**Laporan Tugas Kecil 2**

**IF2211 - Strategi Algoritma**

**Semester II Tahun Ajaran 2021/2022**

**Implementasi *Convex Hull* untuk Visualisasi Tes *Linear Separability Dataset* dengan Algoritma *Divide and Conquer***

Disusun oleh:

Amar Fadil

13520103

Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung

Jl. Ganesha 10, Bandung 4013

**Daftar Isi**

[A. Algoritma Convex Hull 3](#_Toc96852692)

[B. Source Code 4](#_Toc96852693)

[1. lib.py 4](#_Toc96852694)

[2. utils.py 12](#_Toc96852695)

[3. types.py 14](#_Toc96852696)

[4. \_\_main\_\_.py 14](#_Toc96852697)

[C. Screenshot 17](#_Toc96852698)

[1. Dataset iris 17](#_Toc96852699)

[a) petal length (cm) vs petal width (cm) 17](#_Toc96852700)

[b) sepal length (cm) vs sepal width (cm) 17](#_Toc96852701)

[2. Dataset wine 18](#_Toc96852702)

[a) flavanoids vs nonflavanoid\_phenols 18](#_Toc96852703)

[b) ash vs alcalinity\_of\_ash 18](#_Toc96852704)

[3. Dataset breast\_cancer 18](#_Toc96852705)

[a) concavity error vs concave points error 19](#_Toc96852706)

[b) mean area vs worst area 19](#_Toc96852707)

[4. Dataset water potability 19](#_Toc96852708)

[a) pH vs Hardness 20](#_Toc96852709)

[b) Sulfate vs Conductivity 20](#_Toc96852710)

[Assistant Checklist 21](#_Toc96852711)

# A. Algoritma Convex Hull

Algoritma dalam mencari convex hull

# B. Source Code

## lib.py

"""

Main class definitions

"""

import numpy as np

import pandas as pd

from itertools import cycle

from matplotlib import pyplot as plt

from typing import Dict, Iterable, List, Tuple

from myConvexHull.types import Feature, Line, Point, PointIndex, LineIndex

from myConvexHull.utils import det, dist\_to\_line

class ConvexHull(object):

    def \_\_init\_\_(self, dt: Iterable):

        """Create new convex hull instance.

        It will auto process the data by generating

        the convex hull. Only works for static 2D points.

        Args:

            dt (Iterable): List of 2D points, where each element

                is an iterable that has two number, (x, y).

        """

        dt: List[Point] = [(p[0], p[1]) for p in dt]

        self.points = dt

        """All points inside and in the convex hull.

        """

        self.vertices: List[PointIndex] = []

        """List of the point/vertex in the convex hull. Each element

        is an index from self.points.

        """

        self.simplices: List[LineIndex] = []

        """List of the line/edge in the convex hull. Each element

        is a tuple, that is a pair of two index from self.points.

        """

        self.\_\_convexHull()

    def \_\_dnc\_convexHull(self, dt: List[PointIndex], line: LineIndex):

        """Divide and Conquer algo of convex hull.

        It is based on quickhull algorithm.

        Args:

            dt (List[int]): Points to check outside the line.

                Each element is index to self.points.

            line (List[int, int]): Line to check the points.

                Each element is a pair of index to self.points.

        """

        # There are two base cases

        # 1. If there are no point left, then we are done.

        #    Add the line to the simplices list.

        if len(dt) == 0:

            self.simplices += [line]

        # 2. If there is only one point left, then the hull vertices

        #    must be the point. Add the point to vertices list.

        #    Make two lines that starts from point to each line point.

        #    Add both lines to the simplices list (edge of the hull).

        elif len(dt) == 1:

            self.simplices += [

                (dt[0], line[0]),

                (dt[0], line[1]),

            ]

            self.vertices.append(dt[0])

        # Recursive case

        else:

            # DIVIDE

            # 0. Get the line points in coord from self.points

            pline = (self.points[line[0]], self.points[line[1]])

            # 1. Get a point that has maximum distance to the line

            pmax = max(

                dt,

                key=lambda x: dist\_to\_line(pline, self.points[x])

            )

