Reconocimiento Estadístico de Patrones - Tarea 1

Rubén Pérez Palacios Lic. Computación Matemática, Profesor: Johan Van Horebeek

March 8, 2023

1 Librerias Auxiliares

1.1 Operadores de comparación

Nos apoyamos de una clase para la impletación de los operadores de comparación, y de ella hederar con la finalidad facilitar su sobrecarga.

```
template < typename T >
class implement_relational_operators {
    // friend bool operator  < (const T& lhs, const T& rhs);
    // friend bool operator == (const T& lhs, const T& rhs);
    friend bool operator != (const T& lhs, const T& rhs) { return !operator==(lhs,rhs); }
    friend bool operator  > (const T& lhs, const T& rhs) { return  operator <(rhs,lhs); }
    friend bool operator <= (const T& lhs, const T& rhs) { return !operator >(lhs,rhs); }
    friend bool operator >= (const T& lhs, const T& rhs) { return !operator <(lhs,rhs); }
};
```

1.2 Operadores artiméticos (suma y resta)

Nos apoyamos de una clase para la impletación de los operadores de suma y resta, y de ella hederar con la finalidad facilitar su sobrecarga.

```
template < typename T >
class arithmetic_operators {
    // X& operator+=(const X& rhs)
    // {
    //  // actual addition of rhs to *this
    //  return *this;
    // }
    // X& operatoR-=(const X& rhs)
    // {
    //  // actual addition of rhs to *this
    //  return *this;
    // }
    friend T operator+(T lhs, const T& rhs)
    {
        lhs += rhs;
        return lhs;
    }
    friend T operator-(T lhs, const T& rhs)
```

```
    {
        lhs -= rhs;
        return lhs;
    }
};
```

1.3 Punto

Clase que implementa un punto en dos dimensiones, que de no especificarse sus puntos entonces se considera el origen.

```
template < typename T>
class CPoint
{
public:
    T x, y;

    CPoint < T >();

    CPoint < T >(T, T);
};
```

1.4 Vector

Se implemento una clase de un vector dos dimensional, para poder hacer uso de los operadores aritméticos y de comparación, así como poder leer y escribir directo en consola. Junto con operaciones útiles de vectores como producto punto, producto cruz, distancia al cuadrado,

```
template < typename T >
class CVector : public CPoint < T >, public relational_operators < CVector < T > >, public ari
{
public:

    CVector(T, T);

    CVector(CPoint < T >);

    CVector();

    CVector < T > & operator+=(const CVector < T >&);

    CVector < T > & operator-=(const CVector < T >&);

    template < typename S >
    friend inline bool operator  < (const CVector < S >&, const CVector < S >&);
    template < typename S >
    friend inline bool operator == (const CVector < S >&, const CVector < S >&);

    template < typename S >
```

```
    friend istream& operator >> (istream&, CVector<S>&);
    template < typename S >
    friend ostream& operator << (ostream&, const CVector<S>&);

};

template < typename  T >
T dot_product(const CVector<T>,const CVector<T>);

template < typename  T >
T cross_product(const CVector<T>, const CVector<T>);

template < typename  T >
T side(const CVector < T >, const CVector < T >, const CVector < T >);

template < typename  T >
T square_dist(const CVector < T >, const CVector < T >);

template < typename  T >
bool isbetween(const CVector < T >, const CVector < T >, const CVector < T >);

template <typename T>
T sign(T val);
```

1.5 Comparación por ángulo

Impletación de la comparación entre dos puntos `u,v` dada por el ángulo con respecto a un tercer punto fijo `origin`, el cuál debe ser indicado al tiempo de instancia. Tener cuidado puesto que el operador de comparación implementa un orden lexicográfico.

```
template < typename T >
class CAngle_Comparision
{
public:
    CAngle_Comparision(CVector < T > origin) { this->origin = origin; }
    bool operator () (const CVector < T > u, const CVector < T > v)
    {
        if (!side(u, v, origin))
            return square_dist(u, origin) < square_dist(v, origin);
        return (side(u, v, origin) > 0);
    }

    CVector < T > origin;
};
```

2 Convex Hull

2.1 Implementación

La impletación de los algoritmos se realizo en c++. Para ello se diseño una plantilla de clase abstracta `CConvex_Hull<T>` que englobe las campos y metodos comúnes entre las implementaciones, para que ellas solo modifiquen el como calcular los puntos en la envolvente convexa.

```
template < typename T >
class CConvex_Hull
{
    protected:

    vector < CVector < T > > CH;
    vector < CVector < T > > point;
    CVector < T > tmin_point, tmax_point, aux;

    public:

    bool inside(CVector < T >);

    CConvex_Hull();

    CConvex_Hull(vector < CPoint < T > >);

    virtual void calculate() = 0;

    void precalculate();

    template < typename S >
    friend istream& operator >> (istream&, CConvex_Hull<S>&);

    template < typename S >
    friend ostream& operator << (ostream&, const CConvex_Hull<S>&);

};
```

