

```
In [1]: import math
import pandas as pd
import numpy as np
from sklearn.datasets import fetch_rcv1
import time
from scipy import sparse
```

```
In [2]: def freq(x, prob: bool = True) -> list:
    if type(x) == sparse.csr_matrix or type(x) == sparse.csc_matrix:
        nonzero = x.nonzero()[0]
        uniques = set(nonzero)

        count_nonzero = len(nonzero)
        counts = {
            0: x.shape[0] - count_nonzero,
            1: count_nonzero
        }
        total = sum(counts.values())

    return [uniques, counts if prob is False else {key: val / total for key, val in counts.items()}]

    counts = {}
    uniques = []

    for val in x:
        if val not in uniques:
            uniques.append(val)

        if val in counts.keys():
            counts[val] += 1
            continue

        counts[val] = 1

    total = sum(counts.values())
    return [uniques, counts if prob is False else {key: val / total for key, val in counts.items()}]
```

Poniższa funkcja zwraca dla zadanych kolumn danych X i Y unikalne wartości atrybutów X oraz Y, a także w zależności od parametru **prob** zwraca łączny rozkład częstości lub prawdopodobieństwa łączne liczności. Funkcja obsługuje także macierze rzadkie, w których dane zostały binaryzowane - przedstawiają jedynie sam fakt wystąpienia. Możliwe przyspieszenie obliczeń dla tej funkcji po zredukowaniu rozwiązania do pojedynczej pętli zamiast podwójnej dla kolumn danych X i Y.

```
In [3]: def freq2(x, y, prob: bool = True) -> list:
    if (type(x) == sparse.csr_matrix or type(x) == sparse.csc_matrix) and (type(y) == sparse.csr_matrix or type(y) == sparse.csc_matrix):
        x_nonzero = x.nonzero()[0]
        y_nonzero = y.nonzero()[0]

        uniques_x = set(x_nonzero)
        uniques_y = set(y_nonzero)
        intersection_x_y = uniques_x.intersection(uniques_y)

        count_intersection = len(intersection_x_y)
        count_x_nonzero = len(x_nonzero)
        count_y_nonzero = len(y_nonzero)
        count_shared_zeros = x.shape[0] - count_x_nonzero + y.shape[0] - count_y_nonzero

        counts = {

    counts = {
```

```

        (0, 0): count_shared_zeros,
        (0, 1): count_y_nonzero - count_intersection,
        (1, 0): count_x_nonzero - count_intersection,
        (1, 1): count_intersection
    }
    total = sum(counts.values())

    return [uniques_x, uniques_y, counts if prob is False else {key: val / total: value}]

counts = {}
uniques = {'x': [], 'y': []}

for x_val in x:
    if x_val not in uniques['x']:
        uniques['x'].append(x_val)

    for y_val in y:
        key = (x_val, y_val)

        if key not in counts.keys():
            counts[key] = 1

            if y_val not in uniques['y']:
                uniques['y'].append(y_val)
            else:
                counts[key] += 1

        total = sum(counts.values())
    return [uniques['x'], uniques['y'], counts if prob is False else {key: val / total: value}]

```

Dla entropii łącznej został obsłużony przypadek, gdy prawdopodobieństwo łączne liczności byłoby równe 0 - np. brak wspólnych elementów w kolumnach danych X i Y będące zbinaryzowanymi macierzami rzadkimi (klucz (1, 1)).

```
In [4]: def entropy(x, y=None, conditional_reverse: bool = False):
    if y is None:
        uniques, probs = freq(x)
    else:
        uniques_x, uniques_y, probs = freq2(x, y)

        if conditional_reverse is True and y is not None:
            uniques_x, probs_x = freq(y)
            entropy_y = entropy(y)

            return sum(prob * entropy_y for prob in probs_x.values())

    return -sum(prob * math.log2(prob) if prob != 0 else 0 for prob in probs.values)
```

Poniżej wersja dla entropii infogain:

```
In [5]: def infogain(x, y, reverse: bool = False):
    if reverse is False:
        return entropy(x) + entropy(y) - entropy(x, y)
    return entropy(y) - entropy(x, y, conditional_reverse=True)
```

```
In [6]: def kappa(x, y):
    return infogain(x, y) / entropy(y)
```

W poniżej funkcji odpowiadającej wyliczaniu indeksu Giniego został obsłużony przypadek dla indeksu Giniego warunkowego ($Y|X$), bazując na funkcji freq2.

```
In [7]: def gini(x, y=None, conditional_reverse: bool = False):
    if y is None:
        uniques, probs = freq(x)
    else:
        uniques_x, uniques_y, probs = freq2(x, y)

    if conditional_reverse is True and y is not None:
        uniques, probs = freq(x)
        gini_y = gini(y)
        return sum(prob * gini_y for prob in probs.values())

    return 1 - sum(prob ** 2 for prob in probs.values())
```

```
In [8]: def ginigain(x, y):
    return gini(y) - gini(x, y, True)
```

Pomimo, że entropia i indeks Giniego mierzą podobne miary dotyczące informacji, to wyniki entropii są większe od wyników indeksu Giniego. Wskaźnik Giniego jest wykorzystywany przez algorytm CART (drzewo klasyfikacji i regresji), natomiast przyrost informacji poprzez redukcję entropii jest wykorzystywany przez algorytmy takie jak C4.5 (algorytmy do generowania drzewa decyzyjnego).

