Systemy Sztucznej Inteligencji

Dokumentacja Projektu

Porównanie algorytmu KNN oraz Naiwnego Klasyfikatora Bayesa przy klasyfikacji odręcznie pisanych cyfr.

Piotr Skowroński gr. 3/6
Krzysztof Czuba gr. 4/7
Jakub Poreda gr. 3/6

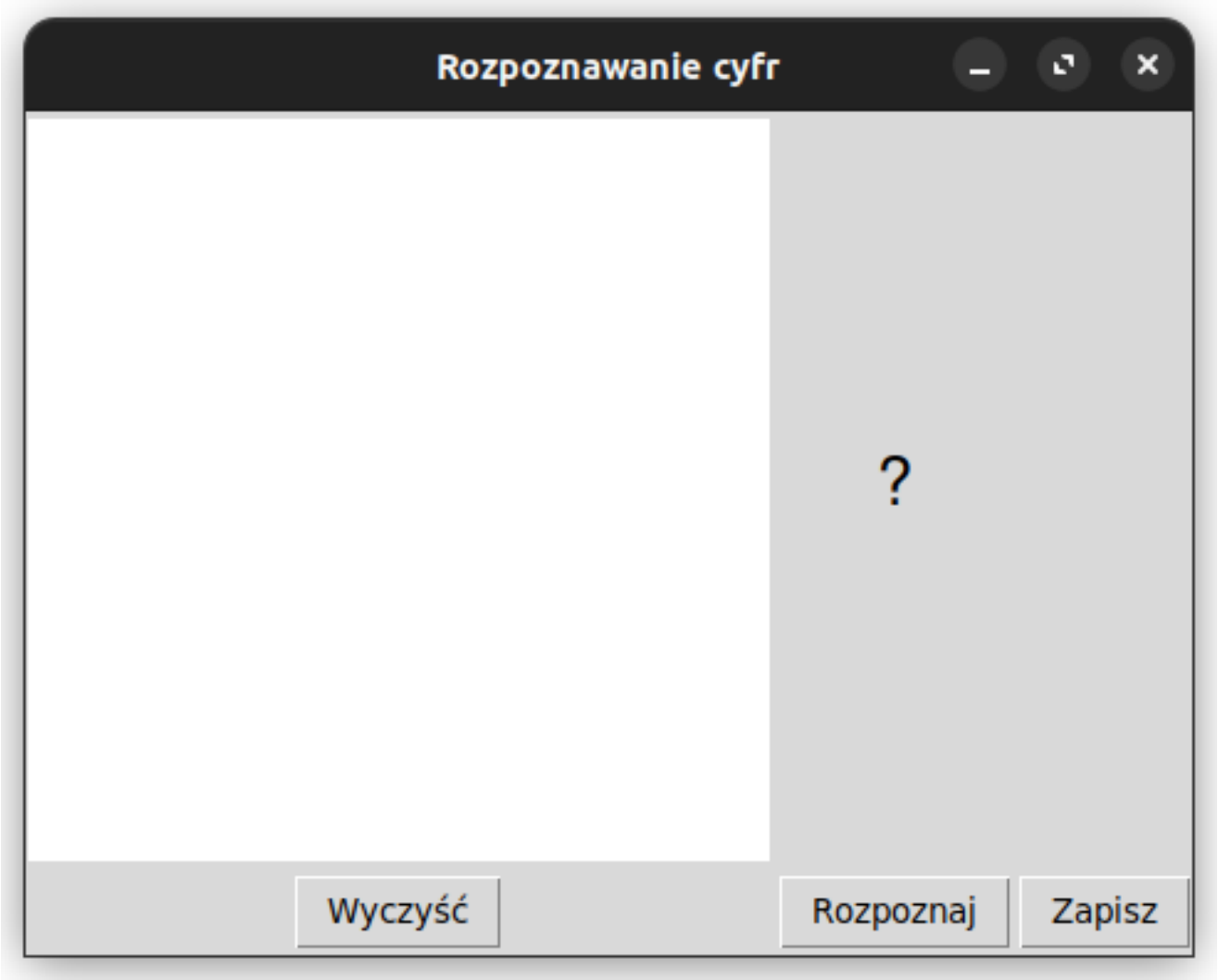
27 czerwca 2023

Spis treści

1 Wstęp

1.1 Opis programu

Celem programu jest klasyfikowanie odręcznie pisanych cyfr przez użytkownika. Aplikacja zawiera proste GUI, które pozwala użytkownikowi narysować cyfrę na płótnie, a następnie po wciśnięciu przycisku 'Rozpoznaj' program klasyfikuje cyfrę za pomocą jednego z klasyfikatorów w celu rozpoznania narysowanej cyfry. Otrzymane wyniki klasyfikacji są wyświetlane w bloku po prawej stronie interfejsu użytkownika.



