Akademia Ekonomiczno-Humanistyczna w Warszawie

SPRAWOZDANIE

INTELIGENTNA ANALIZA DANYCH

LAB3
ANALIZA TEKSTÓW,
PRZETRENOWANIE I OPTYMALIZACJA MODELU

28.11.2021

JOANNA PRAJZENDANC
36358

MIŁOSZ SAKOWSKI
36381

Spis treści

1. Cel i przebieg ćwiczenia

Celem ćwiczenia było utrwalenie wiedzy w zakresie optymalizacji modelu (grid search), zapoznanie się z pojęciem przetrenowania modelu oraz sposobów analizy tekstów na podstawie gotowych wiadomości e-mail.

2. Definicje i założenia

2.1. Wyjaśnienie pojęć

W sprawozdaniu pojawiają się następujące pojęcia:

- ✧ grid search - sposób optymalizacji modelu polegający na zestawieniu ze sobą wszystkich możliwych konfiguracji zadanych parametrów
- ✧ przetrenowanie - stan modelu, który mimo idealnego dopasowania dla danych ze zbioru trenującego dopasowanie do danych ze zbioru testującego jest dużo niższe i niewystarczające

3. Grid search

3.1. Zadanie #1

i. Treść polecenia

Proszę pobrać dowolny zbiór danych ze strony https://archive.ics.uci.edu/ml/index.php

Następnie proszę zoptymalizować las metodą Grid Search (trzy parametry modelu mają zostać optymalizowane).

Jakie wartości parametrów są najlepsze i dlaczego?

ii. Rozwiązanie

A. Zbiór danych

```
import pandas as pd
```

```
fires_origin = pd.read_csv(
    "forestfires.csv", header=0, index_col=False)
d = {
    'X': fires_origin['X'].values,
    'Y': fires_origin['Y'].values,
    'temp': fires_origin['temp'].values,
    'RH': fires_origin['RH'].values,
    'wind': fires_origin['wind'].values,
    'rain': fires_origin['rain'].values,
    'area': fires_origin['area'].values
}
fires = pd.DataFrame(d)
print(fires.head())
```

```
   X  Y  temp  RH  wind  rain  area
0  7  5   8.2  51   6.7   0.0   0.0
1  7  4  18.0  33   0.9   0.0   0.0
2  7  4  14.6  33   1.3   0.0   0.0
3  8  6   8.3  97   4.0   0.2   0.0
4  8  6  11.4  99   1.8   0.0   0.0
```

Obraz 1: Ładowanie bazy danych z pliku

```
import statistics as stat
maximum = max(fires['temp'])
minimum = min(fires['temp'])
median = stat.median(fires['temp'])
diff = round((maximum - median)/4)
max_RH = max(fires['RH'])
min_RH = min(fires['RH'])
median_RH = stat.median(fires['RH'])
max_wind= max(fires['wind'])
min_wind= min(fires['wind'])
median_wind= stat.median(fires['wind'])
```

```
tp = fires['temp']
rh = fires['RH']
w = fires['wind']
serious = fires['temp'].copy()
for i, val in enumerate(fires['temp']):
    #print(val, rh[i], serious[i])
    if (val >= (median + diff) and rh[i] <= 30 and w[i] >= 3.0):
        serious.loc[i] = 4
    else:
        if (val >= median + diff):
            serious.loc[i] = 3
        else:
            if (val >= (median - diff) and rh[i] <= 50 and w[i] >= 4.0):
                serious.loc[i] = 3
            else:
                if (val >= median - 2*diff and rh[i] <= 50 and w[i] >= 5.0):
                    serious.loc[i] = 2
                else:
                    if (val >= median - 2*diff):
                        serious.loc[i] = 1
                    else:
                        serious.loc[i] = 0
```

Obraz 2: Algorytm klasyfikujący

```
x = pd.DataFrame(
    fires.values[:,0:11],
    columns =['X','Y','temp', 'RH', 'wind', 'rain', 'area']
)
y = pd.DataFrame(serious.values, columns =['SERIOUS'])
```

```
print(x.shape)
print(y.shape)
```

```
(517, 7)
(517, 1)
```

```
print(x[1:5])
print(y[1:5])
```

```
     X    Y  temp    RH  wind  rain  area
1  7.0  4.0  18.0  33.0   0.9   0.0   0.0
2  7.0  4.0  14.6  33.0   1.3   0.0   0.0
3  8.0  6.0   8.3  97.0   4.0   0.2   0.0
4  8.0  6.0  11.4  99.0   1.8   0.0   0.0
   SERIOUS
1      1.0
2      1.0
3      0.0
4      0.0
```

Obraz 3: Docelowy zbiór danych

B. Optymalizacja 1 parametru

```
# Grid Search - jedna zmienna
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
max_depth = np.array(
    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25])
model = RandomForestClassifier(n_estimators=2)
grid = GridSearchCV(estimator=model, param_grid=dict(max_depth=max_depth))
grid.fit(x, y.values.ravel())
```

```
GridSearchCV(estimator=RandomForestClassifier(n_estimators=2),
             param_grid={'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25])})
```

```
print(grid)
```

```
GridSearchCV(estimator=RandomForestClassifier(n_estimators=2),
             param_grid={'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,
8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25])})
```

```
print(grid.cv_results_['params'])
```

```
[{'max_depth': 1}, {'max_depth': 2}, {'max_depth': 3}, {'max_depth': 4},
{'max_depth': 5}, {'max_depth': 6}, {'max_depth': 7}, {'max_depth': 8},
{'max_depth': 9}, {'max_depth': 10}, {'max_depth': 11}, {'max_depth': 12},
{'max_depth': 13}, {'max_depth': 14}, {'max_depth': 15}, {'max_depth': 1
6}, {'max_depth': 17}, {'max_depth': 18}, {'max_depth': 19}, {'max_depth':
20}, {'max_depth': 21}, {'max_depth': 22}, {'max_depth': 23}, {'max_dept
h': 24}, {'max_depth': 25}]
```

```
print(grid.cv_results_['mean_test_score'])
```

```
[0.57847274 0.65408887 0.71544063 0.79878641 0.85287528 0.81596341
 0.80477969 0.74665795 0.81643017 0.86273338 0.8626587  0.80070948
 0.83956311 0.85489171 0.80481703 0.84348394 0.83584765 0.79137416
 0.81803585 0.84150485 0.80830844 0.81435773 0.83943241 0.84156087
 0.8664115 ]
```

