Ranking Theory: Theory of Beliefs and Disbeliefs

"Belief is thus the mental state or function of cognising reality" - William James

Uncertainty is not always probabilistic

Theoretical Consideration About Probability:

The difficulties which people have in conceiving about the nature of probability is seen through the gambling and lottery paradox which reveal a common tendency to make incorrect judgments concerning probability.

Nonetheless, studies (most notably the Iowa gambling task) have demonstrated that people possess an intuitive or else subconscious ability to identify what decks are the most rewarding before the judgment reaches their awareness.

Although this study is also related to learning, there is an extension to probability insofar as the study indicates that participants can identify which decks are most probable/likely to provide a positive reward at a subconscious level.

This may suggest that probability (and probabilistic reasoning) is understood ‘procedurally’ or implicitly in the brain as opposed to being primarily explicit/declarative.

This is interesting because, if this were true, it would suggest that probability is something that we ‘do’ as opposed to something that we ‘think’.

An alternative to probability theory is ranking theory, where uncertainty is measured using degrees of surprise or disbelief on the integer scale from 0 to ∞. see (Rienstra, 2019).

These values, called ranks, can be understood as degrees of surprise: 0 for not surprising, 1 for surprising, 2 for even more surprising, and so on, with ∞ for impossible. We can also interpret this as normal means rank 0, while exceptional means rank ≥ 1 (Rienstra, 2019).

> In different terms, a ranking function κ represents a grading of disbelief on a scale from 0 to ∞; where disbelief is maximal for the empty set, representing the contradiction, and minimal for the full set (Raidl, 2019).

Similarities to probability: ranking functions are updated by conditionalization, and can be represented compactly (Rienstra, 2019).

Let $\mathcal{A}$ be an algebra over W. Then κ is a negative *ranking function* for $\mathcal{A}$ iff κ is a function from $\mathcal{A}$ into $\mathbb{R}^*$ = $\mathbb{R}^+$ $\cup\{\infty\}$ i.e., into the set of non-negative reals plus infinity) such that for all $A, B \in A$:

$$\kappa(W) = 0 \text{ and } \kappa(\emptyset) = \infty,$$

$$\kappa(A \cup B) = \min\{\kappa(A), \kappa(B)\} \text{ [law of disjunction (for negative ranl}$$

$\kappa(A)$ is called the *(negative) rank* of A.

It immediately follows for each $A \in \mathcal{A}$

either $\kappa(A)$= 0 or $\kappa(\overline{A})$ = 0 or both [law of negation].

The negative ranking function κ is a grading of disbelief. If κ(A) = 0, A is not disbelieved at all; if κ(A) > 0, A is disbelieved to some positive degree.

Belief in A is the same as disbelief in $\overline{A}$;

> A is believed in κ iff $\kappa(\overline{A})$ > 0 (via the law of negation)
>
> but is not equivalent to κ(A) = 0.
>
> When κ $(\overline{A})$ = 0; κ is neutral or unopinionated concerning A

Positive ranking function (p.r.f): β is defined by β(A) ≔ $\kappa(\overline{A})$

Two-sided ranking function (t.r.f): τ by τ(A) ≔ β(A) − κ(A)

Dynamic part (conditionalisation):

Let κ be a negative ranking function over $\mathcal{A}$ and A $\in$ $\mathcal{A}$ such that κ(A) < ∞

Conditional negative ranking function given A is defined as follows: for all B $\in$ $\mathcal{A}$

$$\kappa(B|A){:} = \kappa(B \cap A) \text{ - } \kappa(A)$$

Three Structural Parallels with probability theory

Static parallel between the definition of a negative ranking function and a probability function:

A probability function over $\mathcal{A}$ is a total function $P : \mathcal{A} \to \mathbb{R}^{+}$ that satisfies

- $P(w)$ = 1 , (normed)

- for all $A, B \in \mathcal{A}$, if $A \cap B = \emptyset$ then $P(A \cup B) = P(A) + P(B)$. (Finite additivity)

We obtain:

- $P(A) + P(A) = 1$, (Probabilistic law of negation)
- if $A \subseteq B$ then $P(A) \leq P(B)$. (Monotonicity)

We can see that: Probability and ranking functions are structurally similar because

- Both functions are non-negative, but the domains are different—it is[0, ∞] for a negative ranking function and [0, 1] for a probability function

> The real difference is encapsulated in the law of how degrees combine when disjoint unions are taken

Conjoining values of disjoint unions, probability theory uses a sum where ranking theory uses a minimum

> The meaning of the order is inverted (w.r.t. ⊆).

Ranking-theoretic independence and probabilistic independence are structurally similar

> Probability theory uses the product formulation $P(A \cap B) = P(A) \times P(B)$ to characterise independence between A and B, ranking theory uses the sum formulation $\kappa(A \cap B) = \kappa(A) + \kappa(B)$
>
> > Or, weaker: For two atomically independent probability functions P, Q, their independent combination is P × Q , whereas for two atomically independent

> > ranking functions κ, γ , their independent combination is κ +γ.

Dynamic parallel:

Let P be a probability function over $\mathcal{A}$ and A $\in$ $\mathcal{A}$ such that P(A) > 0. The conditional probability function given A is defined as follows: for all B $\in$ $\mathcal{A}$,

$$P(B|A) := P(B \cap A)/P(A)$$

Probabilistic conditionalisation and ranking conditionalisation are structurally similar.

Both conditionalisations are undefined iff A takes the value assigned to $\emptyset$

> The crucial difference is: probabilistic conditionalisation divides by P(A), rank conditionalisation subtracts by κ(A) Ranking subtraction as the analogue to probabilistic normalisation by division

Questionnaire

Grade your degrees of disbelief: Language Demographics of Quebec

1. Suppose you meet a person who has only lived in Quebec since bi
Would you be surprised to learn that they only speak _________?

- French
- Spanish
- English
- Greek
- Polish
- Korean

- Urdu
- Japanese
- Spanish or Urdu?

Grade your degrees of disbelief: September Weather in Boston, Massachusetts, United States

2. Suppose you are planning a day trip to Boston in September. Would you be surprised to learn that the daytime high temperatur (°C)?

