PyPair

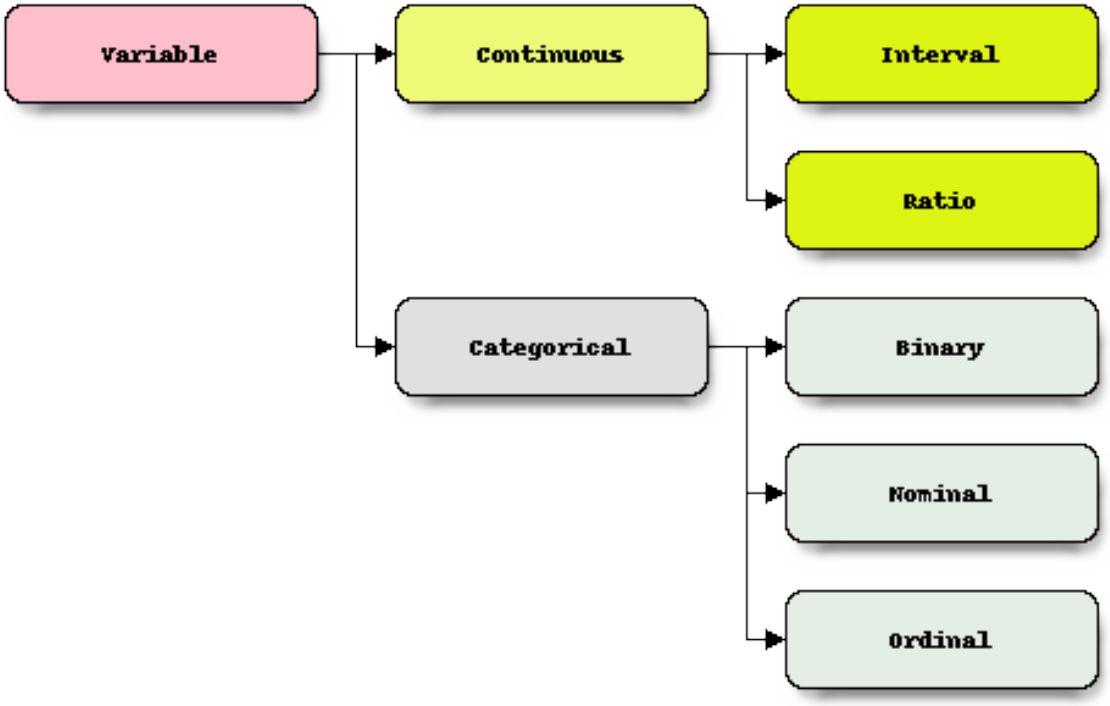
Release 3.0.8

Jee Vang

Dec 08, 2020

CONTENTS

PyPair is a statistical library to compute pairwise association between any two types of variables. You can use the library locally on your laptop or desktop, or, you may use it on a Spark cluster.



CHAPTER

ONE

INTRODUCTION

PyPair is a statistical library to compute pairwise association between any two variables. A reasonable taxonomy of variable types in statistics is as follows [oM][fDRE][Sta][Gra][Min].

- **`Categorical`: A variable whose values have no intrinsic ordering. An example is a variable indicating the continents: N**
  - `Binary`: A categorical variable that has only 2 values. An example is a variable indicating whether or not someone likes to eat pizza; the values could be `yes` or `no`. It is common to encode the binary values to `0` and `1` for storage and numerical convenience, but do not be fooled, there is still no numerical ordering. These variables are also referred to in the wild as *dichotomous* variables.
  - `Nominal`: A categorical variable that has 3 or more values. When most people think of categorical variables, they think of nominal variables.
  - `Ordinal`: A categorical variable whose values have a logical order but the difference between any two values do not give meaningful numerical magnitude. An example of an ordinal variable is one that indicates the performance on a test: good, better, best. We know that good is the base, better is the comparative and best is the superlative, however, we cannot say that the difference between best and good is two numbers up. For all we know, best can be orders of magnitude away from good.
- **`Continuous`: A variable whose values are (basically) numbers, and thus, have meaningful ordering. A continuous varial**
  - `Interval`: A continuous variable that is one whose values exists on a continuum of numerical values. Temperature measured in Celcius or Fahrenheit is an example of an interval variable.
  - `Ratio`: An interval variable with a true zero. Temperature measured in Kelvin is an example of a ratio variable.

Note: If we have a variable capturing eye colors, the possible values may be blue, green or brown. On first sight, this variable may be considered a nominal variable. Instead of capturing the eye color categorically, what if we measure the wavelengths of eye colors? Below are estimations of each of the wavelengths (nanometers) corresponding to these colors.

- blue: 450
- green: 550
- brown: 600

Which variable type does the eye color variable become?

Note: There is also much use of the term `discrete variable`, and sometimes it refers to categorical or continuous variables. In general, a discrete variable has a finite set of values, and in this sense, a discrete variable could be

a categorical variable. We have seen many cases of a continuous variable (infinite values) undergoing *discretization* (finite values). The resulting variable from discretization is often treated as a categorical variable by applying statistical operations appropriate for that type of variable. Yet, in some cases, a continuous variable can also be a discrete variable. If we have a variable to capture age (whole numbers only), we might observe a range $[0, 120]$. There are 121 values (zero is included), but still, we can treat this age variable like a ratio variable.

Assuming we have data and we know the variable types in this data using the taxonomy above, we might want to make a progression of analyses from univariate, bivariate and to multivariate analyses. Along the way, for bivariate analysis, we are often curious about the association between any two pairs of variables. We want to know both the magnitude (the strength, is it small or big?) and direction (the sign, is it positive or negative?) of the association. When the variables are all of the same type, association measures may be abound to conduct pairwise association; if all the variables are continuous, we might just want to apply canonical Pearson correlation.

The tough situation is when we have a mixed variable type of dataset; and this tough situation is quite often the normal situation. How do we find the association between a continuous and categorical variable? We can create a table as below to map the available association measure approaches for any two types of variables [Cal][Unib]. (In the table below, we collapse all categorical and continuous variable types).

The ultimate goal of this project is to identify as many measures of associations for these unique pairs of variable types and to implement these association measures in a unified application programming interface (API).

Note: We use the term *association* over *correlation* since the latter typically connotes canonical Pearson correlation or association between two continuous variables. The term *association* is more general and can cover specific types of association, such as *agreement* measures, along side with those dealing with continuous variables [Lie83].

CHAPTER

TWO

QUICK LIST

Below are just some quick listing of association measures without any description. These association measures are grouped by variable pair types and/or approach.

2.1 Binary-Binary (88)

- adjusted_rand_index
- ample
- anderberg
- baroni_urbani_buser_i
- baroni_urbani_buser_ii
- braun_banquet
- chisq
- chisq
- chisq_dof
- chord
- cole_i
- cole_ii
- contingency_coefficient
- cosine
- cramer_v
- dennis
- dice
- disperson
- driver_kroeber
- euclid
- eyraud
- fager_mcgowan
- faith

- forbes_ii
- forbesi
- fossum
- gilbert_wells
- gk_lambda
- gk_lambda_reversed
- goodman_kruskal
- gower
- gower_legendre
- hamann
- hamming
- hellinger
- inner_product
- intersection
- jaccard
- jaccard_3w
- jaccard_distance
- johnson
- kulcyznski_ii
- kulczynski_i
- lance_williams
- mcconnaughey
- mcnemar_test
- mean_manhattan
- michael
- mountford
- mutual_information
- ochia_i
- ochia_ii
- odds_ratio
- pattern_difference
- pearson_heron_i
- pearson_heron_ii
- pearson_i
- peirce
- person_ii

- phi
- roger_tanimoto
- russel_rao
- shape_difference
- simpson
- size_difference
- sokal_michener
- sokal_sneath_i
- sokal_sneath_ii
- sokal_sneath_iii
- sokal_sneath_iv
- sokal_sneath_v
- sorensen_dice
- sorgenfrei
- stiles
- tanimoto_distance
- tanimoto_i
- tanimoto_ii
- tarantula
- tarwid
- tetrachoric
- tschuprow_t
- uncertainty_coefficient
- uncertainty_coefficient_reversed
- vari
- yule_q
- yule_q_difference
- yule_w
- yule_y

2.2 Confusion Matrix, Binary-Binary (29)

- acc
- ba
- bm
- dor
- f1
- fdr
- fn
- fnr
- fomr
- fp
- fpr
- mcc
- mk
- n
- nlr
- npv
- plr
- ppv
- precision
- prevalence
- pt
- recall
- sensitivity
- specificity
- tn
- tnr
- tp
- tpr
- ts

2.3 Categorical-Categorical (9)

- adjusted_rand_index
- chisq
- chisq_dof
- gk_lambda
- gk_lambda_reversed
- mutual_information
- phi
- uncertainty_coefficient
- uncertainty_coefficient_reversed

2.4 Categorical-Continuous, Biserial (3)

- biserial
- point_biserial
- rank_biserial

2.5 Categorical-Continuous (7)

- anova
- calinski_harabasz
- davies_bouldin
- eta
- eta_squared
- kruskal
- silhouette

2.6 Ordinal-Ordinal, Concordance (3)

- goodman_kruskal_gamma
- kendall_tau
- somers_d

2.7 Continuous-Continuous (4)

- kendall
- pearson
- regression
- spearman

CHAPTER

THREE

QUICKSTART

3.1 Installation

Use PyPi to install the package.

```
pip install pypair
```

3.2 Confusion Matrix

A confusion matrix is typically used to judge binary classification performance. There are two variables, A and P, where A is the actual value (ground truth) and P is the predicted value. The example below shows how to use the convenience method `confusion()` and the class `ConfusionMatrix` to get association measures derived from the confusion matrix.

```
from pypair.association import confusion
from pypair.contingency import ConfusionMatrix


def get_data():
    """
    Data taken from `here <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>`_.
    A pair of binary variables, `a` and `p`, are returned.

    :return: a, p
    """
    tn = [(0, 0) for _ in range(50)]
    fp = [(0, 1) for _ in range(10)]
    fn = [(1, 0) for _ in range(5)]
    tp = [(1, 1) for _ in range(100)]
    data = tn + fp + fn + tp
    a = [a for a, _ in data]
    p = [b for _, b in data]
    return a, p


a, p = get_data()

# if you need to quickly get just one association measure
r = confusion(a, p, measure='acc')
print(r)
```

(continues on next page)

(continued from previous page)

```
print('-' * 15)

# you can also get a list of available association measures
# and loop over to call confusion(...)
# this is more convenient, but less fast
for m in ConfusionMatrix.measures():
    r = confusion(a, p, m)
    print(f'{r}: {m}')

print('-' * 15)

# if you need multiple association measures, then
# build the confusion matrix table
# this is less convenient, but much faster
matrix = ConfusionMatrix(a, p)
for m in matrix.measures():
    r = matrix.get(m)
    print(f'{r}: {m}')
```

3.3 Binary-Binary

Association measures for binary-binary variables are computed using `binary_binary()` or `BinaryTable`.

```
from pypair.association import binary_binary
from pypair.contingency import BinaryTable

get_data = lambda x, y, n: [(x, y) for _ in range(n)]
data = get_data(1, 1, 207) + get_data(1, 0, 282) + get_data(0, 1, 231) + get_data(0, 0, 242)
a = [a for a, _ in data]
b = [b for _, b in data]

for m in BinaryTable.measures():
    r = binary_binary(a, b, m)
    print(f'{r}: {m}')

print('-' * 15)

table = BinaryTable(a, b)
for m in table.measures():
    r = table.get(m)
    print(f'{r}: {m}')
```

