

## Dokumentácia

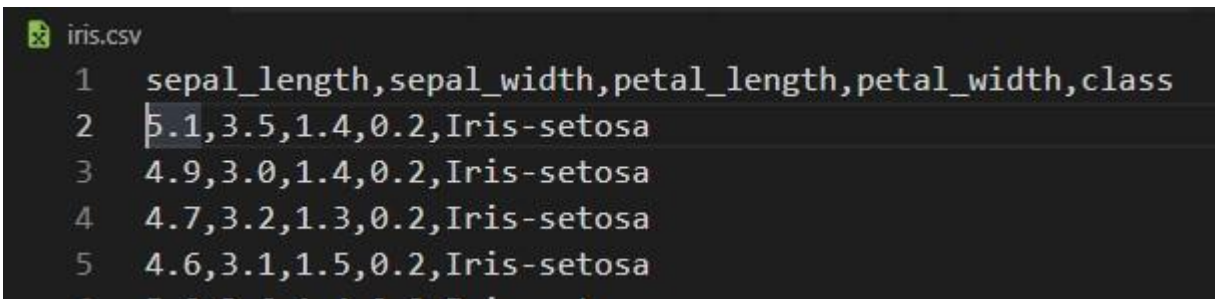
Program knn.py je program, ktorý klasifikuje vstup do prislúchajúcej triedy na základe nejakého datasetu.

Aby spustiť program treba zavolať súbor knn.py a pridať mu vhodné argumenty:

1. Cestu ku datasetu.
2. Triedu ktorú budeme klasifikovať
3. Cestu ku vstupom, ktoré chceme klasifikovať
4. Počet susedov.

Príklad použitia: *knn.py iris.csv class data.txt 20*

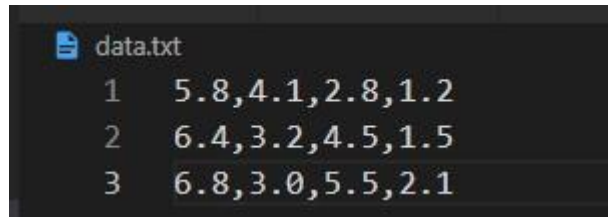
Pričom dataset ma byť v formáte .csv, trieda v datasete môže byť buď numerická buď slovne popísaná, a ostatne prvky musia byť iba numerické.



```
iris.csv
1 sepal_length,sepal_width,petal_length,petal_width,class
2 5.1,3.5,1.4,0.2,Iris-setosa
3 4.9,3.0,1.4,0.2,Iris-setosa
4 4.7,3.2,1.3,0.2,Iris-setosa
5 4.6,3.1,1.5,0.2,Iris-setosa
```

Obrázok 1. Príklad datasetu.

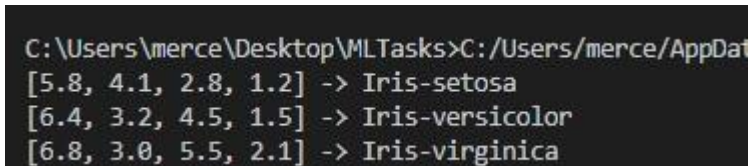
Vstupy sú riadky rozdelené čiarkou, musia byť numerické a počet hodnôt v jednom riadku má byť rovnaký, ako počet stĺpcov v datasete - 1.



```
data.txt
1 5.8,4.1,2.8,1.2
2 6.4,3.2,4.5,1.5
3 6.8,3.0,5.5,2.1
```

Obrázok 2. Príklad vstupov.

Počet susedov je numerická hodnota väčšia, ako 0. Pri správnych parametroch program vyprodukuje nasledujúci výstup.



```
C:\Users\merce\Desktop\MLTasks>C:/Users/merce/AppData
[5.8, 4.1, 2.8, 1.2] -> Iris-setosa
[6.4, 3.2, 4.5, 1.5] -> Iris-versicolor
[6.8, 3.0, 5.5, 2.1] -> Iris-virginica
```

Obrázok 3. Príklad výstupu.

Nižšie je uvedený kód z komentármi ako funguje algoritmus.

```
import numpy as np
import pandas as pd
import argparse

def distance(x, y):
    sum = 0

    # Vypocet suctu rozdelov atributov
```

```

    for xi, yi in zip(x, y):
        sum += (xi - yi) **2

    # Odmocnina suctu
    return sum ** (1/2)

def knn(dataframe, target_class, y, N=3):
    df_vals = dataframe.drop(target_class, axis=1)

    # Vypocet vzdalenosti pre kazdy element v datasete a neznamy
    element
    df_vals['distance'] = df_vals.apply(lambda x: distance(x.to_numpy(), y), axis=1)
    # print(df_vals)

    # Zistime indexy N elementov z najmensou vzdalenostou
    smallest_indexes = df_vals.nsmallest(N, 'distance').index

    # Zistime triedy indexov
    nearest = dataframe.loc[smallest_indexes][target_class]
    # print(nearest)

    #Najdeme najcastiejsie triedy a vyberjeme maximalnu z nych
    most_frequent = nearest.value_counts().index[0]
    return most_frequent

# Parser argumentov
parser = argparse.ArgumentParser(description='KNN classifier')

parser.add_argument('path', metavar='path', type=str, help='path to dataset')
parser.add_argument('target_class', metavar='target_class', type=str, help='target field to classify')

```

```

parser.add_argument('data_path', metavar='data_path', type=str, help='data to classify')
parser.add_argument('N', metavar='N', type=int, help='Neighbours count')

args = parser.parse_args()

dataset_path = args.path
target_class = args.target_class
data_path = args.data_path
N = args.N

if N < 1:
    raise ValueError('Invalid neighbours count')

with open(data_path) as f:

    data = []
    df = pd.read_csv(dataset_path)

    for line in f.readlines():
        elem = line.strip().split(',')

        if len(elem) != len(df.columns)-1:
            raise ValueError(f'Invalid features length. expected {len(df.columns)-1} got {len(elem)}')

        data.append(list(map(float, elem)))

    for elem in data:
        # klasifikacia kazdeho prikladu
        predicted = knn(df, target_class, np.array(elem), N)
        print(f"{elem} -> {predicted}")

```