piflib

Release development

Confidential Computing Team

Sep 08, 2022

CONTENTS

The *piflib* implements several personal information factors. A personal information factor (PIF) provides a way to quantify privacy risks associated with data release.

CHAPTER

ONE

TABLE OF CONTENTS

1.1 Introduction

There are competing interests when it comes to data release:

- Privacy: The privacy of an individual's data must be preserved.
- Utility: The consumer of the data expects it to be complete and accurate.

In reality, these interests are often mutually exclusive. A complete and accurate dataset contains enough information to identify individuals. Thus, a compromise has to be found that sits somewhere in between.

However, due to the lack of fine-grained metrics data custodians often err on the side of caution as privacy breaches have serious consequences. This leads to datasets of limited use, or even data not being release at all.

1.2 What is a PIF

Risk can be divided into two parts:

- The likelihood of a breach
- And the severity (or consequences) of the breach.

In the context of data release, the likelihood of a breach is the chance of an attack succeeding, and the severity is the amount of information an attacker can gain. The different PIFs are attempts to quantify these risks.

1.2.1 Attacker Model

This risk cannot be evaluated in isolation, as personal information accumulates in the public space. In fact, most recent data breaches utilized externally available personal data to de-identify individuals.

We model this from an attacker's point of view. The attacker's aim is to gain more information about individuals in the dataset. Note that this is different to identifying the row associated with a specific individual. Narrowing down the possible rows for an individual might already allow the attacker to reduce the uncertainty around some of the individual's personal information.

The attacker knows the distribution of the feature values for the population. This is not unrealistic, as many summary statistics are freely available (e.g. census). In order to get the true risk, we model the worst case as this represent the highest risk. In terms of auxiliary data, the worst case is that the attacker already knows everything about a target person but one value.

Our personal information factors aim to quantify the risk for that specific value, given that the attacker know all other values of the corresponding individual.

1.3 Personal Information Factors

The personal information factors compute a risk value for each cell, thus providing a detailed risk landscape for the whole dataset. We believe that these risk values allow a data analyst to

- better understand the risk associated with the release of a dataset
- identify the areas in the dataset that need the most attention
- evaluate different treatments to identify the most suited one.

The cell information gain (CIG) quantifies the information that can be learned by an attacker, whereas the cell surprise factor (CSF) quantifies the likelihood of an attack succeeding.

1.4 Cell Information Gain

For simplicity, it is assumed that every cell belongs to a row, and every individual is represented by exactly one row. The CIG quantifies the information gained by learning the value of a cell, given that one already knows all the other cell values of this individual.

We use entropy to measure information. The entropy of a random variable is the average level of uncertainty inherent in the variable's possible outcomes.

As the attacker already has an expectation of the distribution (prior) of that variable, we define the CIG as the change in entropy (or KL-divergence) between the prior and posterior distribution.

1.4.1 Example

Consider the following dataset

Gender	Eye color	Occupation
male	blue	dentist
female	blue	dentist
male	green	accountant
male	green	accountant

For the sake of exposition, we focus on the feature 'Gender'.

First, we need the 'Gender's prior distribution. There are 51% males and 49% females in the Australian population. Then 0.51 and 0.49 form the prior distribution.

Feature	Prior Distribution
male	0.51
female	0.49

Knowing values for 'Eye color' and 'Occupation' gives context. The posterior distribution of 'Gender' is the conditional distribution given its context.

The posterior of 'Gender' is given by P(Gender|Eye color, Occupation) as follows:

Eye color	Occupation	P(Gender=male|Eye color, Occupation)	P(Gender=female|Eye color, Occupation)
blue	dentist	0.5	0.5
green	accountant	1	0

We can see that the posterior distribution for the blue-eyed dentists is very similar to the population prior. As the distribution of ‘Gender’ within the cohort of blue-eyed dentists is essentially the same as the population prior, we associate little risk with ‘Gender’ values for this cohort. The posterior distribution of ‘Gender’ for the green-eyed accountants on the other hand is significantly different from the prior. Thus there is more information to be gained.

1.5 Weighted Cell Information Gain

The CIG assumes that the unknown feature is independent from all other features. Features containing personal information often have inter-dependencies, e.g. firstnames and gender, or postcode and income.

One can look at the CIG as the worst-case scenario. Nothing of the uncertainty of the unknown feature can be explained by the other features.

The Weighted Cell Information Gain explores the best-case scenario: we assume that all observed correlations are due to causal dependencies between the features.

Let X_j be the unknown feature. Then $H(X_j)$, the entropy of feature j, describes the amount of information contained in that feature. The conditional entropy $H\left(X_j \mid X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_m\right)$ describes the amount of information contained in feature j, given that all other feature values are known (taking all possible correlations into account).

Dividing the conditional entropy by the entropy of the feature, we get a factor that describes what fraction of the information in a feature can not be explained by the correlations with all other features.

$$w_j = \frac{H\left(X_j \mid X_1, \ldots, X_{j-1}, X_{j+1}, \ldots, X_m\right)}{H(X_j)},$$

The wCIG is defined as the CIG value multiplied by factor w_j.

$$wCIG(i,j) = w_j * CIG(i,j)$$

1.5.1 Caution

Correlation does not mean causation. A trivial counterexample is the following dataset:

A	B
a	b
c	r
f	e

There is a perfect correlation between feature A and B, thus all wCIG values are zero. However, there is no causal relationship between the two. In fact, the CIG values are quite high, as all values are unique.

1.6 Cell Surprise Factor

Whereas the CIG measures try to quantify the consequences of a breach, the Cell surprise factor (CSF) focuses on the likelihood of a breach.

The different values of the known features form cohorts. The CIG describes the change in entropy of the unknown feature. Thus, all cell values of the unknown features within the same cohort get assigned the same CIG value, irrespective of how much the actual cell value contributed to the change in entropy.

We keep the same setting as with the CIG, assume we know all values of an entity but one. This gives us a prior and posterior distribution. But instead of measuring the change of entropy of the unknown feature, we now look at the change of probability for each value of the unknown feature separately.

1.6.1 Difference to CIG

Whereas the CIG quantifies the difference in information of a unknown cell value, the CSF measures the change in its probability. The CIG measure the difference of two distributions, the CSF the difference of two probabilities. There is a nice visualization of this in the CSF tutorial in the *Tutorials* section.