- -29 to -20
- -19 to -10
- -9 to 0
- 1 to 10
- 11 to 20
- 21 to 30
- 31 to 40
- 41 to 50
- -9 to 0 or 1 to 10

Grade your degrees of disbelief: Music Genre Preference

3. Suppose you are an employee at an automotive service centre.

You have been invited to a work karaoke party, and you know you

Tom is a Caucasian male in his 20s who grew up in a rural area and

Would it surprise you to learn that Tom enjoys singing along to __

- Pop
- Rock
- Hip Hop
- Heavy Metal
- Country
- Jazz

- Opera
- Folk
- Jazz or Opera

Equation translating from probability to rankings

$$P(w) \longmapsto \kappa_P^a(w) := log_a(\frac{P(w)}{p})$$

> we can also rearrange

$$b = \log_a(x) \longleftrightarrow a^b = X$$

$$a^{\kappa_P^a(w)} = \frac{P(w)}{p}$$

>finally we have

$$a = \left(\frac{P(w)}{p}\right)^{1/\kappa_P^a(w)}$$

Equation translating from rankings to probability

$$\kappa(w) \longmapsto P_\kappa^a(w) := \frac{a^{\kappa(w)}}{Z(a,\kappa)}$$

In [25]:
```
# Libraries
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
```

In [26]:
```
# define ranking functions


# return maximum element of a set
def maxSet(list):
    return(np.max(list))


# probability to ranking translation (normalized)
```

```
def prob_to_rank(probSet,base,max_prob):
    return((np.log(probSet)/np.log(base)) - np.log(max_prob)/
np.log(base))

# ranking to probability translation (normalized)
def rank_to_prob(rankingSet,base):
    return(([base**r for r in rankingSet])/np.sum([base**r for r in
rankingSet]))

# convert float to the nearest integer
def integers(list):
     return([int(round(x)) for x in list])

# normalize
def normalize(raw):
    norm = [float(i)/sum(raw) for i in raw]
    return(norm)

# compute base
def computeBase(probSet,rankSet):
    return((probSet/maxSet(probSet)) ** ([1/r for r in rankSet]))
```

Sophia's Probability and Rank Estimates for each Questionnaire:

In [16]:
```
Sophia_Q1_R =   {'French':           0,
                 'Spanish':         75,
                 'English':          5,
                 'Greek':          300,
                 'Polish':          90,
                 'Korean':         250,
                 'Urdu':            80,
                 'Japanese':       200,
                 'Spanish or Urdu': 50}


Sophia_Q1_P =   {'French':           0.97,
                 'Spanish':         0.003,
                 'English':          0.01,
                 'Greek':           0.002,
                 'Polish':          0.003,
                 'Korean':          0.001,
                 'Urdu':            0.004,
                 'Japanese':        0.003,
                 'Spanish or Urdu': 0.004}

Sophia_Q2_R =   {'-29 to -20':        50,
                 '-19 to -10':        20,
                 '-9 to 0':           15,
                 '1 to 10':           10,
                 '11 to 20':           0,
                 '21 to 30':          10,
                 '31 to 40':          40,
                 '41 to 50':          50,
                 '-9 to 0 or 1 to 10': 5}

Sophia_Q2_P =   {'-29 to -20':        0.005,
                 '-19 to -10':        0.020,
                 '-9 to 0':           0.040,
                 '1 to 10':           0.080,
                 '11 to 20':          0.700,
                 '21 to 30':          0.040,
                 '31 to 40':          0.020,
                 '41 to 50':          0.005,
                 '-9 to 0 or 1 to 10':0.090}

Sophia_Q3_R =   {'Pop':               20,
                 'Rock':               5,
                 'Hip Hop':           25,
                 'Heavy Metal':       10,
                 'Country':            0,
                 'Jazz':              20,
                 'Opera':             50,
                 'Folk':               5,
                 'Jazz or Opera':     15}

Sophia_Q3_P =   {'Pop':             0.015,
                 'Rock':            0.020,
                 'Hip Hop':          0.005,
                 'Heavy Metal':      0.010,
                 'Country':          0.900,
                 'Jazz':             0.010,
                 'Opera':            0.005,
                 'Folk':             0.020,
                 'Jazz or Opera':    0.010}
```

Hanbin's Probability and Rank Estimates for each Questionnaire:

In [17]:
```
Hanbin_Q1_R =    {'French':            0,
                 'Spanish':            3,
                 'English':            0,
                 'Greek':             12,
                 'Polish':            12,
                 'Korean':            12,
                 'Urdu':              10,
                 'Japanese':          12,
                 'Spanish or Urdu':    3}

Hanbin_Q1_P =    {'French':            0.88,
                 'Spanish':            0.03,
                 'English':            0.10,
                 'Greek':             0.002,
                 'Polish':            0.002,
                 'Korean':            0.002,
                 'Urdu':              0.003,
                 'Japanese':          0.002,
                 'Spanish or Urdu':    0.03}

Hanbin_Q2_R =    {'-29 to -20':         40,
                 '-19 to -10':         20,
                 '-9 to 0':             5,
                 '1 to 10':             1,
                 '11 to 20':            0,
                 '21 to 30':            0,
                 '31 to 40':            4,
                 '41 to 50':          10,
                 '-9 to 0 or 1 to 10': 5}

Hanbin_Q2_P =   {'-29 to -20':       0.0001,
                 '-19 to -10':        0.002,
                 '-9 to 0':           0.003,
                 '1 to 10':            0.05,
                 '11 to 20':           0.40,
                 '21 to 30':           0.50,
                 '31 to 40':           0.02,
                 '41 to 50':          0.007,
                 '-9 to 0 or 1 to 10':  0.05}

Hanbin_Q3_R =   {'Pop':                 2,
                 'Rock':                1,
                 'Hip Hop':             7,
                 'Heavy Metal':         5,
                 'Country':             0,
                 'Jazz':               10,
                 'Opera':              50,
                 'Folk':                1,
                 'Jazz or Opera':      10}

Hanbin_Q3_P =   {'Pop':               0.05,
                 'Rock':              0.20,
                 'Hip Hop':           0.01,
                 'Heavy Metal':       0.02,
                 'Country':           0.50,
                 'Jazz':              0.01,
                 'Opera':            0.005,
                 'Folk':              0.20,
                 'Jazz or Opera':     0.01}
```

In [18]:

```
import pandas as pd
df1 = pd.DataFrame(list(Sophia_Q1_R.items()), columns = ['elements',
'Sophia K(w)'])
df1['Sophia P(w)'] = df1['elements'].map(Sophia_Q1_P)
df1['Hanbin K(w)'] = df1['elements'].map(Hanbin_Q1_R)
```

```
df1['Hanbin P(w)'] = df1['elements'].map(Hanbin_Q1_P)
df1.style.set_caption("Q1: Rank and Probability Sets")
```

Out[18]:

Q1: Rank and Probability Sets

	elements	Sophia K(w)	Sophia P(w)	Hanbin K(w)	Hanbin P(w)
0	French	0	0.970000	0	0.880000
1	Spanish	75	0.003000	3	0.030000
2	English	5	0.010000	0	0.100000
3	Greek	300	0.002000	12	0.002000
4	Polish	90	0.003000	12	0.002000
5	Korean	250	0.001000	12	0.002000
6	Urdu	80	0.004000	10	0.003000
7	Japanese	200	0.003000	12	0.002000
8	Spanish or Urdu	50	0.004000	3	0.030000

