EDA

January 21, 2020

1 Problem

Problem jaki będę się starał rozwiązać, to sprawdzene czy jesteśmy w stanie w jednoznaczny sposób zgrupować filmy na youtube bazując na ich opisach i sprawdzić w jakim stopniu grupy te pokrywają się z zapytaniem jakie zostało użyte do wyszukania tych filmów. Posłużę się różnymi metodami czyszczenia, wektoryzacji i grupowania danych. Postaram się w jasny sposób opisać moją pracę w niniejszym raporcie.

2 Wczytywanie niezbędnych narzędzi

[1]:
```
import pandas as pd
import numpy as np

from typing import List, Set, Callable

from matplotlib import pyplot as plt
import itertools

from sklearn.decomposition import PCA, KernelPCA, TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer,␣
↪HashingVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics.cluster import homogeneity_score, silhouette_score

from sklearn.cluster import KMeans, DBSCAN

from data_cleaning import DataCleaner
from analysis import clusters_overlapping, mean_distance_from_centroid

from plotly_utils import build_2d_figure, build_3d_figure,␣
↪print_correlation_heatmap
import plotly.graph_objects as go

from feature_extraction import Doc2VecVectorizer
from scipy.spatial.distance import euclidean, mahalanobis, cosine
```

3 Opis i wczytywanie zbioru danych

Zbiór danych którymi będę się posługiwał pochodzi ze strony youtube.com. Są to dane tekstowe opisujące film, takie jak tytuł, opis filmu i zapytanie jakie zostało użyte aby ten film znaleźć. Dane zostały pobrane przy użyciu API developerskiego. Scrapper do danych znajduje się w projekcie pythonowym (`scrapper.py`) dołączonym do tego raportu.

[2]:
```
df = pd.read_csv(
    "dataset_huge.csv",
    usecols=["id", "title", "search_query", "description"]
).dropna().reset_index()
```

4 Wstępna analiza zbioru danych

4.1 Wielkość zbioru

Zbiór składa się z filmów pobranych przy użyciu 20 zapytań do wyszukiwarki youtube. Zapytania to:

[3]:
```
print(df["search_query"].unique().tolist())
```

```
['programming', 'coding', 'data science', 'dogs', 'cats', 'kittens', 'monkeys',
'violin', 'guitar', 'bike', 'cars races', 'japan', 'origami', 'castle', 'disks',
'architecture', 'computers', 'pope', 'iraq', 'ukraine']
```

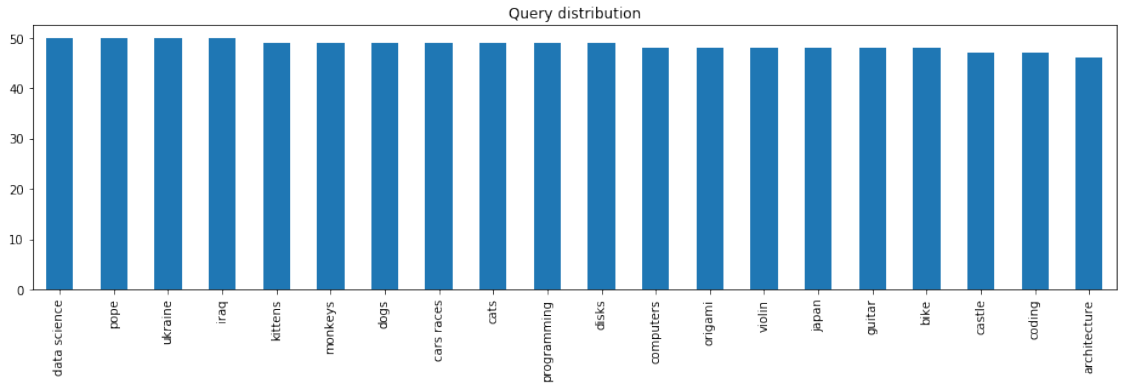
Niektóre z zapytań są powiązane, inne różne, celem było stworzenie różnorodnego zbioru.

4.2 Rozkład ilości filmów dla poszczególnych zapytań

Dane filmowe mają równomierny rozkład, każde z zapytań zawiera około 50 filmów, pojedyńcze zapytania zostały wstępnie usunięte ze względu na brak niektórych informacji.

