Analiza tekstów polskich hitów

by: Gustaw Napiórkowski

Wstęp

W projekcie autor analizuje teksty popularnych polskich artystów muzycznych. Piosenki zostały pobrane ze strony tekstowo.pl i zapisane w formie plików tekstowych.

Wybór artystów był kierowany indywidualnymi upodobaniami autora.

Projekt był inspirowany ciekawością autora w temacie podobieństw między tymi artystami. Autora interesowało w szczególności jak bardzo podobne są zasoby słów artystów. Przeprowadzoną analizę można wykorzystać do badań prowadzonych nad muzyką popularną, a także bardzo łatwo rozszerzyć o większą gamę wykonawców i przeprowadzić tę analizę na szerszym gronie artystów. Obecna wersja jest zoptymalizowana pod język polski, jednak w prosty sposób można projekt dostosować do tekstów anglojęzycznych.

Narzędzia

Głównym narzędziem wykorzystanym przy pracy nad projektem był Python 3.7. Prostota składni tego języka pozawala na szybkie i łatwe przeprowadzanie takich analiz. Dużym atutem jest szeroka gama dostępnych narzędzi do tego języka.

Do procesowania tekstów zostały wykorzystane:

- NLTK
- Morfeusz2
- Polish stopwords list

Najistotniejszym narzędziem z wymienionych okazał się Morfeusz2, ponieważ jest to jedyne narzędzie do analizy morfologicznej polskiego języka, dla Python. Niestety Morfeusz2 nie jest dostępny dla Pythona 3.7 i autor był zmuszony przeprowadzić ten fragment analizy w wersji Pythona 2.6.10. Narzędzie to pozwala na przywrócenie słowa do jego korzenia, czyli np. słowo “czytam” zostanie zamienione na “czytać”. Umożliwia to duż dokładniejsze porównanie podobieństw między tekstami.

Innymi przydatnymi narzędziami wykorzystanymi w projekcie były:

- Scikit-learn
- WordCloud
- PyPlot form Matplotlib
- Pandas
- NetworkX

Najbardziej widowiskowym narzędziem z wymienionych jest WordCloud, pozwala ono w prosty sposób stworzyć piękne chmury tagów.

Tu widzimy przykład zasobu słów zespołu Ich Troje:

Figure 1: alt-text

Pandas, to bardzo przydatne narzędzie umożliwiające organizację danych w tabele podobne do tych, które kojarzymy z programu Excel. Dzięki niemu także zapisywanie danych i odczytywanie ich później było trywialnie proste. Przy próbie użycia plików .json jako media przenoszenia danych np do skryptu napisanego w Python 2.6.10 autor napotkał zawiłe problemy z kodowaniem znaków, a przy użyciu Pandas problem zniknął bez konieczności kombinowania.

PyPlot to także niezwykle przydatne narzędzie, które umożliwia tworzenie wykresów na bieżąco i tym samym klarowną wizualizację danych. Przykładowo tworzenie sieci wizualizującej podobieństwa pomiędzy zasobami słów wśród artysów, w połączeniu z NetworkX było możliwe.

Graf wizualizujący podobieństwa w zasobach słów artystów (ciemniejsza kreska = artyści bardziej podobni do siebie):

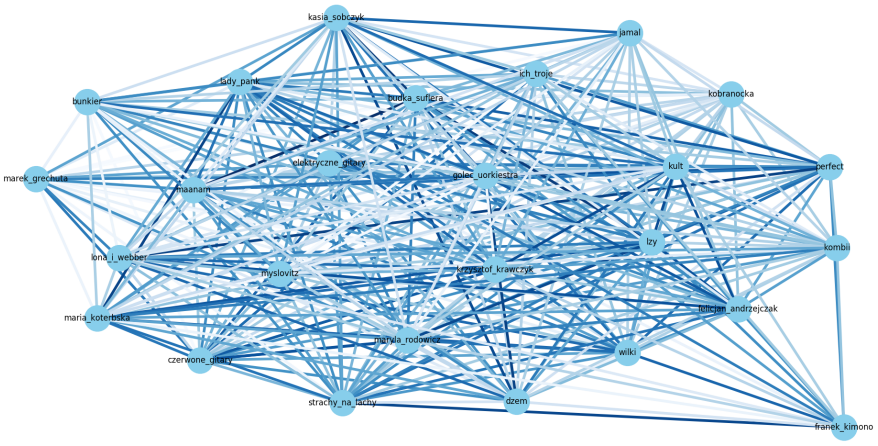

Figure 2: alt-text

Sklearn umożliwił analizę tematów.