3.4 Categorical-Categorical

Association measures for categorical-categorical variables are computed using `categorical_categorical()` or `CategoricalTable`.

```
from pypair.association import categorical_categorical
from pypair.contingency import CategoricalTable

get_data = lambda x, y, n: [(x, y) for _ in range(n)]
data = get_data(1, 1, 207) + get_data(1, 0, 282) + get_data(0, 1, 231) + get_data(0, 0, 242)
a = [a for a, _ in data]
b = [b for _, b in data]

for m in CategoricalTable.measures():
    r = categorical_categorical(a, b, m)
    print(f'{r}: {m}')

print('-' * 15)

table = CategoricalTable(a, b)
for m in table.measures():
    r = table.get(m)
    print(f'{r}: {m}')
```

3.5 Binary-Continuous

Association measures for binary-continuous variables are computed using `binary_continuous()` or `Biserial`.

```
from pypair.association import binary_continuous
from pypair.biserial import Biserial

get_data = lambda x, y, n: [(x, y) for _ in range(n)]
data = get_data(1, 1, 207) + get_data(1, 0, 282) + get_data(0, 1, 231) + get_data(0, 0, 242)
a = [a for a, _ in data]
b = [b for _, b in data]

for m in Biserial.measures():
    r = binary_continuous(a, b, m)
    print(f'{r}: {m}')

print('-' * 15)

biserial = Biserial(a, b)
for m in biserial.measures():
    r = biserial.get(m)
    print(f'{r}: {m}')
```

3.6 Ordinal-Ordinal, Concordance

Concordance measures are used for ordinal-ordinal or continuous-continuous variables using `concordance()` or `Concordance()`.

```
from pypair.association import concordance
from pypair.continuous import Concordance

a = [1, 2, 3]
b = [3, 2, 1]

for m in Concordance.measures():
    r = concordance(a, b, m)
    print(f'{r}: {m}')

print('-' * 15)

con = Concordance(a, b)
for m in con.measures():
    r = con.get(m)
    print(f'{r}: {m}')
```

3.7 Categorical-Continuous

Categorical-continuous association measures are computed using `categorical_continuous()` or `CorrelationRatio`.

```
from pypair.association import categorical_continuous
from pypair.continuous import CorrelationRatio

data = [
    ('a', 45), ('a', 70), ('a', 29), ('a', 15), ('a', 21),
    ('g', 40), ('g', 20), ('g', 30), ('g', 42),
    ('s', 65), ('s', 95), ('s', 80), ('s', 70), ('s', 85), ('s', 73)
]
x = [x for x, _ in data]
y = [y for _, y in data]
for m in CorrelationRatio.measures():
    r = categorical_continuous(x, y, m)
    print(f'{r}: {m}')

print('-' * 15)

cr = CorrelationRatio(x, y)
for m in cr.measures():
    r = cr.get(m)
    print(f'{r}: {m}')
```

3.8 Continuous-Continuous

Association measures for continuous-continuous variables are computed using `continuous_continuous()` or `Continuous`.

```
from pypair.association import continuous_continuous
from pypair.continuous import Continuous

x = [x for x in range(10)]
y = [y for y in range(10)]

for m in Continuous.measures():
    r = continuous_continuous(x, y, m)
    print(f'{r}: {m}')

print('-' * 15)

con = Continuous(x, y)
for m in con.measures():
    r = con.get(m)
    print(f'{r}: {m}')
```

3.9 Recipe

Here's a recipe in using multiprocessing to compute pairwise association with binary data.

```
import pandas as pd
import numpy as np
import random
from random import randint
from pypair.association import binary_binary
from itertools import combinations
from multiprocessing import Pool

np.random.seed(37)
random.seed(37)

def get_data(n_rows=1000, n_cols=5):
    data = [tuple([randint(0, 1) for _ in range(n_cols)]) for _ in range(n_rows)]
    cols = [f'x{i}' for i in range(n_cols)]
    return pd.DataFrame(data, columns=cols)

def compute(a, b, df):
    x = df[a]
    y = df[b]
    return f'{a}_{b}', binary_binary(x, y, measure='jaccard')

if __name__ == '__main__':
    df = get_data()

    with Pool(10) as pool:
        pairs = ((a, b, df) for a, b in combinations(df.columns, 2))
        bc = pool.starmap(compute, pairs)

```

(continues on next page)

(continued from previous page)

```
    bc = sorted(bc, key=lambda tup: tup[0])
    print(dict(bc))
```

Here's another way to use a pandas Dataframe `corr()` method to speed up pairwise association computation.

```
from random import randint

import pandas as pd

from pypair.association import binary_binary


def get_data(n_rows=1000, n_cols=5):
    data = [tuple([randint(0, 1) for _ in range(n_cols)]) for _ in range(n_rows)]
    cols = [f'x{i}' for i in range(n_cols)]
    return pd.DataFrame(data, columns=cols)


if __name__ == '__main__':
    jaccard = lambda a, b: binary_binary(a, b, measure='jaccard')
    tanimoto = lambda a, b: binary_binary(a, b, measure='tanimoto_i')

    df = get_data()
    jaccard_corr = df.corr(method=jaccard)
    tanimoto_corr = df.corr(method=tanimoto)

    print(jaccard_corr)
    print('-' * 15)
    print(tanimoto_corr)
```

3.10 Apache Spark

Spark is supported for some of the association measures. Active support is appreciated. Below are some code samples to get you started.

```
import json
from random import choice

import pandas as pd
from pyspark.sql import SparkSession

from pypair.spark import binary_binary, confusion, categorical_categorical, agreement,
↪ binary_continuous, concordance, \
    categorical_continuous, continuous_continuous


def _get_binary_binary_data(spark):
    """
    Gets dummy binary-binary data in a Spark dataframe.

    :return: Spark dataframe.
    """
    get_data = lambda x, y, n: [(x, y) * 2 for _ in range(n)]
    data = get_data(1, 1, 207) + get_data(1, 0, 282) + get_data(0, 1, 231) + get_
↪data(0, 0, 242)
```

(continues on next page)

(continued from previous page)

```
    pdf = pd.DataFrame(data, columns=['x1', 'x2', 'x3', 'x4'])
    sdf = spark.createDataFrame(pdf)
    return sdf


def _get_confusion_data(spark):
    """
    Gets dummy binary-binary data in Spark dataframe. For use with confusion matrix analysis.

    :return: Spark dataframe.
    """
    tn = [(0, 0) * 2 for _ in range(50)]
    fp = [(0, 1) * 2 for _ in range(10)]
    fn = [(1, 0) * 2 for _ in range(5)]
    tp = [(1, 1) * 2 for _ in range(100)]
    data = tn + fp + fn + tp
    pdf = pd.DataFrame(data, columns=['x1', 'x2', 'x3', 'x4'])
    sdf = spark.createDataFrame(pdf)
    return sdf


def _get_categorical_categorical_data(spark):
    """
    Gets dummy categorical-categorical data in Spark dataframe.

    :return: Spark dataframe.
    """
    x_domain = ['a', 'b', 'c']
    y_domain = ['a', 'b']

    get_x = lambda: choice(x_domain)
    get_y = lambda: choice(y_domain)
    get_data = lambda: {f'x{i}': v for i, v in enumerate((get_x(), get_y(), get_x(), get_y()))}

    pdf = pd.DataFrame([get_data() for _ in range(100)])
    sdf = spark.createDataFrame(pdf)
    return sdf


def _get_binary_continuous_data(spark):
    """
    Gets dummy `binary-continuous data <https://www.slideshare.net/MuhammadKhalil166/point-biserial-correlation-example>`_.

    :return: Spark dataframe.
    """
    data = [
        (1, 10), (1, 11), (1, 6), (1, 11), (0, 4),
        (0, 3), (1, 12), (0, 2), (0, 2), (0, 1)
    ]
    pdf = pd.DataFrame(data, columns=['gender', 'years'])
    sdf = spark.createDataFrame(pdf)
    return sdf


```

(continues on next page)

(continued from previous page)

```
def _get_concordance_data(spark):
    """
    Gets dummy concordance data.

    :return: Spark dataframe.
    """
    a = [1, 2, 3]
    b = [3, 2, 1]
    pdf = pd.DataFrame({'a': a, 'b': b, 'c': a, 'd': b})
    sdf = spark.createDataFrame(pdf)
    return sdf


def _get_categorical_continuous_data(spark):
    data = [
        ('a', 45), ('a', 70), ('a', 29), ('a', 15), ('a', 21),
        ('g', 40), ('g', 20), ('g', 30), ('g', 42),
        ('s', 65), ('s', 95), ('s', 80), ('s', 70), ('s', 85), ('s', 73)
    ]
    data = [tup * 2 for tup in data]
    pdf = pd.DataFrame(data, columns=['x1', 'x2', 'x3', 'x4'])
    sdf = spark.createDataFrame(pdf)
    return sdf


def _get_continuous_continuous_data(spark):
    """
    Gets dummy continuous-continuous data.
    See `site <http://onlinestatbook.com/2/describing_bivariate_data/calculation.html>
↪`_.

    :return: Spark dataframe.
    """
    data = [
        (12, 9),
        (10, 12),
        (9, 12),
        (14, 11),
        (10, 8),
        (11, 9),
        (10, 9),
        (10, 6),
        (14, 12),
        (9, 11),
        (11, 12),
        (10, 7),
        (11, 13),
        (15, 14),
        (8, 11),
        (11, 11),
        (9, 8),
        (9, 9),
        (10, 11),
        (12, 9),
        (11, 12),
        (10, 12),
        (9, 7),
```

(continues on next page)

(continued from previous page)

```
        (7, 9),
        (12, 14)
    ]
    pdf = pd.DataFrame([item * 2 for item in data], columns=['x1', 'x2', 'x3', 'x4'])
    sdf = spark.createDataFrame(pdf)
    return sdf


spark = None

try:
    # create a spark session
    spark = (SparkSession.builder
             .master('local[4]')
             .appName('local-testing-pyspark')
             .getOrCreate())

    # create some spark dataframes
    bin_sdf = _get_binary_binary_data(spark)
    con_sdf = _get_confusion_data(spark)
    cat_sdf = _get_categorical_categorical_data(spark)
    bcn_sdf = _get_binary_continuous_data(spark)
    crd_sdf = _get_concordance_data(spark)
    ccn_sdf = _get_categorical_continuous_data(spark)
    cnt_sdf = _get_continuous_continuous_data(spark)

    # call these methods to get the association measures
    bin_results = binary_binary(bin_sdf).collect()
    con_results = confusion(con_sdf).collect()
    cat_results = categorical_categorical(cat_sdf).collect()
    agr_results = agreement(bin_sdf).collect()
    bcn_results = binary_continuous(bcn_sdf, binary=['gender'], continuous=['years']).
↪collect()
    crd_results = concordance(crd_sdf).collect()
    ccn_results = categorical_continuous(ccn_sdf, ['x1', 'x3'], ['x2', 'x4']).
↪collect()
    cnt_results = continuous_continuous(cnt_sdf).collect()

    # convert the lists to dictionaries
    bin_results = {tup[0]: tup[1] for tup in bin_results}
    con_results = {tup[0]: tup[1] for tup in con_results}
    cat_results = {tup[0]: tup[1] for tup in cat_results}
    agr_results = {tup[0]: tup[1] for tup in agr_results}
    bcn_results = {tup[0]: tup[1] for tup in bcn_results}
    crd_results = {tup[0]: tup[1] for tup in crd_results}
    ccn_results = {tup[0]: tup[1] for tup in ccn_results}
    cnt_results = {tup[0]: tup[1] for tup in cnt_results}

    # pretty print
    to_json = lambda r: json.dumps({f'{k[0]}_{k[1]}': v for k, v in r.items()},␣
↪indent=1)
    print(to_json(bin_results))
    print('-' * 10)
    print(to_json(con_results))
    print('*' * 10)
    print(to_json(cat_results))
    print('~' * 10)
```

(continues on next page)

(continued from previous page)

```
    print(to_json(agr_results))
    print('-' * 10)
    print(to_json(bcn_results))
    print('=' * 10)
    print(to_json(crd_results))
    print('`' * 10)
    print(to_json(ccn_results))
    print('/' * 10)
    print(to_json(cnt_results))
except Exception as e:
    print(e)
finally:
    try:
        spark.stop()
        print('closed spark')
    except Exception as e:
        print(e)
```

CHAPTER

FOUR

SELECTED DEEP DIVES

Let's go into some association measures in more details.

