Systemy Sztucznej Inteligencji

Dokumentacja Projektu Rozpoznawanie cyfr z pisma odręcznego za pomocą algorytmu k-NN

Szymon Hankus
grupa 3/5

Antoni Jaszcz
grupa 3/5

Bartłomiej Pacia
grupa 3/5

20 czerwca 2023

1 Część I

1.1 Opis programu

Projekt składa się z dwóch głównych plików:

- pliku `knn.py`, zawierającego implementację klasyfikatora.
- pliku `knn_project.ipynb`, zawierającego demonstrację klasy i jej możliwości oraz wygenerowane wyniki.

1.2 Instrukcja obsługi oraz wymagania

Do uruchomienia projektu niezbędne jest środowisko `Python3` oraz `ipykernel` (lub zainstalowany w środowisku Pythonowym pakiet `ipykernel`). Dodatkowo, klasa `knn.py` korzysta z pakietów:

- `numpy`
- `math`
- `sklearn`

1.3 Dodatkowe informacje

Klasyfikator `knn` został przez nas napisany zgodnie z założeniami modułu Scikit-Learn. Klasyfikator implementuje zatem:

- **Jednolitość** - jednolitego interfejsu dla obiektów. Nasz klasyfikator posiada m. in. następujące metody:
  - `fit()` - dopasowującą dane uczące do modelu.
  - `predict()` - zwracającą przewidzianą klasę dla nowej próbki.
  - `score()` - zwracającą skuteczność, macierz pomyłek oraz inne wybrane metryki do oceny modelu na podstawie predykcji zestawu danych walidacyjnych.
  - `cross_val_score()` - dokonujący sprawdzianu krzyżowego dla zestawu danych walidacyjnych i zadanego podziału.
- **Nierozprzestrzenialność klas** - zbiory danych są reprezentowane przez macierze `NumPy`, a parametry to standardowe typy języka `Python` służące do opisu zmiennych.
- **Kompozycja** - klasyfikator podzielony jest na elementy składowe, które są wielokrotnie wykorzystywane.
- **Rozsądne wartości domyślne** - w klasyfikatorze zostały dobrane odpowiednie wartości domyślne (m.in. domyślna ilość sąsiadów $k = 5$).

Cały projekt, wraz z dokumentacją, jest dostępny na platformie `GitHub`.

2 Część II

2.1 Wstęp

Wzrost ilości danych i wymagania dotyczące szybkich obliczeń sprawiły, że stare technologie stały się przestarzałe. A przynajmniej tak może się wydawać. Metody te są nadal często stosowane [9, 4], ponieważ są proste, niezawodne i dokładne. Co więcej, warto je ponownie przeanalizować, ponieważ wciąż można je ulepszać i optymalizować.

Głównymi problemami związanymi z metodą k-nn jest jej duża złożoność obliczeniowa i duża ilość danych wymaganych do dobrego działania algorytmu. Może to być poważną wadą, szczególnie w systemach, które w dużym stopniu zależą od szybkości, takich jak serwery SQL. Od lat opracowywane są różne podejścia mające na celu rozwiązanie tego problemu. Wiele z nich polega na zmniejszeniu liczby przeglądanych próbek podczas obliczeń przewidywania algorytmu lub wykorzystaniu narzędzi do przetwarzania danych [7, 10]. W artykule [5] autor proponuje rozwiązanie polegające na uwzględnieniu jedynie reprezentantów danej klasy. W ten sposób nadmiarowe dane mogą zostać usunięte bez utraty mocy danych [1].

Kolejną kwestią jest niesławna klątwa wymiarowości. Problem ten dotyczy nie tylko klastrowania, ale także nowoczesnych technologii, takich jak sieci neuronowe, konwolucyjne sieci neuronowe [6] i rekurencyjne sieci neuronowe [8]. Dlatego ważne jest, aby znaleźć nowe rozwiązania w starszych modelach, które można zastosować również w tych nowszych, w celu rozwiązania problemów, które w inny sposób można łatwo pominąć ze względu na złożoność metod [3].

K-nn jest powszechnie uznawany za metodę rozwiązywania problemów związanych z grupowaniem, takich jak systemy rekomendacji. Jednakże, metoda ta była wielokrotnie odkrywana jako rozwiązanie nietypowych problemów, takich jak nadzór, cyberbezpieczeństwo i detekcja obrazów. Dlatego ważne jest ciągłe poszukiwanie nowych rozwiązań, które bardzo często mogą być następnie wykorzystywane do ulepszania nowoczesnych technologii.