            # 1.1 The point above is the new vertices of the hull

            #     Add the point to the vertices list

            self.vertices.append(pmax)

            # 1.2 Remove that max point from the list

            dt.remove(pmax)

            # 2. Create two new lines that starts from each point

            #    in the line until the max point.

            newline: Tuple[LineIndex, LineIndex] = (

                (line[0], pmax),

                (line[1], pmax),

            )

            # 2.1. Get the points of both line (remember that the

            #      new line from above is just the indexes).

            pnewline: Tuple[Line, Line] = tuple(

                (self.points[p[0]], self.points[p[1]])

                for p in newline

            )

            # 3. Get the points that are outside both the new line

            #    Also split the points into two groups, that is either

            #    outside the first line or the second line.

            dt\_split: List[List[PointIndex], List[PointIndex]] = [[], []]

            for p in dt:

                # First line is in the left side, so the point outside this

                # must be in the left side too, which has determinant of > 0

                if det(pnewline[0], self.points[p]) > 0:

                    dt\_split[0].append(p)

                # Second line is in the right side, so the point outside this

                # must be in the right side too, which has determinant of < 0

                elif det(pnewline[1], self.points[p]) < 0:

                    dt\_split[1].append(p)

            # COMBINE & CONQUER

            # 4. Recursive call

            # 4.1 Check for points outside the first line

            self.\_\_dnc\_convexHull(dt\_split[0], newline[0])

            # 4.2 Check for points outside the second line.

            #     Reverse the order of the line points because we have

            #     to keep side convention (if not reversed, left will

            #     be right and vice versa).

            self.\_\_dnc\_convexHull(dt\_split[1], newline[1][::-1])

    def \_\_convexHull(self):

        """The first step before recursive DnC algo.

        """

        # Get index list of all points

        dt = [i for i in range(len(self.points))]

        # Base case:

        # 1. If there is less than 2 points,

        #    it doesn't have any convex hull, skip.

        # 2. If there is only 2 points, then the hull

        #    is just the line between them.

        if (len(dt) == 2):

            self.vertices = dt

            self.simplices = [(dt[0], dt[1])]

        # If there are more than 2 points,

        # then we need to check for some things

        else:

            # Sort the points ascending by their x and y coordinate

            dt.sort(key=lambda x: self.points[x])

            # Get the line that start from minimum point

            # to maximum point based on their x coordinate.

            line = (dt[0], dt[-1])

            # Both points are the vertex of the hull, so add them

            # to the vertices list.

            self.vertices.extend(line)

            # Also get the actual points instead of the indexes.

            pline = (self.points[line[0]], self.points[line[1]])

            # Remove min and max point from the list

            dt = dt[1:-1]

            # Divide the points into two groups, that is either

            # is in the left side or the right side of the line.

            dt\_split = [[], []]

            for p in dt:

                d = det(pline, self.points[p])

                # If the determinant is > 0, then the point is

                # in the left side of the line.

                if d > 0:

                    dt\_split[0].append(p)

                # If the determinant is < 0, then the point is

                # in the right side of the line.

                elif d < 0:

                    dt\_split[1].append(p)

            # Base case 3

            if len(dt\_split[0]) + len(dt\_split[1]) == 0:

                # If points are in the same line, then the hull will

                # be the line that start from minimum point to

                # maximum point.

                self.simplices = [line]

                self.vertices = [\*line]

            # Recursive case

            else:

                # COMBINE & CONQUER

                # Get convex hull from the left side of the line

                self.\_\_dnc\_convexHull(dt\_split[0], line)

                # Get convex hull from the right side of the line

                #  Reverse the order of the line points because we have

                #  to keep side convention (if not reversed, left will

                #  be right and vice versa).

                self.\_\_dnc\_convexHull(dt\_split[1], line[::-1])

# Color cycle constant

COLOR\_CYCLE = cycle([

    '#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd',

    '#8c564b', '#e377c2', '#7f7f7f', '#bcbd22', '#17becf'

])

class LinearSeparabilityDataset(object):

    def \_\_init\_\_(self,

        frame: pd.DataFrame,

        target\_names: Iterable,

        feature\_names: Iterable=None,

        target\_key: str='target',

        backend: ConvexHull=ConvexHull

    ) -> None:

        """Create new instance of Linearly Separable Data.