4.1 Binary association

The association between binary variables have been studied prolifically in the last 100 years [SSC10][Cox70][Rey84][War19][Pro]. A binary variable has only two values. It is typical to re-encode these values into 0 or 1. How and why each of these two values are mapped to 0 or 1 is subjective, arbitrary and/or context-specific. For example, if we have a variable that captures the handedness, favoring left or right hand, of a person, we could map left to 0 and right to 1, or, left to 1 and right to 0. The 0-1 value representation of a binary variable's values is the common foundation for understanding association. Below is a contingency table created from two binary variables. Notice the main values of the tables are a, b, c and d.

- $a = N_{11}$ is the count of when the two variables have a value of 1
- $b = N_{10}$ is the count of when the row variable has a value of 1 and the column variable has a value of 0
- $c = N_{01}$ is the count of when the row variable has a value of 0 and the column variable has a value of 1
- $d = N_{00}$ is the count of when the two variables have a value of 0

Also, look at how the table is structured with the value 1 coming before the value 0 in both the rows and columns.

Table 1: Contingency table for two binary variables

	1	0	Total
1	a	b	a + b
0	c	d	c + d
Total	a + c	b + d	n = a + b + c + d

Note that a and d are *matches* and b and c are *mismatches*. Sometimes, depending on the context, matching on 0 is not considered a match. For example, if 1 is the presence of something and 0 is the absence, then an observation of absence and absence does not really feel right to consider as a match (you cannot say two things match on what is not there). Additionally, when 1 is presence and 0 is absence, and the data is very sparse (a lot of 0's compared to 1's), considering absence and absence as matching will make it appear that the two variables are very similar.

In [SSC10], there are 76 similarity and distance measures identified (some are not unique and/or redundant). Similarity is how *alike* are two things, and distance is how *different* are two things; or, in other words, similarity is how close are two things and distance is how far apart are two things. If a similarity or distance measure produces a value in $[0, 1]$, then we can convert between the two easily.

- If s is the similarity, then $d = 1 - s$ is the distance.
- If d is the distance, then $s = 1 - d$ is the similarity.

If we use a contingency table to summarize a bivariate binary data, the following similarity and distance measures may be derived entirely from *a*, *b*, *c* and/or *d*. The general pattern is that similarity and distance is always a ratio. The numerator in the ratio defines what we are interested in measuring. When we have *a* and/or *d* in the numerator, it is likely we are measuring similarity; when we have *b* and/or *c* in the numerator, it is likely we are measuring distance. The denominator considers what is important in considering; is it the matches, mismatches or both? The following tables list some identified similarity and distance measures based off of 2 x 2 contingency tables.

Table 2: Similarity measures for 2 x 2 contingency table [2010:choi][2019:warrens][2020:psu-binary]

Name	Computation
3W-Jaccard	$\frac{3a}{3a+b+c}$
Ample	$\left\lvert\frac{a(c+d)}{c(a+b)}\right\rvert$
Anderberg	$\frac{\sigma-\sigma'}{2n}$
Baroni-Urbani-Buser-I	$\frac{\sqrt{ad}+a}{\sqrt{ad}+a+b+c}$
Baroni-Urbani-Buser-II	$\frac{\sqrt{ad}+a-(b+c)}{\sqrt{ad}+a+b+c}$
Braun-Banquet	$\frac{a}{\max(a+b,a+c)}$
Cole [SSC10][War19]	$\frac{\sqrt{2}(ad-bc)}{\sqrt{(ad-bc)^2-(a+b)(a+c)(b+d)(c+d)}}$
	$\frac{ad-bc}{\min((a+b)(a+c),(b+d)(c+d))}$
Cosine	$\frac{a}{\sqrt{(a+b)(a+c)}}$
Dennis	$\frac{ad-bc}{\sqrt{n(a+b)(a+c)}}$
Dice; Czekanowski; Nei-Li	$\frac{2a}{2a+b+c}$
Disperson	$\frac{ad-bc}{(a+b+c+d)^2}$
Driver-Kroeber	$\frac{a}{2}\left(\frac{1}{a+b}+\frac{1}{a+c}\right)$
Eyraud	$\frac{n^2(na-(a+b)(a+c))}{(a+b)(a+c)(b+d)(c+d)}$
Fager-McGowan	$\frac{a}{\sqrt{(a+b)(a+c)}}-\frac{max(a+b,a+c)}{2}$
Faith	$\frac{a+0.5d}{a+b+c+d}$
Forbes-II	$\frac{na-(a+b)(a+c)}{n\min(a+b,a+c)-(a+b)(a+c)}$
Forbesi	$\frac{na}{(a+b)(a+c)}$
Fossum	$\frac{n(a-0.5)^2}{(a+b)(a+c)}$
Gilbert-Wells	$\log a-\log n-\log\frac{a+b}{n}-\log\frac{a+c}{n}$
Goodman-Kruskal	$\frac{\sigma-\sigma'}{2n-\sigma'}$
	$\sigma=\max(a,b)+\max(c,d)+\max(a,c)+\max(b,d)$
	$\sigma'=\max(a+c,b+d)+\max(a+b,c+d)$
Gower	$\frac{a+d}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$
Gower-Legendre	$\frac{a+d}{a+0.5b+0.5c+d}$
Hamann	$\frac{(a+d)-(b+c)}{a+b+c+d}$
Inner Product	$a+d$
Intersection	a
Jaccard [Wikb]	$\frac{a}{a+b+c}$
Johnson	$\frac{a}{a+b}+\frac{a}{a+c}$
Kulczynski-I	$\frac{a}{b+c}$
Kulczynski-II	$\frac{0.5a(2a+b+c)}{(a+b)(a+c)}$
	$\frac{1}{2}\left(\frac{a}{a+b}+\frac{a}{a+c}\right)$
McConnaughey	$\frac{a^2-bc}{(a+b)(a+c)}$

continues on next page

Table 2 – continued from previous page

Name	Computation
Michael	$\frac{4(ad-bc)}{(a+d)^2+(b+c)^2}$
Mountford	$\frac{a}{0.5(ab+ac)+bc}$
Ochiai-I [Exc]; Otsuka; Fowlkes-Mallows Index [Wika]	$\frac{a}{\sqrt{(a+b)(a+c)}}$
	$\sqrt{\frac{a}{a+b}\frac{a}{a+c}}$
Ochiai-II	$\frac{ad}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$
Pearson-Heron-I	$\frac{ad-bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$
Pearson-Heron-II	$\cos\left(\frac{\pi\sqrt{bc}}{\sqrt{ad}+\sqrt{bc}}\right)$
Pearson-I	$\chi^2 = \frac{n(ad-bc)^2}{(a+b)(a+c)(c+d)(b+d)}$
Pearson-II	$\sqrt{\frac{\chi^2}{n+\chi^2}}$
Pearson-II	$\sqrt{\frac{\rho}{n+\rho}}$
	$\rho = \frac{ad-bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$
Peirce	$\frac{ab+bc}{ab+2bc+cd}$
Roger-Tanimoto	$\frac{a+d}{a+2b+2c+d}$
Russell-Rao	$\frac{a}{a+b+c+d}$
Simpson; Overlap [Wikc]	$\frac{a}{\min(a+b,a+c)}$
Sokal-Michener; Rand Index	$\frac{a+d}{a+b+c+d}$
Sokal-Sneath-I	$\frac{a}{a+2b+2c}$
Sokal-Sneath-II	$\frac{2a+2d}{2a+b+c+2d}$
Sokal-Sneath-III	$\frac{a+d}{b+c}$
Sokal-Sneath-IV	$\frac{1}{4}\left(\frac{a}{a+b}+\frac{a}{a+c}+\frac{d}{b+d}+\frac{d}{b+d}\right)$
Sokal-Sneath-V	$\frac{ad}{(a+b)(a+c)(b+d)\sqrt{c+d}}$
Sørensen–Dice [Wikf]	$\frac{2(a+d)}{2(a+d)+b+c}$
Sorgenfrei	$\frac{a^2}{(a+b)(a+c)}$
Stiles	$\log_{10}\frac{n\left(\lvert ad-bc\rvert-\frac{n}{2}\right)^2}{(a+b)(a+c)(b+d)(c+d)}$
Tanimoto-I	$\frac{a}{2a+b+c}$
Tanimoto-II [Wikb]	$\frac{a}{b+c}$
Tarwid	$\frac{na-(a+b)(a+c)}{na+(a+b)(a+c)}$
Tarantula	$\frac{a(c+d)}{c(a+b)}$
Tetrachoric	$\frac{y-1}{y+1}$
	$y = \left(\frac{ad}{bc}\right)^{\frac{\pi}{4}}$
Tversky Index [Wikg]	$\frac{a}{a+\theta b+\phi c}$
	θ and ϕ are user-supplied parameters
Yule-Q	$\frac{ad-bc}{ad+bc}$
Yule-w	$\frac{\sqrt{ad}-\sqrt{bc}}{\sqrt{ad}+\sqrt{bc}}$

Table 3: Distance measures for 2 x 2 contingency table [2010:choi]

Name	Computation
Chord	$\sqrt{2\left(1-\frac{a}{\sqrt{(a+b)(a+c)}}\right)}$
Euclid	$\sqrt{b+c}$
Hamming; Canberra; Manhattan; Cityblock; Minkowski	$b+c$
Hellinger	$2\sqrt{1-\frac{a}{\sqrt{(a+b)(a+c)}}}$
Jaccard distance [Wikb]	$\frac{b+c}{a+b+c}$
Lance-Williams; Bray-Curtis	$\frac{b+c}{2a+b+c}$
Mean-Manhattan	$\frac{b+c}{a+b+c+d}$
Pattern Difference	$\frac{4bc}{(a+b+c+d)^2}$
Shape Difference	$\frac{n(b+c)-(b-c)^2}{(a+b+c+d)^2}$
Size Difference	$\frac{(b+c)^2}{(a+b+c+d)^2}$
Squared-Euclid	$\sqrt{(b+c)^2}$
Vari	$\frac{b+c}{4a+4b+4c+4d}$
Yule-Q	$\frac{2bc}{ad+bc}$

Instead of using *a*, *b*, *c* and *d* from a contingency table to define these association measures, it is common to use set notation. For two binary variables, X and Y, the following are equivalent.

- $|X \cap Y| = a$
- $|X \setminus Y| = b$
- $|Y \setminus X| = c$
- $|X \cup Y| = a + b + c$

You will notice that *d* does not show up in the above relationship.

4.2 Concordant, discordant, tie

Let's try to understand how to determine if a pair of observations are concordant, discordant or tied. We have made up an example dataset below having two variables X and Y. Note that there are 6 observations, and as such, each observation is associated with an index from 1 to 6. An observation has a pair of values, one for X and one for Y.

Warning: Do **not** get the *pair of values of an observation* confused with a *pair of observations*.

Table 4: Raw Data for X and Y

Index	X	Y
1	1	3
2	1	3
3	2	4
4	0	2
5	0	4
6	2	2

Because there are 6 observations, there are $\binom{6}{2} = 15$ possible pairs of observations. If we denote an observation by its corresponding index as O_i, then the observations are then as follows.