In [19]:

```
df2 = pd.DataFrame(list(Sophia_Q2_R.items()), columns = ['elements',
'Sophia K(w)'])
df2['Sophia P(w)'] = df2['elements'].map(Sophia_Q2_P)
df2['Hanbin K(w)'] = df2['elements'].map(Hanbin_Q2_R)
df2['Hanbin P(w)'] = df2['elements'].map(Hanbin_Q2_P)
df2.style.set_caption("Q2: Rank and Probability Sets")
```

Out[19]:

Q2: Rank and Probability Sets

	elements	Sophia K(w)	Sophia P(w)	Hanbin K(w)	Hanbin P(w)
0	-29 to -20	50	0.005000	40	0.000100
1	-19 to -10	20	0.020000	20	0.002000
2	-9 to 0	15	0.040000	5	0.003000
3	1 to 10	10	0.080000	1	0.050000
4	11 to 20	0	0.700000	0	0.400000
5	21 to 30	10	0.040000	0	0.500000
6	31 to 40	40	0.020000	4	0.020000
7	41 to 50	50	0.005000	10	0.007000
8	-9 to 0 or 1 to 10	5	0.090000	5	0.050000

In [20]:

```
df3 = pd.DataFrame(list(Sophia_Q3_R.items()), columns = ['elements',
'Sophia K(w)'])
df3['Sophia P(w)'] = df3['elements'].map(Sophia_Q3_P)
df3['Hanbin K(w)'] = df3['elements'].map(Hanbin_Q3_R)
```

```
df3['Hanbin P(w)'] = df3['elements'].map(Hanbin_Q3_P)
df3.style.set_caption("Q3: Rank and Probability Sets")
```

Out[20]:

Q3: Rank and Probability Sets

	elements	Sophia K(w)	Sophia P(w)	Hanbin K(w)	Hanbin P(w)
0	Pop	20	0.015000	2	0.050000
1	Rock	5	0.020000	1	0.200000
2	Hip Hop	25	0.005000	7	0.010000
3	Heavy Metal	10	0.010000	5	0.020000
4	Country	0	0.900000	0	0.500000
5	Jazz	20	0.010000	10	0.010000
6	Opera	50	0.005000	50	0.005000
7	Folk	5	0.020000	1	0.200000
8	Jazz or Opera	15	0.010000	10	0.010000

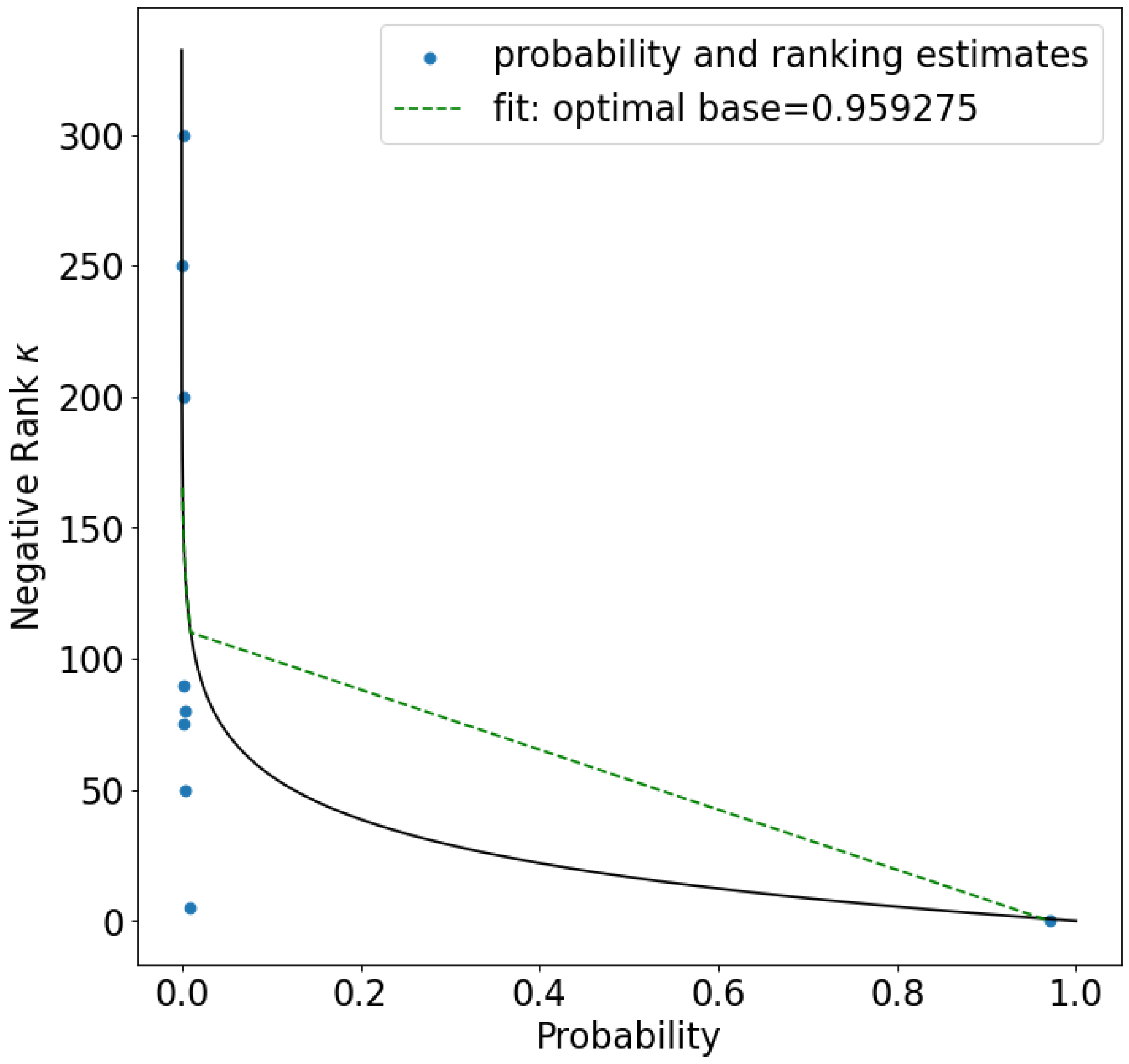
Compute optimal base values, estimate covariance of each optimal values.

