Sieci z radialnymi funkcjami aktywacji

Aleksander Kłak, prowadzący Dr Marek Bazan

18 czerwca 2023

Spis treści

1 Cel ćwiczenia

Zadanie polega na implementacji algorytmu Kohonena w celu zastosowania go do klasteryzacji danych. Należy wykorzystąć sieć RBF na danych giełdowych do przewidywania danych i porównamy wyniki z rzeczywistym wykresem. Testy przeprowadzono dla wielu limitów predykcji oraz wielu różnych wartości centrów danych. Wybrałem dane giełdowe firmy Microsoft.

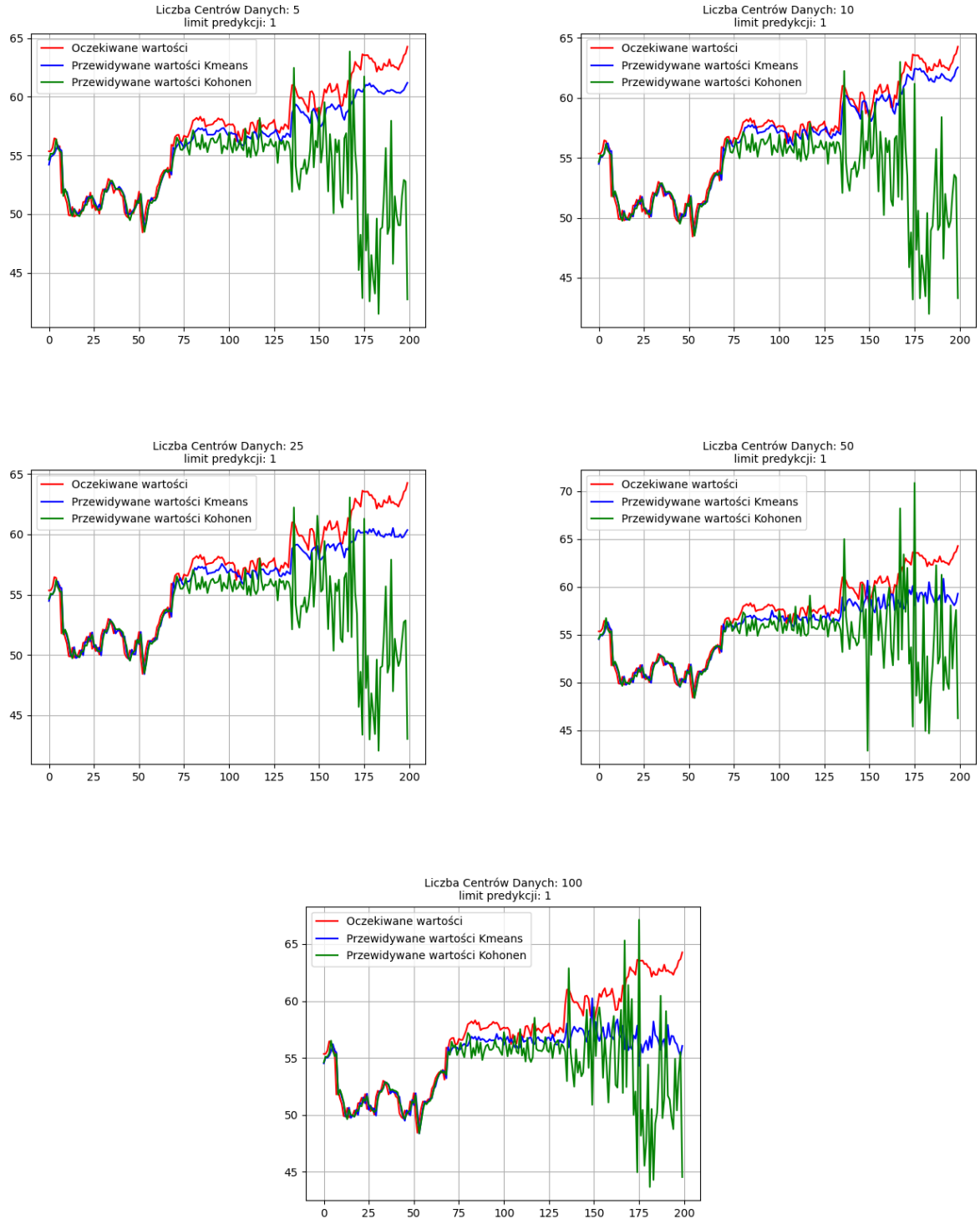
2 Algorytm Kohonena

Algorytm Kohonena jest jednym z algorytmów samoorganizujących map i służy do grupowania danych na podstawie ich podobieństwa. W kontekście tego zadania należy zastosować algorytm Kohonena, aby znaleźć klastry (grupy) danych w zbiorach danych Iris oraz danych giełdowych.