Wszystkie wspomniane narzędzia są dostępne pod linkami na listach powyżej, wraz ze szczegółowymi instrukcjami instalacji.

Dane

Dane zostały pobrane ze strony tekstowo.pl i zapisane w formie plików tekstowych. Plik `getSongs.py` został stworzony w celu automatyzacji procesu pobierania tych danych, ponieważ ręczne pobieranie tysięcy tekstów piosenek było nierealne.

Z poszukiwań autora wynikło, że nie istnieją publicznie dostępne bazy piosenek, gotowe do pobrania, więc zostały one pobrane w ten sposób.

Skrypt ten pobiera automatycznie wszystkie piosenki artystów podanych na liście dostarczonej programowi przez użytkownika. Zdarzają się problemy w około 5% przypadków pobierania, jednak nie wpływają one w istotny sposób na końcowy wynik anazliwy.

Teoria i opis eksperymentów

Wstępne procesowanie tekstu

1. Tokenizacja
2. StopWords
3. Lematyzacja
4. *opcjonalnie* Frequency distribution

W pierwszym kroku należy przeprowadzić tokenizację tekstu, czyli rozbijanie całości na pojedyncze słowa. Wtym celu przydatny jest moduł nltk. Poniżej fragment kodu ilustrujący ten proces:

```
from nltk import tokenize
tokenizedWords = tokenize.word_tokenize(fileContents)
```

`fileContents` to zmienna zawierająca całość tekstu.

Drugim krokiem w procesie jest usunięcie słów nie mających istotnego znaczenia (stopwords).

```
from stop_words import get_stop_words
filteredWords=[]
stop_words = set(get_stop_words('polish'))
for w in tokenizedWords:
    if w not in stop_words:
        filteredWords.append(w)
```

W tym celu używamy przytoczonego w sekcji **Narzędzia** zbioru "polish stopwords".

Następnie powinniśmy przeprowadzić lematyzację, żeby zamienić słowa na ich korzenie. Czyli przykładowo słowo "chcę" zmieni nam się w "chcieć". W tym celu można wykorzystać moduł Morfeusz2 (uwaga - Morfeusz nie współpracuje z niektórymi wersjami Pythona)

```
def lemm(text):
    if type(text) is str and ' ' in text:
        morf = morfeusz2.Morfeusz()
        result_words = []
        for word in text.split(' '):
            try:
                analysis = morf.analyse(word.decode('utf-8'))
                if len(analysis) > 0:
                    result_words.append(analysis[0][2][1].encode('utf-8'))
            except:
                result_words.append(word)
        result = ' '.join(result_words)
        return result
    else:
        return text
```

Funkcja `lemm()` pozwala nam na uzyskanie takiego właśnie wyniku. Zmienna `text` w tym przykładzie jest listą słów oddzielonych spacjami (''), jednak można prosto zmodyfikować tę funkcję, żeby działała z pojedyńczymi słowami

Czwartym krokiem jest użycie funkcji freqDist() z modułu NLTK, co pozwala nam na uzyskanie wykresu pokazującego częstotliwość występowania słów.

```
from nltk import probability
freqDist = probability.FreqDist(filteredWords)
freqDist.plot(30)
```

Po uruchomieniu tego fragmentu kodu otrzymujemy wykres jak poniżej, na którym widzimy zasób słów Marka Grechuty.