        Useful to easy visualize the data given their dataset.

        Target is the predicted column (y) of the rows given their

        features, while features is the column data other than

        target itself (x). You HAVE to always provide the target

        names. If `feature\_names` is not provided, then it will

        automatically get the column names from the `frame` excluding

        the `target\_key`.

        This class will lazy compute the convex hull, meaning that it

        will compute the convex hull of feature pair only once, that

        is when you first time call `getConvex` or `visualize` for

        that feature pair.

        Make sure:

        1. `target\_names` has the same length as unique value

        counts on the `frame['target']`.

        2. `feature\_names` has the same length as total column

        in the `frame` excluding `target\_key`.

        3. `target\_key` is in the frame.

        In any case those condition not met, the program will

        raise exception.

        Args:

            frame (pd.DataFrame): Dataframe of the dataset.

                It should include both target and its features.

            target\_names (np.ndarray): Names of the target.

            feature\_names (list): Names of the features.

            target\_key (str, optional): Target column name.

                Defaults to 'target'.

            backend (ConvexHull, optional): Convex hull computation

                backend. Defaults to custom ConvexHull.

        Raises:

            ValueError: If the length of `target\_names` or

            `feature\_names` is not qualified.

            KeyError: If `target\_key` not exists in the frame.

        """

        if len(target\_names) != frame[target\_key].nunique():

            raise ValueError(

                "The length of `target\_names` should be equal to "

                "the unique value counts on `frame[target\_key]` "

                "(Expected {} but got {}).".format(

                    frame[target\_key].nunique(),

                    len(target\_names),

                )

            )

        if feature\_names is None:

            feature\_names = list(frame.columns)

            feature\_names.remove(target\_key)

        elif len(feature\_names) != len(frame.columns) - 1:

            raise ValueError(

                "The length of `feature\_names` should be the same "

                "as the total column in the `frame` excluding "

                "`target\_key` (Expected {} but got {}).".format(

                    len(frame.columns) - 1,

                    len(feature\_names),

                )

            )

        if target\_key not in frame.columns:

            raise KeyError(

                "The `target\_key` should be in the frame."

            )

        self.\_\_convex: Dict[str, List[ConvexHull]] = {}

        """List of convex hull for each target and for each

        pair of features. The key is joined index of both

        feature in the pair, separated by ';'.

        """

        self.target\_key = target\_key

        """Target column name in the dataframe.

        """

        self.frame = frame

        """Dataframe of the dataset, consist of both

        its features and target.

        """

        self.target\_names = target\_names

        """List of the target name/label.

        """

        self.feature\_names = feature\_names

        """List of the feature name/label.

        """

        self.backend = backend

        """Backend of the convex hull library.

        """

    def \_\_getPair(self, pair1: Feature, pair2: Feature) -> Tuple[int, int]:

        """Get the feature pair index.

        It will get a pair of the feature index

        given both feature name.

        Args:

            pair1 (int | str): Feature pair 1.

            pair2 (int | str): Feature pair 2.

        Returns:

            Tuple[int, int]: Pair of the feature index.

        """

        return ((

            self.feature\_names.index(pair1)

            if isinstance(pair1, str) else pair1,

            self.feature\_names.index(pair2)

            if isinstance(pair2, str) else pair2,

        ))

    def \_\_calculate(self, key:str, p1: int, p2: int) -> None:

        """Calculate the convex hull for each target.

        Args:

            key (str): Key in the dictionary of convex hull.

            p1 (int): First feature index.

            p2 (int): Second feature index.

        """

        self.\_\_convex[key] = []

        for i in range(len(self.target\_names)):

            # Get the dataframe with same target name.

            bucket = self.frame[self.frame[self.target\_key] == i]

            # Get the pair of values from both features.

            # It will be our points.

            bucket = bucket.iloc[:, [p1, p2]].values

            # Create the convex hull and append it to the list.

            self.\_\_convex[key].append(self.backend(bucket))

    def getConvex(self, pair1: Feature, pair2: Feature) -> List[ConvexHull]:

        """Get convex hull given pair of features.