The CSF is a measure of how unexpected a specific value is, given its context. Whereas the CIG measure how much information is contained in that value within its contextual cohort.

1.7 Summarizing Personal Information Factors

The proposed personal information factors provide a detailed risk landscape across the whole dataset. Sometimes it is useful to have single number representing the overall risk. However, summarizing values removes some of the details, e.g.: an average removes the extreme values.

1.7.1 Feature Information Gain

FIG is given by summing all CIG values for each feature. The FIG can be used to identify the features that provides the highest information gain. Higher information gain corresponds to higher risk of of re-identification. The risk of inclusion can be compared to the feature's utility when making the decision to include or exclude it.

1.7.2 Row Information Gain

RIG is determined by summing all the CIG values in the row and is a measure of the information gain associated with a particular individual if their information is revealed through re-identification.

1.7.3 PIF$_n$

The initial definition of the PIF was a summary of the individual row information gain values. The PIF$_n$ is defined as the nth percentile of the individual RIG values of a dataset. E.g.: 95% of the RIG values of a dataset won't exceed the PIF$_{95}$ value. Note that the RIG measures the overall information gain for an individual.

Keep in mind that a PIF$_n$ value only summarises the risk of $n\%$ of the individuals in the dataset.

1.7.4 Caution

We believe that single value summaries are too simplistic and should be used with caution.

1.8 Tutorials

1.8.1 running the tutorials

The notebooks can run online using binder.

Or you can download the tutorials from github. The dependencies are listed in *tutorials-requirements.txt*. Install and start Jupyter from the `docs/tutorials` directory:

```
pip install -r tutorials-requirements.txt
python -m jupyter lab
```

Finally, you can view a static version of the tutorials here.

Personal Information Factor (PIF) - Computing the cell information gain (CIG)

The PIF tries to answer the question:

"Knowing everything about a person but one feature's value, what's the information one would gain learning that value?"

The information gain is a measure of how unexpected the value is. The higher the information gain, the more unusual the value is, given the values for all remaining features.

We compute the information gain as the KL-divergence between the distribution of values of the whole dataset (the features' priors) and the distribution of a feature's values given all remaining features' values (posterior).

[12]:
```
import collections
import matplotlib.pyplot as plt
import matplotlib
import pandas as pd
import numpy as np
import seaborn as sns

import piflib

from tutorial_helpers import horizontal_bar_plot
```

Example dataset

We define a toy dataset to explain the process. Feel free to modify the dataset and examine the behaviour of the corresponding CIG values.

[13]:
```
data = {'gender': (['male'] * 6)+['female'],
        'name': ['Anton', 'Bill', 'Charlie', 'Don', 'Emil', 'Emil', 'Charlie'],
        'eye_color': ['blue', 'green', 'green', 'green', 'blue', 'green', 'green']}
df = pd.DataFrame(data)
df
```

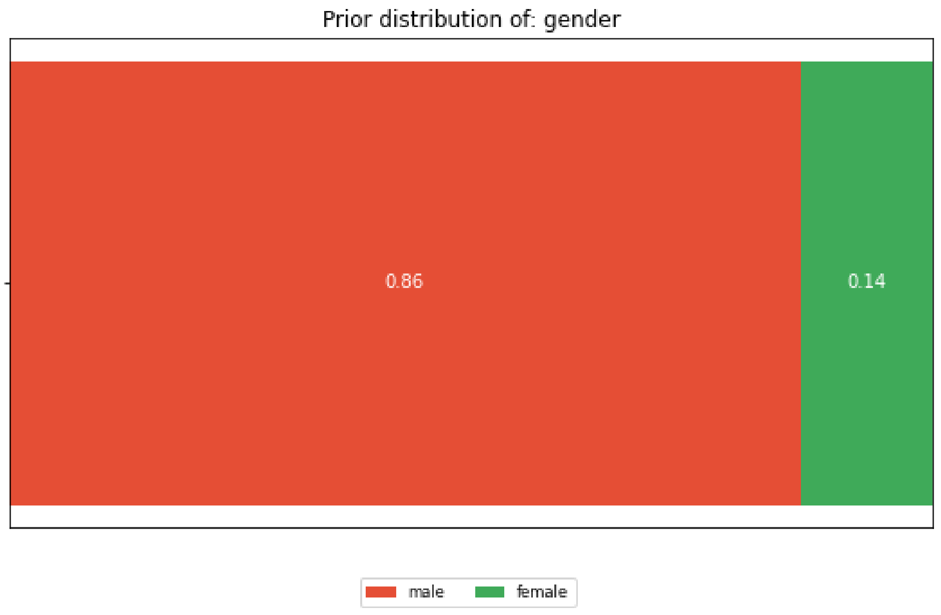
[13]:
```
   gender     name eye_color
0    male    Anton      blue
1    male     Bill     green
2    male  Charlie     green
3    male      Don     green
4    male     Emil      blue
5    male     Emil     green
6  female  Charlie     green
```

The features' priors

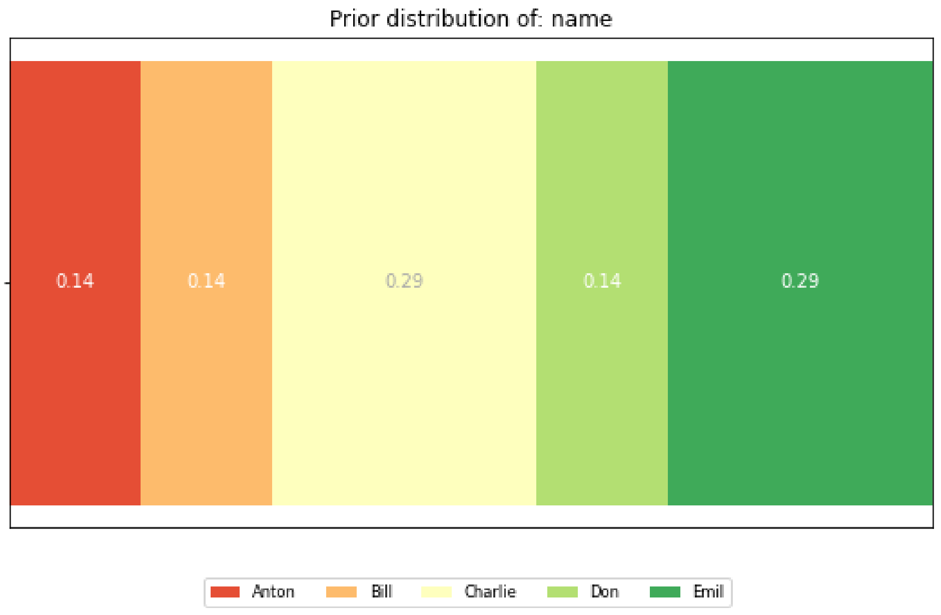
Looking at the dataset as a whole, what is the distribution of the values of each feature. Not having any information about a person, this is what we expect him/her to look like.

```
[14]: from piflib.data_util import calculate_distribution
```

