TRABALHO FINAL – IAA015 – Tópicos de Inteligência Artificial

Equipe 16 (https://equipe16-iaa.github.io/)

Github: https://github.com/Equipe16-IAA

Repositório da disciplina:
https://github.com/Equipe16-IAA/IAA015

Ana Beatriz Kindinger
Daniel Victor Andrade
Igor Buess Atala Y Mansour
Marlon Mateus Prudente de Oliveira

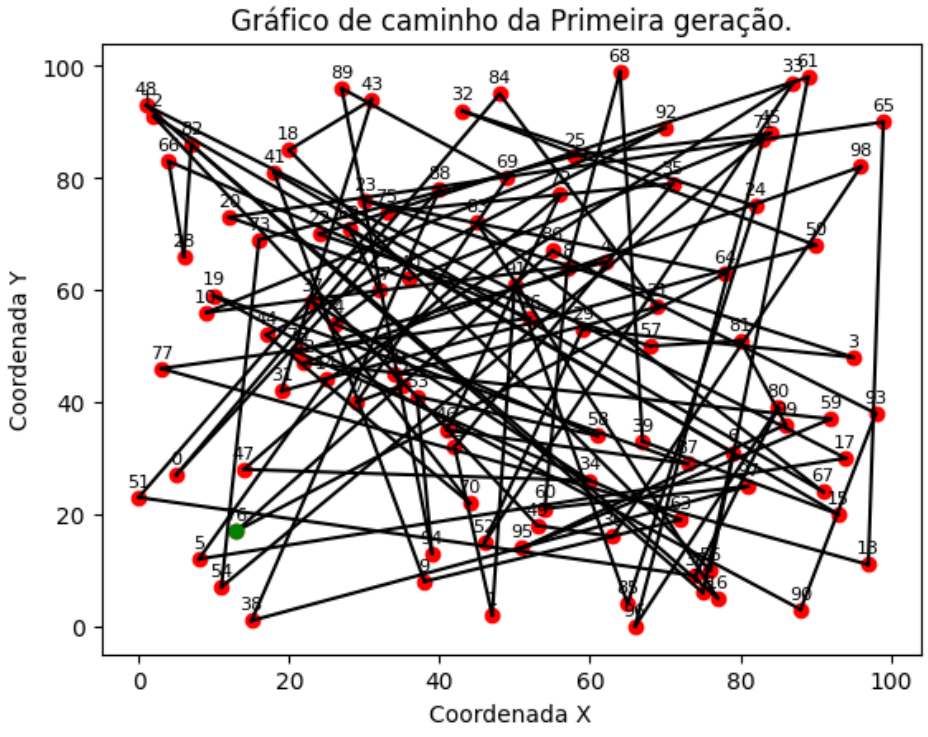
1 – Algoritmo Genético

O Problema do Caixeiro Viajante (PCV) é um problema clássico de otimização combinatória, que busca o menor caminho para visitar um conjunto de cidades. Por ser um problema NP-Completo, sua complexidade aumenta superpolinomialmente com o número de cidades. Neste trabalho, um algoritmo genético foi usado para propor uma solução otimizada em relação a uma solução aleatória, dado que o PCV ainda não possui uma solução ótima para um número muito grande de cidades.

Para o cruzamento, foi utilizada a técnica de Crossover de Ordem (OX) com um ponto de corte, devido à restrição de que os valores das cidades não podem se repetir. Para a mutação, o indivíduo foi dividido em dois segmentos, e os alelos do segundo segmento foram embaralhados.