Obraz 4: Optymalizacja modelu - maksymalna głębokość drzew decyzyjnych

```
print(grid.best_params_)
```

```
{'max_depth': 7}
```

Obraz 5: Wynik optymalizacji modelu dla maksymalnej głębokości drzew decyzyjnych - najlepsze rozwiązanie

C. Optymalizacja 2 parametrów

```
# Grid Search - dwie zmienne
max_depth = np.array(
    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25])
n_estimators = np.array([2,3,4,5,6,7,8,9,10])
model = RandomForestClassifier()
grid = GridSearchCV(estimator=model,
                    param_grid=dict(
                        n_estimators=n_estimators,
                        max_depth=max_depth))
grid.fit(x, y.values.ravel())
print(grid.cv_results_['params'])
```

```
[{'max_depth': 1, 'n_estimators': 2}, {'max_depth': 1, 'n_estimators':
3}, {'max_depth': 1, 'n_estimators': 4}, {'max_depth': 1, 'n_estimator
s': 5}, {'max_depth': 1, 'n_estimators': 6}, {'max_depth': 1, 'n_estimat
ors': 7}, {'max_depth': 1, 'n_estimators': 8}, {'max_depth': 1, 'n_estim
ators': 9}, {'max_depth': 1, 'n_estimators': 10}, {'max_depth': 2, 'n_es
timators': 2}, {'max_depth': 2, 'n_estimators': 3}, {'max_depth': 2, 'n_
estimators': 4}, {'max_depth': 2, 'n_estimators': 5}, {'max_depth': 2,
'n_estimators': 6}, {'max_depth': 2, 'n_estimators': 7}, {'max_depth':
2, 'n_estimators': 8}, {'max_depth': 2, 'n_estimators': 9}, {'max_dept
h': 2, 'n_estimators': 10}, {'max_depth': 3, 'n_estimators': 2}, {'max_d
epth': 3, 'n_estimators': 3}, {'max_depth': 3, 'n_estimators': 4}, {'max
_depth': 3, 'n_estimators': 5}, {'max_depth': 3, 'n_estimators': 6}, {'m
ax_depth': 3, 'n_estimators': 7}, {'max_depth': 3, 'n_estimators': 8},
{'max_depth': 3, 'n_estimators': 9}, {'max_depth': 3, 'n_estimators': 1
0}, {'max_depth': 4, 'n_estimators': 2}, {'max_depth': 4, 'n_estimator
s': 3}, {'max_depth': 4, 'n_estimators': 4}, {'max_depth': 4, 'n_estimat
ors': 5}, {'max_depth': 4, 'n_estimators': 6}, {'max_depth': 4, 'n_estim
ators': 7}, {'max_depth': 4, 'n_estimators': 8}, {'max_depth': 4, 'n_est
imators': 9}, {'max_depth': 4, 'n_estimators': 10}, {'max_depth': 5, 'n_
```

Obraz 6: Optymalizacja modelu - maksymalna głębokość drzew decyzyjnych oraz ilość drzew decyzyjnych

```
print(grid.cv_results_['mean_test_score'])
```

```
[0.51460045 0.57645631 0.60541449 0.61712099 0.60944735 0.56092233
 0.59002987 0.61314414 0.61486184 0.64826363 0.74902913 0.71581404
 0.70617999 0.71603809 0.71796117 0.73144137 0.75614264 0.75828977
 0.79107543 0.74673264 0.77976102 0.77968633 0.7448469  0.82593353
 0.78541822 0.82203137 0.85309933 0.76773712 0.83370052 0.85870052
 0.8317401  0.87050037 0.85888723 0.87038835 0.86663555 0.87236744
 0.81239731 0.8801531  0.85517177 0.85306199 0.87442121 0.90147498
 0.87821135 0.89759149 0.89361464 0.80660941 0.8742345  0.845295
 0.89929052 0.90920463 0.89951456 0.89171023 0.91880134 0.92651232
 0.80644137 0.88024645 0.9360717  0.88769604 0.90918596 0.94199029
 0.94191561 0.93037715 0.93226288 0.79899178 0.87238611 0.89368932
 0.89578043 0.92649365 0.89367065 0.9400112  0.91693428 0.94008588
 0.81794249 0.89557506 0.88986184 0.9150112  0.89951456 0.94385736
 0.94970127 0.94777819 0.93610904 0.81446975 0.88795743 0.89557506
 0.91680358 0.9360717  0.92653099 0.93610904 0.93416729 0.94195295
 0.78920836 0.88190814 0.90517177 0.90713219 0.91497386 0.94002987
 0.92845407 0.93620239 0.95356609 0.84144884 0.85875653 0.90522778
 0.90892457 0.91284541 0.93616505 0.92078043 0.95367812 0.93612771
 0.8084578  0.85677745 0.84133682 0.93806945 0.93424197 0.93616505
 0.9457991  0.94383869 0.95356609 0.83179612 0.88579164 0.89540702
 0.90918596 0.91299477 0.95946602 0.95351008 0.94398805 0.95746826
 0.79292382 0.8800224  0.89156087 0.93030246 0.92447722 0.94012323
 0.92266617 0.94585512 0.9438947  0.8084578  0.88205751 0.8760829
 0.94006721 0.89544436 0.93231889 0.94789022 0.92464526 0.91680358
 0.81430172 0.85694548 0.86077296 0.93237491 0.89927184 0.93612771
 0.92638163 0.94578043 0.92656833 0.81053025 0.89966393 0.8877894
 0.92259149 0.89363331 0.91870799 0.92837939 0.9284354  0.92845407
 0.8220127  0.85886856 0.89540702 0.9245519  0.89749813 0.91695295
 0.93241225 0.94973861 0.93427931 0.81988424 0.88579164 0.90332338
 0.92462659 0.9129761  0.92261016 0.93424197 0.92470127 0.94583645
 0.78711725 0.90328603 0.87050037 0.90909261 0.91682226 0.92649365
 0.94395071 0.93039582 0.93995519 0.82397311 0.88581031 0.88398058
 0.88013443 0.93032114 0.95942868 0.94975728 0.9381068  0.94970127
 0.85498506 0.88980583 0.87815534 0.92064974 0.92063107 0.94581777
 0.91491785 0.93805078 0.94973861 0.81390963 0.89725541 0.87823002
 0.91874533 0.91874533 0.93233757 0.93814414 0.95933532 0.94970127
 0.87247946 0.88588499 0.87408514 0.92261016 0.93041449 0.9284354
 0.91880134 0.93816281 0.94193428]
```

