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
B. Source Code 5
1. lib.py 5
2. utils.py 13
3. types.py 15
4. __main__.py 15
C. Screenshot 18
1. Dataset iris 18
a) petal length (cm) vs petal width (cm) 18
b) sepal length (cm) vs sepal width (cm) 18
2. Dataset wine 19
a) flavanoids vs nonflavanoid_phenols 19
b) ash vs alcalinity_of_ash 19
3. Dataset breast_cancer 19
a) concavity error vs concave points error 20
b) mean area vs worst area 20
4. Dataset water potability 20
a) pH vs Hardness 21
b) Sulfate vs Conductivity 21
Lampiran 22
Assistant Checklist 22
Link Repository 22

A. Algoritma Convex Hull

Algoritma dalam mencari convex hull disadur dari salindia Algoritma Divide and Conquer Bagian 4 untuk Convex Hull. Ide dasar algortima ini mirip seperti quicksort. Terdapat dua tahap utama dalam mencari convex hull. Tahapan pertama (implementasi `__convexHull` pada kelas `ConvexHull` di modul `lib`) yang dilakukan program ini adalah sebagai berikut:

1. Ambil daftar indeks dari daftar semua titik pada range [0, banyaknya titik).
2. **[Conquer] Kasus basis 1**: Jika terdapat kurang dari 2 titik, tidak melakukan apa-apa.
3. **[Conquer] Kasus basis 2**: Jika terdapat tepat 2 titik, maka vertex pembentuk convex hull adalah dua titik tersebut dan simplices pembentuk convex hull adalah garis yang dibentuk dari dua titik tersebut.
4. Jika terdapat lebih dari 2 titik, urutkan titik-titik berdasarkan koordinat x dan y-nya.
5. Ambil garis yang terbentuk dari titik pertama dan terakhir yang telah diurutkan (titik pertama berada paling kiri, titik kedua berada paling kanan), tambahkan kedua titik menjadi vertices pembentuk convex.
6. Hapus titik pertama dan terakhir pada daftar titik.
7. **[Divide]** Bagi titik-titik yang tersisa di daftar titik menjadi dua bagian: sebelah kiri atau sebelah kanan. Untuk setiap titik pada daftar titik, periksa determinan dari titik tersebut ke garis yang dibentuk sebelumnya, jika > 0, kelompokkan menjadi titik di sebelah kiri, jika < 0, kelompokkan menjadi titik di sebelah kanan, jika = 0, abaikan titik karena tak membentuk convex.
8. **[Conquer] Kasus basis 3**: Jika setelah pembagian titik, tidak ada titik yang tersisa (semua titik selain pembentuk garis berada pada garis), maka vertices merupakan kedua titik pembentuk garis tersebut dan simplices merupakan garis tersebut.
9. **[Combine] Rekursif**: Lakukan tahap 2 jika masih ada titik tersisa. Tahap 2 dilakukan sebanyak dua kali, pertama untuk kelompok pertama yang berada di kiri garis, kedua untuk kelompok kedua yang berada di kanan garis. Khusus saat pemanggilan kedua, garis dibalik sehingga dimulai dari titik paling ujung kanan (maks) sampai ke titik paling ujung kiri (min). Hal ini dilakukan untuk melakukan prinsip DIY dengan tidak membagi dua fungsi DnC yang relatif sama, sekaligus menjaga konsistensi pengecekan bagian kiri dan kanan relatif.