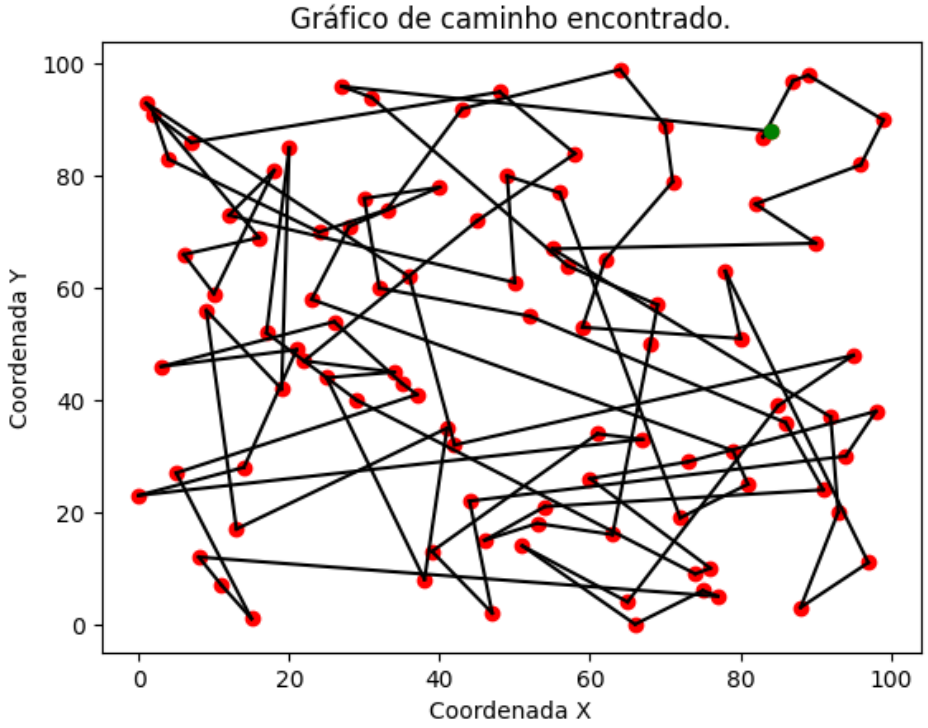
Na população inicial, foram criados 100 indivíduos aleatoriamente, e o indivíduo com o melhor fitness foi selecionado para a segunda geração, implementando o elitismo. Foram observadas 30.000 gerações, com uma taxa de cruzamento de 90% e uma taxa de mutação de 1%.

Melhor indivíduo da primeira geração:



Distância total percorrida: 5569.9 pixels

Melhor indivíduo da última geração:



Distância total percorrida: 2231.6 pixels

Os resultados demonstram a eficácia do AG em reduzir a distância total das rotas. Além da observação da menor distância euclidiana, a plotagem das rotas otimizadas revela uma significativa diminuição no número de cruzamentos entre os caminhos percorridos. Essa redução de cruzamentos sugere uma menor complexidade da rota, o que corrobora a diminuição da distância total percorrida.