Obraz 7: Wyświetlenie średnich dopasowań modelu dla wszystkich kombinacji

```
print(grid.best_params_)
```

```
{'max_depth': 25, 'n_estimators': 9}
```

Obraz 8: Wynik optymalizacji modelu dla maksymalnej głębokości drzew decyzyjnych oraz ilości drzew - najlepsze rozwiązanie

D. Optymalizacja 3 parametrów

```
criterion = np.array(['gini','entropy'])
max_depth = np.array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,
n_estimators = np.array([2,3,4,5,6,7,8,9,10])
model = RandomForestClassifier()
grid = GridSearchCV(estimator=model, param_grid=dict(n_estimators=n_estimato
grid.fit(x, y.values.ravel())
print(grid.cv_results_['params'])
```

```
[{'criterion': 'gini', 'max_depth': 1, 'n_estimators': 2}, {'criterion':
'gini', 'max_depth': 1, 'n_estimators': 3}, {'criterion': 'gini', 'max_d
epth': 1, 'n_estimators': 4}, {'criterion': 'gini', 'max_depth': 1, 'n_e
stimators': 5}, {'criterion': 'gini', 'max_depth': 1, 'n_estimators':
6}, {'criterion': 'gini', 'max_depth': 1, 'n_estimators': 7}, {'criterio
n': 'gini', 'max_depth': 1, 'n_estimators': 8}, {'criterion': 'gini', 'm
ax_depth': 1, 'n_estimators': 9}, {'criterion': 'gini', 'max_depth': 1,
'n_estimators': 10}, {'criterion': 'gini', 'max_depth': 2, 'n_estimator
s': 2}, {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 3}, {'crit
erion': 'gini', 'max_depth': 2, 'n_estimators': 4}, {'criterion': 'gin
i', 'max_depth': 2, 'n_estimators': 5}, {'criterion': 'gini', 'max_dept
h': 2, 'n_estimators': 6}, {'criterion': 'gini', 'max_depth': 2, 'n_esti
mators': 7}, {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 8},
{'criterion': 'gini', 'max_depth': 2, 'n_estimators': 9}, {'criterion':
'gini', 'max_depth': 2, 'n_estimators': 10}, {'criterion': 'gini', 'max_
depth': 3, 'n_estimators': 2}, {'criterion': 'gini', 'max_depth': 3, 'n_
estimators': 3}, {'criterion': 'gini', 'max_depth': 3, 'n_estimators':
4}, {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 5}, {'criterio
n': 'gini', 'max_depth': 3, 'n_estimators': 6}, {'criterion': 'gini', 'm
```