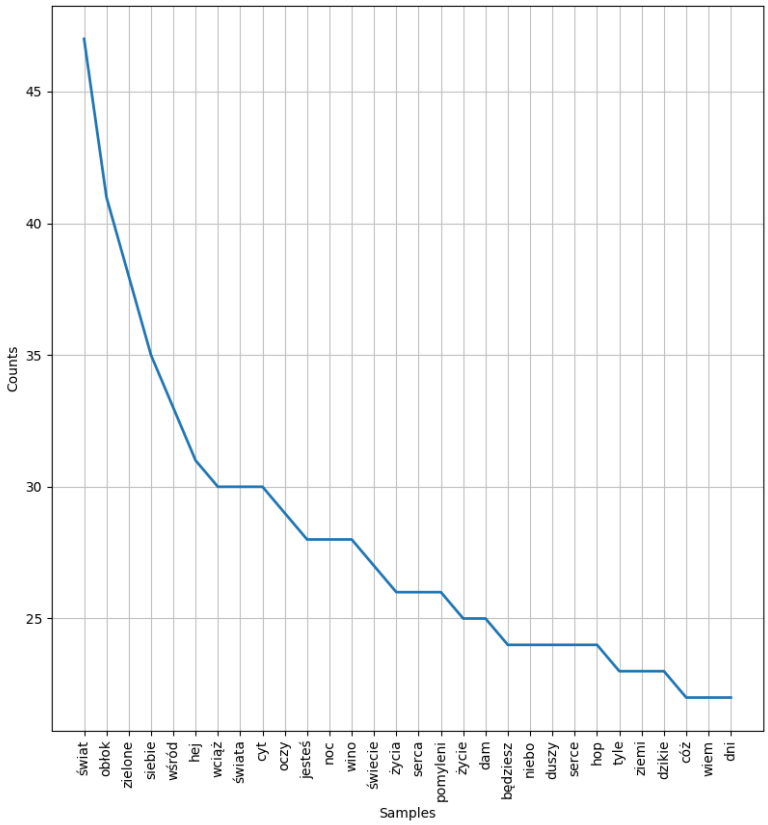

Figure 3: alt-text

Analiza tekstów

Po wstępnej obróbce słów możemy przejść do analizy. Wykres Frequency Distribution można także zaliczyć do tej części procesu.

1. WordCloud
2. Cosine Similarities
3. Similarity Chart
4. Topic Analysis

WordCloud to proste w użyciu narzędzie. Dostarczamy mu tekst i otrzymujemy wykres. Na wykresie najpopularniejsze słowa będą największe i analogicznie te najmiej popularne - najmniejsze. Istnieją opcje maskowania powstałej chmury tagów i upiększania jej, jednak nie zostanie to poruszone w tym tekście.

Poniżej przykłąd użycia WordCloud do wygenerowania chmury tagów:

```
def save_wordcloud_img(analyzed, author):
    print('Generating WordCloud')
    wordcloud = WordCloud(background_color="white", width=1600, height=800).generate(' '.joi

    wordcloud.to_file('./wordcloud/' + author + '.png')
```

Fukcja ta bierze listę słów i autora, a następnie zapisuje gotową chmurę do pliku `.png`.

Przykładowa chmura tagów wygenerowana dla Budki Suflera:

Figure 4: alt-text

W następnym kroku liczym podobieństwo cosinusowe, żeby dowiedzieć się jak bardzo artyści są podobni do siebie.

```
def text_similarities(text):
    similarities = []
    for X, Y in combinations(text.keys(), 2):
        similarities.append((textAnalysis.cosineSimilarity(text[X][2], text[Y][2]), str( X )
    sorted_sim = sorted(similarities, key = take_first)
    for i in sorted_sim:
        print(i)
```

