Fair-IRT_Law_show

October 14, 2024

[1]:

```
import numpy as np
import pandas as pd
import matplotlib.colors as mcolors
import seaborn as sns

from irt import Beta3

from matplotlib import rcParams
import matplotlib.pyplot as plt
```

[2]:

```
pij = pd.read_csv('./Law_Pij.csv')
pij.set_index(pij.columns[0], inplace=True)

random_seed = 42
pij = pij.sample(n=1000, random_state=random_seed)
```

[3]:

```
original_shape = pij.values.shape
array = pij.values.flatten()

data = np.where(array > 1, 1, array)
normalized_array = data.reshape(original_shape)
normalized_df = pd.DataFrame(normalized_array, index=pij.index, columns=pij.
 ↪columns)
```

[4]:

```
normalized_df = 1 - normalized_df
```

[5]:

```
def ICC_function(abilities, difficulties, discriminations):
    a = ((1-abilities)/ abilities)
    b = (difficulties / (1-difficulties))
    c = a*b
    d = c**discriminations
    return (1 / (d+1))
```

[6]:

```
b4 = Beta3(
        learning_rate=100,
        epochs=10000,
        n_respondents=normalized_df.shape[1],
        n_items=normalized_df.shape[0],
        n_workers=-1,
        random_seed=1,
    )
b4.fit(normalized_df.values)
```

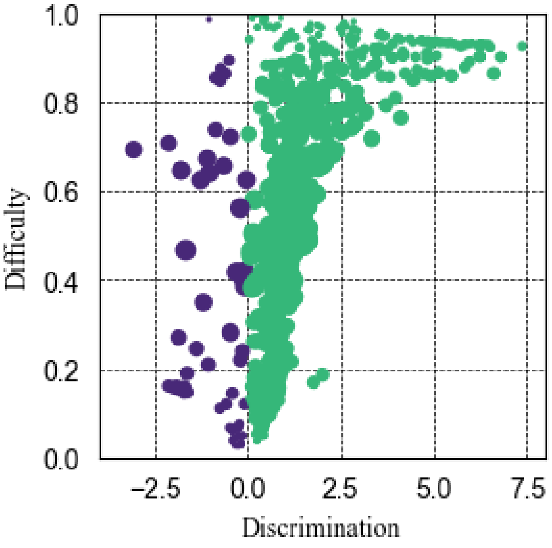
```
100%|      | 10000/10000 [00:16<00:00, 606.70it/s]
```

```
[6]: <irt.Beta3 at 0x343cab3d0>
```

```
[7]: new_pij = pd.DataFrame(index=range(1000), columns=range(48))

     for i in range(1000):
         for j in range(15):
             alpha = (b4.abilities[j] / b4.difficulties[i]) ** b4.discriminations[i]
             beta_val = ((1 - b4.abilities[j]) / (1 - b4.difficulties[i])) ** b4.
      ↪discriminations[i]
             new_pij.iloc[i, j] = (alpha)/(alpha+beta_val)
```

```
[9]: def plot_discriminations_difficulties(discriminations, difficulties,␣
      ↪normalized_df, font_size=10, font_ann_size=5, base_point_size=500):
         rcParams['font.family'] = 'serif'
         rcParams['font.serif'] = ['Times New Roman']

         sns.set_style('whitegrid')
         fig, ax = plt.subplots(figsize=(3, 3))
         ax.grid(color='black', linestyle='--', linewidth=0.5)

         ax.spines['bottom'].set_color('black')
         ax.spines['left'].set_color('black')
         ax.spines['right'].set_color('black')
         ax.spines['top'].set_color('black')
         ax.xaxis.label.set_color('black')
         ax.yaxis.label.set_color('black')
         ax.tick_params(axis='x', colors='black')
         ax.tick_params(axis='y', colors='black')

         point_sizes = base_point_size * (1 - np.abs(difficulties - 0.5) * 2)

         colors = []
         for disc, diff in zip(discriminations, difficulties):
             base_color = '#482878' if disc < 0 else '#35b779'
             intensity = 1
             color = mcolors.to_rgba(base_color, intensity)
             colors.append(color)

         scatter = ax.scatter(discriminations, difficulties, s=point_sizes, c=colors)
```

```
    plt.xlabel(r'Discrimination', fontsize=font_size, family='Times New Roman',␣
 ↪color='black')
    plt.ylabel(r'Difficulty', fontsize=font_size, family='Times New Roman',␣
 ↪color='black')

    ax.set_ylim(0, 1)

    if discriminations.min() < 0:
        x_min = int(discriminations.min()) - 1
    else:
        x_min = int(discriminations.min())

    if discriminations.max() > 0:
        x_max = int(discriminations.max()) + 1
    else:
        x_max = int(discriminations.max())

    ax.set_xlim(x_min, x_max)
    plt.savefig("Figure5a.pdf", format="pdf", bbox_inches='tight')
    plt.show()

plot_discriminations_difficulties(b4.discriminations, b4.difficulties, new_pij,␣
 ↪font_size=10, font_ann_size=8, base_point_size=50)
```