Rysunek 1: Wygląd aplikacji

Rysunek 2: Rozpoznawanie

1.2 Użyte biblioteki

Program korzysta z następujacych zewnętrznych bibliotek:
- Pillow
    - Do transformacji zapisanych cyfr na macierz
    - Do transformacji narysowanej cyfry na macierz
- numpy
- seaborn

1.3 Baza danych

Baza danych składa się z 1000 obrazów cyfr narysowanych przez nas (100 dla każdej cyfry). Każdy piksel obrazu jest reprezentowany w skali szarości (ma wartość od 0 do 255, gdzie 0 to biały, a 255 to czarny kolor). Obrazy są przechowywane w formacie png.

Rysunek 3: Przykładowy obraz z bazy danych

Rysunek 4: Ten sam obraz po zmianie rozdzielczości na 28x28 pikseli

2 Opis działania

2.1 Normalizacja danych

Piksele wczytanego obrazu są konwertowane na skalę szarości tj. 0 - kolor biały, 255 - kolor czarny oraz są normalizowane do przedziału od 0 do 1 co ułatwia modelowi dopasowanie cyfr. Wzór na normalizację pojedyńczego piksela:

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)}$$

gdzie:

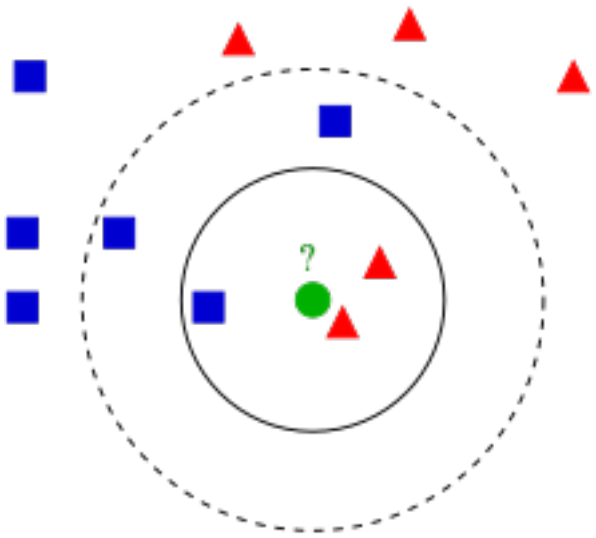
z_i - znormalizowany piksel
x_i - piksel
x - zbiór wszystkich pikseli

Ostatecznie wzór ma postać:

$$z_i = \frac{x_i}{255}$$

2.2 Algorytm k najbliższych sąsiadów

Klasyfikator kNN to jedna z ważniejszych nieparametrycznych metod klasyfikacji. W tej metodzie klasyfikowany obiekt przydzielamy do tej klasy, do której należy większość z k sąsiadów.



Rysunek 5: Przykład klasyfikacji metodą kNN

W przypadku k=3 (mniejszy okrąg), zielona kropka zostanie zakwalifikowana do czerwonych trójkątów. W przypadku k=5 (większy okrąg) - do niebieskich kwadratów.

2.3 Metryka odległości

Użyta została odległość Minkowskiego określona wzorem:

$$L_m(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

gdzie:

L_m - odległość między punktami x i y
x, y - punkty w przestrzeni n wymiarowej
x_i, y_i - i'ta współrzędna punktów x i y
p - parametr określający rodzaj metryki

Przetestowaliśmy skuteczność klasyfikatora kNN dla liczby sąsiadów $k \in \{1, 3, 5, 7, 9, 11, 13, 15\}$ oraz dla wartości parametru $p \in \{1, 2, 3\}$.