Funckja ta bierze zmienną `text`, która jest w tym przypadku słownikiem, gdzie kluczem jest autor tekstu, a wartością lista słów po wstępnej obróbce opisanej wcześniej. Otrzymujemy posortowane podobieństwa między wszystkimi artystami. `textAnalysis` jest modułem napisanym przez autora, który oblicza podobieństwo cosinusowe w ten sposób:

```
def cosineSimilarity(X_set, Y_set):
    l1 = []
    l2 = []
    X_set = set(X_set)
    Y_set = set(Y_set)
    rvector = X_set.union(Y_set)
    for w in rvector:
        if w in X_set: l1.append(1)
        else: l1.append(0)
        if w in Y_set: l2.append(1)
        else: l2.append(0)
    c = 0
    #COSINE FORMULA
    for i in range(len(rvector)):
            c+= l1[i]*l2[i]
    cosine = c / float((sum(l1)*sum(l2))**0.5)
    return cosine
```

Funkcja ta bierze `X_set` i `Y_set`, które są po prostu listą słów wcześniej przeprocesowanych i zamienia je w zestawy bez powtórze. możemy pominąć krok:

```
    X_set = set(X_set)
    Y_set = set(Y_set)
```

i dzięki temu otrzymamy podobieństwo cosinusowe, które uwzględnia częstość powtórzenia się słowa w danym tekście. Obie metody są poprawne.

Żeby otrzymać graf z tymi podobieństwami zwizualizowanymi musimy trochę funkcję `text_similarities()` zmodyfikować i otrzymujemy coś takiego:

```
def text_similarities(text):
    similarities = []
    for X, Y in combinations(text.keys(), 2):
        similarities.append((textAnalysis.cosineSimilarity(text[X][2], text[Y][2]), str( X )
    take_first = lambda x: x[0]
    sorted_sim = sorted(similarities, key = take_first)
    for i in sorted_sim:
        print(i)

def plot_similarities(text):
    similarities = {}
    similarities['from'] = []
    similarities['to'] = []
    similarities['value'] = []
    for X, Y in combinations(text.keys(), 2):
        similarities['from'].append(str(X))
        similarities['to'].append(str(Y))
        similarities['value'].append(textAnalysis.cosineSimilarity(text[X][3], text[Y][3]))
    import pandas as pd
    import networkx as nx
    from matplotlib import pyplot as plt
    import numpy as np
    df = pd.DataFrame(similarities)
    print(df)
    df['value'] = df['value'].apply(lambda x: round((x*10)**2, 2))
    df = df.loc[df['value']>5]
    print(df.sort_values('value'))
    G=nx.from_pandas_edgelist(df, 'from', 'to', create_using=nx.Graph() )
    nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500, edge_color=df['value
    plt.show()
```

Po użyciu tej funkcji otrzymujemy taki wykres:

Funkcja ta pomija najmniejsze podobieństwa, żeby rozluźnić wykres oraz mnoży przez 10, a następnie podnosi do kwadratu wartości podobieństw w celu uwydatnienia różnic na grafie.

Topic analysis

Ostatnim krokiem jest analiza tematów w tekstach. Dzięki temu zobaczymy jakie tematy pojawiają się w nich.

```
df = pd.read_csv('processed.csv')
vectorizer = CountVectorizer(max_df=0.85, min_df=10, token_pattern='\w+|\w+|\$[\d\.]+|\S+')
tf = vectorizer.fit_transform(df['text']).toarray()
```

