eval

May 16, 2024

1 Analysis of Optimal 2c and 3c Contracts

Table of Contents - Setup - Conflict Deal vs. Optimal Contracts - Gains of Granularization of Consent and Content Resolutions - Characteristics of Contracts - honesty / prefs manigpp.»???

1.1 Setup

In this section dependencies are imported and the csv file generated by the JS script is loaded into a pandas dataframe. Each row is based on a variation of user's and site's preferences for each issue (not on a resolution level!). For each combination, different contracts were calculated, based on a systematic variation of resolutions for consent and content. For a given combination of issue preferences, multiple contracts were calculated, varying in consent resolutions (2, 3, 6) or content resolutions (2/4). For each contract the score is included as a column.

Assumptions - Preferences for resolutions (e.g., analytics or 2 EUR) remain same as defined in descriptive analysis

Overview of variables

Variable	Description	Example
`u_cost_rel`	The relevance score a user gave the issue 'cost'	0.2
`u_consent_rel`	The relevance score a user gave the issue 'consent'	0.5
`u_content_rel`	The relevance score a user gave the issue 'content'	0.3
`s_cost_rel`	The relevance score a site gave the issue 'cost'	0.2
`s_consent_rel`	The relevance score a site gave the issue 'consent'	0.5
`s_content_rel`	The relevance score a site gave the issue 'content'	0.3
`score_default`	The score of the conflict or default deal (cost=0, consent=none, content=50%)	1000
`score_binary_2c`	The score of the optimal 2c contract (restricted to binary consent and content)	6000
`score_binary_3c`	The score of the optimal 3c contract (restricted to binary consent and content)	6000
`score_CCC_2c`	The score of the optimal 2c contract. CCC (Cost, Consent, Content) in numbers from 2 to 5 express the number of resolutions. E.g. a 5 Cost options, 3 Consent options and 2 Content options result in a CCC of 532	6000
`score_CCC_3c`	The score of the optimal 3c contract. Same as above but for 3c contracts	6000

Variable	Description	Example
`consent_CCC_2c`	The consent granted for the 2c contract of CCC	analytics personal-izedAds
`consent_CCC_3c`	The consent granted for the 3c contract of CCC	analytics personal-izedAds
`cost_CCC`	The agreed cost of 3c contract CCC	2.5

[ ]:
```python
#imports
import pandas as pd

# load and present dataset
df = pd.read_csv('./output.csv')

# csv creating script always puts one unnamed last column. removed here
df.drop(df.columns[-1], axis=1, inplace=True)

# Top 5 Rows for overview of columns and values
df.head()
```

[ ]:
```
   u_cost_rel  u_consent_rel  u_content_rel  s_cost_rel  s_consent_rel  \
0         0.0            0.1            0.9         0.0            0.1
1         0.0            0.1            0.9         0.0            0.2
2         0.0            0.1            0.9         0.0            0.3
3         0.0            0.1            0.9         0.0            0.4
4         0.0            0.1            0.9         0.0            0.5

   s_content_rel  score_default  score_binary_2c  score_binary_3c  \
0            0.9           1969             9000             9000
1            0.8           1750             9000             9000
2            0.7           1531             9000             9000
3            0.6           1313             9000             9000
4            0.5           1094             9000             9000

   score_562_2c  …  score_524_2c  score_524_3c  score_564_2c  score_564_3c  \
0          9025  …          9000          9000          9025          9025
1          9000  …          9000          9000          9000          9000
2          9000  …          9000          9000          9000          9000
3          9000  …          9000          9000          9000          9000
4          9000  …          9000          9000          9000          9000

                                    consent_562_2c  \
0                     personalizedAds identification
1  analytics marketing personalizedContent person…
2  analytics marketing personalizedContent person…
```

```
3  analytics marketing personalizedContent person…
4  analytics marketing personalizedContent person…

                              consent_532_2c  \
0                              personalizedAds
1  analytics personalizedAds externalContent
2  analytics personalizedAds externalContent
3  analytics personalizedAds externalContent
4  analytics personalizedAds externalContent

                                      consent_562_3c  \
0                      personalizedAds identification
1  analytics marketing personalizedContent person…
2  analytics marketing personalizedContent person…
3  analytics marketing personalizedContent person…
4  analytics marketing personalizedContent person…

                              consent_532_3c cost_562 cost532
0                              personalizedAds        1       1
1  analytics personalizedAds externalContent        1       1
2  analytics personalizedAds externalContent        1       1
3  analytics personalizedAds externalContent        1       1
4  analytics personalizedAds externalContent        1       1

[5 rows x 23 columns]
```

1.2 Conflict Deal vs. Optimal Contracts

Objective: - Are all Nash contracts (2c or 3c) better (higher scores) than the default/conflict deal?

