Analiza komentarzy meczów piłkarskich

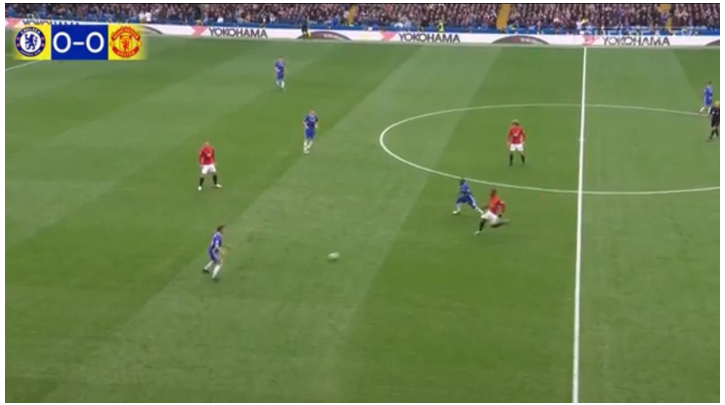
Mikołaj Kaczmarek
we współpracy z dr hab. inż. Szymonem Łukasikiem

Agenda

1. Cel projektu
2. Dane wejściowe oraz ich preprocessing
3. Transkrypcja
4. Wyszukiwanie kluczowych momentów
5. Generowanie podsumowania meczu
6. Potencjalne usprawnienia
7. Podsumowanie

Cel projektu

Cel projektu



Wideo

8'

Attempt missed. Zlatan Ibrahimovic (Manchester United) header from the left side of the six yard box is just a bit too high. Assisted by Antonio Valencia with a cross.

4'

Corner, Manchester United. Conceded by César Azpilicueta.

1'

Booking

Pedro (Chelsea) is shown the yellow card for excessive celebration.

1'

Goal!

Goal! Chelsea 1, Manchester United 0. Pedro (Chelsea) left footed shot from the centre of the box to the centre of the goal. Assisted by Marcos Alonso with a through ball.

Podsumowanie tekstowe

Dane wejściowe oraz ich preprocessing

Dane wejściowe oraz ich preprocessing

- nagranie wideo z meczu wraz z komentarzem

- preprocessing wideo:
  - pobranie nagrania z meczu
  - ekstrakcja audio z nagrania

Pobieranie wideo

- na YouTube znajduje się wiele nagrań meczów w całości, wraz z komentarzem
- pakiet **`pytube`** udostępnia API do pobierania filmów z poziomu Pythona

```python
from pytube import YouTube
import os

video_filename: str = "video_eng.mp4"
data_dir: str = os.path.abspath("data")


YouTube(
    "https://www.youtube.com/watch?v=QP25ucU8vfI&ab_channel=ChelseaFootballClub"
).streams.filter(progressive=True, file_extension="mp4").first().download(
    data_dir, video_filename
)
```

Ekstrakcja ścieżki audio

- pakiet **`moviepy`** pozwala na “obróbkę” plików multimedialnych, m.in. ekstrakcja ścieżki audio z filmu
- formatem najczęściej wykorzystywanym w NLP (a konkretnie Speech To Text Recognition) jest .wav

```python
import os
import moviepy.editor as mp

data_dir: str = os.path.abspath("data")
video_path: str = os.path.join(data_dir, "video_eng.mp4")
audio_path: str = os.path.join(data_dir, "audio.wav")

video: mp.VideoFileClip = mp.VideoFileClip(video_path).subclip()

video.audio.write_audiofile(audio_path, codec="pcm_s16le")
video.close()
```

Transkrypcja

Transkrypcja audio

- do transkrypcji postanowiliśmy skorzystać z gotowej usługi Speech To Text
- kryteria wyboru narzędzia:
  - cena
  - wsparcie dla języka polskiego (lub angielskiego)
  - możliwość rozpoznawania nazw własnych, w tym przypadku nazwisk piłkarzy (*named entities)*
  - możliwość określenia dokładnych momentów w czasie wystąpienia danego słowa (*timestampy*)
  - łatwość użycia

