Systemy Inteligentne

Sprawozdanie

Analiza głównych składowych
Analiza regresji
Algorytm k - nn

Laboratorium czwartek godzina 8:00

Paweł Lurka
Paweł Chmielarski
Informatyka IV rok 2019/2020, IO1

1. Opis struktury danych

Wybrana struktura danych zawiera funkcje wyodrębnione z bazy danych Messidor, aby przewidzieć, czy badany obraz wykazuje oznaki retinopatii cukrzycowej.

Wszystkie cechy reprezentują wykrytą zmianę patologiczną, cechę opisową części ciała lub deskryptor obrazu.

Informacje o atrybutach:

0) Wynik binarny oceny jakości. 0 = zła jakość 1 = wystarczająca jakość.

1) Binarny wynik wstępnego badania przesiewowego, gdzie 1 oznacza ciężką nieprawidłowość siatkówki, a 0 jej brak.

2-7) Wyniki wykrywania mikrotętniaków. Każda wartość funkcji oznacza liczbę mikrotętniaków znalezionych na poziomach ufności odpowiednio alfa = 0,5,. . . ,1.

8-15) zawierają te same informacje co 2-7) dla wysięków. Jednak, ponieważ wysięk jest reprezentowany przez zestaw punktów, a nie liczbę pikseli składających się na zmiany patologiczne, cechy te są znormalizowane poprzez podzielenie liczby zmian patologicznych przez średnicę Obszaru Zainteresowania w celu skompensowania innego rozmiaru obrazu

16) Odległość euklidesowa między środkiem plamki żółtej i środkiem tarczy nerwu wzrokowego w celu zapewnienia ważnych informacji w odniesieniu do stanu pacjenta. Ta cecha jest również znormalizowana ze średnicą Obszaru Zainteresowania.

17) Średnica tarczy nerwu wzrokowego.

18) Wynik binarny klasyfikacji opartej na AM / FM (amplitude-modulation-frequency-modulation).

19) Etykieta klasy. 1 = zawiera oznaki Retinopatii Cukrzycowej, 0 = brak oznak Retinopatii Cukrzycowej.

2. Analiza Głównych Składowych

- # Wstęp teoretyczny

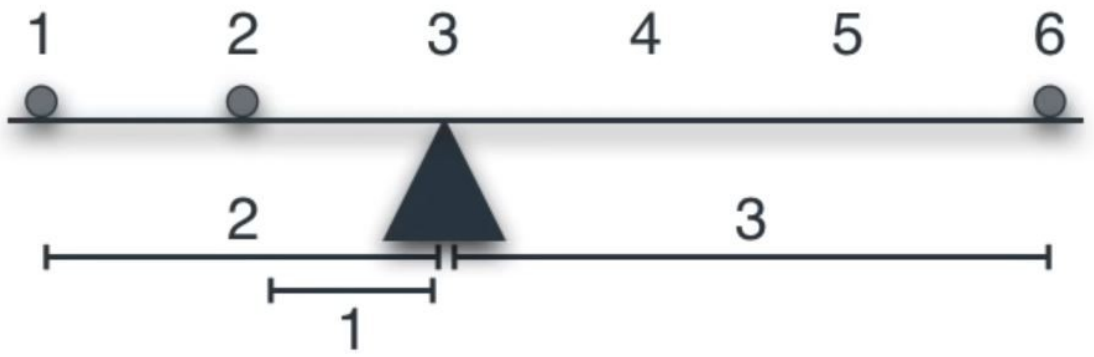
Analiza Głównych Składowych jest jednym z najważniejszych narzędzi stosowanym do redukcji wymiarów.

Polega ona na rzutowaniu punktów z przestrzeni wielowymiarowej na mniejszą ilość wymiarów przy utracie jak najmniejszej ilości informacji, używając wariancji, kowariancji, wektorów własnych i wartości własnych.

Aby dobrze przedstawić działanie Analizy Głównych Składowych należy najpierw wyjaśnić pojęcia z których owo narzędzie korzysta.