Input Parameters: - `score_default` - `nash-score`: `score_binary_2c`, `score_binary_3c`, `score_CCC_2c`, `score_CCC_3c`

Metrics: - gain $= \frac{\text{score\_default}}{\text{nash-score}}$

```
[ ]: # Any contract should be better than the default/conflict deal
score_columns = ['score_binary_2c', 'score_binary_3c', 'score_562_2c',␣
 ↪'score_562_3c',
                 'score_532_2c', 'score_532_3c', 'score_524_2c', 'score_524_3c',
                 'score_564_2c', 'score_564_3c']

# Find the minimum value in each row for the selected columns
min_score = df[score_columns].min(axis=1)

# Calculate the result by dividing each minimum value by the corresponding␣
 ↪value in the 'score_default' column
gain_ratios = min_score / df['score_default']
```

```
# Calc descriptive statistics
gain_ratios.describe().to_frame()
```

Observations - Any optimal contract (no matter if 2c or 3c) is better than the conflict deal (0 EUR, Rejected All, 50% Content) - Median gain is ~550% - Minimal gain is >200%

2 Gains of Granularization of Consent and Content Resolutions

As seen in the descriptive analysis, consent and content are mostly resolved binarily in accept/reject all or full/restricted access. The proposed protocol allows a granularization of these resolutions with six consent options (analytics, marketing, personalizedAds, personalizedContent, externalContent, identification) and any content resolution in percent of all available content. This section compares the binary contracts with the more granular options and answers, if the granularization leads to better deals.

Objective: - Does the granularization of consent and content resolutions lead to better deals compared to binary resolutions? - Which combination of issue relevancies create the best deals?

Input Parameters | Input Parameter | Description | Type | |——————————|——————————————|————| | `Binary Scores` | Scores from binary contracts | Number | | `Granular Scores` | Scores from granular contracts | Number |

Metrics

Metric	Description	Type	Column
`Absolute Difference (points)`	granular score - binary score	Category of 0, 1to100, 101to500, 501to1000, 1001to2000, greaterThan2000	abs_CCC_2c, abs_CCC_3c
`Relative Difference (%)`	absolute difference / binary score	Decimal Number	rel_CCC_2c, rel_CCC_3c

2.0.1 Preparation

Calculating and adding the necessary columns to the dataframe.

```
[ ]: # Copy for gains analysis
gains_df = df.iloc[:, :-7]
# Create additional columns
gains_df['diff_562_2c'] = gains_df['score_562_2c'] - gains_df['score_binary_2c']
gains_df['diff_562_3c'] = gains_df['score_562_3c'] - gains_df['score_binary_3c']

gains_df['diff_532_2c'] = gains_df['score_532_2c'] - gains_df['score_binary_2c']
gains_df['diff_532_3c'] = gains_df['score_532_3c'] - gains_df['score_binary_3c']

gains_df['diff_524_2c'] = gains_df['score_524_2c'] - gains_df['score_binary_2c']
```

```python
gains_df['diff_524_3c'] = gains_df['score_524_3c'] - gains_df['score_binary_3c']

gains_df['diff_564_2c'] = gains_df['score_564_2c'] - gains_df['score_binary_2c']
gains_df['diff_564_3c'] = gains_df['score_564_3c'] - gains_df['score_binary_3c']

# function to calculate the absolute difference range
def get_diff_abs(diff):
    if diff == 0:
        return "0"
    elif 0 < diff <= 100:
        return "1to100"
    elif 100 < diff <= 500:
        return "101to500"
    elif 500 < diff <= 1000:
        return "501to1000"
    elif 1000 < diff <= 2000:
        return "1001to2000"
    elif diff > 2000:
        return "greaterThan2000"
    else:
        return "invalid"

# Calculate absolute difference ranges
gains_df['abs_562_2c'] = gains_df['diff_562_2c'].apply(get_diff_abs)
gains_df['abs_562_3c'] = gains_df['diff_562_3c'].apply(get_diff_abs)

gains_df['abs_532_2c'] = gains_df['diff_532_2c'].apply(get_diff_abs)
gains_df['abs_532_3c'] = gains_df['diff_532_3c'].apply(get_diff_abs)

gains_df['abs_524_2c'] = gains_df['diff_524_2c'].apply(get_diff_abs)
gains_df['abs_524_3c'] = gains_df['diff_524_3c'].apply(get_diff_abs)

gains_df['abs_564_2c'] = gains_df['diff_564_2c'].apply(get_diff_abs)
gains_df['abs_564_3c'] = gains_df['diff_564_3c'].apply(get_diff_abs)

# function to calculate the relative difference
def get_diff_rel(diff):
    return
# Calculate the relative difference
#df['diff_2c_rel'] = round(df['diff_2c']/df['score_binary_2c'],3)
#df['diff_3c_rel'] = round(df['diff_3c']/df['score_binary_3c'],3)
gains_df['rel_562_2c'] = round(gains_df['diff_562_2c']/
 ↪gains_df['score_binary_2c'],3)
gains_df['rel_562_3c'] = round(gains_df['diff_562_3c']/
 ↪gains_df['score_binary_3c'],3)
```

```
gains_df['rel_532_2c'] = round(gains_df['diff_532_2c']/
 ↪gains_df['score_binary_2c'],3)
gains_df['rel_532_3c'] = round(gains_df['diff_532_3c']/
 ↪gains_df['score_binary_3c'],3)

gains_df['rel_524_2c'] = round(gains_df['diff_524_2c']/
 ↪gains_df['score_binary_2c'],3)
gains_df['rel_524_3c'] = round(gains_df['diff_524_3c']/
 ↪gains_df['score_binary_3c'],3)

gains_df['rel_564_2c'] = round(gains_df['diff_564_2c']/
 ↪gains_df['score_binary_2c'],3)
gains_df['rel_564_3c'] = round(gains_df['diff_564_3c']/
 ↪gains_df['score_binary_3c'],3)


# Display the DataFrame (which will be rendered as a table)
gains_df.head()
```

