Systemy Sztucznej Inteligencji

Predykcja cukrzycy u pacjentów z wykorzystaniem miękkiego KNN.

Marian Dorosz
Wojciech Brożek
Oskar Fojcik
Grupa 2/3

4 maja 2022

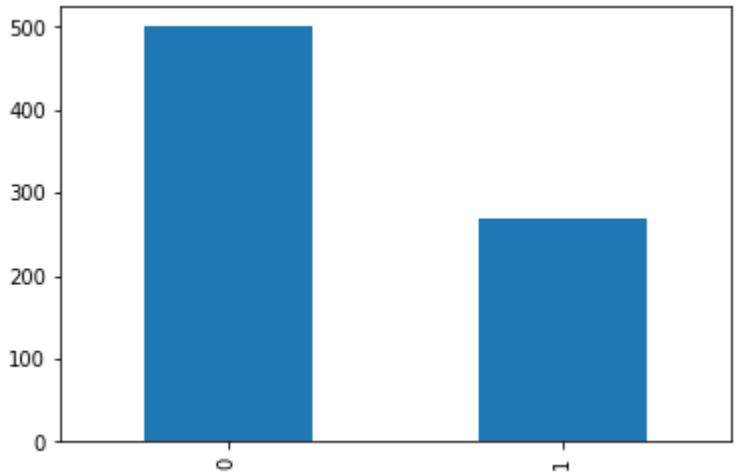
Część I

Opis programu

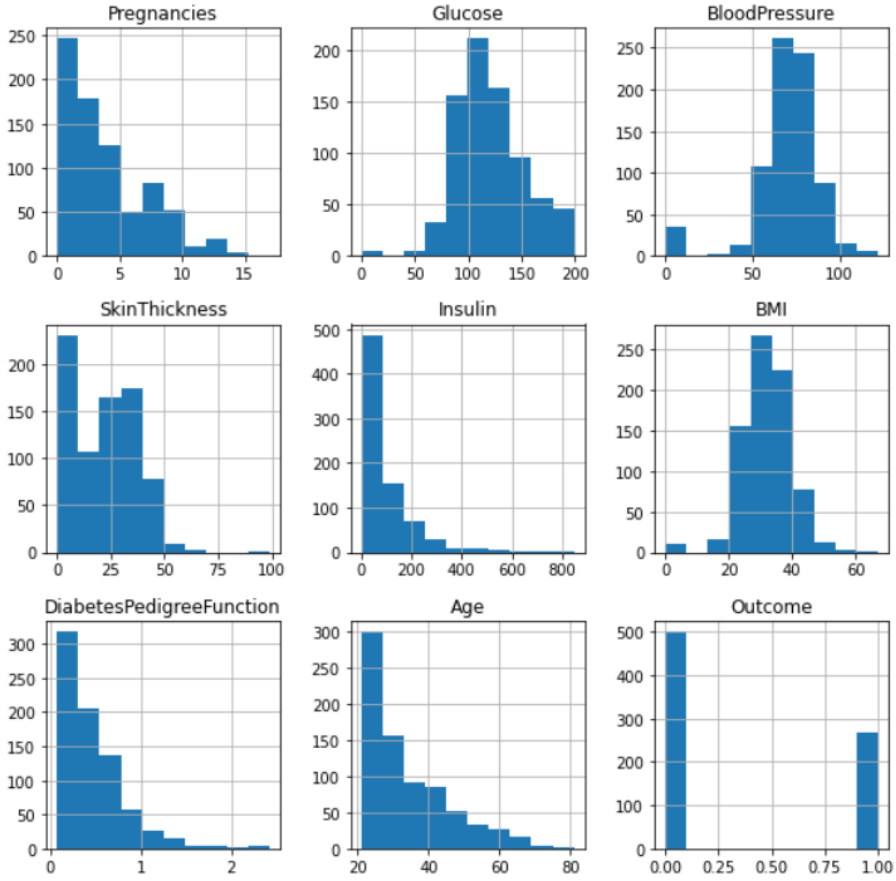
Program to implementacja prostego algorytmu uczenia maszynowego - KNN, który wykorzystując zbiory miękkie decyduje, czy dana osoba ma cukrzycę lub nie.