Wariancja pokazuje rozproszenie danych w obrębie jednego wymiaru (atrybutu).
Aby móc ją obliczyć musimy najpierw znaleźć środek naszych danych, wyciągając średnią arytmetyczną ze wszystkich punktów a następnie dodać do siebie kwadraty odległości każdego punktu od środka danych i podzielić całość przez ilość punktów. Wariancja przyjmuje jedynie wartości dodatnie. Obliczana jest ze wzoru:

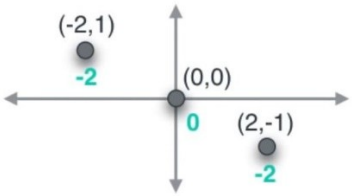
$$\sigma^2 = \frac{(a_1 - \acute{a})^2 + (a_2 - \acute{a})^2 + \ldots + (a_n - \acute{a})^2}{n}$$

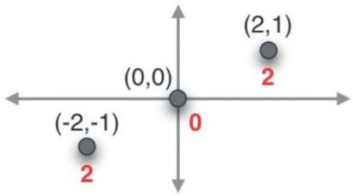


$$\text{Wariancja} = \frac{2^2 + 1^2 + 3^2}{3} = \frac{14}{3}$$

Kowariancja jest miarą zależności liniowej między dwiema zmiennymi, i może przyjmować wartości zarówno ujemne jak i dodatnie co pozwala np. rozróżnić zestawy danych których wariancja na obu osiach jest jednakowa. Im mocniejsza zależność, tym wartość kowariancji bardziej odległa od zera. Obliczana jest ze wzoru:

$$cov(X,Y)=\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})$$





Kowariancja $= \frac{(-2)+0+(-2)}{3}=\frac{-4}{3}$

Kowariancja $= \frac{2+0+2}{3}=\frac{4}{3}$

Macierz kowariancji to transformacja liniowa przedstawiająca kowariancję poszczególnych par zmiennych między sobą, której przekątną są wartości wariancji danej zmiennej. Jest to macierz symetryczna co oznacza, że zarówno nad przekątną jak i pod nią wartości są identyczne.

Cov	a_1	a_2	a_3	a_4
a_1	$Var(a_1)$	$Cov(a_1,a_2)$	$Cov(a_1,a_3)$	$Cov(a_1,a_4)$
a_2	$Cov(a_1,a_2)$	$Var(a_2)$	$Cov(a_1,a_3)$	$Cov(a_2,a_4)$
a_3	$Cov(a_1,a_3)$	$Cov(a_2,a_3)$	$Var(a_3)$	$Cov(a_3,a_4)$
a_4	$Cov(a_1,a_4)$	$Cov(a_2,a_4)$	$Cov(a_3,a_4)$	$Var(a_4)$

Wektory własne to takie wektory, które po przemnożeniu przez macierz kowariancji mają ten sam punkt przyłożenia, kierunek i zwrot ale inną wartość, a więc zachowują się jak przemnożone przez wartość skalarną (**Wartość własną**) co przedstawia poniższy wzór.

$$Av=\lambda v$$

Gdzie:
A – macierz kowariancji
v – wektor własny
λ – wartość własna

Aby zastosować **Analizę Głównych Składowych** musimy najpierw obliczyć macierz kowariancji zgodnie ze wzorami podanymi powyżej.

Przykładowa macierz:

$$A=\begin{pmatrix} 2 & -3 & 1 \\ 3 & 1 & 3 \\ -5 & 2 & -4 \end{pmatrix}$$

Potem trzeba wyprowadzić wzór:

$$Av=\lambda v$$
$$Av-\lambda v=0$$
$$(A-\lambda E)v=0$$
$$det(A-\lambda E)=0$$

Następnie należy obliczyć wyznacznik tej macierzy i zastosować go do powyższego wzoru:

$$det(A-\lambda E)=\begin{vmatrix} 2-\lambda & -3 & 1 \\ 3 & 1-\lambda & 3 \\ -5 & 2 & -4-\lambda \end{vmatrix}$$

$$-\lambda^3-\lambda^2+2\lambda=0$$

Wartościami własnymi są pierwiastki powyższego równania czyli odpowiednio $\lambda=0, \lambda=1, \lambda=-2$ a wektory własne wyliczamy z odpowiadających im równań liniowych.

$$2x_1-3x_2+x_3=\lambda$$
$$3x_1+x_2+3x_3=\lambda$$
$$-5x_1+2x_2-4x_3=\lambda$$

Rozwiązując powyższe równanie za każdym razem podstawiając pod λ jedną z wartości wyliczonych dla λ powyżej dostajemy 3 wektory własne, z których wybieramy te które mają największe wartości własne.

Wybrane wektory własne tworzą macierz W, którą następnie należy przemnożyć przez tabelę początkową w której każdy punkt opisany był wszystkimi atrybutami dzięki czemu dostajemy te same punkty opisane taką ilością zmiennych ile wektorów własnych wybraliśmy w poprzednim kroku.