        Pair of features can be given by their index or their name.

        Args:

            pair1 (int | str): First feature.

            pair2 (int | str): Second feature.

        Returns:

            List[ConvexHull]: List of convex hull for each target.

        """

        # Get the pair of feature index.

        pair1, pair2 = self.\_\_getPair(pair1, pair2)

        # Get the key to use in the convex hull dictionary.

        key = ';'.join([str(pair1), str(pair2)])

        # If the convex hull is already calculated, just return it.

        if key in self.\_\_convex:

            return self.\_\_convex[key]

        # If the convex hull is not calculated,

        # calculate and return it.

        self.\_\_calculate(key, pair1, pair2)

        return self.\_\_convex[key]

    def visualize(self,

        pair1: Feature,

        pair2: Feature,

        figsize: Tuple[int, int]=(10, 6),

        captions: bool=True,

        title: str=None,

        xlabel: str=None,

        ylabel: str=None,

    ) -> None:

        """Visualize the data given pair of features.

        Pair of features can be given by their index or their name.

        Args:

            pair1 (int | str): First feature.

            pair2 (int | str): Second feature.

            figsize (Tuple[int, int], optional): Figure size.

                Defaults to (10, 6).

            captions (bool, optional): Enable caption label.

                Consist of title, xlabel and ylabel. Defaults to True.

            title (str, optional): Title of the figure.

                Defaults to None.

            xlabel (str, optional): Label on the x side of the figure.

                Defaults to None.

            ylabel (str, optional): Label on the y side of the figure.

                Defaults to None.

        """

        # Get the convex of the pair of feature.

        data = self.getConvex(pair1, pair2)

        # Get the pair of feature index.

        pair1, pair2 = self.\_\_getPair(pair1, pair2)

        # Create new figure.

        plt.figure(figsize=figsize)

        # Write captions if enabled.

        if captions:

            # Get the default x and y label

            if xlabel is None:

                xlabel = self.feature\_names[pair1]

            if ylabel is None:

                ylabel = self.feature\_names[pair2]

            # Write the title, x, and y label.

            plt.title(title if title else f'{xlabel} vs {ylabel}')

            plt.xlabel(xlabel)

            plt.ylabel(ylabel)

        # Plot the convex hull for each target.

        for i in range(len(self.target\_names)):

            # Get current color

            col = next(COLOR\_CYCLE)

            # Get the bucket points

            bucket = np.array(data[i].points)

            # Visualize the points with scatter plot.

            # Label them with its corresponding target name.

            plt.scatter(

                bucket[:, 0],

                bucket[:, 1],

                label=self.target\_names[i],

                color=col,

            )

            # Visualize the convex hull.

            # Plot the simplices lines from the convex hull.

            for simplex in data[i].simplices:

                plt.plot(bucket[simplex, 0], bucket[simplex, 1], color=col)

        # Show legends and then show the figure.

        plt.legend()

        plt.show()

## utils.py

"""

Basic utility tools for the library.

Contains many useful functions for processing and computing.

"""

from math import isclose, sqrt

from myConvexHull.types import Vector, Line, Point

def vec\_len(v: Vector) -> float:

    """Calculate the length of a vector.

    Args:

        v (Vector): Vector reference.

    Returns:

        float: Length of he vector.

    """

    return sqrt(v[0] \*\* 2 + v[1] \*\* 2)

def dist\_to\_line(l: Line, p: Point) -> float:

    """Calculate the distance between a point and a line.

