Trabalho 02 — Algoritmo de Clusterização

[201804940002] Eduardo Gil S. Cardoso [201804940016] Gabriela S. Maximino
[201704940007] Igor Matheus S. Moreira

26 de dezembro de 2020

Este trabalho é referente à disciplina de Inteligência Artificial do curso de Bacharelado em Ciência da Computação na Universidade Federal do Pará. Ele propõe a implementação de um algoritmo de clusterização para a resolução de um problema estratégico de localização de três aeroportos a serem construídos em Belém, PA.

Durante todo o ciclo de desenvolvimento deste trabalho, o GitHub foi utilizado como ferramenta de versionamento. O histórico de desenvolvimento da equipe, bem como versões iniciais do código aqui contido, está disponível ao público no GitHub em **@ygarasab/kmeans**.

Observação: antes de interagir com o código no Jupyter Notebook/Lab, é importante observar que certas células anteriores à que se quer executar podem ser necessárias. A fim de evitar isso, é importante certificar-se de executar ao menos uma vez todas as células contendo definições de funções ou importações de módulos.

Sumário

1 Requisitos

1.1 Ambiente

Este trabalho foi feito utilizando a seguinte linguagem de programação:

- `python` 3.8.5

Em adição, os seguintes módulos precisam estar instalados no ambiente em que este notebook for executado:

- `dask` 2.19.0
- `ipypublish` 0.10.12
- `matplotlib` 3.3.2
- `numba` 0.52.0
- `numpy` 1.19.2
- `pandas` 1.1.4
- `scipy` 1.5.3
- `seaborn` 0.11.0

Observe que o código pode funcionar em versões distintas às que foram mencionadas acima; contudo, sabe-se que a sua execução é garantida nas versões mencionadas.

[1]:
```
import decimal as d
import numba as nb
import numpy as np
import random as r
import seaborn as sns
import typing as t

from dask import distributed
from ipypublish import nb_setup
from scipy import stats
from timeit import default_timer as timer
```

[2]:
```
%%capture

plt = nb_setup.setup_matplotlib(output=('pdf',), usetex=False, rcparams={'axes.
 ↪facecolor': 'white', 'figure.facecolor': 'white'})
pd = nb_setup.setup_pandas(escape_latex=True)
```

[3]:
```
cliente = distributed.Client(threads_per_worker=1)
```

Antes de começarmos, é importante mencionar que buscou-se aqui, sempre que possível, utilizar funções `numba`-jitted de forma a compilar as funções redigidas em `python` em código de máquina. Dessa forma, busca-se maximizar o desempenho da implementação desenvolvida.

1.2 Funções de checagem

Uma vez configurado o ambiente em que este notebook será executado, é necessário definir algumas funções de checagem que serão utilizadas mais adiante:

[4]:
```
def verifica_comprimento_igual_a(**parametros):
    numero_de_parametros = len(parametros.keys())

    if numero_de_parametros != 2:
        raise ValueError(
```

```
            f"Apenas um parâmetro pode ser passado para esta função. Foram␣
↪recebidos " f"{numero_de_parametros}."
        )

    parametro, outro_parametro = parametros.keys()

    valor, descricao = parametros[parametro]
    outro_valor, outra_descricao = parametros[outro_parametro]

    if outro_valor is not None and len(valor) != len(outro_valor):
        raise ValueError(
            f"O {descricao} {parametro} precisa ter um comprimento igual ao␣
↪{outra_descricao} " f"{outro_parametro}."
        )
```

[5]:
```
def verifica_comprimento_menor_ou_igual_a(**parametros):
    numero_de_parametros = len(parametros.keys())

    if numero_de_parametros != 2:
        raise ValueError(
            f"Apenas um parâmetro pode ser passado para esta função. Foram␣
↪recebidos " f"{numero_de_parametros}."
        )

    parametro, outro_parametro = parametros.keys()

    valor, descricao = parametros[parametro]
    outro_valor, outra_descricao = parametros[outro_parametro]

    if outro_valor is not None and len(valor) > len(outro_valor):
        raise ValueError(
            f"O {descricao} {parametro} precisa ter um comprimento, no máximo,␣
↪igual ao {outra_descricao} "
            f"{outro_parametro}."
        )
```

[6]:
```
def verifica_dtype(**parametro_dict):
    numero_de_parametros = len(parametro_dict.keys())

    if numero_de_parametros != 1:
        raise ValueError(
            f"Apenas um parâmetro pode ser passado para esta função. Foram␣
↪recebidos " f"{numero_de_parametros}."
        )

    parametro = list(parametro_dict.keys())[0]
```

```
    valor, descricao, dtype = parametro_dict[parametro]

    if dtype == np.int_ and valor.dtype != dtype:
        if valor.dtype == np.float_:
            return valor.astype(np.int_)
        else:
            raise TypeError(
                f"O {descricao} {parametro} precisa ser um numpy array com␣
↪atributo dtype igual a "
                f"{dtype}. O dtype do numpy array recebido é {valor.dtype}."
            )

    if valor.dtype != dtype:
        raise TypeError(
            f"O {descricao} {parametro} precisa ser um numpy array com atributo␣
↪dtype igual a {dtype}. "
            f"O dtype do numpy array recebido é {valor.dtype}."
        )
    else:
        return valor
```

[7]:
```
def verifica_nao_negatividade(**parametros):
    for parametro in parametros.keys():
        valor, descricao = parametros[parametro]

        if valor < 0:
            raise ValueError(f"O {descricao} {parametro} precisa receber um␣
↪número não-negativo.")
```

[8]:
```
def verifica_ndim(**parametros):
    for parametro in parametros.keys():
        valor, descricao, ndim = parametros[parametro]

        if valor.ndim != ndim:
            raise ValueError(f"O o atributo ndim do {descricao} {parametro}␣
↪precisa ser igual a {ndim}.")
```