2 – Comparando a representação de dois modelos vetoriais

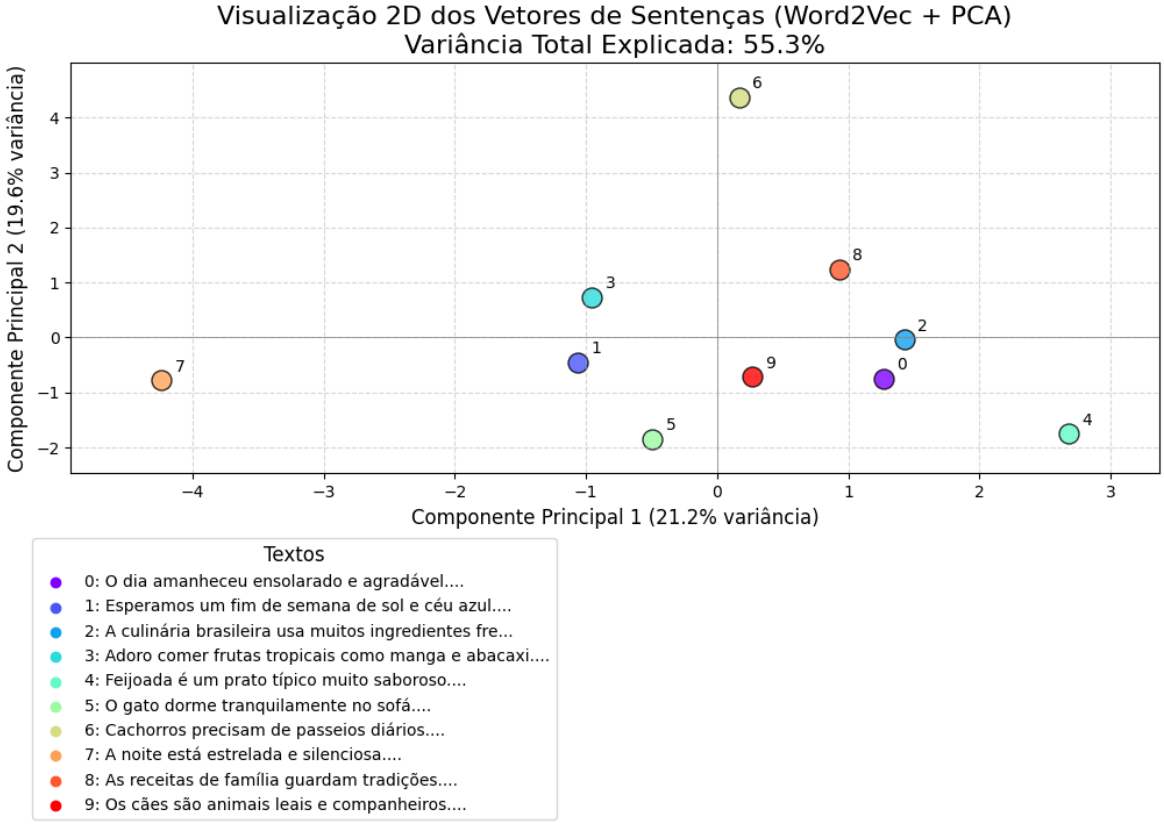
Para visualizar a representação vetorial de textos, selecionamos dez fragmentos curtos, abrangendo temas como clima, culinária e animais, com similaridades intencionais para evidenciar o agrupamento vetorial. O pré-processamento incluiu conversão para minúsculas, remoção de pontuação, números e stopwords, seguido de tokenização.

A representação vetorial foi obtida via Word2Vec, treinado com 100 épocas para capturar nuances semânticas em nosso pequeno corpus. A representação de cada sentença foi calculada pela média dos vetores das palavras, resultando em vetores de alta dimensionalidade.

A Análise de Componentes Principais (PCA) foi aplicada para reduzir a dimensionalidade, facilitando a visualização em gráficos 2D. Os resultados, multiplicados por 100 para clareza visual, demonstram o agrupamento de sentenças semanticamente similares, validando a eficácia da representação vetorial e da PCA na identificação de relações textuais.