2.2 Opis działania

2.2.1 Algorytm k-NN

k-NN jest prostym, dobrze działającym algorytmem używanym w statystyce do klasyfikacji i regresji. Zasada działania algorytmu jest następująca: Biorąc pod uwagę zbiór danych N próbek, wstępnie oznaczonych jako jedna z C klas, z P parametrami liczbowymi, algorytm daje predykcję dla nowej próbki, biorąc pod uwagę jej k "najbliższych sąsiadów"(punkty w $dim(P)$ wymiarowej przestrzeni metrycznej, najmniej oddalone od siebie według zadanej metryki) i zwracając odpowiednie dane wyjściowe:

- **Klasyfikacja:** wektor przynależności do klasy, z którego oceniana klasa jest wybierana podczas głosowania pluralistycznego wśród sąsiadów.
- **Regresja:** wektor średnich parametrów obliczonych na podstawie sąsiadów.

Najczęściej jako metrykę odległości w P-wymiarowej przestrzeni wybiera się normę euklidesową. Z tego powodu początkowa normalizacja parametrów jest kluczowa dla prawidłowego działania algorytmu.

Algorytm w dużym stopniu zależy od liczby próbek. Im więcej próbek k-NN ma do wykorzystania, tym większa szansa na wybranie podobnych obiektów, co skutkuje większą pewnością przewidywanej klasy, a tym samym ogólnie lepszą dokładnością. Niestety, stwarza to następujące problemy:

- Wszelkie anomalie lub zatrucie danych są bardzo uciążliwe.
- Wysoka złożoność obliczeniowa i duża ilość danych wymaganych do prawidłowego funkcjonowania skutkują stosunkowo wysoką złożonością czasową, co utrudnia implementację w scenariuszach czasu rzeczywistego.

2.2.2 Tworzenie uśrednionych reprezentantów

Aby rozwiązać problem związany z wysokim czasem obliczeń wspomnianym w rozdziale 2.2.1, proponujemy metodę redukcji próbek w bazie danych bez utraty jej wartości informacyjnej, opartą na uśrednianiu próbek. Biorąc pod uwagę liczbę próbek n_{c_i} pewnej klasy c_i, możemy przekształcić je w próbki $n_{samples}$ 1, każda utworzona z n_{layer} 2 kolejnych próbek klasy (z wyjątkiem ostatniej, która jest utworzona z pozostałych n_{left} 3 próbek), poprzez ich uśrednienie. Jest to dalej przedstawione na przykładzie w rozdziale 2.3.2.

$$n_{samples} = \lceil \log_2 n_{c_i} \rceil \tag{1}$$

$$n_{layer} = \lceil \frac{n_{c_i}}{n_{samples}} \rceil \tag{2}$$

$$n_{left} = n_{c_i} - ((n_{samples} - 1) * n_{layer}) \tag{3}$$

$$\hat{x}_{i_j} = \begin{cases} \frac{\sum_{k=1}^{n_{layer}} x_k}{n_{layer}} & , for \quad j < n_{samples} \\ \\ \frac{\sum_{k=1}^{n_{left}} x_k}{n_{left}} & , for \quad j = n_{samples} \end{cases} \tag{4}$$

2.2.3 Redukcja wymiarowości poprzez rzutowanie związanych parametrów

Aby zmniejszyć liczbę wymiarów, można zastosować rzutowanie zestawu powiązanych parametrów na nową cechę. Produktem takiej operacji jest nowy zestaw parametrów, reprezentujący nowo utworzone cechy. Można to osiągnąć na przykład poprzez uśrednienie podzbioru powiązanych parametrów P' do nowego pojedynczego parametru $\hat{p}'$ (przedstawionego jako wartość liczbowa).

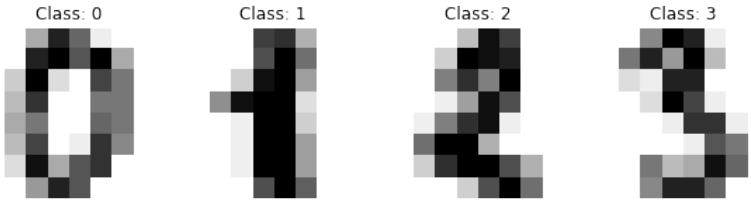
2.3 Eksperymenty

W tej sekcji opisano wszystkie przeprowadzone eksperymenty.

2.3.1 Dane i ustawienia treningowe

W badaniach wykorzystano [2]UCL Handwritten Digits Data Set, popularny zbiór danych do optycznego rozpoznawania pisma odręcznego. Składa się on z 1797 próbek obrazów cyfr 8x8 pikseli w skali szarości (z intensywnością pikseli w zakresie [0,16]). Każda klasa zawiera około 180 próbek. Przykłady klas (od 0 do 4) pokazano na rysunku 1.