Część II

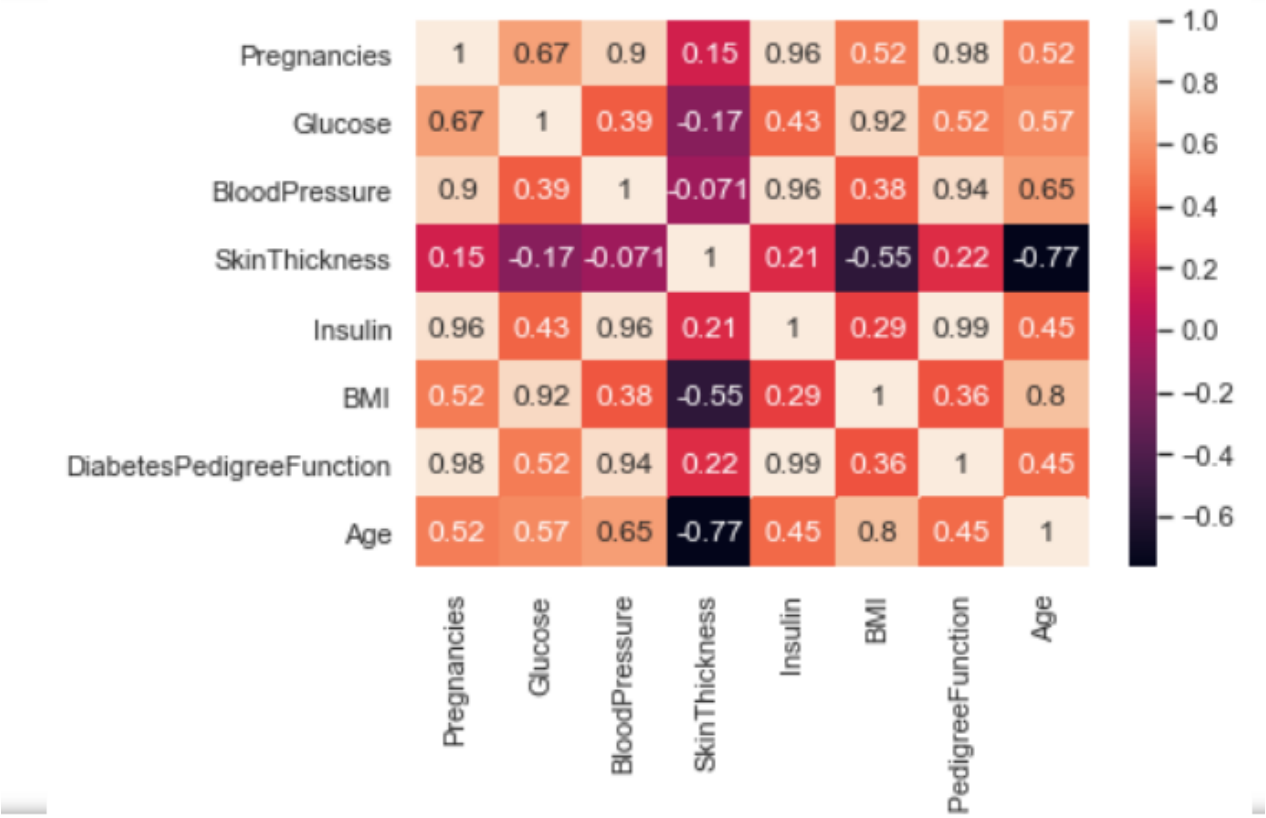
Analiza danych



Rysunek 1: Wykres przedstawiający rozkład osób zdrowych oraz chorych na cukrzycę; gdzie 0 to osoba zdrowa, 1 to osoba chora



Rysunek 2: Wykres przedstawiający jak rozdzielone są dane w poszczególnych klasach



Rysunek 3: Wykres przedstawiający korelacje pomiędzy poszczególnymi danymi im jaśniejsze pole tym większa korelacja

Wnioski:

- Istnieje wysoka korelacja pomiędzy funkcją wykrywającą cukrzycę a poziomem insuliny,glukozy oraz ciśnieniem krwi.
- Funkcja wykrywająca cukrzycę prawdopodobnie mylnie klasyfikuje kobiety w ciąży jako chore na cukrzycę.
- Osoby młode mają statystycznie większe BMI.
- Osoby o wyskoim ciśnienu krwi mają zwykle wysoki poziom glukozy.