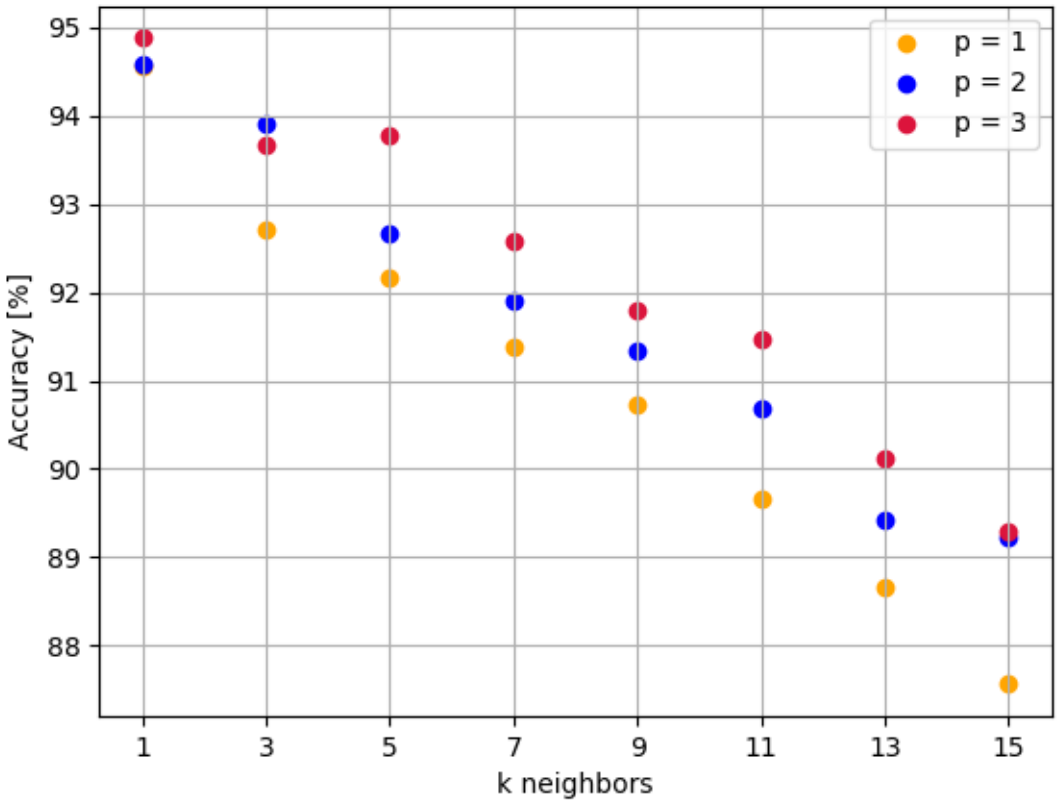
2.4 Pseudokod algorytmu kNN

Data: Dane wejściowe: zbiór treningowy *train_data* zbiór testowy *test_data* liczba sąsiadów k metryka p
Result: Zbiór testowy z przewidzianymi etykietami
foreach *test_instance in test_data* **do**
  distances = [];
  foreach *train_instance in train_data* **do**
    distance = *point_distance*(*test_instance*, *train_instance*, p);
    distances.append((*train_instance*, *distance*));
  end
  sorted_distances = *sort*(*distances*, *by* = *distance*);
  k_nearest_neighbors = *sorted_distances*[: k];
  test_instance.set_predicted_label(*predicted_label*);
end
return *test_data*;