[9]:
```
def verifica_tipo(**parametro_dict):
    numero_de_parametros = len(parametro_dict.keys())

    if numero_de_parametros != 1:
        raise ValueError(
            f"Apenas um parâmetro pode ser passado para esta função. Foram␣
↪recebidos " f"{numero_de_parametros}."
        )

    parametro = list(parametro_dict.keys())[0]
```

```
    valor, descricao, tipos = parametro_dict[parametro]

    if tipos == t.SupportsFloat:
        if not isinstance(valor, tipos):
            raise TypeError(
                f"O {descricao} {parametro} precisa receber um número de ponto␣
↪flutuante ou um objeto que "
                f"possa ser convertido para tal."
            )
        else:
            return float(valor)

    if tipos == t.SupportsInt:
        if not isinstance(valor, tipos):
            raise TypeError(
                f"O {descricao} {parametro} precisa receber um número inteiro ou␣
↪um objeto que possa ser "
                f"convertido para tal."
            )
        else:
            return int(valor)

    if tipos == np.ndarray:
        if not isinstance(valor, np.ndarray):
            if not isinstance(valor, (list, tuple)):
                raise TypeError(
                    f"O {descricao} {parametro} precisa receber um array numpy␣
↪ou um objeto que possa ser "
                    f"convertido para tal."
                )
            else:
                return np.array(valor)

    if tipos == bool:
        if not isinstance(valor, bool):
            if isinstance(valor, np.bool_):
                return bool(valor)
            else:
                raise TypeError(
                    f"O {descricao} {parametro} precisa receber um objeto␣
↪booleano ou um objeto que possa "
                    f"ser convertido para tal."
                )
        else:
            return valor

    if not isinstance(valor, tipos):
```

```
        raise TypeError(f"O {descricao} {parametro} precisa receber um objeto de␣
↪classe {tipos} ou que herde dela.")
    else:
        return valor
```

2 Modelagem do algoritmo, da função-objetivo e das soluções

Em aderência ao estipulado pelo comando do trabalho, o algoritmo *K-Means* foi escolhido como o clusterizador para a resolução do problema dos aeroportos. Antes de definirmos a classe `KMedias`, no entanto, algumas funções devem ser definidas para auxiliar no desempenhar das atribuições do algoritmo. Embora estas funções sejam sabidamente já implementadas na forma de métodos e funções do módulo `numpy`, elas não foram implementadas em código compilável pelo módulo `numba`, o que faz com que o processo de tradução do módulo `llvmlite` falhe por não reconhecer a função ou método em questão. Dessa forma, as implementações relevantes e ausentes foram feitas manualmente, também utilizando o módulo `numba`.

2.1 Funções auxiliares

[10]:
```
@nb.jit(nopython=True, parallel=True)
def arredonda_matriz(matriz, casas_decimais):
    vetor = matriz.copy().ravel()
    comprimento = vetor.shape[0]

    for i in nb.prange(comprimento):
        vetor[i] = np.round_(vetor[i], casas_decimais)

    matriz_arredondada = vetor.reshape(matriz.shape)

    return matriz_arredondada
```

[11]:
```
@nb.jit(nopython=True, parallel=True)
def calcula_distancia_entre_grupos(grupo_um, grupo_dois, tira_raiz=True):
    comprimento_um, comprimento_dois = grupo_um.shape[0], grupo_dois.shape[0]
    distancias = np.empty(shape=(comprimento_um, comprimento_dois))

    for i_um in nb.prange(comprimento_um):
        for i_dois in nb.prange(comprimento_dois):
            distancias[i_um, i_dois] = calcula_distancia_entre_observacoes(
                grupo_um[i_um, :], grupo_dois[i_dois, :], tira_raiz
            )

    return distancias
```

[12]:
```
@nb.jit(nopython=True, parallel=True)
def calcula_distancia_entre_observacoes(vetor_um, vetor_dois, tira_raiz):
    comprimento = vetor_um.shape[0]
```

```
    diferencas = np.empty(shape=comprimento)

    for i in nb.prange(comprimento):
        diferencas[i] = np.square(vetor_um[i] - vetor_dois[i])

    distancia = np.sum(diferencas)

    if tira_raiz is True:
        distancia = np.sqrt(distancia)

    return distancia
```

[13]:
```
@nb.jit(nopython=True)
def verifica_igualdade_aproximada_entre_grupos(grupo_um, grupo_dois,␣
 ↪casas_decimais):
    grupo_um, grupo_dois = arredonda_matriz(grupo_um, casas_decimais),␣
 ↪arredonda_matriz(grupo_dois, casas_decimais)
    ha_igualdade = bool((grupo_um == grupo_dois).all())

    return ha_igualdade
```

[14]:
```
@nb.jit(nopython=True, parallel=True)
def tira_media_das_colunas(dados):
    dimensionalidade = dados.shape[1]
    media_das_colunas_dos_dados = np.empty(shape=dimensionalidade)

    for f in nb.prange(dimensionalidade):
        media_das_colunas_dos_dados[f] = dados[:, f].mean()

    return media_das_colunas_dos_dados
```

2.2 Funções auxiliares relacionadas ao K-Means

Uma vez definidas as funções auxiliares de propósito geral, definamos agora as funções auxiliares relacionadas à classe `KMedias`.

[15]:
```
@nb.jit(nopython=True, parallel=True)
def calcula_erro_da_solucao(dados, centroides):
    rotulos, numero_de_centroides = rotula_dados(dados, centroides), centroides.
 ↪shape[0]
    erro_dos_agrupamentos = np.empty(numero_de_centroides)

    for rotulo in nb.prange(numero_de_centroides):
        membros_do_agrupamento = np.where(rotulos == rotulo)[0]

        if membros_do_agrupamento.shape[0] == 0:
            return np.inf
```

```
        else:
            centroide = centroides[rotulo, :].reshape(1, -1)
            membros = dados[membros_do_agrupamento, :]
            erro_dos_agrupamentos[rotulo] = calcula_distancia_entre_grupos(
                centroide, membros, tira_raiz=False
            ).sum()

    erro_total = erro_dos_agrupamentos.sum()

    return erro_total
```