Ciekawostką jest tutaj, że wektory własne są względem siebie prostopadłe i nie jest to tylko przypadek, ponieważ wektory własne macierzy symetrycznej zawsze zachowują się w ten sposób, a badając kowariancję między atrybutami zawsze utworzymy symetryczną macierz kowariancji.

- # Badania

Na początek plik z danymi został skopiowany do katalogu z programem. Następnie za pomocą biblioteki pandas został wczytany do zmiennej dataset.

```
import pandas as pd
dataset = pd.read_csv('messidor_features.csv', header=None)
```

Kolejnym krokiem było rozbicie danych na wejściowe (badane parametry) i wynikowe (czy występuje retinopatia).

```
# rozdzielamy dane na: X - input data, Y - output data (do regresji)
X, Y = dataset.iloc[:, 0:19].values, dataset.iloc[:, 19].values
```

Następnie wektory zostały przeskalowane przez narzędzie StandardScaler biblioteki sklearn a następnie za pomocą biblioteki numpy (np) wyliczona została macierz kowariancji.

```
x_std = StandardScaler().fit_transform(X)

features = x_std.T
covariance_matrix = np.cov(features)
```

Za pomocą biblioteki numpy obliczone zostały również wektory i własności własne na podstawie macierzy kowariancji.

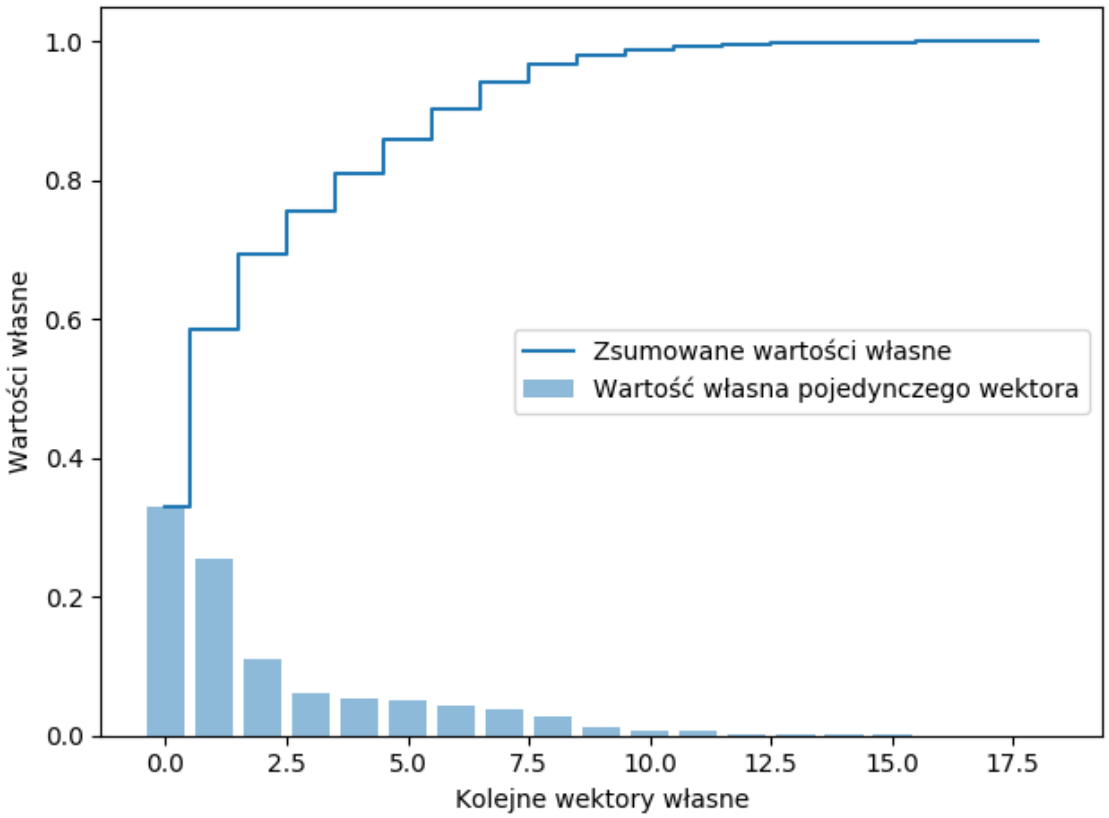
```
eig_vals, eig_vecs = np.linalg.eig(covariance_matrix)
```

