linear_regression

May 5, 2023

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

import plotly.express as px
from jupyter_dash import JupyterDash
from dash import html, dcc, Input, Output
```

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	
…	…	…	…	…	…	…	
3334	4	20.0	8.0	400.0	158.1390	0.013829	
3335	5	16.0	56.0	1500.0	186.1320	0.006951	
3336	4	21.0	8.0	100.0	121.6992	0.002956	

```
3337           3  20.0          28.0       400.0  134.2848         0.005424
3338           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
…                 …              …               …                   …
3334         0.000504              7        0.000217            0.000313
3335         0.000360              6        0.000083            0.000166
3336         0.000241              8        0.000055            0.000208
3337         0.000182              5        0.000000            0.000480
3338         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
…              …               …              …             …
3334      0.003583         36.3990        66.2718        4.5097
3335      0.005414         22.8615        34.7417        4.9309
3336      0.003690         35.5833        57.9585        5.4154
3337      0.003205         18.2927        62.4615        6.0202
3338      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
…                  …           …               …                 …
3334                30     0.001035                7               0.0
3335                38     0.001343                7               0.0
3336                23     0.002014                7               0.0
3337                18     0.000934                5               0.0
3338                20     0.000476                8               0.0

      ComplexAbilitiesUsed
0                 0.000000
1                 0.000208
2                 0.000189
3                 0.000384
```

```
4                 0.000019
…                    …
3334              0.000287
3335              0.000388
3336              0.000000
3337              0.000000
3338              0.000054

[3337 rows x 19 columns]
```

2 Model Contruction

[ ]:
```python
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(100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

    linear = linear_model.LinearRegression()
    # linear = linear_model.Ridge(alpha=0.5)
    # linear = linear_model.Lasso(alpha=0.1)

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

```
Model Construction
------------------
New most accurate model (0.5925175062770893) is saved!
New most accurate model (0.5943494208897688) is saved!
New most accurate model (0.5958461161426049) is saved!
New most accurate model (0.5964122977257302) is saved!
New most accurate model (0.5993649266597412) is saved!
```

3 Prediction

[ ]:
```
model = open('../model/8LeagueSkills_LinearModel.pickle', 'rb')
linear: linear_model.LinearRegression = 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 = linear.predict(x_test)
for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction.round(0)]}, Actual:␣
 ↪{rank[y_test[i].round(0)]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i].round(0)]}')
    else:
        break
```

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

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
```

```
fourLeague_data
```

[ ]:

	LeagueIndex	Age	HoursPerWeek	TotalHours	APM	SelectByHotkeys	\
0	3	27.0	10.0	3000.0	143.7180	0.003515	
1	3	23.0	10.0	5000.0	129.2322	0.003304	
2	2	30.0	10.0	200.0	69.9612	0.001101	
3	2	19.0	20.0	400.0	107.6016	0.001034	
4	2	32.0	10.0	500.0	122.8908	0.001136	
…	…	…	…	…	…	…	
3334	2	20.0	8.0	400.0	158.1390	0.013829	
3335	3	16.0	56.0	1500.0	186.1320	0.006951	
3336	2	21.0	8.0	100.0	121.6992	0.002956	
3337	2	20.0	28.0	400.0	134.2848	0.005424	
3338	2	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	
…	…	…	…	…	
3334	0.000504	7	0.000217	0.000313	
3335	0.000360	6	0.000083	0.000166	
3336	0.000241	8	0.000055	0.000208	
3337	0.000182	5	0.000000	0.000480	
3338	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	
…	…	…	…	…	
3334	0.003583	36.3990	66.2718	4.5097	
3335	0.005414	22.8615	34.7417	4.9309	
3336	0.003690	35.5833	57.9585	5.4154	
3337	0.003205	18.2927	62.4615	6.0202	
3338	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	

```
…                 …          …                 …                 …
3334              30      0.001035              7              0.0
3335              38      0.001343              7              0.0
3336              23      0.002014              7              0.0
3337              18      0.000934              5              0.0
3338              20      0.000476              8              0.0

      ComplexAbilitiesUsed
0                 0.000000
1                 0.000208
2                 0.000189
3                 0.000384
4                 0.000019
…                      …
3334              0.000287
3335              0.000388
3336              0.000000
3337              0.000000
3338              0.000054

[3337 rows x 19 columns]
```

[ ]:
```
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(100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

    linear = linear_model.LinearRegression()

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

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

predictions = linear.predict(x_test)
```

```
for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction.round(0)]}, Actual:␣
 ↪{rank[y_test[i].round(0)]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i].round(0)]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.5088846113585362) is saved!
New most accurate model (0.5400602612280612) is saved!
New most accurate model (0.5424050971919598) is saved!
New most accurate model (0.5644345591594921) is saved!
New most accurate model (0.5788300999465199) is saved!
New most accurate model (0.6141835924610872) is saved!

