

# Analiza porównawcza manuskryptu Wojnicza i tekstu katalońskich

Autorzy: Konrad Reczko, Joanna Wójcicka

## Import bibliotek

```
In [211]:  
import re  
from matplotlib import pyplot as plt  
import numpy as np
```

### Wczytanie danych oraz przygotowanie ich do analizy

```
In [213]:  
class DataGetterManuscript:  
    def __init__(self, path):  
        self.data = None  
        self.paragraphs = []  
        self.words = []  
  
        with open(path, 'r', encoding='utf-8') as file:  
            self.data = file.read()  
        self.data = self.data.split('\n')        pass  
    def get_paragraphs_from_vojnick(self):  
        """  
        Removes comments from data and splits it into the paragraphs  
        """  
        processed_data = []  
        for x in self.data:  
            if x.startswith('#') or len(x) < 2:  
                continue  
            curr_paragraph = []  
            for x_1 in processed_data:  
                if x_1 == '-':  
                    curr_paragraph.append(x)  
                elif x_1 == '#':  
                    curr_paragraph.append(x_1)  
                else:  
                    curr_paragraph = []  
  
        def get_paragraphs(self):  
            """  
            Removes comments from data and splits it into the paragraphs  
            """  
            curr_paragraph = []  
            for x in self.data:  
                if x == '#':  
                    curr_paragraph.append(x)  
                elif x == '\n':  
                    curr_paragraph.append(x)  
                    self.paragraphs.append(curr_paragraph)  
                    curr_paragraph = []  
                else:  
                    curr_paragraph.append(x)  
  
            if curr_paragraph:  
                self.paragraphs.append(curr_paragraph)  
  
    def remove_ambiguous(self):  
        """  
        The vojnick manuscript contains some ambiguous characters.  
        They are being marked as [A|B|C] and so on.  
        This function removes them by choosing only the first option in our  
        example case it would be A  
        """  
        for paragraph in self.paragraphs:  
            for i in range(len(paragraph)):  
                if paragraph[i] == '[':  
                    paragraph[i] = paragraph[i+1]  
                    while paragraph[i+1] != ']':  
                        paragraph.pop(i+1)  
                    paragraph.pop(i+1)  
  
    def get_words(self):  
        for paragraph in self.paragraphs:  
            for line in paragraph:  
                words_in_line = [word for word in re.split(r'\W+', line) if word]  
                self.words.extend(words_in_line)  
  
        return self.words  
  
    def process_formatted_data(self) -> list[list[str]]:  
        """  
        It takes vojnick manuscript data and removes all unnecessary lines and characters.  
        It returns list of paragraphs with words  
        """  
        self.get_paragraphs_from_vojnick()  
        self.remove_ambiguous()  
        self.get_words()  
        return self.words  
  
    def process_unformatted_data(self) -> list[list[str]]:  
        """  
        It takes vojnick manuscript data and removes all unnecessary lines and characters.  
        It returns list of paragraphs with words  
        """  
        self.get_paragraphs()  
        self.get_words()  
        return self.words
```

### Wczytanie danych

Badać będziemy dwa zbiory danych:

- manuskrypt Wojnicza
- teksty katalońskie (będą one połączeniem różnych tekstów, aby zachować podobną liczbę słów) – będzie to Powszechna deklaracja prawa człowieka, konstytucja hiszpańska oraz fragmenty tekstów z Wikipedii (o Ziemi oraz Hiszpanii).