Wartości własne wektorów zostały posegregowane według wielkości malejąco, a także obliczono dokładność odzwierciedlenia danych każdego z wektorów własnych.

```
tot = sum(eig_vals)
var_exp = [(i / tot) for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print("Dla 3 najwiekszych wartosci wlasnych uzyskujemy dokładnosc: ", cum_var_exp[2]*100, "%")

plt.bar(range(0, 19), var_exp, alpha=0.5,
        align='center', label='Wartość własna pojedynczego wektora')
plt.step(range(0, 19), cum_var_exp, where='mid',
         label='Zsumowane wartości własne')
plt.ylabel('Wartości własne')
plt.xlabel('Kolejne wektory własne')
plt.legend(loc='best')
plt.show()
```

Można dzięki temu wyświetlić te dane w wykresie.



Pozwoliło to na odpowiednie dobranie wektorów własnych aby opisywane dane nie straciły zbyt wiele na dokładności.

Wektory i ich wartości własne zostały dobrane w pary a następnie do utworzenia macierzy W przekazane zostały 3 o największych wartościach składające się na 69% dokładności.

```
eigen_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in range(len(eig_vals))]
eigen_pairs.sort(key=lambda k: k[0], reverse=True)
print("eigen_pairs: ", eigen_pairs)

w = np.hstack((eigen_pairs[0][1][:, np.newaxis], eigen_pairs[1][1][:, np.newaxis], eigen_pairs[2][1][:, np.newaxis]))
print('Matrix W:\n', w)
```

Ostatnim krokiem było wymnożenie macierzy W z wektorami z początkowego zbioru danych i wyświetlenie tych danych po redukcji wymiarów.

```
x_pca = x_std.dot(w)
print('Matrix x_pca:\n', x_pca)
```

Dane przed redukcją:

x_std: [[5.90538595e-02 2.98212905e-01 -6.41486299e-01 -6.18782253e-01
-5.76462567e-01 -6.30028889e-01 -5.51116046e-01 -4.73744516e-01
-2.42917347e-01 -2.46002983e-01 -2.96966184e-01 -2.71509102e-01
-2.18324265e-01 -1.94409416e-01 -2.05124486e-01 -1.86169038e-01
-1.29476283e+00 -4.68655678e-01 1.40504812e+00]
[5.90538595e-02 2.98212905e-01 -5.63391135e-01 -5.35777956e-01
-5.76462567e-01 -6.77409696e-01 -6.53676203e-01 -5.39991623e-01
-1.09249944e-01 3.29723203e-02 -4.65224463e-01 -4.08592718e-01
-2.24256126e-01 -1.97211625e-01 -2.05174669e-01 -1.86280844e-01
-8.21678582e-02 2.00605415e+00 -7.11719399e-01]
[5.90538595e-02 2.98212905e-01 9.20416985e-01 9.58299396e-01
1.04666546e+00 1.02829936e+00 9.36006226e-01 7.84950517e-01
-1.41383008e-01 2.27196325e-01 3.44462951e-01 7.69036929e-01
3.35537729e-01 1.52330097e-01 -1.10043399e-01 -1.64808490e-01
2.74282645e-01 1.12151640e+00 -7.11719399e-01]
...
[5.90538595e-02 -3.35330894e+00 4.12798418e-01 4.60273612e-01
5.64113881e-01 6.01872093e-01 7.30885912e-01 7.84950517e-01
-5.75348047e-01 -4.22400863e-01 -6.00325820e-01 -4.33156414e-01
-2.21308923e-01 -2.00905317e-01 -2.14967815e-01 -2.08099798e-01
1.33436273e+00 1.19371332e+00 -7.11719399e-01]
[5.90538595e-02 2.98212905e-01 2.23225969e-02 -3.77521717e-02
-2.69384292e-01 -4.40505661e-01 -8.07516438e-01 -9.37474264e-01
-4.03198507e-01 -4.85477485e-01 -3.42803857e-01 -1.93526820e-01
1.11955245e-03 1.08425986e-01 3.86140169e-01 7.67877689e-01
-1.32796165e+00 -9.70784626e-02 1.40504812e+00]
[5.90538595e-02 2.98212905e-01 -1.22720003e+00 -1.24131448e+00
-1.23448744e+00 -1.19859857e+00 -1.11519691e+00 -1.06996848e+00
9.11204558e-02 -7.43571505e-01 -6.01429122e-01 -4.56428264e-01
-2.17298101e-01 -1.92887649e-01 -2.14967815e-01 -2.08099798e-01
1.17603538e+00 -1.08570243e+00 -7.11719399e-01]]