In [21]:

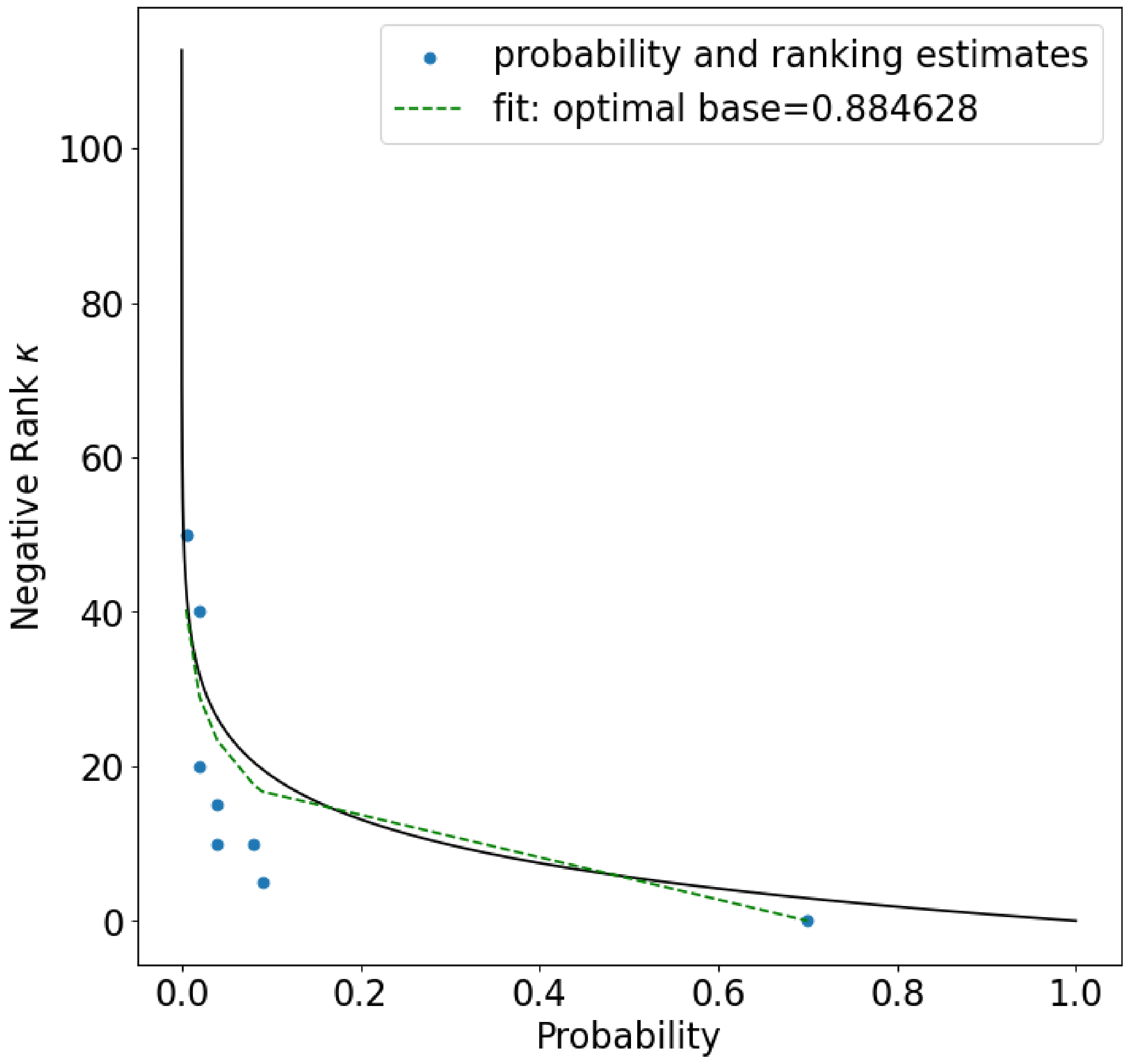
```
#sophia's ranks and probilities:

#ranks
Sophia_ranks =
[list(Sophia_Q1_R.values()),list(Sophia_Q2_R.values()),list(Sophia_Q3_R.

#normalized prob
Sophia_probs =
[normalize(list(Sophia_Q1_P.values())),normalize(list(Sophia_Q2_P.values



#Use non-linear least squares to fit a function, f, to data.

# this function is to estimate the base given x, being our probability
set.
def func(x,a):
    return ((np.log(x)/np.log(a))- (np.log(maxSet(x))/ np.log(a)))

#lower and upper limit
lowerB, upperB = (1.0e-30,0.9999999999999999999999999999)
```

```
for rankSet, probSet in zip(Sophia_ranks,Sophia_probs):
    # Optimal values ,  estimated covariance of popt using our
probability set and ranking set.
    popt, pcov = curve_fit(func, probSet, rankSet, bounds=
(lowerB,upperB))
    print("Optimal base: ", popt, "estimated covariance", pcov)

    #debug
    #popt, pcov = curve_fit(computeBase, S_Q1_prob, S_Q1_rank, bounds=
(lowerB,upperB))

    x = np.arange(0,1, step= 0.000001)[1:-1]
    y = prob_to_rank(x, popt,x[-1])
    plt.figure(figsize=(10,10), dpi=80)
    plt.plot(x,y, 'k')
    plt.scatter(probSet, rankSet,label = "probability and ranking
estimates")
    plt.plot(sorted(probSet), func(sorted(probSet), *popt), 'g--',
             label='fit: optimal base=%f' % tuple(popt))
    plt.legend(fontsize = 20)
    plt.xlabel("Probability")
    plt.ylabel("Negative Rank $\kappa$ ")
    plt.rcParams.update({'font.size': 20})
    plt.show()
```