Tahapan kedua (implementasi `__dnc_convexHull` pada kelas `ConvexHull` di modul `lib`) menerima sebagian titik yang telah dibagi dengan garis yang mengikutinya. Tahapan kedua yang dilakukan program ini adalah sebagai berikut:

1. **[Conquer] Kasus Basis 1**: Jika tidak ada titik tersisa, maka simplices pembentuk convex hull juga termasuk garis pada parameter. Tambahkan garis pada daftar simplices (tidak perlu menambahkan titik pembentuk garis ke dalam daftar vertices, karena pasti sudah ditambah sebelumnya, sehingga tidak terjadi duplikat).
2. **[Conquer] Kasus Basis 2**: Jika hanya ada satu titik tersisa (anggapannya seperti segitiga dengan alas berupa garis pada parameter dan tingginya adalah titik yang tersisa ini), maka simplices pembentuk convex hull juga termasuk dua garis yang dibentuk: dari titik tersisa ke titik awal garis, dan dari titik tersisa ke titik akhir garis. Tambahkan kedua garis pada daftar simplices. Tambahkan juga titik tersisa pada daftar vertices (titik pembentuk garis tidak perlu ditambahkan kembali).
3. Jika banyak titik lebih dari 1, cari titik yang memiliki jarak paling jauh dari garis parameter. Misalnya titik ini diberi nama pmax. Pmax ini termasuk dalam vertices convex hull. Tambahkan pmax ke dalam daftar vertices dan hapus pmax dalam daftar titik yang akan dibagi.
4. Buat dua garis baru, garis pertama (L1) dimulai dari titik awal garis hingga ke pmax dan garis kedua (L2) dimulai dari pmax hingga ke titik akhir garis.

5. **[Divide]** Bagi daftar titik dalam dua bagian, kelompok pertama adalah titik yang berada di luar garis pertama, sedangkan kelompok kedua adalah titik yang berada di luar garis kedua. Untuk semua titik yang akan dibagi:
    a. Jika determinan antara titik dengan garis pertama > 0 (di luar segitiga/di sisi kiri relatif pada vektor L1), maka titik berada pada kelompok pertama.
    b. Jika determinan antara titik dengan garis kedua > 0 (di luar segitiga/di sisi kiri relatif pada vektor L2), maka titik berada pada kelompok kedua.
    c. Untuk titik lain yang tidak memenuhi salah satu syarat diatas, abaikan karena berada di dalam segitiga sehingga tidak mungkin membentuk convex hull.
6. **[Combine] Rekursif**: Lakukan pemanggilan tahap kedua kembali sebanyak dua kali: pertama dengan daftar titik pada kelompok pertama dengan garis pertama, kedua dengan daftar titik pada kelompok kedua dengan garis kedua.

Tahap *conquer* atau penyelesaian subproblem terkecil yakni kasus basis akan selalu memberhentikan jalannya fungsi.

B. Source Code

1. 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),
                (pmax, line[1]),
            )
            # 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 vector (p1,pmax), so the point outside this
                # must be in the left side, which has determinant of > 0
                if det(pnewline[0], self.points[p]) > 0:
                    dt_split[0].append(p)
                # Second line is vector (pmax,p2), so the point outside this
                # must be in the left side (relative to vector direction) 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.
            self.__dnc_convexHull(dt_split[1], newline[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
        elif len(dt) > 2:
            # 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()
```

2. 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
```

3. 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]
```

4. __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),
    )
```

