Geometria Computacional 2020.2

Aplicação de conceitos sobre Fecho Convexo 2D ao problema de detecção de *outliers*

Aluno: Jose Luis Huillca Mango

Podemos definir um *outlier* como uma observação que está a uma distância anormal de outros valores em uma amostra aleatória de uma população (NIST, 2020).

Para este trabalho, tomaremos como base o artigo (Harsh, A. et al., 2018), que propõe uma variação do método *Onion Peeling* para detecção de *outliers*. Apresentaremos e implementamos um algoritmo de *Onion Peeling* modificado para detectar os *k-outliers* principais, em dados bidimensionais, provenientes de uma distribuição Gaussiana e de um banco de dados de reconhecimento de vinhos[1].

Requisitos para executar o algoritmo

O algoritmo foi implementado em Python 3, as bibliotecas utilizadas são:

1. scikit-geometry
2. scipy
3. scikit-learn
4. pandas
5. jupyter

O trabalho também está em docker[2], para isso seria pré-requisito ter docker instalado e seguir os seguintes comandos (O trabalho também pode ser baixado do GitHub):

```
$ git clone https://github.com/josehuillca/GeoComp_trabalho1.git
$ cd docker/
$ docker build -t geocomp:latest .
$ cd ..
$ docker run --rm -it -p 8080:8080 -v $PWD:/workspace geocomp:latest
bash
$ source activate geocomp
$ jupyter notebook --allow-root --ip=0.0.0.0 --port 8080
```

Um exemplo do uso do algoritmo está no arquivo notebook/OnionPeeling.ipynb .

Algoritmo Onion Peeling Outlier Detection

O método (`onionPeelingOutlierDetection`) implementado recebe como entrada um conjunto *S* de pontos 2D, um valor *k* que indica o número de outliers a serem detectados e produz como saída uma lista *L* com os k outliers mais significativos.

---

[1] https://archive.ics.uci.edu/ml/datasets/Wine
[2] https://www.docker.com/

```
import sys
import numpy as np
sys.path.append('../')
from mycode.onionPeeling import OnionPeeling
from mycode.utils import makePoints, drawPoints, drawHull
```

```
def onionPeelingOutlierDetection(S, k, draw=True, cor='green'):
    A = OnionPeeling(S)
    L, hull = A.outlierDetection(k, distance_metric='mahalanobis',
convex_hull='gift-wrapping')
    if draw:
        drawPoints(S, cor='skyblue')
        drawHull(hull, cor=cor)
        drawPoints(L, cor='red')
    return L
```

A função principal (`outlierDetection`) está dentro da classe `OnionPeeling` , a ideia fundamental é que o maior *outlier* no conjunto de dados estará potencialmente no primeiro peel, em outras palavras, um potencial *outlier k* é aquele que tem a distância máxima da média do conjunto de dados. O algoritmo calcula os *outliers* com base na métrica de distância, ou seja, distância euclidiana e distância de Mahalanobis[3] (o parâmetro `distance_metric` pode tomar os valores de *'euclidean'* ou *'mahalanobis'*).

A saída (*L*) do algoritmo é uma lista dos pontos (*x, y*) que representam os outliers.

O algoritmo `outlierDetection` funciona em duas fases:

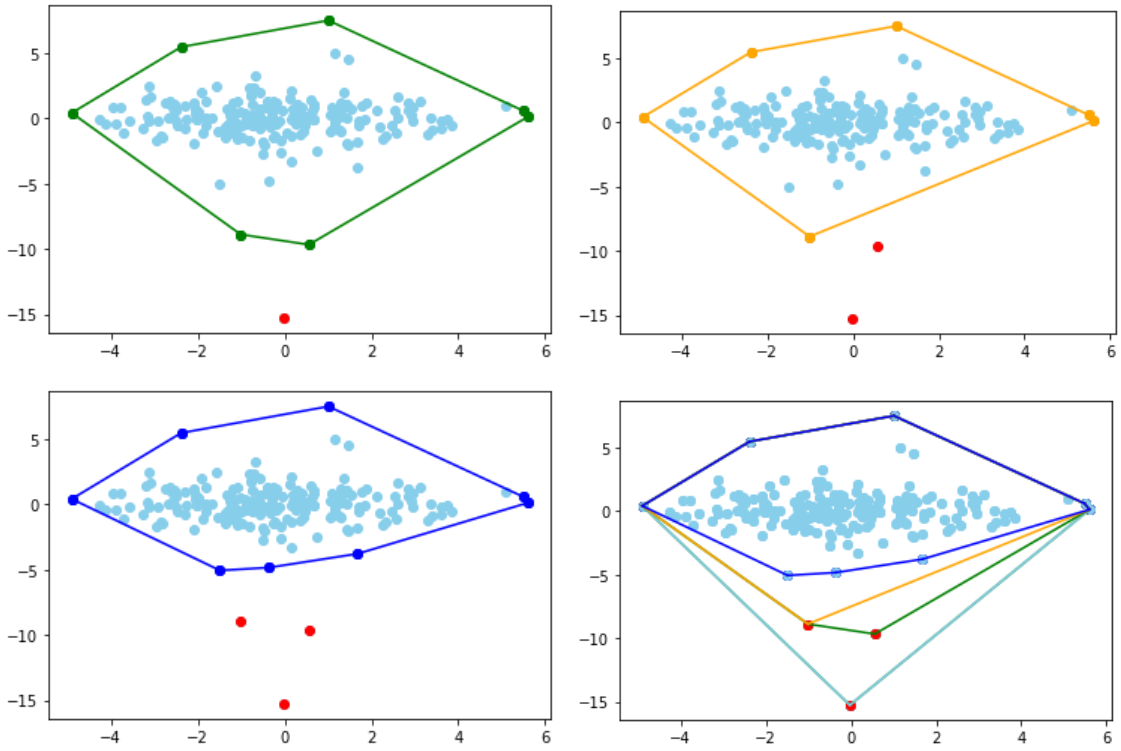
1. Encontre o fecho convexo usando o algoritmo de *Onion Peeling,*
2. Com base no fecho convexo e conjunto de dados, calcule o número de outliers com a distância máxima do centro dos dados.

