Universidade Nova de Lisboa

Faculdade de Ciências e Tecnologia

Aprendizagem Automática

Shakey - Clustering de eventos sísmicos

Autores:
Lucas Fischer
Joana Martins

Professor:
Ludwig Krippahl

dezembro de 2018

1 Introdução

O *dataset* utilizado neste projeto continha diferentes informações sobre eventos sísmicos, nomeadamente a sua latitude, longitude e a falha tectónica a que estavam associados. Fomos desafiados a implementar três algoritmos diferentes de *clustering* (*K-Means*, *Gaussian Mixture Model* e DBSCAN) de maneira a conseguir extrair novas informações sobre os nossos dados, visto este ser o objetivo principal da aprendizagem não supervisionada.

Visto estarmos a trabalhar com dados sobre a mesma escala sem grandes desvios na sua magnitude, e visto os *clusters* serem particularmente sensíveis ao processo de estandardização devido a geralmente representarem dados geométricos, o grupo decidiu não utilizar este processo de estandardização, sendo que a única modificação feita aos dados foi uma transformação de latitude e longitude para coordenadas *Earth-Centered, Earth-Fixed* [1] X, Y e Z.

Nas secções seguintes serão abordadas: a estrutura do código realizado bem como uma descrição sobre os três algoritmos usados, a sua implementação e decisões teóricas sobre a sua aplicação a este contexto.

2 Estrutura do código

De maneira a simplificar a implementação do código e eventuais alterações ao mesmo, o grupo decidiu dividir o mesmo em diferentes ficheiros que seguem a estrutura presente na figura 1.

```
Shakey/
    ├── data
    │   ├── tp2_data.csv ; Dados do problema
    │   └── faults.csv ; Informação sobre as falhas tectónicas
    ├── images ; Diretoria  de output de imagens
    ├── assignment.py ; Classe que executa os algoritmos
    ├── helper_functions.py ; Ficheiro de funções auxiliares
    └── tp2.py ; Ficheiro onde é iniciada a aplicação
```

Figura 1: Representação gráfica da estrutura do código desenvolvido para implementar a solução proposta pelo grupo.

3 *K-Means*

O primeiro algoritmo de *clustering* implementado pelo grupo, foi o algoritmo *K-Means*. O *K-Means* constitui uma das mais utilizadas ferramentas no estudo de *clusters*. Nesta secção será abordado um resumo ao seu funcionamento teórico, uma breve descrição sobre como o grupo procedeu para implementar o algoritmo na nossa solução e por fim algumas vantagens/desvantagens, bem como uma possível aplicação da utilização deste algoritmo neste *dataset* em específico.

3.1 Resumo Teórico

De forma a utilizar o este algoritmo é necessário primeiro explicitar o número de *clusters* que se deseja (**o valor de K**), o que pode ser um ponto negativo à utilização do K-Means. Uma vez explicitado o valor de K, o algoritmo dá inicio e escolhe os *prototypes*, i.e. o centro de um *cluster* que neste específico algoritmo não coincide com um ponto dos dados, dos K *clusters* iniciais de uma das seguintes formas:

- Forgy - Utiliza coordenadas de pontos aleatórios do *dataset*
- Partição aleatória - Utiliza valores completamente aleatórios para servirem de coordenadas para os *clusters*

Na implementação da nossa solução foi utilizada a inicialização *Forgy* pois esta era a implementação por defeito na classe `KMeans` da biblioteca `SciKit-Learn` [6, 3]. Após a escolha inicial dos K *clusters* o algoritmo passa por alocar cada ponto ao *cluster* que minimize a distância entre o seu *prototype* e o ponto em questão. Tendo esta primeira alocação de pontos, o algoritmo recalcula o novo centro do *cluster* (o novo *prototype*), sendo este o ponto médio entre todos os pontos presentes num determinado *cluster*, e repete novamente o seu processo até que se chegue a um ponto de convergência, i.e. onde já não haja alterações nos *clusters*.

Tendo esta descrição do algoritmo *K-Means* podemos identificá-lo como sendo um algoritmo com base em *prototypes*, completo (todos os pontos são alocados a um *clusters*), que cria *clusters* exclusivos (um ponto pode apenas pertencer a um *cluster*) e *partitionals* (os clusters estão todos ao mesmo nível).

3.2 Implementação

A implementação deste algoritmo na solução criada pelo grupo encontra-se no ficheiro `assignment.py` na função `k_means`. A implementação deste algoritmo provém da biblioteca `SciKit-Learn` [6, 3] que dispõe uma classe chamada `KMeans`.

```
kmeans = KMeans(n_clusters = k).fit(coords)
```

Esta classe recebe como argumento de inicialização o número de *clusters* que se pretende criar. Após a sua inicialização, é possível executar o método `fit` passando-lhe um *array* de pontos (que neste caso representam as coordenadas do nosso *dataset*). Tendo um objeto da classe `KMeans` detalhado para os nossos dados, podemos agora executar o método `predict`, para obter a alocação de cada ponto a um *cluster*, e obter o valor do atributo `cluster_centers_`, que fornece os *prototypes*.

```
labels = kmeans.predict(coords)
centroids = kmeans.cluster_centers_
```

3.3 Contextualização com o *dataset*

Tendo em conta a natureza não globular dos eventos sísmicos, visto estes ocorrerem com mais frequência em torno de falhas tectónicas contínuas, a utilização do *K-Means* para este contexto torna-se mais desafiadora, uma vez que os *clusters* são criados com base numa distância a um *prototype* e não respeitam esta natureza não globular dos eventos sísmicos. No entanto, este algoritmo, embora não tenha sido o foco de estudo do grupo para este trabalho, apresenta uma possível utilização prática neste contexto que é a compressão dos dados através de quantização vetorial. Esta técnica foca-se em sumarizar os dados a certos pontos de maior interesse de maneira a simplificar o nosso *dataset*. Visto que existe uma grande quantidade de eventos sísmicos todos os dias, pode ser útil sumarizar os dados a exemplos mais relevantes e, neste contexto, a obrigação da escolha inicial do número de *clusters* necessários torna-se algo desejável, porque permite-nos decidir em relação ao contexto da aplicação, quantos exemplos principais queremos.

Na solução proposta pelo grupo, o objetivo era sumarizar o melhor possível a atividade sísmica que ocorre em cada falha tectónica. Para este efeito foram escolhidos três valores para K para serem estudados:

- **7** - Representa o número de principais falhas tectónicas
- **96** - Representa o número total de falhas tectónicas existentes no planeta (principais e secundárias)
- **51** - Representa um valor a meio dos dois outros valores

Para cada um destes valores foi esboçado um gráfico que representa cada evento sísmico colorido de acordo com o *cluster* a que foi alocado, um gráfico que representa uma visualização a três dimensões (3D) dos eventos sísmicos, bem como foram registados os valores métricos de um índice externo *Rand Index*, que, embora não nos permita tirar conclusões diretas sobre a adequabilidade dos *clusters* criados para o nosso objetivo, permite-nos concluir certas características sobre os *clusters* obtidos que podem ser úteis para fundamentar determinadas decisões tomadas.

3.4 Resultados

O primeiro valor de K estudado foi 7 pois representava o número de principais falhas tectónicas, com o objetivo de tentar sumarizar os eventos sísmicos a um exemplo por cada falha tectónica. Após a execução do algoritmo e a alocação de cada ponto a um determinado *cluster* foi esboçado o gráfico presente na figura 2 e também o gráfico presente na figura 3. Contrastando a figura 2 com a figura 3 podemos observar um defeito de utilizar o K-Means neste contexto em específico: conseguimos visualizar sobre a área da América do Sul (do Chile ao Perú) um *cluster* não globular a ser dividido em dois devido à natureza de *prototypes* do K-Means.

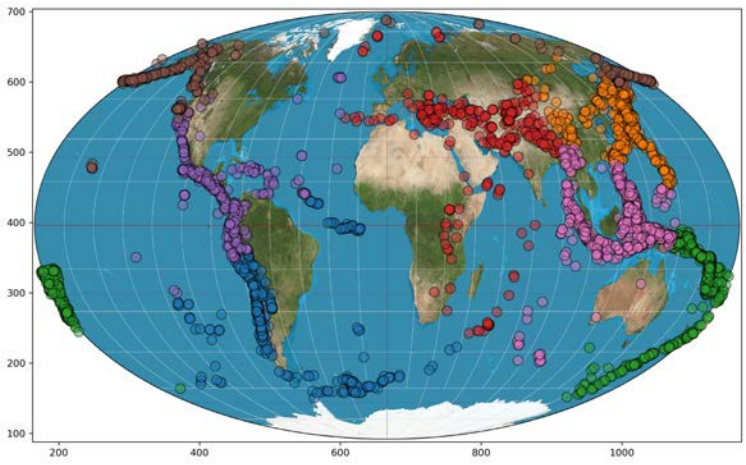

Figura 2: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 7 *cluster*.