Opis działania

Informacje ogólne

Cały program wykorzystuje trzy klasy statyczne, które odpowiadają za:

- Przetwarzanie danych
- Tworzenie zbioru miękkiego oraz obliczanie przynależności do klasy w opraciu o zadany zbiór miękki
- Klasyfikację przy pomocy algorytmu KNN

Klasa ProcessingData

Ta klasa odpowiada za przetwarzanie danych. Posiada ona trzy funkcje:

- splitSet(x,k) - funkcja ta zwraca zbiór danych podzielony w określonej proporcji. Przyjmuje ona dwa argumenty - zbiór danych oraz w jakim stosunku ma go podzielić.

Data: Dane wejściowe zbiór danych x, liczba k
Result: $x[:n]$, $x[:n]$
n = int(len(x)*k);
return $x[:n]$, $x[n:]$;

Algorithm 1: Podział danych wejściowych na dwa zbiory.

- shuffle(x) - funkcja ta zwraca dane przetasowane przy pomocy algorytmu **Fisher-Yates shuffle**. Przyjmuje ona jako argument zbiór danych.

Data: Dane wejściowe zbiór danych x
Result: Przetasowany zbiór x
i=len(x)-1;
while $i > 0$ **do**
  j = random(1, len(x));
  swap($x[i]$, $x[j]$);
end
return x;

Algorithm 2: Algorytm tasowania danych.

- normalize(x) - ta funckja normalizuje zbiór danych wykorzystując algorytm normalizacji min-max. Wynikiem jej działania jest znormalizowany zbiór danych. Przyjmuje ona jeden argument i jest nim zbiór danych.

Data: Dane wejściowe zbiór danych x
Result: Znormalizowany zbiór danych x
Pobierz kolumny pomijając kolumnę 'Outcome' i zapisz je w *values*
Stwórz listę nazw kolumn zbioru *values* i przypisz je do zmiennej *cols*
foreach *col in cols* **do**
    Pobierz wszystkie wiersze dla danej kolumny i przypisz je do zmiennej *data*;
    Pobierz maksymalną wartość ze zmiennej *data* i przypisz do zmiennej *max*;
    Pobierz minimalną wartość ze zmiennej *data* i przypisz do zmiennej *min*;
    for *(r = 0; r < len(x); r++)* **do**
        $x[r, col] = (x[r, col]$ - $min/(max - min))$;
    end
end
return x

Algorithm 3: Algorytm normalizacji danych (min-max).

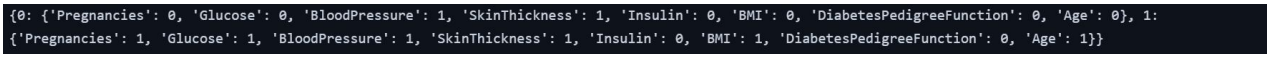
Klasa soft_set

Zadaniem tej klasy jest zbudowanie zbioru miękkiego dla zadanego wektora danych. W jej skład wchodzą następujące funckje:

- build_soft_set(x) - ta funckja służy do zbudowania zbioru miękkiego dla zadanej jako argument kolekcji danych. Jej wynikiem jest słownik z wartościami 0-1. Samo budowanie słownika odbywa się następująco: algorytm wylicza średnie wartości dla zadanej kolumny, a następnie sprawdza ile wartości w kolumnie jest poniżej i powyżej średniej dla osób zdrowych i niezdrowych. W oparciu o te dane uzupełnia słownik wartościami.

Data: Dane wejściowe zbiór danych x
Result: Zbiór miękki w postaci słownika $soft\_set$
Utwórz słownik $soft\_set$;
Utwórz klucz w słowniku: $soft\_set[0]$ i przypisz do niego nowy słownik;
Utwórz klucz w słowniku: $soft\_set[1]$ i przypisz do niego nowy słownik;
foreach *key in* $soft\_set$ **do**
    $data1 = x$;
    Pobierz każdy wiersz, dla którego kolumna 'Outcome' == *key*;
    Z $data1$ oraz $data2$ usuń kolumnę 'Outcome';
    foreach *key in* $data1.columns$ **do**
        $mean$ = średnia wartości całej kolumny $data1[col]$;
        $temp1 = 0$;
        $temp2 = 0$;
    end
    foreach *val in* $data2[col]$ **do**
        if $val < mean$ **then**
            $temp1$++;
        else
            $temp2$++;
        end
    end
    if $temp1 < temp2$ **then**
        $soft\_set[key][col] = 0$;
    else
        $soft\_set[key][col] = 1$;
    end
end
return $soft\_set$

Algorithm 4: Algorytm budujący zbiór miękki.



Rysunek 4: Przykładowy słownik reprezentujący zbiór miękki.