Obraz 9: Optymalizacja modelu - maksymalna głębokość drzew decyzyjnych, ilość drzew oraz sposób sprawdzania dopasowania podczas podziału w drzewach decyzyjnych

```
print(grid.cv_results_['mean_test_score'])
```

```
[0.51051158 0.57615758 0.56282674 0.54570575 0.5627894  0.63030246
 0.63840553 0.51039955 0.66896938 0.68246826 0.6496826  0.71751307
 0.70982076 0.75423824 0.77957431 0.75218447 0.75054145 0.75263256
 0.72522405 0.77561613 0.76023152 0.80261389 0.77959298 0.81250934
 0.77397311 0.8471994  0.84729276 0.70026139 0.70789768 0.84914115
 0.78732263 0.83762136 0.85494772 0.83967513 0.86848394 0.83384989
 0.82025765 0.84144884 0.8568708  0.86269604 0.88203883 0.83956311
 0.89930919 0.84919716 0.88394324 0.84917849 0.86842793 0.91103435
 0.87410381 0.93420463 0.87826736 0.89749813 0.91499253 0.9071882
 0.79301718 0.82796863 0.88000373 0.90321135 0.91872666 0.91290142
 0.92651232 0.93614638 0.94971994 0.8434466  0.8993652  0.89352128
 0.92262883 0.90905527 0.93037715 0.92645631 0.93622106 0.9554705
 0.85089619 0.91114638 0.91110904 0.91678491 0.91295743 0.93220687
 0.94004854 0.94002987 0.9575056  0.85868185 0.85110157 0.87414115
 0.91687827 0.93808813 0.94583645 0.94002987 0.94393204 0.94779686
 0.82410381 0.86454444 0.90134429 0.91680358 0.93614638 0.93045183
 0.93228155 0.9690814  0.94193428 0.82191934 0.88033981 0.86842793
 0.90143764 0.94006721 0.94578043 0.93618372 0.959354   0.94970127
 0.82380508 0.88984317 0.90537715 0.89759149 0.92649365 0.92852875
 0.93805078 0.9420463  0.94576176 0.81803585 0.88202016 0.88401792
 0.93416729 0.918764   0.92839806 0.92458925 0.94970127 0.95746826
 0.80683346 0.92063107 0.89361464 0.89559373 0.92266617 0.93806945
 0.9342233  0.93032114 0.94772218 0.81209858 0.88771471 0.91103435
 0.9092233  0.91480583 0.93039582 0.94970127 0.96325616 0.94960792
 0.81992158 0.84746079 0.90911128 0.9110717  0.93224421 0.94008588
 0.9342233  0.94195295 0.92457058 0.79292382 0.86476848 0.90319268
 0.9264003  0.90511576 0.92656833 0.95160568 0.9304705  0.94383869
 0.7928118  0.84113144 0.89932786 0.89759149 0.93230022 0.94010456
 0.92063107 0.92852875 0.93424197 0.81629948 0.88790142 0.87440254
 0.93228155 0.93612771 0.93032114 0.95356609 0.9438947  0.94583645
 0.82007095 0.89372666 0.84551904 0.92462659 0.90916729 0.93228155
 0.93808813 0.93801344 0.95554518 0.8279873  0.86648618 0.89940254
 0.93808813 0.9148245  0.94391337 0.91112771 0.918764   0.94589246
 0.79699403 0.89176624 0.86648618 0.93429798 0.88784541 0.94387603
 0.94975728 0.93426064 0.94010456 0.84133682 0.8510829  0.91491785
 0.92068708 0.91863331 0.93418596 0.95162435 0.95162435 0.95168036
 0.75817774 0.88030246 0.90716953 0.93041449 0.88793876 0.94975728
 0.93237491 0.94002987 0.94583645 0.61699029 0.52793129 0.61292009
 0.51258402 0.49128081 0.62677371 0.60365945 0.62229276 0.59182226
 0.60522778 0.74477222 0.69258775 0.67681105 0.74662061 0.79693801
 0.7543876  0.74116878 0.72931292 0.69212099 0.78160941 0.73528753
 0.81060493 0.78943241 0.78933906 0.79335325 0.8067401  0.81801718
 0.81629948 0.8026699  0.82214339 0.84927184 0.80072816 0.83771471
 0.87628827 0.87625093 0.8568708  0.76585138 0.85113891 0.87040702
 0.87242345 0.88592233 0.90334205 0.89171023 0.88586632 0.90726288
 0.87038835 0.87050037 0.8935773  0.89749813 0.91103435 0.92645631
 0.92460792 0.93806945 0.92651232 0.77175131 0.80263256 0.91084765
 0.91499253 0.90916729 0.9400112  0.90713219 0.93424197 0.92449589

 0.86256535 0.85694548 0.91689694 0.89962659 0.91874533 0.94975728
 0.92847274 0.91303211 0.95168036 0.82010829 0.87427184 0.89350261
 0.92453323 0.95354742 0.93033981 0.93241225 0.93230022 0.9438947
 0.85113891 0.85866318 0.89938387 0.89382001 0.91301344 0.91687827
 0.93235624 0.93614638 0.93427931 0.86667289 0.88786408 0.90705751
 0.92053771 0.93032114 0.92068708 0.91105302 0.96321882 0.93239358
 0.79895444 0.8760829  0.88995519 0.92249813 0.91502987 0.93814414
 0.9304705  0.95556385 0.95349141 0.80274459 0.86644884 0.91105302
 0.91676624 0.91689694 0.93808813 0.92457058 0.94591113 0.94774085
 0.77180732 0.86458178 0.89945855 0.91678491 0.9321882  0.9246826
 0.93424197 0.92836072 0.93618372 0.75659074 0.81213592 0.89930919
 0.90522778 0.8993652  0.93411128 0.92849141 0.94975728 0.94779686
 0.82408514 0.88991785 0.89353996 0.92654966 0.93622106 0.94774085
 0.96516057 0.95554518 0.9420463  0.80825243 0.86450709 0.92451456
 0.90323002 0.90722554 0.93030246 0.93033981 0.94383869 0.93048917
 0.83948842 0.85877521 0.89957058 0.90532114 0.91308813 0.93995519
 0.92274085 0.92656833 0.9400112  0.85317401 0.87055639 0.87432786
 0.90726288 0.93235624 0.91674757 0.93801344 0.92845407 0.97100448
 0.81615011 0.89171023 0.89563107 0.93414862 0.93623973 0.90528379
 0.91891337 0.93022778 0.93420463 0.81224795 0.87630695 0.90715086
 0.91870799 0.94585512 0.93812547 0.95941001 0.96129574 0.92849141
 0.80646004 0.88775205 0.90330471 0.93220687 0.91880134 0.93039582
 0.9245519  0.93426064 0.94973861 0.76773712 0.86256535 0.84133682
 0.90530246 0.91684093 0.95941001 0.91878267 0.9420463  0.9438947
 0.79889843 0.87229276 0.9071882  0.92654966 0.92262883 0.95164302
 0.91489918 0.93226288 0.92649365 0.76971621 0.86652353 0.85679612
 0.92285288 0.92262883 0.9092233  0.93043316 0.93424197 0.92841673]
```

Obraz 10: Średnie wyniki dla wszystkich kombinacji podczas optymalizacji modelu

```
print(grid.best_params_)
```

```
{'criterion': 'entropy', 'max_depth': 20, 'n_estimators': 9}
```

Obraz 11: Wynik optymalizacji modelu dla maksymalnej głębokość drzew decyzyjnych, ilości drzew oraz sposobu sprawdzania dopasowania podczas podziału w drzewach decyzyjnych - najlepsze rozwiązanie

E. Najlepsze parametry dla wybranego modelu - zestawienie wyników optymalizacji modelu

Tabela 1: Zestawienie najlepszych parametrów w zależności od rodzaju optymalizacji modelu

Parametry optymalizacji	**Najelpszy wynik: max_depth**	**Najelpszy wynik: n_estimators**	**Najlepszy wynik: criterion**
max_depth	7	*default (2)*	*default („gini”)*
max_depth, *n_estimators*	25	*9*	*default („gini”)*
max_depth, *n_estimators*, *criterion*	20	*9*	*„entropy”*

Dla najprostszego modelu wystarczy las drzew decyzyjnych z dwoma drzewami o maksymalnej głębokości równej 7. Dla najbardziej dokładnego modelu potrzeba las drzew decyzyjnych z 9 drzewami o głębokości maksymalnej równej 20 oraz z kryterium dopasowywania ustawionym na „entropy”.

4. Przetrenowanie

4.1. Zadanie #2

i. Treść polecenia

Proszę zaprezentować zjawisko overfitingu na klasyfikatorze random forest.

ii. Rozwiązanie

A. Ładowanie bazy danych i bibliotek

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
```

```
data = pd.read_csv("glass.data", header=None, index_col=0)
print(data.head())
x = pd.DataFrame(data.values[:,0:9], columns =[
    'RI','Na','Mg','Al','Si','K','Ca','Ba','Fe'])
y = pd.DataFrame(data.values[:,9], columns =['Type'])
```

```
        1      2     3     4      5     6     7    8    9   10
0
1  1.52101  13.64  4.49  1.10  71.78  0.06  8.75  0.0  0.0   1
2  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0   1
3  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0   1
4  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0   1
5  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0   1
```

```
print(x[1:5])
print(y[1:5])
```

```
        RI     Na    Mg    Al     Si     K    Ca   Ba   Fe
1  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0
2  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0
3  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0
4  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0
   Type
1   1.0
2   1.0
3   1.0
4   1.0
```