Após verificar que com um K igual a 7 temos bastantes pontos a serem representados como pertencentes a falhas que na realidade não pertencem, o grupo decidiu aumentar o valor de K para 51, que representa um valor intermédio entre os valores 7 e 96. Com este valor projectaram-se novamente os pontos sobre a superfície do planeta, coloridos de acordo com o *cluster* a que foram alocados e obteve-se o gráfico da imagem 4.

Na imagem 4 continuamos a observar uma divisão de *clusters* contínuos inevitável devido à utilização do *K-Means* para este tipo de *clusters*, mas começamos a observar que existem agora novos *clusters* que já não são alocados a falhas às quais não deveriam ser alocados. Com esta

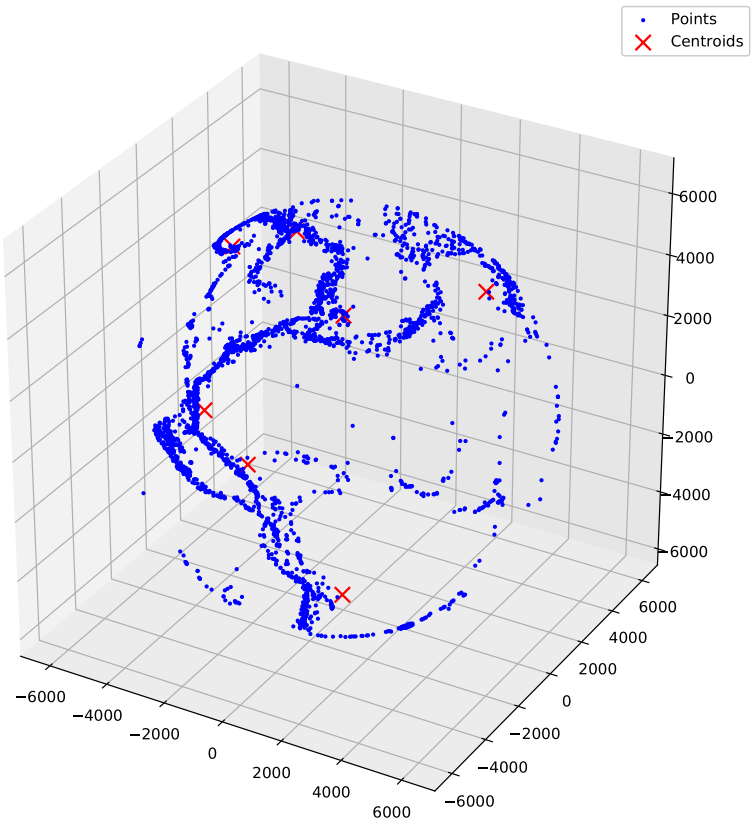

Figura 3: Visualização a três dimensões dos pontos bem como os *prototypes* dos diferentes *clusters*.

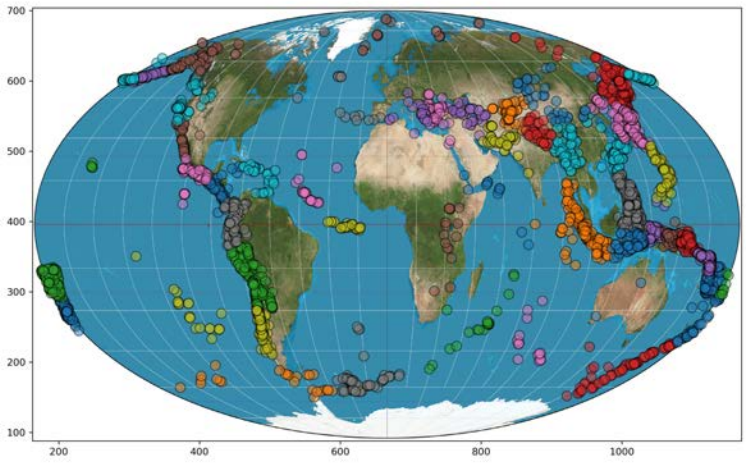

Figura 4: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 51 *cluster*.

observação o grupo decidiu aumentar o valor de K para 96 que representa o número total de falhas (principais e secundárias) existentes no planeta, com o objetivo de representar um ponto por falha. Com este valor, obteve-se a projeção presente na figura 5, de onde se retira uma conclusão semelhante à da execução do K-Means fixando K a 51.

Após a execução do algoritmo para estes três valores, e tendo acesso a um índice externo que

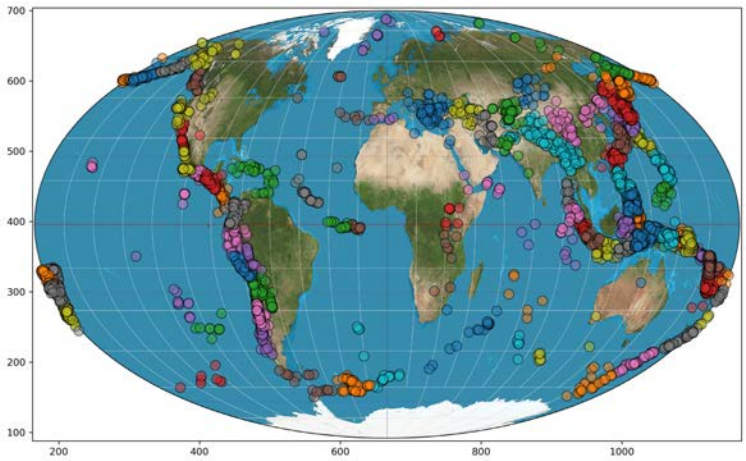

Figura 5: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 96 *cluster*.

continha informação sobre que eventos sísmicos deveriam a pertencer a que falhas tectónicas, torna-se interessante produzir métricas de maneira a conseguirmos a avaliar a adequabilidade dos *clusters*. Embora estas métricas não deem diretamente uma conclusão sobre a adequabilidade de um *cluster*, pois a adequabilidade está muito dependente da aplicação que queremos dar ao *cluster*, elas podem ser indicadores úteis na utilização de *clusters* em certas situações. Por esta razão, esboçou-se um gráfico que regista a variação destas métricas contra a variação do parâmetro K, presente na figura 6.

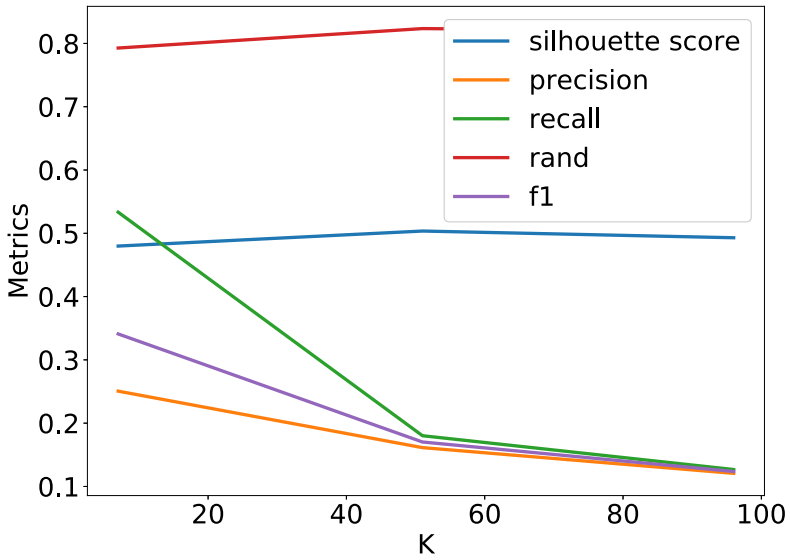

Figura 6: Gráfico que representa a variação das métricas calculadas *vs.* a variação do parâmetro K.

