Python OAM: apresentação e uso de uma biblioteca de explicabilidade para processos de detecção de outliers

Rodrigo F. Silva[1], **Luiz Gomes-Jr**[1]

[1]DAINF - Universidade Tecnológica Federal do Paraná (UTFPR)

rsilva.1998@alunos.utfpr.edu.br, lcjunior@utfpr.edu.br

Resumo. *Detecção de outliers é usada para identificar falhas e fraudes, entre outras aplicações. Algoritmos de detecção são limitados em termos de prover informação sobre a razão da anomalia identificada. Outlier Aspect Mining (OAM) analisa quais aspectos da observação anômala a separam das demais. Este artigo descreve a implementação e uso de uma biblioteca em Python que possibilita o usuário aplicar algoritmos de OAM e analisar os resultados. Demonstramos a aplicação da biblioteca em um caso de uso relacionado à pandemia de COVID-19.*

Abstract. *Outlier detection is used to identify system failures and fraud, among other applications. Detection algorithms are, however, limited in terms of providing information about the reason for the anomaly. Outlier Aspect Mining (OAM) assesses which aspects of the anomalous observation separate it from others. This article describes the implementation and use of a Python library that allows the user to apply OAM algorithms and analyze the results. We demonstrate the application of the library in a use case related to the COVID-19 pandemic.*

1. Introdução

A detecção de *outliers* se refere ao conjunto de soluções propostas para resolver o problema de identificar observações que parecem ser inconsistentes com o restante do conjunto de dados. Existem diversas técnicas de detecção, que são aplicadas em uma vasta variedade de domínios, como fraudes de cartão de crédito, detecção de falhas em equipamentos, entre outros [Samariya et al. 2020a, Chandola et al. 2009]. Apesar da relevância dos algoritmos de detecção, a vasta maioria das técnicas utilizadas com esse intuito não é capaz de responder por que as observações dadas como *outliers* foram consideradas discrepantes.

Para resolver esse problema, uma nova área de pesquisa tem surgido nos últimos anos: a explicabilidade de *outliers* ou *Outlier Aspect Mining* (OAM). Ela pode ser formalmente definida como a tarefa de reconhecer os subconjuntos de dimensões onde um determinado objeto é inconsistente com o restante. Assim como *Outlier Detection* (OD), o OAM também tem muitas aplicações práticas, por exemplo, identificar em quais aspectos específicos um candidato a emprego se difere dos demais, quais equipamentos/sensores estão mais associados a uma falha industrial, entre outros.

Por se tratar de uma área relativamente nova de pesquisa, ainda é difícil encontrar implementações disponíveis para livre utilização. Tendo isso em vista, o objetivo desse trabalho é apresentar a biblioteca *python-oam*[1], desenvolvida com esse intuito. Ela possi-

[1]Vídeo demonstração: https://youtu.be/ltJDCWe6_S0

bilita o usuário aplicar algoritmos de OAM e analisar os resultados em sua própria massa de dados. O código fonte foi disponibilizado para a comunidade não apenas utilizá-lo mas também estendê-lo como desejar. Foi implementado o algoritmo *iPath*, e algumas funcionalidades complementares, como funções de preprocessamento dos dados, comuns em análises de *outliers*, e métodos de visualização que auxiliam a utilização da biblioteca e a tomada de decisão por parte do usuário final.

2. Fundamentos

As técnicas de OAM existentes podem ser classificadas em três grandes grupos [Samariya et al. 2020b]: *Score and Search*, *Feature Selection* e *Hybrid Approach*. A implementação inicial da biblioteca focou na categoria de algoritmos *Score and Search*, pois dela derivou a maior quantidade de estudos e resultados publicados.

2.1. Score and Search

Também conhecido como abordagem de pontuação e pesquisa, a abordagem de *score and search* exige a combinação de duas funções. A de pontuação, *score*, para medir o quanto o objeto destoa do restante do conjunto em uma dimensão específica, e a função de *search*, ou busca, que definirá quais subespaços serão avaliados e comparados entre si. O *search* define os subespaços a serem avaliados pelo *score*, reúne essa informação e, usualmente, retorna uma lista relacionando cada subespaço com o valor referente a quanto uma observação é *outlier* nele. Esta abordagem oferece a possibilidade de combinar algoritmos diferentes de *score* e *search*, a depender do propósito do usuário. Geralmente não existe dependência entre eles.

Dentre as possibilidades de técnicas de busca, ou *search*, apresentadas pela comunidade nos últimos anos, escolhemos utilizar o *Simple Combination* para ser implementada. O algoritmo consiste em realizar todas as combinações de espaços possíveis entre um tamanho mínimo e máximo. Por exemplo, para um *dataset* de *n* dimensões, o *Simple Combination* criará todas as combinações possíveis sem repetição e avaliará cada uma delas. A implementação de *score* escolhida foi o *iPath*.

iPath: O algoritmo *iPath* (*Isolation Path*)[the Vinh et al. 2016] é derivado da abordagem de detecção de anomalias *iForest* (*isolation Forest*). O *iPath* se baseia na ideia de que um um *outlier* geralmente se encontra em uma região de baixa densidade.