- get_membership(x, v) - ta funkcja sprawdza do jakiej klasy przynależy dany wektor. Przyjmuje ona jako argumenty zbiór miękki oraz wektor danych. Jej wynikiem jest liczba zmiennoprzecinkowa.

Data: Dane wejściowe zbiór miękki $soft\_set$, wektor danych v
Result: Liczba zmiennoprzecinkowa
$result$ = nowy słownik;
foreach key *in* $soft\_set$ **do**
  $result[key] = 1$;
  foreach $key2$ *in* $soft\_set[key2]$ **do**
    $result[key]$ += $soft\_set[key][key2]$ * $v[key2]$
  end
end
Zwróć największą wartość przypisaną do jednego z kluczy słownika.;

Algorithm 5: Algorytm tasowania danych.

Klasa soft_KNN

Ta klasa to implementacja algorytmu KNN, który do głosowania wykorzystuje zbiory miękkie. Składa się ona z następujących funkcji:

- calc_euclidian_distance(val1,val2) - ta funkcja zwraca jednowymiarową odległość euklidesową dla dwóch wartości. Innymi słowy implementuje wzór $|val1 - val2|$, gdzie val_i to wartość.

Data: Dane wejściowe liczby $val1$, $val2$
Result: Liczba zmiennoprzecinkowa
return abs($val1 - val2$);

Algorithm 6: Obliczanie metryki euklidesowej.

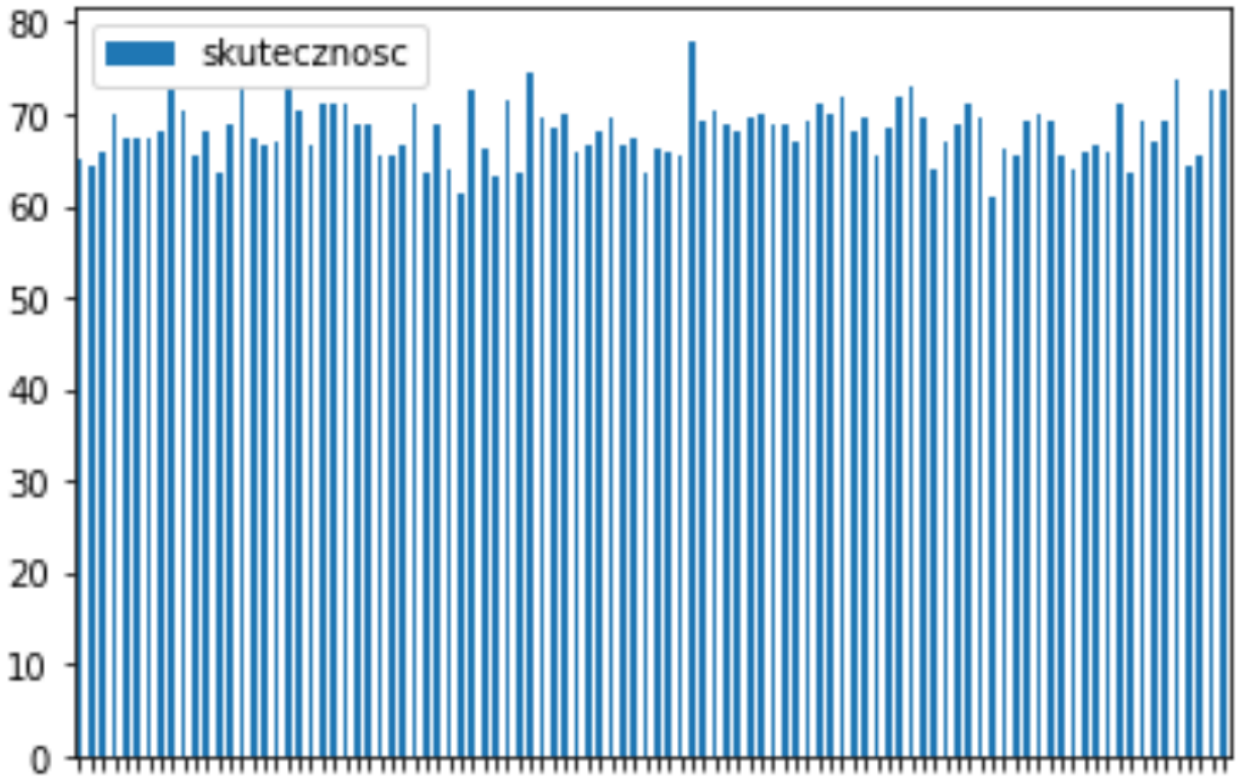
- clustering(x,v,k) - ta funkcja jest implementacją algorytmu KNN, z wykorzystaniem elementów klasy soft_set. Jej zadaniem jest ocena, czy wprowadzone dane należą do osoby zdrowej lub chorej. KNN jest leniwym algorytmem uczenia nadzorowanego, więc dane muszą mieć kolumnę, dzięki której jest możliwe jednoznaczne zidentyfikowanie klasy obiektu. Sam algorytm polega na wyliczeniu k sąsiadów, obok których znajduje się badany element. W tym celu wykorzystuje się metrykę (np.: Euklidesową), która określa jak blisko danego sąsiada znajduje się badany element. Klasa, do której zostanie przypisany element jest zależna od tego, jak wielu sąsiadów danego typu posiada. Jeżeli np.: większość sąsiadów badanego obiektu to cukrzycy, to będzie to oznaczać, że osoba jest najprawdopodobniej cukrzykiem.