    Original Formula

    Distance between a point and a line given the line equation.

    d = |am+bn+c|/sqrt(a\*a+b\*b) ... (1)

    where line equation ax + by + c = 0 ... (2)

    Line equation given two points:

    (y-y1)/(x-x1) = (y2-y1)/(x2-x1)

    (y-y1)(x2-x1) = (y2-y1)(x-x1)

    (y-y1)(x2-x1) - (x-x1)(y2-y1) = 0

    x2y-x1y-x2y1+x1y1 - (xy2-xy1-x1y2+x1y1) = 0

    x2y - x1y - x2y1 + x1y1 - xy2 + xy1 + x1y2 - x1y1 = 0

    (x2-x1)y - x2y1 + x1y1 - x(y2-y1) + x1y2 - x1y1 = 0

    (y1-y2)x + (x2-x1)y - x2y1 + x1y2 = 0 ... (3)

    We can infer from (2) and (3) that

    a = y1-y2

    b = x2-x1

    c = -x2\*y1 + x1\*y2

    Thus the final distance formula is:

    d = abs(a\*m + b\*n + c) / sqrt(a^2 + b^2)

    Args:

        l (Line): Line reference.

        p (Point): Point distance reference.

    Returns:

        float: Distance between a point and a line.

    """

    a = l[0][1] - l[1][1]

    b = l[1][0] - l[0][0]

    c = - l[1][0] \* l[0][1] + l[0][0] \* l[1][1]

    return abs(a \* p[0] + b \* p[1] + c) / vec\_len((a, b))

def det(l: Line, p: Point) -> float:

    """Calculate the determinant between a point and a line.

    Args:

        l (Line): Line reference.

        p (Point): Point determinant reference.

    Returns:

        float: Determinant between a point and a line.

            = 0: point is on the line.

            > 0: point is on the left side of the line.

            < 0: point is on the right side of the line.

    """

    res = (

        p[0] \* l[0][1]

        + l[1][0] \* p[1]

        + l[0][0] \* l[1][1]

        - l[1][0] \* l[0][1]

        - l[0][0] \* p[1]

        - p[0] \* l[1][1]

    )

    if isclose(res, 0, abs\_tol=1e-13):

        res = 0

    return res

## types.py

"""

Custom type definitions.

"""

from typing import Tuple, Union

Vector = Point = Tuple[float, float]

PointIndex = int

Line = Tuple[Point, Point]

LineIndex = Tuple[PointIndex, PointIndex]

Feature = Union[int, str]

## \_\_main\_\_.py

import argparse

import pandas as pd

from myConvexHull.lib import LinearSeparabilityDataset

# Argument Parser

parser = argparse.ArgumentParser(

    description=' '.join([

        'Main driver of linear separability dataset visualizer.',

        'It will generate a plot of convex hull given a dataset.',

    ]),

    epilog=' '.join([

        'NOTE: You can specify the dataset by either',

        'file or dataset name, but NOT BOTH at the ',

        'same time. You always have to specify at least one',

        'of the feature pair you want to visualize.\n\n',

        'If you are using the file input mode, please make sure:\n',

        '(1) The file is in csv format.\n',

        '(2) The file started by a row for header/column name,',

        'continued by the row values.\n',

        '(3) There is a column named "target" for the target value.',

        'If the target column name is different, please specify',

        'the target column name with -tk/--target\_key option.\n',

        '(4) Target column consist of whole number and cannot be skipped',

        '(e.g. you have 3 rows of values, first target is 1, second',

        'target is 3, and third target is 0, then this data is not valid',

        'because it skipped no. 2).\n',

        '(5) Specify the target names / label by -tn/--target\_names option.',

        'Target names should be ordered starting from label for target = 0.',

    ])

)

# Group File Dataset

ginput = parser.add\_argument\_group('File Dataset Input')

ginput.add\_argument('-f', '--file', help='Input datasets file. Should have minimum 3 columns: 2 features and a target.')

ginput.add\_argument('-tk', '--target\_key', help='Target column name.', default='target')

ginput.add\_argument('-tn', '--target\_names', nargs='+', help='Target name list, separated by space.')

# Group Sklearn Dataset

tinput = parser.add\_argument\_group('Sklearn Dataset Input')

tinput.add\_argument('-n', '--dataset\_name', help='Name of the dataset.')

# Group visualization options

vopt = parser.add\_argument\_group('Visualization Options')

vopt.add\_argument('-fp', '--feature\_pair', nargs=2, action='append', help='Feature pair to plot. Should be separated by space. You can supply multiple pair of feature.', required=True)

vopt.add\_argument('-s', '--size', nargs=2, type=int, help='Figure size (width, height) of the plot.', default=(10, 6))

vopt.add\_argument('-nc', '--no\_captions', help='Disable captions (title, x/y label).', action='store\_true')

args = parser.parse\_args()

# Throw error if no dataset specified

if args.dataset\_name is None and args.file is None:

    parser.error('Either dataset name or file should be supplied.')