O processo do *iPath* consiste em realizar sucessivos cortes no espaço, até que um objeto seja isolado do restante. Nesse cenário, se o objeto estiver cercado por vários outros, serão necessários mais cortes para separá-lo do restante, enquanto um *outlier* exigirá menos cortes.

3. A biblioteca

Inicialmente o usuário deve entender se os dados utilizados necessitam de tratamento. Caso precisem, poderão ser utilizados os recursos de pre-processamento disponíveis na própria biblioteca. O próximo passo deve envolver a detecção de *outliers* para que sejam identificadas as observações a serem analisadas. Qualquer técnica de OD pode ser utilizada. Com os *outliers* em mãos, a análise OAM poderá ser iniciada.

Nesse momento, o usuário deve escolher quais observações deseja analisar e qual combinação de *score* e *search* será aplicada. Com os resultados dos algoritmos obtidos,

é possível utilizar as ferramentas de visualização disponibilizadas para analisar graficamente o retorno.

3.1. Estrutura da biblioteca

Os módulos disponíveis possibilitam a realização de um ciclo completo de análise OAM. Os módulos são listados abaixo e detalhados na sequência. Exemplos práticos de aplicação são descritos na Seção 4.

- *Preprocess*: contém uma função de normalização que permite que o usuário atribua pesos para dimensões escolhidas;
- *Score*: contém a classe *iPath* e conterá outras classes de *score* futuramente;
- *Search*: contém a classe *SimpleCombination* e conterá outras classes de *search* futuramente;
- *Visualization*: contém algumas funções para auxiliar a visualização, tanto antes quanto depois da aplicação das técnicas de OAM.

Preprocess: Nesse módulo está implementada uma função de normalização *min/-max*. Variáveis em escalas diferentes podem não contribuir igualmente para o ajuste do modelo, e assim acabar criando vieses. A função resolve esse problema ajustando todas as colunas à mesma escala.

```
1 from oam.preprocess import normalize
2 dataframe = normalize(dataframe, min_value, max_value)
```

Os parâmetros representam respectivamente o *dataframe* de entrada, o valor mínimo e máximo a serem utilizados na normalização. Todas as colunas do *dataframe* fornecido para a função devem ser de valor numérico, caso contrário, uma exceção será lançada. Caso o destaque de uma coluna específica seja desejado, esta função também permite que o usuário passe um dicionário atribuindo pesos para dimensões específicas. As dimensões escolhidas terão seu valor ajustado de acordo com o peso.

Score* and *Search: Apesar da implementação separá-los em módulos diferentes, o *Score* e *Search* funcionam como um conjunto. A criação de uma instância do *iPath* necessita de apenas dois parâmetros: o tamanho das subamostras geradas e a quantidade de árvores a serem geradas.

Em seguida, o *SimpleCombination*, implementado como método de *search*, recebe a instância do método de *score* escolhido, os parâmetros que definem o tamanho dos subespaços gerados e quais dimensões do *dataframe* serão utilizadas. Com as instâncias criadas, deve ser realizada uma chamada para o método *search* do *SimpleCombination* passando o *dataset* e o *index* da observação a ser analisada.

```
1  from oam.score.isolation_path import IsolationPath
2  from oam.search.simple_combination import SimpleCombination
3
4  ipath = IsolationPath(subsample_size=50, number_of_paths=200)
5
6  ipath_simple_combination = SimpleCombination(
7      ipath,
8      min_items_per_subspace=2,
9      max_items_per_subspace=4,
10     dimensions=["variation_mean", "variation_std"]
```

```
)

score = ipath_simple_combination.search(dataframe, query_index=41)
```

A decisão de desacoplar o *Score* do *Search* está ligada à possibilidade de combiná-los de diferentes formas. Caso um novo algoritmo de *score* seja implementado, ele precisará apenas conter um método chamado *score*, retornando um número, para se integrar com o *SimpleCombination*. Da mesma forma, caso um novo método de *search* seja implementado, ele precisará apenas avaliar a dimensão através do método *score* da classe recebida como parâmetro para se integrar ao o restante.

Visualization: No módulo de visualizações temos dois recursos implementados. Um deles, pensado para auxiliar uma visualização prévia à aplicação do OAM, calcula o *z-score*[2] de cada coluna e retorna um *heatmap* com os valores de *z-score* correspondentes. O outro recurso auxilia na compreensão dos resultados do OAM, exibindo para o usuário de forma visual o conjunto de subespaços e o relaciona ao seu *score*.