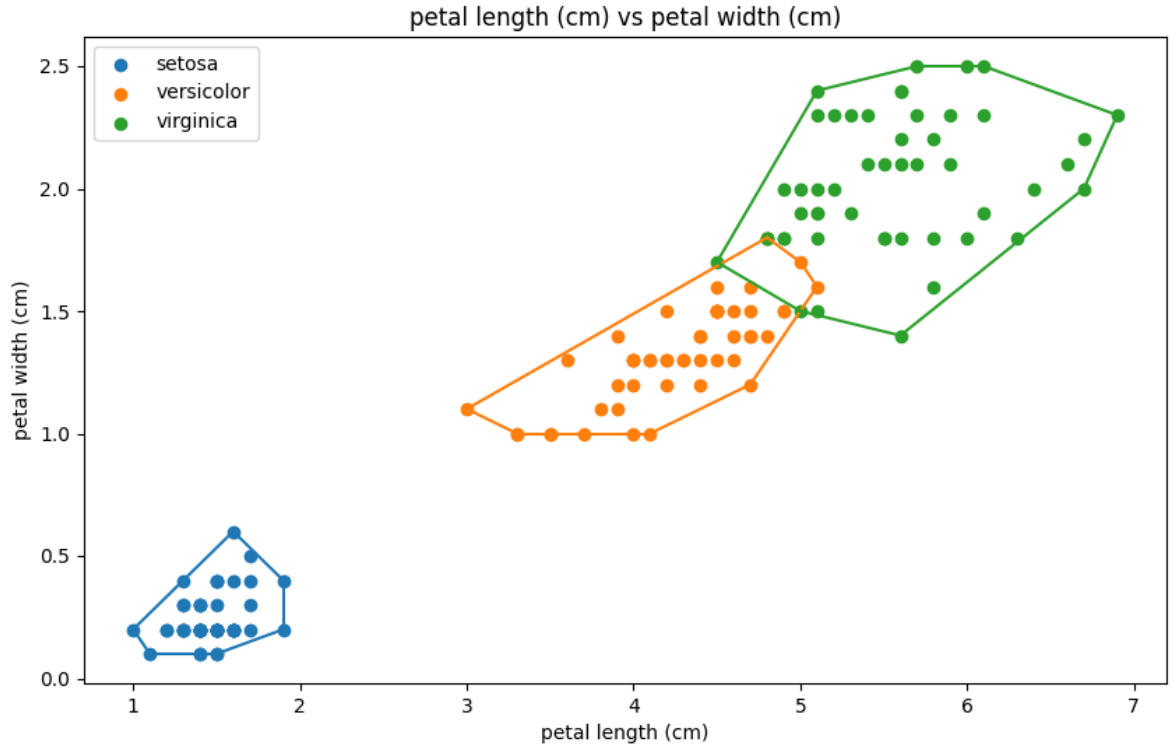
C. Screenshot

1. Dataset iris