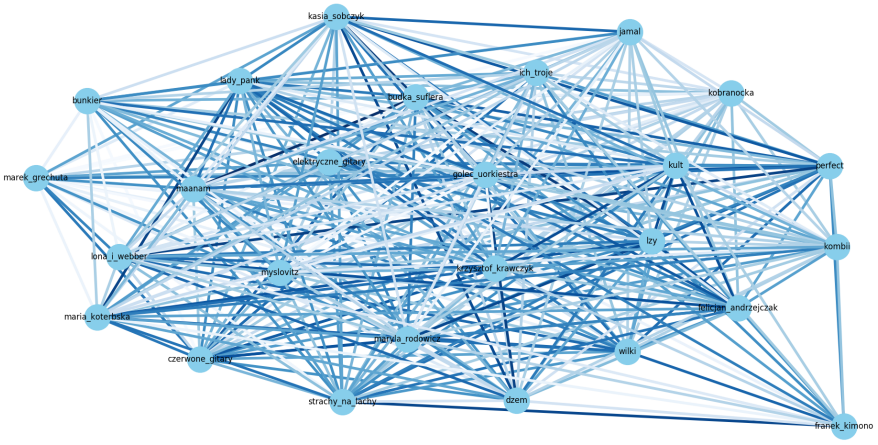

Figure 5: alt-text

```
tf_feature_names = vectorizer.get_feature_names()
number_of_topics = 50
model = LatentDirichletAllocation(n_components=number_of_topics, random_state=0)
model.fit(tf)
def display_topics(model, feature_names, no_top_words):
    topic_dict = {}
    for topic_idx, topic in enumerate(model.components_):
        topic_dict["Topic %d words" % (topic_idx)]= ['{}'.format(feature_names[i])
                        for i in topic.argsort()[:-no_top_words - 1:-1]]
        topic_dict["Topic %d weights" % (topic_idx)]= ['{:.1f}'.format(topic[i])
                        for i in topic.argsort()[:-no_top_words - 1:-1]]
    return pd.DataFrame(topic_dict)

no_top_words = 10
topics = display_topics(model, tf_feature_names, no_top_words)
print('\n ----------Topics----------\n')
print(topics)
topics.to_csv('topics_main.csv')
```

Plik `processed.csv` ma mieć następującą postać:

id	text
0	Wstępnie obrobiony tekst pierwszej piosenki
1	Wstępnie obrobiony tekst drugiej piosenki
...	...

id	text
n	Wstępnie obrobiony tekst ostatniej piosenki

Wynikiem tej analizy jest tabela wyglądająca tak:

id	Topic 0 words	Topic 0 weights	...	Topic 49 words	Topic 49 weights
0	taki	266.3	...	dzień	332.5
1	sam	258.5	...	czas	81.5
2	wysoce	94.4	...	krew	77.3
3	chcieć	69.6	...	zły	76.8
4	zostać	66.6	...	swój	65.2
5	znać	64.9	...	ciszyć	57.9
6	wiedzieć	63.1	...	chwila	57.9
7	morze	61.6	...	czarny	57.5
8	mieć	60.6	...	oczyć	56.8
9	ciągle	59.6	...	zło	54.8

Wyniki i interpretacja

WordCloud

W folderze wordcloud2 znajdują się chmury tagów dla każdego artysty.

Widać po nich, że każdy artysta faktycznie ma swój własny zasób słów - niektórzy bardziej specyficzne od inncyh. Wizualizacja tych statystyk na chmurze tagów powoduje bardzo ciekawy efekt, w którym autor patrząc na chmurę tagów był w stanie co najmiej zgodzić się z tym, że ta chura faktycznie wizualizuje twórczość tego artysty.

Poniżej na przykładzie chmurty tagów Franka Kimono widzimy jak bardzo ten obrazek uchwycił jego twórczość.

Taka wizualizacja nie przyniosła wymiernych naukowych efektów, jednak autor uważa ją za bardzo informatywną i ciekawą.