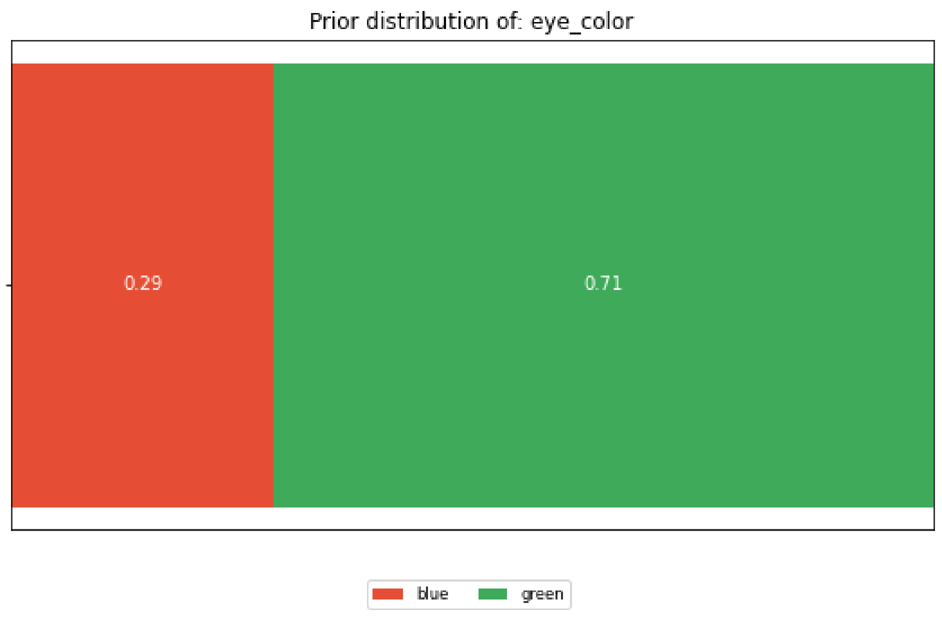
```
[15]: for feature in df.columns:
          dist = calculate_distribution(df[feature])
          horizontal_bar_plot({'': list(dist.values())}, dist.keys())
          plt.title(f'Prior distribution of: {feature}')
```



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

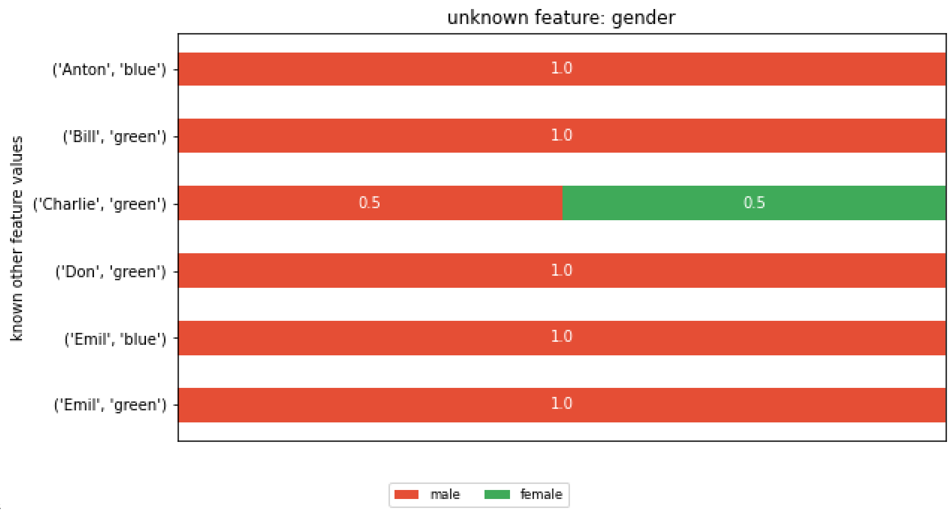


nbsphinx-code-borderwhite

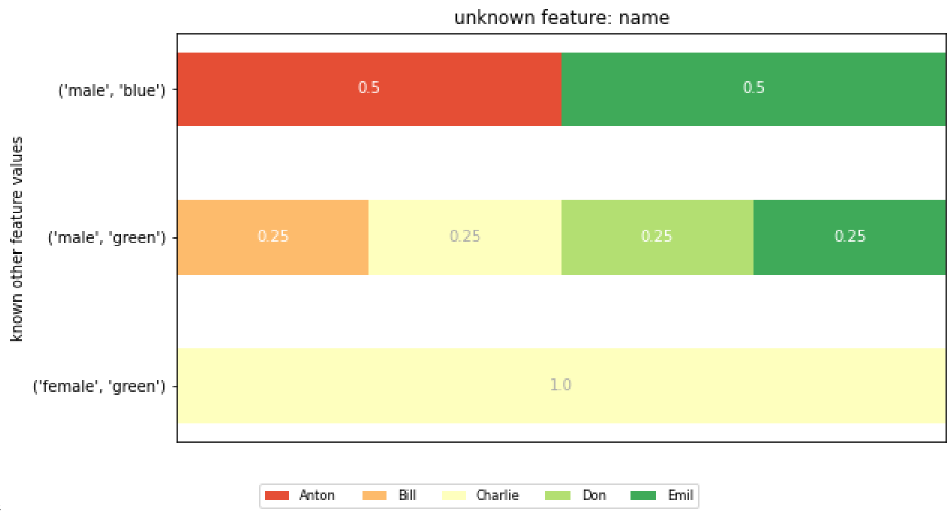
The posterior distributions

To compute a feature's posterior distribution, we have to take its context into account. For example, in the given dataset, there are two posterior distributions for the feature 'name'. One where the pair gender and eye color is "male, green" and one for "male, blue".