[10]:

```
fairness_model = normalized_df.apply(np.mean,axis=0).to_numpy()
```

[11]:

```
def f(theta,delta_j,a_j):
    term1 = (delta_j / (1 - delta_j)) ** a_j
    term2 = (theta / (1 - theta)) ** (-a_j - 1)
    numerator = a_j * term1 * term2
    denominator = (1 + term1 * (theta / (1 - theta)) ** -a_j) ** 2
    return numerator / denominator * (1 / (1 - theta) ** 2)
```

[12]:

```
abilities = np.linspace(0.001, 0.999, 1000)
```

[13]:

```
def create_abilities_fairness_table(name, abilities, fairness_model):
    df = pd.DataFrame({
        'Model': name,
        'Ability': abilities,
        'STS': fairness_model
    })

    return df

df = create_abilities_fairness_table(pij.columns, b4.abilities, fairness_model)
print(df)
```

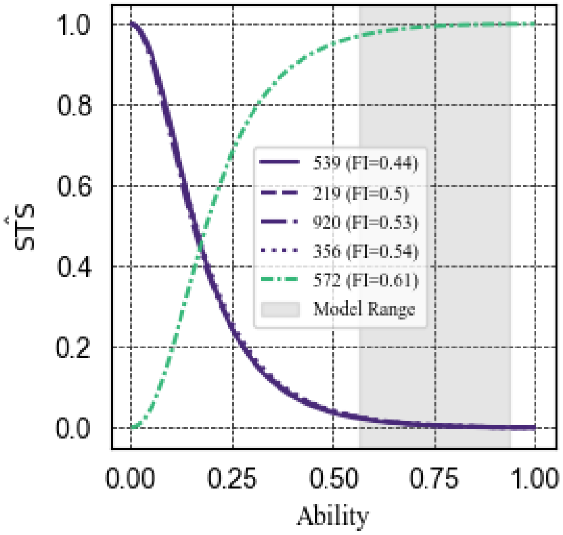
```
     Model   Ability       STS
0    GBM_1  0.904201  0.701359
1    GBM_2  0.899290  0.668577
2     DP_1  0.910154  0.683917
3    GBM_3  0.898711  0.681167
4    GBM_4  0.887553  0.649616
5    GBM_5  0.923858  0.757141
6    GBM_6  0.866921  0.618298
7    GLM_7  0.907184  0.661066
8    GBM_8  0.880953  0.643186
9    GBM_9  0.862184  0.636399
10    DP_2  0.911078  0.676245
11  GBM_10  0.834613  0.591383
12  GBM_11  0.564415  0.387305
13   DRF_1  0.611914  0.424349
14   XRT_1  0.934038  0.750745
```