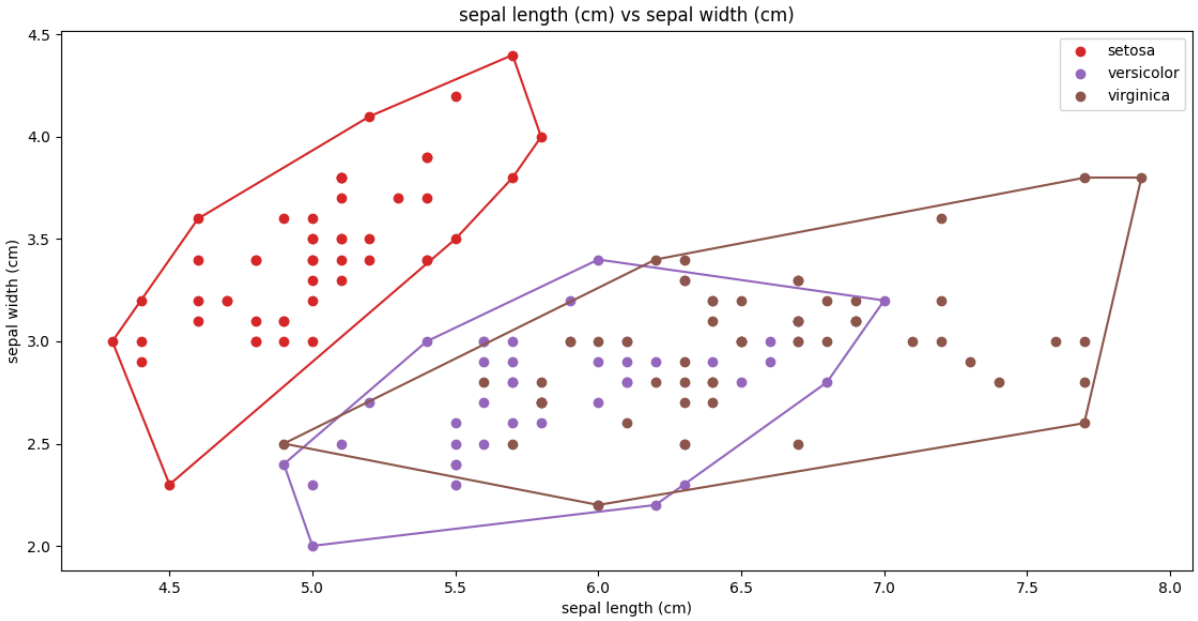
Dua pasang fitur ini dijalankan dengan perintah berikut:

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

a) petal length (cm) vs petal width (cm)



b) sepal length (cm) vs sepal width (cm)

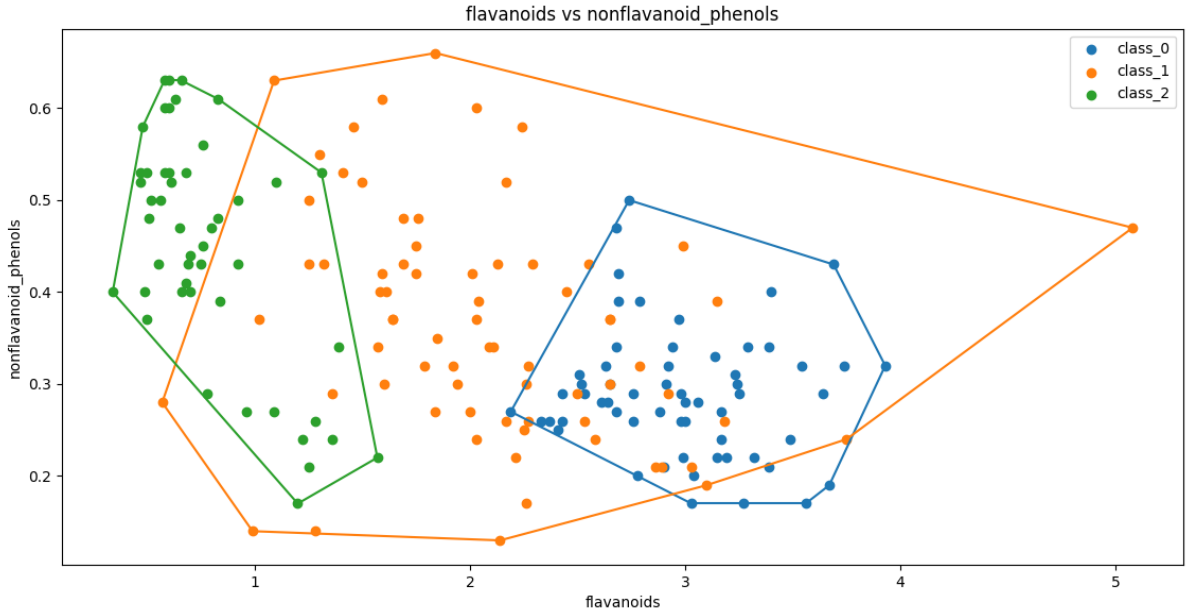


2. Dataset wine

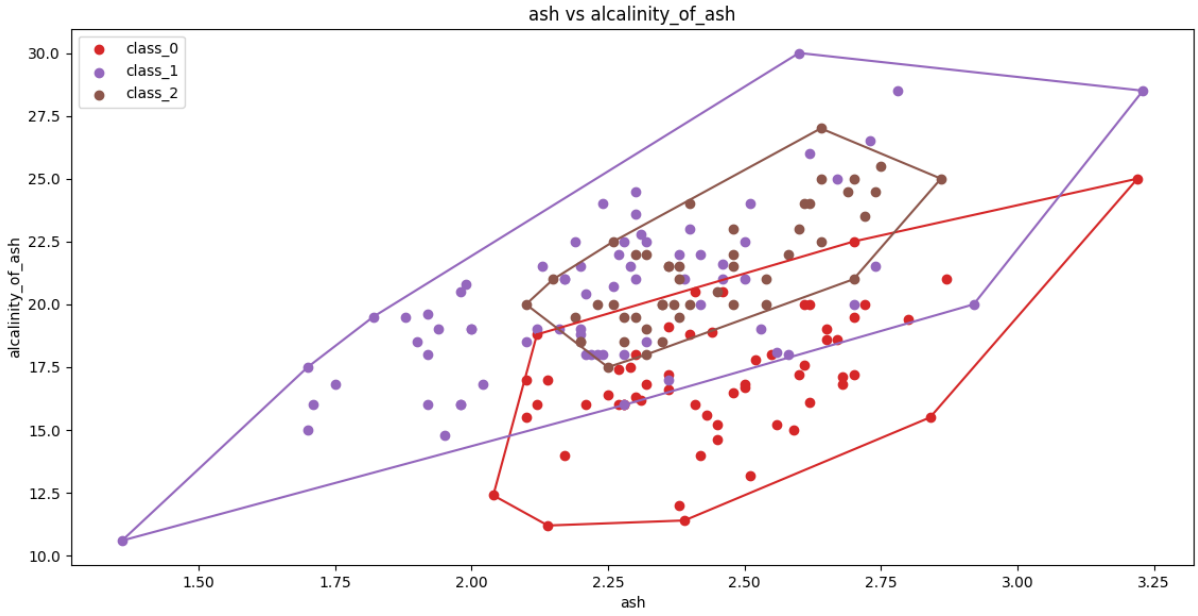
Dua pasang fitur ini dijalankan dengan perintah berikut:

```
python -m myConvexHull -n wine -fp "flavanoids" "nonflavanoid_phenols" -fp "ash"
"alcalinity_of_ash"
```