Na figura 6, conseguimos observar uma descida acentuada nas métricas de *precision*, *recall* e *F1*

score que implica que alguns pontos estão a ser alocados a *clusters* que não representam a sua falha tectónica, algo que podemos constatar visualmente ao observar as figuras acima. Contudo, vemos uma subida nas métricas de *silhouette score* e *rand* que sugerem que os *clusters* criados tornam-se cada vez mais coesos, i.e. possuem uma distância menor entre pontos do mesmo cluster e uma distância maior a pontos de outros clusters.

4 *Gaussian Mixture Model*

O segundo algoritmo de *clustering* implementado foi o *Gaussian Mixture Model*. Trata-se de um método de *clustering* probabilístico visto que se assume uma distribuição de probabilidades para as observações e o seu nome está associado ao facto de esta distribuição ser descrita por uma soma ponderada de curvas Gaussianas ponderadas por pesos [2]. Com base nesta distribuição de probabilidades, é possível gerar novos dados, pelo que se diz que este algoritmo é generativo. Segue-se uma breve descrição dos fundamentos deste algoritmo.

4.1 Resumo Teórico

A diferença fundamental deste algoritmo para os restantes algoritmos estudados, está no facto de o *Gaussian Mixture Model* ser um algoritmo de *clustering* probabilístico. Este facto traduz-se em cada ponto ter um valor de probabilidade de pertença a cada um dos diferentes *clusters*. Visto tratar-se de probabilidades, é natural que a soma dos diferentes valores de pertença para um ponto seja 1, o que faz com que este algoritmo se diferencie de algoritmos de *Fuzzy Clustering*.

Como descrito a cima, esta probabilidade de pertença a um *cluster* é descrita por uma soma ponderada de curvas Gaussianas de acordo com pesos (π_k) que representam a probabilidade *a priori* de um ponto pertencer a essa curva Gaussiana. A probabilidade de um ponto pertencer a uma determinada Gaussiana é então dada por:

$$p(x) = \sum_{k=1}^{K} \pi_k \times \mathcal{N}(x|\mu_k, \Sigma) \tag{1}$$

Sabendo que uma curva Gaussiana é, em parte, determinada pelo seu valor/vetor médio podemos então considerar este algoritmo como sendo baseado em *prototypes* tal como o *K-Means*.

O principal desafio neste algoritmo é o facto de não se ter acesso à informação sobre a que Gaussiana cada ponto pertence, valor a que chamamos *latent variable* ou Z. A solução para este problema é conhecida como *Expectation Maximization*. Esta inicialmente calcula aleatóriamente os parâmetros da Gaussiana, bem como o seu peso associado, de maneira a conseguirmos ter uma estimativa inicial desta *latent variable* para que possamos usá-la para maximizar a log-verosimilhança dos parâmetros, e este processo é repetido até que se convirja para uma estimativa final do peso associado à Gaussiana (π), da sua média (μ) e da sua variância (Σ).

4.2 Implementação

A utilização deste algoritmo na solução proposta pelo grupo utiliza a classe `GaussianMixture` disponibilizada pela biblioteca `SciKit-Learn` [6, 3] e encontra-se no ficheiro `assignment.py` na função `gaussian_mix`.

```
gmm = GaussianMixture(n_components =  num).fit(coords)
```

Tal como para o algoritmo K-Means é necessário fornecer um parâmetro `n_components` que explicita o número de curvas Gaussianas a serem utilizadas. De seguida, e de forma semelhante ao

algoritmo K-Means, é possível executar o método `predict` de maneira a obtermos a alocação de cada ponto a um *cluster* para que seja possível calcular e esboçar as diferentes métricas e gráficos necessários para o estudo de *clusters*.

```
labels = gmm.predict(coords)
```

Note-se que, tal como no caso do *K-means*, também o *Gaussian Mixture Model* é completo, porque alocada todos os pontos a *clusters* e os *clusters* que cria são *partitionals*, por se encontrarem ao mesmo nível.

4.3 Contextualização com o *dataset*

Tal como no caso do algoritmo *K-means*, o *Gaussian Mixture Model* também necessita como parâmetro de *input* com o número de *clusters* a criar. No entanto, ao contrário do *K-means*, o *Gaussian Mixture Model* baseia-se em probabilidades *a posteriori* de cada evento pertencer a um dado *cluster*. Seria possível pensar que este facto o tornaria adequado ao estudo das zonas de maior densidade de eventos sísmicos, a aplicação que nos propusemos estudar neste trabalho. Contudo, é importante aqui frisar que uma das assunções do modelo é a de que estas probabilidades se podem modelizar por médias ponderadas de Gaussianas. Desta forma, está-se a condicionar um possível perfil de densidade de eventos nelas baseado. Conclui-se por isso que este algoritmo também não será o mais indicado para o objectivo fixado. Assim sendo, o estudo paramétrico aqui efectuado da obtenção de *clusters* com o algoritmo *Gaussian Mixture Model* tem apenas o propósito de estudar a sua utilização.

A fixação dos valores de K (número de *clusters*) foi por isso apenas baseada na escolha de alguns valores entre limites que fizessem sentido e igual aos K escolhidos para o caso do *K-means*. O limite mínimo do número de *clusters* escolhido foi o do número das falhas principais (*major fault lines*), visto que um número inferior a este não se justificaria à partida, e o número superior igual ao número total de falhas, i.e., noventa e seis. O terceiro valor corresponde simplesmente ao ponto médio entre estes dois.

4.4 Resultados

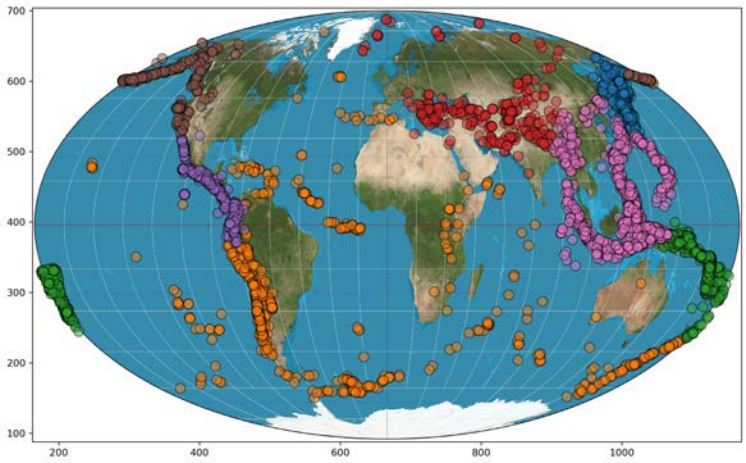

Figura 7: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 7 *cluster* obtidos com o algoritmo *Gaussian Mixture Model*.

Os *clusters* obtidos com o *Gaussian Mixture Model* para $K = 7$, $K = 51$ e $K = 96$ encontram-se identificados graficamente nas figuras 7, 8 e 9, coloridos de acordo com a sua etiqueta (i.e. o *cluster* a que pertencem). Da comparação destes gráficos com os obtidos para o mesmo valor de K

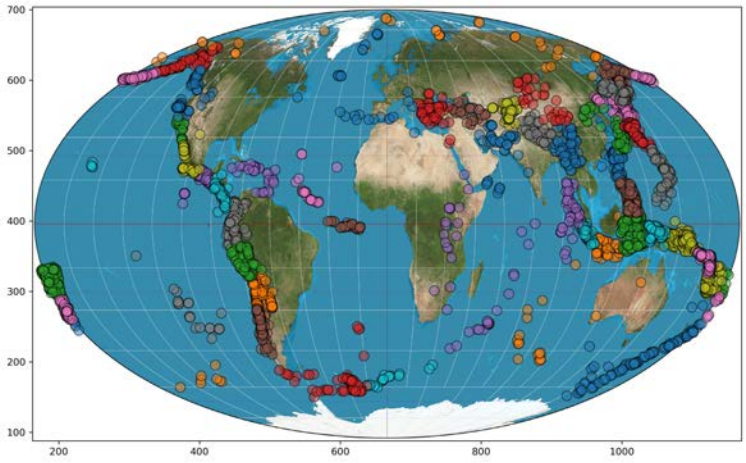

Figura 8: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 51 *cluster* obtidos com o algoritmo *Gaussian Mixture Model*.

