Politechnika Krakowska

Wydział Inżynierii Elektrycznej i Komputerowej

Komputerowe Wspomaganie Decyzji

Projekt zaliczeniowy
zespół nr 3

Ocena czasu potrzebnego na dekodowanie filmu

Aleksandra Bojeś 31i
Aleksandra Kuc 32i

Spis treści

1 Cel i zakres projektu

Celem projektu jest znalezienie najlepiej dopasowanego modelu do otrzymanych danych, służacego do oszacowania czasu potrzebnego na dekodowanie filmu z jednego formatu do innego przy dodatkowej zmianie rozdzielczości.

2 Źródło przetwarzanych danych

Dane wejściowe stanowi baza danych: *Online Video Characteristics and Transcoding Time Dataset Data Set.*
Dane zostały przygotowane przez Tewodros Deneke (tdeneke@abo.fi).

3 Wykorzystane biblioteki

Aby w prostszy sposób zarzadzać danymi, w projekcie wykorzystano nastepujace biblioteki:

- **Numpy** - jest to biblioteka dla jezyka Python, dodajaca obsługe dużych, wielowymiarowych tablic i macierzy, wraz z duża kolekcja wysokiej jakości funkcji matematycznych do pracy na tych tablicach. NumPy jest oprogramowaniem typu open-source.
- **Pandas** - jest open-source'owym narzedziem do analizy danych przeznaczonym dla Pythona. Udostepnia wszystkie niezbedne operacje potrzebne do pracy z modelem w uczeniu maszynowym.
- **Seaborn** - jest to biblioteka dla jezyka Python umożliwiajaca zaawansowane wizualizacje danych. Zapewnia narzedzia do tworzenia atrakcyjnych grafik statystycznych wysokiego poziomu.
- **Matplotlib** - jest to biblioteka do tworzenia wykresów dla jezyka Python i jego rozszerzenia numerycznego NumPy. Zawiera ona API "pylabźaprojektowane tak aby było jak najbardziej podobne do MATLABa, przez co jest łatwy do nauczenia przez jego użytkowników. Matplotlib został napisany i jest utrzymywany głównie przez Johna Huntera, i jest dostepny na licencji przypominajacej licencje BSD.

4 Opis danych wejściowych i ich analiza eksploracyjna

4.1 Pobranie danych

Wczytanie danych oraz wyświetlenie pierwszych 10 wierszy w tabeli:

```
#Pobieranie danych
mesurment_data = pd.read_csv('./transcoding_mesurment.tsv', delimiter='\t')
#Wyswietlanie wszystkich kolumn
pd.options.display.max_columns = None
mesurment_data.head(10)
```

4.2 Ogólne informacje o danych

```
#Iformacje o danych
mesurment_data.info()
```

Wynik:

```
RangeIndex: 68784 entries, 0 to 68783
Data columns (total 22 columns):
id              68784 non-null object
duration        68784 non-null float64
codec           68784 non-null object
width           68784 non-null int64
height          68784 non-null int64
bitrate         68784 non-null int64
framerate       68784 non-null float64
i               68784 non-null int64
p               68784 non-null int64
b               68784 non-null int64
frames          68784 non-null int64
i_size          68784 non-null int64
p_size          68784 non-null int64
b_size          68784 non-null int64
size            68784 non-null int64
o_codec         68784 non-null object
o_bitrate       68784 non-null int64
o_framerate     68784 non-null float64
o_width         68784 non-null int64
o_height        68784 non-null int64
umem            68784 non-null int64
utime           68784 non-null float64
dtypes: float64(4), int64(15), object(3)
```

Na podstawie pierwszej tabeli widać, że wczytane dane posiadaja trzy kolumny nienumeryczne: jest to kolumna *id*, *codec* oraz *o codec*.

```
#Wymiary tablicy
print(mesurment_data.shape)
```

Wynik:

```
(68784, 22)
```

```
#liczba wymiarów
mesurment_data.ndim
```

Wynik:

```
2
```

```
#Atrybuty
print(mesurment_data.columns.values)
```

```
['id' 'duration' 'codec' 'width' 'height' 'bitrate' 'framerate' 'i' 'p'
 'b' 'frames' 'i_size' 'p_size' 'b_size' 'size' 'o_codec' 'o_bitrate'
 'o_framerate' 'o_width' 'o_height' 'umem' 'utime']
```

