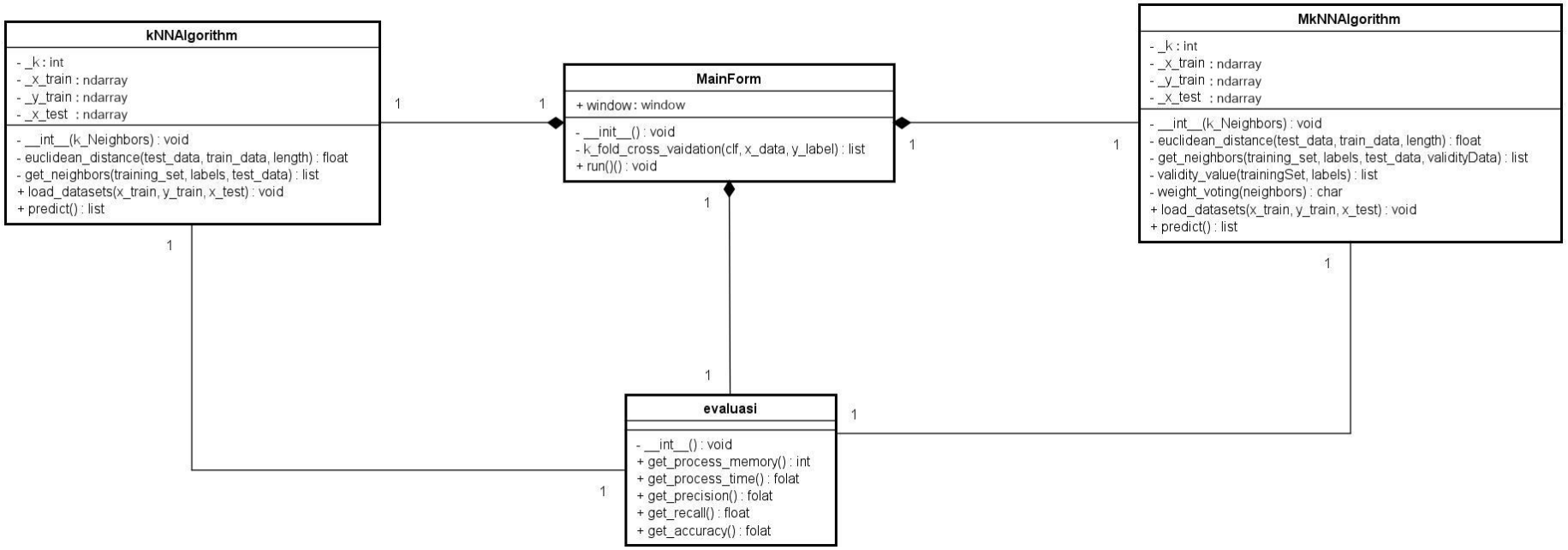
kNNAlgorithm
- _k : int
- _x_train : ndarray
- _y_train : ndarray
- _x_test : ndarray
- __int__(k_Neighbors) : void
- euclidean_distance(test_data, train_data, length) : float
- get_neighbors(training_set, labels, test_data) : list
+ load_datasets(x_train, y_train, x_test) : void
+ predict() : list
MainForm
+ window: window
- __init__() : void
- k_fold_cross_vaidation(clf, x_data, y_label) : list
+ run()() : void
MkNNAlgorithm
- _k : int
- _x_train : ndarray
- _y_train : ndarray
- _x_test : ndarray
- __int__(k_Neighbors) : void
- euclidean_distance(test_data, train_data, length) : float
- get_neighbors(training_set, labels, test_data, validityData) : list
- validity_value(trainingSet, labels) : list
- weight_voting(neighbors) : char
+ load_datasets(x_train, y_train, x_test) : void
+ predict() : list
evaluasi
- __int__() : void
+ get_process_memory() : int
+ get_process_time() : folat
+ get_precision() : folat
+ get_recall() : float
+ get_accuracy() : folat
1

Main.py

```
import PySimpleGUI as sg
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from Evaluasi import Evaluasi
from sklearn.model_selection import KFold
from Classifier import kNNAlgorithm, MkNNAlgorithm  # Classifier

class MainForm:

    def __init__(self):
        sg.theme('BlueMono')

        column1 = [
            [sg.Frame(title="Option", layout=[
                [sg.Radio("k-Nearest Neighbor", disabled=True, key="_rbkNN_", group_id="Option", enable_events=True,
                          default=True)],
                [sg.Radio("Modified k-Nearest Neighbor", disabled=True, key="_rbMkNN_", group_id="Option",
                          enable_events=True)]
            ])],
            [sg.Frame(title="Parameter k", layout=[
                [sg.Text(text="Parameter k", size=(10, 1)),
                 sg.Input(key="_parameter_k_", size=(13, 1), tooltip="Masukan nilai Integer", disabled=True)]
            ])],

            [sg.Button('Mulai', size=(6, 1), disabled=True ),
             sg.ProgressBar(max_value=1, orientation='h', size=(13, 16), key='proses')]
        ]

        column2 = [

            [sg.Frame("Hasil Klasifikasi", layout=[
                [sg.Text(text="Nilai Akurasi : ", size=(40, 1), key="_akurasi_")],
                [sg.Text(text="Nilai recall : ", size=(40, 1), key="_recall_")],
                [sg.Text(text="Nilai precision : ", size=(40, 1), key="_precision_")],
                [sg.Text(text="Waktu Komputasi : ", size=(40, 1), key="_waktu_komputasi_")],
                [sg.Text(text="Memori : ", size=(40, 1), key="_memori_")]
            ])]
        ]

        layout = [
            [sg.FileBrowse('Muat', size=(6, 1), target='_Path_',
                           initial_folder=r"C:\Users\PycharmProjects\MkNNPhishing\venv\Datasets",
                           file_types=[("csv Files", ".csv")]), sg.Column([
                [sg.Input(key="_Path_", disabled=True, size=(80, 1), enable_events=True)]
            ])],
            [sg.Text('-' * 159)],
            [sg.Column(column1), sg.VerticalSeparator(), sg.Column(column2)],
            [sg.Multiline(key="_output_", disabled=True, size=(90, 8))]
        ]

        self.window = sg.Window('kNN, MkNN', layout, default_element_size=(40, 1), grab_anywhere=False)


    def k_fold_cross_validation(self, clf, x_data, y_label):
        eval = Evaluasi()
        kf = KFold(n_splits=10, random_state=42, shuffle=False)
        evaluasi = []
        i = 1
        for train_index, test_index in kf.split(x_data):
            x_train, x_test, y_train, y_test = x_data[train_index], x_data[test_index] \
