

# Systemy sztucznej inteligencji

dokumentacja projektu Rozpoznawanie Liter

Dawid Nowakowski, grupa 8  
Marcin Hajdecki, grupa 1

Informatyka Praktyczna, wydział Matematyki Stosowanej  
Politechnika Śląska

22 czerwca 2023

# Spis treści

<b>1</b>	<b>Wprowadzenie</b>	<b>2</b>
1.1	Opis projektu . . . . .	2
1.2	Dodatkowe informacje . . . . .	2
1.2.1	Użyte wersje . . . . .	2
1.2.2	Opis zbioru danych . . . . .	3
<b>2</b>	<b>Implementacja</b>	<b>4</b>
2.1	Przygotowanie danych . . . . .	4
2.1.1	Opisywanie wierszy . . . . .	4
2.1.2	Zmniejszanie długości zbioru . . . . .	5
2.1.3	Normalizacja . . . . .	8
2.2	Algorytm KNN - K Nearest Neighbours (Najbliższych sąsiadów) . . . . .	9
2.3	Metryki . . . . .	10
<b>3</b>	<b>Podsumowanie</b>	<b>11</b>
3.1	Testy . . . . .	11
3.2	Wnioski . . . . .	12
<b>4</b>	<b>Pełen kod aplikacji</b>	<b>13</b>

# 1 Wprowadzenie

## 1.1 Opis projektu

Projekt ma na celu stworzenie modelu rozpoznawania liter pisma odręcznego na bazie algorytmu KNN, a następnie analizę wyników uzyskanych dla różnych parametrów (ilość sąsiadów, metryka, wielkość zbioru danych). Pierwszym krokiem będzie odpowiednie spreparowanie danych i przygotowanie ich do analizy. Następnie zostaną wykonane testy algorytmem KNN na których podstawie wyciągneliśmy odpowiednie wnioski.

## 1.2 Dodatkowe informacje

### 1.2.1 Użyte wersje

Python 3.9.7

Biblioteki:

numpy 1.24.1

pandas 1.5.3

matplotlib 3.7.1

sklearn 1.2.2

seaborn 0.12.2

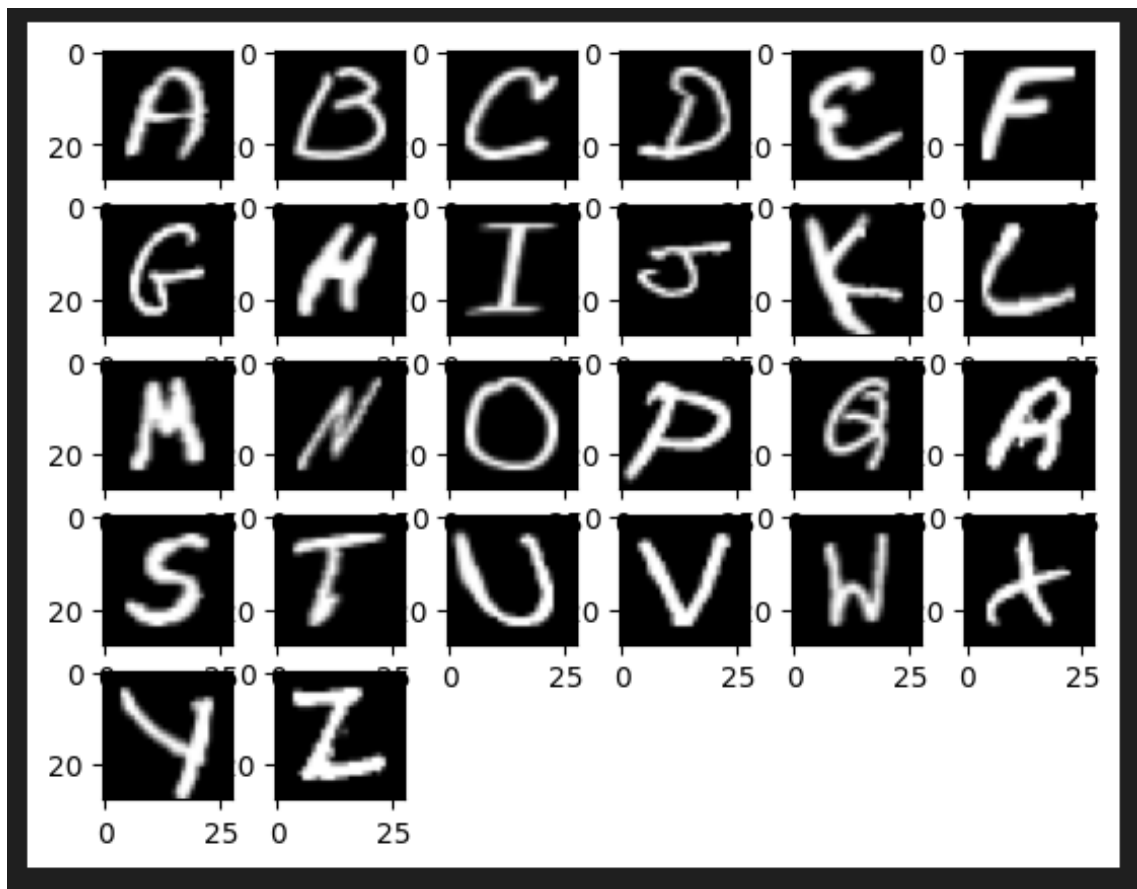
random

math

### 1.2.2 Opis zbioru danych

Do analizy wybraliśmy dataset *A-Z Handwritten Alphabets in .csv format* pobrany z serwisu *Kaggle*.

Baza zawiera 372 450 rekordów reprezentujących różne litery, każdy posiadający 785 kolumn. Pierwsza kolumna zawiera index litery, który w późniejszej fazie jest przekształcany na znak reprezentujący daną literę. Dalsze kolumny są reprezentacją obrazu 28 x 28px ( $28 \times 28 = 784$ ) w formacie 0 - biały piksel, (1-255) - natężenie jasności piksela. Wszystkie litery są wyśrodkowane i przeskalowane do stałego rozmiaru. Rysunek 1 pokazuje przykładowe wizualizacje liter dla tego zestawu danych.



## 2 Implementacja

### 2.1 Przygotowanie danych

#### 2.1.1 Opisywanie wierszy

Pierwszym krokiem, potrzebnym do tego aby dane były bardziej czytelne było zmapowanie pierwszej kolumny z indeksów liter na ich rzeczywistą reprezentację znakową. Krok ten nie jest konieczny do poprawnego funkcjonowania, jednak na etapie tworzenia modelu ułatwił początkową analizę i testy. Na tym etapie można zauważyć jak nierówny jest podział wierszy dla określonych liter.

0	13869	A	13869
1	8668	B	8668
2	23409	C	23409
3	10134	D	10134
4	11440	E	11440
5	1163	F	1163
6	5762	G	5762
7	7218	H	7218
8	1120	I	1120
9	8493	J	8493
10	5603	K	5603
11	11586	L	11586
12	12336	M	12336
13	19010	N	19010
14	57825	O	57825
15	19341	P	19341
16	5812	Q	5812
17	11566	R	11566
18	48419	S	48419
19	22495	T	22495
20	29008	U	29008
21	4182	V	4182
22	10784	W	10784
23	6272	X	6272
24	10859	Y	10859
25	6076	Z	6076
Name: 0, dtype: int64		Name: 0, dtype: int64	

```
1 label = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',
2         6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',
3         12:'M',13:'N',14:'O',15:'P',16:'Q',17:'R',
4         18:'S',19:'T',20:'U',21:'V',22:'W',23:'X',24:'Y',25:'Z'}
5
6 data.iloc[:,0] = data.iloc[:,0].map(label)
```

---

Rezultat działania kodu na powyższym obrazku.