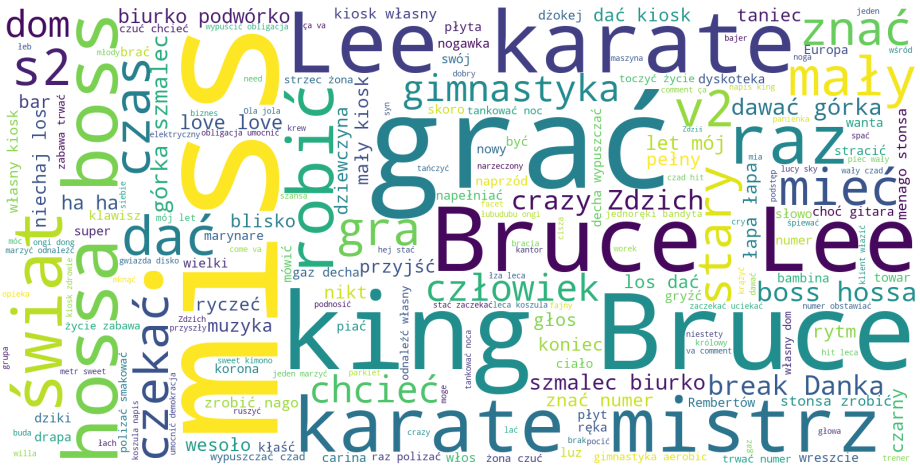

Figure 6: alt-text

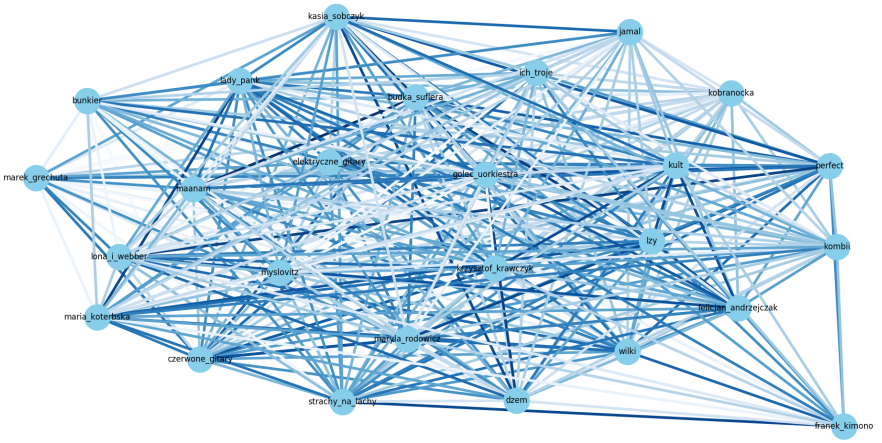

Figure 7: alt-text

Similarities Chart

Na wykresie widzimy, że faktycznie podobieństwa między artystami są różnorakie. Uwydatniają się różnice pomiędzy gatunkami muzycznymi i po głębszej analizie prawdopodobnie byłoby możliwe zakwafilikowanie każdego artysty do jego gatunku muzycznego tylko na podstawie tych podobieństw.

Topic analysis

Jest to najgłębsze zagadnienie badane w tym projekcie. Po wynikach widać, że faktycznie pojawiają się konkretne tematy w badanych tekstach i istnieje duży potencjał do dalszej analizy.

id	Topic 0 words	Topic 0 weights	Topic 1 words	Topic 1 weights	...	Topic 48 words	Topic 48 we
0	taki	266.3	tańczyć	301.7	...	słowo	493.4
1	sam	258.5	dom	144.9	...	zabrać	129.9
2	wysoce	94.4	wracać	116.1	...	mój	120.6
3	chcieć	69.6	bal	105.5	...	wiedzieć	87.5
4	zostać	66.6	mały	94.5	...	serce	85.1
5	znać	64.9	stary	79.4	...	chwila	74.5
6	wiedzieć	63.1	taniec	49.1	...	sen	68.1
7	morze	61.6	droga	44.8	...	pusty	65.2
8	mieć	60.6	niebo	44.1	...	łza	63.9
9	ciągle	59.6	lew	37.1	...	tyli	63.8

Szczegółowe wyniki analizy tematów znajdują się w pliku topics.csv.

Z kolumny słów dla danego tematu możemy wyczytać tematykę, natomiast z kolumy wag słów widzimy na co został położony nacisk w danym temacie. Ciekawym pomysłem jest zbadanie, czy analiza tematu piosenki byłaby w stanie poprawić sustem proponowania utworów przez takie platfomy jak Spotify, ponieważ wyniki uzyskane w tym projekcie pokazują, że istnieje duży potencjał w tej kategorii analizy. Widać, że słowa wysoko na listach tematów częściowo pokrywają się z ogólną częstotliwością wsytępowania słów w tekstach, co oznacza, że **prawo Zipfa** jest w pewnym stopniu spełnione.