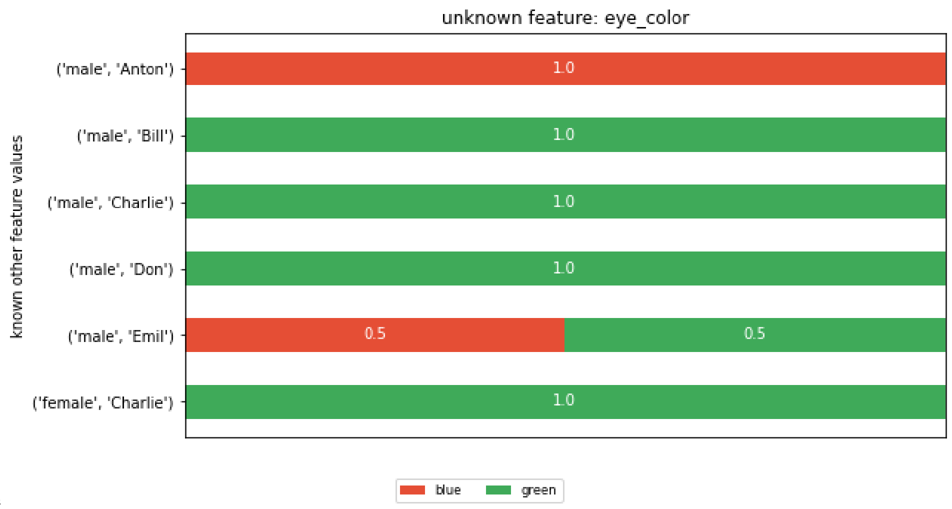
```
[16]: for feature in df.columns:
          known_features = tuple(col_name for col_name in df.columns if col_name != feature)
          bucket = collections.defaultdict(list)
          bucket_map = []
          for idx, row in df.iterrows():
              key = tuple(row[known_feature] for known_feature in known_features)
              bucket[key].append(row[feature])
              bucket_map.append(key)

          bucket_distributions = {key: calculate_distribution(el_bucket) for key, el_bucket in␣
      ↪bucket.items()}
          feature_vals = df[feature].unique()
          dists = {}
          for key, distribution in bucket_distributions.items():
              dists[str(key)] = [distribution.get(feature_val, 0) for feature_val in feature_
      ↪vals]

          horizontal_bar_plot(dists, feature_vals)
          plt.title(f'unknown feature: {feature}')
          plt.ylabel('known other feature values')
          plt.show()
```



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

The CIG values:

Given a features' prior and posterior distributions, one can compute the KL-divergence between the two for each cell in the dataset. This will form the CIG value.

```
[17]: piflib.compute_cigs(df).round(2)
```

```
[17]:    gender  name  eye_color
      0    0.22  1.31       1.81
      1    0.22  0.31       0.49
      2    0.51  0.31       0.49
      3    0.22  0.31       0.49
      4    0.22  1.31       0.15
      5    0.22  0.31       0.15
      6    0.51  1.81       0.49
```

Looking at both CIG values that correspond to a 'blue' eye color, we can see that they got assigned very different CIG values. In row 0 the CIG is 1.81, whereas in row 4 the CIG is 0.15. To understand the difference, you have to appreciate the different cohorts that fed into the CIG computation. The cohort (gender=male, name=Anton) that forms the posterior in row 0 is of size 1, whereas the cohort for row 4 (gender=male, name=Emil) is of size 2. The eye_color distribution of the second cohort is similar to the prior, thus the CIG value is low.

Or looking at it from an attacker perspective, if I know that the target's name is Anton with a male gender, I would learn his eye color. In contrast, if the target's name is Emil with a male gender, the attacker is left with a 50/50 chance for blue and green. This is close to his prior believe of 29/71.

I also want to draw attention to line 1. Why is the CIG different, given that here too, the cohort size is 1? This is explained by the prior believe. The attacker believes that 71% of the people in the dataset have green eyes. Thus, learning that Bill has green eyes is less of a surprise than learning that Anton has blue eyes.

Going bigger - hackathon dataset

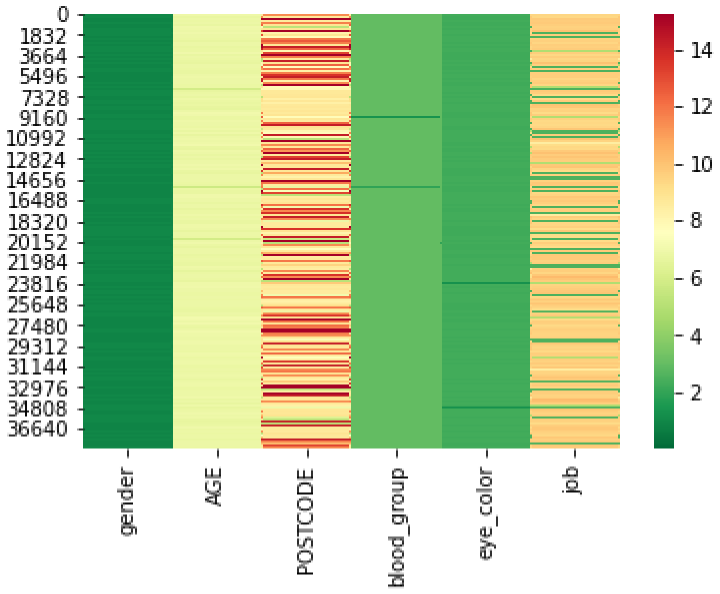
```
[18]: hack_features = ['gender', 'AGE', 'POSTCODE', 'blood_group', 'eye_color', 'job']
      hack_data = pd.read_csv('data/hackathon.csv')[hack_features]
      hack_data = hack_data.fillna('Unemployed')
      hack_data.head()
```

```
[18]:   gender  AGE  POSTCODE blood_group eye_color                        job
      0      F   99      2649          B-     Brown  Psychologist, counselling
      1      M  108      1780          A-     Hazel          Personnel officer
      2      M   59      2940          B+     Hazel            Tourism officer
      3      M   58      2945          B+      Blue                       Make
      4      M   30      2729         AB-     Brown    Forest/woodland manager
```

We now compute the CIG values and display them as a heatmap, with colors ranging from green for 'save' values to red for the most at risk values.

```
[19]: hack_cig = piflib.compute_cigs(hack_data)
      color_map = matplotlib.colors.ListedColormap(
              sns.color_palette("RdYlGn", 256).as_hex()[::-1])
      sns.heatmap(hack_cig, cmap=color_map)
```

```
[19]: <AxesSubplot:>
```



nbsphinx-code-borderwhite

Looking at the distribution of CIG values, we can see that the changes are quite substantial.