Experimentos

Dados Sintéticos

Para o primeiro experimento, usaremos dados sintéticos 2D de uma distribuição Gaussiana. O número de amostras usadas foi *n = 200* pontos aleatórios, e o número de outliers a serem detectados é *k = 3*.
Na Figura~1, imagem superior esquerda, observa-se o resultado obtendo apenas um *outlier (k = 1)* e o *hull* é pintado de verde; a imagem superior direita mostra o resultado obtendo dois *outlier (k = 2)* e o *hull* é pintado de laranja; a imagem inferior esquerda obtendo três *outlier (k = 3)* e o *hull* é pintado de cor azul; e, finalmente, a imagem inferior direita mostra todas os fecho convexos geradas, incluindo o fecho convexo quando *k = 0,* representada em azul claro. Os pontos vermelhos representam os *outliers*.

[3] https://www.machinelearningplus.com/statistics/mahalanobis-distance/



Figura~1: n=200 dados sintéticos (puntos de cor azul claro). Os pontos vermelhos representam os *outliers*.

Dataset Wine

Esses dados são resultados de uma análise química de vinhos cultivados na mesma região da Itália, mas derivados de três cultivares diferentes. A análise determinou as quantidades de 13 constituintes encontrados em cada um dos três tipos de vinhos. Os atributos são:
1) Álcool
2) Ácido málico
3) Cinza
4) Alcalinidade das cinzas
5) Magnésio
6) Fenóis totais
7) Flavanóides
8) Fenóis não flavanoides
9) Proantocianinas
10) Intensidade da cor
11) Matiz
12) OD280 / OD315 de vinhos diluídos
13) Proline

Para este trabalho, usaremos apenas dois atributos: 1) Álcool e 2) Ácido málico. O conjunto de dados é carregado usando a biblioteca Pandas[4] do Python.

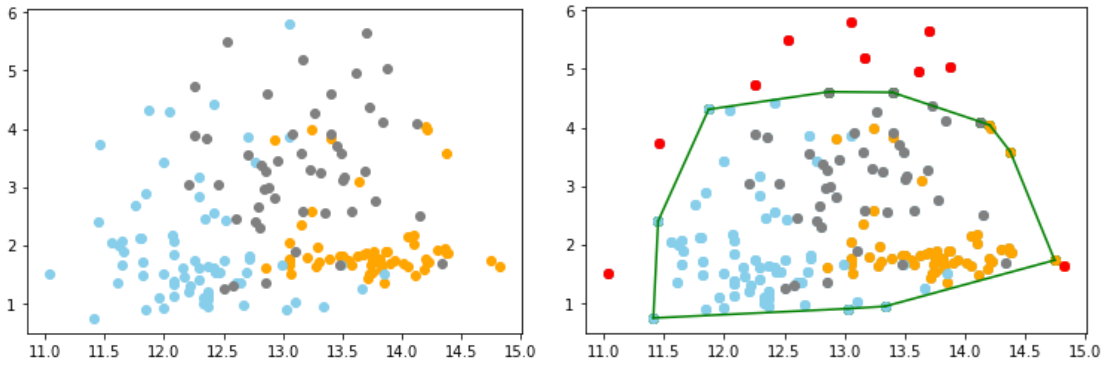
```
import pandas as pd
```

[4] https://pypi.org/project/pandas/

```
names = ['label','Alcohol', 'Malic acid']
df = pd.read_csv('../dataset/wine/wine.data', usecols=[0,1,2],
names=names, header=None)
display(df.head(5)) # mostramos os cinco primeiros dados
```

	label	Alcohol	Malic acid
0	1	14.23	1.71
1	1	13.20	1.78
2	1	13.16	2.36
3	1	14.37	1.95
4	1	13.24	2.59