```

```
            , y_label[train_index], y_label[test_index]
        time_start = eval.get_process_time()
        clf.load_datasets(x_train, y_train, x_test)
        y_predict = clf.predict()
        mem_end = eval.get_process_memory()
        time_end = eval.get_process_time()
        evaluasi.append((i, eval.get_accuracy(y_test, y_predict), eval.get_recall(y_test, y_predict),
eval.get_precision(y_test, y_predict), (time_end - time_start), mem_end))
        i += 1
    return evaluasi

def run(self):

    X = None
    y = None
    while True:
        event, values = self.window.Read()
        file_path = ""
        akurasi = 0.0
        recall = 0.0
        precision = 0.0
        waktu_komputasi = 0.0
        memori = 0

        if event == "_Path_":

            try:
                file_path = Path(values['_Path_'])
                dataset = pd.read_csv(file_path, sep=',', header=None)
                if len(dataset.columns) <= 1:
                    raise ValueError
                X = dataset.iloc[:, :(dataset.shape[1] - 1)].values
                y = dataset.iloc[:, (dataset.shape[1] - 1)].values
                self.window.FindElement('_rbkNN_').Update(disabled=False)
                self.window.FindElement('_rbMkNN_').Update(disabled=False)
                self.window.FindElement('Mulai').Update(disabled=False)
                self.window.FindElement('_parameter_k_').Update(disabled=False)

            except ValueError:
                sg.Popup("\tFormat data salah !!\t", title="Error")


        parameter_k = None
        output = ""

        hasil_klasifikasi = []

        if event == 'Mulai':
            try:
                parameter_k = int(values['_parameter_k_'])

            except ValueError:
                sg.Popup("\nMasukan format data yang benar!!!\n Integer untuk parameter k", title="Error")

            else:
                self.window.FindElement('proses').UpdateBar(0, 5)
                if values["_rbkNN_"] == True:
                    kNN = kNNAlgorithm(k_Neighbors=parameter_k)
                    evaluasi = self.k_fold_cross_validation(kNN, X, y)
                    for eva in evaluasi:
                        akurasi = akurasi + eva[1]
                        recall = recall + eva[2]
                        precision = precision + eva[3]
                        waktu_komputasi = waktu_komputasi + eva[4]
```

```
                    memori = memori + eva[5]
                    output = output + f"Fold : {eva[0]}, Akurasi : {eva[1]} % , recall : {eva[2]:.4f}, precision :
{eva[3]:.4f},Waktu : {eva[4]} detik , Memori : {eva[5]} byte \n"
                    hasil_klasifikasi.append([f"{eva[1]}", f"{eva[2]}", f"{eva[3]}", f"{eva[4]}"  f"{eva[5]}"])
                akurasi = akurasi / 10
                recall = recall / 10
                precision = precision / 10
                waktu_komputasi = waktu_komputasi / 10
                memori = memori / 10

            elif values["_rbMkNN_"] == True:
                mKNN = MkNNAlgorithm(k_Neighbors=parameter_k)
                evaluasi = self.k_fold_cross_validation(mKNN, X, y)
                for eva in evaluasi:
                    akurasi = akurasi + eva[1]
                    recall = recall + eva[2]
                    precision = precision + eva[3]
                    waktu_komputasi = waktu_komputasi + eva[4]
                    memori = memori + eva[5]
                    output = output + f"Fold : {eva[0]}, Akurasi : {eva[1]} % , recall : {eva[2]}, precision : {eva[3]},Waktu :
{eva[4]} detik , Memori : {eva[5]} byte\n"