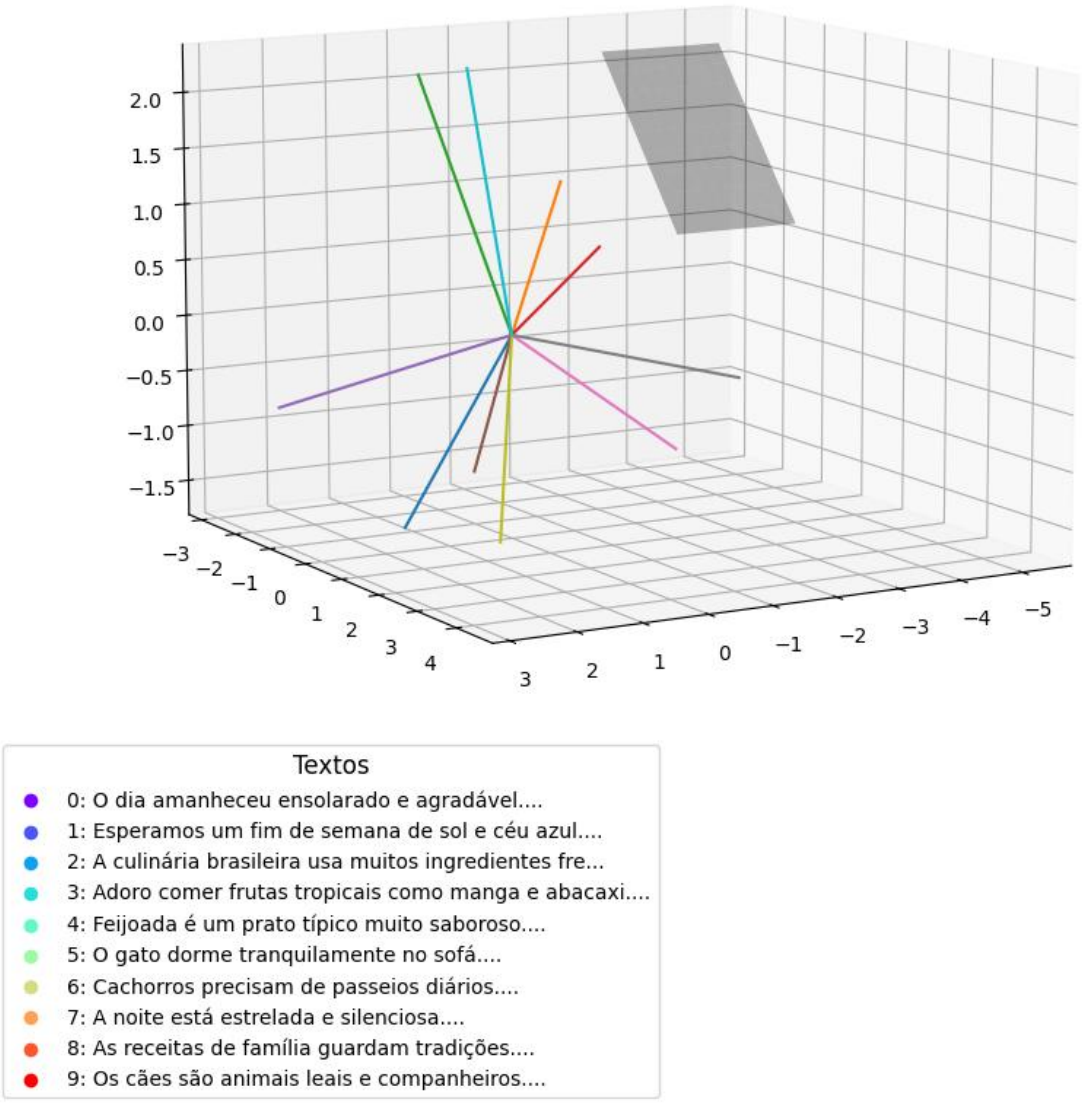
	Frases utilizadas	relações
1	O dia amanheceu ensolarado e agradável.	Clima 1
2	Esperamos um fim de semana de sol e céu azul.	Clima 2 (similar a 1)
3	A culinária brasileira usa muitos ingredientes frescos.	Culinária 1
4	Adoro comer frutas tropicais como manga e abacaxi.	Culinária/frutas 2
5	Feijoada é um prato típico muito saboroso.	Culinária 3 (similar ao 3)
6	O gato dorme tranquilamente no sofá.	Animais 1
7	Cachorros precisam de passeios diários.	Animais 2
8	A noite está estrelada e silenciosa.	Clima/Noite 3 (ligeiramente similar ao 1/2)
9	As receitas de família guardam tradições.	Culinária 4 (similar ao 3/5)
10	Os cães são animais leais e companheiros.	Animais 3 (similar ao 7)