Informacje o atrybutach:

- id = Youtube video id
- duration = duration of video
- bitrate bitrate(video) = video bitrate
- height = height of video in pixles
- width = width of video in pixles
- frame rate = actual video frame rate
- frame rate(est.) = estimated video frame rate
- codec = coding standard used for the video
- category = YouTube video category
- url = direct link to video (has expiration date)
- i = number of i frames in the video
- p = number of p frames in the video
- b = number of b frames in the video
- frames = number of frames in video

- i_size = total size in byte of i videos
- p_size = total size in byte of p videos
- b_size = total size in byte of b videos
- size = total size of video
- o_codec = output codec used for transcoding
- o_bitrate = output bitrate used for transcoding
- o_framerate = output framerate used for transcoding
- o_width = output width in pixel used for transcoding
- o_height = output height used in pixel for transcoding
- umem = total codec allocated memory for transcoding
- utime = total transcoding time for transcoding

5 Zmiany po przeprowadzeniu analizy

```
#sprawdzanie danych w codec i o_codec
print("Classes in data:", np.unique(mesurment_data['codec']))
print("Classes in data:", np.unique(mesurment_data['o_codec']))
```

Wynik:

```
Classes in data: ['flv' 'h264' 'mpeg4' 'vp8']
Classes in data: ['flv' 'h264' 'mpeg4' 'vp8']
```

Nalezy wiec rodzielić kolumny zawierajace dane tekstowe w taki sposób, by zawierały dane numeryczne. Można to osiagnać poprzez zamiane każdej z nich na 4 kolumny odpowiadajace poszczególnym wartościom.

```
#rozdzial kolumn
ohe = ce.OneHotEncoder(cols=['codec', 'o_codec'], return_df=True, use_cat_names
mesurment_data = ohe.fit_transform(mesurment_data)
```

W wyniku tych zmian zmienia nsie rozmiar tablicy danych:

```
#nowe wymairy tablicy
mesurment_data.shape
```

```
(68784, 28)
```

```
#Przeglad typ w
mesurment_data.dtypes
```

```
id                object
duration         float64
codec_mpeg4        int64
codec_h264         int64
codec_vp8          int64
codec_flv          int64
width              int64
height             int64
bitrate            int64
framerate        float64
i                  int64
p                  int64
b                  int64
frames             int64
i_size             int64
p_size             int64
b_size             int64
size               int64
o_codec_mpeg4      int64
o_codec_vp8        int64
o_codec_flv        int64
o_codec_h264       int64
o_bitrate          int64
o_framerate      float64
o_width            int64
o_height           int64
umem               int64
utime            float64
```

6 Podział danych

Podział danych na dane uczace oraz target - ostatnia kolumna w tabeli *utime* jest czasem dekodowania, który w naszym przypadku odpowiada targetowi.

```
data = mesurment_data.iloc[1:,1:-1]
target = mesurment_data.iloc[1:,-1]
```

Należy również przekształcić otrzymane zbiory danych na tablice w rozumieniu biblioteki Numpy.

```
data_ = np.array(data)
target_ = np.array(target)


print(data_.shape)
print(target_.shape)
```

```
(68783, 26)
(68783,)
```

6.1 Badanie korelacji danych

Wyświetlenie korelacji danych poprzez biblioteke Seaborn generuje wykres dużych rozmiarów, dlatego został on dołaczony osobno do projektu.

```
%matplotlib inline
sns.pairplot(data, diag_kind="kde")
```

