Statystyczne metody rozpoznawania obrazu

Zadanie 5

Bartłomiej Gorzela

Do wykonania zadania posłużyłem się gotowym zbiorem danych ‘Digits’ możliwym do zaimportowania bezpośrednio z biblioteki scikit-learn dostępnej w Python. Zbiór ten posiada 1797 rekordów danych podzielonych na 10 klas (cyfry od 0 do 9).

Przed przystąpieniem do zadań zainportowany dataset musiał zostać odpowiednio przygotowany:

a) **Podział zbioru danych:**
   Całość danych została podzielona na zbiór treningowy i testowy w proporcji 80:20 przy użyciu funkcji train_test_split() z biblioteki sklearn.model_selection. Dzięki temu możliwa będzie ocena skuteczności klasyfikatorów na danych, których wcześniej nie „widziały”.
b) **Standaryzacja danych:**
   Dane wejściowe zostały poddane standaryzacji za pomocą obiektu StandardScaler() z biblioteki sklearn.preprocessing w celu dostosowania danych dla klasyfikatorów kNN oraz SVM. Proces ten przekształca każdą cechę w taki sposób, aby miała średnią 0 i odchylenie standardowe 1.

5.1. Użycie klasyfikatora kNN z różnymi miarami odległości i różną liczbą sąsiadów.

W ramach tego zadania zastosowałem klasyfikator kNN do rozpoznawania cyfr. Celem było porównanie skuteczności klasyfikatora przy różnych parametrach:

a) liczba sąsiadów: amount_of_neighbours = {1, 3, 5, 7}
b) miara odległości (metrics):
    1. euclidean – klasyczna odległość euklidesowa
    2. chebyshev – maksymalna różnica na jednej osi
    3. manhattan – suma bezwzględnych różnic

Na uprzednio przeskalowanych danych treningowych za pomocą StandardScaler(), dokonałem treningu modelu kNN dla każdej z wymienionych miar odległości i liczby sądsiadów. Następnie dokonałem predykcji na zbiorze testowym oraz oceniłem skuteczność modelu za pomocą wskaźnika ‘accuracy’ uzyskanego za pomoca funkcji accuracy_score() z biblioteki sklearn.metrics. Uzyskałem następujące wyniki:

```
Zadanie 5.1: Użycie klasyfikatora kNN z różnymi miarami odległości i różną liczbą sąsiadów:
----------------------------------------------------------------------
k = 1, metric = euclidean -> Accuracy: 0.969 (96.9%)
k = 1, metric = chebyshev -> Accuracy: 0.933 (93.3%)
k = 1, metric = manhattan -> Accuracy: 0.978 (97.8%)
----------------------------------------------------------------------
k = 3, metric = euclidean -> Accuracy: 0.981 (98.1%)
k = 3, metric = chebyshev -> Accuracy: 0.922 (92.2%)
k = 3, metric = manhattan -> Accuracy: 0.975 (97.5%)
----------------------------------------------------------------------
k = 5, metric = euclidean -> Accuracy: 0.981 (98.1%)
k = 5, metric = chebyshev -> Accuracy: 0.911 (91.1%)
k = 5, metric = manhattan -> Accuracy: 0.981 (98.1%)
----------------------------------------------------------------------
k = 7, metric = euclidean -> Accuracy: 0.981 (98.1%)
k = 7, metric = chebyshev -> Accuracy: 0.917 (91.7%)
k = 7, metric = manhattan -> Accuracy: 0.975 (97.5%)
----------------------------------------------------------------------
```

Obserwacje:

Uzyskane wyniki pokazują, że klasyfikator kNN bardzo dobrze radzi sobie z rozpoznawaniem cyfr w zbiorze ‘Digits’, osiągając dokładność sięgającą nawet 98,1% dla metryk euklidesowej i manhattan przy liczbach sąsiadów $k = \{3, 5, 7\}$. Dla każdej z tych wartości liczby sąsiadów zastosowanie metryki euklidesowej lub manhattan dawało zbliżone rezultaty (różnice w granicach ±1%), co sugeruje, że może to być skuteczne i stabilne rozwiązanie dla klasyfikacji wieloklasowej w tym przypadku.

Z kolei metryka Chebysheva wypadała zauważalnie słabiej w każdej konfiguracji (dokładność w zakresie 91,1% – 93,3%), co może oznaczać, że sposób obliczania odległości charakterystyczny dla tej metryki nie jest optymalny dla struktury danych zawartej w tym zbiorze.

5.2. Użycie klasyfikatora SVM dla różnych parametrów jądra.

W ramach tego zadania zastosowałem klasyfikator SVM do rozpoznawania cyfr. Celem było porównanie skuteczności klasyfikatora przy użyciu różnych parametrów jądra: { 'linear', 'poly', 'rbf', 'sigmoid'}.