Data: Dane wejściowe zbiór danych x, wektor danych *sample*, liczba całkowita k
Result: Nazwa klasy, do której należy próbka
$soft\_x = soft\_set.build\_soft\_set(x)$;
$distances$ = [ ];
for *(i = 0; i < len(x), i++)* **do**
  $temp\_x$ =$soft\_set.build\_soft\_set(soft\_x, x[i])$;
  $temp\_sample = soft\_set.build\_soft\_set(soft\_x, sample)$;
  $distances.append(soft\_KNN.calc\_euclidian\_distance(temp_x, temp_sample))$;
end
temp = kopia x;
Do zmiennej *temp* dodaj kolumnę 'distance' i przypisz jej tablicę *distances*;
Przesortuj tabelę *temp* zgodnie z wartościami kolumny 'distances';
$classes$ = {0: 0, 1: 0};
for *(i = 0; i < k; i++)* **do**
  $classes[temp[i].Outcome]$ += 1;
end
Zwróć klucz z największą przypisaną wartością;

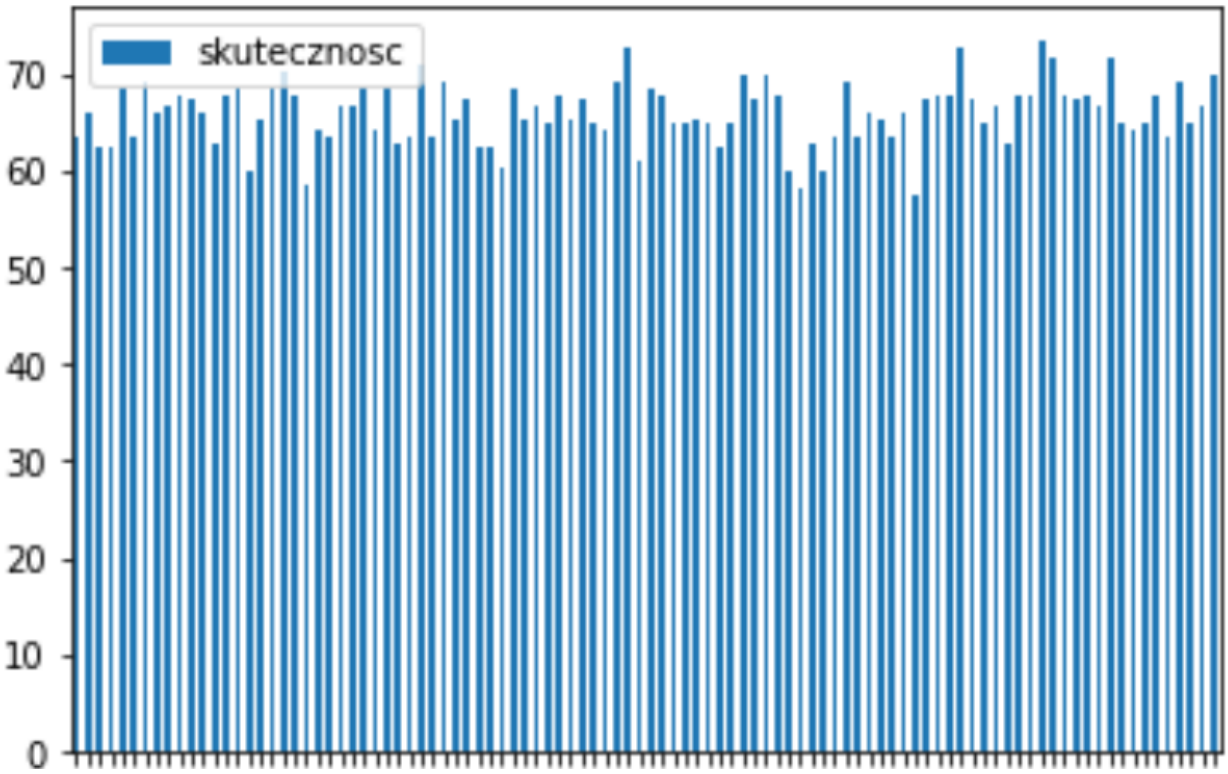
Algorithm 7: Klasteryzacja danych.