Przegląd narzędzi Speech To Text

Nazwa	Darmowe	Angielski	Polski	Named entities (angielski)	Named entities (polski)	Timestampy	Ogólne wrażenie
Azure Speech Services	❌	✅	✅	✅	❌	✅	😊
Google Cloud Speech-To-Text	❌	✅	✅	✅	❌	❌	😐
Amazon Transcribe	❌	✅	❌	❌	❌	❌	😢
IBM Watson Speech-To-Text	❌	✅	❌	❌	❌	❌	😢
Vosk	✅	✅	✅	❌	❌	✅	😐

Link do pełnego przeglądu

“Podkręcanie” skuteczności transkrypcji

- wiele usług Speech To Text (szczególnie tych komercyjnych) ma możliwość drobnego dostosowywania modelu do indywidualnych potrzeb,
- jest to realizowane poprzez przekazywanie do modelu zestawu specyficznych słów (np. nazwisk piłkarzy), których można się spodziewać w transkrypcie,
- zabieg ten poprawia skuteczność rozpoznawania tych nietypowych słów.
- przykład: “david de hey” -> “David De Gea” dzięki rozszerzeniu modelu o słowa “David”, “De”, “Gea”

Wyszukiwanie kluczowych momentów

Wyszukiwanie słów kluczowych

- aby zlokalizować interesujące sytuacje w meczu, dobrym podejściem jest poszukiwanie w transkrypcie konkretnych słów kluczowych, takich jak np. “podanie”, “gol”, “strzał”, “dośrodkowanie”, itp., a także samych nazwisk piłkarzy,
- dzięki temu, że każde wystąpienie słowa jest związane z konkretnym *timestampem*, dzięki wyszukaniu słów kluczowych, mamy dostęp do informacji, które momenty w meczu mogą być uznane za interesujące

Fuzzy String Matching

- poszukiwanie słów kluczowych z dokładnością co do znaku nie jest optymalne, ponieważ drobna różnica w formie gramatycznej może spowodować, że wystąpienie słowa w transkrypcie zostanie pominięte
- dlatego do poszukiwania tych słów, dobrym sposobem jest wykorzystanie tzw. fuzzy string matching (lub tzw. *approximate string matching*)
- jest to technika porównywania słów z pewnym przybliżeniem, a dzięki parametrowi podobieństwa, możliwe jest regulowanie tego jak bardzo dwa słowa mogą się różnić, żeby nadal zostały uznane za te same

Fuzzy String Matching

- działanie algorytmu oparte jest na tzw. Odległości Levenshteina, która dystans między dwoma słowami definiuje jako minimalną liczbę operacji (dodanie/usunięcie/podmiana litery), aby jedno słowo zmienić w drugie
- przykład “bruk” oraz “byk” - odległość to 2, ponieważ ze słowa bruk należy usunąć literę “r” oraz zmienić “u” na “y”
- im większa odległość Levenshteina, tym mniej dwa słowa są do siebie podobne.
- z koncptu tego korzysta pakiet `thefuzz` umożliwiając sprawdzanie podobieństwa między słowami w języku Python