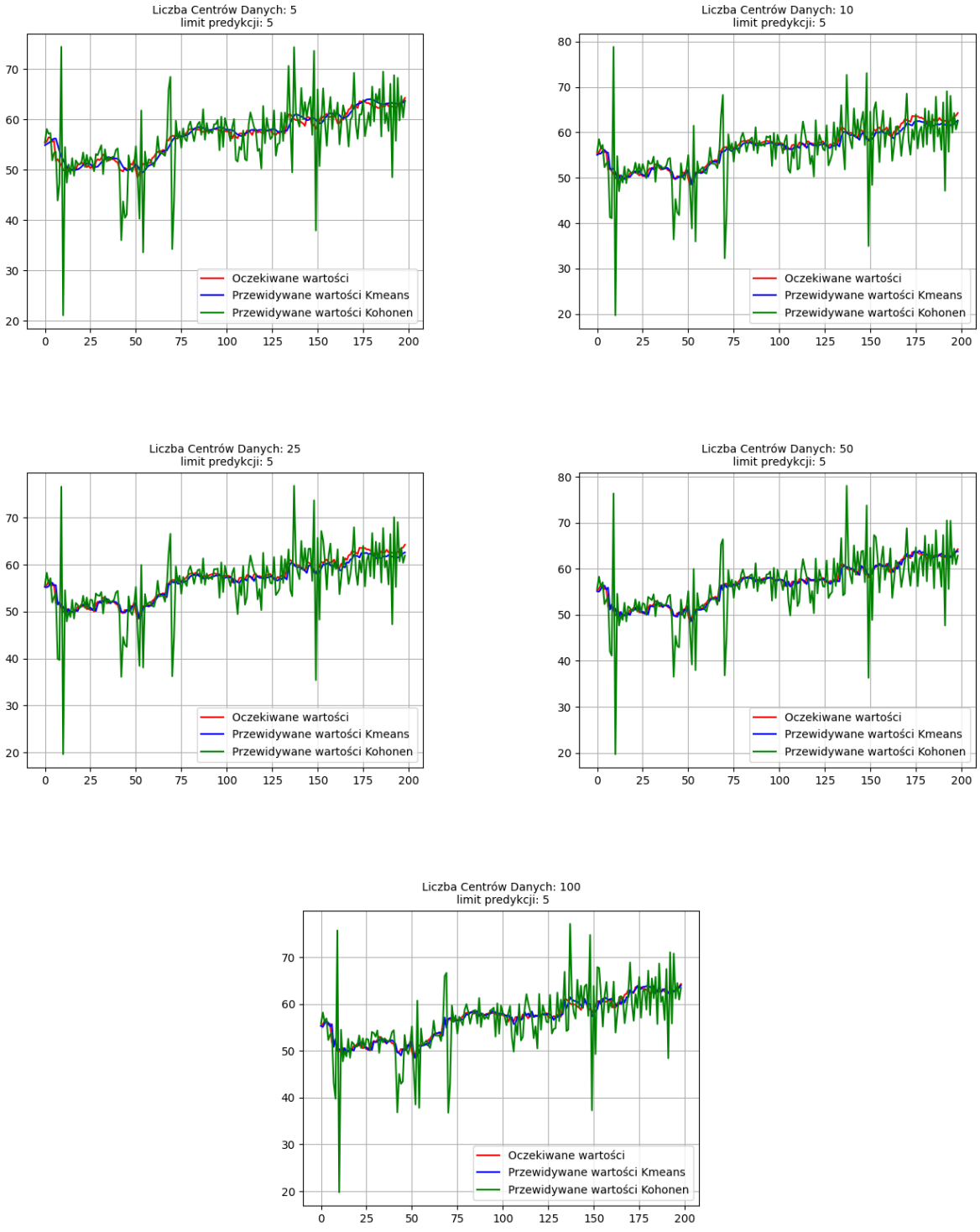
3 Sieć RBF

Sieć RBF (Radial Basis Function) to rodzaj sztucznej sieci neuronowej, która składa się z warstwy wejściowej, warstwy ukrytej i warstwy wyjściowej. Neurony w warstwie ukrytej są ustawione jako funkcje radialne, takie jak funkcje Gaussa, które obliczają odległość między swoim centrum a danymi wejściowymi. Warstwa wyjściowa agreguje informacje z warstwy ukrytej i generuje odpowiedzi na podstawie wyników obliczeń. Sieci RBF są wykorzystywane do aproksymacji funkcji, klasyfikacji i predykcji, szczególnie w przypadkach, gdy dane mają strukturę skupień w przestrzeni wejściowej.