4. Caso de aplicação

A biblioteca foi avaliada em caráter prático em diferentes contextos e conjuntos de dados. Um deles, detalhado abaixo, está relacionado a dados municipais brasileiros durante a pandemia de *COVID-19* (2020-2021). O conjunto traz informações diárias sobre a evolução de casos e mortes[3], de medidas públicas de combate à pandemia[4], e de mobilidade da população[5] durante a primeira e segunda onda de contágio pelo *SARS-CoV-2*. Os dados foram agregados (média) por cidade e onda. O dataset conta com 55 municípios brasileiros (capitais e segunda maior cidade de cada estado).

Resultado Outlier Detection: Segundo o algoritmo usado, *LOF (Local Outlier Factor)* [Cheng et al. 2019], Cuiabá, Porto Alegre e Laranjal do Jari foram as cidades que mais se destoaram das demais. A partir disto, busca-se entender quais características as diferenciaram do restante do conjunto. Com este propósito se iniciou o uso da biblioteca *python-oam*.

Análise Exploratória: Como mostrado, o pacote oferece não apenas a implementação de algoritmos de *score and search*, mas também ferramentas de pré-processamento e visualização de dados. A análise se iniciou com a utilização do *z-score heatmap* (Figura 1, esquerda), que avalia o quanto um valor específico se distancia da média da dimensão. Quanto maior o desvio, maior a chance de aquele espaço compor a razão pela qual aquela observação foi considerada uma anomalia. Esta técnica pode ser muito útil em análises exploratórias unidimensionais de OAM.

Por exemplo, Cuiabá se destacou das demais cidades pelos altos números de contágio e morte (*z-score* acima de 2 sigma), sobretudo com um índice de políticas públicas de combate abaixo da média (Figura 1, esquerda). Ao analisar Porto Alegre, pode-se observar como a variável *%_população_idosa* apresenta um desvio positivo em

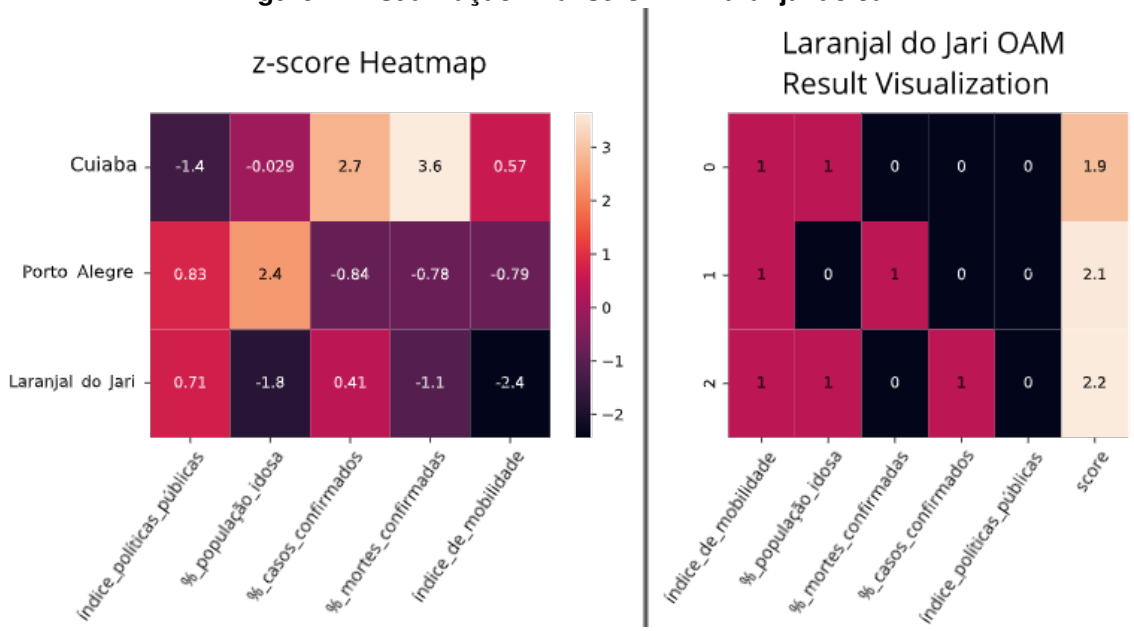
[2]Utilizado para medir quanto um valor dentro de um conjunto está distante da média em desvios padrão ($Z = (X - \mu)/\sigma$, onde X é a observação, μ é a média do conjunto e σ o desvio padrão)

[3]https://brasil.io/covid19/

[4]https://github.com/OxCGRT/Brazil-covid-policy

[5]https://www.google.com/covid19/mobility/

Figure 1. Visualização Análise OAM - Laranjal do Jari



relação à media. Já a variável *%_casos_confirmados* mostra um desvio negativo. Mais um caso de uma cidade com grande proporção de idosos e número de mortes abaixo da média nacional.