2.0.2 Absolute Gains

Results summarized in a table holding contracts and the counts per category.

```
[ ]: abs_gains_df = pd.DataFrame()

abs_gains_df['abs_distribution'] = ['0', '1to100', '101to500', '501to1000',␣
 ↪'1001to2000', "greaterThan2000"]

# Iterate over each column name
for column_name in ['abs_562_2c', 'abs_562_3c', 'abs_532_2c', 'abs_532_3c',␣
 ↪'abs_524_2c', 'abs_524_3c', 'abs_564_2c', 'abs_564_3c',  ]:
    values = gains_df[column_name].value_counts().values
    if (values.size == 1):
        values = [values[0], 0, 0 ,0, 0, 0]
    elif(values.size == 5):
        values = values[0].append(0)

    abs_gains_df[column_name] = values

# Display the resulting DataFrame
abs_gains_df
```

Observations: - most negotiations gained zero points - 236 to 261 have small gains from one to 100 points - 171 to 200 have gains from 100 to 500 points - very little (~13) have gains between 1000 and 2000 - an outlier seems to exist in every case - abs_524_2c abs_564_2c are empty - abs_564_2c abs_564_3c are same as abs_562_2c abs_562_3c

Interpretation: - granularization has a small impact on the contract scores - Both user and site prefer the content resolution of 100% (unrestricetd access). Because there's no actual space for trade-offs, the best contract always contains the best content option. - Granularization of content options is pointless

2.0.3 Relative Gains

Results summarized in a table holding contracts and the counts per category.

```
[ ]: # distribution of absolute differences

rel_gains_df = pd.DataFrame()

rel_gains_df['rel_distribution'] = ['Mean', 'Standard Deviation', 'Minimum',␣
 ↪'25th Percentile', 'Median', "75th Percentile", "90th Percentile", "95th␣
 ↪Percentile", "Maximum"]

# Iterate over each column name
for column_name in ['rel_562_2c', 'rel_562_3c', 'rel_532_2c', 'rel_532_3c',␣
 ↪'rel_524_2c', 'rel_524_3c', 'rel_564_2c', 'rel_564_3c',  ]:
    mean = round(gains_df[column_name].mean(), 3)
    std = round(gains_df[column_name].std(), 3)
    min_value = round(gains_df[column_name].min(), 3)
    q25 = round(gains_df[column_name].quantile(0.25), 3)
    median = round(gains_df[column_name].median(), 3)
    q75 = round(gains_df[column_name].quantile(0.75), 3)
    q90 = round(gains_df[column_name].quantile(0.90), 3)
    q95 = round(gains_df[column_name].quantile(0.95), 3)
    max_value = round(gains_df[column_name].max(), 3)

    # add column
    rel_gains_df[column_name] = [mean, std, min, q25, median, q75, q90, q95,␣
 ↪max]

# Display the resulting DataFrame
rel_gains_df
```

Observations: - First improvements observed in the 75th percentile - All seem to have the same maximum outlier, probably leveraging the mean - rel_524_2c, rel_524_3c show no improvement - min ~10% improvement in 95th percentile

Since all seem to have the same outlier, the following table provides insights into the respective user and site prefernces:

```
[ ]: # Relation between prioritization of issues (relevancies) and relative gains

# Which combination creates the outlier of over 200% relative gains?
gains_df[gains_df['rel_562_2c'] > 2][['u_cost_rel', 'u_consent_rel',␣
 ↪'u_content_rel', 's_cost_rel', 's_consent_rel', 's_content_rel']]
```

Observations: - The outlier is based on extreme interest in consent

Interpretation: - When both user and site only care about privacy/consent and not about content or cost, this happens

2.0.4 Impact of Relevancies on Relative Gains

To determine, which combinations of relevancies lead to the highest relative gains, the relevancies are summarized in one column per user and site.