Poniższa funkcja jest odpowiedzialna za dokonanie eksperymentu na wczytanych danych ze zbioru danych **Reuters Corpus Volume I**. Dostarczane są tutaj kolumny danych jako sparse.csc_matrix (preferowane do operacji kolumnowych) lub sparse.csr_matrix (preferowane do operacji wierszowych). Czas jest mierzony dla obliczenia ilości informacji na temat wybranej zmiennej decyzyjnej `y` oraz dla sortowania wyników i wybierania 50 najlepszych rezultatów. Czasy są wypisywane na ekranie i zwracane są posortowane rezultaty.

Zakres słów ustwiony został na 500 próbek od 1000 próbki do 1500. Czas na wykonanie obliczeń był znaczny już przy takiej ilości, więc nie ich liczba nie została zwiększana.

```
In [9]: def experiment_best_50_words(x, y):
    info_gain_words = []

    time_info_gain1 = time.time()
    for i, word in enumerate(word_list['A'][1000:1500]):
        gain = infogain(x[:, i], y)
        info_gain_words.append({'infogain': gain, 'word': word, 'id': i})
    time_info_gain2 = time.time()

    time_sort_and_select_the_best1 = time.time()
    info_gain_words = sorted(info_gain_words, key=lambda item: item['infogain'], reverse=True)
    best50 = info_gain_words[:50]
    time_sort_and_select_the_best2 = time.time()

    print(f'time of calculating info gain: {time_info_gain2 - time_info_gain1}s')
    print(f'time of sort results and select 50 of the best: {time_sort_and_select_the_best2 - time_sort_and_select_the_best1}s')

    return best50
```

Poniżej wczytanie zbioru autos oraz wypisanie posortowanych atrybutów od najwyższych wartości przyrostu informacji. Tutaj została zastosowana miara entropii:

```
In [10]: autos = pd.read_csv('D:\Programming\Python\computational-intelligence\machine-learnin
```

```
info_gains = {key: entropy(autos[key]) for key in autos.columns}
print(sorted(info_gains.items(), key=lambda x: x[1], reverse=True))

[('animal', 6.638409502553759), ('type', 2.390559682294039), ('legs', 2.0338113440641234), ('predator', 0.9914266810680207), ('catsize', 0.9880162151534646), ('hair', 0.9840304711717017), ('eggs', 0.9794662187017298), ('milk', 0.9743197211096903), ('toothed', 0.9685867165455516), ('aquatic', 0.9396846718728563), ('tail', 0.8228368841492257), ('airborne', 0.7910662980902585), ('breathes', 0.7374895672137456), ('feathers', 0.7179499765002912), ('backbone', 0.6761627418829198), ('fins', 0.653839880626333), ('domestic', 0.5538976334852962), ('venomous', 0.3993820824245975)]
```

Wczytanie danych i ich binaryzacja. Tutaj został wybrany atrybut decyzyjny pod indeksem 98:

```
In [11]: rcv1 = fetch_rcv1()
X = rcv1['data'] > 0
Xr = X[:, 2]
Y = rcv1['target'][:, 98]
```

Wczytanie listy słów niezbędnych do wyznaczenia najlepszych słów w eksperymencie:

```
In [12]: word_list = pd.read_csv('D:\Programming\Python\computational-intelligence\machine-
```

Wykonanie eksperimentu. Domyślnie dane z bazy danych **Reuters Corpus Volume I** wczytane za pomocą `fetch_rcv1()`, są wczytywane jako format wierszowy (sparse.csr_matrix).

```
In [13]: best50 = experiment_best_50_words(X, Y)
```

```
time of calculating info gain: 38.8919997215271s
time of sort results and select 50 of the best: 0.0s
```

Zmiana formatu dla wczytanych danych z formatu wierszowego na kolumnowy i ponowne wykonanie eksperimentu:

```
In [14]: X = X.tocsc()
Y = Y.tocsc()
best50 = experiment_best_50_words(X, Y)
```

```
time of calculating info gain: 5.455239534378052s
time of sort results and select 50 of the best: 0.0s
```