[16]:
```
@nb.jit(nopython=True, parallel=True)
def centraliza_centroides(dados, centroides, centroides_fixos, rotulos):
    numero_de_centroides, dimensionalidade = centroides.shape
    rotulos_unicos = np.arange(numero_de_centroides)[np.
 ↪logical_not(centroides_fixos)]
    numero_de_rotulos = rotulos_unicos.shape[0]
    centroides_centralizados = np.empty((numero_de_centroides, dimensionalidade))
    centroides_centralizados[centroides_fixos, :] = centroides[centroides_fixos,␣
 ↪:]

    for r in nb.prange(numero_de_rotulos):
        rotulo = rotulos_unicos[r]
        membros_do_agrupamento = np.where(rotulos == rotulo)[0]
        agrupamento = dados[membros_do_agrupamento, :]
        centroides_centralizados[rotulo, :] = tira_media_das_colunas(agrupamento)

    return centroides_centralizados
```

[17]:
```
@nb.jit(nopython=True)
def rotula_dados(dados, centroides):
    distancias = calcula_distancia_entre_grupos(dados, centroides)
    numero_de_observacoes = dados.shape[0]
    rotulos = np.empty(numero_de_observacoes, dtype=np.int_)

    for d in nb.prange(numero_de_observacoes):
        rotulos[d] = distancias[d].argmin()

    return rotulos
```

[18]:
```
@nb.jit(nopython=True)
def gera_centroides(dados, numero_de_centroides):
    comprimento, dimensionalidade = dados.shape
    indices = np.arange(comprimento, dtype=np.int_)
    indices_aleatorios = np.random.choice(indices, size=numero_de_centroides,␣
 ↪replace=False)
    centroides = dados[indices_aleatorios].copy()
```

```
    return centroides
```

[19]:
```
@nb.jit(nopython=True)
def roda_k_means(dados, centroides, centroides_fixos, casas_decimais=4,␣
↪numero_maximo_de_iteracoes=1000):
    iteracao = 0

    while iteracao < numero_maximo_de_iteracoes:
        rotulos = rotula_dados(dados, centroides)
        centroides_centralizados = centraliza_centroides(dados, centroides,␣
↪centroides_fixos, rotulos)
        ha_igualdade = verifica_igualdade_aproximada_entre_grupos(
            centroides, centroides_centralizados, casas_decimais
        )
        centroides = centroides_centralizados

        if ha_igualdade is True:
            break
        else:
            iteracao += 1

    return centroides
```

2.2.1 A classe `KMedias`

Uma vez definidas todas as funções necessárias, chegamos à definição da classe `KMedias`. Esta implementação do *K-Means* foi personalizada para suportar a especificação de centroides fixos, i.e., centroides que não são atualizados no decorrer das iterações, mas que compõem a solução final.

Em suma, esta classe é um *wrapper* para as funções `numba` supra-implementadas que serve para armazenar informações e realizar verificações nas entradas fornecidas aos seus métodos e atributos.

[20]:
```
class KMedias:
    def __init__(self, *, numero_de_centroides):
        self.numero_de_centroides = numero_de_centroides
        self.__centroides, self.__centroides_fixos, self.__dados = None, None,␣
↪None

    @property
    def numero_de_centroides(self):
        return self.__numero_de_centroides

    @numero_de_centroides.setter
    def numero_de_centroides(self, novo_numero_de_centroides):
        novo_numero_de_centroides = verifica_tipo(
```

```
            numero_de_centroides=(novo_numero_de_centroides, "atributo", t.
↪SupportsInt)
        )

      ␣
↪verifica_nao_negatividade(numero_de_centroides=(novo_numero_de_centroides,␣
↪"atributo"))

        self.__numero_de_centroides = novo_numero_de_centroides

    @property
    def centroides(self):
        return self.__centroides

    @centroides.setter
    def centroides(self, novos_centroides):
        if novos_centroides is None:
            novos_centroides = gera_centroides(
                dados=self.dados, numero_de_centroides=self.numero_de_centroides
            )
            self.centroides_fixos = novos_centroides.shape[0] * [False]
        elif isinstance(novos_centroides, np.ndarray):
            novos_centroides = verifica_tipo(centroides=(novos_centroides,␣
↪"atributo", np.ndarray))

            verifica_ndim(centroides=(novos_centroides, "atributo", 2))
            verifica_comprimento_menor_ou_igual_a(
                centroides=(novos_centroides, "atributo"), dados=(self.dados,␣
↪"atributo")
            )

            centroides_faltantes = self.numero_de_centroides - novos_centroides.
↪shape[0]

            if centroides_faltantes > 0:
                centroides_complementares = gera_centroides(
                    dados=self.dados, numero_de_centroides=centroides_faltantes
                )
                self.centroides_fixos = novos_centroides.shape[0] * [True] +␣
↪centroides_faltantes * [False]
                novos_centroides = np.concatenate((novos_centroides,␣
↪centroides_complementares), axis=0)
            elif centroides_faltantes < 0:
                novos_centroides = novos_centroides[:centroides_faltantes]
                self.centroides_fixos = novos_centroides.shape[0] * [False]
```

```
        self.__centroides = novos_centroides

    @property
    def centroides_fixos(self):
        return self.__centroides_fixos

    @centroides_fixos.setter
    def centroides_fixos(self, novos_centroides_fixos):
        novos_centroides_fixos = verifica_tipo(
            centroides_fixos=(novos_centroides_fixos, "atributo", np.ndarray)
        )

        verifica_ndim(centroides_fixos=(novos_centroides_fixos, "atributo", 1))
        verifica_dtype(centroides_fixos=(novos_centroides_fixos, "atributo", np.
↪bool_))
        verifica_comprimento_igual_a(
            centroides_fixos=(novos_centroides_fixos, "atributo"),␣
↪centroides=(self.centroides, "atributo")
        )

        self.__centroides_fixos = novos_centroides_fixos

    @property
    def dados(self):
        return self.__dados

    @dados.setter
    def dados(self, novos_dados):
        novos_dados = verifica_tipo(dados=(novos_dados, "atributo", np.ndarray))

        verifica_ndim(dados=(novos_dados, "atributo", 2))

        self.__dados = novos_dados

    @property
    def rotulos(self):
        return rotula_dados(dados=self.dados, centroides=self.centroides)

    def clusteriza_dados(self, *, dados, centroides_fixos,␣
↪numero_de_execucoes=1000):
        melhor_solucao, erro_da_melhor_solucao = None, None
        data_frame = pd.DataFrame(index=range(numero_de_execucoes),␣
↪columns=["Solução", "Erro"])

        for iteracao in range(numero_de_execucoes):
            print(f"Execução {iteracao}...", end="\r")
```

```
            solucao = self._clusteriza_dados(dados=dados,␣
↪centroides_fixos=centroides_fixos)
            erro_da_solucao = calcula_erro_da_solucao(dados, solucao)

            data_frame.iloc[iteracao, :] = np.array([solucao, erro_da_solucao],␣
↪dtype=object)

            if melhor_solucao is None or erro_da_solucao <␣
↪erro_da_melhor_solucao:
                melhor_solucao, erro_da_melhor_solucao = solucao, erro_da_solucao

        data_frame = data_frame.infer_objects().sort_values(by="Erro")

        return data_frame

    def _clusteriza_dados(self, *, dados, centroides_fixos):
        self.dados = dados
        self.centroides = centroides_fixos

        self.centroides = roda_k_means(
            dados=dados, centroides=self.centroides, centroides_fixos=self.
↪centroides_fixos
        )

        return self.centroides
```