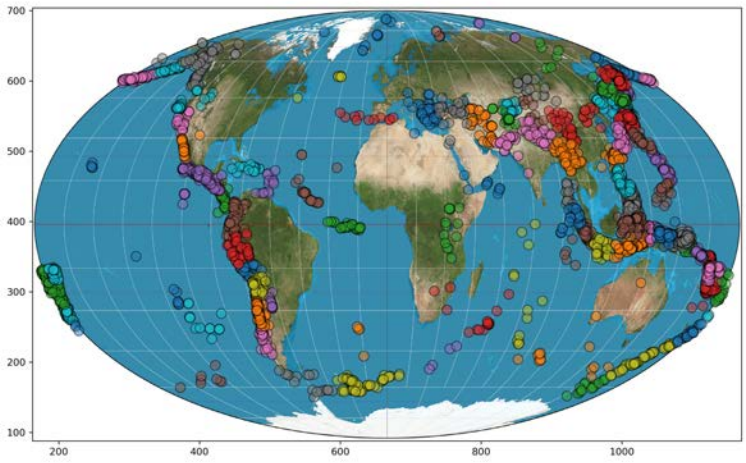

Figura 9: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos 96 *cluster* obtidos com o algoritmo *Gaussian Mixture Model*.

com o algoritmo *K-means* verifica-se que produzem *clusters* diferentes. Como exemplo, observa-se que no caso K=7, na figura 7, o cluster a laranja contém agora eventos no sudeste africano e a Sul da Península Ibérica que anteriormente não pertenciam ao *cluster* que englobava os eventos sísmicos na América do Sul. Tal não é de estranhar uma vez que o processo de os construir é diferente. Conforme o peso e variância da Gaussiana que representa este *cluster*, o seu centróide estará possivelmente mais deslocado para Este da América do Sul.

Importa também referir que nos gráficos com maior número de *clusters*, $K = 51$ e $K = 96$ deverão existir mais do que um *cluster* com a mesma cor visto que o número *default* de cores pelas quais o `MatplotLib` percorre é de sete. Como aqui se pretende apenas distinguir visualmente a

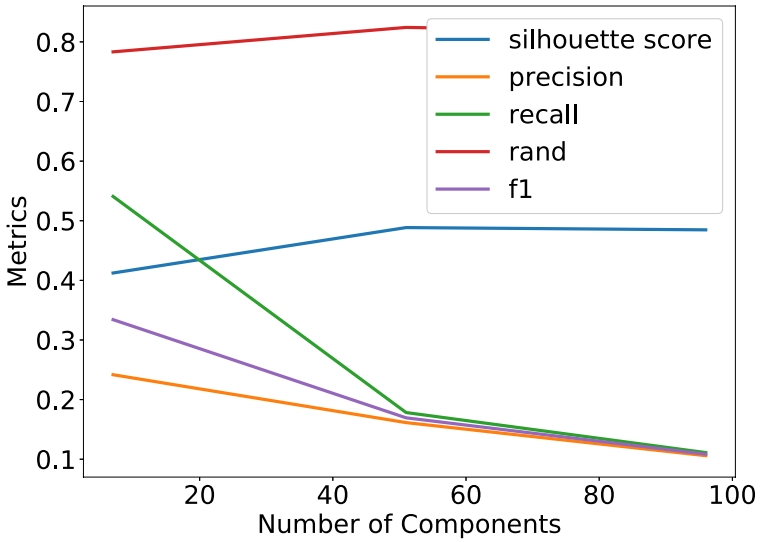

Figura 10: Gráfico que representa a variação das métricas calculadas *vs.* a variação do parâmetro K para o algoritmo *Gaussian Mixture Model*.

localização de diferentes *clusters*, este facto não deverá ser significativamente limitativo.

Na figura 10 apresentam-se diferentes métricas para o algoritmo *Gaussian Mixture Model* em função do número de *clusters* K. A análise da métrica *silhouette score* sugere, pelo seu aumento, que a distância entre pontos dentro de cada *cluster* estará a diminuir ligeiramente e a distância média entre *clusters* distintos a aumentar. Isto parece ser corroborado pelas imagens que representam os *clusters* sobre a projecção de Mollweide. Com o aumento do K, diferentes *clusters* parecem tornar-se mais compactos. Por outro lado, a precisão, o *recall* e o F1 diminuem significativamente com o aumento de K. Esta tendência parece indicar um aumento do número de eventos que estão alocados a um mesmo *cluster* mas que não pertencem a um mesmo grupo, neste caso a uma mesma falha, ou o contrário.

5 DBSCAN

O DBSCAN é um algoritmo que se baseia no conceito de densidade para determinar *clusters* em *datasets* relativos a quantidades espaciais [4]. Entre as suas vantagens, encontram-se o facto de não necessitar de muita informação acerca do domínio do *dataset*, permitir descobrir *clusters* de formas arbitrárias e ser eficiente para *datasets* de grande volume de dados [4]. Por outro lado, necessita de dois parâmetros de *input*, o número mínimo de pontos na vizinhança de um ponto e ϵ (distância entre os pontos na vizinhança), sendo que o número mínimo de pontos na vizinhança é fixado a 4 como descrito em [4] e sendo fornecido um método para auxiliar o utilizador a encontrar o parâmetro ϵ [4].

Uma distinção importante do DBSCAN em relação a algoritmos previamente estudados neste trabalho é a de que não existem protótipos associados a cada *cluster*. Não existem pontos representativos dos *clusters*. A pertença ou não de um ponto a um *cluster* é baseada em condições que envolvem apenas a distância e o número mínimo de pontos na sua vizinhança de modo a não ser considerado como ruído.

O DBSCAN é ainda um algoritmo de *clustering* dito parcial, ao contrário dos restantes algoritmos aqui estudados (que se dizem completos) porque permite ter pontos não alocados a nenhum

cluster. Este facto está intimamente associado à noção de ruído, que o DBSCAN contempla.

5.1 Resumo Teórico

A descoberta de *clusters* e o seu aglomeramento são determinados com base na ideia intuitiva de que cada ponto pertencente a um *cluster* deverá conter na sua vizinhança (de raio ϵ) um determinado número mínimo de pontos (o que equivale a escolher um limite mínimo para a densidade) [4]. A implementação deste conceito é contudo um pouco mais complexa visto que os pontos na fronteira necessariamente terão um número de vizinhos menor que pontos mais interiores.

Assim, define-se um *cluster* C como sendo o conjunto de pontos do *dataset* tais que (para um determinado ϵ e $MinPts$): se $p \in C$ e q é um ponto *density-reachable* de p, então q também pertence ao mesmo cluster; e se p, q são um qualquer par de pontos pertencente a C, então p e q são *density-connected* [4]. Nesta definição, a vizinhança N_ϵ de p é o conjunto de pontos não inferior a $MinPts$ que se encontram a uma distância igual ou inferior a ϵ de p [4]. Além disso, dois pontos dizem-se *density-reachable* se um deles é um *core point* e entre eles existe uma sequência de um ou mais pontos intermédios tais que cada um deles está na vizinhança ϵ do anterior [4]. Finalmente, dois pontos dizem-se *density-connected* se existe entre eles um ponto ao qual ambos são *density-reachable* [4].

O processo de descoberta de *clusters* processa-se em linhas gerais em duas etapas [4]. Primeiramente, é escolhido um ponto arbitrariamente e são procurados todos os pontos *density-reachable* a partir dele (para um dado ϵ e $MinPts$). Se forem encontrados pontos nestas condições, eles formam um cluster. Se tal não for possível, procura-se um novo ponto para iniciar a construção de um *cluster*, que deverá ter no mínimo $MinPts$ [4]. Sempre dois clusters estiverem separados por uma distância inferior a ϵ, estes são aglutinados num único *cluster* [4].

5.2 Implementação

A implementação do algoritmo DBSCAN está contida no método `dbscan` que se encontra no ficheiro `assignment.py`. Este método recebe como argumento opcional uma lista de valores de ϵ e encontra-se dividida em duas partes que são executadas ou não de acordo com a presença do mesmo. A ideia por detrás desta organização é a de permitir dois modos de utilização do algoritmo. Se o utilizador quer fazer uma análise paramétrica dos resultados do algoritmo em função do parâmetro ϵ, este método é executado com uma lista de valores. Se, por outro lado, não é passado nenhum valor para este parâmetro, antes de se proceder ao *clustering* é executada a função `select_epsilon` que visa auxiliar o utilizador a escolher um valor para ϵ, e que se encontra no ficheiro `helper_functions.py`. Seguidamente será descrito este processo, que segue o método descrito em [4].