4 Wyniki

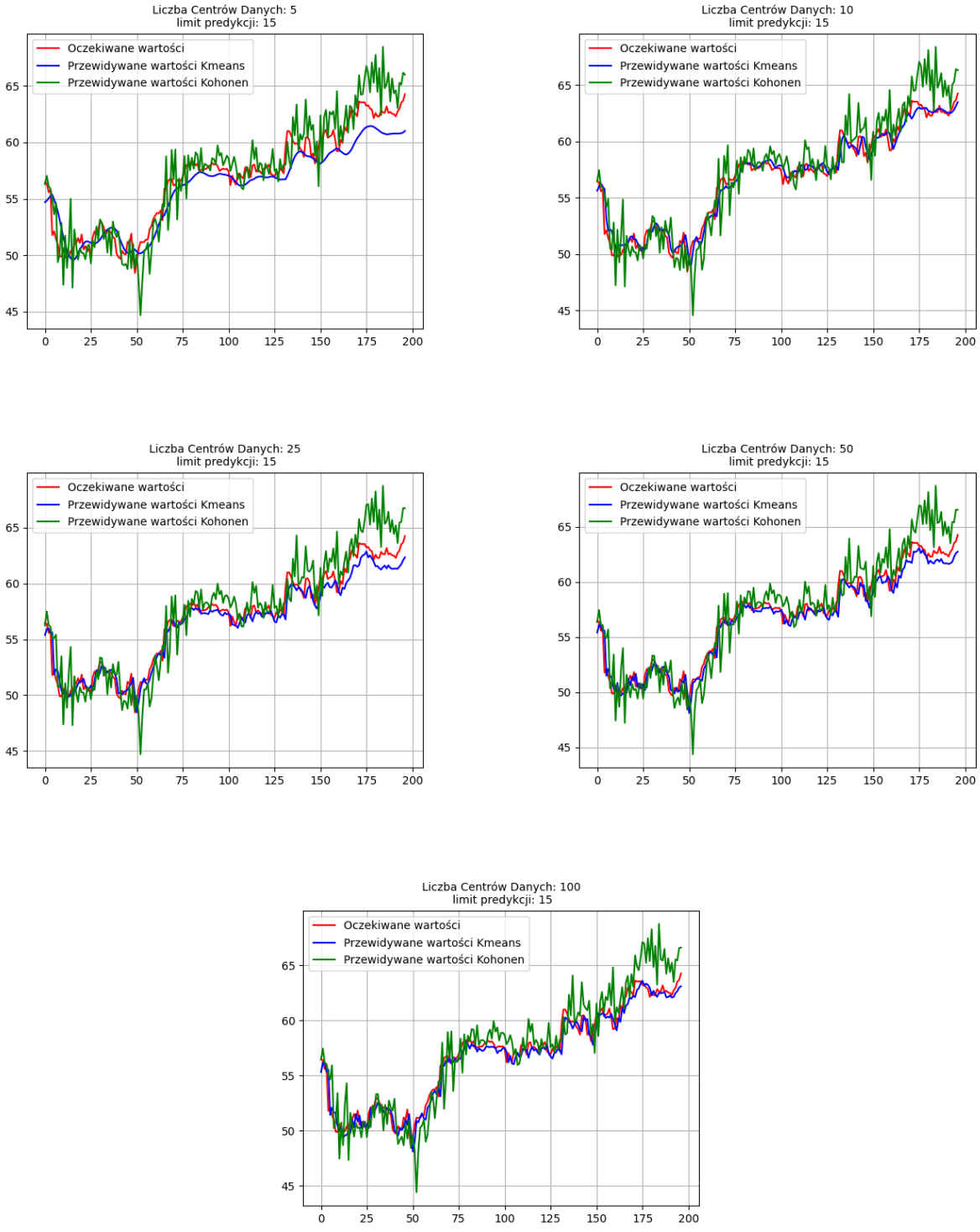


Rysunek 1: Różna liczba centr danych dla tego samego limitu predykcji

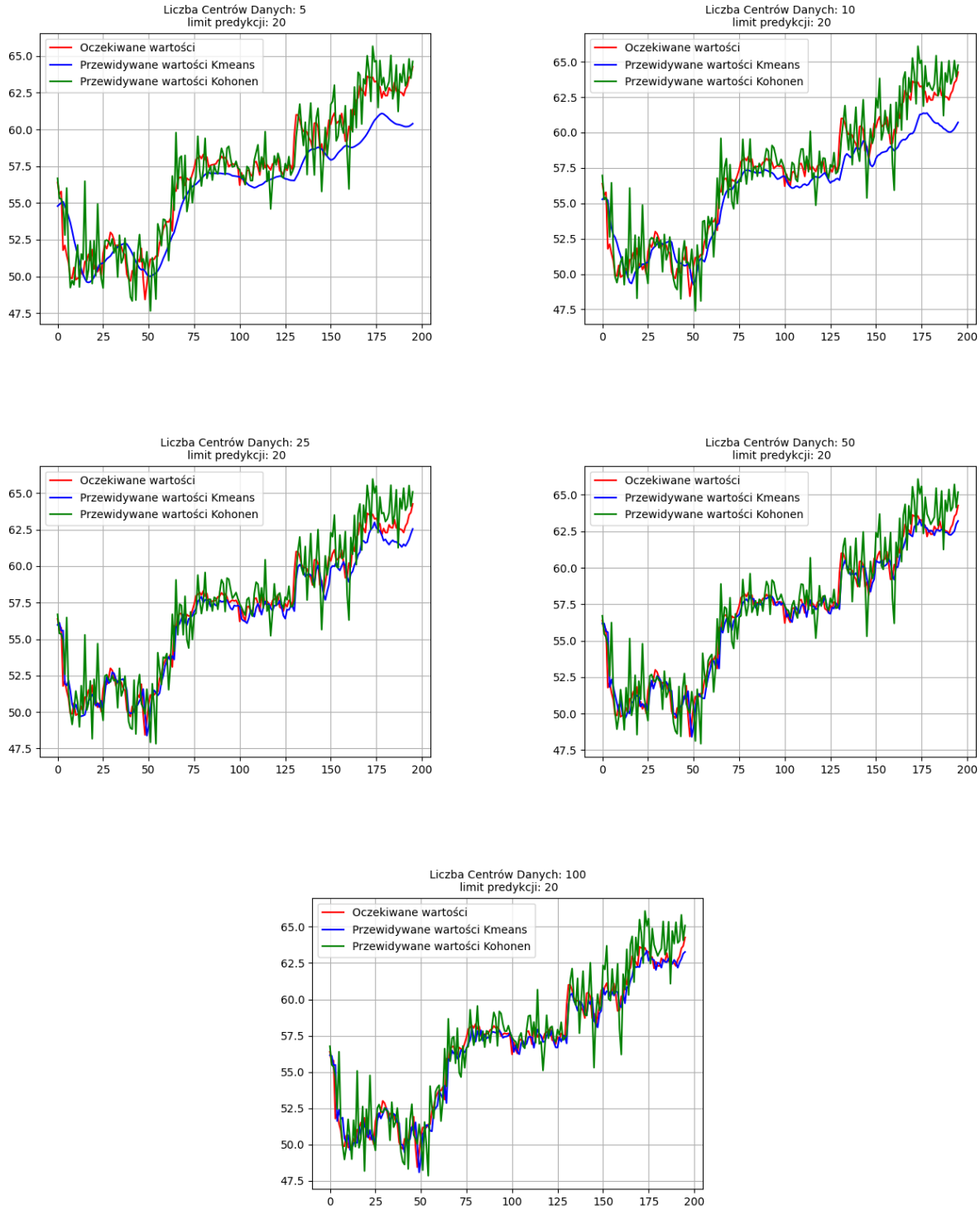


Rysunek 2: Różna liczba centr danych dla tego samego limitu predykcji

Rysunek 3: Różna liczba centr danych dla tego samego limitu predykcji



Rysunek 4: Różna liczba centr danych dla tego samego limitu predykcji



Rysunek 5: Różna liczba centr danych dla tego samego limitu predykcji

Tabela 1: Porównanie Kohonena i Kmeans

Cluster	Limit	MAE Kohonen	MAE Kmeans
5	1	3.6713	1.0814
5	5	3.543	0.7265
5	10	2.2378	0.8677
5	15	1.4716	1.2483
5	20	1.1281	1.5057
10	1	3.5615	0.6744
10	5	3.669	0.661
10	10	2.2528	0.8554
10	15	1.4608	0.5663
10	20	1.1598	1.2501
25	1	3.6379	1.1248
25	5	3.6369	0.6432
25	10	2.2009	0.6474
25	15	1.5236	0.6944
25	20	1.1609	0.664
50	1	3.057	1.4627
50	5	3.6071	0.4933
50	10	2.2201	0.6187
50	15	1.474	0.622
50	20	1.1592	0.5144
100	1	3.2638	2.0689
100	5	3.5959	0.5952
100	10	2.2629	0.5855
100	15	1.5156	0.5751
