Przetwarzanie języka naturalnego w systemach sztucznej inteligencji

Autorzy:

- Piotr Gacek
- Jakub Szpunar

Część 1. Prawo Zipfa

Pobranie tekstu Manuskryptu Wojnicza

```
import requests

target_url = 'https://www.ic.unicamp.br/~stolfi/voynich/mirror/reeds/docs/FSG.txt'

response = requests.get(target_url)
data = response.text

sep = '# page 119 and 120'
valid_text = data.split(sep, 1)[0]
# usunięcie '-' i '='
valid_text = valid_text.replace('-', '').replace('=', '')
# usunięcie komentarzy i pustych linii
text = [x for x in valid_text.split("\n") if not x.startswith('#') and x != '\x0c']
```

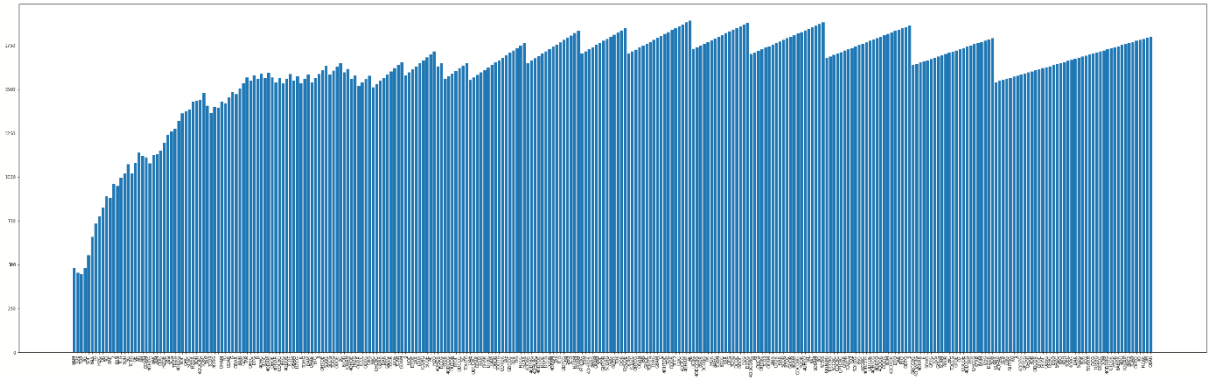
Budowanie tablicy rankingowej słów zawartych w manuskrypcie

```
count_dict = {}
words_in_text_count = 0

for line in text:
  for word in line.split(','):
      if word not in count_dict.keys():
        count_dict[word] = 1
      else:
        count_dict[word] += 1
      words_in_text_count += 1
# im większa ranga tym mniejsza częstotliwość występowania w tekście
sorted_words_count = dict(reversed(sorted(count_dict.items(),key=lambda item: item[1])))

# ranga przemnożona przez częstotliwość słowa
rank_freq_data = {}

for i, (k,v) in enumerate(sorted_words_count.items()):
```

```
  rank_freq_data[k] = (i+1) * v
```

Wykres słupkowy iloczynu rangi i częstotliwości dla 300 najczęstszych słów

```
import matplotlib.pyplot as plt
plt.bar(*zip(*list(rank_freq_data.items())[:300]))
fig = plt.gcf()
fig.set_size_inches(50, 15)
plt.xticks(rotation='vertical')
plt.show()
```



Pobranie artykułu z czeskiej Wikipedii

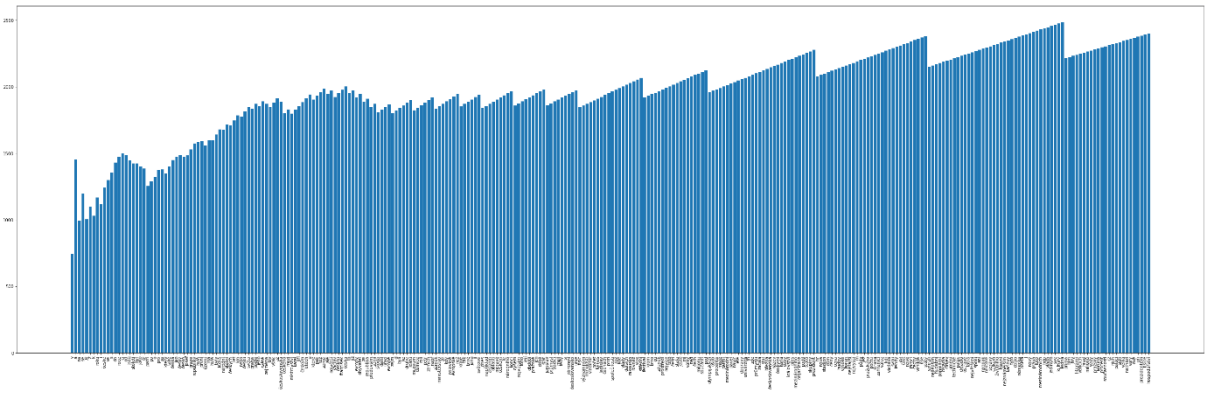
```
import wikipedia

wikipedia.set_lang("cz")
wiki = wikipedia.page('Česko')
text_cz = wiki.content


import string

for s in string.digits:
  text_cz = text_cz.replace(s,",")

forbidden_signs = string.punctuation + string.whitespace + "–"

for s in forbidden_signs:
  text_cz = text_cz.replace(s,",")

while(text_cz.find(",,") != -1):
  text_cz = text_cz.replace(",,",",")


if text[-1]==",":
  text=text[:-1]
```