[4]:
```
df["search_query"].value_counts().plot(kind="bar", title='Query distribution',␣
↪figsize=(16, 4))
```

[4]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee2ea7cf98>
```



4.3 Analiza opisów filmów

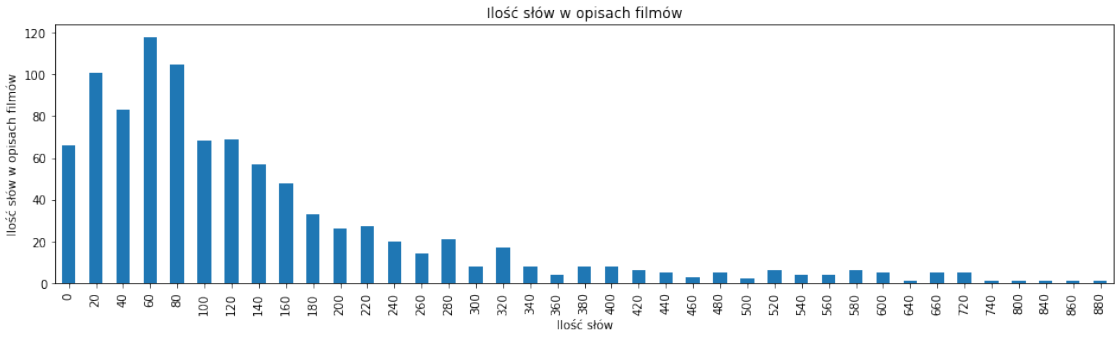
Głównym elementem z którym będę pracował w tym raporcie będą opisy pod filmami. Zawierają one najwięcej informacji tekstowych i mam nadzieję że dadzą najciekawsze wyniki podczas dalszych analiz.

Sprawdźmy zatem jaki jest rozkład ilości słów w nie przetworzonych opisach filmów z pobranym zbiorze danych.

[5]:
```
df["description"].apply(
    lambda words: int(len(words.split()) / 20) * 20
).value_counts().sort_index().plot(kind="bar", figsize=(16, 4))

plt.title('Ilość słów w opisach filmów')
plt.xlabel('Ilość słów')
plt.ylabel('Ilość słów w opisach filmów')
```

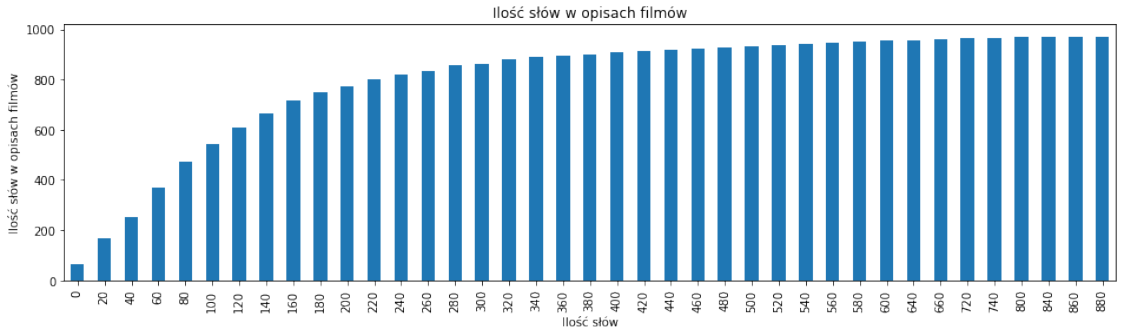
[5]:
```
Text(0, 0.5, 'Ilość słów w opisach filmów')
```



[6]:
```
df["description"].apply(
    lambda words: int(len(words.split()) / 20) * 20
).value_counts().sort_index().cumsum().plot(kind="bar", figsize=(16, 4))

plt.title('Ilość słów w opisach filmów')
plt.xlabel('Ilość słów')
plt.ylabel('Ilość słów w opisach filmów')
```

[6]:
```
Text(0, 0.5, 'Ilość słów w opisach filmów')
```



W zbiorze dominują krótkie opisy. Ponad 75% opisów posiada tekst krótszy niż 240 słów.

4.4 Analiza słownika

4.4.1 Dane nie przetworzone

[7]:
```
def build_dictionary(data: pd.Series) -> Set[str]:
    return np.unique(pd.Series(itertools.chain.from_iterable(data.str.split())))

print("Słownik zawiera: {} unikalnych słów.".
 ↪format(len(build_dictionary(df["description"]))))
```