Visualização dos vetores

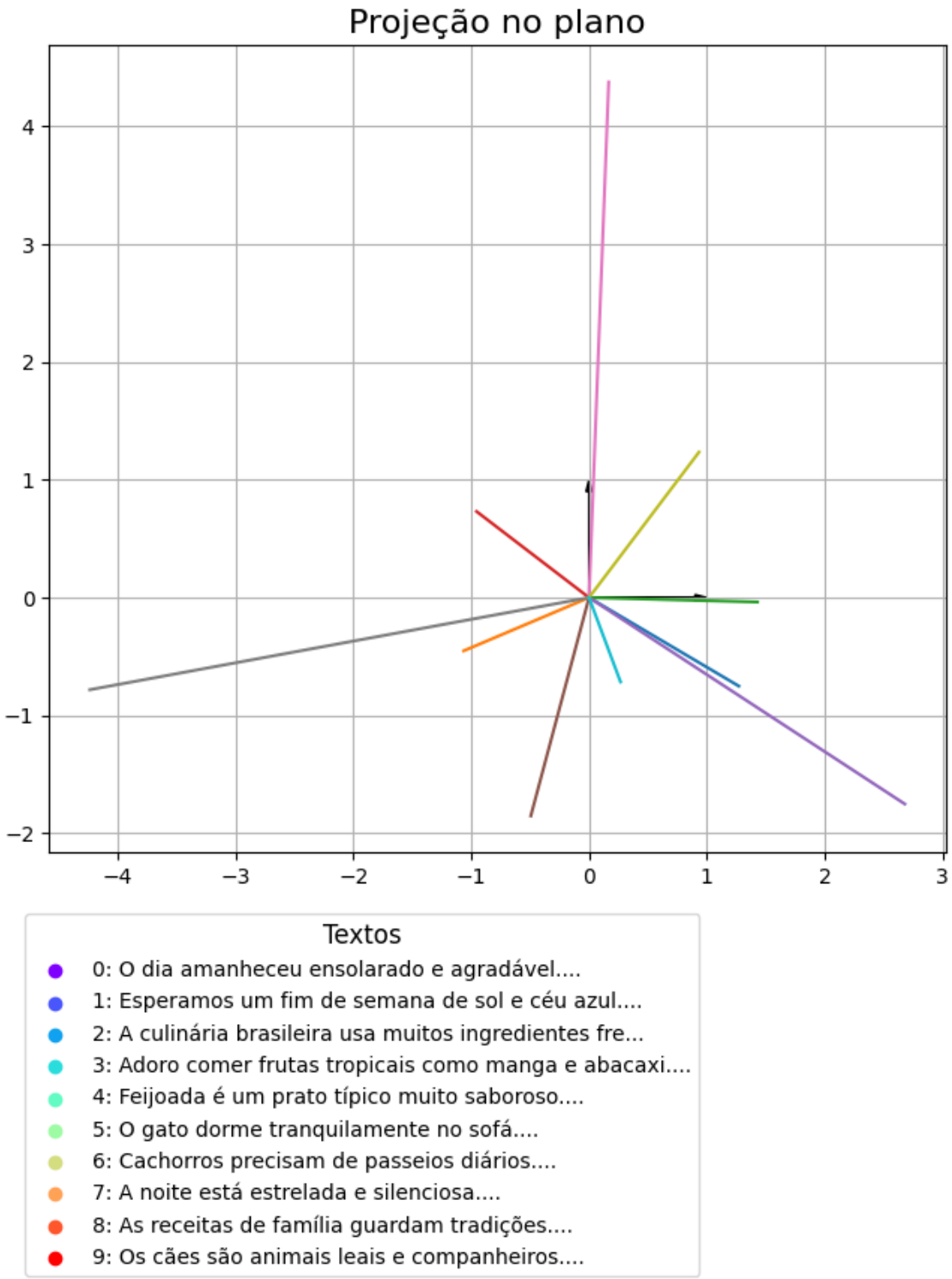


O gráfico acima mostra cada texto original como um ponto no espaço 2D. A posição de cada ponto é determinada pelo vetor da sentença (média dos vetores das palavras) após a redução com PCA. Textos com conteúdo semanticamente similar (segundo o modelo Word2Vec) devem aparecer mais próximos uns dos outros. Com um corpus pequeno, as relações capturadas pelo Word2Vec podem ser fracas ou baseadas em coocorrências simples de palavras.