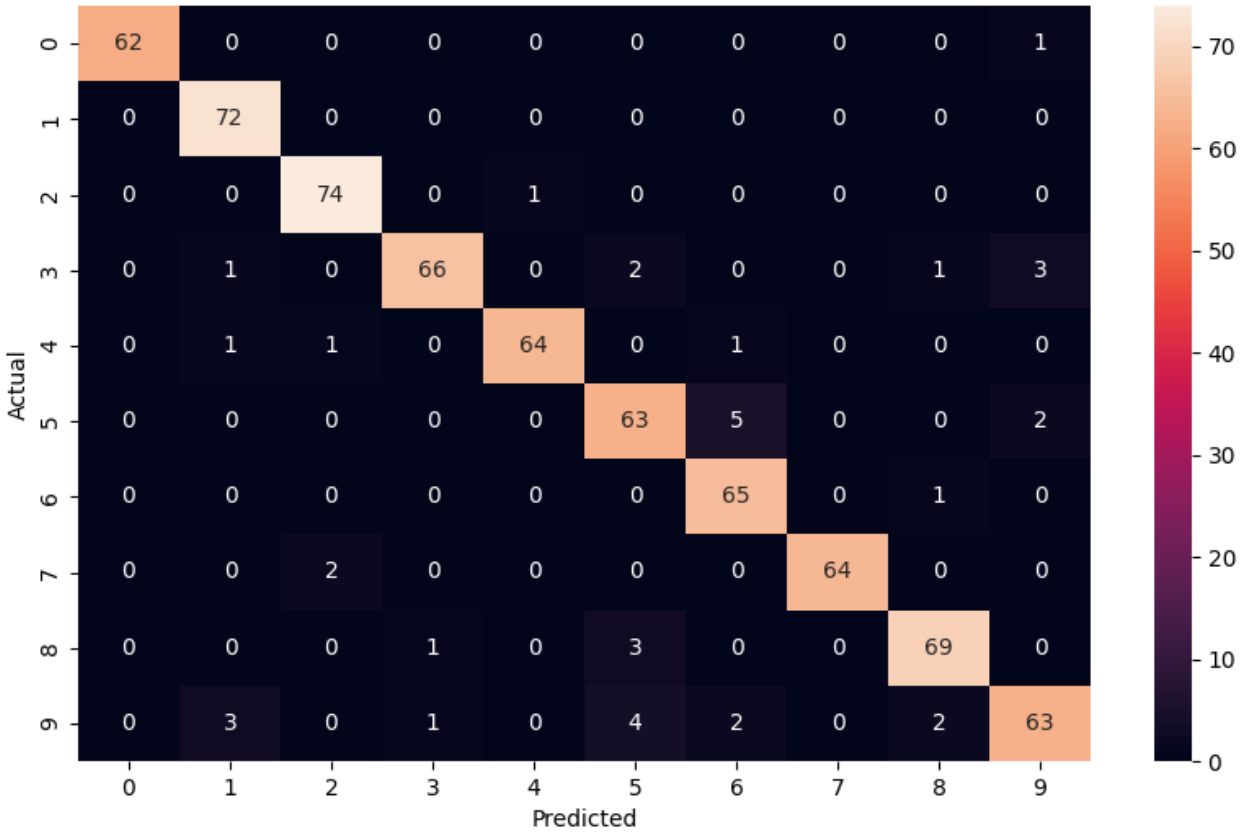
Algorithm 1: Algorytm k najbliższych sąsiadów.

2.5 Wyniki klasyfikatora kNN

Po przeanalizowaniu różnych wartości parametrów k i p, algorytm kNN klasyfikuje cyfry z największą dokładnością na poziomie 94.9% dla pary parametrów $k = 2$ i $p = 3$.



Rysunek 6: Zależność dokładności klasyfikacji od liczby sąsiadów k i parametru p



Rysunek 7: Macierz błędu algorytmu kNN dla $k = 1$ i $p = 3$

2.6 Algorytm Naiwnego Bayesa

Klasyfikator Naiwnego Bayesa jest to klasyfikator probabilistyczny, oparty na twierdzeniu Bayesa. Zakłada on niezależność cech. Naiwny klasyfikator bayesowski predyktuje klasę nowego obiektu w zbiorze na podstawie prawdopodobieństwa warunkowego.

