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

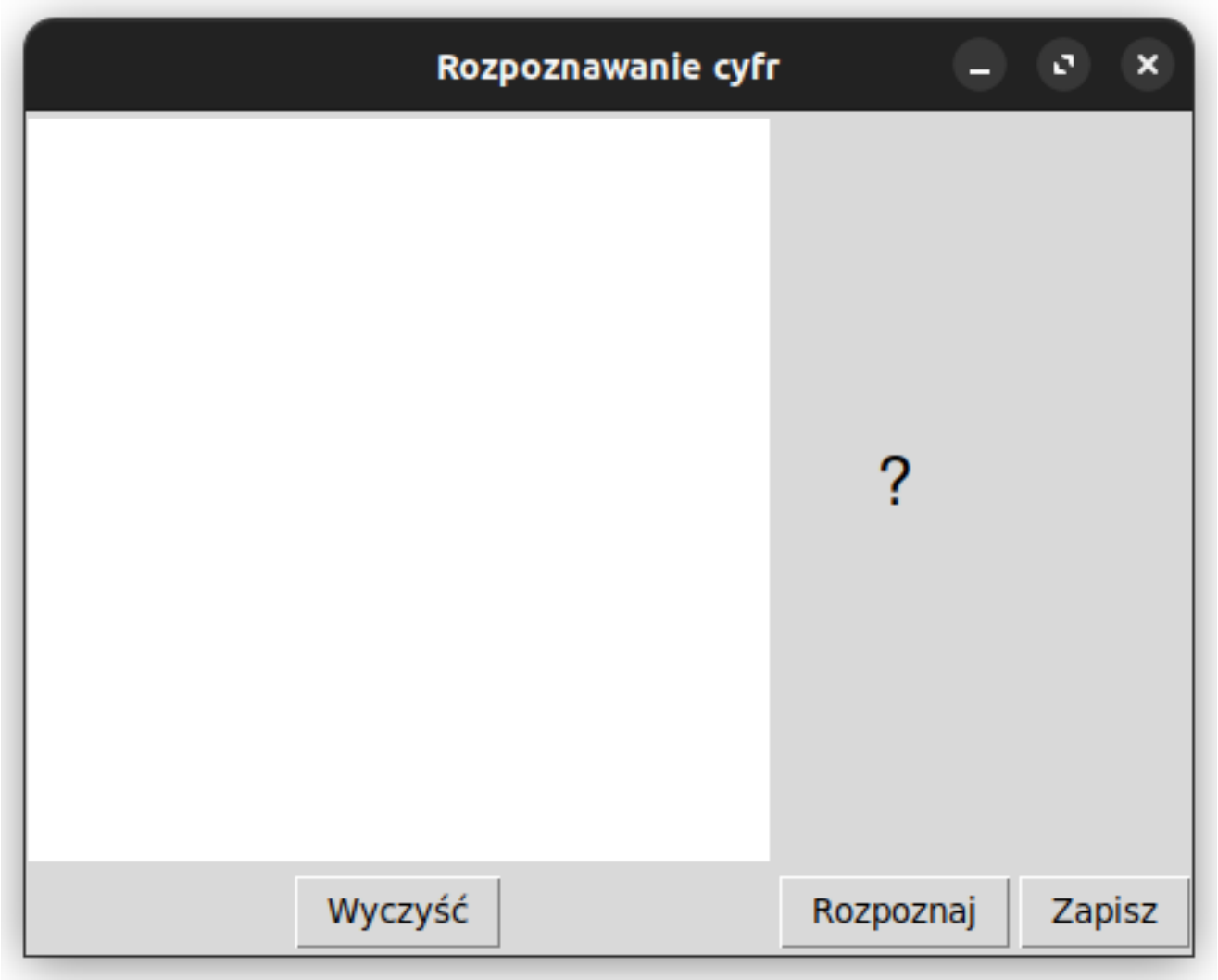
26 czerwca 2023

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:

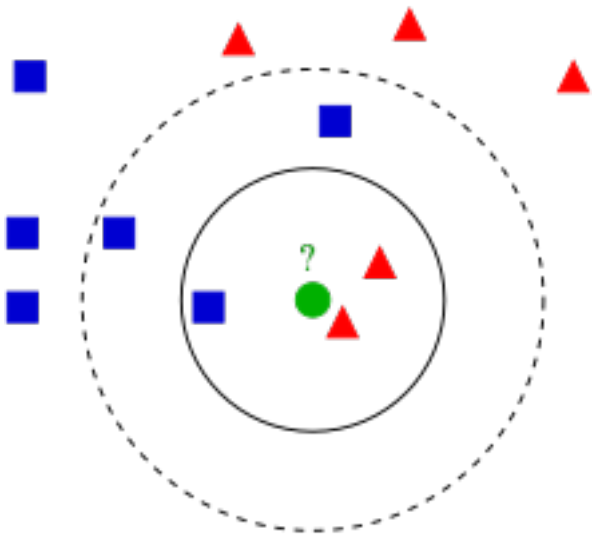
z_i - znormalizowany piksel
x_i - piksel
x - zbiór wszystkich pikseli

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 $p \in \{1, 2, 3\}$.

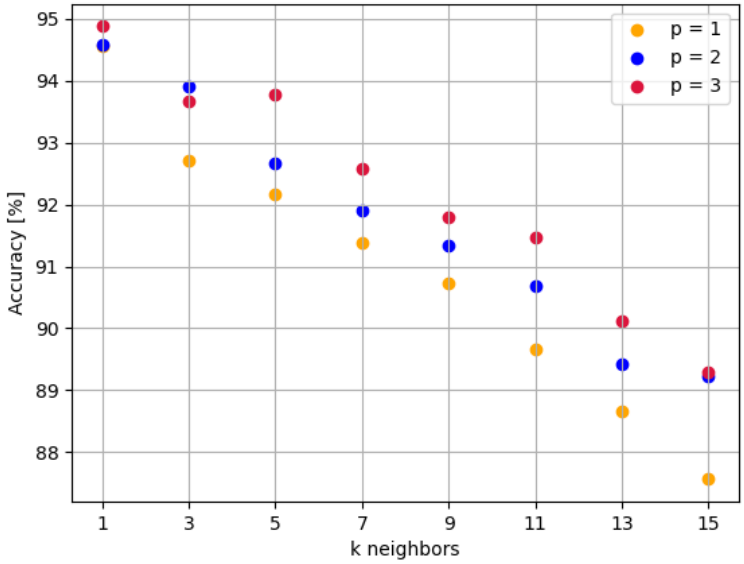
2.4 Pseudokod algorytmu kNN

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*;

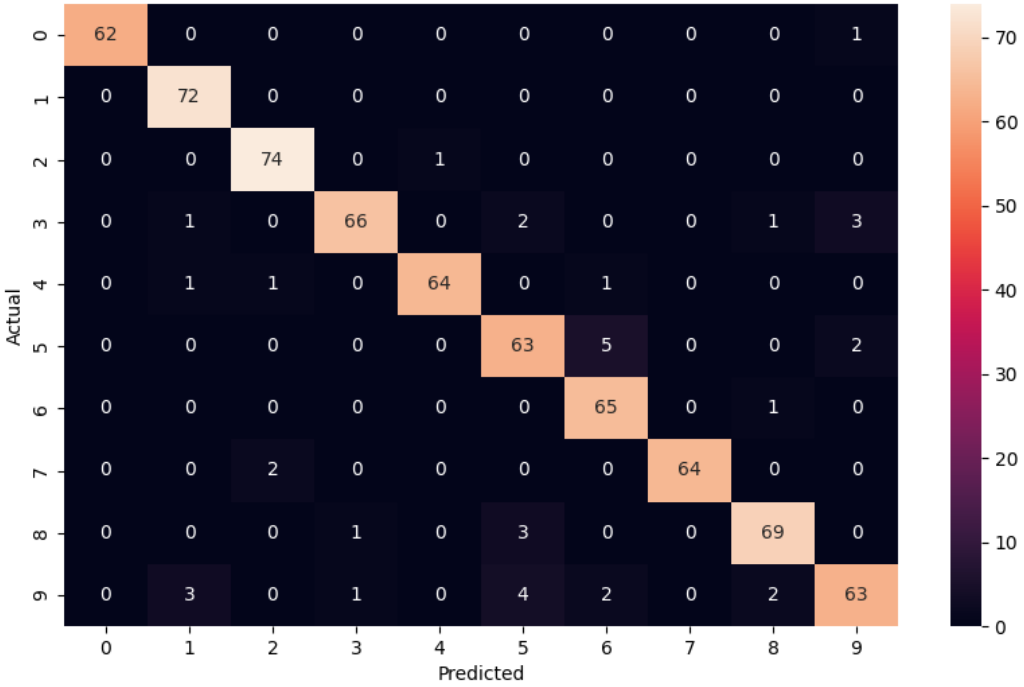
Algorithm 1: Algorytm k najbliższych sąsiadów.