No total, temos *n = 178* pontos, onde cada coordenada de um ponto é formada como *x = 'Alcohol', e y = 'Malic Acid*". Como mencionado acima, existem três tipos de vinho (*label = [1,2,3]*). Estes tipos de vinhos são representados pelas cores laranja, azul claro e cinza, respectivamente (ver Figura~2, imagem à esquerda).



Figura~2: Dataset-Wine com n=178 pontos. Na imagem esquerda cada cor representa um tipo de vinho. Na imagem direita, os pontos vermelhos são os *outliers (k=10)* e o fecho convexo é representado de cor verde.

Comparação com os métodos MCD (Minimum Covariance Determinant) e OCSVM (One-class SVM)

Para comparar os métodos MCD e OCMSVD com o algoritmo realizado (One Peeling Outlier Detection), usaremos o banco de dados de vinhos, mostrado acima.
Na Figura~3 podemos ver a detecção dos outliers para cada método, MDC[5], OCSVM[6] e One Peeling Outlier Detection nas cores verde, azul e vermelho, respectivamente.

---

5 https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html#sklearn.covariance.EllipticEnvelope

6 https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

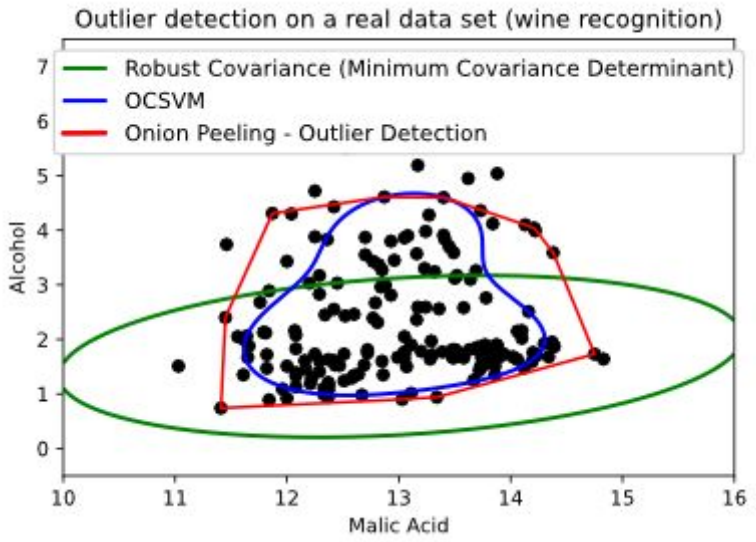
Neste exemplo, no algoritmo One Peeling Outlier Detection *k = 10 outliers* foram usados, e para os outros algoritmos os parâmetros foram os seguintes:

```
'MCD': EllipticEnvelope(contamination=0.25)
```

- **Contamination**: A quantidade de contaminação do conjunto de dados, ou seja, a proporção de outliers no conjunto de dados. O intervalo é (0, 0.5).

```
"OCSVM": OneClassSVM(nu=0.25, gamma=0.35)
```

- **nu:** Limite superior da fração de erros de treinamento e um limite inferior da fração de vetores de suporte. Deve estar no intervalo (0, 1]. Por padrão, 0.5 será usado.
- **gamma:** O parâmetro Gamma determina a influência do raio no kernel. O intervalo deste parâmetro depende de seus dados e aplicativo.



Figura~3: Detecção de outliers em um conjunto de dados real (reconhecimento de vinho).

Pontos fortes e limitações da abordagem implementada em comparação com os métodos MCD e OCSVM:

	MCD	OCSVM	One Peeling Outlier Detection
Pontos fortes	-Detecção automática de outliers	-Detecção automática de outliers	- Fácil de implementar e entender.
Limitações	- Problemas de estimar a covariância de acordo com a forma de distribuição dos dados.	- Dificuldade em ajustar o parâmetro de largura de banda do kernel (gama) para ter um bom comportamento na forma de dispersão de dados	- A detecção de outliers não é automática, o usuário não precisa passar o parâmetro de outliers para serem detectados

Referências

1. NIST, .. (2020). *What are outliers in the data?* http://www.itl.nist.gov/div898/handbook/prc/section1/prc16.htm
2. Harsh, A., Ball, J., & Wei, P. (2018). Onion-Peeling Outlier Detection in 2-D data Sets. *ArXiv, abs/1803.04964*.