Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Komputerowe Wspomaganie Decyzji

Projekt zaliczeniowy
zespół nr 2

Ocena czasu potrzebnego na dekodowanie filmu

Dariusz Hatala 31i
Oxana Danilova 31i

Pełny kod źródłowy dostępny pod adresem:
`https://github.com/caspinos/kwd_ml_project`

3 lutego 2020

Spis treści

1 Cel i zakres projektu

Celem projektu jest utworzenie modelu mogącego służyć w celu oceny czasu potrzebnego na transkodowanie filmu z jednego formatu do innego przy jednoczesnej zmianie rozdzelczości.

1.1 Źródło danych wejściowych

Dane wejściowe stanowi baza danych:
Online Video Characteristics and Transcoding Time Dataset Data Set.

Dane zostały przygotowane przez *Tewodros Deneke* (tdeneke@abo.fi).

1.2 Zawartość danych wejściowych

Dane wejściowe składają się z ponad 68 tyś wpisów zawierających parametry filmów źródłowych i docelowyc oraz czas transkodowania.

Opis poszczególnych danych:

id Youtube video id

duration czas trwania filmu

bitrate bitrate(video) bitrate filmu

height wysokość filmu w pikselach

width wysokość filmu w pikselach

frame rate framerate filmu

codec standard kodowania filmu

i liczba klatek typu i (pełne kaltki)

p liczba klatek typu p (przewidywana klatka)

b liczba klatek typu b (obustronnie przewidywana klatka)

frames całkowita liczba klatek

i_size całkowity rozmiar kaltek typu i [bytes]

p_size całkowity rozmiar kaltek typu p [bytes]

b_size całkowity rozmiar kaltek typu b [bytes]

size całkowity rozmiar filmu

o_codec wyjściowy kodek

o_bitrate wyjściowy bitrate

o_framerate wyjściowy framerate

o_width wyjściowa szerokość filmu

o_height wyjściowa wysokość filmu

umem całkowita ilość pamięci zaalokowana w czasie transkodowania

utime całkowity czas transkodowania

2 Przygotowanie środowiska

Do analizy oraz uczenia maszynowego zostało użyte środowisko Jupyter Notebook oraz język Python 3.

Dodatkowo, wykorzystane zostały następujące biblioteki:

- NumPy (numpy.org)
- Pandas (pandas.pydata.org)
- Matplotlib (matplotlib.org)
- scikit-learn (scikit-learn.org)
- Category Encoders (contrib.scikit-learn.org/categorical-encoding)

3 Analiza eksploracyjna danych

3.1 Ogólne informacje o danych

Baza danych zawiera 68784 wierszy oraz 22 kolumny.
Dane nie zawierają brakujący wartości.
Dane nie zawierają zduplikowanych wierszy.

Kod użyty do przeprowadzenia analizy:

```
print(data.shape)
print(data.columns)
```

```
data.head(10)
```

```
data.info()
```

```
data.describe()
```

```
len(data[data.duplicated()])
```

3.2 Badanie korelacji

Mapa korelacji ze względu na rozmiar została załączona poza dokumentem (plik *correlations.png*).

Poza oczywistymi zależnościami, można zauważyć, że czas trankodowania jest (liniowo) zależny w znacznym stopniu tylko od parametrów docelowego formatu filmu.

```
sns.set()

corr = data.sample(1000).corr()

fig, ax = plt.subplots(figsize=(50,30))
sns.heatmap(corr, annot=True, linewidths=.5, ax=ax)
plt.show()
```

3.3 Szczegółowe badanie poszczególnych kolumn numerycznych

Dla każdej kolumny zostały wyświetlone następujące dane:

- nazwa kolumny
- liczba unikalnych wartości
- histogram

```
numeric_columns =
        data.select_dtypes(include=np.number).columns
for col in numeric_columns:
print(f'kolumna: {col}')
print(f'unikalnych wartosci: {len(data[col].unique())}')
data[col].plot.hist(bins=40)
plt.show()
```

Wnioski: Kolumna 'b_size' nie zawiera niezerowych wartości. Można ją usunąć.

Kolumna 'b' jest niezerowa w 1% przypadków.

Kolumna 'b' jest niezerowy tylko w przypadku użycia kodeka 'h264'.

3.4 Szczegółowe badanie poszczególnych kolumn nienumerycznych

Dla każdej kolumny zostały wyświetlone następujące dane:

- nazwa kolumny
- liczba unikalnych wartości
- histogram

```
non_numeric_columns
        = data.select_dtypes(exclude=np.number).columns
for col in non_numeric_columns:
  print(f'kolumna: {col}')
  print(f'unikalnych wartosci: {len(data[col].unique())}')
  if len(data[col].unique()) < 15:
    data[col].value_counts().plot.bar()
    plt.show()
```

4 Obsługa outlierów

Do obcięcia outlierów pomocniczo wykorzystujemy poniższe wyznaczniki:

$$IQR(interquartilerange) = P(75) - P(25) \tag{1}$$

$$Dolna_granica = P(25) - 1,5 * IQR \tag{2}$$

$$Gorna_granica = P(75) + 1,5 * IQR \tag{3}$$

gdzie P oznacza percentyl.