$$p(D_i|X_1, X_2, ..., X_n)$$

gdzie:
D_i - klasa obiektu
$X_1, X_2, ..., X_n$ - cechy obiektu

Prawdopodobieństwo wystąpienia elementu w danym zbiorze

$$p(D_i) = \frac{|D_i|}{N}$$

gdzie:
$|D_i|$ - moc zbioru D_i
N - liczba wszystkich elementów w zbiorze

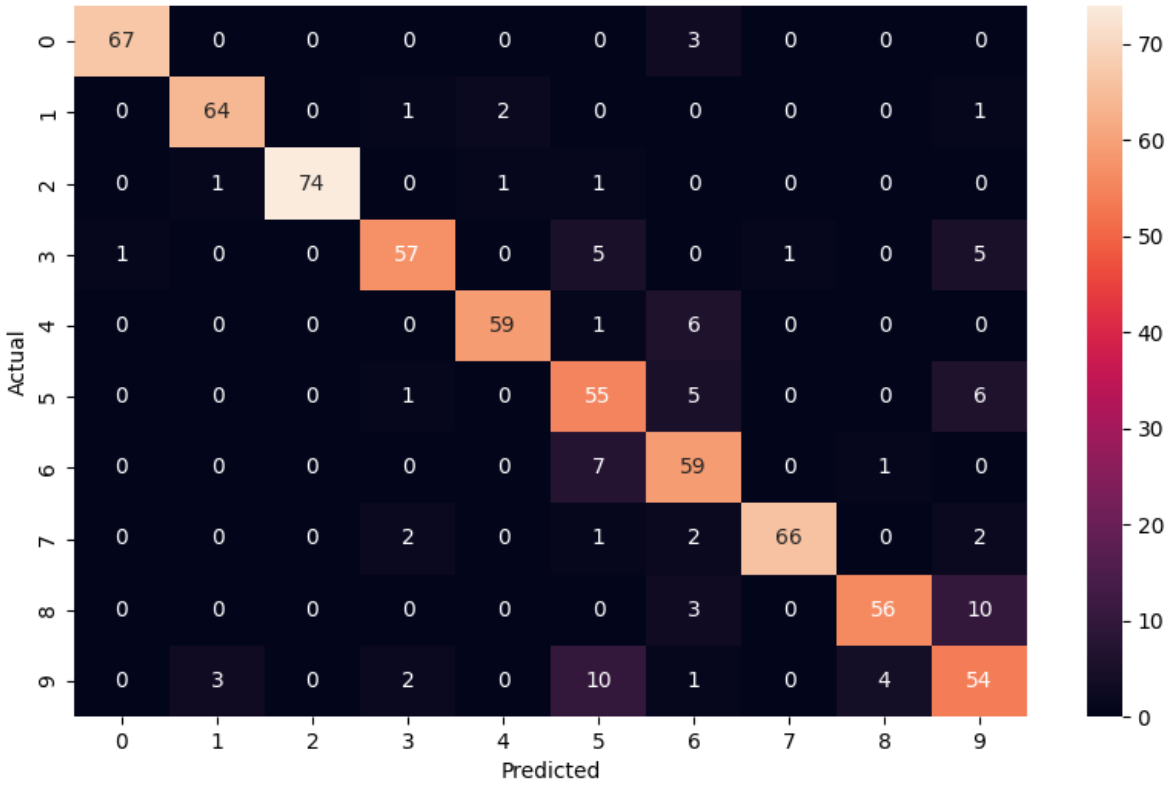
2.7 Pseudokod algorytmu Naiwnego Bayesa

Data: Dane wejściowe: Dane D Nowy punkt NP
Result: Przewidziana klasa dla NP
$probabilities = []$;
foreach c *in* $classes$ **do**
  $probabilities.append$(probability of class c);
end
$chosen\_class = None$;
$max\_probability = -1$;
foreach p, c *in* $zip(probabilities, classes)$ **do**
  if $p > max\_probability$ **then**
    $max\_probability = p$;
    $chosen\_class = c$;
  end
end
return $chosen\_class$

Algorithm 2: Algorytm Naiwnego Bayesa.

2.8 Wyniki klasyfikatora Naiwnego Bayesa

Po przeprowadzeniu kilkunasu testów algorytm Naiwnego Bayesa uzyskuje dokładność na poziomie 88.7%.



Rysunek 8: Macierz błędu algorytmu Naiwnego Bayesa

2.9 Implementacja

1. main.py - główny plik programu
2. knn.py - plik zawierający implementację algorytmu kNN
3. bayes.py - plik zawierający implementację naiwnego klasyfikatora Bayesa
4. utils.py - plik zawierający funkcje pomocnicze (np. wczytywanie danych)
5. test.py - plik zawierający funkcje testujące klasyfikatory

```
# Algorytm kNN
class KNN:
    def __init__(self, k: int = 3, p: int = 2):
        self.k = k
        self.p = p

    @staticmethod
    def minkowski_distance(X1, X2, p) -> float:
```

```
9           return sum([abs((a - b) ** p) for a, b in zip(X1, X2)]) ** (1 /
                p)
10
11      def fit(self, X, Y) -> None:
12          self.X_train = X
13          self.Y_train = Y
14
15      def predict(self, X) -> list[float]:
16          return [self.predict_point(x) for x in X]
17
18      def predict_point(self, x) -> float:
19          distances = [
20              KNN.minkowski_distance(x, x_train, self.p)
21              for x_train in self.X_train
22          ]
23          k_indices = np.argsort(distances)[: self.k]
24          k_nearest_labels = [self.Y_train[i] for i in k_indices]
25          most_common = Counter(k_nearest_labels).most_common(1)
26          return most_common[0][0]
27
28  # Algorytm Naiwnego Bayesa
29  class NaiveBayesClassifier:
30      def fit(self, X, y):
31          self.classes = np.unique(y)
32          self.num_classes = len(self.classes)
33          self.num_features = X.shape[1]
34          self.class_probs = np.zeros(self.num_classes)
35          self.feature_probs = np.zeros((self.num_classes, self.
                num_features))
36          for i, c in enumerate(self.classes):
37              X_c = X[y == c]
38              self.class_probs[i] = len(X_c) / len(X)
39              self.feature_probs[i] = np.mean(X_c, axis=0)
40
41      def predict(self, X):
42          y_pred = np.zeros(X.shape[0], dtype=np.str_)
43          for i, x in enumerate(X):
44              posterior_probs = []
45              for j in range(self.num_classes):
46                  prior = np.log(self.class_probs[j])
47                  likelihood = np.sum(
48                      np.log(self.compute_feature_prob(self.feature_probs[
                            j], x))
49                  )
50                  posterior = prior + likelihood
51                  posterior_probs.append(posterior)
52              y_pred[i] = self.classes[np.argmax(posterior_probs)]
53          return y_pred
54
55      def compute_feature_prob(self, feature_prob, x):
56          epsilon = 1e-9
57          return feature_prob * x + (1 - feature_prob) * (1 - x + epsilon)
58
59  # Funkcja klasyfikujaca narysowana cyfre
60  def predict_digit(img):
```

```
61      img = img.resize((28, 28))
62      img = img.convert("L")
63      img = np.array(img).flatten()
64      img = np.invert(img)
65      img = img / 255
66      x_train, y_train = read_digits("imgs")
67      x_train = x_train / 255
68      model = KNeighborsClassifier(n_neighbors=1, p=3)
69      model.fit(x_train, y_train)
70      y_pred = model.predict(np.array([img]))[0]
71      return y_pred
```