- $O_1 = (1, 3)$
- $O_2 = (1, 3)$
- $O_3 = (2, 4)$
- $O_4 = (0, 2)$
- $O_5 = (0, 4)$
- $O_6 = (2, 2)$

The 15 possible *combinations* of observation pairings are as follows.

- O_1, O_2
- O_1, O_3
- O_1, O_4
- O_1, O_5
- O_1, O_6
- O_2, O_3
- O_2, O_4
- O_2, O_5
- O_2, O_6
- O_3, O_4
- O_3, O_5
- O_3, O_6
- O_4, O_5
- O_4, O_6
- O_5, O_6

For each one of these observation pairs, we can determine if such a pair is concordant, discordant or tied. There's a couple ways to determine concordant, discordant or tie status. The easiest way to determine so is mathematically. Another way is to use rules. Both are equivalent. Because we will use abstract notation to describe these math and rules used to determine concordant, discordant or tie for each pair, and because we are striving for clarity, let's expand these observation pairs into their component pairs of values and also their corresponding X and Y indexed notation.

- $O_1, O_2 = (1, 3), (1, 3) = (X_1, Y_1), (X_2, Y_2)$
- $O_1, O_3 = (1, 3), (2, 4) = (X_1, Y_1), (X_3, Y_3)$
- $O_1, O_4 = (1, 3), (0, 2) = (X_1, Y_1), (X_4, Y_4)$
- $O_1, O_5 = (1, 3), (0, 4) = (X_1, Y_1), (X_5, Y_5)$
- $O_1, O_6 = (1, 3), (2, 2) = (X_1, Y_1), (X_6, Y_6)$
- $O_2, O_3 = (1, 3), (2, 4) = (X_2, Y_2), (X_3, Y_3)$
- $O_2, O_4 = (1, 3), (0, 2) = (X_2, Y_2), (X_4, Y_4)$
- $O_2, O_5 = (1, 3), (0, 4) = (X_2, Y_2), (X_5, Y_5)$
- $O_2, O_6 = (1, 3), (2, 2) = (X_2, Y_2), (X_6, Y_6)$

- $O_3, O_4 = (2, 4), (0, 2) = (X_3, Y_3), (X_4, Y_4)$
- $O_3, O_5 = (2, 4), (0, 4) = (X_3, Y_3), (X_5, Y_5)$
- $O_3, O_6 = (2, 4), (2, 2) = (X_3, Y_3), (X_6, Y_6)$
- $O_4, O_5 = (0, 2), (0, 4) = (X_4, Y_4), (X_5, Y_5)$
- $O_4, O_6 = (0, 2), (2, 2) = (X_4, Y_4), (X_6, Y_6)$
- $O_5, O_6 = (0, 4), (2, 2) = (X_5, Y_5), (X_6, Y_6)$

Now we can finally attempt to describe how to determine if any pair of observations is concordant, discordant or tied. If we want to use math to determine so, then, for any two pairs of observations (X_i, Y_i) and (X_j, Y_j), the following determines the status.

- concordant when $(X_j - X_i)(Y_j - Y_i) > 0$
- discordant when $(X_j - X_i)(Y_j - Y_i) < 0$
- tied when $(X_j - X_i)(Y_j - Y_i) = 0$

If we like rules, then the following determines the status.

- concordant if $X_i < X_j$ and $Y_i < Y_j$ **or** $X_i > X_j$ and $Y_i > Y_j$
- discordant if $X_i < X_j$ and $Y_i > Y_j$ **or** $X_i > X_j$ and $Y_i < Y_j$
- tied if $X_i = X_j$ **or** $Y_i = Y_j$

All pairs of observations will evaluate categorically to one of these statuses. Continuing with our dummy data above, the concordancy status of the 15 pairs of observations are as follows (where concordant is C, discordant is D and tied is T).

Table 5: Concordancy Status

(X_i, Y_i)	(X_j, Y_j)	status
$(1, 3)$	$(1, 3)$	T
$(1, 3)$	$(2, 4)$	C
$(1, 3)$	$(0, 2)$	C
$(1, 3)$	$(0, 4)$	D
$(1, 3)$	$(2, 2)$	D
$(1, 3)$	$(2, 4)$	C
$(1, 3)$	$(0, 2)$	C
$(1, 3)$	$(0, 4)$	D
$(1, 3)$	$(2, 2)$	D
$(2, 4)$	$(0, 2)$	C
$(2, 4)$	$(0, 4)$	C
$(2, 4)$	$(2, 2)$	T
$(0, 2)$	$(0, 4)$	T
$(0, 2)$	$(2, 2)$	T
$(0, 4)$	$(2, 2)$	D

In this data set, the counts are $C = 6$, $D = 5$ and $T = 4$. If we divide these counts with the total of pairs of observations, then we get the following probabilities.

- $\pi_C = \frac{C}{n2} = \frac{6}{15} = 0.40$
- $\pi_D = \frac{D}{n2} = \frac{5}{15} = 0.33$
- $\pi_T = \frac{T}{n2} = \frac{4}{15} = 0.27$

Sometimes, it is desirable to distinguish between the types of ties. There are three possible types of ties.

- T^X are ties on only X
- T^Y are ties on only Y
- T^{XY} are ties on both X and Y

Note, $T = T^X + T^Y + T^{XY}$. If we want to distinguish between the tie types, then the status of each pair of observations is as follows.

Table 6: Concordancy Status

(X_i, Y_i)	(X_j, Y_j)	status
$(1, 3)$	$(1, 3)$	T^{XY}
$(1, 3)$	$(2, 4)$	C
$(1, 3)$	$(0, 2)$	C
$(1, 3)$	$(0, 4)$	D
$(1, 3)$	$(2, 2)$	D
$(1, 3)$	$(2, 4)$	C
$(1, 3)$	$(0, 2)$	C
$(1, 3)$	$(0, 4)$	D
$(1, 3)$	$(2, 2)$	D
$(2, 4)$	$(0, 2)$	C
$(2, 4)$	$(0, 4)$	C
$(2, 4)$	$(2, 2)$	T^X
$(0, 2)$	$(0, 4)$	T^X
$(0, 2)$	$(2, 2)$	T^Y
$(0, 4)$	$(2, 2)$	D

Distinguishing between ties, in this data set, the counts are $C = 6$, $D = 5$, $T^X = 2$, $T^Y = 1$ and $T^{XY} = 1$. The probabilities of these statuses are as follows.

- $\pi_C = \frac{C}{n2} = \frac{6}{15} = 0.40$
- $\pi_D = \frac{D}{n2} = \frac{5}{15} = 0.33$
- $\pi_{T^X} = \frac{T^X}{n2} = \frac{2}{15} = 0.13$
- $\pi_{T^Y} = \frac{T^Y}{n2} = \frac{1}{15} = 0.07$
- $\pi_{T^{XY}} = \frac{T^{XY}}{n2} = \frac{1}{15} = 0.07$

There are quite a few measures of associations using concordance as the basis for strength of association.

Table 7: Association measures using concordance

Association Measure	Formula
Goodman-Kruskal's γ	$\gamma = \frac{\pi_C - \pi_D}{1 - \pi_T}$
Somers' d	$d_{Y \cdot X} = \frac{\pi_C - \pi_D}{\pi_C + \pi_D + \pi_{TY}}$
	$d_{X \cdot Y} = \frac{\pi_C - \pi_D}{\pi_C + \pi_D + \pi_{TX}}$
Kendall's tau	$\tau = \frac{C - D}{n2}$

Note: Sometimes *Somers' d* is written as *Somers' D*, *Somers' Delta* or even incorrectly as *Somer's D* [Gle][Wike]. Somers' d has two versions, one that is symmetric and one that is asymmetric. The asymmetric Somers' d is the one

most typically referred to [Gle]. The definition of Somers' d presented here is the asymmetric one, which explains $d_{Y \cdot X}$ and $d_{X \cdot Y}$.

4.3 Goodman-Kruskal's λ

Goodman-Kruskal's lambda $\lambda_{A|B}$ measures the *proportional reduction in error* `PRE` for two categorical variables, A and B, when we want to understand how knowing B reduces the probability of an error in predicting A. $\lambda_{A|B}$ is estimated as follows.

$\lambda_{A|B} = \frac{P_E - P_{E|B}}{P_E}$

Where,

- $P_E = 1 - \frac{\max_c N_{+c}}{N_{++}}$
- $P_{E|B} = 1 - \frac{\sum_r \max_c N_{rc}}{N_{++}}$

In meaningful language.

- P_E is the probability of an error in predicting A
- $P_{E|B}$ is the probability of an error in predicting A given knowledge of B

The terms N_{+c}, N_{rc} and N_{++} comes from the contingency table we build from A and B (A is in the columns and B is in the rows) and denote the column marginal for the *c-th* column, total count for the *r-th* and *c-th* cell and total, correspondingly. To be clear.

- N_{+c} is the column marginal for the *c-th* column
- N_{rc} is total count for the *r-th* and *c-th* cell
- N_{++} is total number of observations

The contingency table induced with A in the columns and B in the rows will look like the following. Note that A has *C* columns and B has *R* rows, or, in other words, A has *C* values and B has *R* values.

Table 8: Contingency Table for A and B

	A_1	A_2	$\cdots$	A_C
B_1	N_{11}	N_{12}	$\cdots$	N_{1C}
B_2	N_{21}	N_{22}	$\cdots$	N_{2C}
$\vdots$	$\vdots$	$\vdots$		$\vdots$
B_R	N_{R1}	N_{R2}	$\cdots$	N_{RC}

The table above only shows the cell counts $N_{11}, N_{12}, \ldots, N_{RC}$ and **not** the row and column marginals. Below, we expand the contingency table to include

- the row marginals $N_{1+}, N_{2+}, \ldots, N_{R+}$, as well as,
- the column marginals $N_{+1}, N_{+2}, \ldots, N_{+C}$.

Table 9: Contingency Table for A and B

	A_1	A_2	$\cdots$	A_C	
B_1	N_{11}	N_{12}	$\cdots$	N_{1C}	N_{1+}
B_2	N_{21}	N_{22}	$\cdots$	N_{2C}	N_{2+}
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\vdots$
B_R	N_{R1}	N_{R2}	$\cdots$	N_{RC}	N_{R+}
	N_{+1}	N_{+2}	$\cdots$	N_{+C}	N_{++}

Note that the row marginal for a row is the sum of the values across the columns, and the column marginal for a colum is the sum of the values down the rows.

- $N_{R+} = \sum_C N_{RC}$
- $N_{+C} = \sum_R N_{RC}$

Also, N_{++} is just the sum over all the cells (excluding the row and column marginals). N_{++} is really just the sample size.

- $N_{++} = \sum_R \sum_C N_{RC}$

Let's go back to computing P_E and $P_{E|B}$.

P_E is given as follows.

- $P_E = 1 - \frac{\max_c N_{+c}}{N_{++}}$

$\max_c N_{+c}$ is returning the maximum of the column marginals, and $\frac{\max_c N_{+c}}{N_{++}}$ is just a probability. What probability is this one? It is the largest probability associated with a value of A (specifically, the value of A with the largest counts). If we were to predict which value of A would show up, we would choose the value of A with the highest probability (it is the most likely). We would be correct $\frac{\max_c N_{+c}}{N_{++}}$ percent of the time, and we would be wrong $1 - \frac{\max_c N_{+c}}{N_{++}}$ percent of the time. Thus, P_E is the error in predicting A (knowing nothing else other than the distribution, or *probability mass function* `PMF` of A).

$P_{E|B}$ is given as follows.