Ważnym jest, aby w obu przypadkach zachować podobną liczbę słów, aby móc mniej więcej porównać wyniki.

```
In [214]:  
vojnick = DataGetterManuscript("text_samples/vojnick.txt")  
vojnick.process_formatted_data()  
  
catalan = DataGetterManuscript("text_samples/catalan.txt")  
catalan.process_unformatted_data()  
  
print("Vojnick manuscript words: ", len(vojnick.words))  
print("Catalan words: ", len(catalan.words))  
  
Vojnick manuscript words: 35655  
Catalan words: 3542
```

## Prawo Zipfa

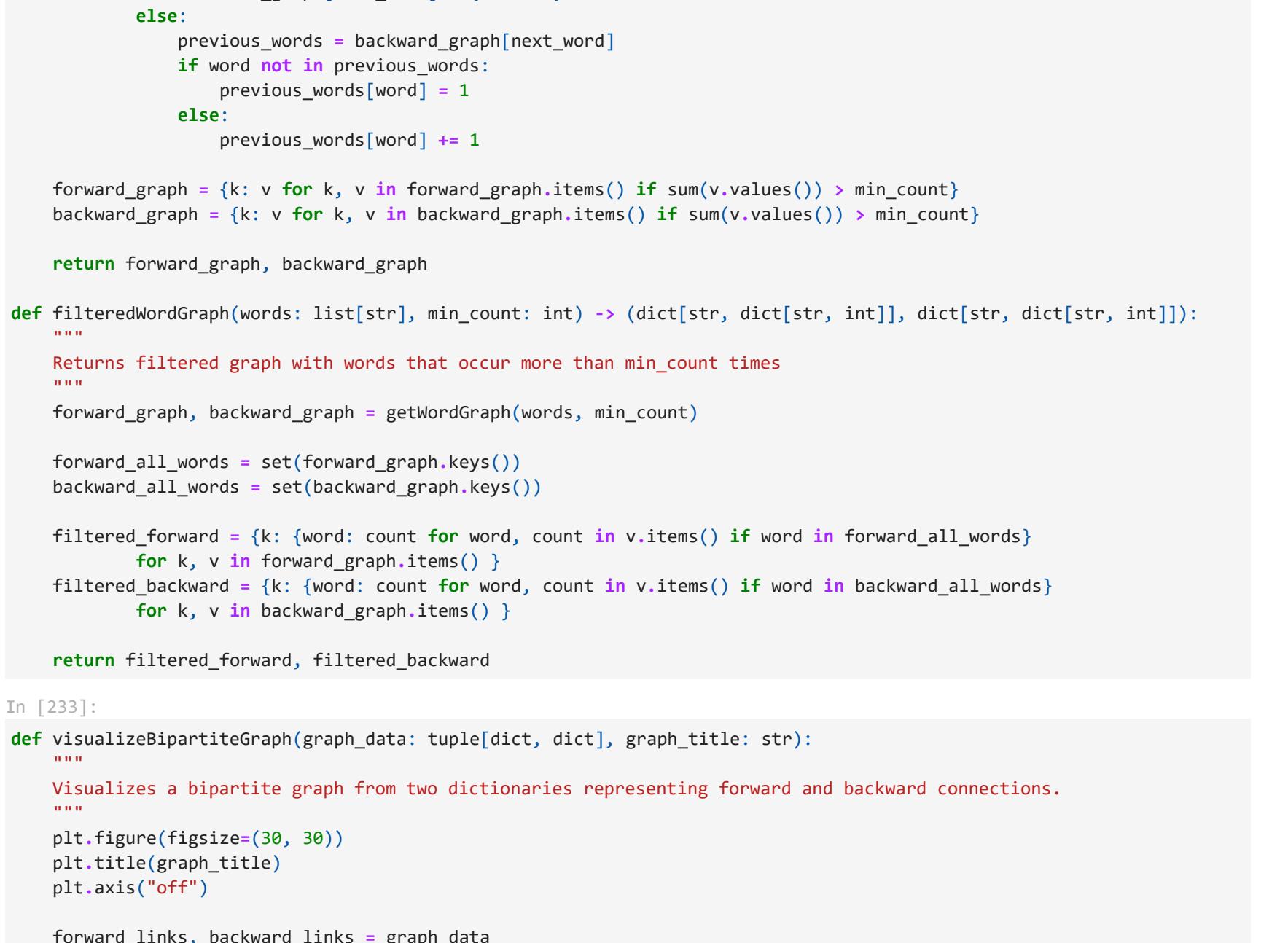
Prawo Zipfa jest jednym z empirycznych praw, które opisuje zjawisko występujące w różnych zbiorach danych, zwłaszcza w językoznawstwie. Stwierdza ono, że w każdym naturalnym języku najczęściej używanego słowa pojawia się dwa razy częściej niż drugie najczęściej używane słwo, trzy razy częściej niż trzecie najczęściej używanego słowa, i tak dalej. Inny sposób, częstotliwość występowania każdego słowa jest odwrotnie proporcjonalna do jego rangi w rankingu częstotliwości

```
In [215]:  
def NGramsCount(words: list[str], n: int) -> dict[str, int]:  
    """  
    Returns dictionary of n-grams and their counts  
    words - list of words  
    c - n-gram count  
    """  
    ngrams = {}  
    for i in range(len(words) - n + 1):  
        ngram = ''.join(words[i:i+n])  
        if ngram in ngrams:  
            ngrams[ngram] += 1  
        else:  
            ngrams[ngram] = 1  
  
    return ngrams  
def getNGramFrequencySorted(ngrams: dict[str, int, int]) -> list[tuple[str, int]]:  
    """  
    Returns list of tuples (ngram, frequency) sorted by frequency  
    """  
    ngram_freq = {}  
    for ngram in ngrams.keys():  
        ngram_freq[ngram] = ngrams[ngram] / len(ngrams)  
  
    sorted_freq = sorted(ngram_freq.items(), key=lambda x: x[1], reverse=True)  
  
    return sorted_freq  
  
def getNGramScores(ngrams: dict[str, int, int]) -> list[tuple[str, int]]:  
    """  
    Returns list of tuples (ngram, score) sorted by score (score = rank * frequency)  
    """  
    sorted_freq = getNGramFrequencySorted(ngrams)  
  
    scores = []  
    for i in range(len(sorted_freq)):  
        scores.append((i + 1 * sorted_freq[i][1], sorted_freq[i][0]))  
    return scores
```