100	20	1.1618	0.5541

5 Wnioski

Zwiększenie liczby centr danych pozytywnie wpływa na predykcje szczególnie dla K-Means. Kohonen nie radzi sobie najlepiej, jednak Kmeans szczególnie dla wysokiej liczby centr danych radzi sobie nieźle. Mimo wszystko nie buduje to we mnie wystarczającego zaufania by skorzystać z tych metod w rzeczywistości.

6 Kod

6.1 Implementacja RBF

Listing 1: RBF

```
from scipy import *
from scipy.linalg import norm, pinv, svd
from matplotlib import pyplot as plt
from numpy.linalg import det
from scipy.spatial.distance import cdist
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
from sklearn.model_selection import train_test_split
from sklearn.cluster import KMeans
from sklearn.metrics import mean_absolute_error

LAMBDA=0.0

def gendist(X,Y):
  return cdist(X,Y,metric='euclidean')

class RBF:

    def __init__(self, centers, outdim, R):
        indim = centers.shape[1]
        numCenters = centers.shape[0]
        self.indim = indim
        self.outdim = outdim
        self.numCenters = numCenters
        self.centers = centers
        self.R = R
        self.W = np.random.random((self.numCenters, self.outdim))
        self.TRAINING_X_DATA = []

    def _basisfunc(self, c, d):
        assert len(d) == self.indim
        return np.exp(-1/(self.R)**2 * norm(c-d)**2)

    def _basisfuncFast(self, c, d, dist_mat):
        print("_basisfuncFast ---¿ STARTED")
        ret = np.exp(-1/(self.R)**2 * dist_mat**2)
        print("_basisfuncFast ---¿ ENDED")
        return ret

    def _calcAct(self, X):
        # calculate activations of RBFs
        G = np.zeros((X.shape[0], self.numCenters), float)
        for ci, c in enumerate(self.centers):
            for xi, x in enumerate(X):
                G[xi,ci] = self._basisfunc(c, x)

        if G.shape[0] == G.shape[1]:
           print('det(G) = ', det(G))

        return G

    def _calcActFast(self, X):
        G = np.zeros((X.shape[0], self.numCenters), float)