#### Opis:

1. Słownik label zawiera informacje o tym jaka litera ma zostać przypisana do danego indexu.
2. Następnie mamy nadpisanie kolumny 0 dla wszystkich wierszy na kolumnę zmapowaną.

### 2.1.2 Zmniejszanie długości zbioru

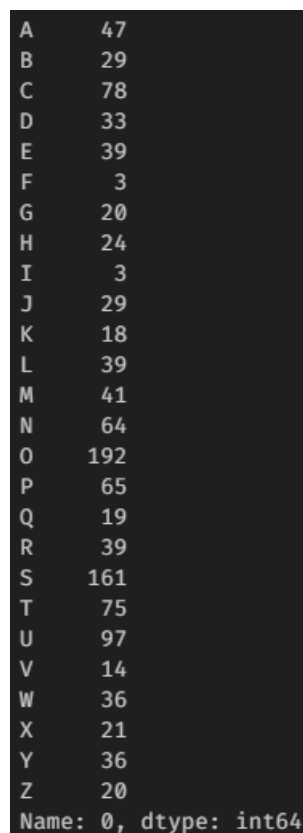
Ponieważ dataset ma długość 372 450 rekordów, przeprowadzenie kilku testów na tak dużym zbiorze zajęło by zbyt duże ilości czasu, zdecydowaliśmy o podzieleniu zbioru na kilka mniejszych 1300, 2080, 2600 i 5200 elementowych zbiorów i na tych zmniejszonych zbiorach przeprowadziliśmy testy. Kolejnym czynnikiem, który zdecydował o zmniejszeniu zbioru była nierównomierność ilości wierszy dla liter np. litera O ma 57825 wierszy, zaś F już tylko 1163 wierszy. Chaotyczność tego podziału można zobaczyć na obrazku z poprzedniej strony.

Pierwsza próba zmniejszenia zbioru była bardzo prymitywna i jak później się okazało, nie wydajna, ponieważ mimo, że dataset stał się mniej liczny, chaotyczny podział pozostał co drastycznie wpłynęło na zmniejszenie dokładności.

```
1 short_data = pd.DataFrame(columns=data.columns)
2
3 for i in range(len(data)):
4     if i % 300 == 0:
5         short_data = short_data.append(data.loc[i], ignore_index=True)
```

---

Sposób ten polegał na wybieraniu modulo n-tego elementu. Był on nie wydajny ponieważ bez żadnej kontroli poprostu "przeskakiwał" część danych. Rezultat wykonania na poniższym obrazku. Już na pierwszy rzut oka można zauważyć jak niekorzystna jest ta metoda. Czas wykonania ok. 35sec.



A	47
B	29
C	78
D	33
E	39
F	3
G	20
H	24
I	3
J	29
K	18
L	39
M	41
N	64
O	192
P	65
Q	19
R	39
S	161
T	75
U	97
V	14
W	36
X	21
Y	36
Z	20
Name: 0, dtype: int64	

Drugi sposób zapewnił już satysfakcjonujący równy podział, jednak dalej nie był zadowalający ze względu na długi czas wykonywania (ok 4x dłuższy niż sposób 1).

```
1 short_data = pd.DataFrame(columns=data.columns)
2
3 lw = {}
4
5 for letter in string.ascii_uppercase:
6     lw[letter] = 0
7
8 for i in range(len(data)):
9
10     elem = data.loc[i][0]
11     #print(elem, " ", lw[elem])
12
13     if lw[elem] < 50:
14         short_data = short_data.append(data.loc[i], ignore_index=True)
15         lw[elem] += 1
```

---

Ta metoda była oparta o liczbę wystąpień. Jeżeli liczba wystąpień danej litery w nowym zbiorze była mniejsza od wybranego n, dołączały się kolejne wiersze, w przeciwnym wypadku wiersze były pomijane aż do napotkania kolejnej litery. Czas wykonania ok. 2min 18sec.

```
A 50
B 50
C 50
D 50
E 50
F 50
G 50
H 50
I 50
J 50
K 50
L 50
M 50
N 50
O 50
P 50
Q 50
R 50
S 50
T 50
U 50
V 50
W 50
X 50
Y 50
Z 50
Name: 0, dtype: int64
```