Aby sprawdzić zachowanie prawa Zipfa dla obu zbiorów danych, wyodrębniliśmy 1-gramy i posortowaliśmy je według częstości występowania.

```
In [216]:  
plt.figure(figsize=(10, 10))  
plt.title("Prawo Zipfa dla manuskryptu Wojnicza i tekstu katalońskiego dla N")  
plt.xlabel("Ranga")  
y_vojnick = [x for x in range(0, len(getNGramScores(NGramsCount(vojnick.words, 1)))][100:700]  
y_catalan = [x for x in range(0, len(getNGramScores(NGramsCount(catalan.words, 1)))][100:700]  
plt.plot(x_vojnick, y_vojnick, 'r')  
plt.plot(x_catalan, y_catalan, 'b')  
plt.legend(['Vojnick', 'Catalan'])  
plt.show()
```

Prawo Zipfa dla manuskryptu Wojnicza i tekstu katalońskiego dla N



Następnie wyliczony został score w kontekście prawa Zipfa. Score jest to iloraz rangi oraz częstości występowania danego słowa. W idealnym przypadku zgadza się z prawem Zipfa, wartość tego ilorazu powinna być mniejsza od 1,0000000000000002, co oznacza, że iloraz rangi i częstości jest stosunkowo stały, a zatem teksty te dobrze zgadzają się z prawem Zipfa. Dla manuskryptu Wojnicza wartość po pokrywająca się z wartościami dla tekstów katalońskich.

```
In [217]:  
plt.figure(figsize=(10, 10))  
plt.title("Prawo Zipfa dla manuskryptu Wojnicza i tekstu katalońskiego")  
plt.xlabel("Ranga")  
y_vojnick = [x for x in range(0, len(getNGramScores(NGramsCount(vojnick.words, 1)))][100:700]  
y_catalan = [x for x in range(0, len(getNGramScores(NGramsCount(catalan.words, 1)))][100:700]  
plt.bar(x_vojnick, y_vojnick, 'r')  
plt.bar(x_catalan, y_catalan, 'b')  
plt.legend(['Vojnick', 'Catalan'])  
plt.show()
```