Dane po redukcji:

x_pca:
[[-1.57025412 0.44753359 -0.25956574]
[-1.16330534 0.8673969 -0.61837035]
[2.31848392 -0.57361908 0.68610833]
...
[1.6443359 1.21276547 -0.58158556]
[-0.99596628 -0.30896048 -0.98835013]
[-2.50863779 1.18508175 -1.08283931]]

3. Analiza regresji

Regresja jest jedną z podstawowych technik analizy danych statystycznych. Polega ona na przewidywaniu wyniku pewnej zmiennej na podstawie znanych wartości innych zmiennych. Analiza regresji tworzy funkcję matematyczną pomiędzy zmiennymi niezależnymi (predyktorami) a zmienną zależną. Istnieje wiele odmian regresji: liniowa, nieliniowa, logistyczna, krokowa, porządkowa. Jednak na potrzeby przeprowadzanego tu badania opisana zostanie regresja liniowa.

Realizując regresje liniową dla pewnej zmiennej zależnej y na zbiorze zmiennych niezależnych $x=(x_1,\ldots,x_r)$ gdzie r jest liczbą predyktorów, przyjmuje się liniową zależność pomiędzy y i $x: y=(\beta_0+\beta_1 x_1+\ldots+\beta_r x_r+\varepsilon)$. Jest to równanie regresji, gdzie $\beta_0,\beta_1,\ldots,\beta_r$ są współczynnikami regresji, a ε jest błędem losowym.

Regresja liniowa oblicza prognozowane wagi dla poszczególnych współczynników regresji oznaczane za pomocą $b_0,b_1,\ldots,b_r$. Definiują one oszacowaną funkcje regresji $f(x)=b_0+b_1 x_1+\ldots+b_r x_r$. Funkcja ta powinna wystarczająco dobrze wychwytywać zależności między danymi wejściowymi i wyjściowymi.

Oszacowana wartość funkcji $f(x_i)$ dla każdej obserwacji $i=1,\ldots,n$, powinna być jak najbardziej zbliżona do odpowiadającej jej rzeczywistej wartości y_i. Dla wszystkich obserwacji $i=1,\ldots,n$ oblicza się różnice $y_i-f(x_i)$ z której otrzymujemy resztę. Regresja polega na określeniu najlepszych przewidywanych wag, czyli wag odpowiadających najmniejszym resztom.

Aby otrzymać najlepsze wagi, minimalizuje się sumę kwadratów reszt (SSR) dla wszystkich obserwacji $i=1,\ldots,n: SSR=min\sum_i (y_i-f(x_i))^2$. Jest to klasyczna metoda najmniejszych kwadratów.

Badania

Dla analizowanego zbioru danych retinopatii cukrzycowej, metodę regresji liniowej zaimplementowano z użyciem biblioteki scikit-learn. W tym celu zaimportowano:

```
from sklearn.linear_model import LinearRegression
```

Regresja będzie obliczona dla wcześniej zredukowanych danych za pomocą analizy głównych składowych. Tablica `x_pca` zawiera 1151 obserwacji, z których każda jest reprezentowana przez 3 predyktory:

```
x_pca:
 [[-1.57025412  0.44753359 -0.25956574]
 [-1.16330534  0.8673969  -0.61837035]
 [ 2.31848392 -0.57361908  0.68610833]
```

```
 ...
 [ 1.6443359   1.21276547 -0.58158556]
 [-0.99596628 -0.30896048 -0.98835013]
 [-2.50863779  1.18508175 -1.08283931]]
```

Wyjściowe dane zależne reprezentuje tablica `Y`, która zawierająca wartości 0 lub 1. Wartości te odpowiadają kolejnym wektorom w tabeli `x_pca`.

```
Y: [0 0 1 ... 0 1 0]
```

Następnie utworzono model regresji liniowej do którego wprowadzono powyższe dane:

```
model = LinearRegression().fit(x_pca, Y)
```

Uzyskano następujące współczynniki regresji wykorzystując metody `model.intercept_` oraz `model.coef_` :

```
b0 = 0.5308427454387489
b1,...,br: [ 0.04147849 -0.03618781 -0.00218987]
```

W kolejnym kroku za pomocą utworzonego modelu obliczono przewidywane dane wyjściowe dla podanej tablicy `x_pca`.

```
y_pred = model.predict(x_pca)
y_pred: [0.45008413 0.45255555 0.64626548 ... 0.55643358
0.50287653 0.38627401]
```