Budowanie tablicy rankingowej słów zawartych w manuskrypcie

```
count_dict_cz = {}
words_in_text_count_cz = 0

for word in text_cz.split(","):
  if word not in count_dict_cz.keys():
    count_dict_cz[word] = 1
  else:
    count_dict_cz[word] += 1
  words_in_text_count_cz += 1
sorted_words_count_cz = dict(reversed(sorted(count_dict_cz.items(), key
=lambda item: item[1])))
rank_freq_data_cz = {}

for i, (k,v) in enumerate(sorted_words_count_cz.items()):
  rank_freq_data_cz[k] = (i+1) * v
```

Wykres słupkowy iloczynu rangi i częstotliwości dla 300 najczęstszych słów

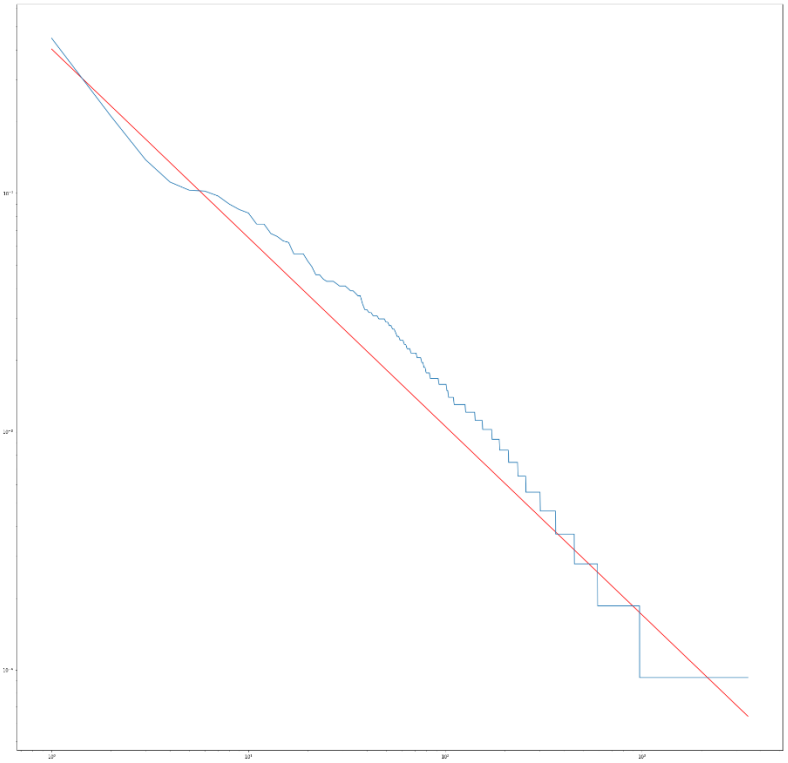
```
import matplotlib.pyplot as plt
plt.bar(*zip(*list(rank_freq_data_cz.items())[:300]))
fig = plt.gcf()
fig.set_size_inches(50, 15)
plt.xticks(rotation='vertical')

plt.show()
```