3 Wnioski

3.1 Porównanie wyników klasyfikatorów

W badaniu porównującym algorytm kNN i naiwny klasyfikator bayesowski w rozpoznawaniu pisma odręcznego stwierdzono, że k-NN osiągnął lepsze wyniki i dokładność klasyfikacji niż naiwny klasyfikator bayesowski. Wykorzystano zbiór danych, który został podzielony na dane treningowe (75%) i dane testowe (25%), gdzie każda próbka była reprezentowana jako obraz 28 x 28 pikseli przedstawiający odręczne cyfry. Naiwny klasyfikator bayesowski uzyskał dokładność na poziomie około 88.7%, podczas gdy kNN osiągnął dokładność na poziomie około 94.9%. Wynika z tego, że k-NN jest bardziej skutecznym narzędziem do rozpoznawania cyfr pisma odręcznego niż naiwny klasyfikator bayesowski, zwłaszcza w przypadku większej liczby danych treningowych i większego wymiaru wejściowego. Naiwny klasyfikator bayesowski ma jednak pewne zalety, takie jak krótszy czas potrzebny na dopasowanie i testowanie. W porównaniu do tego, klasyfikator kNN wymagał znacznie więcej czasu. Warto jednak zauważyć, że k-NN może być dobrym wyborem dla mniejszych zbiorów danych i problemów o mniejszej liczbie wymiarów.

3.2 Potencjał rozwoju

Projekt rozpoznawania cyfr przy użyciu uczenia maszynowego ma potencjał rozwoju w wielu różnych obszarach: - Zwiększenie dokładności rozpoznawania: Można podjąć działania mające na celu zwiększenie dokładności rozpoznawania cyfr. Można eksperymentować z różnymi algorytmami uczenia maszynowego, takimi jak głębokie sieci neuronowe, konwolucyjne sieci neuronowe (CNN).

- Rozpoznawanie innych zestawów danych: Oprócz rozpoznawania cyfr można rozważyć rozwinięcie projektu w celu rozpoznawania innych zestawów danych. Można badać możliwość rozpoznawania liter, symboli matematycznych lub innych obiektów.

- Adaptacja do innych języków i pism: Projekty rozpoznawania cyfr można rozszerzyć na inne języki i pisma. Na przykład, można badać możliwość rozpoznawania cyfr w języku chińskim, arabskim, japońskim lub innych językach, które posiadają swoje własne znaki i pismo. Odpowiednie dane treningowe i dostosowanie modelu będą kluczowe w takim przypadku.