„Do trzech razy sztuka”

Trzeci sposób okazał się strzałem w dziesiątkę ponieważ gwarantował równomierne rozłożenie wierszy oraz bardzo szybki czas wykonania. (ok 40x szybszy niż sposób 1).

```
1 short_data = data.groupby('0').apply(lambda x: x.sample(n=50, replace=False))
2 short_data = short_data.reset_index(drop=True) #resetowanie indeksacji dataframeu
```

---

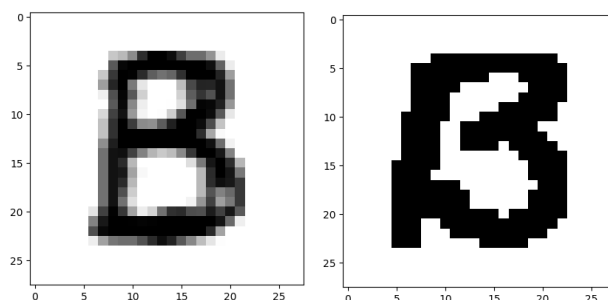
Sekret tkwił w całkowitej rezygnacji z wielokrotnie powtarzających się ręcznie napisanych pętli i wykorzystaniu funkcji przygotowanych dla dataframe'ów w bibliotece Pandas. Funkcja **groupby('0')** grupuje dane na podstawie unikalnych wartości kolumny '0'. **.apply()** stosuje dla każdej z grup funkcję **lambda x: x.sample(n=50, replace=False)**, która losowo wybiera n wierszy z każdej z grup bez powtórzeń, za co odpowiada **replace=False**. Czas wykonania ok. 0.8sec.

```
A 50
B 50
C 50
D 50
E 50
F 50
G 50
H 50
I 50
J 50
K 50
L 50
M 50
N 50
O 50
P 50
Q 50
R 50
S 50
T 50
U 50
V 50
W 50
X 50
Y 50
Z 50
Name: 0, dtype: int64
```



### 2.1.3 Normalizacja

Jako, iż przetwarzamy obrazek, zdecydowaliśmy w ramach normalizacji aby każdy z wierszy pozbawić natężenia koloru, a na jego miejsce wstawić informację czy piksel jest biały czy czarny. Spowodowało to, że zamiast dużych 3 - cyfrowych liczb, mamy jedynie 0 i 1. Zastosowanie normalizacji takiego typu okazało się skuteczne, ponieważ zmniejszyło czas wykonywania algorytmu oraz zwiększyło jego dokładność.



```
1 def normalize(data_list):
2     for col in data_list.columns:
3         if col != '0':
4             for i in range(len(data_list[col])):
5                 if data_list.at[i, col] != 0:
6                     data_list.at[i, col] = 1
```

---

Algorytm normalizacyjny przechodzi po każdej kolumnie i wierszu, z pominięciem kolumny etykiety, sprawdza czy dana wartość w komórce jest różna od 0 i jeśli tak, to wartość tą zamienia na 1. Efekt takiego zabiegu jest widoczny na powyższym obrazku. W podrozdziale z wynikami znajduje się porównanie czasu i dokładności dla testów z i bez normalizacji.