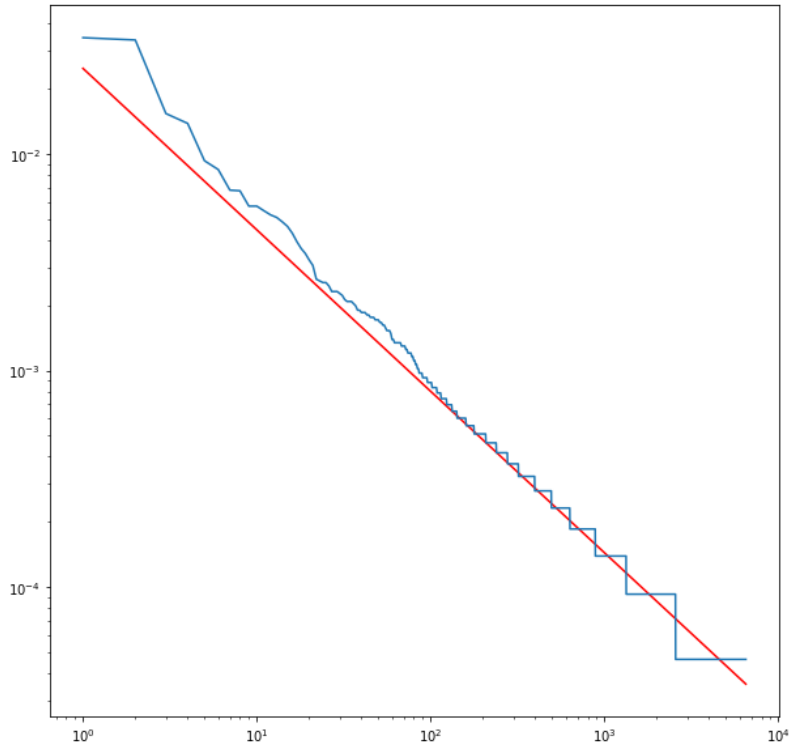


Jak widać na wykresie wartości współczynnika dla każdego słowa utrzymują się na podobnym poziomie, zwłaszcza jeśli porównamy ze sobą dwóch ‘sąsiadów’.

Wcześniej został przedstawiony wykres dla manuskryptu Wojnicza napisanego w nieznanym języku - kształt obu wykresów jest podobny.

Wykresy częstotliwości od rangi słowa dla języka w manuskrypcie Wojanowicza (pierwszy) i dla języka czeskiego z artykułu z Wikipedii (drugi) w skali logarytmicznej





Na podstawie powyższych wykresów można zauważyć, że ranga słowa i jego częstotliwość są odwrotnie proporcjonalne, co pozwala na stwierdzenie, że Prawo Zipfa jest spełnione.

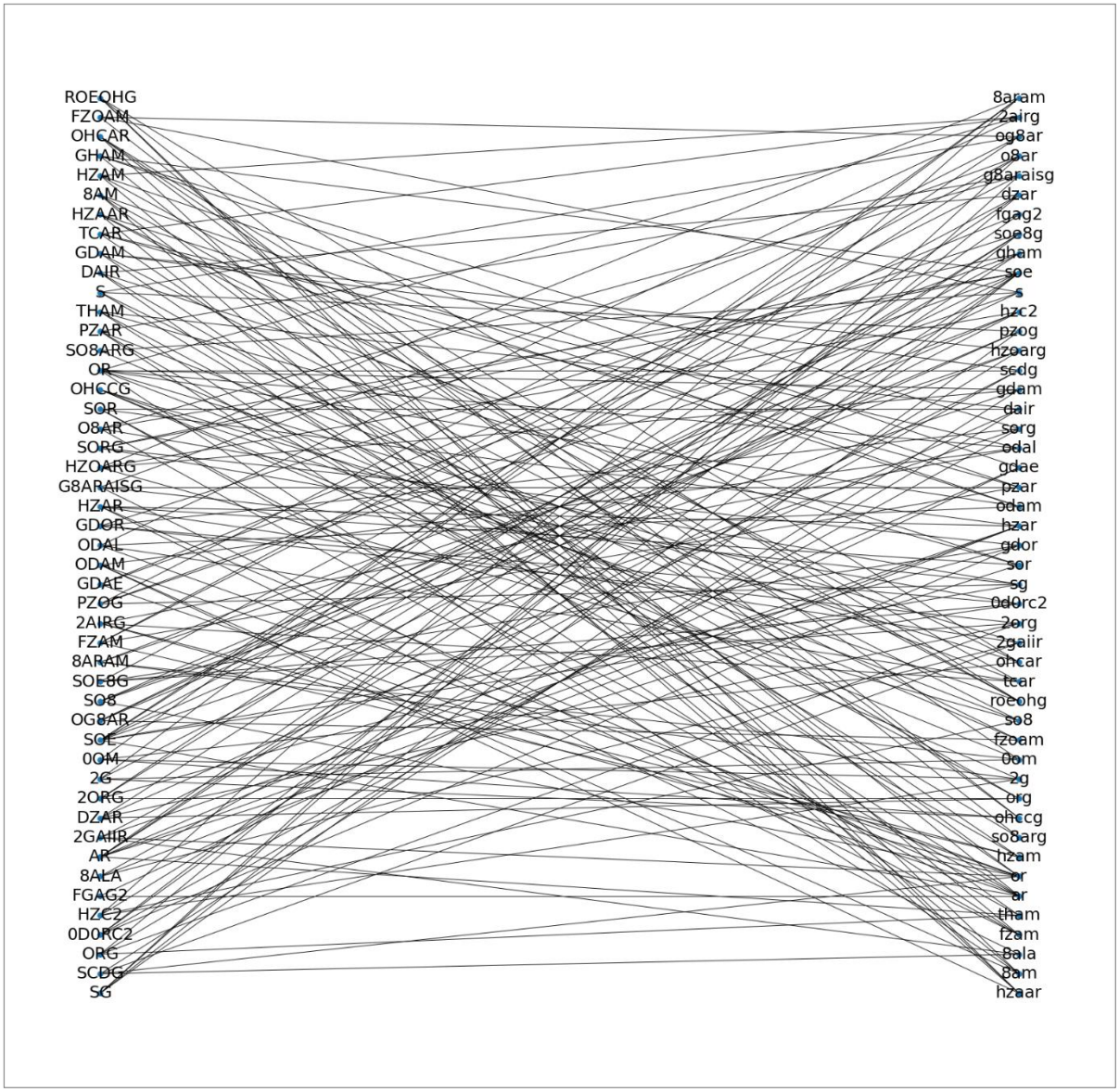
Część 2. Grafy dwudzielne

```
import networkx as nx
from networkx.algorithms import bipartite

import matplotlib.pyplot as plt
plt.rcParams["figure.figsize"] = (30,30)
```