                    hasil_klasifikasi.append([f"{eva[1]}", f"{eva[2]}", f"{eva[3]}", f"{eva[4]}", f"{eva[5]}"])
                akurasi = akurasi / 10
                recall = recall / 10
                precision = precision / 10
                waktu_komputasi = waktu_komputasi / 10
                memori = memori / 10

            self.window.FindElement("_akurasi_").Update("Akurasi :" + repr(akurasi) + "%")
            self.window.FindElement("_recall_").Update("Recall : " + repr(recall))
            self.window.FindElement("_precision_").Update("Precision : " + repr(precision))
            self.window.FindElement("_waktu_komputasi_").Update("Waktu Komputasi : " +repr(waktu_komputasi)+
"detik")
            self.window.FindElement("_memori_").Update("Memori : "+ repr(memori)+ "byte")
            self.window.FindElement("_output_").Update(output)

        if event is None or event == 'Exit':
            self.window.Close()
            break
        print(event, values)

if __name__ == "__main__":
    form = MainForm()
    form.run()
```

classifier.py

```
from operator import itemgetter
import numpy as np

class kNNAlgorithm:

    def __init__(self, k_Neighbors):
        self.__k = k_Neighbors

    # Memuat data
    def load_datasets(self, x_train, y_train, x_test):
        self.__x_train = x_train
        self.__y_train = y_train
        self.__x_test = x_test

    # Jarak Euclidean Distance
    def euclidean_distance(self, test_data, train_data, length):
        distance = 0
```

```
        for dx in range(length):
            distance += np.square(test_data[dx] - train_data[dx])
        return np.sqrt(distance)

    # menghitung jarak data trining dan data testing
    def get_neighbors(self, training_set, labels, test_data):
        distances = []
        for dx in range(len(training_set)):
            dist = self.euclidean_distance(test_data, training_set[dx], (len(test_data)))
            distances.append((training_set[dx], dist, labels[dx]))

        distances.sort(key = itemgetter(1))
        neighbors = distances[:self.__k]
        return neighbors

    # Voting hasil prediksi berdasar nilai n tetangga terdekat
    def class_vote(self, neighbors):
        classVotes = {}
        for dx in range(len(neighbors)):
            vote = neighbors[dx][-1]
            if vote in classVotes:
                classVotes[vote] += 1
            else:
                classVotes[vote] = 1
        sortedVoted = sorted(iter(classVotes.items()), key = itemgetter(1), reverse=True)
        return sortedVoted[0][0]

    def predict(self):
        prediction = []
        neighbors = []

        for dx in range(len(self.__x_test)):
            neighbors = self.get_neighbors(self.__x_train, self.__y_train, self.__x_test[dx])
            result = self.class_vote(neighbors=neighbors)
            prediction.append(result)
        return prediction

class MkNNAlgorithm:

    def __init__(self, k_Neighbors):
        self.__k = k_Neighbors

    def load_datasets(self, x_train, y_train, x_test):
        self.__x_train = x_train
        self.__y_train = y_train
        self.__x_test = x_test

    def euclidean_distance(self, test_data, train_data, length):
        distance = 0
        for dx in range(length):
            distance += np.square(test_data[dx] - train_data[dx])
        return np.sqrt(distance)

    def get_neighbors(self, training_set, labels, test_data, validityData):
        distances = []
        for dx in range(len(training_set)):
            dist = self.euclidean_distance(test_data, training_set[dx], (len(test_data)))