A função `select_epsilon` recebe as coordenadas dos eventos sísmicos e retorna o valor escolhido pelo utilizador. Com as posições dos eventos, começa-se por determinar as distâncias até ao quarto vizinho mais próximo. Isto é feito tomando partido da classe `KNeighborsClassifier` da biblioteca `scikit-learn` [6, 3], que é aqui utilizada apenas para obter as distâncias referidas e não para efeitos de classificação. A classe é inicializada com um valor de quatro para o número de vizinhos mais próximos, como descrito em [4], pois este valor não se torna computacionalmente dispendioso e retorna resultados semelhantes ao que se obteria fixando o número de vizinhos a um valor maior que quatro. Depois de feito o *fit* aos dados (coordenadas dos eventos), é chamado o método `kneighbors` que retorna uma lista com as distâncias ao quarto ponto vizinho para todos os pontos no *dataset*. Esta lista é então ordenada por ordem descendente e enviada para a função `get_user_epsilon` onde efectivamente é feita a escolha do valor de ϵ.

Na função `get_user_epsilon`, é pedido ao utilizador que indique a percentagem de pontos que

considera fazerem parte do ruído (i.e., não pertencerem a nenhum *cluster*). É então apresentado ao utilizador um *scatter plot* da distância ao quarto vizinho mais próximo (*4-dist*, figura 11) em função do índice dos pontos ordenados.

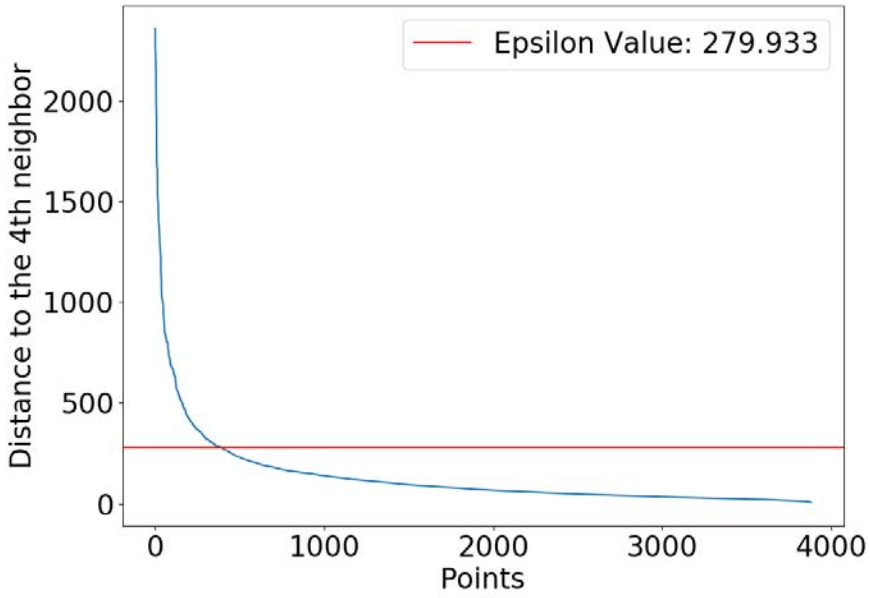

Figura 11: *Scatter plot* da distância ao quarto vizinho mais próximo em função do índice dos pontos ordenados decrescentemente.

Sobreposto a este, é também representada a linha horizontal de separação dos pontos de ruído (acima da linha) e dos pontos pertencentes a *clusters* (abaixo da linha) de acordo com a percentagem de ruído estimada. Com base neste gráfico, o utilizador pode verificar visualmente se a sua estimativa corresponde ao ponto de inflexão onde se dá uma mudança significativa nas *4-dist*. Esta mudança está associada a uma transição de pontos espacialmente mais distantes, que constituem ruído, de pontos mais próximos, que pertencerão a *clusters*. A qualidade da estimativa inicial do utilizador não é importante visto que após a representação gráfica da mesma é pedida a sua aceitação ou não. Se for rejeitado o valor de ϵ correspondente à estimativa proposta é repetido o processo descrito anteriormente, voltando-se a pedir uma nova estimativa até que o utilizador aceite o valor de ϵ. É este valor que é seguidamente devolvido pela função `select_epsilon` ao método `dbscan`.

O passo seguinte é a execução efectiva do algoritmo DBSCAN, utilizando a classe `DBSCAN` da biblioteca `scikit-learn` [6, 3]. Esta é inicializada com o parâmetro ϵ escolhido e com o número mínimo de pontos num *cluster* igual quatro. Depois de feito o *fit* ao *dataset* deste estudo, podem-se obter as etiquetas de cada ponto, que distinguem o *cluster* a que pertencem, chamando o método *labels_*.

Quer seja escolhido um único valor de ϵ, quer seja passada uma lista de valores, é chamado o método `plot_classes`, adaptado a partir do código fornecido no enunciado [5] e feita a representação gráfica dos *clusters* sobre a projecção de Mollweide da superfície da Terra. Cada evento sísmico é representado por um círculo colorido de acordo com a sua etiqueta, i.e., de acordo com o *cluster* a que pertence. Os pontos que foram considerados como ruído são representados por círculos mais pequenos e de cor preta para melhor distinção.

Após a representação gráfica dos *cluster* e tal como nos algoritmos anteriormente apresenta-

dos, são calculadas várias métricas: o *silhouette score* através do método `silhouette_score` da biblioteca `sklearn.metrics` (com um índice interno), e a *precision*, *recall*, *rand index* e F1, com o auxílio de um índice externo determinado a partir da informação das falhas. Estas métricas são apresentadas na forma gráfica quando é utilizado mais do que um valor de ϵ (com a chamada ao método `plot_params`), e impressas na linha de comandos no caso contrário.

No caso em que o utilizador fornece uma lista de valores de ϵ são ainda apresentados dois gráficos de barras para auxílio na interpretação dos resultados: um com o número de pontos em cada *cluster* encontrado, e o outro com a média de todas as distâncias entre qualquer par de pontos de cada *cluster*; e os histogramas correspondentes.

5.3 Contextualização com o *dataset*

Sendo este algoritmo baseado na noção de densidade e capaz de descobrir *clusters* de forma arbitrária, torna-se natural aplicá-lo ao estudo da densidade de eventos sísmicos. Neste contexto, os *clusters* corresponderão a zonas de densidade elevada de eventos sísmicos (no sentido de zonas de maior número de eventos por unidade de área) e o ruído estará associado a zonas em que existem poucos eventos.

Por outro lado, uma vantagem deste algoritmo é a de que ajuda o utilizador a encontrar o parâmetro de *input*: a distância associada à densidade mínima de um *cluster*, de forma intuitiva. O utilizador não tem que pesquisar ou ter conhecimento previamente do que será uma distância/densidade significativa para este parâmetro porque o processo gráfico de auxílio à escolha fornece de forma natural este valor como se irá detalhar na secção seguinte.

Note-se que seria possível implementar alternativamente uma escolha de ϵ baseada num limite mínimo de densidade (por unidade de área neste caso).

Também a escolha de qual a distância a usar na pesquisa gráfica da transição ruído - *cluster* poderia ser outra que não a distância ao quarto vizinho mais próximo. No entanto, é referido em [4] que os gráficos equivalentes para distâncias ao k-ésimo vizinho mais próximo para $k > 4$ não variam significativamente em relação ao caso $k = 4$ pelo que não se exploraram outras distâncias.