Análise OAM: Apesar da utilidade do *z-score heatmap*, como o conjunto trabalhado é multidimensional, analisar a influencia de *features* individuais no coeficiente de *outlier* pode não ser suficiente. Para compreender como conjuntos de dimensões específicos se comportam em cada uma das cidades mais destoantes, utilizou-se da técnica *score and search*. Assim, foram instanciadas tanto uma classe do *iPath* quanto do algoritmo *Simple Combination*.

Avaliaram-se todos os subespaços, com tamanho menor ou igual a três dimensões, compostos pela combinação simples das colunas do conjunto. Os resultados revelam informações importantes sobre as politicas de combate à pandemia adotadas, e como elas refletiram nos indicadores de contágio e morte de cada cidade. O gráfico à direita da Figura 1 apresenta o resultado do algoritmo de OAM para a cidade Laranjal do Jari, Amapá. Cada linha representa um subespaço diferente composto pelas variáveis (colunas) com valor 1. A coluna de *score* representa o retorno do *iPath*. Por exemplo, a primeira linha revela que o subespaço composto pelo *indice_de_mobilidade* e *%_população_idosa* teve um *score* atribuído de 1.9. Quanto menor o valor, mais *outlier* a amostra é, naquele subespaço.

Assim, na dimensão mais destoante de Laranjal do Jari, primeira linha, destaca-se à combinação da proporção de população idosa, com mais de 60 anos, consideravelmente abaixo da média, com um dos menores índices de mobilidade presente no Brasil. O retorno do OAM aponta que poucas cidades tão jovens adotaram as medidas de isolamento tão bem.

O terceiro subespaço, elencado entre os mais destoantes, é particularmente inter-

essante. O subespaço é composto por três dimensões, incluindo *%_casos_confirmados* que, de acordo com o *z-score*, não destoa dos demais por si só. Apesar disso, a análise multidimensional sugere que, para o nível de mobilidade e de população idosa da cidade, o número de casos observado é inesperado.

Essa informação pode motivar investigações sobre o que aconteceu na cidade, podendo levar à identificação de erros nos dados, peculiaridades sobre a população, ou sobre as políticas públicas adotadas na cidade. Este tipo de análise mostra a importância da abordagem multidimensional possibilitada pelo OAM. Análises unidimensionais, apesar de simples e amplamente utilizadas, podem desconsiderar informações importantes do problema.

5. Conclusão

A biblioteca *Python-OAM* foi construída para que usuários finais possam usar as técnicas de OAM em seus cenários de aplicação. No caso de uso apresentado, com poucas linhas de código foi possível extrair informações valiosas sobre o combate regional à pandemia de *COVID-19*. Os testes demonstram a aplicabilidade da ferramenta num cenário desafiador, mas familiar. Na prática, a biblioteca foi feita para ser usada nos mais diversos cenários, incluindo casos de alta dimensionalidade, como dados de sensores da Industria 4.0 ou fraudes bancárias.

O projeto da biblioteca visa facilitar a utilização e futuras contribuições da comunidade. Para complementar as técnicas implementadas, também foram desenvolvidos módulos auxiliares de pré-processamento e visualizações de dados. A biblioteca conta com testes unitários para garantir a funcionalidade e extensibilidade do código e a integridade dos resultados gerados. A documentação traz informações sobre OAM, as técnicas escolhidas e sobre cada função disponível. A biblioteca foi disponibilizada com licença Apache License 2.0. O código fonte e a documentação podem ser obtidos em `https://github.com/rodrigo-fss/python-oam`.

References

[Chandola et al. 2009] Chandola, V., Banerjee, A., and Kumar, V. (2009). Anomaly detection: A survey. *ACM Computing Surveys*, 41(3):15:1–15:58.

[Cheng et al. 2019] Cheng, Z., Zou, C., and Dong, J. (2019). Outlier detection using isolation forest and local outlier factor. In *Proceedings of the Conference on Research in Adaptive and Convergent Systems*, RACS '19, pages 161–168, New York, NY, USA. Association for Computing Machinery.

[Samariya et al. 2020a] Samariya, D., Aryal, S., and Ting, K. M. (2020a). A new effective and efficient measure for outlying aspect mining. arXiv: 2004.13550.

[Samariya et al. 2020b] Samariya, D., Ma, J., and Aryal, S. (2020b). A Comprehensive Survey on Outlying Aspect Mining Methods. arXiv: 2005.02637.

[the Vinh et al. 2016] the Vinh, N., Chan, J., Romano, S., Bailey, J., Leckie, C., Ramamohanarao, K., and Pei, J. (2016). Discovering outlying aspects in large datasets. *Data Mining and Knowledge Discovery*, 30.