            distances.append((training_set[dx], dist, validityData[dx], labels[dx]))

        distances.sort(key = itemgetter(1))
        neighbors = distances[:self.__k]
        return neighbors

    def validity_value(self, trainingSet, labels):
        validity = []
        length = len(trainingSet[0]) - 1

        x = 0
        for x in range(len(trainingSet)):
```

```
            distances = []
            dist = []
            y = 0
            for y in range(len(trainingSet)):
                if x == y: continue
                dist = self.euclidean_distance(trainingSet[x], trainingSet[y], length)
                distances.append((trainingSet[y], dist, labels[y]))
            distances.sort(key = itemgetter(1))

            neighbors = []  # variable tetangga terdekat ke data uji
            # menggambil k tetangga terdekat
            neighbors = distances[:self.__k]

            validityData = 0
            # menghitung validitas data latih terhadap k data tetangga terdekatnya
            for dx in range(self.__k):
                if labels[x] == neighbors[dx][-1]: validityData += 1
            validity.append(validityData / self.__k)
        return validity

    def weight_voting(self, neighbors):
        weight = []
        # menghitung weight voting berdasarkan nilai validity dan jarak antar data latih dan data uji
        for dx in range(len(neighbors)):
            w = neighbors[dx][2] * (1 / (neighbors[dx][1] + 0.5))
            weight.append((neighbors[dx], w))

        weightVotes = {}
        # menjumlahkan nilai bobot dengan label yang sama
        # label dengan nilai bobot yang terbesar akan di pilih sebagai label klasifikasi data uji
        for dx in range(len(weight)):
            response = weight[dx][0][-1]
            if response in weightVotes:
                weightVotes[response] += weight[dx][-1]
            else:
                weightVotes[response] = weight[dx][-1]
        sortedWeightVotes = sorted(iter(weightVotes.items()), key=itemgetter(1), reverse=True)
        return sortedWeightVotes[0][0]

    def predict(self):
        prediction = []
        neighbors = []
        validityData = self.validity_value(self.__x_train, self.__y_train)

        for dx in range(len(self.__x_test)):
            neighbors = self.get_neighbors(self.__x_train, self.__y_train, self.__x_test[dx], validityData)
            result = self.weight_voting(neighbors)
            prediction.append(result)
        return prediction
```

evaluasi.py

```
import os
import psutil
import timeit
import numpy as np
from sklearn.metrics import confusion_matrix

class Evaluasi:

    def __init__(self):
        pass

    def get_process_memory(self):
        process = psutil.Process(os.getpid())
        return process.memory_info().rss
```

```
def get_process_time(self):
    process_time = timeit.default_timer()
    return process_time

def get_recall(self, y_test, predict):
    a = np.unique(y_test)
    cm = confusion_matrix(y_test, predict, labels=[a[0], a[1]])
    TP = cm[1, 1]
    FN = cm[1, 0]
    recall =(TP / (TP + FN))
    return recall

def get_precision(self, y_test, predict):
    a = np.unique(y_test)
    cm = confusion_matrix(y_test, predict, labels=[a[0], a[1]])
    TP = cm[1, 1]
    FP = cm[0, 1]
    precision = (TP / (TP + FP))
    return precision

def get_accuracy(self, y_test, predict):
    a = np.unique(y_test)
    cm = confusion_matrix(y_test, predict, labels=[a[0], a[1]])
    TP = cm[1, 1]
    FP = cm[0, 1]
    TN = cm[0, 0]
    FN = cm[1, 0]
    accuracy =((TP + TN) / (TP + TN + FP + FN))
    return accuracy
```