5.4 Resultados

5.4.1 ϵ escolhido pelo utilizador

No modo de execução em que o utilizador fornece uma estimativa do ruído nos dados, foi escolhido um nível de ruído de 10%. O gráfico de *scatter plot* que representa as $4-dist$ (distância ao quarto vizinho mais próximo) para todos os pontos do *dataset* ordenadas decrescentemente encontra-se na figura 11. Nela é possível observar também a linha que identifica a ordenada da distância correspondente a ter os 10% de pontos de ruído. Esta distância foi determinada pelo programa, sendo $\epsilon = 279.9\,\text{km}$. Com este valor de ϵ e o número de pontos mínimo igual a quatro (sugerido em [4]), a execução do algoritmo produziu os *clusters* representados na figura 12. Torna-se interessante contrastar este resultado com um dos obtidos com o $K-means$, que é um algoritmo completo, e portanto atribui todos os pontos a *clusters*. Assim, comparando as figuras 12 e 2, por exemplo, verifica-se que uma parte significativa dos eventos que apareciam praticamente isolados nos oceanos foram considerados ruído, como seria de esperar. No entanto, também eventos a Sul da Península Ibérica, no Sudeste de África e em algumas zonas da Ásia desapareceram, o que talvez não fosse tão fácil de prever. Isto demonstra o potencial do algoritmo para providenciar informação acerca da intensidade (em termos de densidade de eventos) sísmica. Adicionalmente, a escolha de múltiplos valores de ϵ permite fazer estudos exploratórios sobre as zonas de diferentes níveis de densidade

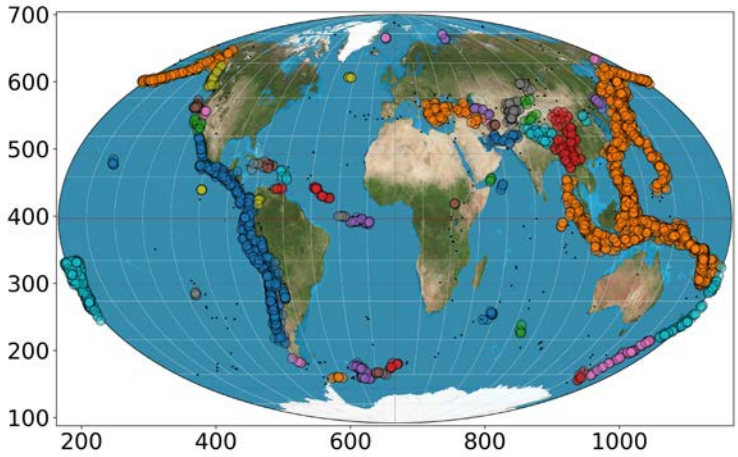

Figura 12: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos *clusters* obtidos com o algoritmo DBSCAN para $\epsilon = 279.9\,\text{km}$.

(ϵs mais baixos conduzem a *clusters* com pontos cada vez mais próximos e portanto identificarão zonas de cada vez maior intensidade).

5.4.2 Estudo paramétrico de ϵ

O algoritmo DBSCAN também foi executado no modo em que é fornecida uma lista de valores de ϵ como input. Para o estudo aqui apresentado optou-se por escolher diferentes percentagens de ruído previamente e correr o programa previamente em modo interactivo para obter os valores de ϵ correspondentes que foram posteriormente fornecidos como input na forma de uma lista. Os

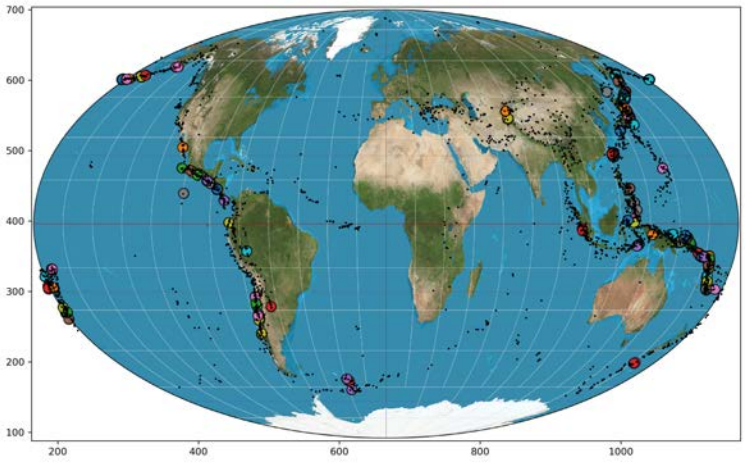

Figura 13: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos *clusters* obtidos com o algoritmo DBSCAN para $\epsilon = 23.3\,\text{km}$.

resultados aqui apresentados correspondem à execução do DBSCAN com $\epsilon = 23.3\,\text{km}$, $36.7\,\text{km}$ e $69.7\,\text{km}$, associados a níveis de ruído de 90%, 75% e 50% respectivamente. Os *clusters* obtidos encontram-se identificados com cores diferentes nas figuras 13, 14 e 15, sendo os pontos mais pequenos a negro os eventos considerados como ruído. Nestas figuras é desde logo notória a diferença

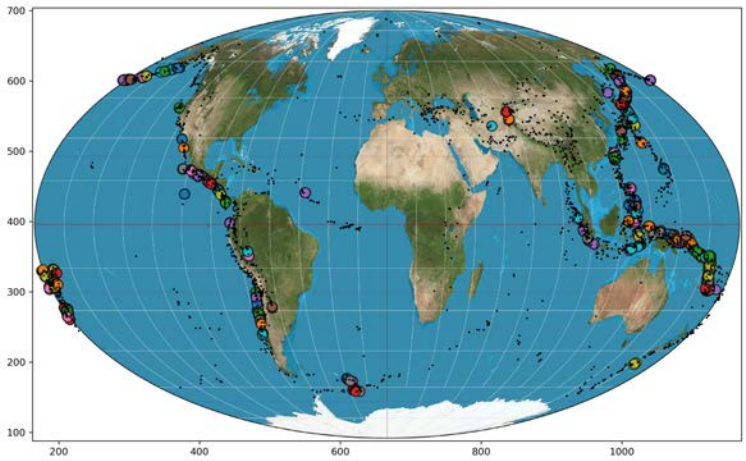

Figura 14: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos *clusters* obtidos com o algoritmo DBSCAN para $\epsilon = 36.7\,\text{km}$.

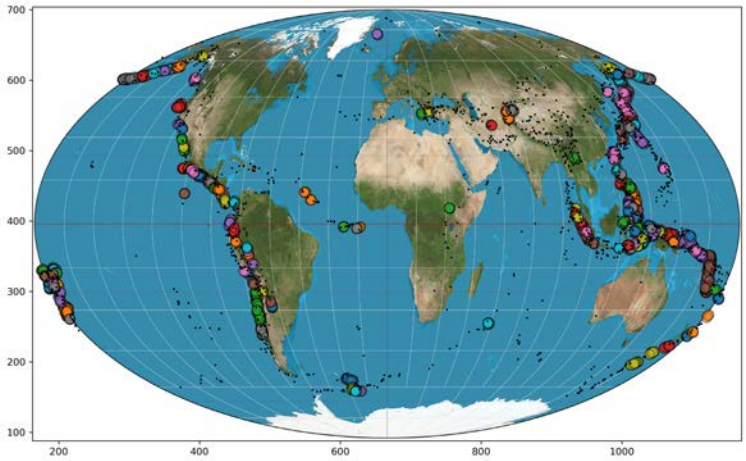

Figura 15: Projeção dos eventos sísmicos sobre o planeta coloridos de acordo a sua alocação a um dos *clusters* obtidos com o algoritmo DBSCAN para $\epsilon = 69.6\,\text{km}$.

em relação ao caso apresentado na secção anterior (figura 12). Tal como esperado, uma vez que os valores de ϵ agora utilizados são muito inferiores ao da secção anterior, observamos um número de pontos alocados a *clusters* bastante inferior. Quando menor o valor de ϵ, mais restrito se torna o número de eventos que se encontram a uma distância inferior a ϵ do quarto vizinho mais próximo e que constituem grupos de pelo menos quatro pontos nestas condições. Posto de outra forma, quanto menor o ϵ, maior o nível mínimo de densidade de eventos que utilizamos como *threshold*, pelo que naturalmente menor é o número de *clusters* de eventos nessas condições.

Em particular, é notório que muitos dos eventos na zona da Grécia e Turquia, na zona a norte da Índia, e nas zonas a norte e a este da Arábia Saudita, não se encontram agora alocados a nenhum *cluster*, ao contrário do que acontecia com o $\epsilon = 279.9\,\text{km}$. De facto, as zonas de maior densidade de eventos para os valores de ϵ mais restringentes usados nesta secção parecem encontrar-se essencialmente na zona do anel de fogo do Pacífico. Este exemplo demonstra mais uma vez a utilidade deste algoritmo. Quando apenas se apresenta a totalidade dos eventos sísmicos sobre

a projecção de Mollweide não é possível identificar com clareza as zonas de diferentes níveis de densidade devido à sobreposição dos pontos. Se quiséssemos ser mais específicos e tivéssemos em mente procurar intervalos de densidade estipulados, seria possível inclusivamente formular este estudo paramétrico em termos de limites mínimos de densidade de eventos, uma vez que é possível associar a cada ϵ uma densidade *threshold*. Aqui pretendeu-se apenas demonstrar e ilustrar o potencial do algoritmo.