```

```
57          self.centers = np.vstack(self.centers[:,:]).astype(np.float64)
58          X = np.vstack(X[:,:]).astype(np.float64)
59
60          dist_mat = gendist(self.centers, X)
61          print("distance matrix ...  beeing calculated - STARTED")
62          G = self._basisfuncFast(self.centers, X, dist_mat)
63          print("distance matrix ...  beeing calculated - ENDEDED")
64
65          print('X.shape[0]=',X.shape[0], 'self.numCenters=',self.numCenters)
66
67          if G.shape[0] == G.shape[1]:
68              print('det(G) = ', det(G))
69
70          return G
71
72      def wypiszZbior(self, X, tekst):
73          print(tekst)
74          for i in range(X.shape[0]):
75              print(i,X[i,:])
76
77      def train(self, X, Y):
78          """ X: matrix of dimensions n x indim
79              y: column vector of dimension n x 1 """
80
81          self.TRAINING_X_DATA = X
82
83          # self.wypiszZbior(X,'ZBIOR TRENINGOWY ----------------------¿')
84          G = self._calcAct(X)
85          # calculate output weights (pseudoinverse)
86          self.W = np.dot(pinv(G), Y)
87
88      def trainRegularized(self, X, Y, _lambda=0):
89          print("Regularized RBF training.")
90          self.wypiszZbior(X,'ZBIOR TRENINGOWY ----------------------¿')
91
92          self.TRAINING_X_DATA = X
93
94          G = self._calcActFast(X)
95          U, S, VT = svd(G)
96          self.wartosci_szczegolne = S
97
98          S_inv = S*S-_lambda
99
100         print('VT.shape=',VT.shape)
101         print('S.shape=',S.shape)
102         print('U.shape=',U.shape)
103
104         N = G.shape[0]
105
106         r = N
107         for i in range(N):
108             if S[i] ¡ _lambda:
109                 r = i
110                 break
111
112         W_final = np.zeros((G.shape[0], Y.shape[1]))
113
114         for ii in range(Y.shape[1]):
115             w_lambda = np.zeros(G.shape[0])
116             for i in range(r):
117                 sigma = S[i]
118                 uTi = U[:,i].transpose()
```

```
119                     y = Y[:,ii]
120                     vi = VT.transpose()[:,i]
121                     f = sigma**2/(sigma**2+`lambda**2)
122                     w`lambda += 1/sigma*f*np.dot(np.dot(uTi,y),vi)
123                 W`final[:,ii] = w`lambda
124 
125             print('r=',r)
126 
127             self.W = W`final
128 
129         def test(self, X, treningowy = False):
130             """ X: matrix of dimensions n x indim """
131             G = self.`calcAct(X)
132             Y = np.dot(G, self.W)
133             return Y
```

6.2 Funkcje pomocnicze

Listing 2: Funkcja standaryzująca

```
def standardize_dataset(data):
    """
    Standaryzuje dane: (wartoś - średnia) / odchylenie standardowe.

    :param data: dane do standaryzacji
    :return: standaryzowane dane
    """
    mean_value = np.mean(data)
    std_dev = np.std(data)
    standardized_data = [(value - mean_value) / std_dev for value in data]

    return np.array(standardized_data)
```

Listing 3: Funkcja przygotowująca dataset

```
def prepare_data(data, target_index, sequence_length):
    """
    Przygotowuje dane do modelu sekwencyjnego.

    :param data: dane wejściowe
    :param target_index: indeks kolumny docelowej
    :param sequence_length: długość sekwencji wejściowych
    :return: przekształcone dane wejściowe i docelowe
    """
    input_data = []
    target_data = []

    for i in range(len(data) - sequence_length - 1):
        sequence_data = data[i:(i + sequence_length)]
        reshaped_sequence = sequence_data.values.reshape(sequence_data.shape[0] *
            sequence_data.shape[1])

        target_sequence = data[(i + sequence_length):(i + sequence_length + 1)][target_index]

        input_data.append(reshaped_sequence)
        target_data.append(target_sequence)

    return np.array(input_data), np.array(target_data)
```

Listing 4: Funkcja obliczająca maksymalny dystans między punktami

```
def compute_max_distance(data):
    """
    Oblicza najdłuższy dystans między dowolnymi dwoma punktami w zestawie danych.

    :param data: zestaw danych
    :return: najdłuższy obliczony dystans
    """
    max_distance = 0

    for i in range(len(data)):
        for j in range(i+1, len(data)):
            current_distance = np.linalg.norm(data[i] - data[j])
            max_distance = max(max_distance, current_distance)

    return max_distance
```