Uzyskane wyniki zaokrąglono do liczb całkowitych, a następnie porównano z oryginalnymi wartościami z tabeli `Y` w celu obliczenia procentowego pokrycia wyników.

```
y_pred_round = np.round(y_pred)
print('rounded predicted response:', y_pred_round)
print('oryginal response:', Y)
all_elem = len(Y)
match_elem = 0
for i in range(all_elem):
    if(y_pred_round[i] == Y[i]):
        match_elem += 1
print("percent match: ", 100*(match_elem/all_elem), "%" )
```

```
rounded predicted response:[0 0 1 ... 1 1 0]
oryginal response: [0 0 1 ... 0 1 0]
percent match:  58.905299739357076 %
```

Uzyskano pokrycie wyników ~59%, co jest rozsądnym wynikiem biorąc pod uwagę rodzaj analizowanych danych, przeprowadzoną analizę głównych składowych oraz zastosowaną regresję. Spadek dokładności wynosi tu ~10% względem stanu po analizie głównych składowych gdzie wynosiła ona ~69%.

4. Algorytm k – nn

Algorytm k - nn (ang. k nearest neighbours) jest nieparametryczną metodą używaną w statystyce i wykorzystywaną do rozwiązywania problemów klasyfikacji i regresji. Jest to algorytm nadzorowanego uczenia maszynowego, co oznacza że polega on na zbiorze danych posiadających cechy warunkowe i przyporządkowane im klasy decyzyjne. Jego celem jest wyznaczenie funkcji która umożliwia wygenerowanie klas decyzyjnych dla nowo otrzymanych nieoznakowanych danych.

Algorytm oblicza odległość nowo otrzymanej obserwacji (bez przypisanej klasy decyzyjnej) do wszystkich innych punktów które zostały podane w danych uczących (treningowych). Może to być odległość dowolnego rodzaju np. euklidesowa, kątowa. Odległość euklidesową obliczamy:

$$d(x,y)=\sqrt{\sum_i (x_i-y_i)^2}$$

Gdzie x, y to punkty w przestrzeni, a i to liczba wymiarów. Następnie wybierane jest K najbliższych punktów danych, gdzie K może być dowolną liczbą całkowitą. Na koniec przypisuje się nowej obserwacji klasę decyzyjną (dana wyjściowa) do której należy większość K najbliższych punktów.

Badania

Dla analizowanego zbioru danych retinopatii cukrzycowej, algorytm k – nn zaimplementowano z użyciem biblioteki scikit-learn. W tym celu zaimportowano:

```
from sklearn.neighbors import KneighborsClassifier
from sklearn.model_selection import train_test_split
```

W pierwszej kolejności należało podzielić uzyskane wcześniej zredukowane dane z tablicy `x_pca` na zestaw treningowy i testowy w proporcji 80:20.

```
X_train, X_test, y_train, y_test = train_test_split(x_pca, Y,
test_size=0.20)
```

Następnie utworzono obiekt klasyfikujący z parametrem n_neigbours, który określa liczbę sprawdzanych najbliższych sąsiadów, oraz parametrem metric określającym rodzaj wykorzystywanej metryki. Liczbę najbliższych sąsiadów określono na 20, gdyż dla tej wartości uzyskano najlepsze pokrycie wyników. Do obiektu klasyfikującego podano zestaw danych testowych.

```
classifier = KNeighborsClassifier(n_neighbors=20,
metric='euclidean')
classifier.fit(X_train, y_train)
```

Dalej podano do obiektu klasyfikującego zestaw danych testowych w celu wygenerowania dla nich przewidywanych klas decyzyjnych.

```
y_pred = classifier.predict(X_test)
y_pred:  [0 0 0 ... 0 1 1]
```

Wyniki porównano z oryginalnymi wartościami z tabeli `y_test` w celu obliczenia procentowego pokrycia wyników. Sposób obliczeń jest identyczny jak w przypadku wcześniej analizowanej regresji liniowej.

```
percent y element match: 67.09956709956711 %
```

Obliczenia algorytmem k – nn przyniosły ~67% pokrycia wyników, co jest jedynie o 2% niższą wartością względem stanu po analizie głównych składowych gdzie wynosiła ona ~69%. Ponadto algorytm k-nn okazał się bardziej efektywny w przewidywaniu wartości klas decyzyjnych od metody regresji liniowej.