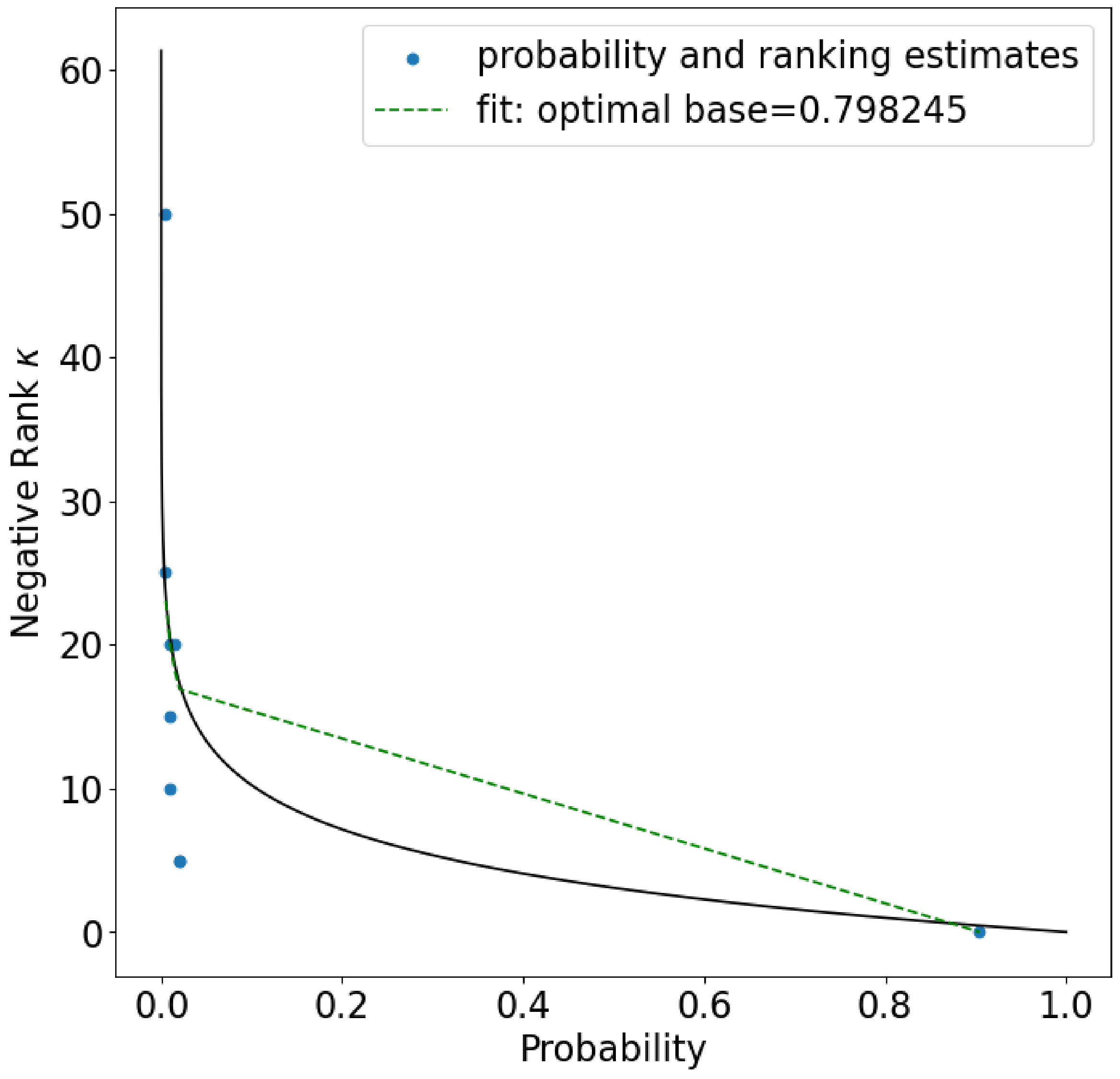
```
Optimal base:  [0.9592746] estimated covariance [[7.82011479e-05]]
```



```
Optimal base:  [0.88462776] estimated covariance [[0.00018543]]
```



```
Optimal base:  [0.79824458] estimated covariance [[0.0014603]]
```

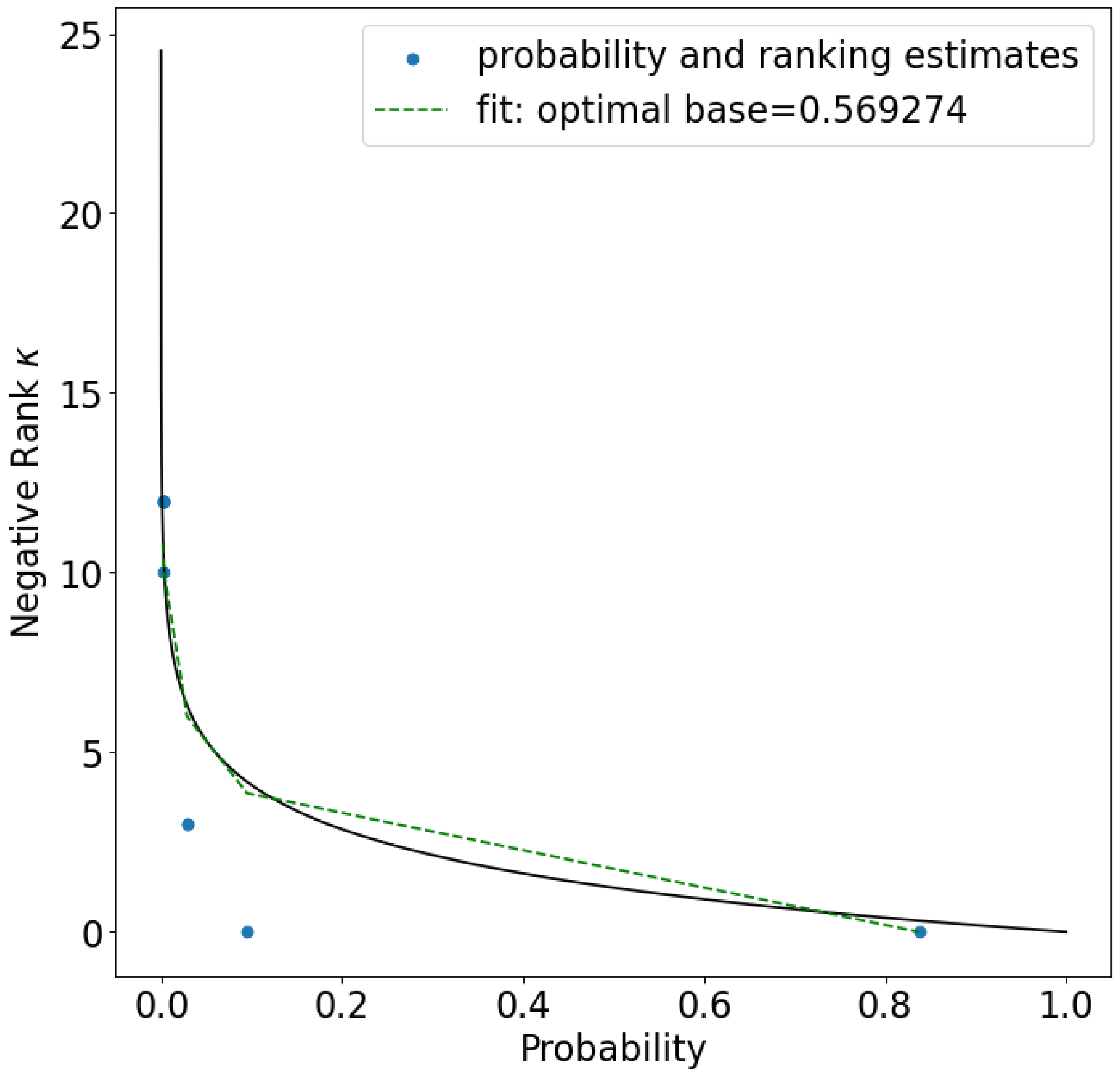


In [22]:

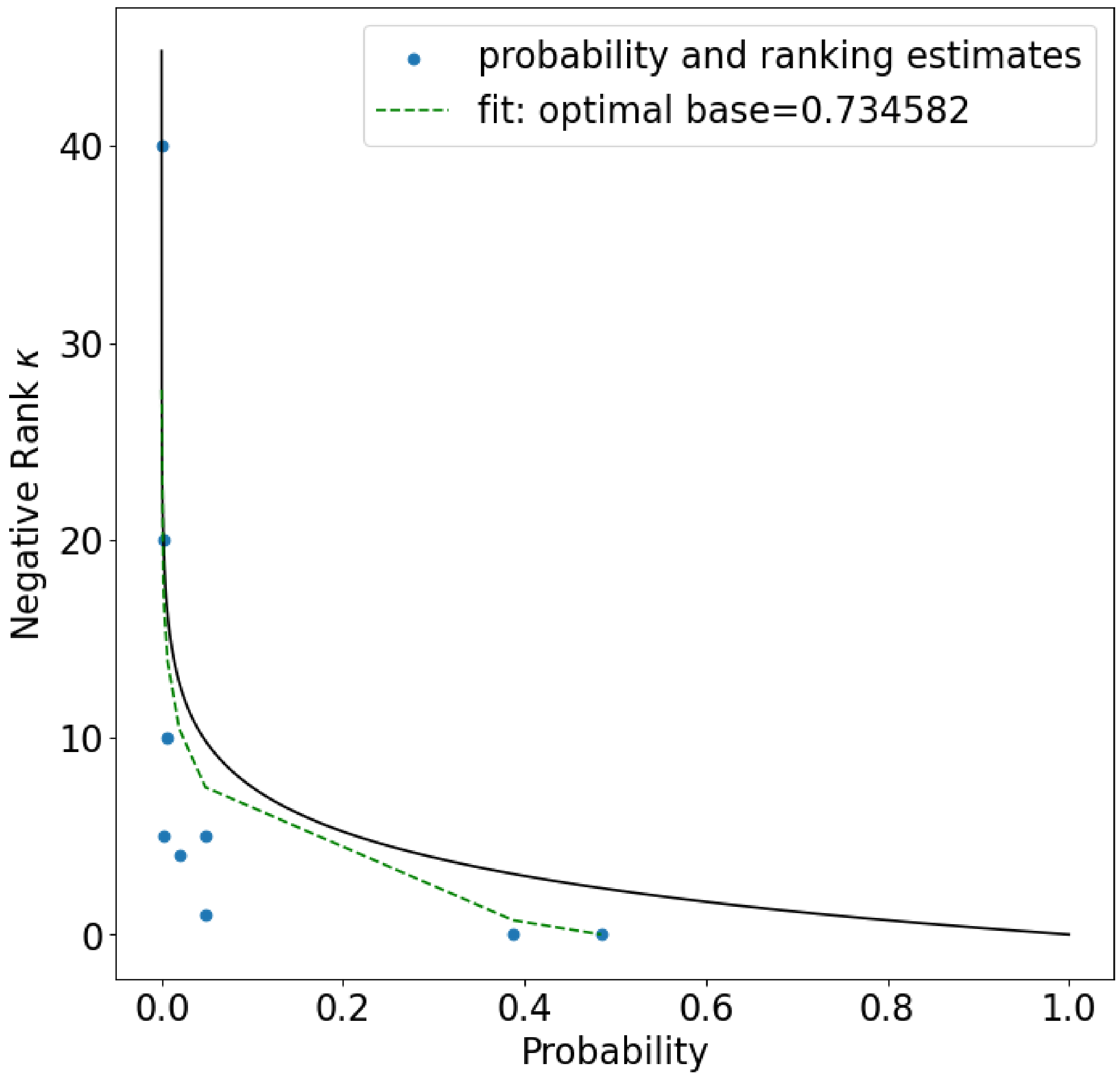
```
#Hanbin's ranks and probilities:

Hanbin_ranks =
[list(Hanbin_Q1_R.values()),list(Hanbin_Q2_R.values()),list(Hanbin_Q3_R.

Hanbin_probs =
[normalize(list(Hanbin_Q1_P.values())),normalize(list(Hanbin_Q2_P.values



#Use non-linear least squares to fit a function, f, to data.
def func(x,a):
    return ((np.log(x)/np.log(a))- (np.log(maxSet(x))/ np.log(a)))

#lower and upper limit
lowerB, upperB = (1.0e-30,0.9999999999999999999999999)
```

```
for rankSet, probSet in zip(Hanbin_ranks,Hanbin_probs):
    # Optimal values ,  estimated covariance of popt
    popt, pcov = curve_fit(func, probSet, rankSet, bounds=
(lowerB,upperB))
    print("Optimal base: ", popt, "estimated covariance", pcov)

    #debug
    #popt, pcov = curve_fit(computeBase, S_Q1_prob, S_Q1_rank, bounds=
(lowerB,upperB))

    x = np.arange(0,1, step= 0.000001)[1:-1]
    y = prob_to_rank(x, popt,x[-1])
    plt.figure(figsize=(10,10), dpi=80)
    plt.plot(x,y, 'k')
    plt.scatter(probSet, rankSet,label = "probability and ranking
estimates")
    plt.plot(sorted(probSet), func(sorted(probSet), *popt), 'g--',
             label='fit: optimal base=%f' % tuple(popt))
    plt.legend(fontsize = 20)
    plt.xlabel("Probability")
    plt.ylabel("Negative Rank $\kappa$ ")
    plt.rcParams.update({'font.size': 20})
    plt.show()
```