Listing 5: Kohonen

```
def apply_kohonen_algorithm(data, num_iterations, initial_learning_rate, num_representatives,
    learning_rate_decay, measure, decay_rate1, decay_rate2, standardize):
    """
    Implementacja algorytmu Kohonena do uczenia sieci neuronowych bez nauczyciela.

    :param data: dane wejściowe
    :param num_iterations: liczba iteracji do wykonania
    :param initial_learning_rate: początkowa wartość współczynnika uczenia
    :param num_representatives: liczba reprezentantów
    :param learning_rate_decay: typ zmniejszania współczynnika uczenia (1: liniowe, 2:
        wykładnicze, 3: hiperboliczne)
    :param measure: typ miary (1, 2, 3)
    :param decay_rate1: parametr C1 dla wykładniczego i hiperbolicznego zmniejszania
        współczynnika uczenia
    :param decay_rate2: parametr C2 dla hiperbolicznego zmniejszania współczynnika uczenia
    :param standardize: czy dane mają być standaryzowane
    :return: wektory reprezentantów, tablica m, dane wejściowe
    """

    # Standaryzacja danych
    if standardize:
        data = standardize_dataset(data)

    # Inicjalizacja i normalizacja wektorów reprezentantów
    representative_vectors = initialize_representative_vectors(len(data), num_representatives
        , len(data[0]))

    # Wybór miary i wykonanie iteracji
    measure_table = [0]*len(data)
    learning_rate = initial_learning_rate

    for iteration in range(num_iterations):
        chosen_representative = apply_measure(measure, data, iteration, num_representatives,
            representative_vectors)
        measure_table[iteration%len(data)] = chosen_representative

        # Modyfikacja wektorów reprezentantów
        representative_vectors[chosen_representative] += learning_rate * (data[iteration%len(
            data)] - representative_vectors[chosen_representative])
        representative_vectors[chosen_representative] /= np.linalg.norm(
            representative_vectors[chosen_representative])

        # Aktualizacja współczynnika uczenia
        learning_rate = update_learning_rate(learning_rate_decay, initial_learning_rate,
            num_iterations, iteration, decay_rate1, decay_rate2)

    return representative_vectors, measure_table, data


def initialize_representative_vectors(data_length, num_representatives, data_width):
    vectors = []
    random_generator = np.random.RandomState(0)

    for _ in range(num_representatives):
        for _ in range(data_length):
            temp_vector = random_generator.normal(loc=0.0, scale=0.01, size=data_width)
            vectors.append(temp_vector/np.linalg.norm(temp_vector))

    return np.array(vectors)

```

```
53
54  def apply_measure(measure, data, iteration, num_representatives, representative_vectors):
55      if measure == 1:
56          return dot_product_measure(data, iteration, num_representatives,
                representative_vectors)
57
58      if measure == 2:
59          return euclidean_distance_measure(data, iteration, num_representatives,
                representative_vectors)
60
61      if measure == 3:
62          return manhattan_distance_measure(data, iteration, num_representatives,
                representative_vectors)
63
64
65  def update_learning_rate(learning_rate_decay, initial_learning_rate, num_iterations,
        current_iteration, decay_rate1, decay_rate2):
66      if learning_rate_decay == 1:  # Liniowe zmniejszanie
67          return initial_learning_rate * (num_iterations - current_iteration) / num_iterations
68
69      if learning_rate_decay == 2:  # WykĆadnicze zmniejszanie
70          return initial_learning_rate * math.exp(-decay_rate1 * current_iteration)
71
72      if learning_rate_decay == 3:  # Hiperboliczne zmniejszanie
73          return decay_rate1 / (decay_rate2 + current_iteration)
```

Listing 6: Metody podobieństwa

```
1   def dot_product_measure(data, iteration_index, num_representatives, representative_vectors):
2       """
3       Miara podobie stwa oparta na iloczynie skalarnym (miara 3).
4
5       :param data: dane wej Żciowe
6       :param iteration_index: indeks iteracji
7       :param num_representatives: liczba reprezentant w
8       :param representative_vectors: wektory reprezentant w
9       :return: indeks reprezentanta o najwi Źkszej miarze
10      """
11      measurements = [np.dot(representative_vectors[i], data[iteration_index % len(data)]) for
            i in range(num_representatives)]
12
13      max_index = np.argmax(measurements)
14      return max_index
15
16
17  def euclidean_distance_measure(data, iteration_index, num_representatives,
        representative_vectors):
18      """
19      Miara podobie stwa oparta na odleg Ćo Żci euklidesowej (miara 4).
20
21      :param data: dane wej Żciowe
22      :param iteration_index: indeks iteracji
23      :param num_representatives: liczba reprezentant w
24      :param representative_vectors: wektory reprezentant w
25      :return: indeks reprezentanta o najmniejszej miarze
26      """
27      measurements = [np.linalg.norm(representative_vectors[i] - data[iteration_index % len(
            data)]) for i in range(num_representatives)]
28
29      min_index = np.argmin(measurements)
30      return min_index
31
```

```
32
33  def manhattan_distance_measure(data, iteration_index, num_representatives,
        representative_vectors):
34      """
35      Miara podobie stwa oparta na odleg Śo Żci Manhattan (miara 5).
36
37      :param data: dane wej Żciowe
38      :param iteration_index: indeks iteracji
39      :param num_representatives: liczba reprezentant w
40      :param representative_vectors: wektory reprezentant w
41      :return: indeks reprezentanta o najmniejszej miarze
42      """
43      measurements = [sum(abs(representative_vectors[i][j] - data[iteration_index % len(data)][
            j]) for j in range(len(representative_vectors[0]))) for i in range(
            num_representatives)]
44
45      min_index = np.argmin(measurements)
46      return min_index
```