Pode-se notar que o método privado `KMedias._clusteriza_dados()`, que é chamado pelo método público `KMedias.clusteriza_dados()`, se vale da função `numba`-jitted `roda_k_means()`, que por sua vez chama as demais funções `numba`-jitted para desempenhar a rotina principal do *K-Means*.

2.3 A função-objetivo

A função-objetivo a ser minimizada, doravante referida como erro, foi definida na função `numba`-jitted `calcula_erro_da_solucao()`. Ela segue à risca a função estipulada no trabalho, que é a que segue:

$$f(X, C) = \sum_{k=1}^{K} \sum_{\vec{x} \in \vec{C}_i} \left\| \vec{x} - \vec{C}_i \right\|^2$$

Nela, X corresponde ao conjunto de dados, enquanto C corresponde ao conjunto de K centroides. Busca-se, por meio dela, minimizar a distância das observações em relação ao respectivo centroide ao qual elas pertencem.

2.4 As soluções

O algoritmo retorna um conjunto de K centroides (onde K é definido pelo usuário no parâmetro `numero_de_centroides` do construtor da classe `KMedias`) capaz de particionar o conjunto de dados

fornecido em clusters.

3 O problema

A partir do mencionado no arquivo `.pdf` contendo a descrição do trabalho, tem-se um cenário hipotético em que 3 novos aeroportos serão construídos na cidade Belém, localizada no estado Pará. Os seguintes distritos foram elencados para consideração:

- Distrito administrativo de Belém - Centro (DABEL)
- Distrito administrativo do Entroncamento (DAENT)
- Distrito administrativo do Guamá (DAGUA)
- Distrito administrativo do Benguí – Nova Belém (DABEN)
- Distrito administrativo da Sacramenta (DASAC)

A solução deve levar em conta o já existente Aeroporto Internacional Val-de-Cans que atende a região atualmente. A fim de descobrir soluções para esse problema, os seguintes componentes fazem-se necessários:

- **Um algoritmo de clusterização**, que deve ser executado várias vezes para assegurar que a melhor solução possível será encontrada;
- **Um conjunto de dados**, que deve descrever o centro dos bairros dos distritos considerados; e
- **As coordenadas do Aeroporto Internacional Val-de-Cans**, que deve ser considerado como um centroide fixo na solução a ser descoberta.

Em adição, os seguintes requisitos precisam ser atendidos:

- Cada novo aeroporto deve atender ao máximo de bairros próximos a ele; e
- Cada novo aeroporto deve atender a pelo menos um bairro.

4 O conjunto de dados

Antes de explorarmos o conjunto de dados criado, definamos uma função para a sua visualização.

[21]:
```
def gera_grafico_de_dispersao(*, nomes, dados, centroides=None):
    if centroides is None:
        rotulos = None
    else:
        rotulos = rotula_dados(dados=dados, centroides=centroides)

    figura, eixo = plt.subplots(1, 1, squeeze=True, figsize=(11, 11))
    figura.tight_layout()

    sns.scatterplot(
        x=dados[:, 0],
        y=dados[:, 1],
        palette=sns.color_palette("bright", np.unique(rotulos).size),
        hue=rotulos,
        ax=eixo,
        legend=False
```

```
    )

    for n in range(len(nomes)):
        eixo.annotate(nomes[n], dados[n] + .0005, fontsize=8)

    eixo.scatter(x=centroides[:, 0], y=centroides[:, 1], c="k", marker="s")

    return figura, eixo
```

Em seguimento às estipulações do problema proposto, os centros dos bairros dos distritos sob análise foram obtidos e armazenados no arquivo `coordenadas_bairros.csv`. Vamos carregar e visualizar um trecho de seu conteúdo.

[22]:
```
data_frame_aeroporto = pd.read_csv("dados/coordenadas_aeroporto.csv",␣
↪index_col=0)
data_frame_aeroporto
```

[22]:

Aeroporto	Coordenadas
Val-de-Cans	-1.382092, -48.477506

[23]:
```
data_frame_aeroporto.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 1 entries, Val-de-Cans to Val-de-Cans
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Coordenadas  1 non-null      object
dtypes: object(1)
memory usage: 16.0+ bytes
```

[24]:
```
data_frame_bairros = pd.read_csv("dados/coordenadas_bairros_final.csv",␣
↪index_col=0)
data_frame_bairros.head(5)
```

[24]:

Bairro	Coordenadas
Águas Lindas	-1.4106819, -48.3925318
Aurá	-1.4255977, -48.3847762
Barreiro	-1.4130584, -48.4836502
Batista Campos	-1.4609694, -48.4891385
Benguí	-1.3755029, -48.4561909

[25]:
```
data_frame_bairros.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Index: 40 entries, Águas Lindas to Val-de-Cães
Data columns (total 1 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Coordenadas  40 non-null     object
dtypes: object(1)
memory usage: 640.0+ bytes
```

4.1 Pré-processamento

Como se pode ver, as coordenadas do aeroporto e dos bairros são descritas por objetos `str` e de forma conjunta, impedindo que nós visualizemos o conjunto de dados e o clusterizemos. Em adição, os dados estão em um objeto `pandas.DataFrame`, quando a implementação realizada na classe `KMedias` suporta apenas objetos `numpy.ndarray`.