- $P_{E|B} = 1 - \frac{\sum_r \max_c N_{rc}}{N_{++}}$

What is $\max_c N_{rc}$ giving us? It is giving us the maximum cell count for the *r-th* row. $\sum_r \max_c N_{rc}$ adds up the all the largest values in each row, and $\frac{\sum_r \max_c N_{rc}}{N_{++}}$ is again a probability. What probability is this one? This probability is the one associated with predicting the value of A when we know B. When we know what the value of B is, then the value of A should be the one with the largest count (it has the highest probability, or, equivalently, the highest count). When we know the value of B and by always choosing the value of A with the highest count associated with that value of B, we are correct $\frac{\sum_r \max_c N_{rc}}{N_{++}}$ percent of the time and incorrect $1 - \frac{\sum_r \max_c N_{rc}}{N_{++}}$ percent of the time. Thus, $P_{E|B}$ is the error in predicting A when we know the value of B and the PMF of A given B.

The expression $P_E - P_{E|B}$ is the reduction in the probability of an error in predicting A given knowledge of B. This expression represents the *reduction in error* in the phrase/term `PRE`. The proportional part in `PRE` comes from the expression $\frac{P_E - P_{E|B}}{P_E}$, which is a proportion.

What $\lambda_{A|B}$ is trying to compute is the reduction of error in predicting A when we know B. Did we reduce any prediction error of A by knowing B?

- When $\lambda_{A|B} = 0$, this value means that knowing B did not reduce any prediction error in A. The only way to get $\lambda_{A|B} = 0$ is when $P_E = P_{E|B}$.
- When $\lambda_{A|B} = 1$, this value means that knowing B completely reduced all prediction errors in A. The only way to get $\lambda_{A|B} = 1$ is when $P_{E|B} = 0$.

Generally speaking, $\lambda_{A|B} \neq \lambda_{B|A}$, and λ is thus an asymmetric association measure. To compute $\lambda_{B|A}$, simply put B in the columns and A in the rows and reuse the formulas above.

Furthermore, λ can be used in studies of causality [Lie83]. We are not saying it is appropriate or even possible to entertain causality with just two variables alone [Pea20][Pea16][Pea09][Pea88], but, when we have two categorical variables and want to know which is likely the cause and which the effect, the asymmetry between $\lambda_{A|B}$ and $\lambda_{B|A}$ may prove informational [Wikd]. Causal analysis based on two variables alone has been studied [NIP].

CHAPTER

FIVE

BIBLIOGRAPHY

CHAPTER

SIX

PYPAIR

6.1 Contingency Table Analysis

These are the basic contingency tables used to analyze categorical data.

- CategoricalTable
- BinaryTable
- ConfusionMatrix
- AgreementTable

class pypair.contingency.**AgreementMixin**
Bases: object

Agreement computations.

property chohen_k
Computes Cohen's κ.

- $\kappa = \frac{\theta_1 - \theta_2}{1 - \theta_2}$
- $\theta_1 = \sum_i p_{ii}$
- $\theta_2 = \sum_i p_{i+} p_{+i}$

Returns κ.

property cohen_light_k
Cohen-Light κ. κ is a measure of conditional agreement. Several κ, one for each unique value, will be computed and returned.

- $\kappa = \frac{\theta_1 - \theta_2}{1 - \theta_2}$
- $\theta_1 = \frac{p_{ii}}{p_{i+}}$
- $\theta_2 = p_{+i}$

Returns A list of κ.

class pypair.contingency.**AgreementStats**(*table*)
Bases: *pypair.contingency.AgreementMixin*, *pypair.contingency.ContingencyTable*

Computes agreement stats.

__init__(*table*)
ctor.

Parameters **table** – Contingency table.

class pypair.contingency.**AgreementTable**(*a*, *b*, *a_vals=None*, *b_vals=None*)
Bases: *pypair.contingency.AgreementMixin*, *pypair.contingency.ContingencyTable*

Represents a contingency table for agreement data against one variable. The variable is typically a rating variable (e.g. dislike, neutral, like), and the data is a pairing of ratings over the same set of items. The agreement table that is induced by the data is typically squared, where the number of rows and columns are equal.

__init__(*a*, *b*, *a_vals=None*, *b_vals=None*)
ctor.

Parameters

- **a** – Categorical variable.
- **b** – Categorical variable.
- **a_vals** – Values in *a*. Default *None*; figure out empirically.
- **b_vals** – Values in *b*. Default *None*; figure out empirically.

class pypair.contingency.**BinaryMixin**
Bases: object

Binary computations based off of *a*, *b*, *c* and *d* from a 2x2 contingency table.

property ample
Ample

$\left|\frac{a(c+d)}{c(a+b)}\right|$

Returns Ample.

property anderberg
Anderberg

$\frac{\sigma-\sigma'}{2n}$

Returns Anderberg.

property baroni_urbani_buser_i
Baroni-Urbani-Buser-I

$\frac{\sqrt{ad}+a}{\sqrt{ad}+a+b+c}$

Returns Baroni-Urbani-Buser-I.

property baroni_urbani_buser_ii
Baroni-Urbani-Buser-II

$\frac{\sqrt{ad}+a-(b+c)}{\sqrt{ad}+a+b+c}$

Returns Baroni-Urbani-Buser-II.

property braun_banquet
Braun-Banquet

$\frac{a}{\max(a+b,a+c)}$

Returns Braun-Banquet.

property chisq
χ^2 (alias for Pearson-I)

Returns χ^2.

property chord
Chord

$$\sqrt{2\left(1-\frac{a}{\sqrt{(a+b)(a+c)}}\right)}$$

Returns Chord (distance).

property cole_i
Cole-I

$$\frac{\sqrt{2}(ad-bc)}{\sqrt{(ad-bc)^2-(a+b)(a+c)(b+d)(c+d)}}$$

Returns Cole-I.

property cole_ii
Cole-II

$$\frac{ad-bc}{\min((a+b)(a+c),(b+d)(c+d))}$$

Returns Cole-II.

property contingency_coefficient
Contingency coefficient.

Returns Contingency coefficient.

property cosine
Cosine

$$\frac{a}{(a+b)(a+c)}$$

Returns Cosine.

property cramer_v
Cramer's V.

Returns Cramer's V.

property dennis
Dennis

$$\frac{ad-bc}{\sqrt{n(a+b)(a+c)}}$$

Returns Dennis.

property dice
Dice; Czekanowski; Nei-Li

$$\frac{2a}{2a+b+c}$$

Returns Dice.

property disperson
Disperson

$$\frac{ad-bc}{(a+b+c+d)^2}$$

Returns Disperson.

property driver_kroeber
Driver-Kroeber

$$\frac{a}{2}\left(\frac{1}{a+b}+\frac{1}{a+c}\right)$$

Returns Driver-Kroeber.

property euclid
Euclid

$\sqrt{b+c}$

Returns Euclid (distance).

property eyraud
Eyraud

$\frac{n^2(na-(a+b)(a+c))}{(a+b)(a+c)(b+d)(c+d)}$

Returns Eyraud.

property fager_mcgowan
Fager-McGowan

$\frac{a}{\sqrt{(a+b)(a+c)}} - \frac{max(a+b,a+c)}{2}$

Returns Fager-McGowan.

property faith
Faith

$\frac{a+0.5d}{a+b+c+d}$

Returns Faith.

property forbes_ii
Forbes-II

$\frac{na-(a+b)(a+c)}{n\min(a+b,a+c)-(a+b)(a+c)}$

Returns Forbes-II.

property forbesi
Forbesi

$\frac{na}{(a+b)(a+c)}$

Returns Forbesi.

property fossum
Fossum

$\frac{n(a-0.5)^2}{(a+b)(a+c)}$

Returns Fossum.

property gilbert_wells
Gilbert-Wells

$\log a - \log n - \log \frac{a+b}{n} - \log \frac{a+c}{n}$

Returns Gilbert-Wells.

property goodman_kruskal
Goodman-Kruskal

$\frac{\sigma-\sigma'}{2n-\sigma'}$

Returns Goodman-Kruskal.

property gower
Gower

$\frac{a+d}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$

Returns Gower.

property gower_legendre
Gower-Legendre

$\frac{a+d}{a+0.5b+0.5c+d}$

Returns Gower-Legendre.

property hamann
Hamann.

$\frac{(a+d)-(b+c)}{a+b+c+d}$

Returns Hamann.

property hamming
Hamming; Canberra; Manhattan; Cityblock; Minkowski

$b + c$

Returns Hamming (distance).

property hellinger
Hellinger

$2\sqrt{1 - \frac{a}{\sqrt{(a+b)(a+c)}}}$

Returns Hellinger (distance).

property inner_product
Inner-product.

$a + d$

Returns Inner-product.

property intersection
Intersection

a

Returns Intersection.

property jaccard
Jaccard

$\frac{a}{a+b+c}$

Returns Jaccard.

property jaccard_3w
3W-Jaccard

$\frac{3a}{3a+b+c}$

Returns 3W-Jaccard.

property jaccard_distance
Jaccard

$\frac{b+c}{a+b+c}$

Returns Jaccard (distance).

property johnson
Johnson.

$\frac{a}{a+b} + \frac{a}{a+c}$

Returns Johnson.

property kulczynski_ii
Kulczynski-II

$\frac{0.5a(2a+b+c)}{(a+b)(a+c)}$

Returns Kulczynski-II.

property kulczynski_i
Kulczynski-I

$\frac{a}{b+c}$

Returns Kulczynski-I.

property lance_williams
Lance-Williams; Bray-Curtis

$\frac{b+c}{2a+b+c}$

Returns Lance-Williams (distance).

property mcconnaughey
McConnaughey

$\frac{a^2-bc}{(a+b)(a+c)}$

Returns McConnaughey.

property mcnemar_test
McNemar's test.

Returns A tuple. First element is chi-square test statistics. Second element is p-value.

property mean_manhattan
Mean-Manhattan

$\frac{b+c}{a+b+c+d}$

Returns Mean-Manhattan (distance).

property michael
Michael

$\frac{4(ad-bc)}{(a+d)^2+(b+c)^2}$

Returns Michael.

property mountford
Mountford

$\frac{a}{0.5(ab+ac)+bc}$

Returns Mountford.

property ochia_i
Ochia-I

Also known as Fowlkes-Mallows Index. This measure is typically used to judge the similarity between two clusters. A larger value indicates that the clusters are more similar.

$\frac{a}{\sqrt{(a+b)(a+c)}}$

Returns Ochai-I.

property ochia_ii

Ochia-II

$\frac{ad}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$

Returns Ochia-II.

property odds_ratio

Odds ratio. The odds ratio is also referred to as the *cross-product ratio*.