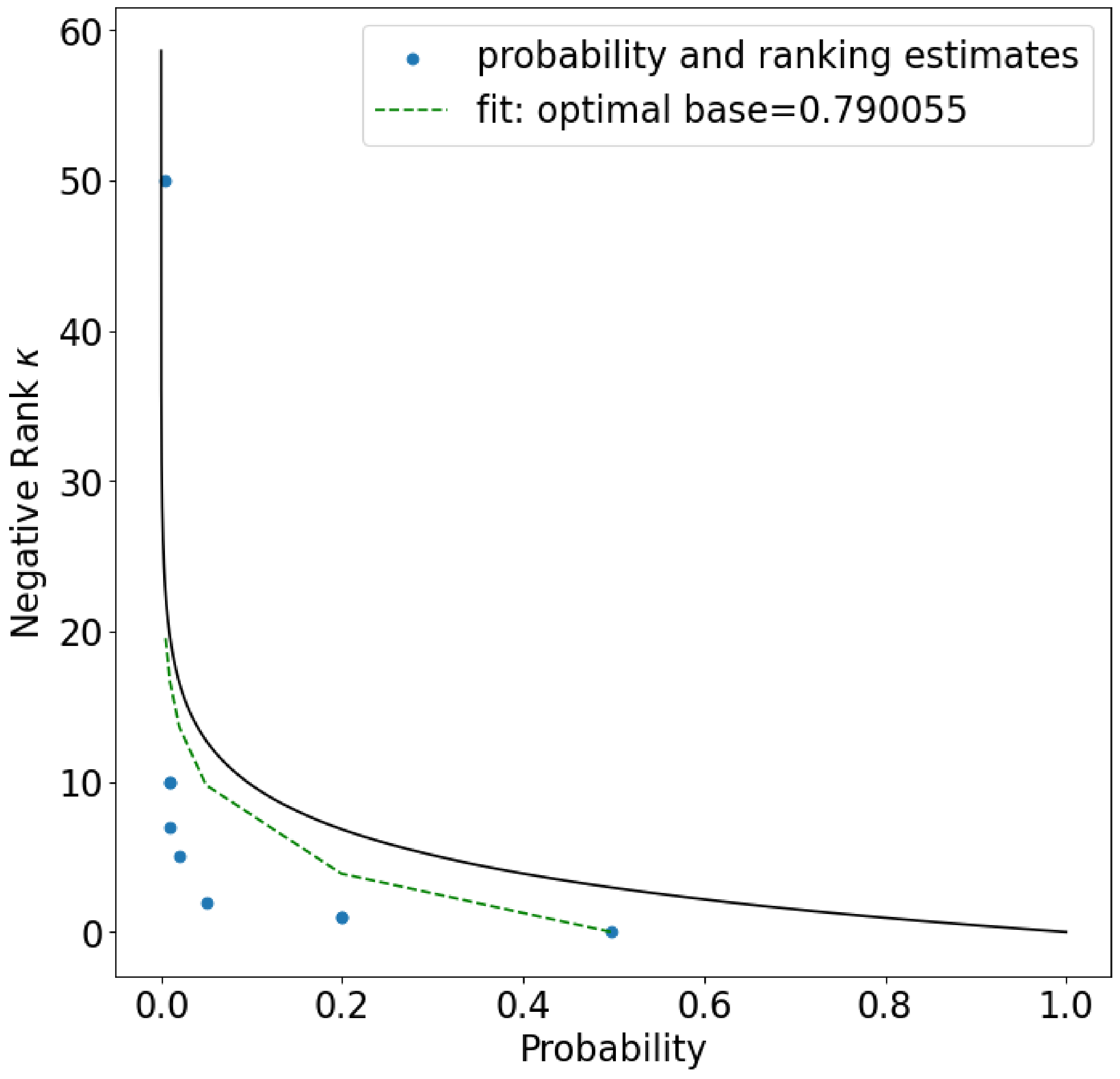
```
Optimal base:  [0.56927351] estimated covariance [[0.00075721]]
```



```
Optimal base:  [0.73458166] estimated covariance [[0.00143782]]
```



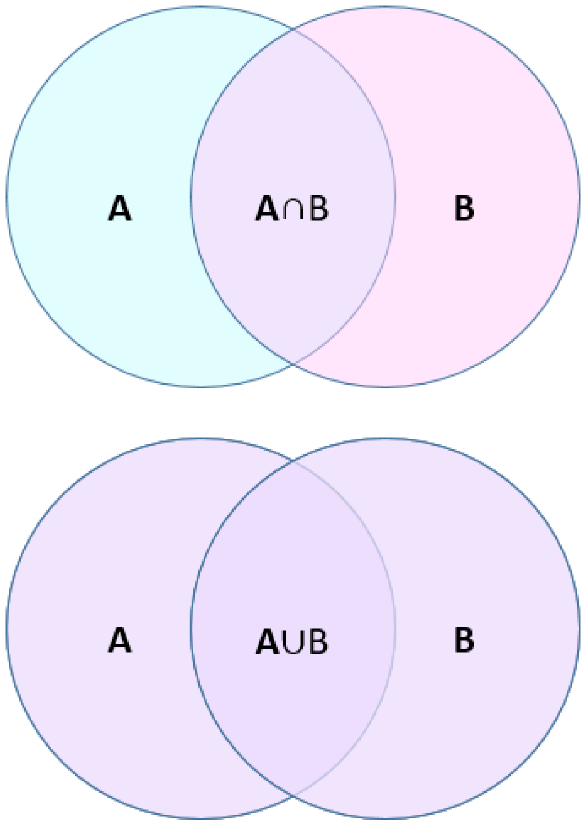
```
Optimal base:  [0.79005484] estimated covariance [[0.00358675]]
```



It appears that within subjects, optimal base values vary across questionnaires, this is because the ranks and probability estimates weren't consistent.

The Jaccard index. is used to gauge the similarity and diversity of sample sets. Given two objects, A and B, each with n binary attributes, the Jaccard coefficient is a useful measure of the overlap that A and B share with their attributes. Each attribute of A and B can either be 0 or 1:

$$J(A,B) = \frac{|A \cap B|}{|A \uplus B|} = \frac{|A \cap B|}{|A| + |B|}$$



Intersection and union of two sets A and B

Using the Jaccard Index, we wanted to see how similar the elements of the two sets from the same questionnaires were between subjects.

In [23]:
```
# jaccard similarity index
def jaccard_similarity(A, B):
    #Find intersection of two sets
    nominator = A.intersection(B)

    #Find union of two sets
    denominator = A.union(B)

    #Take the ratio of sizes
    similarity = len(nominator)/len(denominator)

    return similarity
```

```
#Convert our ranking and probability list to a set(e.g., curly braces
in python using the set function)
Sophia_rankSets =
[set(list(Sophia_Q1_R.values())),set(list(Sophia_Q2_R.values())),set(li

Sophia_probSets =
[set(normalize(list(Sophia_Q1_P.values()))),set(normalize(list(Sophia_Q


Hanbin_rankSets =
[set(list(Hanbin_Q1_R.values())),set(list(Hanbin_Q2_R.values())),set(li

Hanbin_probSets =
[set(normalize(list(Hanbin_Q1_P.values()))),set(normalize(list(Hanbin_Q




sim_index_ranks = {'Language Demographics of
Quebec':jaccard_similarity(Sophia_rankSets[0], Hanbin_rankSets[0]),
                   'September Weather in
Boston':jaccard_similarity(Sophia_rankSets[1], Hanbin_rankSets[1]),
                   'Music Genre
Preference':jaccard_similarity(Sophia_rankSets[2],
Hanbin_rankSets[2])}

print("ranks", sim_index_ranks)

sim_index_probs = {'Language Demographics of
Quebec':jaccard_similarity(Sophia_probSets[0], Hanbin_probSets[0]),
                   'September Weather in
Boston':jaccard_similarity(Sophia_probSets[1], Hanbin_probSets[1]),
                   'Music Genre
Preference':jaccard_similarity(Sophia_probSets[2],
Hanbin_probSets[2])}

print("probs", sim_index_probs)
```

```
ranks {'Language Demographics of Quebec': 0.08333333333333333, 'September Weathe
r in Boston': 0.5555555555555556, 'Music Genre Preference': 0.4}
probs {'Language Demographics of Quebec': 0.0, 'September Weather in Boston': 0.
0, 'Music Genre Preference': 0.0}
```

The Jaccard Index would predictably return a similarity value of 0 for the probability sets, as any given member in the two probability sets is unlikely to be the same. The Jaccard Index is great for comparing rankings, as it compares members for two sets to see which members are shared and which are distinct. However, it is unable to compare the magnitude of the differences between the two members.