```
Słownik zawiera: 32603 unikalnych słów.
```

4.4.2 Czyszczenie opisów

Oczyszczanie tekstu jest procesem złożonym, składającym się najczęściej z kilku kroków. Zostają one dobrane dla indywidualnych porzeb analiz i opierają się na konkretnym zbiorze danych. Proces czyszczenia tekstu wykorzystany w tym raporcie zawiera kroki: - usuwanie adresów URL - usuwanie znaków innych niż litery cyfry i białe znaki - lematyzacja - stemming

Implementacja zawiera się w module `data_cleaning.py` dołączonym do tego raportu.

[8]:
```
cleaner = DataCleaner()
df["cleaned_description"] = df["description"].apply(cleaner.clean_description)
```

```
[nltk_data] Downloading package wordnet to /home/fisz/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
```

[9]:
```
print("Po czyszczeniu słownik zawiera: {} unikalnych słów.".format(
    len(build_dictionary(df["cleaned_description"]))
))
```

```
Po czyszczeniu słownik zawiera: 16269 unikalnych słów.
```

5 Wektoryzacja danych

5.1 Metody wektoryzacji

Raport zawiera implementację 4 możliwych wektoryzacji danych. W wersji uruchomieniowej mozna zastosować każdą z nich. Wystarczy odkomentować linijkę kodu. Użyte metody wektoryzacji: * Bag of Words (CountVectorizer) - utworzenie dla każdego opisu wektora rzadkiego o wielkości całego słownika, słowa znajdujące się w konkretnym opisie reprezentowane są przez ich liczność na odpowiadającym im miejscu w wektorze danego opisu. Implementacja zawiera N-grams, tzn. wektorze są reprezentowane nie tylko pojedyńcze słowa ale też zestawy n-słów znajdujące się obok siebie (najczęsciej używane do max 3 sąsiednich słów).

- Bag of Words z hashowaniem (HashingVectorizer) - metoda bardzo podobna do klasycznego Bag of Words z tą różnicą że na wektor nie składają się konkretne słowa lub n-gramsy, a sumy kontrolne tych słów a słowa są przypisywane do konkretnych kolumn przy użyciu funkcji hashującej. Metoda daje zbliżone wyniki w reprezentacji tekstu do Bag of Words ale przy tym jest mniej zasobożerna ze względu na stałą wielkość wektora definiowaną przez użytkownika. Zawiera implementację n-grams.
- Bag of Words + TfIdf (TfidfVectorizer) - metoda bliźniacza do Bag of Words z tą różnicą że wektor wyjściowy został przetworzony algorytmem TfIdf zmieniającym wagi poszczeglnych słów używając w tym celu częstości ich występowania w danym opisie i we wszystkich opisach. Zawiera implementację n-grams.
- Doc2vec (Doc2VecVectorizer) - metoda wykorzystująca rozszerzony algorytm word2vec opisany w tej publikacji. Algorytm Doc2Vec stosuje się do tekstów w przeciwieństwie do Word2Vec który znajduje wektory słów. Algorytm działa w ten sposób że uczy płytką sieć neuronową na dwa możliwe sposoby. Sieć może przewidywać jakie słowo powinno znajdowac się w podobnym otoczeniu lub algorytm może być uczony w celu przewidywania kolejnych słów. Wektor tekstu jest reprezentowany przez wagi przedostatniej warstwy w pełni połączonej (FC - fully connected layer).

5.2 Metody redukcji cech

Raport zawiera implementację dwóch sposobów redukcji cech. W wersji uruchomieniowej mozna zastosować każdą z nich. Wystarczy odkomentować linijkę kodu. Użyte metody redukcji cech:

- SVD (TruncatedSVD) - Singular Value Decomposition - algorytm dekompozycji głównych składowych. Metoda służąca redukcji wymiaru macierzy. W połączeniu z wektoryzacją TFIdf Bag of Words stanowi technikę LSA (latent semantic analysis).
- PCA (PCA) - Principal Component Analysis - Analiza głównych składowych, algorytm ma na celu takie obrócenie układu współrzędnych macierzy aby maksymalizować wariancję kolejnych współrzędnych.

5.3 Definicja wektoryzatora i reduktora cech

[10]:
```
# vectorizer = HashingVectorizer(ngram_range=(1, 3), n_features=1000)
# vectorizer = CountVectorizer(ngram_range=(1, 3))
# vectorizer = TfidfVectorizer(ngram_range=(1, 3))
vectorizer = Doc2VecVectorizer(vector_size=400)
```