Returns Odds ratio.

property pattern_difference

Pattern difference

$\frac{4bc}{(a+b+c+d)^2}$

Returns Pattern difference (distance).

property pearson_heron_i

Pearson-Heron-I

$\frac{ad-bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$

Returns Pearson-Heron-I.

property pearson_heron_ii

Pearson-Heron-II

$\sqrt{\frac{\chi^2}{n+\chi^2}}$

Returns Pearson-Heron-II.

property pearson_i

Pearson-I

$\chi^2 = \frac{n(ad-bc)^2}{(a+b)(a+c)(c+d)(b+d)}$

Returns Pearson-I.

property peirce

Peirce

$\frac{ab+bc}{ab+2bc+cd}$

Returns Peirce.

property person_ii

Pearson-II

$\sqrt{\frac{\rho}{n+\rho}}$

- $\rho = \frac{ad-bc}{\sqrt{(a+b)(a+c)(b+d)(c+d)}}$

Returns Pearson-II.

property roger_tanimoto

Roger-Tanimoto

$\frac{a+d}{a+2b+2c+d}$

Returns Roger-Tanimoto.

property russel_rao
Russel-Rao

$$\frac{a}{a+b+c+d}$$

Returns Russel-Rao.

property shape_difference
Shape difference

$$\frac{n(b+c)-(b-c)^2}{(a+b+c+d)^2}$$

Returns Shape difference (distance).

property simpson
Simpson (or Overlap).

$$\frac{a}{\min(a+b,a+c)}$$

Returns Simpson.

property size_difference
Size difference

$$\frac{(b+c)^2}{(a+b+c+d)^2}$$

Returns Size difference (distance).

property sokal_michener
Sokal-Michener

$$\frac{a+d}{a+b+c+d}$$

Returns Sokal-Michener.

property sokal_sneath_i
Sokal-Sneath-I

$$\frac{a}{a+2b+2c}$$

Returns Sokal-Sneath-I.

property sokal_sneath_ii
Sokal-Sneath-II

$$\frac{2a+2d}{2a+b+c+2d}$$

Returns Sokal-Sneath-II.

property sokal_sneath_iii
Sokal-Sneath-III

$$\frac{a+d}{b+c}$$

Returns Sokal-Sneath-III.

property sokal_sneath_iv
Sokal-Sneath-IV

$$\frac{ad}{(a+b)(a+c)(b+d)\sqrt{c+d}}$$

Returns Sokal-Sneath-IV.

property sokal_sneath_v
Sokal-Sneath-V

$\frac{1}{4}\left(\frac{a}{a+b}+\frac{a}{a+c}+\frac{d}{b+d}+\frac{d}{b+d}\right)$

Returns Sokal-Sneath-V.

property sorensen_dice
Sørensen–Dice

$\frac{2(a+d)}{2(a+d)+b+c}$

Returns Sørensen–Dice,

property sorgenfrei
Sorgenfrei

$\frac{a^2}{(a+b)(a+c)}$

Returns Sorgenfrei.

property stiles
Stiles

$\log_{10}\frac{n\left(|ad-bc|-\frac{n}{2}\right)^2}{(a+b)(a+c)(b+d)(c+d)}$

Returns Stiles.

property tanimoto_distance
Tanimoto similarity and distance.

Returns Tanimoto distance.

property tanimoto_i
Tanimoto-I

$\frac{a}{2a+b+c}$

Returns Tanimoto-I.

property tanimoto_ii
Tanimoto-II

$\frac{a}{b+c}$

Returns Tanimoto-II.

property tarantula
Tarantula

$\frac{a(c+d)}{c(a+b)}$

Returns Tarantula.

property tarwid
Tarwind

$\frac{na-(a+b)(a+c)}{na+(a+b)(a+c)}$

Returns Tarwind.

property tetrachoric
Tetrachoric correlation ranges from $[-1, 1]$, where 0 indicates no agreement, 1 indicates perfect agreement and -1 indicates perfect disagreement.

- if $b = 0$ or $c = 0$, 1.0

- if $a = 0$ or $b = 0$, -1.0
- else, $\frac{y-1}{y+1}, y = \left(\frac{da}{bc}\right)^{\frac{\pi}{4}}$

References

- Tetrachoric correlation.
- Tetrachoric Correlation: Definition, Examples, Formula.
- Tetrachoric Correlation Estimation.

Returns Tetrachoric correlation.

property tschuprow_t
Tschuprow's T.

Returns Tschuprow's T.

tversky_index(*theta=1*, *phi=0*)
Compute's Tversky's Index.

$\frac{a}{a+\theta b+\phi c}$

θ and ϕ are typically between $[0, 1]$ and $\theta + \phi = 1$.

Parameters

- **theta** – Weight $[0, 1]$ of how important match on row variable is. Default 1.
- **phi** – Weight $[0, 1]$ of how important match on column variable is. Default 0.

Returns Tversky's Index.

property vari
Vari

$\frac{b+c}{4a+4b+4c+4d}$

Returns Vari (distance).

property yule_q
Yule's Q

$\frac{ad-bc}{ad+bc}$

Also, Yule's Q is based off of the odds ratio or cross-product ratio, α.

$Q = \frac{\alpha-1}{\alpha+1}$

Yule's Q is the same as Goodman-Kruskal's λ for 2 x 2 contingency tables and is also a measure of proportional reduction in error (PRE).

Returns Yule's Q.

property yule_q_difference
Yule's q

$\frac{2bc}{ad+bc}$

Returns Yule's q (distance).

property yule_w
Yule's w

$\frac{\sqrt{ad}-\sqrt{bc}}{\sqrt{ad}+\sqrt{bc}}$

Returns Yule's w.

property yule_y

Yule's Y is based off of the odds ratio or cross-product ratio, α.

$Y = \frac{\sqrt{\alpha}-1}{\sqrt{\alpha}+1}$

Returns Yule's Y.

class pypair.contingency.**BinaryStats**(*table*)

Bases: *pypair.contingency.CategoricalMixin*, *pypair.contingency.BinaryMixin*, *pypair.contingency.ContingencyTable*

Computes binary stats.

__init__(*table*)

ctor.

Parameters **table** – Contingency table.

class pypair.contingency.**BinaryTable**(*a*, *b*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)

Bases: *pypair.contingency.CategoricalMixin*, *pypair.contingency.BinaryMixin*, *pypair.contingency.ContingencyTable*

Represents a contingency table for binary variables.

__init__(*a*, *b*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)

ctor.

Parameters

- **a** – Iterable list.
- **b** – Iterable list.
- **a_0** – The zero value for a. Defaults to 0.
- **a_1** – The one value for a. Defaults to 1.
- **b_0** – The zero value for b. Defaults to 0.
- **b_1** – The zero value for b. Defaults to 1.

class pypair.contingency.**CategoricalMixin**

Bases: object

Categorical computations based off a contingency table.

property adjusted_rand_index

The Adjusted Rand Index (ARI) should yield a value between [0, 1], however, negative values can also arise when the index is less than the expected value. This function uses *binom()* from *scipy.special*, and when n >= 300, the results are too large and may cause overflow.

TODO: use a different way to compute binomial coefficient

References

- Adjusted Rand Index.
- Python binomial coefficient.

Returns Adjusted Rand Index.

property chisq

The chi-square statistic χ^2, is defined as follows.

$\sum_i \sum_j \frac{(O_{ij}-E_{ij})^2}{E_{ij}}$

In a contingency table, O_ij is the observed cell count corresponding to the i row and j column. E_ij is the expected cell count corresponding to the i row and j column.

$E_i = \frac{N_{i*}N_{*j}}{N}$

Where N_{i*} is the i-th row marginal, N_{*j} is the j-th column marginal and N is the sum of all the values in the contingency cells (or the total size of the data).

References

- Chi-Square (2) Statistic Definition

Returns Chi-square statistic.

property chisq_dof

Returns the degrees of freedom form χ^2, which is defined as $(R-1)(C-1)$, where R is the number of rows and C is the number of columns in a contingency table induced by two categorical variables.

Returns Degrees of freedom.

property gk_lambda

Goodman-Kruskal's lambda is the *proportional reduction in error* of predicting one variable *b* given another *a*: $\lambda_{B|A}$.

- The probability of an error in predicting the column category: $P_e = 1 - \frac{\max_c N_{*c}}{N}$
- The probability of an error in predicting the column category given the row category: $P_{e|r} = 1 - \frac{\sum_r \max_c N_{rc}}{N}$

Where,

- $\max_c N_{*c}$ is the maximum of the column marginals
- $\sum_r \max_c N_{rc}$ is the sum over the maximum value per row
- N is the total

Thus, $\lambda_{B|A} = \frac{P_e - P_{e|r}}{P_e}$.

The way the contingency table is setup by default is that *a* is on the rows and *b* is on the columns. Note that Goodman-Kruskal's lambda is not symmetric: $\lambda_{B|A}$ does not necessarily equal $\lambda_{A|B}$. By default, $\lambda_{B|A}$ is computed, but if you desire the reverse, use *goodman_kruskal_lambda_reversed()*.

References

- Goodman-Kruskal's lambda.
- Correlation.

Returns Goodman-Kruskal's lambda.

property gk_lambda_reversed

Computes $\lambda_{A|B}$.

Returns Goodman-Kruskal's lambda.

property mutual_information

The mutual information between two variables X and Y is denoted as $I(X;Y)$. $I(X;Y)$ is unbounded and in the range $[0, \infty]$. A higher mutual information value implies strong association. The formula for $I(X;Y)$ is defined as follows.

$I(X;Y) = \sum_y \sum_x P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$

Returns Mutual information.

property phi
Gets ϕ.

$\phi = \sqrt{\frac{\chi^2}{N}}$

Returns ϕ.

property uncertainty_coefficient
The uncertainty coefficient $U(X|Y)$ for two variables X and Y is defined as follows.

$U(X|Y) = \frac{I(X;Y)}{H(X)}$

Where,

- $H(X) = -\sum_x P(x) \log P(x)$
- $I(X;Y) = \sum_y \sum_x P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$

$H(X)$ is called the entropy of X and $I(X;Y)$ is the mutual information between X and Y. Note that $I(X;Y) < H(X)$ and both values are positive. As such, the uncertainty coefficient may be viewed as the normalized mutual information between X and Y and in the range $[0, 1]$.

Returns Uncertainty coefficient.

property uncertainty_coefficient_reversed
Uncertainty coefficient.

Returns Uncertainty coefficient.

class pypair.contingency.**CategoricalStats**(*table*)
Bases: *pypair.contingency.CategoricalMixin*, *pypair.contingency.ContingencyTable*

Computes categorical stats.

__init__(*table*)
ctor.

Parameters **table** – Contingency table.

class pypair.contingency.**CategoricalTable**(*a*, *b*, *a_vals=None*, *b_vals=None*)
Bases: *pypair.contingency.CategoricalMixin*, *pypair.contingency.ContingencyTable*

Represents a contingency table for categorical variables.

References

- Contingency table
- More Correlation Coefficients

__init__(*a*, *b*, *a_vals=None*, *b_vals=None*)
ctor. If *a_vals* or *b_vals* are *None*, then the possible values will be determined empirically from the data.

Parameters

- **a** – Iterable list.
- **b** – Iterable list.
- **a_vals** – All possible values in a. Defaults to *None*.
- **b_vals** – All possible values in b. Defaults to *None*.

class pypair.contingency.**ConfusionMatrix**(*a*, *b*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)

Bases: *pypair.contingency.ConfusionMixin*, *pypair.contingency.ContingencyTable*

Represents a confusion matrix. The confusion matrix looks like what is shown below for two binary variables *a* and *b*; *a* is in the rows and *b* in the columns. Most of the statistics around performance comes from the counts of *TN*, *FN*, *FP* and *TP*.

Table 1: Confusion Matrix

	b=0	b=1
a=0	TN	FP
a=1	FN	TP

__init__(*a*, *b*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)

ctor. Note that *a* is the ground truth and *b* is the prediction.

Parameters

- **a** – Binary variable (iterable). Ground truth.
- **b** – Binary variable (iterable). Prediction.
- **a_0** – The zero value for a. Defaults to 0.
- **a_1** – The one value for a. Defaults to 1.
- **b_0** – The zero value for b. Defaults to 0.
- **b_1** – The zero value for b. Defaults to 1.

class pypair.contingency.**ConfusionMixin**

Bases: object

Confusion matrix computations.

property acc

Accuracy.

$ACC = \frac{TP+TN}{TP+TN+FP+FN}$

Returns Accuracy.

property ba

Balanced accuracy.

$BA = \frac{TPR+TNR}{2}$

Returns Balanced accuracy.

property bm

Bookmaker informedness.

$BI = TPR + TNR - 1$

Returns BM.

property dor

Diagnostic odds ratio.

$\frac{PLR}{NLR}$

Returns DOR.

property f1

F1 score: harmonic mean of precision and sensitivity.

$F1 = \frac{PPV \times TPR}{PPV + TPR}$

Returns F1.

property fdr

False discovery rate.

$FDR = \frac{FP}{FP+TP}$

Returns FDR.

property fn

FN

Returns FN.

property fnr

False negative rate.