dla języka w manuskrypcie Wojnicza

```
word_list=[]
for line in text:
  for word in line.split(','):
    if word.isnumeric():
      continue
    word_list.append(word)
  if len(word_list)>50:
    break

words_for_graphL=[]
for k in word_list:
  if k not in words_for_graphL:
    words_for_graphL.append(k)


words_for_graphR=[]
for k in words_for_graphL:
  words_for_graphR.append(k.lower())


G = nx.Graph()
G.add_nodes_from(words_for_graphL, bipartite=0)
G.add_nodes_from(words_for_graphR, bipartite=1)



# ustawianie krawędzi dla wyrazów odległych od siebie o max. 2 wyrazy
for i in range(len(word_list)):
  if i==0:
    if word_list[i] in words_for_graphL
    and word_list[i+1] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i+1].lower())])
    if word_list[i] in words_for_graphL and word_list[i+2] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+2].lower())])
  elif i==1:
```

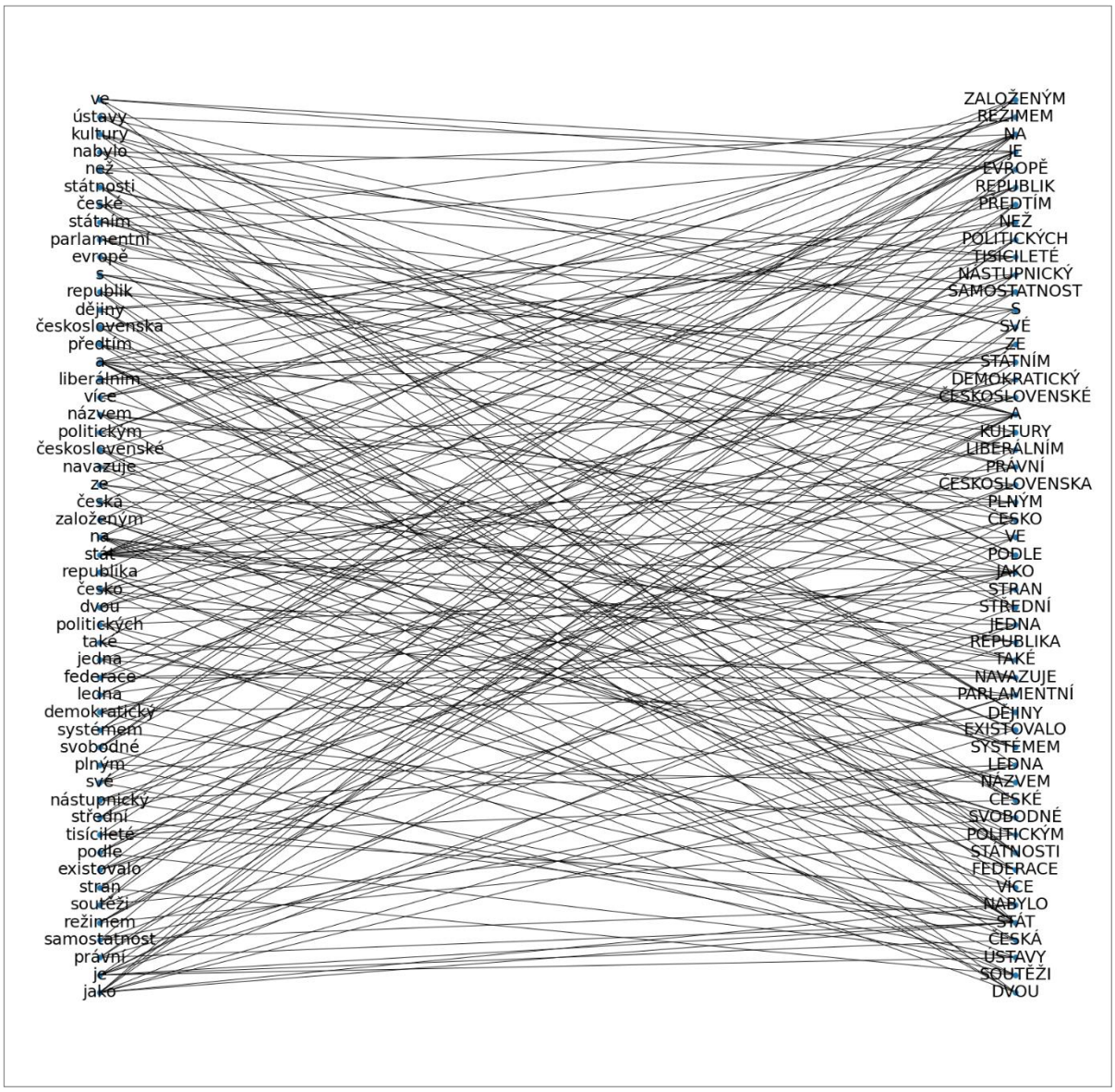
```
    if word_list[i] in words_for_graphL and word_list[i-
1] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-1].lower())])
    if word_list[i] in words_for_graphL and word_list[i+1] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+1].lower())])
    if word_list[i] in words_for_graphL and word_list[i+2] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+2].lower())])
  elif i==len(word_list)-2:
    if word_list[i] in words_for_graphL and word_list[i+1] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+1].lower())])
    if word_list[i] in words_for_graphL and word_list[i-
1] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-1].lower())])
    if word_list[i] in words_for_graphL and word_list[i-
2] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-2].lower())])
  elif i==len(word_list)-1:
    if word_list[i] in words_for_graphL and word_list[i-
1] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-1].lower())])
    if word_list[i] in words_for_graphL and word_list[i-
2] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-2].lower())])
  else:
    if word_list[i] in words_for_graphL and word_list[i+1] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+1].lower())])
    if word_list[i] in words_for_graphL and word_list[i+2] in words_for
_graphL:
      G.add_edges_from([(word_list[i], word_list[i+2].lower())])
    if word_list[i] in words_for_graphL and word_list[i-
1] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-1].lower())])
    if word_list[i] in words_for_graphL and word_list[i-
2] in words_for_graphL:
      G.add_edges_from([(word_list[i], word_list[i-2].lower())])


nx.draw_networkx(G, pos = nx.drawing.layout.bipartite_layout(G,
words_for_graphL), width = 1,node_size=60,font_size=25)
```



dla języka czeskiego

```
word_list= text_cz.split(",")[:60]
words_for_graphL=[]
for k in word_list:
  if k not in words_for_graphL:
    words_for_graphL.append(k)

words_for_graphR=[]
for k in words_for_graphL:
  words_for_graphR.append(k.upper())


G = nx.Graph()
G.add_nodes_from(words_for_graphL, bipartite=0)
G.add_nodes_from(words_for_graphR, bipartite=1)
```

```
# ustawianie krawędzi tak samo jak dla manuskryptu Wojnicza

nx.draw_networkx(G, pos = nx.drawing.layout.bipartite_layout(G,
words_for_graphL), width = 1,node_size=60,font_size=25)
```



Grafy dwudzielne dla obu tekstów wygenerowane zostały na podstawie par słów odległych od siebie o maksymalnie 2 wyrazy. Wizualnie – tak jak w przypadku wykresów słupkowego w 1. części – widać mocne podobieństwo patrząc na ilość i ”grubość” (oznaczającą częstotliwość) połączeń między słowami.