4 Pełen kod aplikacji

4.1 main.py

```
import os
import sys
import tkinter as tk
from datetime import datetime

import numpy as np
from PIL import Image, ImageGrab  # Pillow
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier  # scikit-learn

DIRNAME = os.path.dirname(os.path.dirname(__file__))
sys.path.append(DIRNAME)

from digit_recognition.bayes import NaiveBayesClassifier
from digit_recognition.knn import KNN
from digit_recognition.utils import read_digits


class Settings:
    WIDTH = HEIGHT = 300
    FONTSIZE = 20


def predict_digit(img):
    img = img.resize((28, 28))
    img = img.convert("L")
    img = np.array(img).flatten()
    img = np.invert(img)
    img = img / 255
    x_train, y_train = read_digits("imgs")
    x_train = x_train / 255
    model = KNeighborsClassifier(n_neighbors=1, p=3)
    model.fit(x_train, y_train)
    y_pred = model.predict(np.array([img]))[0]
    return y_pred


class App(tk.Tk):
    def __init__(self):
        tk.Tk.__init__(self)
        tk.Tk.title(self, "Rozpoznawanie cyfr")
        self.x = self.y = 0
        self.canvas = tk.Canvas(
            self,
            width=Settings.WIDTH,
            height=Settings.HEIGHT,
            bg="white",
            cursor="cross",
        )
        self.label = tk.Label(
```

```
            self, text="?", font=("Helvetica", Settings.FONTSIZE)
        )
        self.classify_button = tk.Button(
            self, text="Rozpoznaj", command=self.classify_handwriting
        )
        self.clear_button = tk.Button(
            self, text="Wyczysc", command=self.clear_all
        )
        self.save_button = tk.Button(
            self, text="Zapisz", command=self.save_to_file
        )
        self.canvas.grid(
            row=0,
            column=0,
            pady=2,
            sticky=tk.W,
        )
        self.label.grid(row=0, column=1, pady=2, padx=2)
        self.classify_button.grid(row=1, column=1, pady=2, padx=2)
        self.clear_button.grid(row=1, column=0, pady=2)
        self.save_button.grid(row=1, column=2, pady=2)
        self.canvas.bind("<B1-Motion>", self.draw_lines)

    def clear_all(self):
        self.canvas.delete("all")

    def get_canvas_image(self) -> Image:
        x = self.winfo_rootx() + self.canvas.winfo_x()
        y = self.winfo_rooty() + self.canvas.winfo_y()
        x1 = x + self.canvas.winfo_width()
        y1 = y + self.canvas.winfo_height()
        return ImageGrab.grab().crop((x, y, x1, y1))

    def classify_handwriting(self):
        im = self.get_canvas_image()
        digit = predict_digit(im)
        self.label.configure(text=str(digit))

    def save_to_file(self):
        folder = "imgs"
        filename = datetime.today().strftime("%d-%m-%Y %Hh%Mm%Ss") + ".
            png"
        im = self.get_canvas_image()
        im.save(os.path.join(folder, filename))

    def draw_lines(self, event):
        self.x = event.x
        self.y = event.y
        r = 20
        self.canvas.create_oval(
            self.x - r,
            self.y - r,
            self.x + r,
            self.y + r,
            fill="black",
```

```
105             outline="black",
106         )
107
108
109 def main() -> int:
110     app = App()
111     app.mainloop()
112     return 0
113
114
115 if __name__ == "__main__":
116     raise SystemExit(main())
```

4.2 bayes.py

```
1   import numpy as np
2
3
4   class NaiveBayesClassifier:
5     def fit(self, X, y):
6       self.classes = np.unique(y)
7       self.num_classes = len(self.classes)
8       self.num_features = X.shape[1]
9       self.class_probs = np.zeros(self.num_classes)
10      self.feature_probs = np.zeros((self.num_classes, self.num_features
            ))
11      for i, c in enumerate(self.classes):
12        X_c = X[y == c]
13        self.class_probs[i] = len(X_c) / len(X)
14        self.feature_probs[i] = np.mean(X_c, axis=0)
15
16    def predict(self, X):
17      y_pred = np.zeros(X.shape[0], dtype=np.str_)
18      for i, x in enumerate(X):
19        posterior_probs = []
20        for j in range(self.num_classes):
21          prior = np.log(self.class_probs[j])
22          likelihood = np.sum(
23            np.log(self.compute_feature_prob(self.feature_probs[j], x))
24          )
25          posterior = prior + likelihood
26          posterior_probs.append(posterior)
27        y_pred[i] = self.classes[np.argmax(posterior_probs)]
28      return y_pred
29
30    def compute_feature_prob(self, feature_prob, x):
31      epsilon = 1e-9
32      return feature_prob * x + (1 - feature_prob) * (1 - x + epsilon)
33
34    def accuracy_score(self, y_test, y_pred):
35      return sum([y_test[i] == y_pred[i] for i in range(len(y_test))]) /
            len(
36        y_test
```

```
37        )
```