# Throw error if both mode (file and dataset name) is specified

elif args.dataset\_name is not None and args.file is not None:

    parser.error('Only one mode can be used, either dataset name or file should be supplied but not both.')

# Load and create visualizer object

vis: LinearSeparabilityDataset = None

if args.dataset\_name:

    # Lazy load sklearn datasets

    from sklearn import datasets

    # Get the dataset in sklearn

    f = getattr(datasets, f'load\_{args.dataset\_name}')

    # If specified dataset name is not in sklearn, throw error.

    if f is None:

        parser.error(f'Dataset with name "{args.dataset\_name}" is not exists.')

    # Create the dataset object

    data = f(as\_frame=True)

    vis = LinearSeparabilityDataset(

        frame=data.frame,

        target\_names=data.target\_names,

    )

else:

    # Load the dataset from file

    data = pd.read\_csv(args.file)

    data.dropna(inplace=True)

    vis = LinearSeparabilityDataset(

        frame=data,

        target\_key=args.target\_key,

        target\_names=args.target\_names,

    )

# Sanitize the feature pair

# (integer if number, else string)

args.feature\_pair = [

    [

        int(fp[i])

        if fp[i].isnumeric()

        else fp[i]

        for i in range(2)

    ]

    for fp in args.feature\_pair

]

# Visualize each feature pair

for fp in args.feature\_pair:

    vis.visualize(

        fp[0], fp[1],

        figsize=args.size,

        captions=(not args.no\_captions),

    )

# C. Screenshot

## Dataset iris

Dua pasang fitur ini dijalankan dengan perintah berikut:

python -m myConvexHull -n iris -fp "petal length (cm)" "petal width (cm)" -fp "sepal length (cm)" "sepal width (cm)"

### petal length (cm) vs petal width (cm)

Chart, scatter chart

Description automatically generated

### sepal length (cm) vs sepal width (cm)

Chart, scatter chart

Description automatically generated

## Dataset wine

Dua pasang fitur ini dijalankan dengan perintah berikut:

python -m myConvexHull -n wine -fp "flavanoids" "nonflavanoid\_phenols" -fp "ash" "alcalinity\_of\_ash"

### flavanoids vs nonflavanoid\_phenols

Chart, radar chart, scatter chart

Description automatically generated

### ash vs alcalinity\_of\_ash

Chart, scatter chart

Description automatically generated

## Dataset breast\_cancer

Dua pasang fitur ini dijalankan dengan perintah berikut:

python -m myConvexHull -n breast\_cancer -fp "concavity error" "concave points error" -fp "mean area" "worst area"

### concavity error vs concave points error

Chart, line chart

Description automatically generated

### mean area vs worst area

Chart, scatter chart

Description automatically generated

## Dataset water potability

Dataset ini diambil dari open source database di kaggle yang dapat diunduh [disini](https://www.kaggle.com/adityakadiwal/water-potability). Dua pasang fitur ini dijalankan dengan perintah berikut:

python -m myConvexHull -f "datasets/water\_potability.csv" -tn "Not Potable" "Potable" -tk "Potability" -fp 0 1 -fp Sulfate Conductivity

### pH vs Hardness

Chart, scatter chart

Description automatically generated

### Sulfate vs Conductivity

Chart, scatter chart

Description automatically generated

# Lampiran

Assistant Checklist

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Pustaka *myConvexHull* berhasil dibuat dan tidak ada kesalahan | √ |  |
| 1. *Convex hull* yang dihasilkan sudah benar | √ |  |
| 1. Pustaka *myConvexHull* dapat digunakan untuk menampilkan *convex hull* setiap label dengan warna yang berbeda. | √ |  |
| 1. **Bonus**: program dapat menerima input dan menuliskan output untuk dataset lainnya. | √ |  |

Link Repository