Część 3. Bigramy

dla języka w manuskrypcie Wojnicza

```
valid_text = data.split(sep, 1)[0]
lines = valid_text.split('\n')
not_empty_lines=[]
```

```
for line in lines:
  if line!='':
    not_empty_lines.append(line)
valid_lines =[]
for line in not_empty_lines:
  if line[0] not in ['#','\x0c','-',]:
    valid_lines.append(line.replace('-',''))


clear_text=''
for line in valid_lines:
  clear_text+=line


paragraphs= clear_text.split('=')[:-1]


bigrams = {}
for paragraph in paragraphs:
  words=paragraph.split(',')
  for i in range(len(words)):
    if i!=len(words)-1:
      if bigrams.get(words[i]+' '+words[i+1]) is None:
        bigrams[words[i]+' '+words[i+1]]=1
      else:
         bigrams[words[i]+' '+words[i+1]]+=1


sorted_bigrams_count = dict(reversed(sorted(bigrams.items(), key=lambda
 item: item[1])))
list(sorted_bigrams_count.items())[:10]
```

Lista najczęściej występujących bigramów dla artykułu w manuskrypcie Wojnicza:

```
[('TOE 8AM', 23),
('TOE TOE', 16),
('8AM 8AM', 9),
('TOR 8AM', 8),
('TG 8AM', 8),
('TOE SOE', 7),
('TOR TOE', 7),
('OR AM', 6),
('SOE 8AM', 6),
('TOE TOR', 6)]
```

dla języka czeskiego

```
text_cz = wiki.content
text_cz = text_cz[:].lower()
for sep in string.punctuation:
    text_cz = text_cz.replace(sep, ' ')

import re
text_cz = re.sub(' +', ' ', text_cz)


paragraphs_cz = text_cz.split('\n')
paragraphs_cz =  [x.strip() for x in paragraphs_cz if x]

bigrams_cz = {}
for paragraph in paragraphs_cz:
    # filter out empty strings
    words= list(filter(None, paragraph.split(' ')))
    for i in range(len(words)):
        if i != len(words)-1:
            if bigrams_cz.get(words[i]+' '+words[i+1]) is None:
                bigrams_cz[words[i]+' '+words[i+1]]=1
            else:
                bigrams_cz[words[i]+' '+words[i+1]]+=1

sorted_bigrams_count_cz = dict(reversed(sorted(bigrams_cz.items(), key=
lambda item: item[1])))
list(sorted_bigrams_count_cz.items())[:10]
```

Lista najczęściej występujących bigramów dla artykułu w języku czeskim:

```
[('v roce', 95),
('v praze', 44),
('v česku', 37),
('v čr', 32),
('české republiky', 32),
('od roku', 31),
('na světě', 28),
('se v', 26),
('20 století', 23),
('19 století', 21)]
```

W przypadku bigramów występuje nieco większa różnica. Czeski artykuł posiada bigramy występujące nawet 95 razy, gdzie następne w rankingu pod względem częstości bigramy mają po 40, 30 wystąpień, a pierwszą dziesiątkę zamyka bigram o liczbie 21 wystąpień.

W przypadku manuskryptu Wojnicza liczby wystąpień biogramów sięgają maksymalnie 23 dla bigramu ‘TOE 8AM’, kolejne wartości drastycznie spadają, a pierwszą dziesiątkę zamyka liczba 6 wystąpień.

W przypadku bigramów dostrzec można dość duże różnice pod względem częstotliwości wystąpień. Różnica może wynikać z trudności odszyfrowania faktycznych wyrazów w manuskrypcie przez co niektóre słowa mogą mieć kilka różnych reprezentacji w tekście.

Podsumowanie:

Na podstawie przeprowadzonych badań Manuskryptu Wojnicza oraz porównania go z artykułem w języku czeskim nie można jednoznacznie stwierdzić, czy manuskrypt jest sekwencją przypadkowych symboli czy został napisany w istniejącym języku.

Analiza obu tekstów pokazała, że manuskrypt wyróżnia się kilkoma cechami wspólnymi z czeskim artykułem co kieruje nas w stronę języka kiedyś istniejącego, lecz są to dowody niewystarczające.