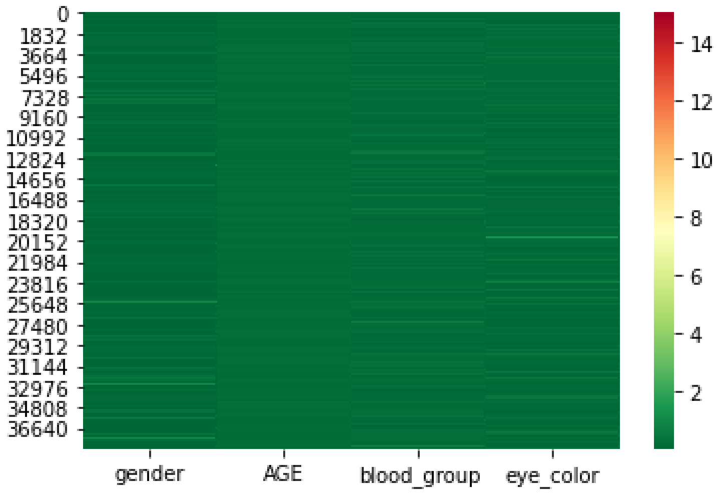
```
[20]: hack_cig.describe()
```

```
[20]:              gender           AGE      POSTCODE   blood_group     eye_color  \
count  38462.000000  38462.000000  38462.000000  38462.000000  38462.000000
mean       0.996891      6.827838     10.058929      2.987481      2.313486
std        0.068946      0.223181      2.583055      0.121209      0.093568
min        0.000887      4.029292      4.754066      0.667983      0.730567
25%        0.950303      6.788203      8.572935      2.979960      2.311166
50%        0.950303      6.860459      8.821755      2.996329      2.324443
75%        1.051470      6.904717     11.909218      3.011675      2.330845
max        1.051470     11.909218     15.231146      3.023827      2.332356

                job
count  38462.000000
mean       8.306507
std        2.667701
min        2.606579
25%        9.300409
50%        9.530706
75%        9.707584
max       10.707584
```

Let’s try to reduce the CIG values by removing the features ‘job’ and ‘POSTCODE’. Removing features will lead to larger cohort sizes for the posterior distributions. Alternatively, you could also

```
[21]: cols = ['gender', 'AGE', 'blood_group', 'eye_color']
sub_hack_cig = piflib.compute_cigs(hack_data[cols])
sns.heatmap(sub_hack_cig, cmap=color_map, vmax=15)
```

```
[21]: <AxesSubplot:>
```



nbsphinx-code-borderwhite

[22]:
```
sub_hack_cig.describe()
```

[22]:
```
             gender           AGE   blood_group     eye_color
count  38462.000000  38462.000000  38462.000000  38462.000000
mean       0.094231      0.181759      0.164312      0.147753
std        0.157085      0.028178      0.106338      0.122034
min        0.000114      0.129416      0.015208      0.000030
25%        0.008432      0.161430      0.094626      0.063763
50%        0.040053      0.180967      0.142773      0.121803
75%        0.099452      0.197749      0.211684      0.190624
max        1.051470      0.253509      3.023827      2.332356
```

This already looks a lot better. The mean CIG values are a lot lower now. However, there are still some rows in the dataset with high CIG values. These rows still stand out and thus have a higher risk of re-identification.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib

from piflib.data_util import calculate_distribution
from tutorial_helpers import horizontal_bar_plot, compute_posterior_distributions
import piflib
```

Cell surprise factor (CSF) as a measure of information risk

The cell information gain (CIG), as introduced in the ACS data sharing white paper, offers a novel way to reason about the personal information contained in a dataset. By measuring the personal information factor on a cell level, an analyst is able to quickly locate the areas in the dataset that need the most attention.

However, we identified two problems with this approach and propose a new measure to address those issues.

Problem 1: overcounting

The CIG is defined as the KL-divergence of the prior distribution of the feature and the posterior of that feature, given all other features. This means though that the CIG value is a measure of surprise of how the distribution of the values changed, i.e., a combination of **all** the values in the posterior distribution. Thus, the CIG is not really a fair representation of the risk of the single cell value, but rather represents the risk of that feature for the cohort comprising that particular posterior distribution.

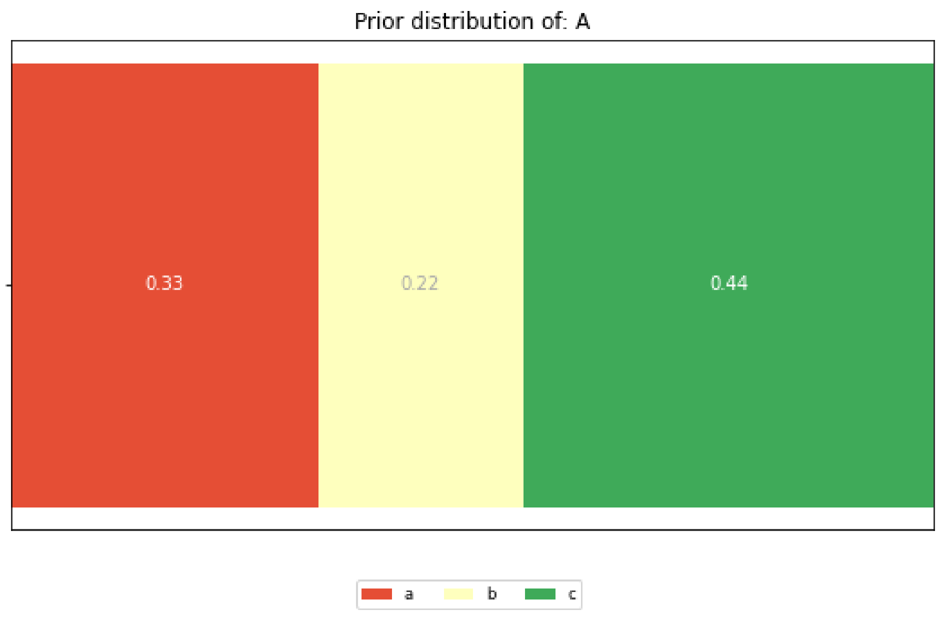
For example, consider the following dataset:

```
[2]: data = {'A': ['a', 'a', 'a', 'b', 'b', 'c', 'c', 'c', 'c'],
             'B': ['g', 'h', 'i', 'g', 'g', 'h', 'h', 'h', 'i']}
     df = pd.DataFrame(data)
     df
```

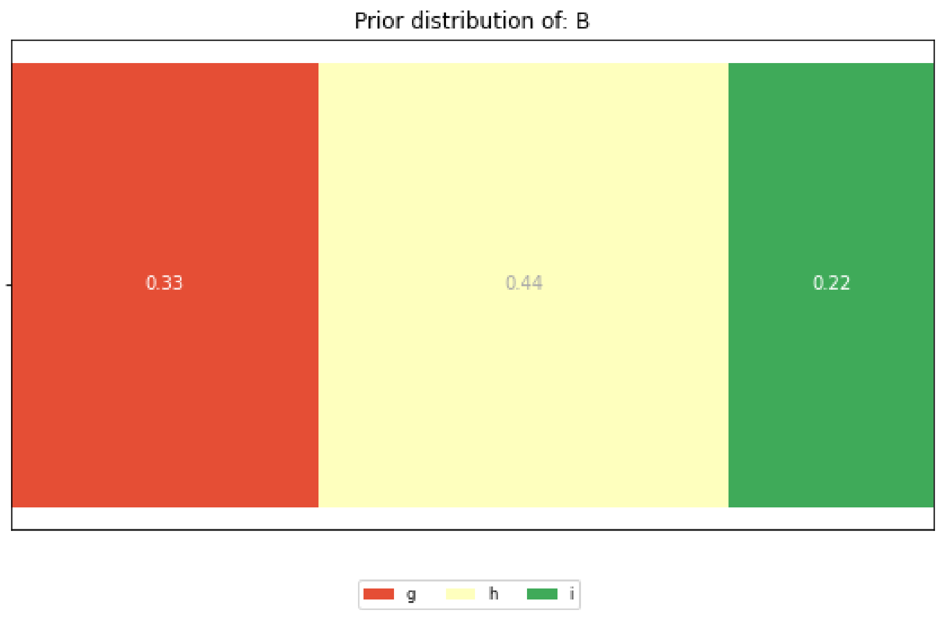
```
[2]:    A  B
     0  a  g
     1  a  h
     2  a  i
     3  b  g
     4  b  g
     5  c  h
     6  c  h
     7  c  h
     8  c  i
```

The prior distributions for the features ‘A’ and ‘B’ look like this:

```
[3]: for feature in df.columns:
         dist = calculate_distribution(df[feature])
         horizontal_bar_plot({'': list(dist.values())}, dist.keys())
         plt.title(f'Prior distribution of: {feature}')
```



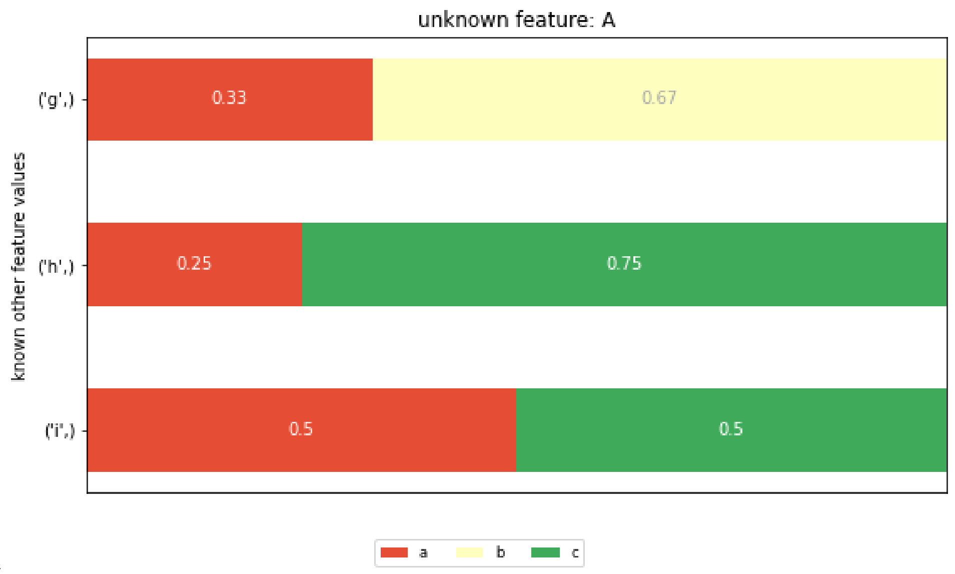
nbsphinx-code-borderwhite



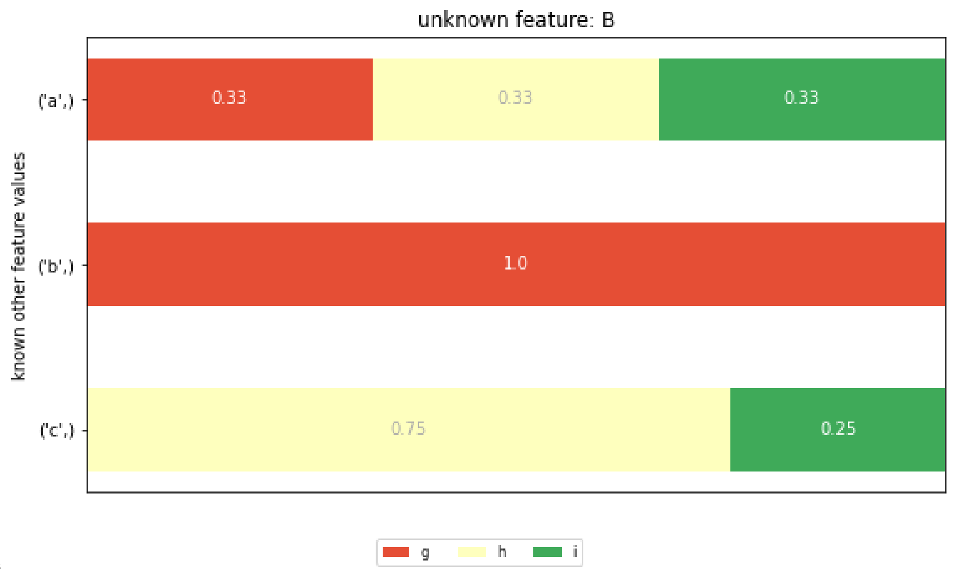
nbsphinx-code-borderwhite

And these are the posterior distributions:

```
[4]: for feature in df.columns:


         horizontal_bar_plot(*compute_posterior_distributions(feature, df))
         plt.title(f'unknown feature: {feature}')
         plt.ylabel('known other feature values')
         plt.show()
```



nbsphinx-code-borderwhite



nbsphinx-code-borderwhite

The CIG value for row 0, feature 'A' is computed as the KL divergence between the prior distribution for feature 'A' and the posterior distribution for the unknown feature 'A' and given value 'g' (the first row in the posterior distribution graph). The KL divergence is defined as a weighted sum of the logarithmic differences of **all** corresponding elements in prior and posterior distribution. This leads to two potential problems: - The CIG value measures the element of surprise of the **combined** set of possible cell values in the posterior distribution, thus overestimating the real element of surprise of a single cell value. - All cell values of a feature with the same posterior distribution get assigned the same CIG value, irrespective of their individual change in probability.

Problem 2: unbounded CIG values

The KL divergence has no upper bound. This makes it hard to find an acceptance threshold for which to consider the data to be safe.

Cell Surprise Factor (CSF), an alternative measure of surprise

The underlying idea of the CIG is to measure the element of surprise of a cell value when its context changes – suppose we know every cell value of a person but one, how surprising is this value compared to what we previously though about it?

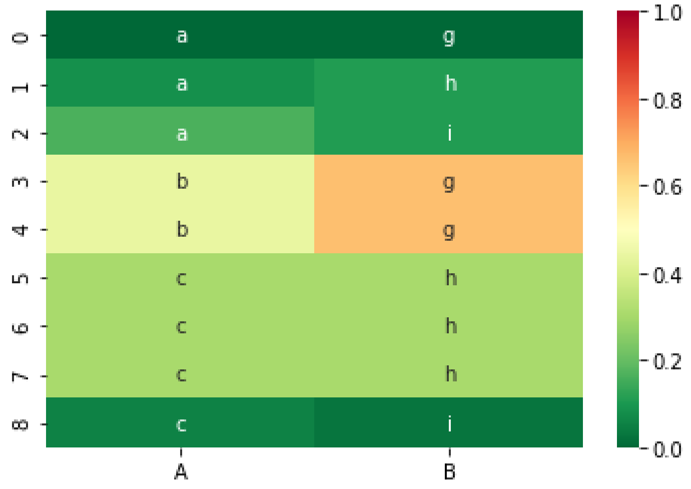
What if instead, we measure the change of probabilities of the cell value between prior and posterior distribution? Compute prior and posterior as before, but now look at how the probability of the cell value changes between those two distributions.

Example

For our dataset from before, the CSF values are as follows:

```
[5]: df_csf = piflib.compute_csfs(df)
     color_map = matplotlib.colors.ListedColormap(
             sns.color_palette("RdYlGn", 256).as_hex()[::-1])
     sns.heatmap(df_csf, vmin=0, vmax=1, cmap=color_map, annot=df, fmt='s')
```

```
[5]: <AxesSubplot:>
```

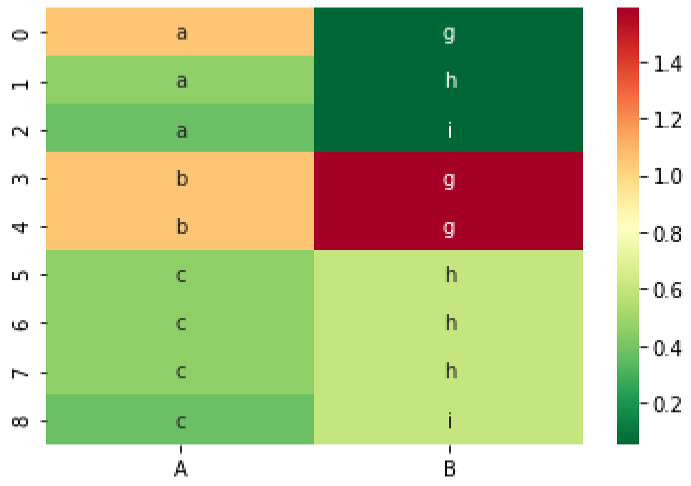


nbsphinx-code-borderwhite

And the corresponding CIG values:

```
[6]: df_cig = piflib.compute_cigs(df)
     sns.heatmap(df_cig, cmap=color_map, annot=df, fmt='s')
```