Analogicznie jak w przypadku poprzedniego podpunktu, na uprzednio przeskalowanych danych treningowych za pomocą StandardScaler(), dokonałem treningu modelu SVM dla każdej z wymienionych funkcji jądra. Następnie dokonałem predykcji na zbiorze testowym oraz oceniłem skuteczność modelu za pomocą wskaźnika ‘accuracy’ uzyskanego za pomoca funkcji accuracy_score() z biblioteki sklearn.metrics. Uzyskałem następujące wyniki:

```
Zadanie 5.2: Użycie klasyfikatora SVM dla różnych parametrów jądra:
----------------------------------------------------------------------
Kernel = linear -> Accuracy: 0.972 (97.2%)
Kernel = poly -> Accuracy: 0.953 (95.3%)
Kernel = rbf -> Accuracy: 0.975 (97.5%)
Kernel = sigmoid -> Accuracy: 0.950 (95.0%)
----------------------------------------------------------------------
```

Obserwacje:

Uzyskane wyniki pokazują, że klasyfikator SVM bardzo dobrze poradził sobie z klasyfikacją cyfr w zbiorze ‘Digits’, osiągając wysoką skuteczność niezależnie od zastosowanego parametru jądra. Najlepszy wynik uzyskano dla jądra ‘rbf’, które osiągnęło dokładność 97,5%, tuż przed jądrem liniowym, które dało 97,2%. Jądra wielomianowe oraz sigmoidalne również zapewniły dobre rezultaty, odpowiednio 95,3% i 95,0%, co świadczy o ogólnej efektywności algorytmu SVM w tym zadaniu dla użytego datasetu.

Warto jednak zauważyć, że uzyskane dokładności klasyfikacji dla SVM były nieco niższe niż najlepsze wyniki osiągnięte przez klasyfikator kNN (który przy odpowiedniej konfiguracji osiągał dokładność do 98,1%). Może to świadczyć o tym, że w przypadku tego konkretnego zbioru danych klasyfikator kNN mógłby być lepszym wyborem, niż SVM.

Podsumowując, wszystkie konfiguracje SVM okazały się skuteczne, a różnice w dokładności są niewielkie (dokładność w zakresie 95,0% – 97,5%),. Niemniej jednak, w tym konkretnym przypadku to kNN wypadł minimalnie lepiej pod względem precyzji klasyfikacji.

Kod źródłowy:

Bartłomiej_Gorzela_Zadanie_5.py U | Requirements.txt U

StatisticalPatternRecognitionMethods > Task5 > Bartłomiej_Gorzela_Zadanie_5.py > ...

```python
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

#Wczytanie zbioru digits (cyfry 0-9)
digits = load_digits()
digits_data, digits_target = digits.data, digits.target

#Podział na zbiór treningowy i testowy
digits_data_train, digits_data_test, digits_target_train, digits_target_test = train_test_split(digits_data, digits_target, test_size=0.2, random_state=50)

#Standaryzacja cech
scaler = StandardScaler()
digits_data_train_scaled = scaler.fit_transform(digits_data_train)
digits_data_test_scaled = scaler.transform(digits_data_test)

#Zadanie 5.1
print("Zadanie 5.1: Użycie klasyfikatora kNN z różnymi miarami odległości i różną liczbą sąsiadów:")
amount_of_neighbours = [1, 3, 5, 7]
metrics = ['euclidean','chebyshev', 'manhattan']
print("--------------------------------------------------------------------------------")
for i in amount_of_neighbours:
    for metric in metrics:
        kNN = KNeighborsClassifier(n_neighbors = i, metric = metric)
        kNN.fit(digits_data_train_scaled, digits_target_train)
        y_predicted = kNN.predict(digits_data_test_scaled)
        acc = accuracy_score(digits_target_test, y_predicted)
        print(f"k = {i}, metric = {metric} -> Accuracy: {acc:.3f} ({acc*100:.1f}%)")
    print("--------------------------------------------------------------------------------")

#Zadanie 5.2
print("\nZadanie 5.2: Użycie klasyfikatora SVM dla różnych parametrów jądra:")
kernels = ['linear', 'poly', 'rbf', 'sigmoid']
print("--------------------------------------------------------------------------------")
for kernel in kernels:
    SVM = SVC(kernel = kernel)
    SVM.fit(digits_data_train_scaled, digits_target_train)
    y_predicted = SVM.predict(digits_data_test_scaled)
    acc = accuracy_score(digits_target_test, y_predicted)
    print(f"Kernel = {kernel} -> Accuracy: {acc:.3f} ({acc*100:.1f}%)")
print("--------------------------------------------------------------------------------")
```