[14]:

```
min_ability = np.min(b4.abilities)
max_ability = np.max(b4.abilities)

plt.figure(figsize=(3, 3))
plt.rcParams["font.family"] = "Times New Roman"

sns.set_style('whitegrid')
fig, ax = plt.subplots(figsize=(3, 3))
ax.grid(color='black', linestyle='--', linewidth=0.5)
ax.spines['bottom'].set_color('black')
ax.spines['left'].set_color('black')
ax.spines['right'].set_color('black')
ax.spines['top'].set_color('black')
ax.xaxis.label.set_color('black')
ax.yaxis.label.set_color('black')
ax.tick_params(axis='x', colors='black')
ax.tick_params(axis='y', colors='black')

markers = ['o', 's', 'D', '^', 'v', 'P']
num_markers = len(markers)
colors = ['red', 'blue', 'green', 'orange', 'purple', 'brown']
num_colors = len(colors)
linestyles = ['-', '--', '-.', ':', (0, (3, 1, 1, 1)),
              (0, (5, 1)), (0, (5, 10)), (0, (1, 1)),
              (0, (3, 5, 1, 5)), (0, (3, 10, 1, 10))]
num_linestyles = len(linestyles)
i = 0
j = 0
id = 0
list = [0.25,0.11,0.02,0.76,0.86]

added_labels = set()

for index in [538,218,919,355,571]:
    total_f_theta = 0

    for i in range(b4.abilities.shape[0]):
        f_theta = abs(f(b4.abilities[i], b4.difficulties[index], b4.
 ↪discriminations[index]))
        total_f_theta += f_theta

    linestyle = linestyles[j % num_linestyles]
    fairness_2 = ICC_function(abilities, b4.difficulties[index], b4.
 ↪discriminations[index])
    if b4.discriminations[index]>0:
        plt.plot(abilities, fairness_2, label=f'{index+1}␣
 ↪(FI={round(total_f_theta, 2)})',color='#35b779', linestyle=linestyle)
    else:
        plt.plot(abilities, fairness_2, label=f'{index+1}␣
 ↪(FI={round(total_f_theta, 2)})',color='#482878', linestyle=linestyle)
    j += 1
    id += 1

plt.fill_betweenx(np.arange(-0.05, 1.15, 0.1), min_ability, max_ability,␣
 ↪color='gray', alpha=0.2, label='Model Range')
```

```
plt.xlim(-0.05, 1.05)
plt.ylim(-0.05, 1.05)


plt.gca().set_aspect('equal', adjustable='box')

legend = plt.legend(loc='upper left', fontsize=7, bbox_to_anchor=(0.3, 0.7), ncol=1)

for text in legend.get_texts():
    text.set_fontname('Times New Roman')
    text.set_color('black')

plt.xlabel(r'Ability', fontsize=10, family='Times New Roman')
plt.ylabel(r'$\hat{\mathrm{STS}}$', fontsize=10, family='Times New Roman')

plt.grid(True)
plt.savefig("Figure5b.pdf", format="pdf", bbox_inches='tight')
plt.show()
```

```
<Figure size 300x300 with 0 Axes>
```



[15]:

```
from irt_special import Beta3
```

[16]:

```
def ICC_function_special(abilities, difficulties):
    a = ((1-abilities)/ abilities)
    b = (difficulties / (1-difficulties))
    c = a*b
    d = c
    return (1 / (d+1))
```

[17]:

```
b4_special = Beta3(
        learning_rate=100,
        epochs=10000,
        n_respondents=normalized_df.shape[1],
        n_items=normalized_df.shape[0],
        n_workers=-1,
        random_seed=1,
    )
b4_special.fit(normalized_df.values)
```

```
100%|     | 10000/10000 [00:12<00:00, 772.48it/s]
```

[17]:

```
<irt_special.Beta3 at 0x3447d0df0>
```

[18]:

```
new_pij = pd.DataFrame(index=range(1000), columns=range(24))

for i in range(1000):
    for j in range(15):
        alpha = (b4_special.abilities[j] / b4_special.difficulties[i])
        beta_val = ((1 - b4_special.abilities[j]) / (1 - b4_special.
  ↪difficulties[i]))
        new_pij.iloc[i, j] = (alpha)/(alpha+beta_val)
```

[19]:

```
def generate_table(b4_special, new_pij, index):
    delta_j_values = b4_special.difficulties[index]
    Delta_j_values = delta_j_values / (1 - delta_j_values)
    log_Delta_j_values = np.log(Delta_j_values)

    print(f"log_Delta_j_values (for difficulty at index {index}):␣
  ↪{round(log_Delta_j_values, 3)}")

    results = []

    for i in range(b4_special.abilities.shape[0]):
        theta_i_values = b4_special.abilities[i]
        Theta_i_values = (1 - theta_i_values) / theta_i_values
        log_Telta_j_values = np.log(Theta_i_values)
        res = np.log(1 - new_pij[i][index]) - np.log(new_pij[i][index])

        results.append({
            "log_Telta_j_values": round(log_Telta_j_values, 3),
            "res": round(res, 3)
        })

    df = pd.DataFrame(results)
    return df

generate_table(b4_special, new_pij, 538)
```

```
log_Delta_j_values (for difficulty at index 538): 4.438
```

```
[19]:     log_Telta_j_values    res
      0              -1.737  2.701
      1              -1.586  2.852
      2              -1.670  2.769
      3              -1.585  2.853
      4              -1.409  3.030
      5              -2.361  2.078
      6              -1.106  3.333
      7              -1.567  2.872
      8              -1.315  3.124
      9              -1.244  3.194
      10             -1.524  2.914
      11             -0.897  3.541
      12              0.469  4.908
      13              0.298  4.736
      14             -2.461  1.977
```

```
[ ]:
```