```
print(x.shape)
print(y.shape)
```

```
(214, 9)
(214, 1)
```

```
print(x[1:5])
print(y[1:5])
```

```
        RI     Na    Mg    Al     Si     K    Ca   Ba   Fe
1  1.51761  13.89  3.60  1.36  72.73  0.48  7.83  0.0  0.0
2  1.51618  13.53  3.55  1.54  72.99  0.39  7.78  0.0  0.0
3  1.51766  13.21  3.69  1.29  72.61  0.57  8.22  0.0  0.0
4  1.51742  13.27  3.62  1.24  73.08  0.55  8.07  0.0  0.0
   Type
1   1.0
2   1.0
3   1.0
4   1.0
```

Obraz 12: Ładowanie bazy danych z pliku

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
clf = RandomForestClassifier()
```

Obraz 13: Dodanie bibliotek do trenowania modelu i przypisanie klasyfikatora drzewa losowego do zmiennej

B. Duże przetrenowanie

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.9, random_state=42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(21, 9)
(21, 1)
(193, 9)
(193, 1)
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.5751295336787565
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.5544041450777202
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.6010362694300518
```

Obraz 14: Trenowanie modelu na bardzo małym zbiorze trenującym (10%) oraz wyniki dokładności modelu - występuje duże przetrenowanie modelu

C. Małe przetrenowanie

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.1, random_state=42)
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(192, 9)
(192, 1)
(22, 9)
(22, 1)
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.9545454545454546
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.9545454545454546
```

```
clf = clf.fit(x_train, y_train.values.ravel())
y_train_pred_from_clf = clf.predict(x_train)
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_train, y_train_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
1.0
0.8636363636363636
```

Obraz 15: Trenowanie modelu na dużym zbiorze trenującym (90%) i wyniki dokładności - występuje małe przetrenowanie modelu

iii. Wnioski

Bardzo dokładne trenowanie modelu nie jest wskazane, ponieważ może prowadzić do przetrenowania - model będzie się idealnie sprawdzał na danych ze zbioru trenującego, jednak będzie niewystarczająco skuteczny na danych ze zbioru testującego. Lepszym rozwiązaniem jest mniejsza dokładność na modelu trenującym, aby dokładność na danych testujących była zbliżona. Przetrenowaniu można zapobiec zwiększając zbiór trenujący oraz zmniejszając rygorystyczność warunków podczas trenowania modelu.

5. Analiza tekstu

5.1. Zadanie #3

i. Treść polecenia

Proszę powtórzyć ćwiczenia dla innego podzbioru kategorii, np. rec.autos, rec.motorcycles, rec.sport.baseball, rec.sport.hockey.

ii. Rozwiązanie

```
# http://qwone.com/~jason/20Newsgroups/
```

```
#1. Prosze powtorzyc cwiczenia dla innego podzbioru kategorii
```

```
#wczytanie zbioru trenujacego danych tekstowych
from sklearn.datasets import fetch_20newsgroups
categories = [
    'sci.crypt',
    'rec.sport.baseball',
    'rec.motorcycles',
    'sci.space']
train_dataset = fetch_20newsgroups(
    subset='train',
    categories=categories,
    shuffle=True,
    random_state=42
)
test_dataset = fetch_20newsgroups(
    subset='test',
    categories=categories,
    shuffle=True,
    random_state=42
)
```

```
#klasy
print(train_dataset.target_names)
print(test_dataset.target_names)
```

```
['rec.motorcycles', 'rec.sport.baseball', 'sci.crypt', 'sci.space']
['rec.motorcycles', 'rec.sport.baseball', 'sci.crypt', 'sci.space']
```

```
#liczba wiadomosci
print(len(train_dataset.data))
print(len(test_dataset.data))
```

```
2383
1585
```

Obraz 16: Załadowanie tekstów wiadomości e-mail

```
#wyswietlenie wiadomosci ze zbioru trenujacego o indeksie 180
print(train_dataset.data[180])
```

Obraz 17: Wyświetlenie wiadomości o indeksie 180

From: ajjb@adam4.bnsc.rl.ac.uk (Andrew Broderick)
Subject: Re: Solar Sail Data
Keywords: Solar Sail
Organization: Rutherford Appleton Lab, UK
Lines: 79

In article <1993Apr15.051746.29848@news.duc.auburn.edu> snydefj@eng.auburn.edu writes:
>
>I am looking for any information concerning projects involving Solar
> Sails

I was at an interesting seminar at work (UK's R.A.L. Space Science Dept.) on this subject, specifically on a small-scale Solar Sail proposed as a student space project. The guy giving the talk was keen to generate interest in the project. I'll typein the handout he gave out at the meeting. Here goes :

The Microlight Solar Sail
---

1. Introduction
The solar sail is a well-established concept. Harnessing the pressure of sunlight, a spacecraft would have unlimited range. In principle, such a vehicle could explore the whole Solar System with zero fuel consumption.

However it is more difficult to design a practical solar sail than most people realize. The pressure of sunlight is only about one kilogram per square kilometer. Deploying and controlling the large area of aluminized fabric which would be necessary to transport a 'conventional' type spacecraft is a daunting task. This is why, despite the potential of hte idea, no such craft has actually been launched to date.

2.Design
Recent advances in microelectronics make possible a different concept: a tiny sail just a few metres in diameter which could be controlled purely be electronics, with no mechanical parts. Several attitude control methods are feasible: for example the pressure sunlight exerts on a panel of solar cells varies according to whether power is being drawn.

The key components of the craft will be a minute CCD camera developed at Edinburgh University which can act as both attitude sensor and data gathering device; solar cells providing ~1 watt power for control and communication; and a directional radio antenna etched onto the surface of the sail itself. Launched as a piggyback payload, the total cost of the mission can be limited to a few tens of thousands of dollars.

3.Missions
The craft would be capable of some ambitious missions. For example:
a) It could rendezvous with a nearby asteroid from the Apollo or Amor groups. Closeup pictures could be transmitted back to Earth at a low bit

rate.
b) It could be steered into a lunar polar orbit. Previously unobserved areas around the lunar poles could be viewed. By angling the sail to reflect sunlight downwards, polar craters whose bases never receive sunlight could be imaged. Bright reflections would confirm that volatiles such as water ice have become trapped in these locations.[Immensely valuable information for setting up a manned lunar base, BTW]
c) It could be sent to rendezvous with a small asteroid or comet nucleus. Impacting at low speed, a thin wire probe attached to the craft causes it to rebound while capturing a tiny sample is a sharp-edged tube, like performing a biopsy. Returning to Earth, the sail acts as an ideal re-entry parachute: load per unit area 20 gm/m2 ensures that heat is reradiated so efectively that the sail temperature cannot exceed ~300 deg C. The material sample is recovered, enclosed in a small insulating container.