4.3 knn.py

```
1   from collections import Counter
2
3   import numpy as np
4
5
6   class KNN:
7     def __init__(self, k: int = 3, p: int = 2):
8       self.k = k
9       self.p = p
10
11    @staticmethod
12    def minkowski_distance(X1, X2, p) -> float:
13      return sum([abs((a - b) ** p) for a, b in zip(X1, X2)]) ** (1 / p)
14
15    def fit(self, X, Y) -> None:
16      self.X_train = X
17      self.Y_train = Y
18
19    def predict(self, X) -> list[float]:
20      return [self.predict_point(x) for x in X]
21
22    def predict_point(self, x) -> float:
23      distances = [
24        KNN.minkowski_distance(x, x_train, self.p)
25        for x_train in self.X_train
26      ]
27      k_indices = np.argsort(distances)[: self.k]
28      k_nearest_labels = [self.Y_train[i] for i in k_indices]
29      most_common = Counter(k_nearest_labels).most_common(1)
30      return most_common[0][0]
31
32    def accuracy_score(self, y_test, y_pred):
33      return sum([y_test[i] == y_pred[i] for i in range(len(y_test))]) /
            len(
34        y_test
35      )
```

4.4 test.py

```
1   import os
2   import sys
3   from collections import defaultdict
4
5   import matplotlib.pyplot as plt
6   from sklearn.metrics import accuracy_score, confusion_matrix
7   from sklearn.model_selection import train_test_split
8   from sklearn.neighbors import KNeighborsClassifier
```

```
9
10  DIRNAME = os.path.dirname(os.path.dirname(__file__))
11  sys.path.append(DIRNAME)
12
13
14  from digit_recognition.bayes import NaiveBayesClassifier
15  from digit_recognition.knn import KNN
16  from digit_recognition.utils import read_digits
17
18
19  def test_knn(
20    X,
21    y,
22    k_list: list[int],
23    p_list: list[int],
24    filename: str,
25    n_tests: int = 100,
26    plot: bool = False,
27    colors: list[str] = [],
28  ) -> None:
29    m2 = {}
30    for p in p_list:
31      m = defaultdict(float)
32      for _ in range(n_tests):
33        for k in k_list:
34          X_train, X_test, y_train, y_test = train_test_split(
35            X, y, train_size=0.75
36          )
37          model = KNeighborsClassifier(n_neighbors=k, p=p)
38          model.fit(X_train, y_train)
39          y_pred = model.predict(X_test)
40          accuracy = accuracy_score(y_test, y_pred)
41          m[k] += 100 * accuracy / n_tests
42      m2[p] = m
43    if plot:
44      assert len(colors) == len(p_list)
45      for i, (k, v) in enumerate(m2.items()):
46        plt.scatter(k_list, v.values(), label=f"p={k}", color=colors[i])
47      plt.grid()
48      plt.xlabel("k neighbors")
49      plt.ylabel("Accuracy [%]")
50      plt.legend([f"p = {p}" for p in p_list], loc="upper right")
51      plt.xticks(k_list)
52      plt.savefig(filename)
53    else:
54      print(m2)
55
56
57  def test_bayes(X, y, n_tests: int = 100) -> None:
58    acc = 0
59    for _ in range(n_tests):
60      X_train, X_test, y_train, y_test = train_test_split(
61        X, y, train_size=0.75
62      )
63      model = NaiveBayesClassifier()
```

```
64        model.fit(X_train, y_train)
65        y_pred = model.predict(X_test)
66        accuracy = model.accuracy_score(y_test, y_pred)
67        acc += 100 * accuracy / n_tests
68      print(f"accuracy: {round(acc, 3)}")
```

4.5 utils.py

```
1   import os
2
3   import numpy as np
4   from PIL import Image
5
6
7   def read_digits(dirname: str):
8     """Read digits' images from dirname directory.
9     Return a tuple of two lists:
10    - X: list of image pixels (each pixel is a number from
11    0 (white) to 255 (black).
12    - y: list of digit labels from 0 to 9.
13    Dimensions:
14    - X: nsamples * 28 * 28 (each image is resized to 28x28 pixels).
15    - y: nsamples."""
16    X = []
17    y = []
18    for subdir, _, files in os.walk(dirname):
19      for file in files:
20        label = subdir[-1]
21        filepath = subdir + os.sep + file
22        img = Image.open(filepath)
23        img = img.resize((28, 28))
24        img = img.convert("L")
25        img = np.array(img).flatten()
26        img = np.invert(img)
27        X.append(img)
28        y.append(label)
29    return np.array(X), np.array(y)
```