Dessa forma, precisamos

- separar as coordenadas em `Latitude` e `Longitude`; e
- converter os objetos `pandas.DataFrame` em objetos `numpy.ndarray`.

Comecemos pela definição das colunas `Latitude` e `Longitude`.

[26]:
```
for data_frame in [data_frame_aeroporto, data_frame_bairros]:
    data_frame["Longitude"] = data_frame["Coordenadas"].apply(lambda x: x.
 ↪split(", ")[1])
    data_frame["Latitude"] = data_frame["Coordenadas"].apply(lambda x: x.
 ↪split(", ")[0])

    data_frame.drop("Coordenadas", axis=1, inplace=True)

    data_frame["Longitude"] = pd.to_numeric(data_frame["Longitude"])
    data_frame["Latitude"] = pd.to_numeric(data_frame["Latitude"])
```

[27]:
```
data_frame_aeroporto
```

[27]:

Aeroporto	Longitude	Latitude
Val-de-Cans	-48.477506	-1.382092

[28]:
```
data_frame_bairros.head(5)
```

[28]:

Bairro	Longitude	Latitude
Águas Lindas	-48.392532	-1.410682
Aurá	-48.384776	-1.425598
Barreiro	-48.483650	-1.413058
Batista Campos	-48.489139	-1.460969
Benguí	-48.456191	-1.375503

Por fim, convertamos os objetos `pandas.DataFrame` em objetos `numpy.ndarray`.

[29]:
```
aeroporto, bairros = data_frame_aeroporto.to_numpy(), data_frame_bairros.
↪to_numpy()
```

[30]:
```
aeroporto
```

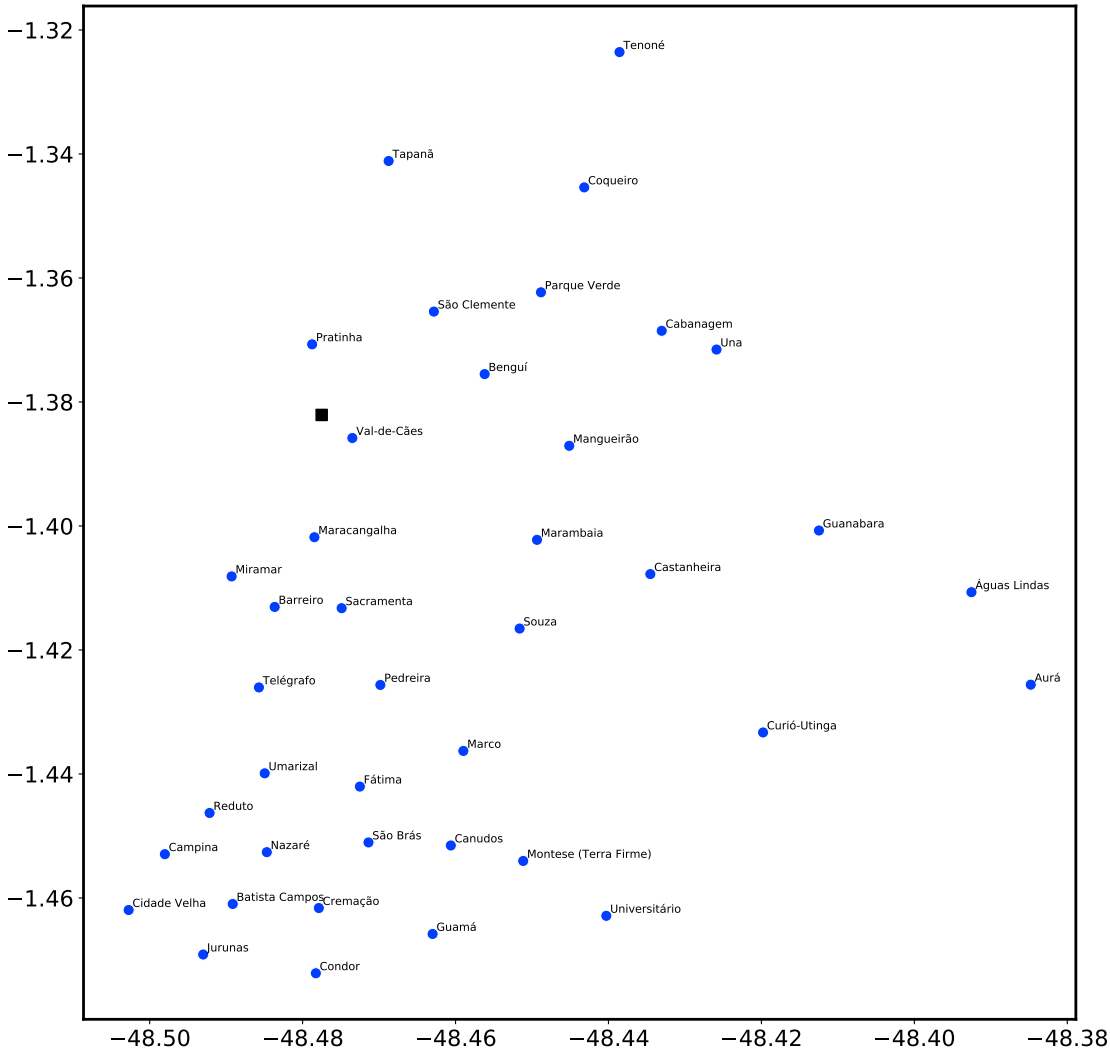
[30]:
```
array([[-48.477506,  -1.382092]])
```

[31]:
```
bairros[:5, :]
```

[31]:
```
array([[-48.3925318,  -1.4106819],
       [-48.3847762,  -1.4255977],
       [-48.4836502,  -1.4130584],
       [-48.4891385,  -1.4609694],
       [-48.4561909,  -1.3755029]])
```

Agora, podemos visualizar adequadamente o conjunto de dados à mão, bem como a localização do já existente aeroporto.