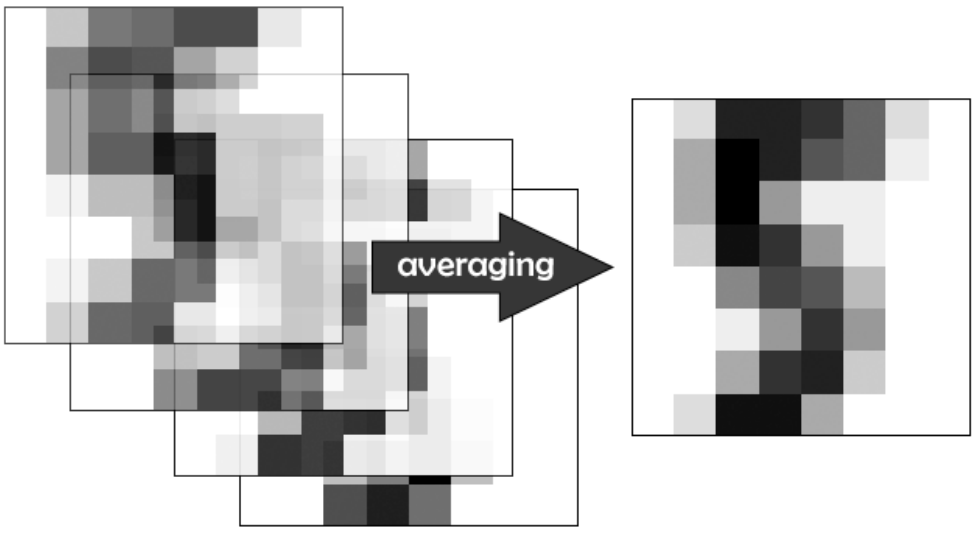
Ponieważ wszystkie wartości uwzględniane przez algorytm mieszczą się w zakresie [0,16], surowy zbiór danych jest już znormalizowany.



Rysunek 1: Przykładowe próbki danych

2.3.2 Reprezentanci klas

W eksperymentach przetestowano wpływ zastosowania metody średniej reprezentatywnej opisanej w sekcji 2.2.2, przekształcając próbki treningowe w odpowiednią liczbę reprezentantów dla każdej klasy. Innymi słowy, tacy reprezentanci zostali utworzeni poprzez nałożenie na siebie pewnej liczby próbek danej klasy, a następnie podzielenie iloczynu przez ich ilość. Najlepiej ilustruje to rysunek 2.



Rysunek 2: Tworzenie przedstawicieli z próbek klasy 5

2.3.3 Rzutowanie kolumn

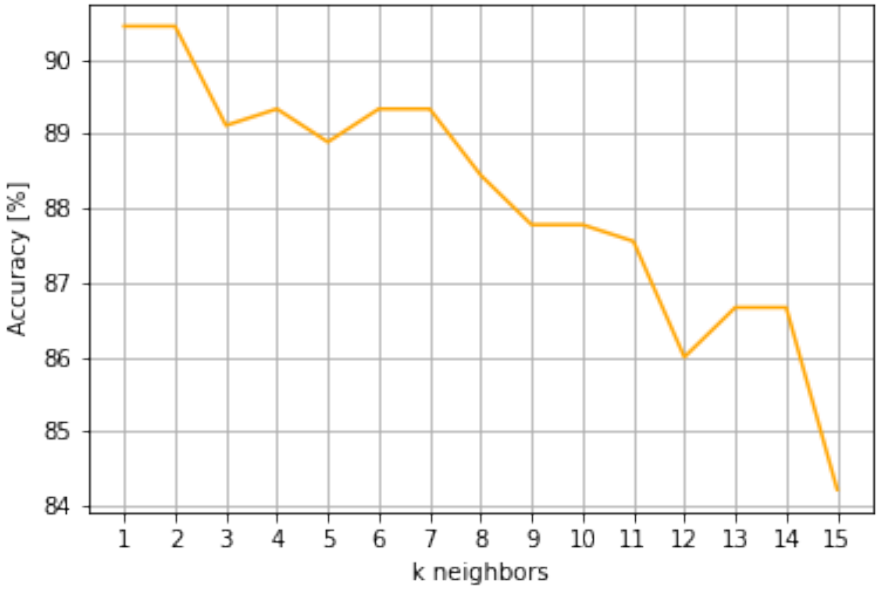
Jak opisano w sekcji 2.2.3, nowe cechy zostały utworzone przez uśrednienie powiązanych parametrów. W tym przypadku za powiązane parametry uznano piksele w tej samej kolumnie. Uśredniając wartości pikseli w każdej kolumnie, liczba parametrów została zmniejszona z $8x8 = 64$ do zaledwie $8x1 = 8$. Musiało to jednak zostać wykonane dla próbek w zestawie treningowym, a także dla każdej próbki, która była przewidywana.