```
dimension_reductor = TruncatedSVD(n_components=3)
# dimension_reductor = PCA(n_components=3)
```

5.4 Wektoryzacja i redukcja cech

[11]:
```
embeddings = vectorizer.fit_transform(df["cleaned_description"])#.todense()

# normalization over columns (subtract min and divide by peak-to-peak method -␣
↪guarantee [0, 1] values)
# embeddings = (embeddings - embeddings.min(axis=0)) / embeddings.ptp(axis=0)

## embeddings truncation
truncated_embeddings = dimension_reductor.fit_transform(embeddings)

# labels encodding for colors purpose
encoded_labels = LabelEncoder().fit_transform(df["search_query"])
df["encoded_labels"] = encoded_labels
```

6 Grupowanie

Do grupowania zostaną użyte dwa algorytmy. Kmeans (K-średnich) i Aglomerative clustering (grupowanie aglomeracyjne).

6.1 Wstęp

6.1.1 Kmeans

Algorytm K-średnich to najpopularniejszy algorytm grupowania. Polega na wybraniu k centroidów i sprawdzaniu odległości pomiędzy obiektami i najbliższym centroidem. Obiekty przypisywane są klasie najbliższego centroidu. Pozycje centroidów są aktualizowane najczęściej na podstawie średniej arytmetycznej z obiektów przypisanych do klasy danego centroidu. Po aktualizacji pozycji centroidu algorytm wykonywany jest ponownie, iterujemy tak do momentu braku potrzeby aktualizacji centroidu.

6.1.2 DBScan

Density-Based Spatial Clustering of Applications with Noise - algorytm grupowania bazujący na gęstości obserwacji. Znajduje na początku obserwacje bazowe i na podstawie gęstości wyszukuje najbliższe obserwacje i grupuje je w klasy.

6.2 Klastrowanie

[12]:
```
# same clusters number as original dataset
clusters_number = df["search_query"].unique().shape[0]
```

```
df["predicted_classes"] = KMeans(n_clusters=clusters_number).
 ↪fit_predict(embeddings)
# df["predicted_classes"] = DBSCAN(eps=0.5, min_samples=2, metric='euclidean').
 ↪fit(embeddings).labels_
# df["predicted_classes"] = DBSCAN(eps=0.5, min_samples=2, metric='cityblock').
 ↪fit(embeddings).labels_
# df["predicted_classes"] = DBSCAN(eps=0.5, min_samples=2, metric='cosine').
 ↪fit(embeddings).labels_
# df["predicted_classes"] = DBSCAN(eps=0.5, min_samples=2, metric='manhattan').
 ↪fit(embeddings).labels_
```

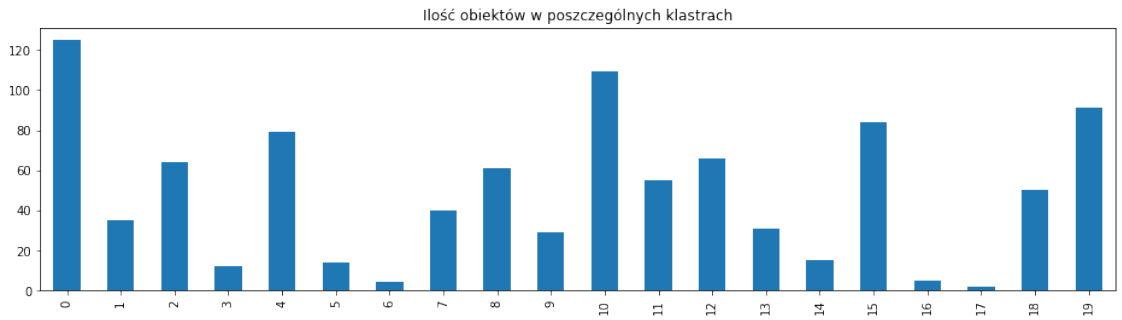
6.3 Badanie klastrów

Celem badań jest sprawdzenie w jak dużym stopniu klastry wykryte przez algorytm pokrywają się z rzeczywistymi klastrami wygenerowanymi przez zapytania do youtube. Będę badać rozpiętość klastrów w przestrzeni cech i rozłożenie ich elementów. Zostaną wykorzystane też metryki dedykowane do walidacji jakości klastrowania na konkretnych danych.

6.3.1 Wielkość utworzonych klastrów

[13]:
```
df["predicted_classes"].value_counts().sort_index().plot(
    kind="bar", figsize=(16, 4), title="Ilość obiektów w poszczególnych␣
 ↪klastrach"
)
```

[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee2ea7cbe0>
```



