n------------------')
best_acc = 0
for _ in range(10):
    for k in range(3, 14, 2):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

        model = KNeighborsClassifier(n_neighbors=k)

        model.fit(x_train, y_train)
        acc = model.score(x_test, y_test)
        if acc > best_acc:
            best_acc = acc
            with open('../model/3LeagueSkills_KNearestNeighborModel.pickle',␣
 ↪'wb') as f:
                pickle.dump(model, f)
                print(f'New most accurate model ({best_acc}) using {k}␣
 ↪neighbors is saved!')

print('\nPredition\n---------')
predictions = model.predict(x_test)

for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction]}, Actual: {rank[y_test[i]]}')
        except KeyError:
```

```
            print(f'Prediction: Unknown, Actual: {rank[y_test[i]]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.7574850299401198) using 3 neighbors is saved!
New most accurate model (0.7784431137724551) using 9 neighbors is saved!
New most accurate model (0.7994011976047904) using 11 neighbors is saved!
New most accurate model (0.8143712574850299) using 5 neighbors is saved!
New most accurate model (0.8263473053892215) using 9 neighbors is saved!

Predition
---------
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Bronze-Silver-Gold
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Bronze-Silver-Gold, Actual: Bronze-Silver-Gold
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Bronze-Silver-Gold
Prediction: Bronze-Silver-Gold, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
Prediction: Platinum-Diamond-Master, Actual: Platinum-Diamond-Master
```

3.3 2 Leagues Categorization

[ ]:
```
twoLeague_data = filtered_data.copy()

rank: dict = {1: 'Bronze-Silver-Gold-Platinum', 2:␣
 ↪'Diamond-Master-GrandMaster-Professional'}
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 2, 'LeagueIndex'] = 1
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 3, 'LeagueIndex'] = 1
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 4, 'LeagueIndex'] = 1
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 5, 'LeagueIndex'] = 2
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 6, 'LeagueIndex'] = 2
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 7, 'LeagueIndex'] = 2
twoLeague_data.loc[twoLeague_data['LeagueIndex'] == 8, 'LeagueIndex'] = 2

predict = 'LeagueIndex'

x = np.array(twoLeague_data.drop([predict], axis=1))
y = np.array(twoLeague_data[predict])

print('Model Construction\n------------------')
best_acc = 0
for _ in range(10):
```

```
    for k in range(3, 14, 2):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

        model = KNeighborsClassifier(n_neighbors=k)

        model.fit(x_train, y_train)
        acc = model.score(x_test, y_test)
        if acc > best_acc:
            best_acc = acc
            with open('../model/3LeagueSkills_KNearestNeighborModel.pickle',␣
 ↪'wb') as f:
                pickle.dump(model, f)
                print(f'New most accurate model ({best_acc}) using {k}␣
 ↪neighbors is saved!')

print('\nPredition\n---------')
predictions = model.predict(x_test)

for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction]}, Actual: {rank[y_test[i]]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i]]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.7125748502994012) using 3 neighbors is saved!
New most accurate model (0.7754491017964071) using 5 neighbors is saved!
New most accurate model (0.7844311377245509) using 3 neighbors is saved!
New most accurate model (0.7964071856287425) using 7 neighbors is saved!
New most accurate model (0.8023952095808383) using 5 neighbors is saved!
New most accurate model (0.8143712574850299) using 9 neighbors is saved!
New most accurate model (0.8203592814371258) using 13 neighbors is saved!

Predition
---------
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Bronze-Silver-Gold-
Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Diamond-Master-GrandMaster-
Professional
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
```

```
GrandMaster-Professional
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
Prediction: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
Prediction: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
```