$FNR = \frac{FN}{FN+TP}$

Aliases

- miss rate

Returns FNR.

property fomr

False omission rate.

$FOR = \frac{FN}{FN+TN}$

Returns FOR.

property fp

FP

Returns FP.

property fpr

False positive rate.

$FPR = \frac{FP}{FP+TN}$

Aliases

- fall-out
- probability of false alarm

Returns FPR.

property mcc

Matthew's correlation coefficient.

$MCC = \frac{TP+TN-FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$

Returns

MCC.

property mk

Markedness.

$MK = PPV + NPV - 1$

Aliases

- deltaP

Returns Markedness.

property n

$N = TP + FN + FP + TN$

Returns

N.

property nlr

Negative likelihood ratio.

$NLR = \frac{FNR}{TNR}$

Aliases

- LR-

Returns NLR.

property npv

Negative predictive value.

$NPV = \frac{TN}{TN+FN}$

Returns NPV.

property plr

Positive likelihood ratio.

$PLR = \frac{TPR}{FPR}$

Aliases

- LR+

Returns PLR.

property ppv

Positive predictive value.

$PPV = \frac{TP}{TP+FP}$

Aliases

- precision

Returns PPV.

property precision

Alias to PPV.

Returns PPV.

property prevalence

Prevalence.

$\frac{TP+FN}{N}$

Returns Prevalence.

property pt
Prevalence threshold.

$PT = \frac{\sqrt{TPR(-TNR+1)}+TNR-1}{TPR+TNR-1}$

Returns Prevalence threshold.

property recall
Alias to TPR.

Returns TPR.

property sensitivity
Alias to TPR.

Returns Sensitivity.

property specificity
Alias to TNR.

Returns Specificity.

property tn
TN

Returns TN.

property tnr
True negative rate.

$TNR = \frac{TN}{TN+FP}$

Aliases

- specificity
- selectivity

Returns TNR.

property tp
TP

Returns TP.

property tpr
True positive rate.

$TPR = \frac{TP}{TP+FN}$

Aliases

- sensitivity
- recall
- hit rate
- power
- probability of detection

Returns TPR.

property ts
Threat score.

$TS = \frac{TP}{TP+FN+FP}$

Aliases

- critical success index (CSI).

Returns TS.

class pypair.contingency.**ConfusionStats**(*table*)
Bases: *pypair.contingency.ConfusionMixin*, *pypair.contingency.ContingencyTable*

Computes confusion matrix stats.

__init__(*table*)
ctor.

Parameters **table** – Contingency table.

class pypair.contingency.**ContingencyTable**(*table*)
Bases: *pypair.util.MeasureMixin*, abc.ABC

Abstract contingency table. All other tables inherit from this one.

__init__(*table*)
ctor.

Parameters **table** – A table of counts (list of lists).

6.2 Biserial

These are the biserial association measures.

class pypair.biserial.**Biserial**(*b*, *c*, *b_0=0*, *b_1=1*)
Bases: *pypair.util.MeasureMixin*, *pypair.biserial.BiserialMixin*, object

Biserial association between a binary and continuous variable.

__init__(*b*, *c*, *b_0=0*, *b_1=1*)
ctor.

Parameters

- **b** – Binary variable (iterable).
- **c** – Continuous variable (iterable).
- **b_0** – Value for b is zero. Default 0.
- **b_1** – Value for b is one. Default 1.

class pypair.biserial.**BiserialMixin**
Bases: object

Biserial computations based off of $n, p, q, y_0, y_1, \sigma$.

property biserial
Computes the biserial correlation between a binary and continuous variable. The biserial correlation r_b can be computed from the point-biserial correlation r_{pb} as follows.

$r_b = \frac{r_{\text{pb}}}{h}\sqrt{pq}$

The tricky thing to explain is the h parameter. h is defined as the height of the standard normal distribution at z, where $P(z' < z) = q$ and $P(z' > z) = p$. The way to get h in practice is take the inverse standard normal of q, and then take the standard normal probability of that result. Using Scipy *norm.pdf(norm.ppf(q))*.

References

- Point-Biserial Correlation & Biserial Correlation: Definition, Examples
- Point-Biserial and Biserial Correlations
- Real Statistics Using Excel
- NORM.S.DIST function
- NORM.S.INV function
- scipy.stats.norm
- How to calculate the inverse of the normal cumulative distribution function in python?

Returns Biserial correlation coefficient.

property point_biserial

Computes the point-biserial correlation coefficient between a binary variable X and a continuous variable Y.

$r_{\text{pb}} = \frac{(Y_1 - Y_0)\sqrt{pq}}{\sigma_Y}$

Where

- Y_0 is the average of Y when $X = 0$
- Y_1 is the average of Y when $X = 1$
- σ_Y is the standard deviation of Y
- p is $P(X = 1)$
- q is $1 - p$

Returns Point-biserial correlation coefficient.

property rank_biserial

Computes the rank-biserial correlation between a binary variable X and a continuous variable Y.

$r_r = \frac{2(Y_1 - Y_0)}{n}$

Where

- Y_0 is the average of Y when $X = 0$
- Y_1 is the average of Y when $X = 1$
- n is the total number of data

Returns Rank-biserial correlation.

class pypair.biserial.**BiserialStats**(*n*, *p*, *y_0*, *y_1*, *std*)

Bases: *pypair.util.MeasureMixin*, *pypair.biserial.BiserialMixin*, object

Computes biserial stats.

__init__(*n*, *p*, *y_0*, *y_1*, *std*)

ctor.

Parameters

- **n** – Total number of samples.
- **p** – $P(Y|X=0)$.
- **y_0** – Average of Y when $X=0$. $\bar{Y}_0$
- **y_1** – Average of Y when $X=1$. $\bar{Y}_1$
- **std** – Standard deviation of Y, σ.

6.3 Continuous

These are the continuous association measures.

class pypair.continuous.**Concordance**(*x*, *y*)

Bases: *pypair.util.MeasureMixin*, *pypair.continuous.ConcordanceMixin*, object

Concordance for continuous and ordinal data.

__init__(*x*, *y*)

ctor.

Parameters

- **x** – Continuous or ordinal data (iterable).
- **y** – Continuous or ordinal data (iterable).

class pypair.continuous.**ConcordanceMixin**

Bases: object

property goodman_kruskal_gamma

Goodman-Kruskal γ is like Somer's D. It is defined as follows.

$\gamma = \frac{\pi_c - \pi_d}{1 - \pi_t}$

Where

- $\pi_c = \frac{C}{n}$
- $\pi_d = \frac{D}{n}$
- $\pi_t = \frac{T}{n}$
- C is the number of concordant pairs
- D is the number of discordant pairs
- T is the number of ties
- n is the sample size

Returns γ.

property kendall_tau

Kendall's τ is defined as follows.

$\tau = \frac{C - D}{\binom{n}{2}}$

Where

- C is the number of concordant pairs

- D is the number of discordant pairs
- n is the sample size

Returns τ.

property somers_d

Computes Somers' d for two continuous variables. Note that Somers' d is defined for $d_{X \cdot Y}$ and $d_{Y \cdot X}$ and in general $d_{X \cdot Y} \neq d_{Y \cdot X}$.

- $d_{Y \cdot X} = \frac{\pi_c - \pi_d}{\pi_c + \pi_d + \pi_t^Y}$
- $d_{X \cdot Y} = \frac{\pi_c - \pi_d}{\pi_c + \pi_d + \pi_t^X}$

Where

- $\pi_c = \frac{C}{n}$
- $\pi_d = \frac{D}{n}$
- $\pi_t^X = \frac{T^X}{n}$
- $\pi_t^Y = \frac{T^Y}{n}$
- C is the number of concordant pairs
- D is the number of discordant pairs
- T^X is the number of ties on X
- T^Y is the number of ties on Y
- n is the sample size

Returns $d_{X \cdot Y}$, $d_{Y \cdot X}$.

class pypair.continuous.**ConcordanceStats**(*d*, *t_xy*, *t_x*, *t_y*, *c*, *n*)

Bases: *pypair.util.MeasureMixin*, *pypair.continuous.ConcordanceMixin*

Computes concordance stats.

__init__(*d*, *t_xy*, *t_x*, *t_y*, *c*, *n*)

ctor.

Parameters

- **d** – Number of discordant pairs.
- **t_xy** – Number of ties on XY pairs.
- **t_x** – Number of ties on X pairs.
- **t_y** – Number of ties on Y pairs.
- **c** – Number of concordant pairs.
- **n** – Total number of pairs.

class pypair.continuous.**ConcordantCounts**(*d*, *t_xy*, *t_x*, *t_y*, *c*)

Bases: object

Stores the concordance, discordant and tie counts.

__init__(*d*, *t_xy*, *t_x*, *t_y*, *c*)

ctor.

Parameters

- **d** – Discordant.
- **t_xy** – Tie.
- **t_x** – Tie on X.
- **t_y** – Tie on Y.
- **c** – Concordant.

class pypair.continuous.**Continuous**(*a*, *b*)

Bases: *pypair.util.MeasureMixin*, object

__init__(*a*, *b*)

ctor.

Parameters

- **a** – Continuous variable (iterable).
- **b** – Continuous variable (iterable).

property kendall

Kendall's tau.

Returns Kendall's tau, p-value.

property pearson

Pearson's r.

Returns Pearson's r, p-value.

property regression

Line regression.

Returns Coefficient, p-value

property spearman

Spearman's r.

Returns Spearman's r, p-value.

class pypair.continuous.**CorrelationRatio**(*x*, *y*)

Bases: *pypair.util.MeasureMixin*, object

Correlation ratio.

__init__(*x*, *y*)

ctor.

Parameters

- **x** – Categorical variable (iterable).
- **y** – Continuous variable (iterable).

property anova

Computes an ANOVA test.

Returns F-statistic, p-value.

property calinski_harabasz

Calinski-Harabasz Index.

Returns Calinski-Harabasz Index.

property davies_bouldin
Davies-Bouldin Index.

Returns Davies-Bouldin Index.

property eta
Gets η.

Returns η.

property eta_squared
Gets $\eta^2 = \frac{\sigma_{\bar{y}}^2}{\sigma_y^2}$

Returns η^2.

property kruskal
Computes the Kruskal-Wallis H-test.

Returns H-statistic, p-value.

property silhouette
Silhouette coefficient.

Returns Silhouette coefficient.

6.4 Associations

Some of the functions here are just wrappers around the contingency tables and may be looked at as convenience methods to simply pass in data for two variables. If you need more than the specific association, you are encouraged to build the appropriate contingency table and then call upon the measures you need.

pypair.association.**agreement**(*a*, *b*, *measure='chohen_k'*, *a_vals=None*, *b_vals=None*)
Gets the agreement association.

Parameters

- **a** – Categorical variable (iterable).
- **b** – Categorical variable (iterable).
- **measure** – Measure. Default is *chohen_k*.
- **a_vals** – The unique values in *a*.
- **b_vals** – The unique values in *b*.

Returns Measure.

pypair.association.**binary_binary**(*a*, *b*, *measure='chisq'*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)
Gets the binary-binary association.

Parameters

- **a** – Binary variable (iterable).
- **b** – Binary variable (iterable).
- **measure** – Measure. Default is *chisq*.
- **a_0** – The a zero value. Default 0.
- **a_1** – The a one value. Default 1.
- **b_0** – The b zero value. Default 0.

- **b_1** – The b one value. Default 1.

Returns Measure.

pypair.association.**binary_continuous**(*b*, *c*, *measure='biserial'*, *b_0=0*, *b_1=1*)
Gets the binary-continuous association.

Parameters

- **b** – Binary variable (iterable).
- **c** – Continuous variable (iterable).
- **measure** – Measure. Default is *biserial*.
- **b_0** – Value when *b* is zero. Default 0.
- **b_1** – Value when *b* is one. Default is 1.