2.3.4 Dobranie parametru k

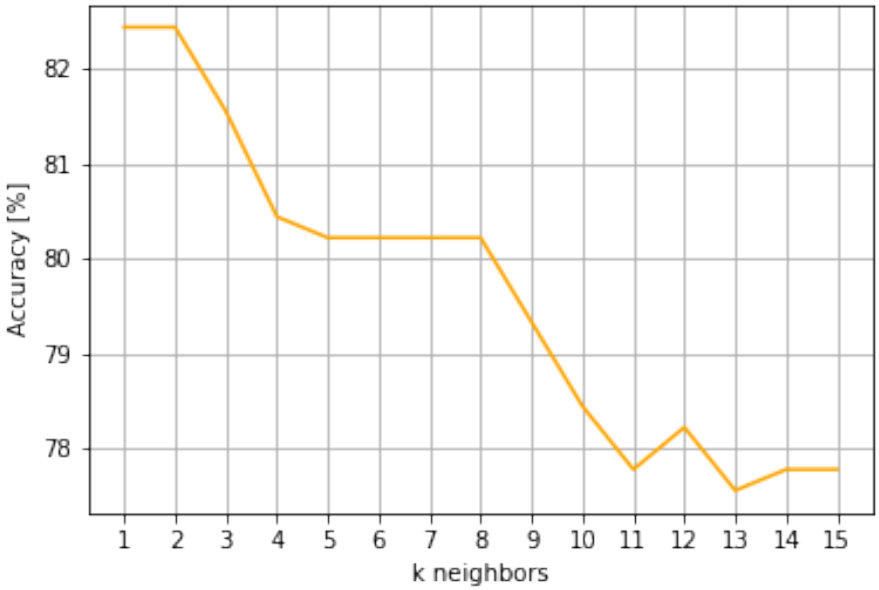
Kolejną kwestią związaną z algorytmem jest wybór odpowiedniego parametru k (liczba branych pod uwagę sąsiadów). Aby wybrać go efektywnie, przetestowano wydajność każdego proponowanego modelu z różnymi parametrami k, w zakresie od 1 do 15. k dobrano na podstawie uzyskanej dokładności modelu. Każdy model został wytrenowany i przetestowany na podzbiorach podzielonych 3:1 z oryginalnego zbioru danych. Zestawy treningowe i testowe były takie same dla każdego modelu. Dokładność uzyskana dla różnych wartości k jest pokazana na Rys. 3-5. Na podstawie uzyskanych wyników wybrano następujące parametry k:

- 7 dla bazowego modelu k-nn
- 2 dla modelu k-nn z zaimplementowanym uśrednianiem reprezentantów
- 2 dla modelu k-nn z zaimplementowanym rzutowaniem kolumn

Rysunek 3: Zależność dokładności bazowego modelu k-nn od różnych wartości k



Rysunek 4: Zależność dokładności modelu k-nn z zaimplementowaną metodą średniej reprezentatywnej od różnych wartości k



Rysunek 5: Zależność dokładności modelu k-nn z zaimplementowaną metodą rzutowania cech od różnych wartości k

2.4 Wyniki

Aby właściwie ocenić wydajność modeli, przeprowadzono 4-krotną walidację krzyżową. Modele zostały ocenione za pomocą czterech wskaźników:

- Dokładność
- Precyzja (macro śr.)
- Pełność (macro śr.)
- wynik f1 (macro śr.)

Calculated metrics are displayed in Tab. 1-4.

Tabela 1: Dokładność modeli

Model	fold 1	fold 2	fold 3	fold 4
bazowy k-nn	96.44	95.55	97.33	96.21
		śr = 96.38		
reprezentanci	91.76	91.54	93.76	90.42
		śr = 91.87		
rzutowanie	73.50	75.95	81.74	82.63
		śr = 78.45		

Tabela 2: Precyzja modeli (macro)

Model	fold 1	fold 2	fold 3	fold 4
bazowy k-nn	96.54	95.58	97.40	96.25
		śr = 96.44		
reprezentanci	92.44	92.04	93.93	90.72
		śr = 92.28		
rzutowanie	74.63	76.03	82.15	82.59
		śr = 78.85		

Tabela 3: Pełność modeli (macro)