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 $k = 1$ 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

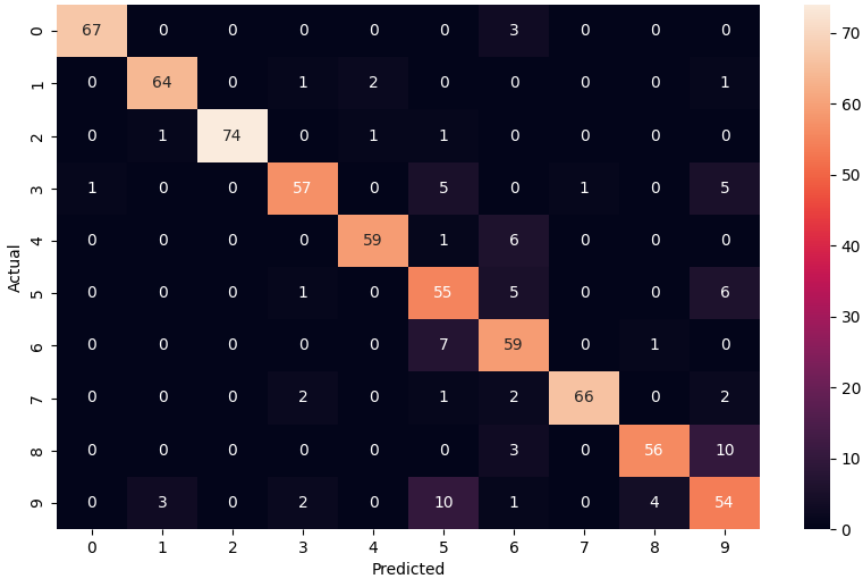
$$p(D_i) = \frac{|D_i|}{N}$$

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

2.7 Pseudokod algorytmu Naiwnego Bayesa

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

```python
# 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:
        return sum([abs((a - b) ** p) for a, b in zip(X1, X2)]) ** (1 / p)

    def fit(self, X, Y) -> None:
        self.X_train = X
        self.Y_train = Y

    def predict(self, X) -> list[float]:
        return [self.predict_point(x) for x in X]

    def predict_point(self, x) -> float:
        distances = [
            KNN.minkowski_distance(x, x_train, self.p)
            for x_train in self.X_train
        ]
        k_indices = np.argsort(distances)[: self.k]
        k_nearest_labels = [self.Y_train[i] for i in k_indices]
        most_common = Counter(k_nearest_labels).most_common(1)
        return most_common[0][0]

# Algorytm Naiwnego Bayesa
class NaiveBayesClassifier:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.num_classes = len(self.classes)
        self.num_features = X.shape[1]
        self.class_probs = np.zeros(self.num_classes)
        self.feature_probs = np.zeros((self.num_classes, self.num_features))
        for i, c in enumerate(self.classes):
            X_c = X[y == c]
            self.class_probs[i] = len(X_c) / len(X)
            self.feature_probs[i] = np.mean(X_c, axis=0)

    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

```
1 Tutaj wklejamy pelen kod.
```