Vetores e plano calculado para projeção



O gráfico acima representa a visualização dos vetores das sentenças em três dimensões, essa técnica é uma alternativa para gerar uma visualização 2D desses vetores com a dimensionalidade menor e, portanto, mais compreensível para a interpretação humana, como segue abaixo:



Rua Dr. Alcides Vieira Arcoverde, 1225
CEP 81520-260 – Jardim das Américas
Curitiba/PR - Brasil

Anexo I – Código Fonte do Exercício 1

Rua Dr. Alcides Vieira Arcoverde, 1225
CEP 81520-260 – Jardim das Américas
Curitiba/PR - Brasil

```python
# -*- coding: utf-8 -*-
"""IAA015_Caixeiro_Viajante_GA_v2.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/github/Equipe16-IAA/IAA015/blob/main/IAA015_Caixeiro_Viajante_GA_v2.ipynb
"""

import numpy as np
from random import randint, random
import random
import matplotlib.pyplot as plt
import math
import copy

"""#Criando as funções principais do Algorítmo Genético

## Criando as coordenadas das 100 cidades
"""

NRO_CIDADES = 100

x_points = list(range(NRO_CIDADES))
y_points = list(range(NRO_CIDADES))

random.shuffle(x_points)
random.shuffle(y_points)

plt.title("Gráfico de cidades geradas.")
plt.xlabel("Coordenada X")
plt.ylabel("Coordenada Y")
plt.plot(x_points, y_points, 'ro')
plt.show()

"""###População Inicial"""

def populacaoInicial(tamanho):
  percursos = []
  percurso = []
  for i in range(0, tamanho):
    cidades = list(range(0, NRO_CIDADES))
    listaCidades = cidades.copy()
    random.shuffle(listaCidades)
    for cidade in cidades:
      cidadePercurso = random.choice(listaCidades)
      listaCidades.remove(cidadePercurso)
      percurso.append(cidadePercurso)
    percursos.append(percurso)
    percurso = []
    cidades = []
  return percursos;

##DEBUG função População Inicial

percursos = populacaoInicial(2)

for percurso in percursos:
  print(percurso)

"""###Verifica Elementos Duplicados"""

def has_duplicates(lst):
    n = max(set(lst), key=lst.count)
    return lst.count(n) > 1

##DEBUG função duplicados
listaDuplicados = [2, 1, 2, 5, 2]
print(has_duplicates(listaDuplicados))
```

```
71  listaSemDuplicados = [1, 2, 3, 4, 5]
72  print(has_duplicates(listaSemDuplicados))
73
74  """###Função de Avaliação (fit)"""
75
76  ##Calcular Distância Euclidiana
77  ##distancia = sqrt(((x2 - x1) ^ 2) + ((y2 - y1) ^ 2));
78  def FuncaoDeAvaliacao(percursos):
79    soma_distancia = np.zeros(len(percursos));
80    for index, percurso in enumerate(percursos):
81      for i, cidade in enumerate(percurso):
82        if i < (len(percurso) - 1):
83          soma_distancia[index] += math.sqrt((x_points[percurso[i + 1]] - x_points[
            percurso[i]])**2 + (y_points[percurso[i + 1]] - y_points[percurso[i]])**2);
84        else:
85          soma_distancia[index] += math.sqrt((x_points[percurso[i]] - x_points[percurso[
            0]])**2 + (y_points[percurso[i]] - y_points[percurso[0]])**2);
86      if has_duplicates(percurso):
87        soma_distancia[index] = 999999999
88    return soma_distancia;
89
90
91  def FuncaoDeAvaliacaoV2(percurso):