[32]:
```
figura, eixo = gera_grafico_de_dispersao(nomes=data_frame_bairros.index,␣
↪dados=bairros, centroides=aeroporto)
```



4.2 Processamento

Uma vez pré-processado o conjunto de dados, temos as variáveis `aeroporto`, que contém o centroide fixo representativo do Aeroporto Internacional Val-de-Cans, e `bairros`, que contém as coordenadas de todos os bairros relevantes. Já podemos seguir para o instanciamento e uso da classe `KMedias`; contudo, definamos antes algumas funções auxiliares para execução dos testes e exposição dos resultados.

[33]:
```
def clusteriza(*, dados, numero_de_centroides, centroides_fixos):
    k_medias = KMedias(numero_de_centroides=numero_de_centroides)
    centroides = k_medias._clusteriza_dados(dados=dados,␣
 ↪centroides_fixos=centroides_fixos)
    erro = calcula_erro_da_solucao(dados, centroides)
```

```
    return centroides, erro
```

[34]:
```
def clusteriza_em_paralelo(*, dados, numero_de_centroides, centroides_fixos,␣
↪cliente, numero_de_execucoes):
    solucoes = pd.DataFrame(index=range(numero_de_execucoes),␣
↪columns=["Solução", "Erro"])
    futuros = [cliente.submit(
        clusteriza, dados=dados, numero_de_centroides=numero_de_centroides,␣
↪centroides_fixos=centroides_fixos, pure=False
    ) for _ in range(numero_de_execucoes)]

    for futuro in distributed.as_completed(futuros):
        indice_do_futuro = futuros.index(futuro)
        solucoes.iloc[indice_do_futuro, :] = np.array(futuro.result(),␣
↪dtype=object)

    solucoes = solucoes.sort_values(by="Erro", ignore_index=True).infer_objects()

    return solucoes
```

[35]:
```
def gera_graficos_de_dispersao(*, dados, melhor_solucao, pior_solucao):
    rotulos_da_melhor_solucao = rotula_dados(dados=dados,␣
↪centroides=melhor_solucao)
    rotulos_da_pior_solucao = rotula_dados(dados=dados, centroides=pior_solucao)

    figura, eixos = plt.subplots(1, 2, squeeze=True, sharey=True, figsize=(20,␣
↪10))
    figura.tight_layout()

    sns.scatterplot(
        x=dados[:, 0],
        y=dados[:, 1],
        palette=sns.color_palette("bright", np.unique(rotulos_da_melhor_solucao).
↪size),
        hue=rotulos_da_melhor_solucao,
        ax=eixos[0],
        legend=False
    )

    eixos[0].scatter(x=melhor_solucao[:, 0], y=melhor_solucao[:, 1], c="k",␣
↪marker="s")
    eixos[0].set_title("Melhor solução")

    sns.scatterplot(
        x=dados[:, 0],
        y=dados[:, 1],
```

```
        palette=sns.color_palette("bright", np.unique(rotulos_da_pior_solucao).
↪size),
        hue=rotulos_da_pior_solucao,
        ax=eixos[1],
        legend=False
    )

    eixos[1].scatter(x=pior_solucao[:, 0], y=pior_solucao[:, 1], c="k",␣
↪marker="s")
    eixos[1].set_title("Pior solução")

    return figura, eixos
```

Isto feito, prossigamos agora para a rotina de clusterização em si. A fim de explorar bem o espaço de busca, executemos o `KMedias` 1000 vezes.

[36]:
```
solucoes = clusteriza_em_paralelo(
    dados=bairros, numero_de_centroides=4, centroides_fixos=aeroporto,␣
↪cliente=cliente, numero_de_execucoes=1000
)
```

Observação: caso você queira ver o `dask.Client` executando as 1000 tarefas em paralelo, clique aqui enquanto o notebook gera as soluções.

Agora, visualizemos as vinte melhores soluções.

[37]:
```
solucoes.head(20)
```

[37]:

	Solução	Erro
0	[[-48.477506, -1.382092], [-48.47635086006692,...	0.021012
1	[[-48.477506, -1.382092], [-48.47635086006692,...	0.021012
2	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
3	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
4	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
5	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
6	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
7	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
8	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
9	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
10	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
11	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
12	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
13	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
14	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
15	[[-48.477506, -1.382092], [-48.47635086006692,...	0.021012
16	[[-48.477506, -1.382092], [-48.420721699999994...	0.021012
17	[[-48.477506, -1.382092], [-48.47635086006692,...	0.021012
18	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012
19	[[-48.477506, -1.382092], [-48.44692907777778,...	0.021012

E agora, as vinte piores.