6.3.2 Korelacja klas rzeczywistych z utworzonymi

Diagram przedstawia ilość obserwacji znajdujących się w obu klasach.

[14]:
```
print_correlation_heatmap(
    clusters_overlapping(df["search_query"], df["predicted_classes"])
)
```

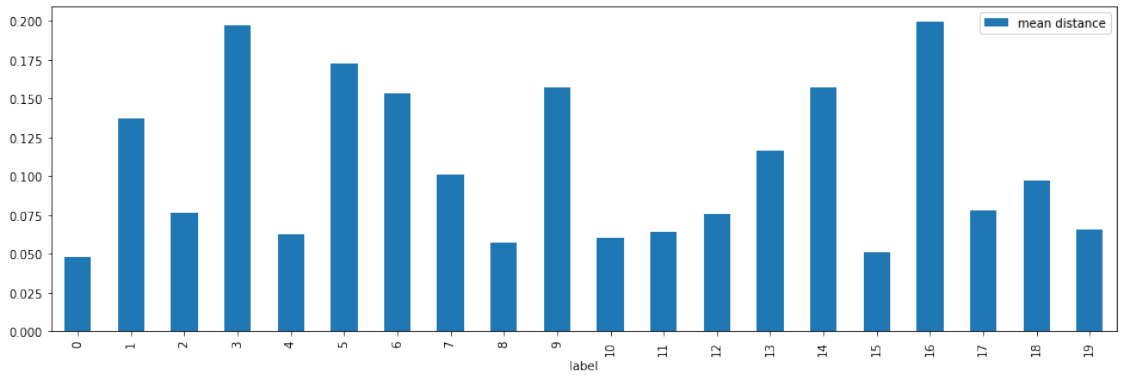
Wiele klas jest skorelowanych z jedną konkretną klasą utworzoną algorytmem grupowania. Można domniemywać iż są to klasy odpowiadające sobie. Maksymalna ilość obserwacji pokrywających się jest równa 11 co jest 1/5 wielkości rzeczywistego klastra.

6.3.3 Średnia odległość obserwacji od centroidów klas

Badanie to pozwoli nam sprawdzić jak mocno skupione są klasy wokół ich centroidów, im mniejszy wynik tym grupa obserwacji jest bardziej skupiona.

[15]:
```
mean_distance_from_centroid(euclidean, df["predicted_classes"], embeddings).
↪sort_values("label").plot(
   kind="bar", x="label", y="mean distance", figsize=(16,5))
```