92    soma_distancia = 0
93    for i, cidade in enumerate(percurso):
94      if i < (len(percurso) - 1):
95        soma_distancia += math.sqrt((x_points[percurso[i + 1]] - x_points[percurso[i
          ]])**2 + (y_points[percurso[i + 1]] - y_points[percurso[i]])**2);
96      else:
97        soma_distancia += math.sqrt((x_points[percurso[i]] - x_points[percurso[0]])**2 +
           (y_points[percurso[i]] - y_points[percurso[0]])**2);
98    if has_duplicates(percurso):
99      soma_distancia = 999999999
100   return soma_distancia;
101
102 ##DEBUG Função de Avaliação
103
104 percursos = populacaoInicial(2)
105 print(percursos)
106 avaliacao = FuncaoDeAvaliacao(percursos)
107 print(avaliacao)
108
109 print(f'funcao v2 {FuncaoDeAvaliacaoV2(percursos[0])}')
110
111 """###Preserva Melhor da Geração"""
112
113 def preservaMelhor(geracao, novaGeracao):
114   avaliacao = FuncaoDeAvaliacao(geracao)
115   maior = 0
116   for i in range(len(avaliacao)):
117     if avaliacao[maior] > avaliacao[i]:
118       maior = i
119   #print(avaliacao[maior])
120   novaGeracao.append(geracao[maior])
121
122 #Debug preservar melhor geraçao
123 novaGeracao = []
124 preservaMelhor(percursos, novaGeracao)
125 print(novaGeracao)
126
127 """##Cruzamento Genético"""
128
129 ##TODO - Implementar cruzamento OX, para evitar repetição
130
131 def cruzamento(geracao, num, nova):
132   while True:
133     for i in range(0, num):
134       indA = random.randrange(0,len(geracao))
135       indB = indA;
136       while indA==indB:
137         indB = random.randrange(0,len(geracao))
138       #print(f'cruzamento {indA} e {indB}')
```

```
        ponto1 = random.randrange(1,round((len(geracao[0]) - 1)))
        #print(f'ponto de corte {ponto1}')

        paiA = geracao[indA]
        paiB = geracao[indB]

        res1 = list(set(paiA[0:ponto1]) & set(paiB[ponto1:]))
        res2 = list(set(paiB[0:ponto1]) & set(paiA[ponto1:]))

        cromossomosA = paiA[0:ponto1]
        cromossomosB = paiB[ponto1:]
        for i in range(0, len(res1)):
          cromossomosB.remove(res1[i])
          cromossomosB.append(res2[i])

        filho1 = cromossomosA
        filho1.extend(cromossomosB)
        #print(f'filho1 {filho1} duplicado? {has_duplicates(filho1)}')
        nova.append(filho1)
      break

##DEBUG cuzamento
novaGeracao = []
geracao = populacaoInicial(5)
print(geracao)
#preservaMelhor(geracao, novaGeracao)
cruzamento(geracao, 4, novaGeracao)
print(geracao)
print(novaGeracao)

"""##Mutação"""

#TODO: Implementar verificação de avaliação antes de inserir o individuo que sofreu mutação na solução, incluir porcentagem para isso ocorrer

def mutacao(geracao, taxaMutacao):
  if randint(1,100) <= taxaMutacao:
    individuoMutacao = random.randrange(0,len(geracao))
    print(f'Realizando mutacao no Individuo:  {individuoMutacao}')
    gene_1 = random.randrange(0,len(geracao[0]))
    gene_2 = random.randrange(0,len(geracao[0]))
    while gene_1 == gene_2:
      gene_2 = random.randrange(0,len(geracao[0]))
    copiaGeracao = copy.deepcopy(geracao)