```
[6]: <AxesSubplot:>
```



nbsphinx-code-borderwhite

Here I want to highlight the difference between CIG and CSF. The probability of the first cell value, the 'a' in row 0 is 1/3 in the distribution of the feature 'A'. In the posterior distribution of 'A' given 'B' = 'g' the probability for 'a' stays the same at 1/3. However, the probability for 'b' changes from 2/9 to 2/3, the one for 'c' from 4/9 to 0.

This in turn means that the CIG value will be > zero, as the posterior is different from the prior distribution, but the CSF value is 0, as the probability of 'a' did not change between the two distributions.

We argue that the CSF is a better measure of surprise for a cell value, as it reflects the change, or surprise, of the cell value alone, without interference of the other elements in the posterior.

In the case above, as the probability for 'A' does not change, there is no element of surprise. However, as the two distributions are different, the KL divergence will be greater than zero, thus falsely indicating that the 'A' value in row 0 is unusual.

Going bigger - hackathon dataset

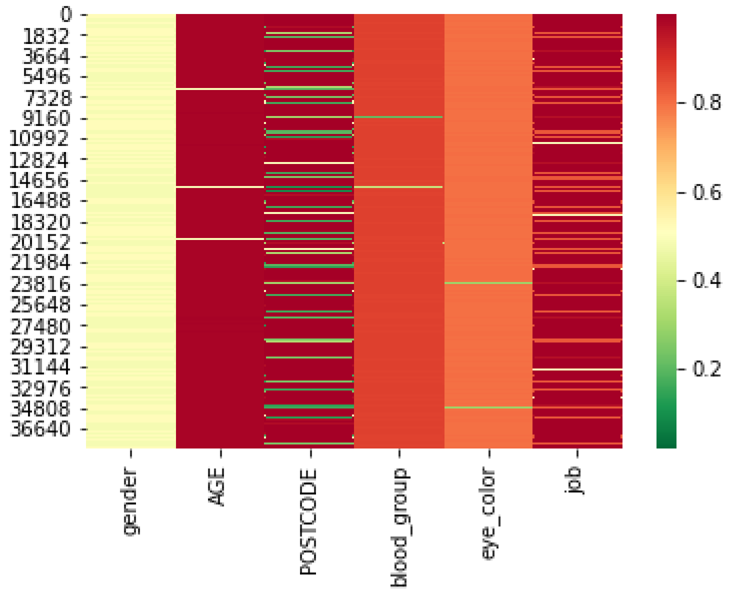
[7]:
```
hack_features = ['gender', 'AGE', 'POSTCODE', 'blood_group', 'eye_color', 'job']
hack_data = pd.read_csv('data/hackathon.csv')[hack_features]
hack_data = hack_data.fillna('Unemployed')
hack_data.head()
```

[7]:
```
  gender  AGE  POSTCODE blood_group eye_color                        job
0      F   99      2649          B-     Brown  Psychologist, counselling
1      M  108      1780          A-     Hazel          Personnel officer
2      M   59      2940          B+     Hazel            Tourism officer
3      M   58      2945          B+      Blue                       Make
4      M   30      2729         AB-     Brown    Forest/woodland manager
```

We now compute the CSF values and display them as a heatmap, with colors ranging from green for 'save' values to red for the most at risk values.

[8]:
```
hack_csf = piflib.compute_csfs(hack_data)
sns.heatmap(hack_csf, cmap=color_map)
```

[8]:
```
<AxesSubplot:>
```



nbsphinx-code-borderwhite

Looking at the distribution of CSF values, we can see that the changes are quite substantial.

```
[9]: hack_csf.describe()
```

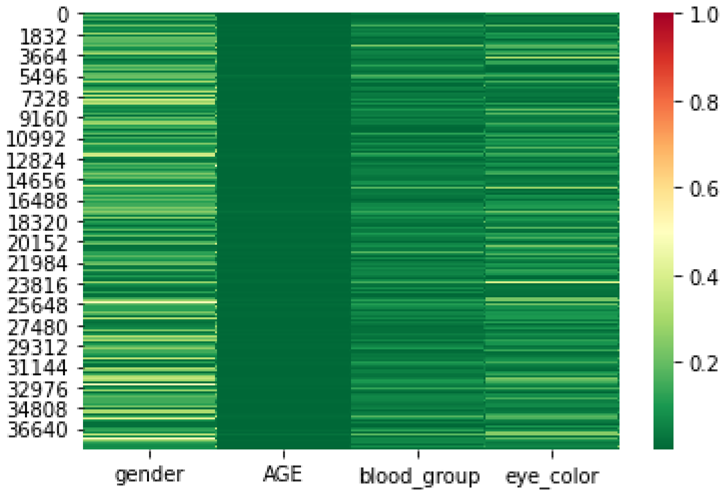
```
[9]:             gender           AGE      POSTCODE   blood_group     eye_color  \
      count  38462.000000  38462.000000  38462.000000  38462.000000  38462.000000
      mean       0.498328      0.977707      0.846887      0.869045      0.795840
      std        0.028408      0.086362      0.307341      0.056100      0.045859
      min        0.017524      0.133263      0.049805      0.073070      0.131810
      25%        0.482476      0.990952      0.996880      0.873252      0.798502
      50%        0.482476      0.991394      0.997660      0.874682      0.800348
      75%        0.517524      0.991654      0.999740      0.876007      0.801232
      max        0.517524      0.999740      0.999974      0.877047      0.801440

                      job
      count  38462.000000
      mean       0.966596
      std        0.074689
      min        0.331747
      25%        0.998414
      50%        0.998648
      75%        0.998804
      max        0.999402
```

CSF values range from 0 to 1, where 0 stands for no surprise and 1 for maximum surprise (or difference from the prior believe). We can see that the mean CSF values are very high. This is due to the fact that almost every row is unique, thus forming mostly cohorts of 1 for the posterior distributions.

Let's try to reduce the numbers by removing the features 'job' and 'POSTCODE'.

```
[10]: cols = ['gender', 'AGE', 'blood_group', 'eye_color']
      sub_hack_csf = piflib.compute_csfs(hack_data[cols])
      sns.heatmap(sub_hack_csf, cmap=color_map, vmax=1)
```

[10]:
```
<AxesSubplot:>
```



nbsphinx-code-borderwhite

[12]:
```
sub_hack_csf.describe()
```

[12]:
```
             gender           AGE   blood_group     eye_color
count  38462.000000  38462.000000  38462.000000  38462.000000
mean       0.136386      0.003600      0.048995      0.072766
std        0.104873      0.003025      0.041834      0.059569
min        0.006286      0.000001      0.000305      0.000348
25%        0.053905      0.001375      0.018391      0.027585
50%        0.117524      0.002807      0.039136      0.058666
75%        0.184190      0.005016      0.069355      0.103322
max        0.517524      0.022063      0.877047      0.801440
```

This already looks a lot better. The mean CSF values are very low now. However, there are some rows in the dataset with high CSF values. These rows still stand out and thus have a higher risk of re-identification.

Conclusions

The CSF, defined as the absolute difference between two probabilities, is always bounded between 0 and 1. Thus, we have fixed bounds and might be able to transfer learnings from one dataset to another more easily.

The CSF measures the element of surprise of a single cell value, whereas the CIG measures the element of surprise of the posterior distribution. We believe that the CSF provides are finer granular, and true-er, picture of the element of surprise of a cell value contained in a dataset.

[ ]:
```

```

CHAPTER

TWO

API DOCUMENTATION

- modindex