a) flavanoids vs nonflavanoid_phenols



b) ash vs alcalinity_of_ash

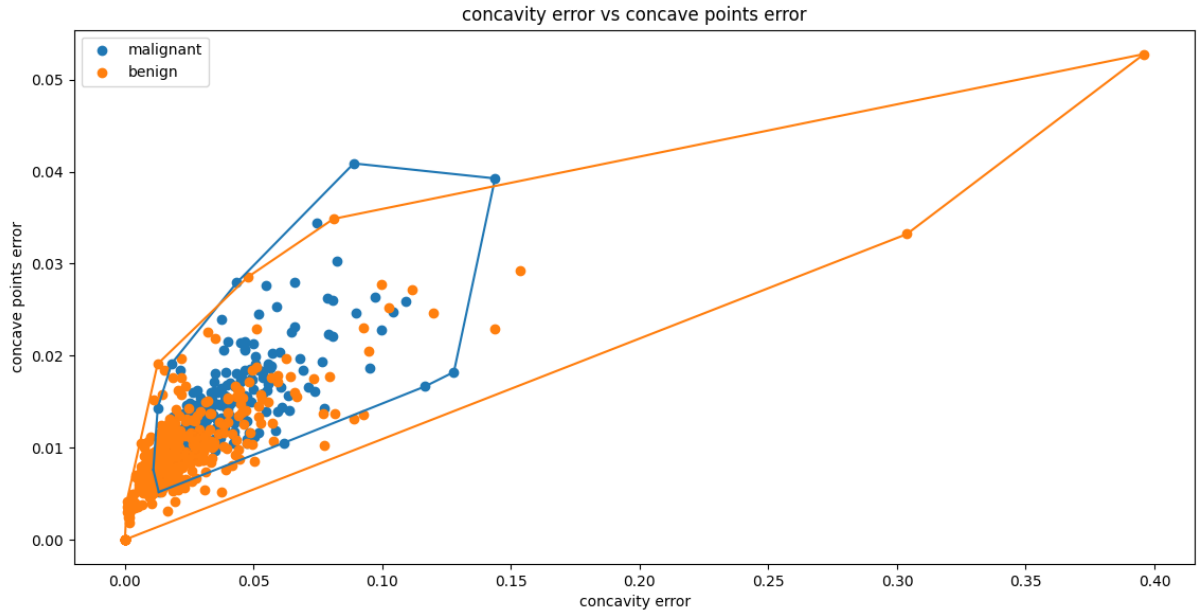


3. Dataset breast_cancer

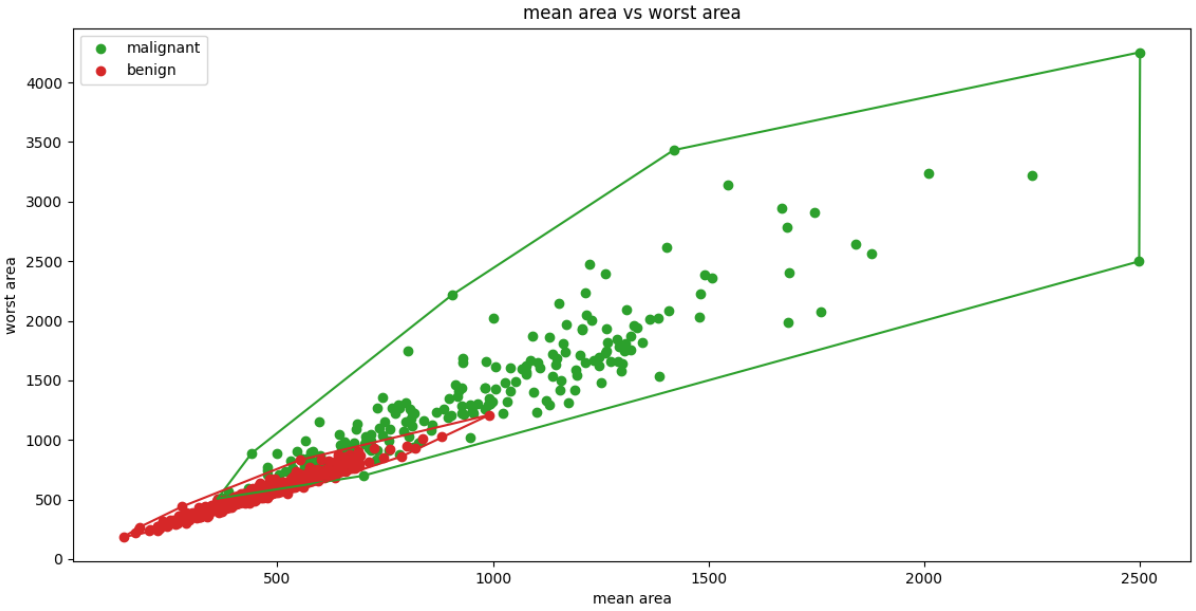
Dua pasang fitur ini dijalankan dengan perintah berikut:

```
python -m myConvexHull -n breast_cancer -fp "concavity error" "concave points error" -fp
"mean area" "worst area"
```