2.1.1 Graham Scan

Puesto que nuestra clase base ya cálculo nuestro punto `t_min` más a la izquierda –y más abajo– de nuestro conjunto de puntos, basta con ordenar nuestros puntos gradialmente con respecto a este con ello nos apoyaremos de nuestra clase de comparación `CAngle_Comparision`. Posteriormente solo seguiremos comparando punto por punto, evaluando si este está o no en el interior de nuestra actual envolvente convexa; de no ser así incluirlo en ella.

```
template < typename T >
class CGraham_Scan : public CConvex_Hull < T >
{
    private:
```

```
    public:

    void calculate()
    {
        sort(all(this->point), CAngle_Comparision<T>(this->tmin_point));

        for (T i = 0; i < this->point.size(); i++)
        {
            this->aux = (CVector < T >) (*(new CVector < T >(this->point[i])));
            while (this->CH.size() > 1 && side(this->CH.back(), this->aux, this->CH[this->CH.s
                this->CH.pop_back();
            this->CH.push_back(this->aux);
        }
    }

};
```

2.1.2 Quick Hull

Puesto que nuestra clase base ya cálculo nuestro punto `t_min` "más a la izquierda" –y más abajo– y nuestro punto `t_max` "más a la derecha" –y más arriba– de nuestro conjunto de puntos, entonces obtemos los puntos extremos de nuestra segmento de línea de partida (`tl_point,tr_point` respectivamente), para de ella recursivamente sobre cada lado encontrar la envolvente convexa de su respectiva mitad y así tomar la unión. En cada paso de la recursión se obtiene el punto más distante del respectivo lado a la línea con distancia `tmax_dist` e índice `ind` de nustros puntos, posteriormente recursivamente encontramos la envolvente convexa de los puntos de nuestro conjunto que quedan fuera de de nuestro triángulo `tl_point,point_ind,tr_point` para ello encontramos la envolvente convexa de los puntos contenidos en el semiplano contrario al del triángulo con respecto a los segmentos de línea `tl_point,point_ind` y `point_ind,tr_pointy` devolver la unión.

```
template < typename T >
class CQuick_Hull : public CConvex_Hull < T >
{
private:

    set < CVector < T > > bucket;

public:

    void calculate()
    {
        quick_hull(this->tmin_point, this->tmax_point, 1);
        quick_hull(this->tmin_point, this->tmax_point, -1);

        this->CH.reserve(bucket.size ());
        copy(bucket.begin (), bucket.end (), back_inserter (this->CH));
    }
```

```
    void quick_hull(CVector < T > tl_point, CVector < T > tr_point, int orient)
    {
        int ind = -1, tmax_dist = 0;

        for (int i=0; i<this->point.size(); i++)
        {
            if (side(tl_point, tr_point, this->point[i]) != orient)
                continue;
            int dist = abs(cross_product(tr_point-tl_point, this->point[i]-tl_point));
            if (dist > tmax_dist)
            {
                ind = i;
                tmax_dist = dist;
            }
        }

        if (ind == -1)
        {
            bucket.insert(tl_point);
            bucket.insert(tr_point);
            return;
        }

        quick_hull(this->point[ind], tl_point, -side(this->point[ind], tl_point, tr_point));
        quick_hull(this->point[ind], tr_point, -side(this->point[ind], tmax_point, tl_point));
    }
};
```

3 Ejemplos

3.1 Funciones útiles

Implementaremos una función para crear un archivo con nuestro conjunto de puntos y que llame al programa `Convex_Hull.exe` para que este cálcule su respectiva envolvente convexa con las implementaciones anteriormente mencionadas. Así como otra función para graficar dichos resultados.

[ ]:
```
import os
from random import random
import numpy as np
from tqdm import tqdm
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
import plotly.io as pio
pio.renderers.default = 'pdf'

sample_size = 5
```

```python
def calculate(exe_path, points_path, convexhull_path, points):
    file = open(points_path,"w+")
    for p in points:
        for xi in p:
            file.write(f"{xi} ")
        file.write("\n")
    file.close()
    os.system(exe_path + " " + points_path + " " + convexhull_path)

def graph(points_path, convexhull_path, title):
    points = np.fromfile(points_path,sep=" ").reshape((-1,2))
    fig = go.Figure()
    fig.add_scatter(
        x = points[:,0],
        y = points[:,1],
        name="Puntos",
        mode="markers"
    )

    ch = np.fromfile(convexhull_path, sep=" ").reshape((-1,2))
    fig.add_trace(
        go.Scatter(
            x = np.append(ch[:,0],ch[0,0]),
            y = np.append(ch[:,1],ch[0,1]),
            name="Envolvente Convexa",
            fill="toself"
        )
    )

    fig.update_layout(
        template="simple_white",
        title=title,
        width=1000,
        height=1000
    )

    fig.update_yaxes(
        scaleanchor="x",
        scaleratio=1,
    )

    fig.show()
```