## 2.2 Algorytm KNN - K Nearest Neighbours (Najbliższych sąsiadów)

Całość modelu opiera się o algorytm KNN, czyli jeden z podstawowych i prostych algorytmów klasyfikacji. Przed wykonaniem algorytmu zbiór jest dzielony na zbiór treningowy i testowy w proporcjach 6:4.

```
Data: data_list, data_row, k, m
Result: category
distances  $\leftarrow []$ ;
for each row in data_list do
    | Oblicz odległość między row a data_row
    | Dodaj odległość do distances
end
Posortuj distances rosnąco
k_dist  $\leftarrow k$  pierwszych elementów z distances
lw  $\leftarrow []$ 
for each elem in k_dist do
    | Zlicz liczbę wystąpień etykiety elem
end
return etykieta z największą liczbą wystąpień
Algorithm 1: Pseudokod algorytmu KNN.
```

```
Data: k, m, train_data, test_data
Result: good, all
all  $\leftarrow 0$ ;
good  $\leftarrow 0$ ;
for each row in test_data do
    | result  $\leftarrow$  wynik KNN dla train_data, row, k, m;
    | if result == etykieta row then
        | good  $\leftarrow$  good + 1
    | end
    | all  $\leftarrow$  all + 1
end
return good/all * 100;
```

**Algorithm 2:** Pseudokod obliczania poprawności wyników

## 2.3 Metryki

Podczas analizy Algorytmu 1 zastanawiająca może wydawać się *obliczana odległość*. Jest ona obliczana na podstawie dwóch metryk. Opis metryk użytych do klasyfikacji:

1. Metryka euklidesowa:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

2. Metryka Manhattan:

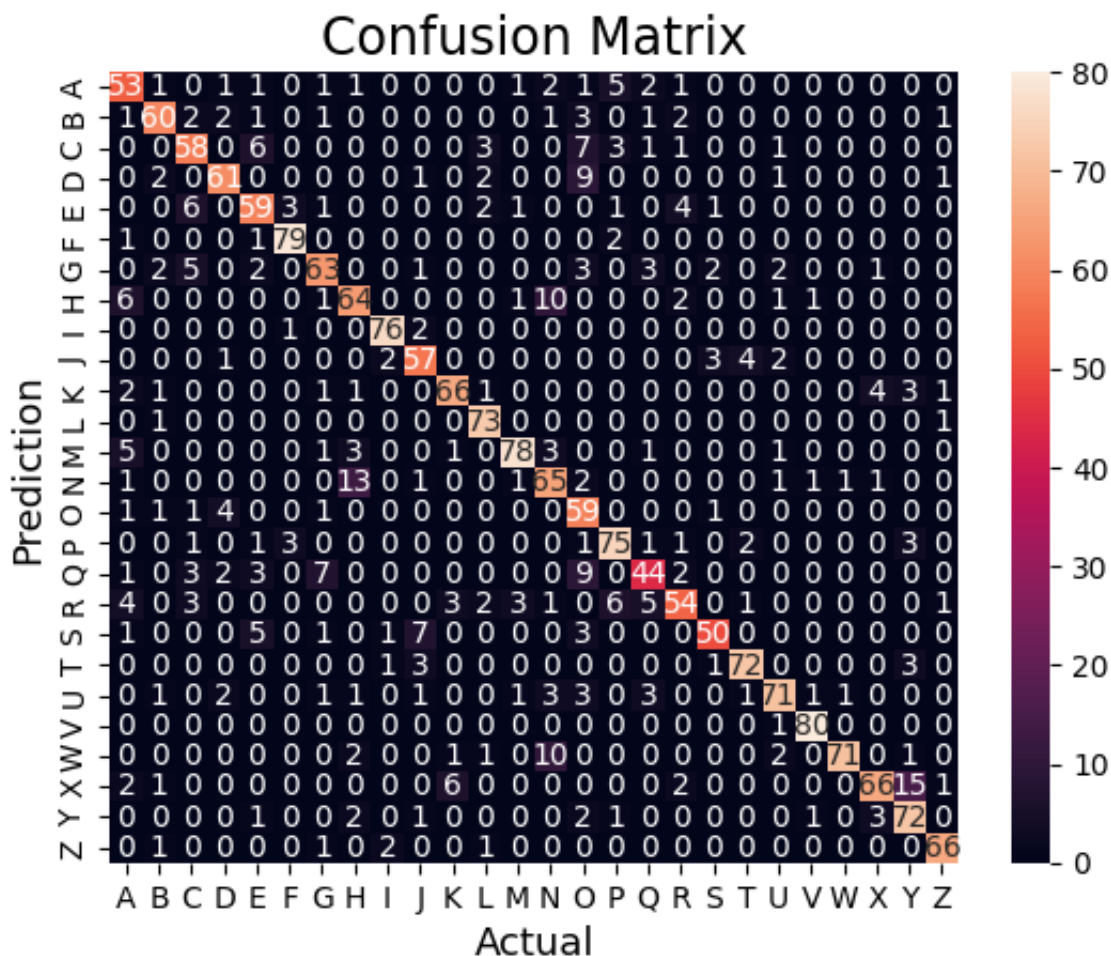
$$d(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^n |x_i - y_i|$$