Przygotowanie danych do wyświetlenia i wyświetlenie 50 zmiennych (słów) dostarczających najwięcej informacji na temat wybranej zmiennej decyzyjnej - w tym przypadku zmienna decyzyjna pod indeksem 98.

```
In [15]: data_to_display = [f'word: `{best["word"]}`', infogain: {best["infogain"]}], k={kappa}
print(np.array(data_to_display))
```

```
[ 'word: `amad`, infogain: 0.43772683504110554, k=0.8962659789528651'  
'word: `alten`, infogain: 0.35141054733702126, k=0.7195293799928918'  
'word: `alli`, infogain: 0.32120664476413185, k=0.6576854898298737'  
'word: `alpic`, infogain: 0.3022579064953083, k=0.6188870701423215'  
'word: `altavist`, infogain: 0.29260101405584266, k=0.5991141353733407'  
'word: `allan`, infogain: 0.27993397354517624, k=0.5731777829386951'  
'word: `altan`, infogain: 0.2762790928649723, k=0.5656942453792689'  
'word: `allout`, infogain: 0.2642281581190378, k=0.5410193980481753'  
'word: `altogether`, infogain: 0.2601776094614102, k=0.53272571196977'  
'word: `almog`, infogain: 0.25925269133139656, k=0.5308318992379779'  
'word: `allay`, infogain: 0.2517584103315219, k=0.5154870116067004'  
'word: `allocat`, infogain: 0.24730467230754505, k=0.5063677766169508'  
'word: `amang`, infogain: 0.24243779008315258, k=0.4964026097318001'  
'word: `allflex`, infogain: 0.2393683602914099, k=0.4901178099962184'  
'word: `alia`, infogain: 0.23154709901117093, k=0.4741034150886701'  
'word: `alexandrou`, infogain: 0.22588902320502385, k=0.4625182426810664'  
'word: `alicia`, infogain: 0.22576325262936286, k=0.46226072159920795'  
'word: `akpoborie`, infogain: 0.22071700636644653, k=0.4519282985334728'  
'word: `allahu`, infogain: 0.2205762251838404, k=0.4516400425383046'  
'word: `alaniy`, infogain: 0.21844255656786615, k=0.44727125717316335'  
'word: `alien`, infogain: 0.21778001891843934, k=0.44591468063405104'  
'word: `alot`, infogain: 0.2173691834214519, k=0.44507347591590424'  
'word: `alleah`, infogain: 0.21617003226935083, k=0.4426181583634645'  
'word: `alistair`, infogain: 0.2158182160118174, k=0.4418977982730439'  
'word: `allerg`, infogain: 0.2157280628340914, k=0.4417132054639631'  
'word: `alfio`, infogain: 0.21517268134859563, k=0.44057603613610263'  
'word: `akl`, infogain: 0.21365526238973254, k=0.43746905050082774'  
'word: `amba`, infogain: 0.21154804084017154, k=0.4331544167297214'  
'word: `alloc`, infogain: 0.21122353567541102, k=0.4324899773673459'  
'word: `alienat`, infogain: 0.2105031333911509, k=0.4310149203069725'  
'word: `alusaf`, infogain: 0.21009973019823186, k=0.43018893357581783'  
'word: `alg`, infogain: 0.20599441156106646, k=0.42178310342627495'  
'word: `allow`, infogain: 0.20534305086517302, k=0.42044941221746085'  
'word: `alrent`, infogain: 0.20522779029117627, k=0.4202134108510421'  
'word: `alai`, infogain: 0.20462928649096657, k=0.41898794658552224'  
'word: `akerholm`, infogain: 0.2042346708918702, k=0.4181799528599355'  
'word: `aluminim`, infogain: 0.20405197969995087, k=0.4178058841785939'  
'word: `alkal`, infogain: 0.2034396604523006, k=0.4165521321442335'  
'word: `alfred`, infogain: 0.2031960904670862, k=0.4160534113125046'  
'word: `alil`, infogain: 0.2031171838083674, k=0.41589184627225334'  
'word: `aldcroft`, infogain: 0.20164450490364005, k=0.4128764679710698'  
'word: `alloush`, infogain: 0.20034714091735756, k=0.41022005508947323'  
'word: `albert`, infogain: 0.2002549984908638, k=0.4100313892013589'  
'word: `aloft`, infogain: 0.20019378498963253, k=0.40990605172096195'  
'word: `allenescu`, infogain: 0.20012130992841276, k=0.4097576556746293'  
'word: `allik`, infogain: 0.19996215879574047, k=0.4094317863554394'  
'word: `alford`, infogain: 0.1996655639671801, k=0.4088244946997952'  
'word: `alvarez`, infogain: 0.19966062490241582, k=0.4088143817358147'  
'word: `ambi`, infogain: 0.19953841360084706, k=0.4085641484325964'  
'word: `algecir`, infogain: 0.19904493593560457, k=0.40755373004531975']
```