a) concavity error vs concave points error



b) mean area vs worst area

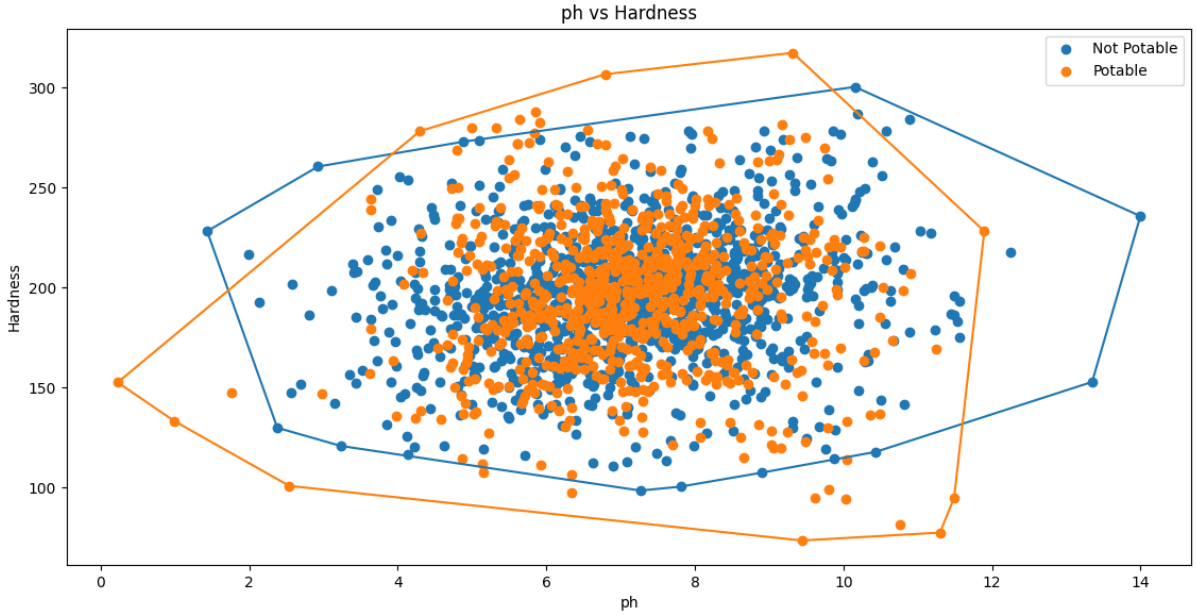


4. Dataset water potability

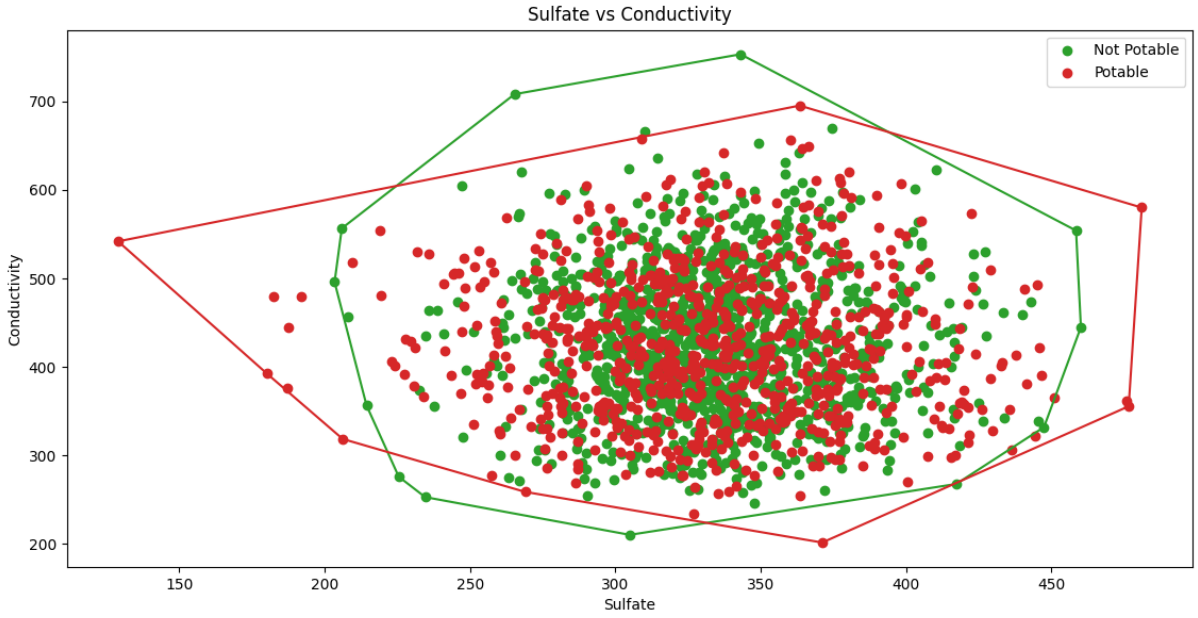
Dataset ini diambil dari open source database di kaggle yang dapat diunduh disini. Dua pasang fitur ini dijalankan dengan perintah berikut:

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

a) pH vs Hardness



b) Sulfate vs Conductivity



Lampiran

Assistant Checklist

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

Link Repository

https://github.com/marfgold1/LinearSeparabilityDataset