Contact: Colin Jack Tel. 0865-200447
Oxford Mathematical Designs, 131 High Street, Oxford OX1 4DH, England

---

This guy would love to hear from anyone interested in this project or seeking details or anything, and would be most happy to send you more information.

Andy

--

Andy Jonathan J. Broderick, | "I have come that they might have |
Rutherford Lab., UK | life, and have it to the full" |
Mail : ajjb@adam2.bnsc.rl.ac.uk | - Jesus Christ |

```
#wyswietlenie klasy wiadomosci o indeksie 180
print(train_dataset.target_names[train_dataset.target[180]])
```

```
sci.space
```

Obraz 18: Dopasowana kategoria wiadomości o indeksie 180

Wybrana wiadomość została przydzielona do kategorii „sci.space”, ponieważ zawiera słowa kluczowe typu „science”, „space” czy „Earth”.

```
from sklearn.feature_extraction.text import CountVectorizer
count_vect = CountVectorizer(analyzer='word', ngram_range=(1, 1))
X_train_counts = count_vect.fit_transform(train_dataset.data)
```

```
word_list = count_vect.get_feature_names_out()
count_list = X_train_counts.toarray().sum(axis=0)
word_count = dict(zip(word_list,count_list))
sorted(word_count.items(), key=lambda x: x[1], reverse=True)
```

```
[('the', 30530),
 ('to', 15205),
 ('of', 13501),
 ('and', 11909),
 ('in', 9861),
 ('is', 8380),
 ('that', 6952),
 ('it', 6349),
 ('for', 5829),
 ('on', 4651),
 ('from', 4554),
 ('you', 4510),
 ('edu', 4471),
 ('be', 4399),
 ('this', 4193),
 ('are', 3351),
 ('with', 3291),
 ('have', 3193),
 ('as', 3122),
```

Obraz 19: Wyświetlenie najczęściej występujących słów we wszystkich wiadomościach

```
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
clf = Pipeline([
    ('vect', CountVectorizer(analyzer='word',ngram_range=(1, 1))),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])
```

```
#wytrenowanie pipelinu
clf.fit(train_dataset.data, train_dataset.target)
```

```
Pipeline(steps=[('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                ('clf', MultinomialNB())])
```

Obraz 20: Trenowanie modelu za pomocą naiwnego klasyfikatora bayesowskiego

```
#proste testy wytrenowanego modelu
sentences = ['Asteroids are dangerous for Earth.',
             'I am dreaming about new motorcycle.',
             'I hate baseball games.',
             'Did you see my C implementations?']

predicted_categories = clf.predict(sentences)

for sentence, predicted_category in zip(sentences, predicted_categories):
    print('%r => %s' % (sentence, train_dataset.target_names[predicted_category]))
```

```
'Asteroids are dangerous for Earth.' => sci.space
'I am dreaming about new motorcycle.' => rec.motorcycles
'I hate baseball games.' => rec.sport.baseball
'Did you see my C implementations?' => sci.crypt
```

Obraz 21: Proste testy wytrenowanego modelu

```
import numpy as np
predicted = clf.predict(test_dataset.data)
np.mean(predicted == test_dataset.target)
```

```
0.9735015772870662
```

Obraz 22: Dokładność wytrenowanego modelu

```
#macierz pomyłek
#'sci.crypt', 'rec.sport.baseball', 'rec.motorcycles', 'sci.space'
from sklearn.metrics import confusion_matrix
confusion_matrix(test_dataset.target, predicted, labels=[0,1,2,3])
```

```
array([[389,   0,   7,   2],
       [  1, 390,   3,   3],
       [  1,   4, 391,   0],
       [  1,   0,  20, 373]], dtype=int64)
```

Obraz 23: Macierz pomyłek dla wytrenowanego modelu

Na podstawie macierzy pomyłek widzimy, że model bardzo dobrze radzi sobie z dopasowywaniem kategorii do wiadomości; najbardziej myli „sci.crypt” z „rec.motorcycles” oraz „sci.space” z „rec.motorcycles”. Kategoria „rec.motorcycles” jest kategorią, którą najczęściej model błędnie wybiera.

5.2. Zadanie #4

i. Treść polecenia

Proszę zmienić użyty klasyfikator na drzewo decyzyjne i zbadać jakość wyników klasyfikacji.

ii. Rozwiązanie

```
from sklearn import tree
clf = Pipeline([
    ('vect', CountVectorizer(analyzer='word',ngram_range=(1, 1))),
    ('tfidf', TfidfTransformer()),
    ('clf', tree.DecisionTreeClassifier()),
])
```

```
#wytrenowanie pipelinu
clf.fit(train_dataset.data, train_dataset.target)
```

```
Pipeline(steps=[('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                ('clf', DecisionTreeClassifier())])
```

Obraz 24: Trenowanie pipelinu z klasyfikatorem drzewa decyzyjnego - porównywanie na raz jednego słowa z jednym słowem

```
#proste testy wytrenowanego modelu
sentences = ['Asteroids are dangerous for Earth.',
             'I am dreaming about new motorcycle.',
             'I hate baseball games.',
             'Did you see my C implementations?']

predicted_categories = clf.predict(sentences)

for sentence, predicted_category in zip(sentences, predicted_categories):
    print('%r => %s' % (sentence, train_dataset.target_names[predicted_category]
```

```
'Asteroids are dangerous for Earth.' => rec.sport.baseball
'I am dreaming about new motorcycle.' => rec.motorcycles
'I hate baseball games.' => rec.sport.baseball
'Did you see my C implementations?' => rec.sport.baseball
```

Obraz 25: Proste testy wytrenowanego modelu

Warto zauważyć, że przy zmianie klasyfikatora na drzewo decyzyjne model inaczej przypisuje kategorie już dla prostych testów wytrenowanego modelu - dwie z czterech wiadomości błędnie przypisuje do „rec.sport.baseball”.

```
#testy wytrenowanego modelu
import numpy as np
predicted = clf.predict(test_dataset.data)
np.mean(predicted == test_dataset.target)
```

```
0.8145110410094637
```

Obraz 26: Dokładność wytrenowanego modelu po zmianie klasyfikatora

Wyraźnie spada też dokładność wytrenowanego modelu.

```
#macierz pomyłek
# 'sci.crypt', 'rec.sport.baseball', 'rec.motorcycles', 'sci.space'
confusion_matrix(test_dataset.target, predicted, labels=[0,1,2,3])
```

```
array([[340,  28,  18,  12],
       [ 21, 329,  17,  30],
       [ 21,  38, 323,  14],
       [ 24,  45,  26, 299]], dtype=int64)
```

Obraz 27: Macierz pomyłek dla wytrenowanego modelu po zmianie klasyfikatora

Macierz pomyłek również wyraźnie się różni - model dużo częściej błędnie przypisuje kategorie - najczęściej myli się w wiadomościach z kategorii „sci.space”, a najczęściej błędnie przypisywaną kategorią jest „rec.sport.baseball”.