Listing 7: Funkcja tworząca wykresy

```
1   def plot_results(expected, predicted_kmeans, predicted_kohonen, num_clusters, limit):
2       plt.grid()
3       plt.plot(expected, '-', label='Oczekiwane warto Żci', c='red')
4       plt.plot(predicted_kmeans, '-', label='Przewidywane warto Żci Kmeans',c= 'blue')
5       plt.plot(predicted_kohonen, '-', label='Przewidywane warto Żci Kohonen',c= 'green')
6
7       plt.title(f"Liczba Centr w Danych: -num_clusters""n Limit Predykcji: -limit"", size=10)
8       plt.legend()
9
10      # Zapisanie wykresu z ustandaryzowan  nazw
11      filename = f"plot_cluster_-num_clusters"_limit_-limit"_.png"
12      # plt.savefig(filename)
13      plt.show()
```

6.3 Skrypt

Listing 8: Skrypt

```
# wczytanie danych
data = pd.read_csv('all_stocks_5yr.csv')
df = data[data['Name'] == 'MSFT']
X_df = df[['open', 'high', 'low', 'close']].head(1000)

# parametry
alpha = 0.1
iter = 5000
measure = 1
learning_rate = 1
C1 = 0.5
C2 = 0.5
standardize_data = True

cluster_values = [5]#,10,25,50,100]
limit_values = [1]#,5,10,15,20]

import warnings
warnings.filterwarnings("ignore")
# Otwórz plik tekstowy do zapisu
with open("results1.txt", "w") as file:
    for num_clusters in cluster_values:
        for limit in limit_values:
            # Preparacja danych
            data,expected_vals = prepare_data(X_df, 'close', limit)
            standardized_data = standardize_dataset(data)
            X_train, X_test, y_train, y_test = train_test_split(standardized_data,
                expected_vals, test_size=0.2, shuffle=False)

            # Kohonen
            centers, predicts, X = apply_kohonen_algorithm(X_train, iter, alpha, num_clusters
                , learning_rate, measure, C1, C2, standardize_data)
            max_distance = compute_max_distance(standardized_data)
            rbf = RBF(centers, num_clusters, max_distance)
            rbf.train(X_train, y_train)
            output_kohonen = rbf.test(X_test)
            MAEValue_kohonen = round(mean_absolute_error(y_test, output_kohonen),4)
            print('Mean Absolute Error Value for Kohonen is : ', MAEValue_kohonen)

            # KMeans
            kmeans = KMeans(n_clusters=num_clusters, random_state=0)
            kmeans.fit(X_train)
            centers = kmeans.cluster_centers_
            centers = np.array(centers)
            max_distance = compute_max_distance(standardized_data)
            rbf = RBF(centers, num_clusters, max_distance)
            rbf.train(X_train, y_train)
            output_kmeans = rbf.test(X_test)
            MAEValue_kmeans = round(mean_absolute_error(y_test, output_kmeans),4)
            print('Mean Absolute Error Value for Kmeans is : ', MAEValue_kmeans)

            # Rysowanie wykresu
            plot_results(y_test, output_kmeans, output_kohonen, num_clusters, limit)

            # Zapisz wyniki MAE do pliku tekstowego
            #file.write(f"Cluster: {num_clusters}, Limit: {limit}, MAE for Kohonen: {
                MAEValue_kohonen}, MAE for Kmeans: {MAEValue_kmeans}\n")
```