Generowanie podsumowania meczu

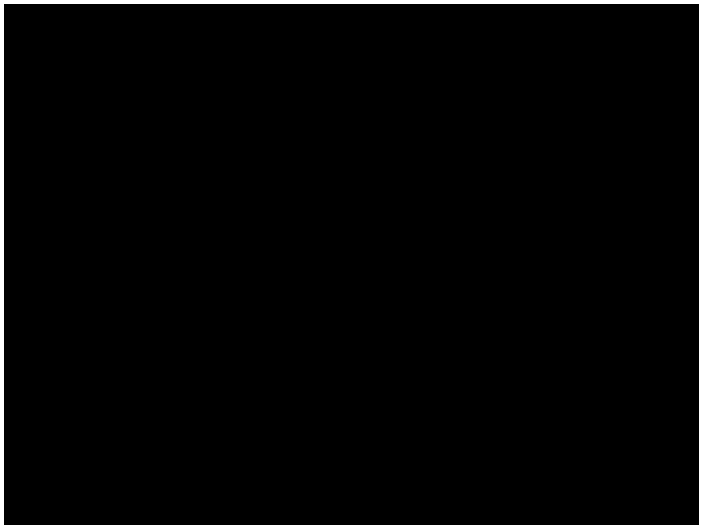
Generowanie podsumowania meczu

- wydarzenia generowane są na podstawie timestampów uzyskanych z lokalizacji słów kluczowych
- algorytm polega na iteracji po posortowanych timestampach i przypisywania do danego wydarzenia słów z transkryptu tak długo aż między aktualnym timestampem, a poprzednim różnica w czasie jest nie większa niż określony limit (np. 5 sekund)
- po przeiterowaniu po wszystkich timestampach, otrzymujemy listę zdarzeń, gdzie do każdego z nich przypisany jest timestamp związany z pierwszym słowem zdarzenia

Generowanie podsumowania meczu

- dodatkowo, aby poprawnie określić minutę zdarzenia, należy uwzględnić momenty rozpoczęcia pierwszej oraz drugiej połowy.
- wydaje się, że dobrym wyznacznikiem tych zdarzeń jest dźwięk gwizdka sędziego, jednak wiąże się z tym wiele trudności i problemów (o tym później)
- niemniej jednak tutaj manualnie wyznaczono początki obu połów i wykorzystano te momenty do dokładnego określenia momentu wystąpienie wyznaczonego zdarzenia.

Przykład



Wideo


```
{
    "word": "Pedro",
    "offset_s": 38.15,
    "duration_s": 0.61
},
{
    "word": "gets",
    "offset_s": 38.77,
    "duration_s": 0.28
},
{
    "word": "the",
    "offset_s": 39.06,
    "duration_s": 0.12
},
{
    "word": "goal",
    "offset_s": 39.19,
    "duration_s": 0.45
},
{
    "word": "put",
    "offset_s": 40.35,
    "duration_s": 0.2
},
{
    "word": "it",
    "offset_s": 40.56,
    "duration_s": 0.05
},
{
    "word": "away",
    "offset_s": 40.62,
    "duration_s": 0.2
},
{
    "word": "to",
    "offset_s": 40.83,
    "duration_s": 0.15
},
{
    "word": "mark",
    "offset_s": 40.99,
    "duration_s": 0.22
},
{
    "word": "his",
    "offset_s": 41.22,
    "duration_s": 0.14
},
{
    "word": "fiftieth",
    "offset_s": 41.37,
    "duration_s": 0.47
},
{
    "word": "appearance",
    "offset_s": 41.85,
    "duration_s": 0.53
},
```


Transkrypt


```
"predicted_report": {
    "first_half": {
        "1": [
            "chance of kicking towards",
            "one centre forward one come pushing on from midfield and nothing else now might be a chance for petro who's",
            "Pedro gets the goal put it away to mark his fiftieth appearance in blue chelsea ahead with less than"
        ],
        "2": [
```


Raport

Potencjalne usprawnienia

Automatyczna lokalizacja dźwięków gwizdka

- aby poprawnie określić minutę wystąpienia akcji w meczu, potrzebne jest określenie kiedy dokładnie zaczyna się pierwsza i druga połowa
- Przykładowy algorytm: artykuł "Sound Pattern Recognition with Python"

Trudności:

- dźwięk gwizdka nie występuje tylko na początku “połów”, ale przy każdym przerwaniu akcji
- dźwięki gwizdka oraz gwizdów kibiców są bardzo podobne i łatwo je pomylić
- gwizdek może być zagłuszony przez komentatora

Rozszerzenie modelu STT o dodatkowe słowa

- oprócz nazwisk, do “podkręcenia” skuteczności STT, można by wykorzystać również słowa kluczowe wykorzystane do wykrywania interesujących sytuacji w meczu

Podsumowanie

Podsumowanie i wnioski

- projekt był świetną okazją do zapoznania z NLP oraz narzędziami do STT
- brak skomplikowanych algorytmów UM, a jedynie oparcie się na prostych heurystykach dało zaskakujące dobre wyniki
- odpowiednie sprawdzenie skuteczności algorytmu (przez porównanie z “prawdziwym” podsumowaniem) pozwoliłoby na optymalne dostosowanie parametrów algorytmu
- wprowadzenie wspomnianych usprawnień mogłoby polepszyć algorytm jeszcze bardziej
- repozytorium na GitHub’ie