Prawo Zipfa dla manuskryptu Wojnicza i tekstu katalońskiego



Wnioski

Wyniki analizy prawa Zipfa pokazują, że manuskrypt Wojnicza jest bardzo zbliżony do tekstu w języku katalońskim. W obu przypadkach wykazuje się prawidłowe położenie słów w znaczącym stopniu. Można zauważyc, że tekst Wojnicza przyjmuje pewne charakterystyki języka naturalnego, co może sugerować, że jest to język naturalny, a nie sztuczny. Jednakże spełnienie prawa Zipfa nie jest wystarczające, aby móc stwierdzić, że dany tekst jest rzeczywiście językiem naturalnym.

## N-gramy i ich interpretacja

N-gram to termin używany w językoznawstwie i przetwarzaniu języka naturalnego (NLP), który odnosi się do ciągu "n" kolejnych elementów z danego tekstu lub nowej. Elementy te mogą być słownicami, frazami, literami, słowami lub bazowymi parami złożonymi z zapisów.

```
In [219]:  
def getNGramsOcurringMoreThanOnce(words: list[str], n: int) -> dict[str, int]:  
    """  
    Returns dictionary of n-grams and their counts  
    words - list of words  
    c - n-gram count  
    """  
    ngrams = {}  
    for i in range(len(words) - n + 1):  
        ngram = ''.join(words[i:i+n])  
        if ngram in ngrams:  
            ngrams[ngram] += 1  
        else:  
            ngrams[ngram] = 1  
  
    return {k: v for k, v in ngrams.items() if v > 1}
```

```
In [220]:  
Nmax = 10  
a = [len(getNGramsOcurringMoreThanOnce(vojnick.words, x)) for x in range(2, Nmax + 2)]  
b = [len(getNGramsOcurringMoreThanOnce(catalan.words, x)) for x in range(2, Nmax + 2)]  
x = np.arange(Nmax) # to the label of the bars  
width = 0.25 # the width of the bars  
multiplier = 0  
  
fig, ax = plt.subplots(layout='constrained')
```

```
for attribute, measurement in [('Vojnick', a), ('Catalan', b)]:  
    offset = width * multiplier  
    rects = ax.bar(x + offset, measurement, width=width, label=attribute)  
    ax.bar_label(rects, padding=3)  
    ax.set_xticks(x * width + multiplier, measurement)  
    ax.legend(loc='upper right', ncol=2)
```

Prawo Zipfa dla manuskryptu Wojnicza i tekstu katalońskiego



Wnioski

Na podstawie przedstawionego wykresu, który pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu w języku katalońskim, można wynieść kilka wniosków:

- Spadek liczy N-gramów z wzrostem N. Dla obu zestawów widać wyraźny spadek liczy N-gramów wraz z wzrostem N.
- Jest spadek liczy N-gramów, ponieważ mniej liczące się N-gramy są mniej powszechnie – im więcej elementów w skwencji, tym mniejsza szansa, że sekwencja ta pojawi się w przyszłym stopniu. Można zauważyc, że tekst Wojnicza ma mniejsze N-gramy niż tekst kataloński.

```
In [221]:  
def getNGramScores(NGramsCount: dict[str, int, int]) -> dict[str, float]:  
    """  
    Returns list of tuples (ngram, score) sorted by score (score = rank * frequency)  
    """  
    sorted_freq = getNGramFrequencySorted(NGramsCount)  
  
    scores = []  
    for i in range(len(sorted_freq)):  
        scores.append((i + 1 * sorted_freq[i][1], sorted_freq[i][0]))  
    return scores
```