Model	fold 1	fold 2	fold 3	fold 4
bazowy k-nn	96.43	95.53	97.39	96.14
		śr = 96.37		
reprezentanci	91.69	91.47	93.89	90.41
		śr = 91.87		
rzutowanie	73.83	75.67	81.93	82.40
		śr = 78.46		

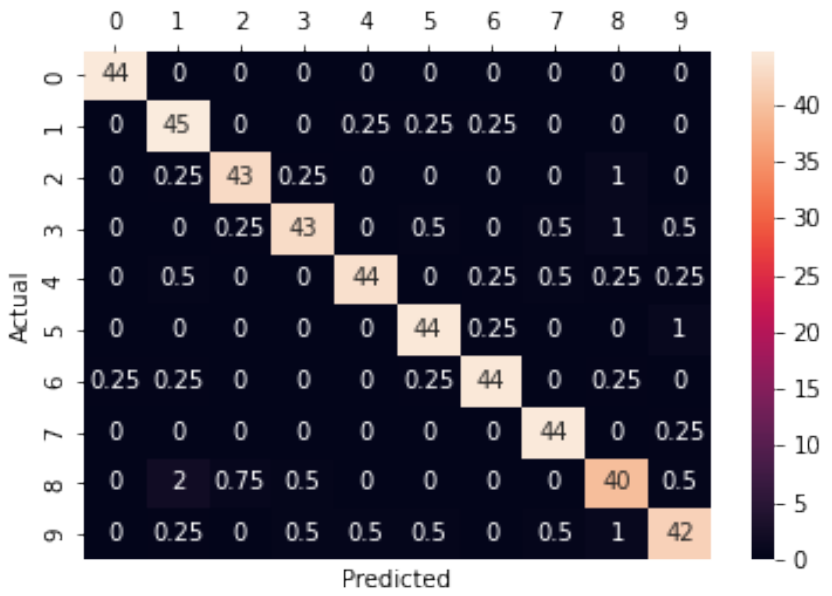
Aby dodatkowo zwizualizować uzyskane wyniki, dla każdego fałdu utworzono macierze pomyłek, a ich średnie przedstawiono na rys. 6-8. Porównanie średniej dokładności proponowanych modeli jest dodatkowo na Rys. 9

Tabela 4: Wynik f1 modeli (macro)

Model	fold 1	fold 2	fold 3	fold 4
bazowy k-nn	96.41	95.52	97.39	96.10
		śr = 96.36		
reprezentanci	91.81	91.50	93.88	90.32
		śr = 91.88		
rzutowanie	73.54	75.54	81.82	82.13
		śr = 78.26		

Tabela 5: Porównanie czasów dopasowania danych i wykonywania predykcji dla dokonanej walidacji krzyżowej (4 złożenia)

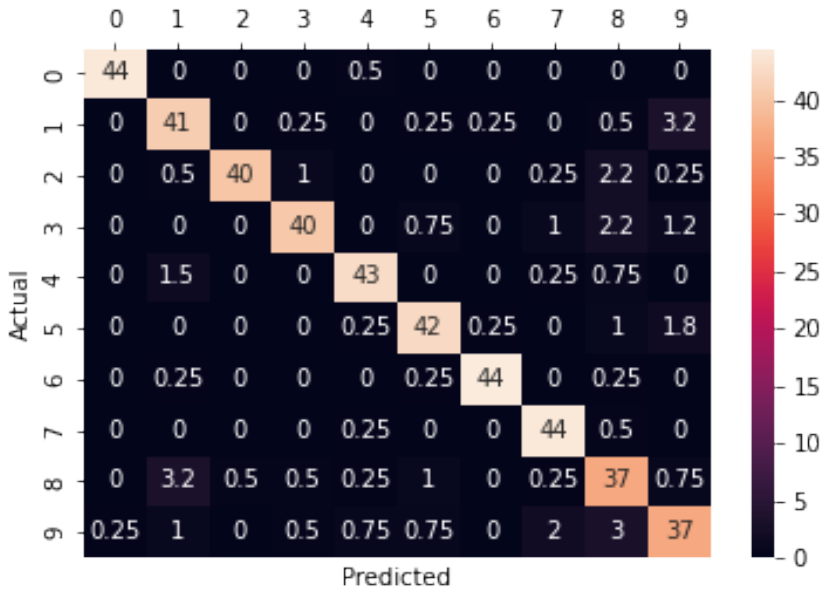
Model	czas
bazowy k-nn	19.875s
reprezentanci	1.250s
rzutowanie	19.531s