7 Standaryzacja danych

Wyświetlenie danych:

```
target_
```

```
array([0.98  , 1.216, 1.692, ..., 0.752, 5.444, 3.076])
```

```
data_
```

```
array([[1.3035667e+02, 1.0000000e+00, 0.0000000e+00, ..., 3.2000000e+02,
        2.4000000e+02, 2.5164000e+04],
       [1.3035667e+02, 1.0000000e+00, 0.0000000e+00, ..., 4.8000000e+02,
        3.6000000e+02, 2.9228000e+04],
       [1.3035667e+02, 1.0000000e+00, 0.0000000e+00, ..., 6.4000000e+02,
        4.8000000e+02, 3.4316000e+04],
       ...,
       [2.4968000e+02, 0.0000000e+00, 0.0000000e+00, ..., 1.7600000e+02,
        1.4400000e+02, 8.8708000e+04],
       [1.8362334e+02, 0.0000000e+00, 1.0000000e+00, ..., 3.2000000e+02,
        2.4000000e+02, 8.8724000e+04],
       [2.9461334e+02, 1.0000000e+00, 0.0000000e+00, ..., 1.7600000e+02,
        1.4400000e+02, 8.8736000e+04]])
```

Wyświetlenie średniej danych

```
np.mean(data_, axis=0)
```

```
array([2.86416190e+02, 1.74621636e-01, 4.58616228e-01, 2.67318960e-01,
       9.94431764e-02, 6.24940698e+02, 4.12576131e+02, 6.93710792e+05,
       2.32414840e+01, 1.00869386e+02, 6.53176483e+03, 9.14798715e+00,
       6.64178220e+03, 2.83902704e+06, 2.21808798e+07, 0.00000000e+00,
       2.50232932e+07, 2.51370251e-01, 2.51181251e-01, 2.49116788e-01,
       2.48331710e-01, 1.39505542e+06, 2.11909953e+01, 8.02345463e+02,
       5.03830772e+02, 2.28227709e+05])
```

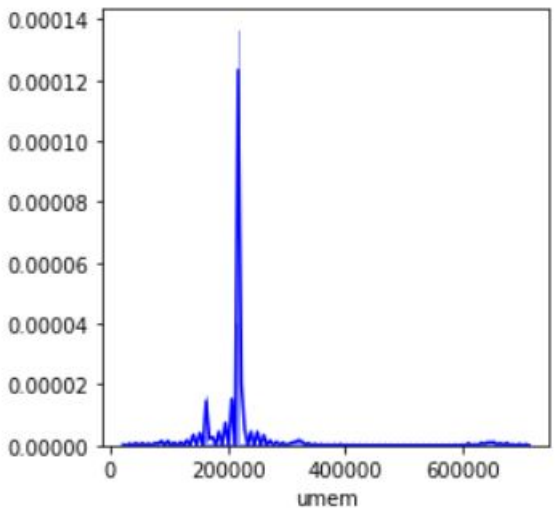
Wyświetlenie odchylenia standardowego dla każdej kolumny

```
np.std(data_, axis=0)
```

```
array([2.87257034e+02, 3.79643149e-01, 4.98284440e-01, 4.42560203e-01,
       2.99256129e-01, 4.63165906e+02, 2.40613293e+02, 1.09562484e+06,
       7.22472081e+00, 8.47643228e+01, 6.07584190e+03, 9.25161705e+01,
       6.15331199e+03, 4.32512366e+06, 5.09729963e+07, 0.00000000e+00,
       5.41439372e+07, 4.33800931e-01, 4.33692553e-01, 4.32501577e-01,
       4.32045220e-01, 1.74934406e+06, 6.66861071e+00, 6.09955122e+02,
       3.15967459e+02, 9.74277209e+04])
```

Rozkład danych przed przeprowadzeniem standaryzacji:

```
plt.figure(figsize=(4,4))
sns.distplot(data['umem'].dropna(),kde=True,bins=170, color='blue')
```



Z analizy danych wynika, iż należy przeprowadzić standaryzacje tak, aby rozkład danych miał średnia wartość równa 0 i odchylenie standardowe równe 1.

```
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data_)
k = scaler.transform(data)
```