Aby sprawdzić zachowanie prawa Zipfa dla obu zbiorów danych, wyodrębniliśmy 1-gramy i posortowaliśmy je według częstości występowania.

```
In [222]:  
plt.figure(figsize=(10, 10))  
plt.title("Liczba N-gramów w zależności od N")  
plt.xlabel("N")  
plt.xscale("log")  
plt.yscale("log")  
plt.plot(x_vojnick, y_vojnick, 'r')  
plt.plot(x_catalan, y_catalan, 'b')  
plt.legend(['Vojnick', 'Catalan'])  
plt.show()
```

Liczba N-gramów w zależności od N



Wnioski

Analiza liczy N-gramów pokazuje, że manuskrypt Wojnicza ma mniejszą liczbę N-gramów niż tekst w języku katalońskim. W obu przypadkach wykazuje się prawidłowe położenie słów w znaczącym stopniu. Można zauważyc, że tekst Wojnicza przyjmuje pewne charakterystyki języka naturalnego, co może sugerować, że jest to język naturalny, a nie sztuczny. Jednakże spełnienie prawa Zipfa nie jest wystarczające, aby móc stwierdzić, że dany tekst jest rzeczywiście językiem naturalnym.

```
In [223]:  
def getWordGraph(words: list[str], min_count: int) -> dict[str, dict[str, int]], dict[str, dict[str, int]]:  
    """  
    Returns forward and backward graph of words with word counts  
    words - list of words  
    c - n-gram count  
    """  
    forward_graph = {}  
    backward_graph = {}  
    for i in range(len(words) - 1):  
        word = words[i]  
        next_word = words[i + 1]  
        if word not in forward_graph:  
            forward_graph[word] = {next_word: 1}  
        else:  
            forward_graph[word][next_word] += 1  
  
        if next_word not in backward_graph:  
            backward_graph[next_word] = {word: 1}  
        else:  
            backward_graph[next_word][word] += 1  
  
    return forward_graph, backward_graph
```

```
def filteredWordGraph(words: list[str], min_count: int) -> (dict[str, dict[str, int]], dict[str, dict[str, int]]):  
    """  
    Returns filtered graph with words that occur more than min_count times  
    """  
    forward_graph, backward_graph = getWordGraph(words, min_count)  
  
    forward_graph = {k: v for k, v in forward_graph.items() if sum(v.values()) > min_count}  
    backward_graph = {k: v for k, v in backward_graph.items() if sum(v.values()) > min_count}
```

Przedstawiony wykres pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu katalońskiego.

```
In [224]:  
plt.figure(figsize=(10, 10))  
plt.title("Ilość N-gramów występujących więcej niż raz w zależności od N")  
plt.xlabel("N")  
y_vojnick = [x for x in range(0, len(getNGramScores(NGramsCount(vojnick.words, 1)))][100:700]  
y_catalan = [x for x in range(0, len(getNGramScores(NGramsCount(catalan.words, 1)))][100:700]  
plt.bar(x_vojnick, y_vojnick, 'r')  
plt.bar(x_catalan, y_catalan, 'b')  
plt.legend(['Vojnick', 'Catalan'])  
plt.show()
```

Ilość N-gramów występujących więcej niż raz w zależności od N



Wnioski

Analiza liczy N-gramów pokazuje, że manuskrypt Wojnicza ma mniejszą liczbę N-gramów niż tekst w języku katalońskim. W obu przypadkach wykazuje się prawidłowe położenie słów w znaczącym stopniu. Można zauważyc, że tekst Wojnicza przyjmuje pewne charakterystyki języka naturalnego, co może sugerować, że jest to język naturalny, a nie sztuczny. Jednakże spełnienie prawa Zipfa nie jest wystarczające, aby móc stwierdzić, że dany tekst jest rzeczywiście językiem naturalnym.

```
In [225]:  
def visualizeBipartiteGraph(graph_data: tuple[dict, dict], graph_title: str):  
    """  
    Visualizes a bipartite graph from two dictionaries representing forward and backward connections.  
    """  
    plt.figure(figsize=(30, 30))  
    plt.title(graph_title)  
    plt.xlabel("Vojnick N-gramy")  
    plt.ylabel("Catalan N-gramy")  
    plt.xscale("log")  
    plt.yscale("log")  
    plt.plot(x_vojnick, y_vojnick, 'r')  
    plt.plot(x_catalan, y_catalan, 'b')  
    plt.legend(['Vojnick', 'Catalan'])  
    plt.show()
```