5.3. Zadanie #5

i. Treść polecenia

Proszę wytrenować klasyfikator na zbiorze testującym a przetestować na zbiorze trenującym.

ii. Rozwiązanie

```
test_dataset = fetch_20newsgroups(
    subset='train',
    categories=categories,
    shuffle=True,
    random_state=42)
train_dataset = fetch_20newsgroups(
    subset='test',
    categories=categories,
    shuffle=True,
    random_state=42)
#klasy
print(train_dataset.target_names)
print(test_dataset.target_names)
#liczba wiadomosci
print(len(train_dataset.data))
print(len(test_dataset.data))
```

```
['rec.motorcycles', 'rec.sport.baseball', 'sci.crypt', 'sci.space']
['rec.motorcycles', 'rec.sport.baseball', 'sci.crypt', 'sci.space']
1585
2383
```

Obraz 28: Trenowanie modelu na zbiorze testującym a testowanie na zbiorze trenującym

```
count_vect = CountVectorizer(analyzer='word', ngram_range=(1, 1))
X_train_counts = count_vect.fit_transform(train_dataset.data)
word_list = count_vect.get_feature_names_out()
count_list = X_train_counts.toarray().sum(axis=0)
word_count = dict(zip(word_list,count_list))
sorted(word_count.items(), key=lambda x: x[1], reverse=True)
```

```
[('the', 30530),
 ('to', 15205),
 ('of', 13501),
 ('and', 11909),
 ('in', 9861),
 ('is', 8380),
 ('that', 6952),
 ('it', 6349),
 ('for', 5829),
 ('on', 4651),
 ('from', 4554),
 ('you', 4510),
 ('edu', 4471),
 ('be', 4399),
 ('this', 4193),
 ('are', 3351),
 ('with', 3291),
 ('have', 3193),
 ('as', 3122),
```

Obraz 29: Najczęściej pojawiające się słowa we wszystkich wiadomościach

```
clf = Pipeline([
    ('vect', CountVectorizer(analyzer='word',ngram_range=(1, 1))),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])
#wytrenowanie pipelinu
clf.fit(train_dataset.data, train_dataset.target)
```

```
Pipeline(steps=[('vect', CountVectorizer()), ('tfidf', TfidfTransformer()),
                ('clf', MultinomialNB())])
```

Obraz 30: Trenowanie pipelinu - porównywanie na raz jednego słowa z jednym słowem

```
#proste testy wytrenowanego modelu
sentences = ['Asteroids are dangerous for Earth.',
             'I am dreaming about new motorcycle.',
             'I hate baseball games.',
             'Did you see my C implementations?']

predicted_categories = clf.predict(sentences)

for sentence, predicted_category in zip(sentences, predicted_categories):
    print('%r => %s' % (sentence, train_dataset.target_names[predicted_category]))
```

```
'Asteroids are dangerous for Earth.' => sci.space
'I am dreaming about new motorcycle.' => rec.motorcycles
'I hate baseball games.' => rec.sport.baseball
'Did you see my C implementations?' => rec.motorcycles
```

Obraz 31: Proste testy wytrenowanego modelu

Model przypisał błędnie tylko 1 wiadomość w prostych testach - zamiast „sci.crypt” oznaczył kategorię „rec.motorcycles”.

```
predicted = clf.predict(test_dataset.data)
np.mean(predicted == test_dataset.target)
```

```
0.9697859840537139
```

Obraz 32: Dokładność wytrenowanego modelu

Model mimo wszystko posiada wysoką dokładność predykcji.

```
# 'sci.crypt', 'rec.sport.baseball', 'rec.motorcycles', 'sci.space'
confusion_matrix(test_dataset.target, predicted, labels=[0,1,2,3])
```

```
array([[576,  10,   7,   5],
       [  5, 583,   8,   1],
       [  1,   1, 592,   1],
       [  2,  10,  21, 560]], dtype=int64)
```

Obraz 33: Macierz pomyłek wytrenowanego modelu

Model najczęściej myli „sci.space” z „rec.motorcycles” oraz „sci.crypt” z „rec.sport.baseball”. Kategoria „rec.motorcycles” jest najczęściej błędnie przypisywana.

5.4. Zadanie #6

i. Treść polecenia

Proszę ustawić w odpowiednim miejscu parametr analyzer='char' i sprawdzić jak zmieniają się wyniki.

ii. Rozwiązanie

```
count_vect = CountVectorizer(analyzer='char', ngram_range=(1, 1))
X_train_counts = count_vect.fit_transform(train_dataset.data)
word_list = count_vect.get_feature_names_out()
count_list = X_train_counts.toarray().sum(axis=0)
word_count = dict(zip(word_list,count_list))
sorted(word_count.items(), key=lambda x: x[1], reverse=True)
```