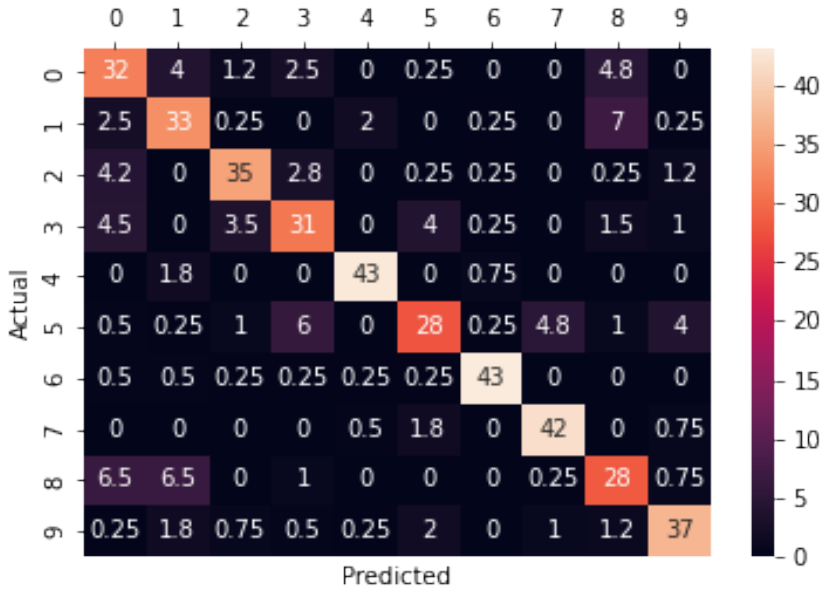
Rysunek 6: Średnia macierz pomyłek modelu bazowego

Wnioski

Jak widać w Tab. 1, model k-nn wypadł wyjątkowo dobrze w danym zadaniu. Nie tylko dokładność, ale wszystkie inne mierzone wskaźniki przedstawione w Tab. 2-4 pozostają na wysokim poziomie. Jak na tak prosty model, wynik jest zdumiewający. Jednakże, jak wspomniano w rozdziale 2.2.1, jednym z największych problemów jest jego złożoność obliczeniowa, skutkująca wysokim czasem obliczeń wraz ze wzrostem ilości danych. Problem ten można rozwiązać

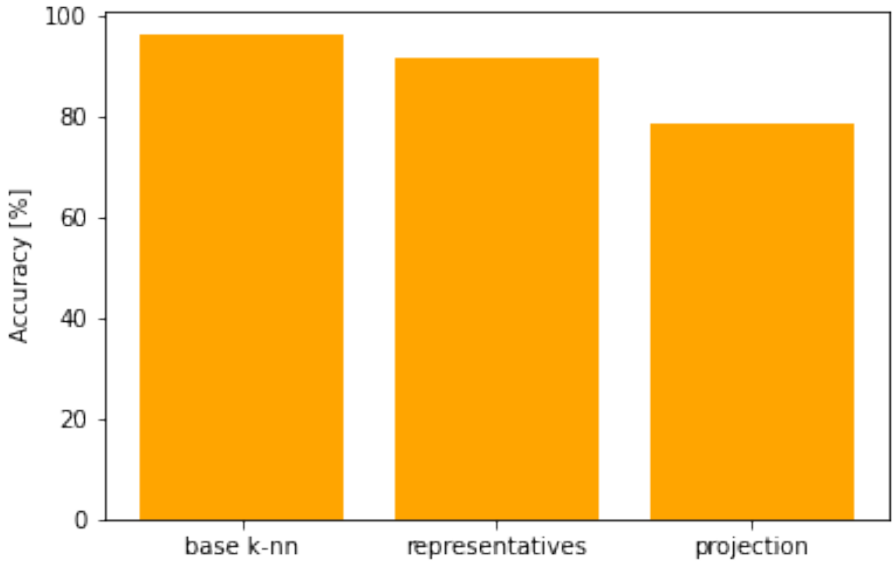


Rysunek 7: Średnia macierz pomyłek modelu z zaimplementowanymi średnimi reprezentantami



Rysunek 8: Średnia macierz pomyłek modelu z zaimplementowanym rzutowaniem cech

poprzez implementację średnich reprezentantów. Jak można zauważyć w Tab. 5, zastosowanie tej metody znacznie skraca czas obliczeń wymagany do przewidywania nowych próbek. Nie jest to jednak pozbawione konsekwencji. Jak można wywnioskować z Tab. 1, metoda reprezentantów wypadła o około 4.5 punktu procentowego gorzej pod względem dokładności niż



Rysunek 9: Porównanie średniej dokładności modeli

model podstawowy. Chociaż kompromis jakości na rzecz szybkości jest nieunikniony, wzrost szybkości znacznie przewyższa spadek dokładności, a metoda może być stosowana jako prawidłowe rozwiązanie, w którym szybkość jest najważniejsza. Metoda rzutowania wypadła raczej słabo zarówno pod względem szybkości, jak i dokładności. Metoda rzutowania cech może być używana jako prawidłowe rozwiązanie do zmniejszania wymiarów danych, nawet jeśli w tym przypadku okazała się daremna. Zmniejszenie liczby parametrów nie poprawiło znacząco ani szybkości, ani dokładności.