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

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(100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

    linear = linear_model.LinearRegression()

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

print('\nPredition\n---------')
predictions = linear.predict(x_test)
for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction.round(0)]}, Actual:␣
 ↪{rank[y_test[i].round(0)]}')
        except KeyError:
            print(f'Prediction: Unknown, Actual: {rank[y_test[i].round(0)]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.410722637071642) is saved!
New most accurate model (0.42101056413598503) is saved!
New most accurate model (0.42466878521904095) is saved!
New most accurate model (0.4260166803959079) is saved!
New most accurate model (0.42891908061117534) is saved!
New most accurate model (0.4503484274540769) is saved!

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

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

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(100):
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.1)

    linear = linear_model.LinearRegression()

    linear.fit(x_train, y_train)
    acc = linear.score(x_test, y_test)
    if acc > best_acc:
        best_acc = acc
        with open('../model/2LeagueSkills_LinearModel.pickle', 'wb') as f:
            pickle.dump(linear, f)
        print(f'New most accurate model ({best_acc}) is saved!')

print('\nPredition\n---------')
predictions = linear.predict(x_test)
for i, prediction in enumerate(predictions):
    if i < 10:
        try:
            print(f'Prediction: {rank[prediction.round(0)]}, Actual:␣
 ↪{rank[y_test[i].round(0)]}')
        except KeyError:
```

```
            print(f'Prediction: Unknown, Actual: {rank[y_test[i].round(0)]}')
    else:
        break
```

```
Model Construction
------------------
New most accurate model (0.3549680912069525) is saved!
New most accurate model (0.35898329117617056) is saved!
New most accurate model (0.40697280976209327) is saved!
New most accurate model (0.4272821544312735) is saved!
New most accurate model (0.44325507428509003) is saved!
New most accurate model (0.4760365514408649) 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: Diamond-Master-GrandMaster-Professional, Actual: Diamond-Master-
GrandMaster-Professional
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: Bronze-Silver-Gold-Platinum, Actual: Bronze-Silver-Gold-Platinum
```

4 Visualization Tool

[ ]:
```
app = JupyterDash(__name__)

app.layout = html.Div([
    html.Div([
        html.Div([
            dcc.Dropdown(
                options=list(filtered_data.columns),
                value='LeagueIndex',
                id='y-axis'),
            dcc.RadioItems(
                options=['linear', 'log'],
                value='linear',
```

```
                id='y-axis-type',
                inline=True)],
            style={'width': '48%', 'display': 'inline-block'}),
        html.Div([
            dcc.Dropdown(
                options=list(filtered_data.columns),
                value='APM',
                id='x-axis'),
            dcc.RadioItems(
                options=['linear', 'log'],
                value='linear',
                id='x-axis-type',
                inline=True)],
            style={'width': '48%', 'float': 'right', 'display': 'inline-block'})
    ]),
    dcc.Graph(id='graph', figure={}),
])

@app.callback(
    Output('graph', 'figure'),
    Input('y-axis', 'value'),
    Input('y-axis-type', 'value'),
    Input('x-axis', 'value'),
    Input('x-axis-type', 'value'))
def update_graph(y_axis, y_axis_type, x_axis, x_axis_type):
    fig = px.scatter(
        filtered_data,
        y=y_axis if y_axis_type == 'linear' else np.
 ↪log10(filtered_data[y_axis]),
        x=x_axis if x_axis_type == 'linear' else np.
 ↪log10(filtered_data[x_axis]),
        color='LeagueIndex',
        color_continuous_scale=px.colors.sequential.Rainbow,
        title="Custom Graph Generator")
    return fig

# Run app and display result inline in the notebook
app.run_server(mode='inline')
```

```
Dash is running on http://127.0.0.1:8050/


<IPython.lib.display.IFrame at 0x152d0de40>

/Users/next/opt/anaconda3/envs/tensorflow/lib/python3.10/site-
packages/pandas/core/arraylike.py:402: RuntimeWarning:

divide by zero encountered in log10
```

```
/Users/next/opt/anaconda3/envs/tensorflow/lib/python3.10/site-
packages/pandas/core/arraylike.py:402: RuntimeWarning:

divide by zero encountered in log10

/Users/next/opt/anaconda3/envs/tensorflow/lib/python3.10/site-
packages/pandas/core/arraylike.py:402: RuntimeWarning:

divide by zero encountered in log10

/Users/next/opt/anaconda3/envs/tensorflow/lib/python3.10/site-
packages/pandas/core/arraylike.py:402: RuntimeWarning:

divide by zero encountered in log10

/Users/next/opt/anaconda3/envs/tensorflow/lib/python3.10/site-
packages/pandas/core/arraylike.py:402: RuntimeWarning:

divide by zero encountered in log10
```