Gdzie  $d(\mathbf{x}, \mathbf{y})$  to odległość między wektorami  $\mathbf{x}$  i  $\mathbf{y}$  w  $n$ -wymiarowej przestrzeni. Natomiast  $x_i$  i  $y_i$  oznaczają  $i$ -tą współrzędną wektorów  $\mathbf{x}$  i  $\mathbf{y}$ .

## 3 Podsumowanie

### 3.1 Testy

Dla danych znormalizowanych przeprowadziliśmy różne testy dla 1300, 2080, 2600 i 5200 elementowych zbiorów, dla metryki euklidesowej, dla 1 i 4 najbliższych sąsiadów oraz metryki Manhattan dla 1 najbliższego sąsiada. Dodatkowo dla sprawdzenia efektywności normalizacji, wykonaliśmy testy na danych nieznormalizowanych dla 1300 i 2080 elementowych zbiorów, k, m jak wyżej. **Najlepszą dokładność równą 81.35% otrzymaliśmy dla 5200 elementowego, znormalizowanego zbioru, 1 najbliższego sąsiada i metryki euklidesowej.** Poniżej macierz błędów dla najlepszego wyniku.



Reszta wyników przeprowadzonych testów prezentuje się następująco:

**Dokładność dla danych znormalizowanych:**

Ilość rekordów	k=1, m=2	k=4, m=2	k=1, m=1
<b>1300</b>	72.12%	69.62%	72.12%
<b>2080</b>	77.04%	78.37%	77.04%
<b>2600</b>	80.0%	80.0%	80.0%
<b>5200</b>	<b>81.35%</b>	80.62%	81.35%

**Czas wykonania dla danych znormalizowanych:**

Ilość rekordów	k=1, m=2	k=4, m=2	k=1, m=1
<b>1300</b>	4min 40.1s	4min 40.3s	4min 40.5s
<b>2080</b>	11min 57.9s	11min 57.9s	11min 55.2s
<b>2600</b>	18min 45.4s	18min 45.6s	18min 42.4s
<b>5200</b>	74min 43.9s	75min 3.5s	83min 26.9s

**Dokładność dla danych bez normalizacji:**

Ilość rekordów	k=1, m=2	k=4, m=2	k=1, m=1
<b>1300</b>	69.81%	65.38%	69.04%
<b>2080</b>	74.04%	72.72%	73.56%

**Czas wykonania dla danych bez normalizacji:**

Ilość rekordów	k=1, m=2	k=4, m=2	k=1, m=1
<b>1300</b>	4min 42.6s	4min 42.1s	4min 43.4s
<b>2080</b>	12min 1.3s	11min 58.7s	12min 1.0s

## 3.2 Wnioski

Pierwszą ciekawostką jest to, że dla danych znormalizowanych metryka euklidesowa i Manhattan dały w każdym przypadku ten sam wynik procentowy, ta ciekawa zależność może wynikać z zero - jedynkowego sposobu normalizacji. Śmiało można również stwierdzić, że zastosowanie takiej normalizacji mocno wpłynęło na zwiększenie dokładności o kilka procent jak i na skrócenie czasu wykonywania, zapewne poprzez uniknięcie wykonywania obliczeń na dużych liczbach.