Średnia kolumn po standaryzacji:

```
np.std(scaled_data, axis=0)
```

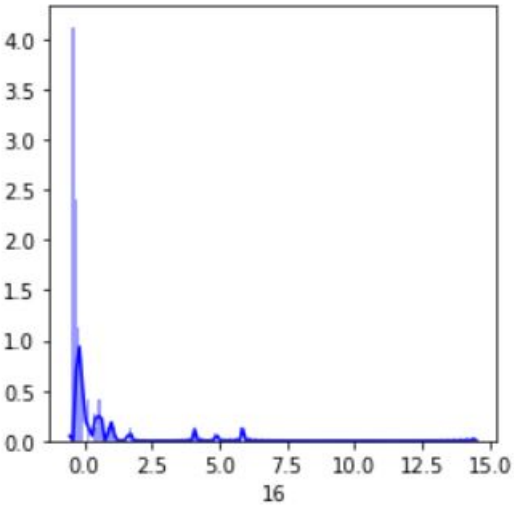
```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Po przeprowadzonej standaryzacji należy sprawdzić, czy zbiór danych jest tabela tak, jak rozumie to biblioteka Pandas.

```
kf = pd.DataFrame(k)
```

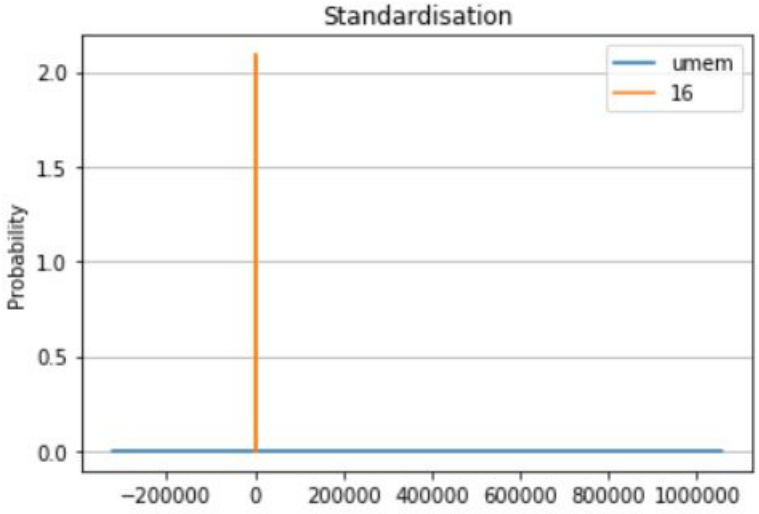
Rozkład danych po przeprowadzeniu standaryzacji:

```
plt.figure(figsize=(4,4))
sns.distplot(kf[16].dropna(),kde=True,bins=170, color='blue')
```



Porównanie danych:

```
fig, ax = plt.subplots()
data['umem'].plot.kde(ax=ax, legend=True, title='Standardisation')
kf[16].plot.kde(ax=ax, legend=True)
ax.set_ylabel('Probability')
ax.grid(axis='y')
```



8 Podział danych na zbiory uczacy oraz testowy

Domyślny podział zbiorów na zbiór danych uczacych oraz zbiór danych tekstowych

```
mesurment_train_data, mesurment_test_data, \
mesurment_train_target, mesurment_test_target = \
train_test_split(scaled_data,target_, test_size=0.1, random_state=101)
```

Informacje o zbiorze uczacym:

```
print("Training dataset:")
print("mesurment_train_data:", mesurment_train_data.shape)
print("mesurment_train_target:", mesurment_train_target.shape)
```

```
Training dataset:
mesurment_train_data: (61904, 26)
mesurment_train_target: (61904,)
```

Informacje o zbiorze testowym:

```
print("Testing dataset:")
print("mesurment_test_data:", mesurment_test_data.shape)
print("mesurment_test_target:", mesurment_test_target.shape)
```

```
Testing dataset:
mesurment_test_data: (6879, 26)
mesurment_test_target: (6879,)
```

9 Trenowanie modelu

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

9.1 Model regresji liniowej

```
# Model Regresji Liniowej
lr = LinearRegression(normalize=True)
lr.fit(mesurment_train_data, mesurment_train_target)

#Blad sredniokwadratowy
precission = mean_squared_error(mesurment_test_target,
    lr.predict(mesurment_test_data))
print("Mean squared error of a learned model: {0:0.2f}".format(precission))

#Trafnosc zbioru
score = lr.score(mesurment_test_data, mesurment_test_target)
print("Linear Regression variance score: %.2f" % score) #r2_score
```

Wynik:

```
Mean squared error of a learned model: 105.14
Linear Regression variance score: 0.65
```

Wykorzystanie krosswalidacji w celu ulepszenia modelu.