As diferenças entre as representações gráficas dos eventos coloridos de acordo com o *cluster* a que pertencem para os três valores de ϵ empregues nesta secção não são tão grandes como quando comparadas com a da secção anterior. Assim sendo, para compreender melhor os resultados obtidos, procedeu-se à representação gráfica do número de eventos em cada *cluster* (figures 16,17 e 18) e da média das distâncias entre todos os pares de pontos dentro de cada *cluster* (figuras 19, 20 e 21).

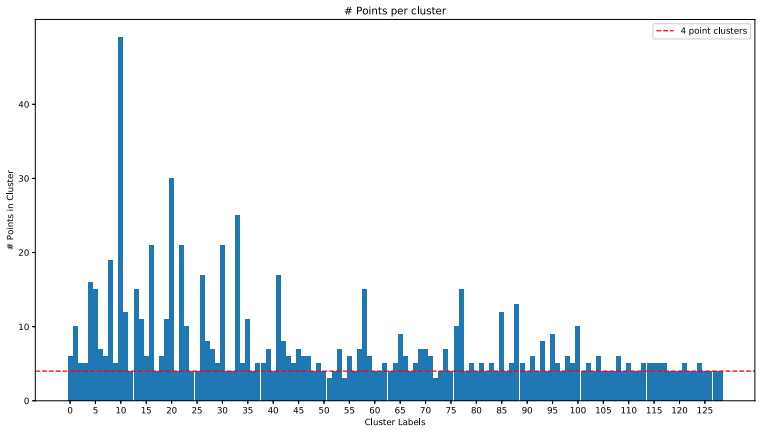

Figura 16: Gráfico que representa o número de eventos alocados a cada *cluster* calculado com algoritmo DBSCAN para $\epsilon = 23.3\,\text{km}$.

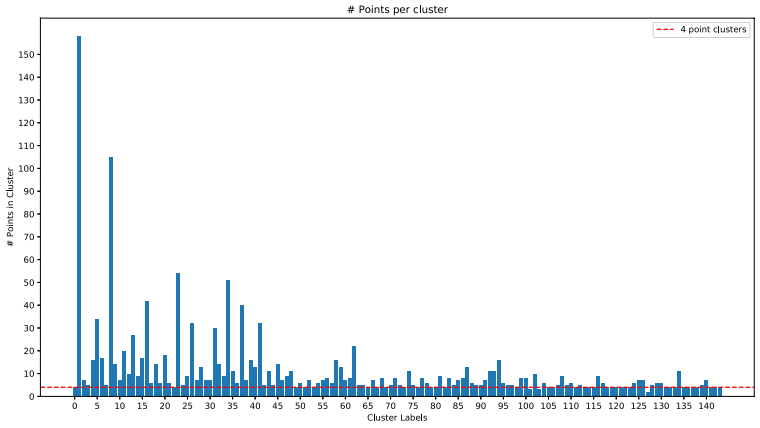

Figura 17: Gráfico que representa o número de eventos alocados a cada *cluster* calculado com algoritmo DBSCAN para $\epsilon = 36.7\,\text{km}$.

Da análise destes gráficos de barras (figuras 16, 17 e 18) e dos histogramas relativos às mesmas quantidades (figura 22), foi possível observar vários efeitos resultantes variação de ϵ.

Quando aumentamos o valor de ϵ de 23.3 km para 36.7 km, por exemplo, verifica-se que por um lado, o número de eventos nos *clusters* com mais pontos aumenta significativamente; e por outro,

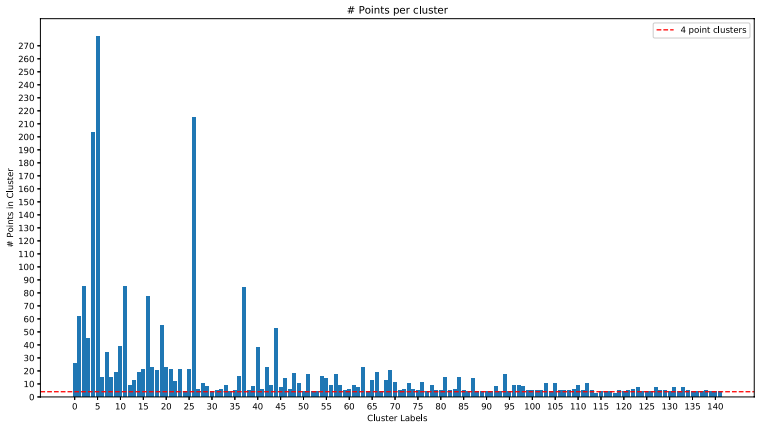

Figura 18: Gráfico que representa o número de eventos alocados a cada *cluster* calculado com algoritmo DBSCAN para $\epsilon = 69.6\,\text{km}$.

o número total de *clusters* aumenta ligeiramente. Isto sugere que o aumento do ϵ veio permitir que eventos que anteriormente estavam demasiado longe de outros pontos, e portanto não poderiam formar com estes um *cluster* agora sejam agrupados num mesmo. Se estes últimos já formavam por si um *cluster*, então o número de eventos deste aumenta; se pelo contrário ainda não formavam um *cluster* com o ϵ anterior, então é criado um novo *cluster*.

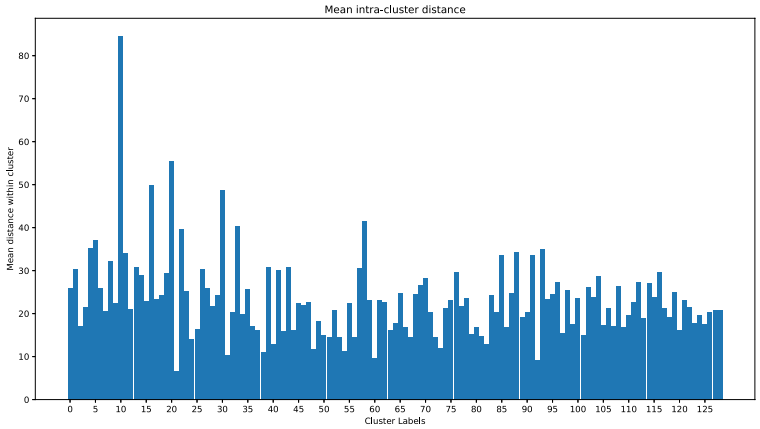

Figura 19: Gráfico que representa a média das distâncias entre dois pontos, para todos os pares de pontos em cada *cluster* obtido com algoritmo DBSCAN para $\epsilon = 23.3\,\text{km}$.

Por outro lado, nos histogramas relativos ao número de pontos por *cluster*, verifica-se quando ϵ passa de 23.3 km para 36.7 km (painéis (a) e (c) da figura 22), a barra correspondente aos *clusters* com até quatro pontos diminui, o que sugere também que alguns deles aumentam o seu número de pontos à custa da fusão com outros.

Na transição de $\epsilon = 36.7\,\text{km}$ para $\epsilon = 69.6\,\text{km}$, verifica-se novamente nos histogramas (painéis (c), (d), (e) e (f)) uma diminuição do número de *clusters* com até quatro pontos ou quatro a oito pontos (duas primeiras barras), o que sugere que alguns dos pontos destes estão a ser aglutinados a outros *clusters* maiores.

Em relação à média das distâncias entre pontos *intra-cluster* para estes dois casos, observa-se que, se por um lado o número de *clusters* para os intervalos de médias de distâncias mais baixas diminui, por outro surgem para $\epsilon = 69.6\,\text{km}$ mais clusters em *bins* com médias de distâncias internas

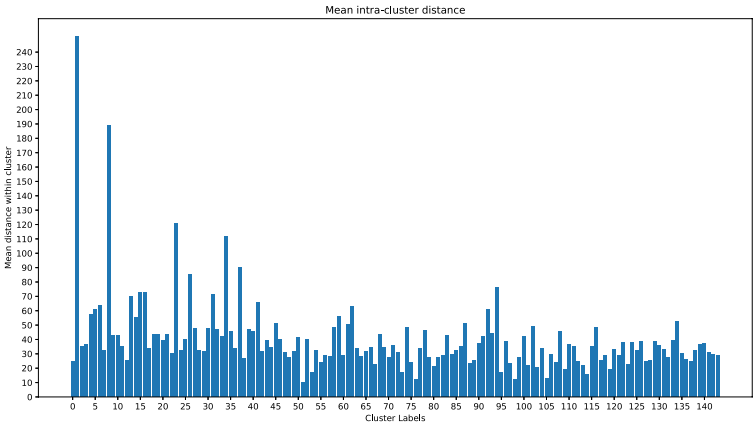

Figura 20: Gráfico que representa a média das distâncias entre dois pontos, para todos os pares de pontos em cada *cluster* obtido com algoritmo DBSCAN para $\epsilon = 36.7\,\text{km}$.