## 4 Pełen kod aplikacji

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 from sklearn.neighbors import KNeighborsClassifier
5 from sklearn import metrics
6 from sklearn.model_selection import train_test_split
7 import random
8 import math
9 import seaborn
10
11 data = pd.read_csv('A_Z Handwritten Data.csv')
12
13 label = {0:'A',1:'B',2:'C',3:'D',4:'E',5:'F',
14          6:'G',7:'H',8:'I',9:'J',10:'K',11:'L',
15          12:'M',13:'N',14:'O',15:'P',16:'Q',17:'R',
16          18:'S',19:'T',20:'U',21:'V',22:'W',23:'X',24:'Y',25:'Z'}
17
18 data.iloc[:,0] = data.iloc[:,0].map(label)
19
20 row_count = data['0'].value_counts() # pobieranie liczby wierszy dla
    danej kategorii
21 min_row_count = row_count.min()
22
23 short_data = data.groupby('0').apply(lambda x: x.sample(n=80, replace=
    False))
24
25 short_data = short_data.reset_index(drop=True)
26
27 class DataProcessing:
28     @staticmethod
29     def shuffling(data_list):
30         for i in range(len(data_list)-1,0,-1):
31             index = random.randint(0,i-1)
32             data_list.loc[i], data_list.loc[index] = data_list.loc[index
    ], data_list.loc[i]
33
34     @staticmethod
35     def normalize(data_list):
36         for col in data_list.columns:
37             if col != '0':
38                 for i in range(len(data_list[col])):
39                     if data_list.at[i, col] != 0:
40                         data_list.at[i, col] = 1
41
42     @staticmethod
43     def train_test_split(data_list,prc):
44         train_len = round((len(data_list)) * prc)
45         #test_len = round((len(data_list)) * 0.4)
46
47         train = data_list[0:train_len]
48         test = data_list[train_len:len(data_list)]
49
```

```

50         test = test.reset_index(drop=True)
51
52         return train, test
53
54     @staticmethod
55     def label_split(data_list):
56         feature_list = []
57         label_list = []
58
59         sum = 0
60         for col in data_list.columns:
61             if col != '0':
62                 sum +=1
63
64         for i in range(len(data_list)):
65             feature_list.append(data_list.loc[i].to_list()[len(data_list)
66                                     .columns)-sum:])
67
68         for i in range(len(data_list)):
69             label_list.append(data_list.loc[i].to_list()[0:len(data_list)
70                                     .columns)-sum][0])
71
72         return feature_list, label_list
73
74     DataProcessing.normalize(short_data)
75     DataProcessing.shuffling(short_data)
76
77     train_data, test_data = DataProcessing.train_test_split(short_data, 0.6)
78
79     def Minkowski_dist(x,y,m):
80         res=0
81         for i in range(len(x)):
82             res += (abs(x[i] - y[i]))**m
83
84         res = math.pow(res,1.0/m)
85
86         return res
87
88     def KNN_algorithm(list, data, k, m):
89
90         distances = []
91
92         feature_list, label_list = DataProcessing.label_split(list)
93
94         for i, elem in enumerate(feature_list):
95             distances.append([Minkowski_dist(elem,data,m),i])
96
97         distances = sorted(distances)
98
99         k_dist = distances[:k]
100
101         lw={}
102
103         for x in label_list:
104             lw[x] = 0

```