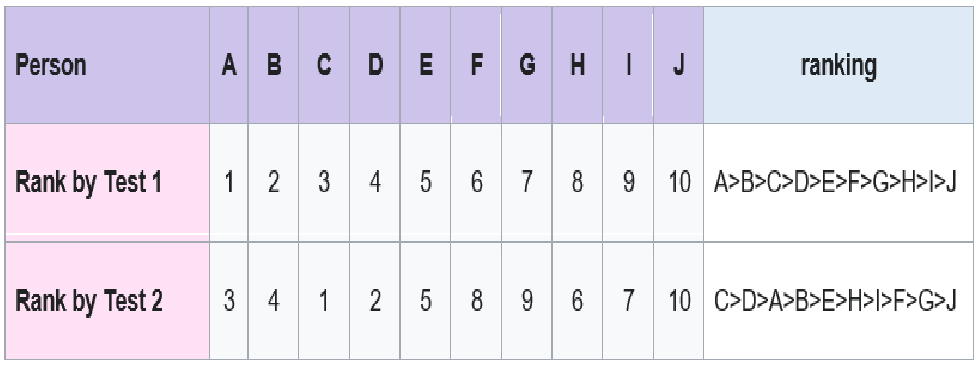
The Kendall tau rank distance is a metric (distance function) that counts the number of pairwise disagreements between two ranking lists. The greater the distance between the two lists, the more dissimilar they are.

The Kendall tau ranking distance between two lists τ_1 and τ_2:

$$K_d\tau_1, \tau_2) = |\{(i,j) : i < j, [\tau_1(i) < \tau_1(j) \wedge \tau_2(i) > \tau_2(j)] \vee [\tau_1(i) > \tau_1($$

where

$\tau_1(i)$ and $\tau_2(i)$ are the rankings of the element i in τ_1 and τ_2, respectively.



Person	A	B	C	D	E	F	G	H	I	J	ranking
Rank by Test 1	1	2	3	4	5	6	7	8	9	10	A>B>C>D>E>F>G>H>I>J
Rank by Test 2	3	4	1	2	5	8	9	6	7	10	C>D>A>B>E>H>I>F>G>J

Pair	Test1	Test 2	Count	Pair	Test1	Test 2	Count	Pair	Test1	Test 2	Count
(A,B)	1 < 2	3 < 4		(A,C)	1 < 3	3 > 1	x	(A,D)	1 < 4	3 > 2	x
(A,E)	1 < 5	3 < 5		(A,F)	1 < 6	1 < 8		(A,G)	1 < 7	1 < 9	
(A,H)	1 < 8	1 < 6		(A,I)	1 < 9	1 < 7		(A,J)	1 < 10	1 < 10	
(B,C)	2 < 3	4 > 1	x	(B,D)	2 < 4	4 > 2	x	(B,E)	2 < 5	4 < 5	
(B,F)	2 < 6	4 < 8		(B,G)	2 < 7	4 < 9		(B,H)	2 < 8	4 < 6	
(B,I)	2 < 9	4 < 7		(B,J)	2 < 10	4 < 10					
(C,D)	3 < 4	1 < 2		(C,E)	3 < 5	1 < 5		(C,F)	3 < 6	1 < 8	
(C,G)	3 < 7	1 < 9		(C,H)	3 < 8	1 < 6		(C,I)	3 < 9	1 < 7	
(C,J)	3 < 10	1 < 10									
(D,E)	4 < 5	2 < 5		(D,F)	4 < 6	2 < 8		(D,G)	4 < 7	2 < 9	
(D,H)	4 < 8	2 < 6		(D,I)	4 < 9	2 < 7		(D,J)	4 < 10	2 < 10	
(E,F)	5 < 6	5 < 8		(E,G)	5 < 7	5 < 9		(E,H)	5 < 8	5 < 6	
(E,I)	5 < 9	5 < 7		(E,J)	5 < 10	5 < 10					
(F,G)	6 < 7	8 < 9		(F,H)	6 < 8	8 > 6	X	(F,I)	6 < 9	8 > 7	X
(F,J)	6 < 10	8 < 10									
(G,H)	7 < 8	9 > 6	X	(G,I)	7 < 9	9 > 7	X	(G,J)	7 < 10	9 < 10	
(H,I)	8 < 9	6 < 7		(H,J)	8 < 10	6 < 10					
(I,J)	9 < 10	7 < 10									

Since there are eight pairs whose values are in opposite order, the Kendall tau distance is eight.

In [24]:
```
# kendall tau distance function
def normalized_kendall_tau_distance(values1, values2):
    """Compute the Kendall tau distance."""
    n = len(values1)
    assert len(values2) == n, "Both lists have to be of equal
length"
    i, j = np.meshgrid(np.arange(n), np.arange(n)) #create a
```

```
rectangular grid out of two given one-dimensional arrays
    a = np.argsort(values1)
    b = np.argsort(values2)

    # logical_or, if one of the and statements are true, return
true. we take the sum() so a true would be a value of 1, and false
would be a value of 0.
    ndisordered = np.logical_or(np.logical_and(a[i] < a[j], b[i] >
b[j]), np.logical_and(a[i] > a[j], b[i] < b[j])).sum()
    return ndisordered / (n * (n - 1))


tau_distance_ranks = {'Language Demographics of
Quebec':normalized_kendall_tau_distance(Sophia_ranks[0],
Hanbin_ranks[0]),
                      'September Weather in
Boston':normalized_kendall_tau_distance(Sophia_ranks[1],
Hanbin_ranks[1]),
                      'Music Genre
Preference':normalized_kendall_tau_distance(Sophia_ranks[2],
Hanbin_ranks[2])}

print("ranks", tau_distance_ranks)

tau_distance_probs = {'Language Demographics of
Quebec':normalized_kendall_tau_distance(Sophia_probs[0],
Hanbin_probs[0]),
                      'September Weather in
Boston':normalized_kendall_tau_distance(Sophia_probs[1],
Hanbin_probs[1]),
                      'Music Genre
Preference':normalized_kendall_tau_distance(Sophia_probs[2],
Hanbin_probs[2])}

print("probs", tau_distance_probs)
```

```
ranks {'Language Demographics of Quebec': 0.5277777777777778, 'September Weath
er in Boston': 0.4166666666666667, 'Music Genre Preference': 0.361111111111111
1}
probs {'Language Demographics of Quebec': 0.3055555555555556, 'September Weath
er in Boston': 0.4444444444444444, 'Music Genre Preference': 0.361111111111111
1}
```

In contrast to the Jaccard index, the Kendall tau distance function allowed us to compute the dissimilarity for our probability sets. As can be seen, the greatest disparity was discovered in our two ranking sets of the Language Demographics of Quebec questionnaire. The disadvantage of this calculation is that it does not take into account the actual values of the ranks.