- L: relevance <0.3
- M: relevance >=0.3 & <0.5
- H: relevance >=0.5

For example - input: cost (0.2), consent (0.7), content (0.1) - output: LHL

[ ]:
```
# Create columns to analyze combinations of issue relevancies
def map_relevancy_dec_to_category(value):
    if value < 0.3:
        return 'L'
    elif value <= 0.5:
        return 'M'
    else:
        return 'H'

def map_relevancies_to_string(cost, consent, content):
    cost_string = map_relevancy_dec_to_category(cost)
    consent_string = map_relevancy_dec_to_category(consent)
    content_string = map_relevancy_dec_to_category(content)
    return cost_string + consent_string + content_string

def map_user_relevancies_to_string(row):
    return map_relevancies_to_string(row['u_cost_rel'], row['u_consent_rel'],␣
 ↪row['u_content_rel'])

def map_sites_relevancies_to_string(row):
    return map_relevancies_to_string(row['s_cost_rel'], row['s_consent_rel'],␣
 ↪row['s_content_rel'])


gains_df['u_issue_rel'] = gains_df.apply(map_user_relevancies_to_string, axis=1)
gains_df['s_issue_rel'] = gains_df.apply(map_sites_relevancies_to_string,␣
 ↪axis=1) # doesnt yet exist.

# Print head for verification
gains_df.head()
```

Users

```
[ ]: # Which combination of user relavancies creates the highest (90th percentile)␣
      ↪relative gains?
     # EXCLUDE: rel_524_2c, rel_524_3c, because no gains
     rel_columns = ['rel_562_2c', 'rel_562_3c', 'rel_532_2c', 'rel_532_3c',␣
      ↪'rel_564_2c', 'rel_564_3c']

     result_df = pd.DataFrame()

     for column_name in rel_columns:
         # Calculate the 90th percentile value
         top_ten_percentile_value = gains_df[column_name].quantile(0.9)

         top_percentile_df = gains_df[gains_df[column_name] >␣
      ↪top_ten_percentile_value][['u_issue_rel']]

         counts = top_percentile_df['u_issue_rel'].value_counts()

         result_df[column_name] = counts

     result_df
```

Observations: - LHL, LHM, LMM and MML are the most common combinations of user relevancies - LMH and MMM are already far less common - in the most and secondmost common (LHL, LHM) consent preferences are H (high) or at least 0.5 - in the most common combinations, consent preferences are always >L or M/H

Interpretation: - The top four combinations occur for extremly privacy concerned users, that care more about their consent than about cost and content combined

Sites

```
[ ]: # Which combination of sites relavancies creates the highest (90th percentile)␣
      ↪relative gains?
     # EXCLUDE: rel_524_2c, rel_524_3c, because no gains
     rel_columns = ['rel_562_2c', 'rel_562_3c', 'rel_532_2c', 'rel_532_3c',␣
      ↪'rel_564_2c', 'rel_564_3c']

     result_df = pd.DataFrame()

     for column_name in rel_columns:
         top_ten_percentile_value = gains_df[column_name].quantile(0.9)

         top_percentile_df = gains_df[gains_df[column_name] >␣
      ↪top_ten_percentile_value][['s_issue_rel']]

         counts = top_percentile_df['s_issue_rel'].value_counts()

         result_df[column_name] = counts
```

```
result_df
```

Observations: - as for the user LHL, LHM, LMM and MML are the most common combinations of user relevancies - the other combinations are less common - in the most common combinations, consent preferences are always >L or M/H

Interpretation: - The top four combinations occur when a site is way more interested in consent, than granting content or getting to some cost

3 Characteristics of Contracts

Objective: - Do the optimal contracts rely heavily on the same resolutions? - (Privacy Friendliness) How often does the contract require a user to consent to all six consent resolutions

Input Parameters -> content irrelevant -> 2 or 4 doesnt make a differences -> binary consent is anyways not allowed in the protocol!! => analyze for 6 and 3 consent resolutions - `consent_562_2cs`
- `consent_532_2c`
- `consent_562_3c`
- `consent_532_3c`

Metrics count of categories

```
[ ]: # 'consent_562_2c', 'consent_532_2c', 'consent_562_3c', 'consent_532_3c',␣
     ↪'cost_562', 'cost532',
     print(df['consent_562_2c'].value_counts().head())
     print(df['consent_532_2c'].value_counts().head())
     print(df['consent_562_3c'].value_counts().head())
     print(df['consent_532_3c'].value_counts().head())

     print(df['cost_562'].value_counts().head())
     print(df['cost532'].value_counts().head())
```