Pełen kod klasyfikatora

```
import numpy as np
import math
from sklearn.metrics import precision_score,recall_score,f1_score

class KNN():
    def __init__(self,n_neighbours=5, representatives=False, project=False):
        self.n_neighbours = n_neighbours
        self.representatives = representatives
        self.project = project
        pass

    def fit(self, X_train, Y_train):
        if self.representatives:
            self.X_train, self.Y_train = create_representatives(X_train, Y_train)
        else:
            self.X_train = X_train
            self.Y_train = Y_train
        if self.project:
            self.project = True
            self.X_train = self.project_datapoints(X_train)

    @staticmethod
    def project_datapoints(X):
        return np.mean(X,axis=2)

    def predict_datapoint(self, datapoint):
        class_distances = np.empty((0, 2))  # class labels and distances to the datapoint to other datapoints in X_train
        for x, y in zip(self.X_train, self.Y_train):
            dist = self.distance(datapoint, x)
            class_distances = np.append(class_distances, np.array([[y, dist]]), axis=0)

        class_distances = class_distances[class_distances[:, 1].argsort()]
        n_classes = class_distances[:self.n_neighbours, 0]

        _, counts = np.unique(n_classes, return_counts=True)  # find the most common class
        ind = np.argmax(counts)
        return int(n_classes[ind])

    def predict_datapoint_for_n_in_range(self, datapoint, n_range):
        class_distances = np.empty((0, 2))  # class labels and distances to the datapoint to other datapoints in X_train
        for x, y in zip(self.X_train, self.Y_train):
            dist = self.distance(datapoint, x)
            class_distances = np.append(class_distances, np.array([[y, dist]]), axis=0)