```
scores = cross_val_score(lr, scaled_data, target_, cv=4)
```

Sprawdzenie umiejetności predykcji wartości modelu dla wiersza o id = 4.

```
id=4
linear_regression_prediction = lr.predict
(mesurment_test_data[id,:].reshape(1,-1))

print("Model predicted for mesurment {0} value {1}"
    .format(id, linear_regression_prediction))

print("Real value for mesurment \"{0}\" is {1}"
    .format(id, mesurment_test_target[id]))
```

Wynik:

```
Model predicted for mesurment 4 value [11.47812925]
Real value for mesurment "4" is 12.433
```

9.2 Wykorzystanie regresji wielomianowej 2 stopnia

```
pt = PolynomialFeatures(2, interaction_only=True)
```

Transformacja zbioru uczacego

```
mesurment_train_poly = pt.fit_transform(mesurment_train_data)
mesurment_train_poly.shape
```

```
(61904, 352)
```

Transformacja zbioru testujacego

```
mesurment_test_poly = pt.fit_transform(mesurment_test_data)
mesurment_test_poly.shape
```

```
(6879, 352)
```

Trenowanie danych z regresja wielomianowa

```
lri = LinearRegression(normalize=True)
lri.fit(mesurment_train_poly, mesurment_train_target)
```

Blad średniokwadratowy i trafność przewidywanych wyników dla regresji wielomianowej.

```
#B ad sredniokwadratowy
print("Mean squared error of a linear moderl using
polynomial features: %.2f" % mean_squared_error
    (mesurment_test_target, lri.predict(mesurment_test_poly)))

#Trafno
score = lri.score(mesurment_test_poly, mesurment_test_target) #r2_score
print("Linear Regression variance score using
    polynomial features: %.2f" % score)
```

Wynik:

```
Mean squared error of a linear moderl using polynomial features: 83.82
Linear Regression variance score using polynomial features: 0.72
```

Sprawdzenie przewidywanego wyniku dla wiersza o id = 4

```
id=4
lrp = lri.predict(mesurment_test_poly[id,:].reshape(1,-1))
```

```
#Przewidywanie
print("Model predicted for mesurment {0} value {1}".format(id, lrp))

#Aktualna wartosc
print("Real value for mesurment \"{0}\" is {1}"
    .format(id, mesurment_test_target[id]))
```

Wynik:

```
Model predicted for mesurment 4 value [8.18212891]
Real value for mesurment "4" is 12.433
```

10 Wykorzystanie regresji Lasso

Przygotowanie modelu:

```
lasso_regression = Lasso(alpha=0.05)
lasso_regression.fit(mesurment_train_poly, mesurment_train_target)
```

Sprawdzenie działania tak wytrenowanego modelu

```
#B d redniokwadratowy
print("Mean squared error of a linear moderl using
    polynomial features: %.2f" % mean_squared_error
    (mesurment_test_target, lasso_regression.predict(mesurment_test_poly)))

#Trafno
print("Lasso regression variance score: %.2f" % score)
```

Otrzymany wynik:

```
Mean squared error of a linear moderl using polynomial features: 37.37
Lasso regression variance score: 0.88
```

Sprawdzenie przewidywanego wyniku dla wiersza o id = 4

```
id=4
lss = lasso_regression.predict(mesurment_test_poly[id,:].reshape(1,-1))

#Przewidywanie
print("Model predicted for mesurment {0} value {1}".format(id, lss))

#Aktualna wartosc
print("Real value for mesurment \"{0}\" is {1}"
      .format(id, mesurment_test_target[id]))
```

Wynik:

```
Model predicted for mesurment 4 value [10.47109003]
Real value for mesurment "4" is 12.433
```

11 Kod źródłowy

Pełny kod źródłowy dostepny pod adresem: https://github.com/BAleksandra/KWD-2019-2020-Projekt-

12 Podsumowanie

Analizujac dany zbiór danych i dobierajac odpowiednie metody udało nam sie uzyskać zadowalajacy wynik dla naszego modelu. Najlepszy okazała sie model regresji lasso z użyciem cech wielomianowych (stopnia drugiego).