Obraz 34: Najczęściej pojawiające się znaki we wszystkich wiadomościach

```
[(' ', 611986),
 ('e', 356576),
 ('t', 274214),
 ('a', 244374),
 ('o', 235500),
 ('i', 226347),
 ('n', 216152),
 ('s', 208562),
 ('r', 192674),
 ('h', 130791),
 ('l', 128842),
 ('c', 112438),
 ('d', 107127),
 ('u', 93488),
 ('m', 85388),
 ('p', 73989),
 ('.', 70497),
 ('g', 65579),
 ('y', 62716),
 ('f', 60465),
 ('-', 58052),
 ('b', 55879),
 ('w', 52034),
 ('\n', 48768),
 ('v', 33854),
 (',', 32937),
 ('k', 29792),
 (':', 23327),
 ('1', 21961),
 ('>', 21654),
 ('0', 20665),
 ("'", 12544),
 (')', 12342),
 ('=', 12049),
 ('2', 11888),
 ('(', 11469),
 ('j', 11011),
 ('3', 10130),
 ('9', 9656),
 ('@', 9244),
 ('_', 9067),
 ('*', 8902),
 ('x', 8593),
 ('5', 8571),
 ('4', 7558),
 ('"', 7244),
 ('z', 6870),
 ('6', 6142),
 ('8', 6060),
 ('7', 5630),
 ('?', 5533),
 ('|', 5082),
 ('/', 4533),
 ('q', 3984),
 ('!', 3021),
 ('<', 2891),
 (';', 1978),
 ('~', 1898),
 ('\t', 1635),
 ('#', 1520),
 ('^', 1369),
 (']', 1287),
 ('[', 1273),
 ('+', 1206),
 ('$', 1172),
 ('`', 847),
 ('&', 820),
 ('\\', 720),
 ('%', 495),
 ('}', 257),
 ('{', 133),
 ('\xa0', 12),
 ('º', 9),
 ('é', 5),
 ('ñ', 5),
 ('ç', 3),
 ('è', 3),
 ('\x1a', 1),
 ('ê', 1)]
```

```
count_vect = CountVectorizer(analyzer='char', ngram_range=(5, 5))
X_train_counts = count_vect.fit_transform(train_dataset.data)
word_list = count_vect.get_feature_names()
count_list = X_train_counts.toarray().sum(axis=0)
word_count = dict(zip(word_list,count_list))
sorted(word_count.items(), key=lambda x: x[1], reverse=True)
```

```
[('-----', 31948),
 (' the ', 27356),
 (' and ', 10438),
 ('=====', 9864),
 ('ation', 7643),
 (' that', 6619),
 ('that ', 6043),
 ('_____', 5571),
 ('*****', 5337),
 (' for ', 5231),
 ('n the', 4746),
 ('tion ', 4451),
 (' of t', 4195),
 ('f the', 4010),
 ('t the', 3967),
 (' this', 3902),
 ('of th', 3848),
 ('this ', 3710),
 (' with', 3511),
```

Obraz 35: Najczęściej pojawiające się 5 znakowe ciągi we wszystkich wiadomościach

```
clf = Pipeline([
    ('vect', CountVectorizer(analyzer='char', ngram_range=(1, 1))),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])
#wytrenowanie pipelinu
clf.fit(train_dataset.data, train_dataset.target)
```

```
Pipeline(steps=[('vect', CountVectorizer(analyzer='char')),
                ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])
```

```
#proste testy wytrenowanego modelu
sentences = ['Asteroids are dangerous for Earth.',
             'I am dreaming about new motorcycle.',
             'I hate baseball games.',
             'Did you see my C implementations?']

predicted_categories = clf.predict(sentences)

for sentence, predicted_category in zip(sentences, predicted_categories):
    print('%r => %s' % (sentence, train_dataset.target_names[predicted_category]
```

```
'Asteroids are dangerous for Earth.' => sci.crypt
'I am dreaming about new motorcycle.' => sci.crypt
'I hate baseball games.' => rec.sport.baseball
'Did you see my C implementations?' => sci.crypt
```

Obraz 36: Wytrenowanie pipelinu dla porównania pojedynczych znaków

Przy porównaniu pojedynczych znaków model w prostych testach przypisał prawie wszystkie wiadomości do kategorii „sci.crypt”.

```
predicted = clf.predict(test_dataset.data)
np.mean(predicted == test_dataset.target)
```

```
0.3747634069400631
```

```
# 'sci.crypt', 'rec.sport.baseball', 'rec.motorcycles', 'sci.space'
confusion_matrix(test_dataset.target, predicted, labels=[0,1,2,3])
```

```
array([[116,  90, 161,  31],
       [ 31, 119, 231,  16],
       [ 29,  32, 296,  39],
       [ 27,  56, 248,  63]], dtype=int64)
```

Obraz 37: Dokładność i macierz pomyłek dla modelu porównywającego pojedyncze znaki

Dokładność modelu jest bardzo niska. Model najczęście myli „sci.crypt" z „rec.motorcycles" a kategorię „sci.space" prawie zawsze oznacza jako „rec.motorcycles".

```
clf = Pipeline([
    ('vect', CountVectorizer(analyzer='char',ngram_range=(5, 5))),
    ('tfidf', TfidfTransformer()),
    ('clf', MultinomialNB()),
])
#wytrenowanie pipelinu
clf.fit(train_dataset.data, train_dataset.target)
```

```
Pipeline(steps=[('vect', CountVectorizer(analyzer='char', ngram_range=(5, 5))),
                ('tfidf', TfidfTransformer()), ('clf', MultinomialNB())])
```

```
#proste testy wytrenowanego modelu
sentences = ['Asteroids are dangerous for Earth.',
             'I am dreaming about new motorcycle.',
             'I hate baseball games.',
             'Did you see my C implementations?']

predicted_categories = clf.predict(sentences)

for sentence, predicted_category in zip(sentences, predicted_categories):
    print('%r => %s' % (sentence, train_dataset.target_names[predicted_category]
```

```
'Asteroids are dangerous for Earth.' => sci.space
'I am dreaming about new motorcycle.' => rec.motorcycles
'I hate baseball games.' => rec.sport.baseball
'Did you see my C implementations?' => sci.crypt
```

Obraz 38: Trenowanie pipelinu w modelu porównującym 5 znakowe ciągi

Model przypisał poprawnie wszystkie wiadomości w prostych testach.

```
predicted = clf.predict(test_dataset.data)
np.mean(predicted == test_dataset.target)
```

```
0.8845425867507887
```

```
# 'sci.crypt', 'rec.sport.baseball', 'rec.motorcycles', 'sci.space'
confusion_matrix(test_dataset.target, predicted, labels=[0,1,2,3])
```

```
array([[327,   4,  64,   3],
       [ 16, 344,  36,   1],
       [  6,   2, 385,   3],
       [  2,   1,  45, 346]], dtype=int64)
```

Obraz 39: Dokładność i macierz pomyłek dla modelu porównującego 5 znakowe ciągi

Model ma dość wysoką dokładność, jednak nie tak wysoką jak model porównujący ze sobą pojedyncze znaki. Najczęściej błędnie przypisuje „sci.crypt" jako „rec.motorcycles" oraz „sci.space" jako „rec.motorcycles".