```

```
45          class_distances = class_distances[class_distances[:, 1].argsort
                ()]
46          predictions = np.empty((n_range),dtype=int)
47          for i in range(n_range):
48              n_classes = class_distances[:i+1, 0]
49              _, counts = np.unique(n_classes, return_counts=True)  # find
                    the most common class
50              ind = np.argmax(counts)
51              predictions[i]=int(n_classes[ind])
52          return predictions
53
54      @staticmethod
55      def distance(datapoint1, datapoint2):  # euclidian distance
56          flattened1 = np.ravel(datapoint1)
57          flattened2 = np.ravel(datapoint2)
58          return np.linalg.norm(flattened1 - flattened2)
59
60      def predict(self,X):
61          predicted = np.empty(len(X),dtype=int)
62          for i in range(len(X)):
63              predicted[i]=self.predict_datapoint(X[i])
64          return predicted
65
66      def predict_for_n_in_range(self,X,n_range): #
67          predicted = np.empty([len(X),n_range],dtype=int)
68          for i in range(len(X)):
69              predicted[i]=self.predict_datapoint_for_n_in_range(X[i],
                    n_range)
70          return predicted
71
72      def score(self,X,y):
73          if self.project:
74              X = self.project_datapoints(X)
75          correct = 0
76          # number of classes
77          confusion_matrix = np.zeros([10,10],dtype=int)
78          metrics= np.zeros((3),dtype=float)
79          #accuracy
80          predicted_lables = self.predict(X)
81          for predicted,actual in zip(predicted_lables,y):
82              if predicted==actual:
83                  correct+=1
84              confusion_matrix[actual][predicted]+=1
85
86          metrics[0] = precision_score(y,predicted_lables,average="macro")
87          metrics[1]  = recall_score(y,predicted_lables,average="macro")
88          metrics[2] = f1_score(y,predicted_lables,average="macro")
89          return correct/len(y), confusion_matrix,metrics
90
91      def score_for_n_in_range(self,X,y,n_range):
92          if self.project:
93              X = self.project_datapoints(X)
94          score = np.zeros(n_range,dtype=np.float32)
95
96          # creates empty cm with fixed 10 number of classes
```

```
97          confusion_matrix = np.zeros([n_range,10,10],dtype=int)
98
99          #accuracy score
100         i = 0
101         for predicted_column in self.predict_for_n_in_range(X,n_range).T
                :
102             correct = 0
103             for predicted, actual in zip(predicted_column,y):
104                 if predicted==actual:
105                     correct+=1
106                 confusion_matrix[i][actual][predicted]+=1    # the same
                        way sklearn produces CM
107             score[i]=correct/len(y)
108             i+=1
109         return score,confusion_matrix
110
111     def cross_val_score(self,X,Y, n_folds=4):
112         rest = len(X)%n_folds
113         if rest!=0:
114             print(f"WARNING! Cannot divide given dataset equally into {
                    n_folds} parts. Ignoring last {rest} elements!")
115         X_sets = np.split(X[:-rest],n_folds)
116         Y_sets = np.split(Y[:-rest],n_folds)
117
118         # create new model, not to overwrite current fit
119         model = KNN(self.n_neighbours,
120                     self.representatives,
121                     self.project)
122
123         # accuracy
124         scores = np.empty(n_folds,dtype=float)
125         cm= np.empty([n_folds,10,10],dtype=int)
126         metrics = np.zeros([n_folds,3],dtype=float)
127
128         for i in range(n_folds):
129             X_train = X_sets.copy()
130             Y_train = Y_sets.copy()
131             X_val = X_train.pop(i)
132             Y_val = Y_train.pop(i)
133
134             model.fit(np.concatenate(X_train),
135                       np.concatenate(Y_train))
136             scores[i],cm[i],metrics[i] = model.score(X_val,Y_val)
137         return scores,cm,metrics
138
139 def create_representatives(X,Y):
140     n_classes = np.zeros(10,dtype=int)  # number of classes
141     for y in Y:
142         n_classes[y]+=1
143
144     # sorts accoring to labels
145     X_sorted = [x for _,x in sorted(zip(Y,X),key=lambda el : el[0])]
146
147     index_x=0
148     class_representants = []
```

```
149     for n in n_classes:
150         n_representatives = math.ceil(math.log2(n)) # eq. 1
151         representatives = np.empty((n_representatives),dtype=type(
                X_sorted[0]))
152         step = math.ceil(n/n_representatives) # eq. 2
153         for i in range(n_representatives-1):
154             representatives[i]=sum(X_sorted[index_x+i*step      :
                    index_x+(i+1)*step])//step
155         representatives[n_representatives-1] = sum(X_sorted[index_x+(
                n_representatives-1)*step      :     index_x+n])//(n-(
                n_representatives-1)*step) # eq. 4
156         index_x+=n
157         for el in representatives:
158             class_representants.append(el)
159     return np.array(class_representants,dtype=int), np.ravel([np.full((
            math.ceil(math.log2(n))),i,dtype=int) for i in range(10)])
```

Literatura

[1] D.A. Adeniyi, Z. Wei, and Y. Yongquan. Automated web usage data mining and recommendation system using k-nearest neighbor (knn) classification method. *Applied Computing and Informatics*, 12(1):90–108, 2016.

[2] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[4] Antoni Jaszcz and Dawid Połap. Aimm: Artificial intelligence merged methods for flood ddos attacks detection. *Journal of King Saud University-Computer and Information Sciences*, 34(10):8090–8101, 2022.

[5] Antoni Jaszcza. Reducing the number of calculations in k-nn by class representatives atb voting. 2021.

[6] Dawid Połap, Natalia Wawrzyniak, and Marta Włodarczyk-Sielicka. Side-scan sonar analysis using roi analysis and deep neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–8, 2022.

[7] Dawid Połap, Marcin Woźniak, Robertas Damaševičius, and Rytis Maskeliūnas. Bio-inspired voice evaluation mechanism. *Applied Soft Computing*, 80:342–357, 2019.

[8] Jakub Siłka, Michał Wieczorek, and Marcin Woźniak. Recurrent neural network model for high-speed train vibration prediction from time series. *Neural Computing and Applications*, 34(16):13305–13318, 2022.

[9] Mindaugas Ulinskas, Marcin Woźniak, and Robertas Damaševičius. Analysis of keystroke dynamics for fatigue recognition. In Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Giuseppe Borruso, Carmelo M. Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O.

Apduhan, Elena Stankova, and Alfredo Cuzzocrea, editors, *Computational Science and Its Applications – ICCSA 2017*, pages 235–247, Cham, 2017. Springer International Publishing.

[10] Marcin Woźniak, Andrzej Sikora, Adam Zielonka, Kuljeet Kaur, M Shamim Hossain, and Mohammad Shorfuzzaman. Heuristic optimization of multipulse rectifier for reduced energy consumption. *IEEE Transactions on Industrial Informatics*, 18(8):5515–5526, 2021.