Returns Measure.

pypair.association.**categorical_categorical**(*a*, *b*, *measure='chisq'*, *a_vals=None*, *b_vals=None*)
Gets the categorical-categorical association.

Parameters

- **a** – Categorical variable (iterable).
- **b** – Categorical variable (iterable).
- **measure** – Measure. Default is *chisq*.
- **a_vals** – The unique values in *a*.
- **b_vals** – The unique values in *b*.

Returns Measure.

pypair.association.**categorical_continuous**(*x*, *y*, *measure='eta'*)
Gets the categorical-continuous association.

Parameters

- **x** – Categorical variable (iterable).
- **y** – Continuous variable (iterable).
- **measure** – Measure. Default is *eta*.

Returns Measure.

pypair.association.**concordance**(*x*, *y*, *measure='kendall_tau'*)
Gets the specified concordance between the two variables.

Parameters

- **x** – Continuous or ordinal variable (iterable).
- **y** – Continuous or ordinal variable (iterable).
- **measure** – Measure. Default is *kendall_tau*.

Returns Measure.

pypair.association.**confusion**(*a*, *b*, *measure='acc'*, *a_0=0*, *a_1=1*, *b_0=0*, *b_1=1*)
Gets the specified confusion matrix stats.

Parameters

- **a** – Binary variable (iterable).
- **b** – Binary variable (iterable).
- **measure** – Measure. Default is *acc*.
- **a_0** – The a zero value. Default 0.
- **a_1** – The a one value. Default 1.
- **b_0** – The b zero value. Default 0.
- **b_1** – The b one value. Default 1.

Returns Measure.

pypair.association.**continuous_continuous**(*x*, *y*, *measure='pearson'*)
Gets the continuous-continuous association.

Parameters

- **x** – Continuous variable (iterable).
- **y** – Continuous variable (iterable).
- **measure** – Measure. Default is 'pearson'.

Returns Measure.

6.5 Decorators

These are decorators.

pypair.decorator.**distance**(*f*)
Marker for distance functions.

pypair.decorator.**similarity**(*f*)
Marker for similarity functions.

pypair.decorator.**timeit**(*f*)
Benchmarks the time it takes (seconds) to execute.

6.6 Utility

These are utility functions.

class pypair.util.**MeasureMixin**
Bases: abc.ABC

Measure mixin. Able to get list the functions decorated with *@property* and also access such property based on name.

get(*measure*)
Gets the specified measure.

Parameters **measure** – Name of measure.

Returns Measure.

get_measures()
Gets a list of all the measures.

Returns List of all the measures.

classmethod measures()
Gets a list of all the measures.

Returns List of all the measures.

pypair.util.**get_measures**(*clazz*)
Gets all the measures of a clazz.

Parameters **clazz** – Clazz.

Returns List of measures.

6.7 Spark

These are functions that you can use in a Spark. You must pass in a Spark dataframe and you will get a `pair-RDD` as output. The pair-RDD will have the following as its keys and values.

- key: in the form of a tuple of strings `(k1, k2)` where k1 and k2 are names of variables (column names)
- value: a dictionary `{'acc': 0.8, 'tpr': 0.9, 'fpr': 0.8, ...}` where keys are association measure names and values are the corresponding association values

pypair.spark.**agreement**(*sdf*)
Gets all pairwise categorical-categorical *agreement* association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and metrics e.g. {'kappa': 0.9, 'delta': 0.2}. Each record in the pair-RDD is of the form.

- (k1, k2), {'kappa': 0.9, 'delta': 0.2, ... }

Parameters **sdf** – Spark dataframe. Should be strings or whole numbers to represent the values.

Returns Spark pair-RDD.

pypair.spark.**binary_binary**(*sdf*)
Gets all the pairwise binary-binary association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and measures e.g. {'phi': 1, 'lambda': 0.8}. Each record in the pair-RDD is of the form.

- (k1, k2), {'phi': 1, 'lambda': 0.8, ... }

Parameters **sdf** – Spark dataframe. Should be all 1's and 0's.

Returns Spark pair-RDD.

pypair.spark.**binary_continuous**(*sdf*, *binary*, *continuous*, *b_0=0*, *b_1=1*)
Gets all pairwise binary-continuous association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and metrics e.g. {'biserial': 0.9, 'point_biserial': 0.2}. Each record in the pair-RDD is of the form.

- (k1, k2), {'biserial': 0.9, 'point_biserial': 0.2, ... }

All the binary fields/columns should be encoded in the same way. For example, if you are using 1 and 0, then all binary fields should only have those values, not a mixture of 1 and 0, True and False, -1 and 1, etc.

Parameters

- **sdf** – Spark dataframe.
- **binary** – List of fields that are binary.

- **continuous** – List of fields that are continuous.
- **b_0** – Zero value for binary field.
- **b_1** – One value for binary field.

Returns Spark pair-RDD.

pypair.spark.**categorical_categorical**(*sdf*)

Gets all pairwise categorical-categorical association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and metrics e.g. {'phi': 0.9, 'chisq': 0.2}. Each record in the pair-RDD is of the form.

- (k1, k2), {'phi': 0.9, 'chisq': 0.2, ... }

Parameters **sdf** – Spark dataframe. Should be strings or whole numbers to represent the values.

Returns Spark pair-RDD.

pypair.spark.**categorical_continuous**(*sdf*, *categorical*, *continuous*)

Gets all pairwise categorical-continuous association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and metrics e.g. {'eta_sq': 0.9, 'eta': 0.95}. Each record in the pair-RDD is of the form.

- (k1, k2), {'eta_sq': 0.9, 'eta': 0.95}

For now, only `eta` η^2 is supported.

Parameters

- **sdf** – Spark dataframe.
- **categorical** – List of categorical variables.
- **continuous** – List of continuous variables.

Returns Spark pair-RDD.

pypair.spark.**concordance**(*sdf*)

Gets all the pairwise ordinal-ordinal concordance measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and measures e.g. {'kendall': 1, 'gamma': 0.8}. Each record in the pair-RDD is of the form.

- (k1, k2), {'kendall': 1, 'gamma': 0.8, ... }

Parameters **sdf** – Spark dataframe. Should be all ordinal data (numeric).

Returns Spark pair-RDD.

pypair.spark.**confusion**(*sdf*)

Gets all the pairwise confusion matrix metrics. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and metrics e.g. {'acc': 0.9, 'fpr': 0.2}. Each record in the pair-RDD is of the form.

- (k1, k2), {'acc': 0.9, 'fpr': 0.2, ... }

Parameters **sdf** – Spark dataframe. Should be all 1's and 0's.

Returns Spark pair-RDD.

pypair.spark.**continuous_continuous**(*sdf*)

Gets all the pairwise continuous-continuous association measures. The result is a Spark pair-RDD, where the keys are tuples of variable names e.g. (k1, k2), and values are dictionaries of association names and measures e.g. {'pearson': 1}. Each record in the pair-RDD is of the form.

- (k1, k2), {'pearson': 1}

Only pearson is supported at the moment.

Parameters **sdf** – Spark dataframe. Should be all ordinal data (numeric).

Returns Spark pair-RDD.

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

EIGHT

ABOUT

One-Off Coder is an educational, service and product company. Please visit us online to discover how we may help you achieve life-long success in your personal coding career or with your company's business goals and objectives.

-
-
-
-
-
-

CHAPTER

NINE

COPYRIGHT

9.1 Documentation

9.2 Software

```
Copyright 2020 One-Off Coder

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

   http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

9.3 Art

```
Copyright 2020 Daytchia Vang
```

CHAPTER

TEN

CITATION

```
@misc{oneoffcoder_pypair_2020,
title={PyPair, A Statistical API for Bivariate Association Measures},
url={https://github.com/oneoffcoder/py-pair},
author={Jee Vang},
year={2020},
month={Nov}}
```

CHAPTER

ELEVEN

AUTHOR

Jee Vang, Ph.D.

-

CHAPTER

TWELVE

HELP

-
-

BIBLIOGRAPHY

[Cal] Keith G. Calkins. More correlation coefficients. URL: https://www.andrews.edu/~calkins/math/edrm611/edrm13.htm.

[Cox70] D. R. Cox. *Analysis of binary data*. Chapman and Hall, 1970.

[Exc] Stack Exchange. Measures for binary data. URL: https://stats.stackexchange.com/questions/61705/similarity-coefficients-for-binary-data-why-choose-jaccard-over-russell-and-rao.

[fDRE] Institute for Digital Research and Education. What is the difference between categorical, ordinal and numerical variables? URL: https://stats.idre.ucla.edu/other/mult-pkg/whatstat/what-is-the-difference-between-categorical-ordinal-and-numerical-variables/.

[Gle] Stephanie Glen. What is somers' delta? URL: https://www.statisticshowto.com/somers-d.

[Gra] GraphPad. What is the difference between ordinal, interval and ratio variables? why should i care? URL: https://www.graphpad.com/support/faq/what-is-the-difference-between-ordinal-interval-and-ratio-variables-why-should-i-care/.

[Lie83] Albert M. Liebetrau. *Measures of association*. Sage Publications, Inc., 1983.

[Min] Minitab. What are categorical, discrete, and continuous variables? URL: https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/supporting-topics/basics/what-are-categorical-discrete-and-continuous-variables/.

[NIP] NIPS. Nips 2008 workshop on causality. URL: http://clopinet.com/isabelle/Projects/NIPS2008/.

[oM] University of Minnesota. Types of variables. URL: https://cyfar.org/types-variables.

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[Pea09] Judea Pearl. *Causality: Models, Reasoning and Inference*. Chapman and Hall, 2009.

[Pea16] Judea Pearl. *Causal Inference in Statistics - A Primer*. Wiley, 2016.

[Pea20] Judea Pearl. *The Book of Why: The New Science of Cause and Effect*. Basic Books, 2020.

[Pro] IBM Proximities. Measures for binary data. URL: https://www.ibm.com/support/knowledgecenter/SSLVMB_24.0.0/spss/base/syn_proximities_measures_binary_data.html.

[Rey84] H. T. Reynolds. *Analysis of nominal data*. Sage Publications, Inc., 1984.

[SSC10] Charles C. Tappert Seung-Seok Choi, Sung-Hyuk Cha. A survey of binary similarity and distance measures. *Systemics, Cybernetics and Informatics*, 2010.

[Sta] Laerd Statistics. Types of variable. URL: https://statistics.laerd.com/statistical-guides/types-of-variable.php.

[Unia] Penn State University. Measures of association for binary variables. URL: https://online.stat.psu.edu/stat505/lesson/14/14.3.

[Unib] Penn State University. Measures of association for continuous variables. URL: https://online.stat.psu.edu/stat505/lesson/14/14.2.

[War19] Matthijs J. Warrens. Similarity measures for 2 x 2 tables. *Journal of Intelligent & Fuzzy Systems*, 2019.

[Wika] Wikipedia. Fowlkes-mallows index. URL: https://en.wikipedia.org/wiki/Fowlkes%E2%80%93Mallows_index.

[Wikb] Wikipedia. Jaccard index. URL: https://en.wikipedia.org/wiki/Jaccard_index.

[Wikc] Wikipedia. Overlap coefficient. URL: https://en.wikipedia.org/wiki/Overlap_coefficient.

[Wikd] Wikipedia. Prospect theory. URL: https://en.wikipedia.org/wiki/Prospect_theory.

[Wike] Wikipedia. Somer's d. URL: https://en.wikipedia.org/wiki/Somers%27_D.

[Wikf] Wikipedia. Sørensen–dice coefficient. URL: https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient.

[Wikg] Wikipedia. Tversky index. URL: https://en.wikipedia.org/wiki/Tversky_index.

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

R

S

T

U

V

Y