[15]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fee0ff3edd8>
```



Zbadanie średnich odległości obserwacji od centroidów klas pozwala nam stwierdzić jak mocno klasy rozłożone są w przestrzeni cech.

6.3.4 Homogeniczność grup

Algorytm Silhouette Jest to algorytm którego wynik pozwala stwierdzić czy w odpowiedni sposób została dobrana ilość klastrów do danych. Działanie tego algorytmu oparte jest na dwóch wielkościach `a` - odległość obserwacji od centroidu jej klasy i `b` czyli odległość do najbliższego centroidu klasy do której obserwacja nie należy. Wzór na współczynnik silhouette to : `(b - a) / max(a, b)`. Wartość współczynnika mieści się w granich [-1, 1] gdzie 1 to wynik najlepszy, -1 to wynik najgorszy.

Wartość współczynnika dla klas rzeczywistych:

[16]:
```
silhouette_score(embeddings, labels=df["search_query"])
```

[16]:
```
-0.25688064
```

Wartość jest ujemna, znaczy to że grupowanie po klasie rzeczywistej nie jest zbyt dobre. Obserwacje przypisane do poszczególnych klas są bardzo mocno rozrzucone w przestrzeni cech, grupy

nie są spójne i separowalne w prosty sposób. Wynik metryki dla ilości klas odpowiadającej ilości rzeczywistej przewidzianych klas:

[17]:
```
silhouette_score(embeddings, labels=df["predicted_classes"])
```

[17]:
```
0.39417335
```

Wynik jest o wiele lepszy niż ten osiągnięty dla klas rzeczywistych. Prawdopodobnie zmiana ilości klas jeszcze bardziej poprawiła by wynik.

6.3.5 Wizualna reprezentacja przestrzeni obserwacji 2d - klasy rzeczywiste

[18]:
```
build_2d_figure(df, df["encoded_labels"], truncated_embeddings).show()
```

6.3.6 Wizualna reprezentacja przestrzeni obserwacji 2d - klasy przewidywane

[19]:
```
build_2d_figure(df, df["predicted_classes"], truncated_embeddings).show()
```

6.3.7 Wizualna reprezentacja przestrzeni obserwacji 3d - klasy rzeczywiste

[20]:
```
build_3d_figure(df, df["encoded_labels"], truncated_embeddings).show()
```

6.3.8 Wizualna reprezentacja przestrzeni obserwacji 3d - klasy przewidywane

[21]:
```
build_3d_figure(df, df["predicted_classes"], truncated_embeddings).show()
```

7 Perspektywy dalszego rozwoju

7.1 Większa ilość danych

Zbiór danych jest reprezentowany przez 1000 obserwacji rozłożonych na 20 klas. Reprezentacja poszczególnych klas wynosi 50 obserwacji. Prawdopodobnie nie jest to wystarczająca ilościowa reprezentacja poszczególnych klas dla tak złożonych danych jakimi są opisy tekstowe. Prawdopodobnie zwiększenie reprezentacji o rząd wielkości dla każdej klasy, tzn. 500 obserwacji, bardzo mocno poprawiła by wyniki grupowania.

W przypadku rozszerzenia zbioru danych być może należało by zastosować inne narzędzia do ich przetwarzania. Dla bardzo dużych wolumenów danych zasadnym staje się użycie technologii chmurowych np. Google Cloud Platform (GCP). Chmura google zawiera dedykowane rozwiązania do zarządzania dużymi wolumenami danych takie jak np. BigQuery bądź nawet BigTable (jeśli wolumen danych przekroczy 2TB). Rozwiązanie takie wymagało by poważnego zmodyfikowana przedstawionego tutaj algorytmu, jednakże jest możliwe.

7.2 Reprezentacja wektorowa obserwacji

Przy rozwoju aplikacji można by było zastosować bardziej złożone sposobu wektoryzacji obserwacji np. wykorzystanie najnowszych rozwiązań w NLP jakim jest Transformer BERT. Transfer wiedzy

z modelu przeuczonego na słowniku dla języka angielskiego, prawdopodobnie dała by bardziej reprezentatywne wektory obserwacji.

8 Podsumowanie rezultatów

Cel podstawiony na początku tego opracowania udało się osiągnąć. Jednak rezultat nie jest jednoznacznie dobry. Udało się stwiedzić że grupy utowrzone przy pomocy algorytmów grupowanie są znacząco rózne od etykiet które zostały przypisane w momencie wyszukiwania filmów. Pewne klasy w danych są spójne, ale w większości klasy są reprezentowane w dużej części przestrzeni cech i nie można jednoznacznie stwierdzić przygotować grupowania które było by spólne z rzeczywistym przypisaniem do klas.

Być może zwiększenie wolumenu danych i zastosowanie bardziej zaawansowanych metod wektoryzacji pozwoliły by osiągnąć lepsze wyniki.