```

103
104     #print(lw)
105
106     for elem in k_dist:
107         lw[label_list[elem[1]]] += 1
108
109     max_elem = float("-inf")
110
111     for elem in lw:
112         if lw[elem] > max_elem:
113             max_elem = lw[elem]
114             category = elem
115
116     return category
117
118 all=0
119 good=0
120 bad=0
121
122 sum = 0
123 for col in short_data.columns:
124     if col != "0":
125         sum +=1
126
127 data, actual = DataProcessing.label_split(test_data)
128
129 predicted = []
130
131 for i in range(len(test_data)):
132
133     result = KNN_algorithm(train_data, test_data.loc[i].to_list()[len(
134         short_data.columns)-sum:], 1, 2)
135     predicted.append(result)
136
137     if result == test_data.loc[i].to_list()[0]:
138         good += 1
139     else:
140         bad += 1
141
142     all += 1
143
144 print("k = 1, m = 2 (k - ilosc sasiadow branych pod uwage)")
145 print(f"All: {all}, good: {good}, bad: {bad}")
146 print(f"Test statistic: {round(good/all*100,2)}%")
147
148 cm = metrics.confusion_matrix(actual, predicted)
149 seaborn.heatmap(cm,
150                 annot=True,
151                 fmt='g',
152                 xticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
153                             'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
154                             'V', 'W', 'X', 'Y', 'Z'],
155                 yticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
156                             'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
157                             'V', 'W', 'X', 'Y', 'Z'])

```



```

153 plt.ylabel('Prediction',fontsize=13)
154 plt.xlabel('Actual',fontsize=13)
155 plt.title('Confusion Matrix',fontsize=17)
156 plt.show()
157
158 all=0
159 good=0
160 bad=0
161
162 sum = 0
163 for col in short_data.columns:
164     if col != "0":
165         sum +=1
166
167 data, actual = DataProcessing.label_split(test_data)
168
169 predicted = []
170
171 for i in range(len(test_data)):
172
173     result = KNN_algorithm(train_data, test_data.loc[i].to_list()[len(
174         short_data.columns)-sum:], 4, 2)
175     predicted.append(result)
176
177     if result == test_data.loc[i].to_list()[0]:
178         good += 1
179     else:
180         bad += 1
181
182     all += 1
183
184 print("k = 4, m = 2 (k - ilosc sasiadow branych pod uwage)")
185 print(f"All: {all}, good: {good}, bad: {bad}")
186 print(f"Test statistic: {round(good/all*100,2)}%")
187
188 cm = metrics.confusion_matrix(actual, predicted)
189 seaborn.heatmap(cm,
190     annot=True,
191     fmt='g',
192     xticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
193         'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
194         'V', 'W', 'X', 'Y', 'Z'],
195     yticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
196         'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
197         'V', 'W', 'X', 'Y', 'Z'])
198
199 plt.ylabel('Prediction',fontsize=13)
200 plt.xlabel('Actual',fontsize=13)
201 plt.title('Confusion Matrix',fontsize=17)
202 plt.show()
203
204 all=0
205 good=0
206 bad=0
207
208 sum = 0

```

```

203 for col in short_data.columns:
204     if col != "0":
205         sum +=1
206
207 data, actual = DataProcessing.label_split(test_data)
208
209 predicted = []
210
211 for i in range(len(test_data)):
212
213     result = KNN_algorithm(train_data, test_data.loc[i].to_list()[len(
214         short_data.columns)-sum:], 1, 1)
215     predicted.append(result)
216
217     if result == test_data.loc[i].to_list()[0]:
218         good += 1
219     else:
220         bad += 1
221
222     all += 1
223
224 print("k = 1, m = 1 (k - ilosc sasiadow branych pod uwage)")
225 print(f"All: {all}, good: {good}, bad: {bad}")
226 print(f"Test statistic: {round(good/all*100,2)}%")
227
228 cm = metrics.confusion_matrix(actual, predicted)
229 seaborn.heatmap(cm,
230     annot=True,
231     fmt='g',
232     xticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
233         'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
234         'V', 'W', 'X', 'Y', 'Z'],
235     yticklabels=['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J',
236         'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
237         'V', 'W', 'X', 'Y', 'Z'])
238 plt.ylabel('Prediction',fontsize=13)
239 plt.xlabel('Actual',fontsize=13)
240 plt.title('Confusion Matrix',fontsize=17)
241 plt.show()

```

---