Funkcja do wyznaczania powyższych zakresów:

```
def outliners_range(data, column_name):
    rows = data[column_name]
    iqr = np.nanpercentile(rows, 75) - np.nanpercentile(rows, 25)
    lower = (np.nanpercentile(rows, 25) - 1.5*iqr)
    upper = (np.nanpercentile(rows, 75) + 1.5*iqr)
    return lower, upper
```

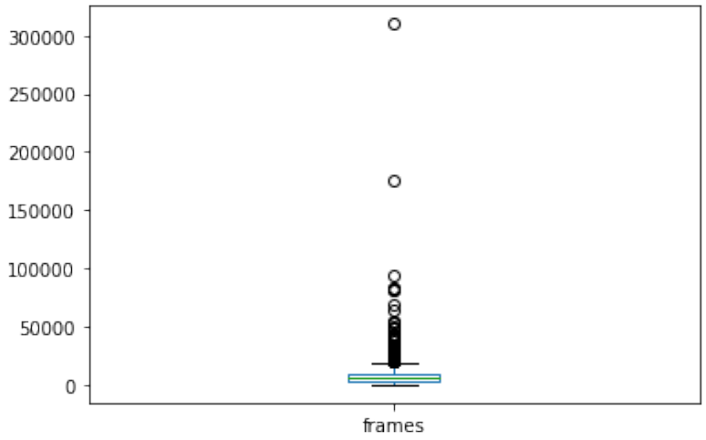
Dla każdej kolumny zostały przygotowane 2 wykresy:

- histogram
- wykres pudełkowy

```
for column in numeric_data:
  print(f'Kolumna: {column}')
  numeric_data[column].plot.hist()
  plt.show()
  numeric_data[column].plot.box()
  plt.show()
```

4.0.1 Kolumna 'frames'

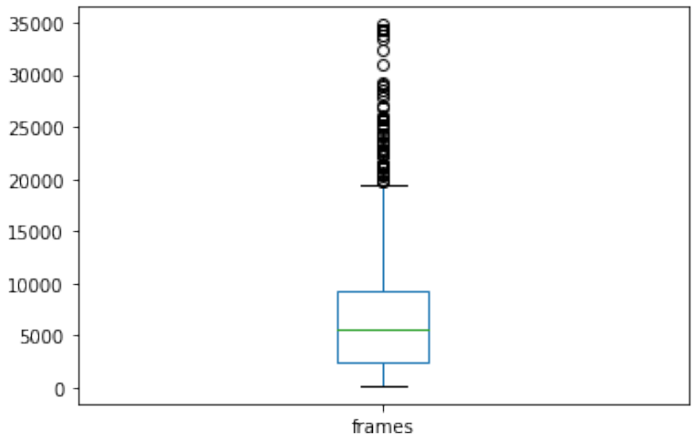
lower, upper = (-7805.5, 19454.5)
wartości >19454.5: 3076 (4.47%)
wartości >30k: 869 (1.26%)
wartości >35k: 20 (0.03%)



Zdecydowaliśmy się na usunięcie wierszy z wartościami frames > 35000

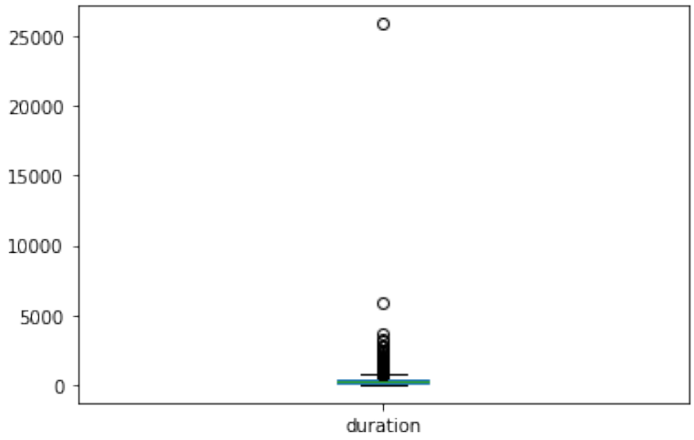
```
data_2 = data_2[data_2['frames'] <= 35000 ]
```

Po usunięciu:



4.0.2 Kolumna 'duration'

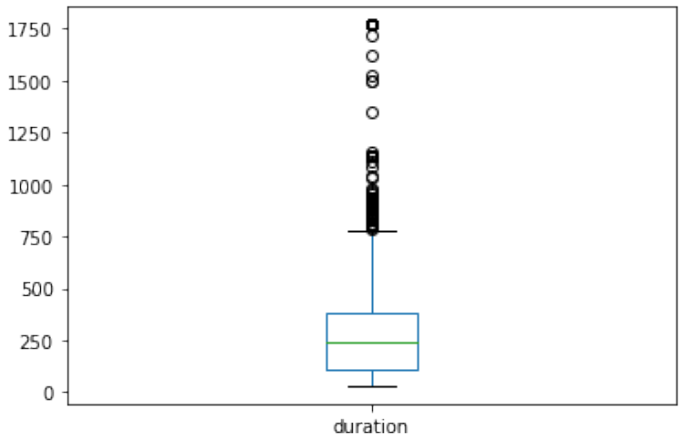
lower, upper = (-302.0675, 788.1525)
wartości >788: 3080 (4.48%)
wartości >1000: 1718 (2.5%)
wartości >1500: 865 (1.26%)
wartości >2000: 13 (0.02%)



Zdecydowaliśmy się na usunięcie wierszy z wartościami duration > 2000

```
data_2 = data_2[data_2['duration'] <= 2000 ]
```

Po usunięciu:



5 Feature engineering

5.1 Dodatkowe pole 'pixels'

```
data_3['pixels'] = data_3['width'] * data_3['height']
```

5.2 Dodatkowe pole 'o_pixels'

```
data_3['o_pixels'] = data_3['o_width'] * data_3['o_height']
```

6 Feature selection - usunięcie zbędnych kolumn

Zdecydowaliśmy usunąć następujące pola:

- pole id
- pole b_size (brak niezerowych wartości)
- pole umem (nie podlegająca analizie wartość wynikowa)

```
data_4 = data_4.drop(['id', 'b_size', 'umem'], axis=1)
```

7 Data Preparation - przygotowanie danych do uczenia

7.1 Kodowanie pól kategorycznych

```
categorical_cols = ['codec', 'o_codec']

ohe = ce.OneHotEncoder(cols=categorical_cols, return_df=True,
        use_cat_names=True, handle_unknown=0)
data_5 = ohe.fit_transform(data_5)
```

7.2 Podział danych na wejściowe i wynikowe

```
target = data_5['utime']
features = data_5.drop('utime', axis=1)
```

7.3 Skalowanie danych

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)
```

7.4 Podział danych na trenujące i testujące

```
X_train, X_test, y_train, y_test =
                train_test_split(
                        features_scaled,
                        target,
                        test_size=0.3,
                        random_state=0)
```

8 Trenowanie oraz ewaluacja modeli

Ewaluacja oraz porównanie modeli bazuje na błędzie średnio-kwadratowym oraz parametrze R2.

8.1 Podstawowy model regresji liniowej

```
lr = LinearRegression()
lr.fit(X_train, y_train)

print("Mean squared error of a linear model: %.2f" %
      mean_squared_error(y_test, lr.predict(X_test)))
score = lr.score(X_test, y_test) #r2_score
print("Linear Regression R2 score: %.2f" % score)
```

Wyniki:
Mean squared error of a linear model: 124.55
Linear Regression R2 score: 0.53

8.2 Wyszukiwanie optymalnego modelu z wykorzystaniem Polynominal Features, ElasticNet oraz GridSearch

```
parameters=[{
      'alpha':[0.1, 0.2, 0.5, 0.9],
      'l1_ratio':[0.1, 0.2, 0.5, 0.9],
      }
]

for pf_level in range(1,3):
    print(f"polynominal features level {pf_level}:")

    pf = PolynomialFeatures(pf_level)
    train_poly = pf.fit_transform(X_train)
    test_poly  = pf.fit_transform(X_test)

    model = GridSearchCV( ElasticNet(), parameters, cv=5 )
    model.fit(train_poly, y_train)

    print(f"  {model.best_estimator_}")
    print("    Mean squared error: %.2f" %
    mean_squared_error(y_test, model.predict(test_poly)))
    score = model.score(test_poly, y_test) #r2_score
    print("    R2 score: %.2f" % score)
```

Wyniki:

```
polynominal features level 1:
  ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.9,
          max_iter=1000, normalize=False, positive=False, precompute=False,
          random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
    Mean squared error: 127.97
    R2 score: 0.52
polynominal features level 2:
  ElasticNet(alpha=0.1, copy_X=True, fit_intercept=True, l1_ratio=0.9,
          max_iter=1000, normalize=False, positive=False, precompute=False,
          random_state=None, selection='cyclic', tol=0.0001, warm_start=False)
    Mean squared error: 57.47
    R2 score: 0.78
```

8.3 Podsumowanie

Przy użyciu cech wielomianowych oraz optymalizacji parametrów udało nam się znacząco zwiększyć dokładność modelu.