Przedstawiony wykres pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu katalońskiego.

```
In [226]:  
def visualizeBipartiteGraph(filteredWordGraph: WordGraph, Nmax: int, title: str):  
    """  
    Visualizes a bipartite graph from two dictionaries representing forward and backward connections.  
    """  
    plt.figure(figsize=(30, 30))  
    plt.title(title)  
    plt.xlabel("Vojnick N-gramy")  
    plt.ylabel("Catalan N-gramy")  
    plt.xscale("log")  
    plt.yscale("log")  
    plt.plot(x_vojnick, y_vojnick, 'r')  
    plt.plot(x_catalan, y_catalan, 'b')  
    plt.legend(['Vojnick', 'Catalan'])  
    plt.show()
```

Przedstawiony wykres pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu katalońskiego.

```
In [227]:  
def visualizeBipartiteGraph(filteredWordGraph: WordGraph, Nmax: int, title: str):  
    """  
    Visualizes a bipartite graph from two dictionaries representing forward and backward connections.  
    """  
    plt.figure(figsize=(30, 30))  
    plt.title(title)  
    plt.xlabel("Vojnick N-gramy")  
    plt.ylabel("Catalan N-gramy")  
    plt.xscale("log")  
    plt.yscale("log")  
    plt.plot(x_vojnick, y_vojnick, 'r')  
    plt.plot(x_catalan, y_catalan, 'b')  
    plt.legend(['Vojnick', 'Catalan'])  
    plt.show()
```

Przedstawiony wykres pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu katalońskiego.

```
In [228]:  
def visualizeBipartiteGraph(graph_data: tuple[dict, dict], graph_title: str):  
    """  
    Visualizes a bipartite graph from two dictionaries representing forward and backward connections.  
    """  
    plt.figure(figsize=(30, 30))  
    plt.title(graph_title)  
    plt.xlabel("Vojnick N-gramy")  
    plt.ylabel("Catalan N-gramy")  
    plt.xscale("log")  
    plt.yscale("log")  
    plt.plot(x_vojnick, y_vojnick, 'r')  
    plt.plot(x_catalan, y_catalan, 'b')  
    plt.legend(['Vojnick', 'Catalan'])  
    plt.show()
```

Przedstawiony wykres pokazuje liczbę N-gramów występujących więcej niż raz w zależności od N dla manuskryptu Wojnicza i tekstu katalońskiego.

```
In [229]:  
def visualizeBipartiteGraph(filteredWordGraph: WordGraph, Nmax: int, title: str):  
    """  
    Visualizes a bipartite graph from two dictionaries representing forward and backward connections.  
    """  
    plt
```

## **Podsumowanie i wnioski**

Analiza manuskryptu Wojnicza i porównanie go z tekstami w języku katalońskim ujawniają, że manuskrypt zawiera elementy charakterystyczne dla języków naturalnych. Niemniej jednak, nie ma wystarczających dowodów, by z całą pewnością określić manuskrypt jako zapis w języku naturalnym. Struktury językowe znalezione w manuskrypcie mogą być także właściwe dla języków sztucznych, które naśladowują naturalne modele lingwistyczne. Jeśli zatem przyjąć, że manuskrypt Wojnicza reprezentuje język skonstruowany, to musiałby być on wyjątkowo zaawansowany i wyrafinowany, naśladowując złożoność prawdziwych języków ludzkich w stopniu dotąd niespotykanym w innych językach sztucznych.