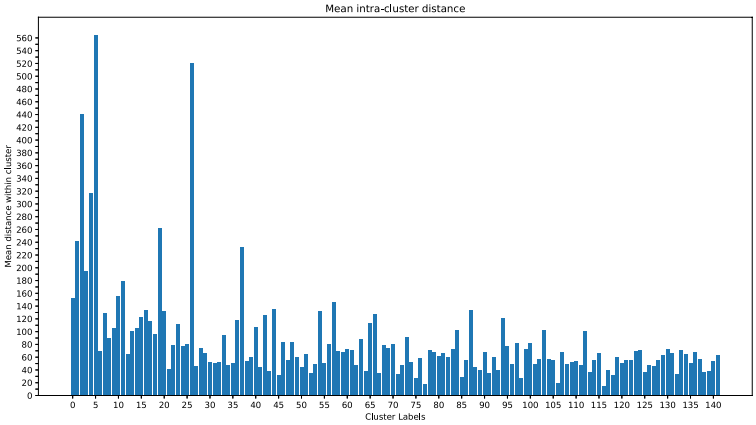

Figura 21: Gráfico que representa a média das distâncias entre dois pontos, para todos os pares de pontos em cada *cluster* obtido com algoritmo DBSCAN para $\epsilon = 69.6\,\text{km}$.

muito superiores às verificadas para $\epsilon = 36.7\,\text{km}$. Isto vai de encontro à ideia de que alguns eventos de *clusters* mais pequenos estão a fundir-se a outros maiores e que, alguns destes *clusters* resultantes da fusão de outros têm eventos que se extendem por uma área maior, levando a um aumento da média de distâncias *intra-cluster* nestes casos.

É interessante notar que, nos gráficos de barras com a média das distâncias para todos os pares de eventos dentro de um mesmo *cluster* (figuras 19,20 e 21) e nos correspondentes histogramas (painéis (b),(d) e (f) da figura 22) existem valores muito acima do ϵ respectivo. Tal deve-se ao facto de esta média ser calculada sobre todos os pares de pontos de cada *cluster*. Se um *cluster* for relativamente grande pode verificar-se simultaneamente a existência de pontos em extremos opostos do *cluster* que estejam a uma distância significativamente maior que ϵ mas que são *density-connected* para o ϵ e número mínimo de pontos fixado.

Tal como para os outros algoritmos foram também calculadas as diferentes métricas a partir de um índice externo. Na figura 23 está presente uma representação da variação destas métricas em função dos diferentes ϵ escolhidos. Como dito anteriormente, estas métricas não nos permitem tirar conclusões diretas sobre os nossos *clusters*, mas permitem-nos observar certas características sobre os mesmos. Uma primeira observação que podemos fazer sobre a figura 23 é o facto da métrica de

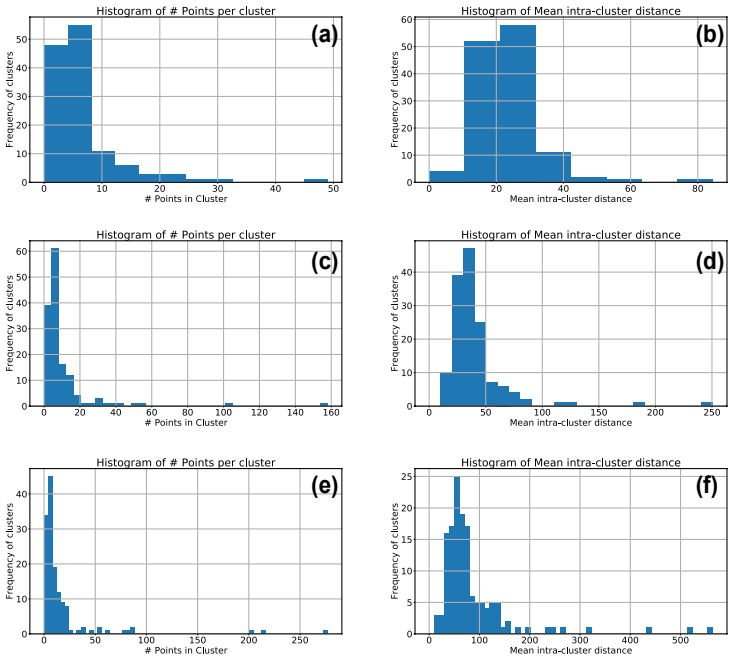

Figura 22: Gráfico que os histogramas relativos à frequência de *clusters* vs o número de pontos que contêm (painéis (a),(c),(e)) e à média das distâncias entre pares de pontos em cada *cluster*, obtidos com algoritmo DBSCAN para $\epsilon = 23.3\,\text{km}$, $36.7\,\text{km}$ e $69.6\,\text{km}$.

precision manter-se relativamente constante em relação à variação de ϵ. Esta situação deve-se ao facto de neste estudo considerarmos apenas valores ϵ muito pequenos, fazendo com que pequenos grupos de pontos sejam considerados como *clusters* (como podemos verificar nas figuras 16, 17 e 18). Esta situação acaba por criar vários *clusters* sobre uma falha tectónica só que deveria ser considerado como um só *cluster*, isto leva a que o número de *False Positives* seja muito mais elevado do que o número de *True Positives* que por sua vez faz manter a *Precision* perto de 0. Poderia ser tentador maximizar esta métrica como seria desejável em aprendizagem supervisionada, no entanto isto desviar-nos-ia do nosso objetivo principal de encontrarmos zonas de maior densidade sísmica.

Outra observação importante será o crescimento da métrica *silhouette score*. Inicialmente esta métrica é aproximadamente -0.5 o que pode indicar que os pontos dos nossos *clusters* têm uma distância média grande entre si, ou que a distância média entre os *clusters* é muito pequena. Contrastando este facto com as figuras 16, 17 e 18, em que vemos que temos diferentes *clusters* muito próximos entre si concluímos então que é este o factor que está a determinar o *silhouette score*. Conforme é aumentado o valor de ϵ vemos um aumento também no *silhouette score*, o que nos indica que alguns *clusters* estão a ser agrupados em *clusters* maiores, aumentando assim a distância entre *clusters* e por sua vez aumentando também o *silhouette score*.

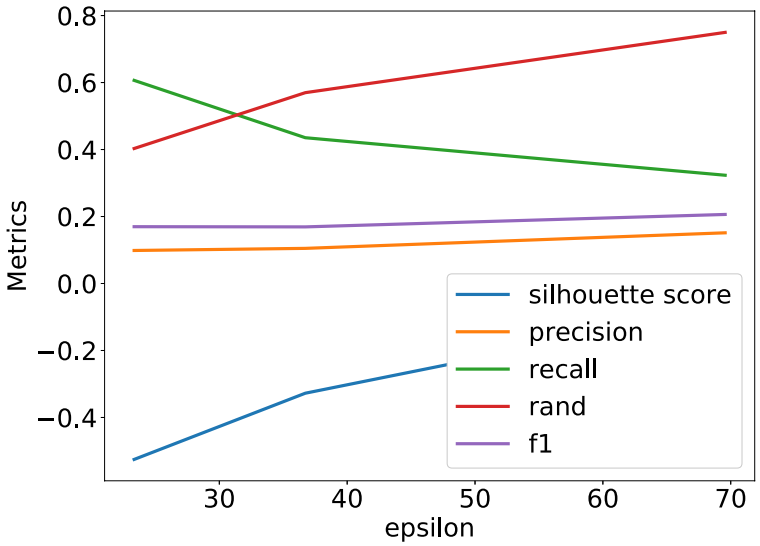

Figura 23: Gráfico que representa a variação das métricas calculadas *vs.* o parâmetro ϵ para o algoritmo DBSCAN.

Referências

[1] ECEF, 2018. URL: https://en.wikipedia.org/wiki/ECEF.

[2] Ethem Alpaydin. *Instroduction to Machine Learning*, chapter Supervised learning. The MIT Press, 2014.

[3] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[4] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996. URL: http://dl.acm.org/citation.cfm?id=3001460.3001507.

[5] Ludwig Krippahl. Aprendizagem automática 2018/19, 2018. URL: http://aa.ssdi.di.fct.unl.pt/tp2.html.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.