3.2 Primer ejemplo

[ ]:
```
exe_path = "..\\Executables\\Convex_Hull.exe"
```

[ ]:
```
title = "House"
points_path = f"..\\Inputs\\{title}.in"
convexhull_path = f"..\\Outputs\\{title}.out"
points = [[0,0],[0,2],[1,3],[1,2],[1,1],[1,0],[2,0],[2,2]]
calculate(exe_path,points_path,convexhull_path,points)
graph(points_path,convexhull_path+".gs",title)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[122], line 6
      4 points = [[0,0],[0,2],[1,3],[1,2],[1,1],[1,0],[2,0],[2,2]]
      5 calculate(exe_path,points_path,convexhull_path,points)
----> 6 graph(points_path,convexhull_path+".gs",title)

Cell In[120], line 54, in graph(points_path, convexhull_path, title)
     42 fig.update_layout(
     43     template="simple_white",
     44     title=title,
     45     width=1000,
     46     height=1000
     47 )
     49 fig.update_yaxes(
     50     scaleanchor="x",
     51     scaleratio=1,
     52 )
---> 54 fig.show()

File ~\AppData\Roaming\Python\Python39\site-packages\plotly\basedatatypes.py:
 ↪3390, in BaseFigure.show(self, *args, **kwargs)
   3357 """
   3358 Show a figure using either the default renderer(s) or the renderer(s)
   3359 specified by the renderer argument
   (…)
   3386 None
   3387 """
   3388 import plotly.io as pio
-> 3390 return pio.show(self, *args, **kwargs)

File ~\AppData\Roaming\Python\Python39\site-packages\plotly\io\_renderers.py:
 ↪388, in show(fig, renderer, validate, **kwargs)
    385 fig_dict = validate_coerce_fig_to_dict(fig, validate)
    387 # Mimetype renderers
--> 388 bundle = renderers._build_mime_bundle(fig_dict,␣
 ↪renderers_string=renderer, **kwargs)
```

```
    389 if bundle:
    390     if not ipython_display:

File ~\AppData\Roaming\Python\Python39\site-packages\plotly\io\_renderers.py:
 ↪296, in RenderersConfig._build_mime_bundle(self, fig_dict, renderers_string,
 ↪**kwargs)
    293             if hasattr(renderer, k):
    294                 setattr(renderer, k, v)
--> 296         bundle.update(renderer.to_mimebundle(fig_dict))
    298 return bundle

File ~\AppData\Roaming\Python\Python39\site-packages\plotly\io\_base_renderers.
 ↪py:127, in ImageRenderer.to_mimebundle(self, fig_dict)
    126 def to_mimebundle(self, fig_dict):
--> 127     image_bytes = to_image(
    128         fig_dict,
    129         format=self.format,
    130         width=self.width,
    131         height=self.height,
    132         scale=self.scale,
    133         validate=False,
    134         engine=self.engine,
    135     )
    137     if self.b64_encode:
    138         image_str = base64.b64encode(image_bytes).decode("utf8")

File ~\AppData\Roaming\Python\Python39\site-packages\plotly\io\_kaleido.py:133,
 ↪in to_image(fig, format, width, height, scale, validate, engine)
    131     # Raise informative error message if Kaleido is not installed
    132     if scope is None:
--> 133         raise ValueError(
    134             """
    135 Image export using the "kaleido" engine requires the kaleido package,
    136 which can be installed using pip:
    137     $ pip install -U kaleido
    138 """
    139         )
    141     # Validate figure
    142     # ---------------
    143     fig_dict = validate_coerce_fig_to_dict(fig, validate)

ValueError:
Image export using the "kaleido" engine requires the kaleido package,
which can be installed using pip:
    $ pip install -U kaleido
```

[ ]:
```
title = "Típico"
points_path = f"..\\Inputs\\{title}.in"
convexhull_path = f"..\\Outputs\\{title}.out"
points = [[30, 60],[50, 40],[70, 30],[55, 20],[50, 10],[20, 0],[15, 25],[0, 30]]
calculate(exe_path,points_path,convexhull_path,points)
graph(points_path,convexhull_path+".gs",title)
```

[ ]:
```
title = "Complejo"
points_path = f"..\\Inputs\\{title}.in"
convexhull_path = f"..\\Outputs\\{title}.out"
points = [[-7,8], [-4,6], [2,6], [6,4], [8,6], [7,-2], [4,-6], [8,-7],[0,0],␣
 ↪[3,-2],[6,-10],[0,-6],[-9,-5],[-8,-2],[-8,0],[-10,3],[-2,2],[-10,4]]
calculate(exe_path,points_path,convexhull_path,points)
graph(points_path,convexhull_path+".gs",title)
```

[ ]:
```
title = "Gaussiano"
points_path = f"..\\Inputs\\{title}.in"
convexhull_path = f"..\\Outputs\\{title}.out"
points = np.random.standard_normal((2,100))
calculate(exe_path,points_path,convexhull_path,points)
graph(points_path,convexhull_path+".gs",title)
```