[38]:
```
solucoes.tail(20)
```

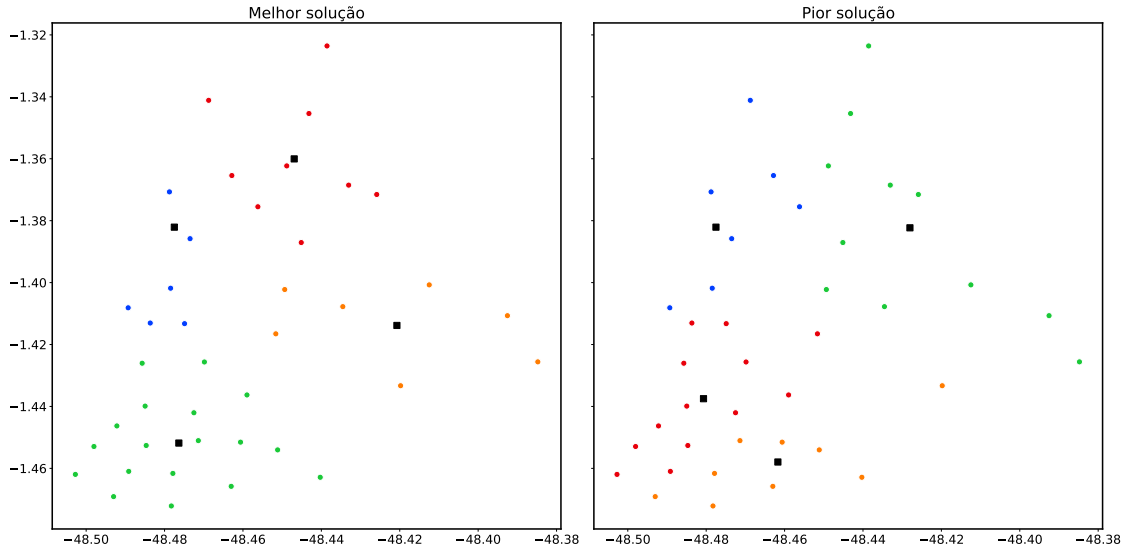
[38]:

	Solução	Erro
980	[[-48.477506, -1.382092], [-48.48243335, -1.45...	0.028972
981	[[-48.477506, -1.382092], [-48.48243335, -1.45...	0.028972
982	[[-48.477506, -1.382092], [-48.48243335, -1.45...	0.028972
983	[[-48.477506, -1.382092], [-48.446939735150586...	0.028972
984	[[-48.477506, -1.382092], [-48.48017982500001,...	0.029045
985	[[-48.477506, -1.382092], [-48.42399934530117,...	0.029045
986	[[-48.477506, -1.382092], [-48.430566174999996...	0.029045
987	[[-48.477506, -1.382092], [-48.42399934530117,...	0.029045
988	[[-48.477506, -1.382092], [-48.42399934530117,...	0.029045
989	[[-48.477506, -1.382092], [-48.42920324615385,...	0.029274
990	[[-48.477506, -1.382092], [-48.42920324615385,...	0.029439
991	[[-48.477506, -1.382092], [-48.469413364578294...	0.029622
992	[[-48.477506, -1.382092], [-48.461464075689406...	0.029690
993	[[-48.477506, -1.382092], [-48.48716275454546,...	0.029700
994	[[-48.477506, -1.382092], [-48.41471361428572,...	0.029700
995	[[-48.477506, -1.382092], [-48.42802176363636,...	0.029883
996	[[-48.477506, -1.382092], [-48.46395274374588,...	0.029883
997	[[-48.477506, -1.382092], [-48.42802176363636,...	0.029883
998	[[-48.477506, -1.382092], [-48.42802176363636,...	0.029969
999	[[-48.477506, -1.382092], [-48.420524422222215...	0.030032

Temos acima um objeto `pandas.DataFrame` contendo as soluções e os erros delas para cada uma das 1000 execuções que realizamos. Vejamos a melhor e a pior solução dentre as encontradas nas 1000 execuções.

[39]:
```
melhor_solucao, pior_solucao = solucoes.iloc[0, :], solucoes.iloc[-1, :]
```

[40]:
```
figura, eixos = gera_graficos_de_dispersao(dados=bairros,␣
↪melhor_solucao=melhor_solucao[0], pior_solucao=pior_solucao[0])
```



Observe que o o centroide do cluster azul corresponde ao aeroporto já existente, enquanto que os demais centroides representam os aeroportos a serem construídos.

[41]:
```
print(f"Coordenadas da melhor solução:\n"
      f"{melhor_solucao[0]}\n\n"
      f"Coordenadas da pior solução:\n"
      f"{pior_solucao[0]}")
```

```
Coordenadas da melhor solução:
[[-48.477506    -1.382092  ]
 [-48.47635086  -1.45182401]
 [-48.44692908  -1.36004946]
 [-48.4207217   -1.41383774]]

Coordenadas da pior solução:
[[-48.477506    -1.382092  ]
 [-48.42052442  -1.38745556]
 [-48.47635086  -1.45182401]
 [-48.45433452  -1.39281943]]
```

Observe também como a primeira coordenada não muda entre a melhor e a pior solução. Esta é a

coordenada do já existente Aeroporto Internacional Val-de-Cans.

[42]:
```
print(f"Erro da melhor solução: {melhor_solucao[1]}\n"
      f"Erro da pior solução:   {pior_solucao[1]}\n"
      f"Diferença dos erros:    {pior_solucao[1] - melhor_solucao[1]}")
```

```
Erro da melhor solução: 0.021012302464794186
Erro da pior solução:   0.030032458117698592
Diferença dos erros:    0.009020155652904406
```

Embora, no papel, a diferença do erro entre a melhor e a pior solução não seja tão grande, na prática é possível visualizar com razoável notoriedade como os centroides na melhor solução estão melhor posicionados em relação àqueles da pior solução.

Abaixo encontram-se os rótulos para os bairros na melhor solução. Observe que os rótulos estão seguindo a ordem dos centroides apresentados acima, e.g., um bairro com o rótulo 3 na melhor solução significa que o bairro em questão pertence ao terceiro centroide, ou à terceira linha da matriz exposta acima para a melhor solução.