Testy

Przeprowadzone zostały testy dla większych i mniejszych zbiorów testowych przeasowanych i znormalizowanych. Większy zawierał 537 elementów a mniejszy 375.



Rysunek 5: Wykres skuteczności dla większego zbioru treningowego i walidacyjnego. Średnia skutecznośc wynosi 68.19 %



Rysunek 6: Wykres skuteczności dla mniejszego zbioru treningowego i walidacyjnego. Średnia skutecznośc wynosi 65.98 %

Pełen kod aplikacji

```
import pandas as pd
import numpy as np
import seaborn as sns

class ProcessingData:
    @staticmethod
    def splitSet(x: pd.DataFrame,k: int) -> pd.DataFrame:
        n = int(len(x)*k)
        xTrain = x[:n]
        xVal = x[n:]
        return xTrain, xVal

    @staticmethod
    def shuffle(x: pd.DataFrame) -> pd.DataFrame:
        for i in range(len(x)-1, -1, -1):
            j = np.random.randint(i, len(x))
            x.iloc[i], x.iloc[j] = x.iloc[j], x.iloc[i]
        return x

    @staticmethod
    def normalize(x: pd.DataFrame) -> pd.DataFrame:
```

```
        values = x.loc[:, x.columns != 'Outcome']
        columnNames=values.columns.tolist()
        for column in columnNames:
            data = x.loc[:,column]
            max1 = max(data)
            min1 = min(data)
            for row in range(0,len(x),1):
                x.at[row,column] = (x.at[row,column]-min1)/(max1-min1)
        return x

class soft_set:

    @staticmethod
    def build_soft_set(x: pd.DataFrame) -> dict:
        soft_set = {}
        soft_set[0] = {}
        soft_set[1] = {}
        for key in soft_set:
            data1 = x
            data2 = x.loc[x['Outcome']==key]
            data1 = data1.drop(['Outcome'], axis=1)
            data2 = data2.drop(['Outcome'], axis=1)
            for col in data1.columns:
                mean = data1[col].mean()
                temp1 = 0 # lower or equal to mean
                temp2 = 0 # greater than mean
                for val in data2[col]:
                    if val < mean:
                        temp1 += 1
                    else:
                        temp2 += 1
                if temp1 > temp2:
                    soft_set[key][col] = 0
                else:
                    soft_set[key][col] = 1
        return soft_set

    @staticmethod
    def get_membership(soft_set: dict, vector: pd.Series) -> float:
        vector = vector.to_dict()
        result = dict()
        for key in soft_set:
            result[key] = 1
            for key2 in soft_set[key]:
                result[key] += soft_set[key][key2] * vector[key2]
        return float(result[max(result, key=result.get)])

class soft_KNN:
    @staticmethod
    def calc_euclidian_distance(val1: float, val2: float) -> float:
        return np.abs(val1-val2)

    @staticmethod
    def clustering(x: pd.DataFrame, sample: pd.Series, k: int) -> str:
        soft_x = soft_set.build_soft_set(x)
```

```
77          distances = []
78          for i in range(0, len(x)):
79              temp_x = soft_set.get_membership(soft_x, x.iloc[i])
80              temp_sample = soft_set.get_membership(soft_x, sample)
81              distances.append(soft_KNN.calc_euclidian_distance(temp_x,
                    temp_sample))
82          tempdf = x.copy()
83          tempdf['distance'] = distances
84          tempdf = tempdf.sort_values(by='distance')
85          classes = {
86                      0: 0,
87                      1: 0
88                     }
89          for i in range(k):
90              classes[tempdf.iloc[i].Outcome] += 1
91
92          return max(classes, key = classes.get)
```