[43]:
```
rotulos_da_melhor_solucao = pd.DataFrame(
    rotula_dados(bairros, melhor_solucao[0]), index=data_frame_bairros.index,␣
 ↪columns=["Rótulo"]
)
```

[44]:
```
pd.concat((data_frame_bairros, rotulos_da_melhor_solucao), axis=1)
```

[44]:

Bairro	Longitude	Latitude	Rótulo
Águas Lindas	-48.392532	-1.410682	3
Aurá	-48.384776	-1.425598	3
Barreiro	-48.483650	-1.413058	0
Batista Campos	-48.489139	-1.460969	1
Benguí	-48.456191	-1.375503	2
Cabanagem	-48.433036	-1.368521	2
Campina	-48.498012	-1.452936	1
Canudos	-48.460617	-1.451533	1
Castanheira	-48.434523	-1.407763	3
Cidade Velha	-48.502750	-1.461945	1
Condor	-48.478268	-1.472144	1
Coqueiro	-48.443167	-1.345390	2
Cremação	-48.477876	-1.461625	1
Curió-Utinga	-48.419783	-1.433306	3
Fátima	-48.472512	-1.442036	1
Guamá	-48.462998	-1.465804	1
Guanabara	-48.412475	-1.400733	3
Jurunas	-48.493002	-1.469122	1
Mangueirão	-48.445130	-1.387065	2
Maracangalha	-48.478455	-1.401814	0
Marambaia	-48.449343	-1.402239	3
Marco	-48.458977	-1.436286	1
Miramar	-48.489282	-1.408133	0
Montese (Terra Firme)	-48.451152	-1.454032	1
Nazaré	-48.484681	-1.452611	1
Parque Verde	-48.448829	-1.362309	2
Pedreira	-48.469833	-1.425640	1
Pratinha	-48.478752	-1.370700	0
Reduto	-48.492163	-1.446291	1
Sacramenta	-48.474894	-1.413257	0
São Brás	-48.471383	-1.451043	1
São Clemente	-48.462833	-1.365422	2
Souza	-48.451620	-1.416544	3
Tapanã	-48.468747	-1.341128	2
Telégrafo	-48.485710	-1.426042	1
Tenoné	-48.438557	-1.323566	2
Umarizal	-48.484956	-1.439893	1
Una	-48.425871	-1.371542	2
Universitário	-48.440286	-1.462879	1
Val-de-Cães	-48.473493	-1.385814	0

Por fim, abaixo encontram-se os rótulos para os bairros na pior solução. Observe que os rótulos estão seguindo a ordem dos centroides apresentados acima, e.g., um bairro com o rótulo 3 na pior solução significa que o bairro em questão pertence ao terceiro centroide, ou à terceira linha da matriz exposta acima para a pior solução.

[45]:
```
rotulos_da_pior_solucao = pd.DataFrame(
    rotula_dados(bairros, pior_solucao[0]), index=data_frame_bairros.index,␣
 ↪columns=["Rótulo"]
)
```

[46]:
```
pd.concat((data_frame_bairros, rotulos_da_pior_solucao), axis=1)
```

[46]:

Bairro	Longitude	Latitude	Rótulo
Águas Lindas	-48.392532	-1.410682	1
Aurá	-48.384776	-1.425598	1
Barreiro	-48.483650	-1.413058	0
Batista Campos	-48.489139	-1.460969	2
Benguí	-48.456191	-1.375503	3
Cabanagem	-48.433036	-1.368521	1
Campina	-48.498012	-1.452936	2
Canudos	-48.460617	-1.451533	2
Castanheira	-48.434523	-1.407763	1
Cidade Velha	-48.502750	-1.461945	2
Condor	-48.478268	-1.472144	2
Coqueiro	-48.443167	-1.345390	1
Cremação	-48.477876	-1.461625	2
Curió-Utinga	-48.419783	-1.433306	1
Fátima	-48.472512	-1.442036	2
Guamá	-48.462998	-1.465804	2
Guanabara	-48.412475	-1.400733	1
Jurunas	-48.493002	-1.469122	2
Mangueirão	-48.445130	-1.387065	3
Maracangalha	-48.478455	-1.401814	0
Marambaia	-48.449343	-1.402239	3
Marco	-48.458977	-1.436286	2
Miramar	-48.489282	-1.408133	0
Montese (Terra Firme)	-48.451152	-1.454032	2
Nazaré	-48.484681	-1.452611	2
Parque Verde	-48.448829	-1.362309	3
Pedreira	-48.469833	-1.425640	2
Pratinha	-48.478752	-1.370700	0
Reduto	-48.492163	-1.446291	2
Sacramenta	-48.474894	-1.413257	3
São Brás	-48.471383	-1.451043	2
São Clemente	-48.462833	-1.365422	0
Souza	-48.451620	-1.416544	3
Tapanã	-48.468747	-1.341128	0
Telégrafo	-48.485710	-1.426042	2
Tenoné	-48.438557	-1.323566	1
Umarizal	-48.484956	-1.439893	2
Una	-48.425871	-1.371542	1
Universitário	-48.440286	-1.462879	2
Val-de-Cães	-48.473493	-1.385814	0

5 Considerações finais

Neste trabalho, uma versão customizada do algoritmo *K-Means*, intitulada `KMedias`, foi criada. Esta implementação possui a capacidade adicional de receber especificações para centroides fixos, i.e., centroides que não serão atualizados durante o desenvolvimento da rotina do algoritmo. Em seu desenvolvimento, foram utilizados os módulos `numba` (na forma do decorador `@nb.jit`) e `dask` (na forma da classe `dask.distributed.Client`) sempre que possível para acelerar a execução do algoritmo ao máximo e tirar proveito de todos os núcleos disponíveis pelo computador em que este notebook está sendo executado.

A classe `KMedias` foi utilizada para, ao longo de 1000 execuções, explorar o espaço de busca composto por bairros de alguns distritos da cidade Belém, localizada no estado Pará. Para cada uma das execuções, uma solução foi gerada, e a melhor e a pior solução dentre as encontradas foi analisada. Constatou-se nos resultados uma discrepância considerável entre a melhor e a pior solução, bem como o fato de que, em todas elas, o já existente Aeroporto Internacional Val-de-Cans permaneceu nas mesmas coordenadas, respeitando a especificação de centroide fixo passada ao algoritmo.

Como produto deste trabalho, tem-se uma implementação eficiente do algoritmo *K-Means*. Ela pode ser reutilizada para outros problemas em que a especificação de centroides fixos seja desejável ou requisitada.

Para detalhes adicionais acerca desta implementação, por favor leia o artigo elaborado acerca deste trabalho. Assim como este notebook, ele está disponível em **@ygarasab/kmeans**.