    temp = copiaGeracao[individuoMutacao][gene_1]
    copiaGeracao[individuoMutacao][gene_1] = copiaGeracao[individuoMutacao][gene_2]
    copiaGeracao[individuoMutacao][gene_2] = temp
    #print(f'Individuo a ser mutado:  {geracao[individuoMutacao]} //Resultado após mutação:  {copiaGeracao[individuoMutacao]}')
    avaliacaoCopia = FuncaoDeAvaliacaoV2(copiaGeracao[individuoMutacao])
    avaliacaoGeracao = FuncaoDeAvaliacaoV2(geracao[individuoMutacao])
    #print(f'Avaliação antes da mutação: {avaliacaoGeracao}')
    #print(f'Avaliação após a mutação: {avaliacaoCopia}')
    ##30% de chances de verificar se a mutação criou individuo melhor antes de executar a mutação
    if random.randrange(0,100) < 101: ##Forcando a sempre entrar aqui para testes
      print(f'Entrou nos 30% de chance de avaliação da Mutação - Avaliando ...')
      if avaliacaoCopia < avaliacaoGeracao:
        print(f'Avaliação superior ao anterior, enviando para a mansão X')
        geracao[individuoMutacao] = copiaGeracao[individuoMutacao]
      else:
        print(f'Avaliação abaixo do anterior, tentando uma nova mutacao ...')
        mutacao(geracao,100)
    else:
      print(f'Mutação não considerou avaliação')
      geracao[individuoMutacao] = copiaGeracao[individuoMutacao]

#Teste Mutacao

geracao = populacaoInicial(100)
print(geracao)
```

```
208   for i in range(0,100):
209     mutacao(geracao, 100)
210   print(geracao)
211
212   """##Execução do Algoritmo Genético"""
213
214   #PRESETS AG
215   #------------------------------
216   numGeracoes = 10000
217   populacao = 100
218   taxaCruzamento = 90
219   taxaMutacao = 1
220   #------------------------------
221   p0 = populacaoInicial(populacao)
222   melhorPrimeiraGeracao = []
223   preservaMelhor(p0, melhorPrimeiraGeracao)
224   print(f'Distância percorrida: {FuncaoDeAvaliacaoV2(melhorPrimeiraGeracao[0])}')
225
226
227   plt.title("Gráfico do primeiro caminho encontrado.")
228   plt.xlabel("Coordenada X")
229   plt.ylabel("Coordenada Y")
230   plt.plot(x_points, y_points, 'ro')
231
232   for i in range(0, len(melhorPrimeiraGeracao[0]) - 1):
233     plt.plot([x_points[melhorPrimeiraGeracao[0][i]], x_points[melhorPrimeiraGeracao[0][i
        + 1]]], [y_points[melhorPrimeiraGeracao[0][i]], y_points[melhorPrimeiraGeracao[0][i
        + 1]]], 'k-')
234     if i == (len(melhorPrimeiraGeracao[0]) - 2):
235       plt.plot([x_points[melhorPrimeiraGeracao[0][i + 1]], x_points[
          melhorPrimeiraGeracao[0][0]]], [y_points[melhorPrimeiraGeracao[0][i + 1]],
          y_points[melhorPrimeiraGeracao[0][0]]], 'k-')
236   plt.show()
237
238   avaliacaoP0 = FuncaoDeAvaliacao(p0)
239
240   geracao = copy.deepcopy(p0)
241   while numGeracoes > 0:
242
243     if numGeracoes % 100 == 0:
244       print(f'Geraçao {numGeracoes}')
245
246     novaGeracao = []
247     mutacao(geracao, taxaMutacao)
248     preservaMelhor(geracao, novaGeracao)
249     cruzamento(geracao, taxaCruzamento, novaGeracao)
250     numGeracoes = numGeracoes - 1
251     avaliacaoNovaGeracao = FuncaoDeAvaliacao(novaGeracao)
252     novaGeracao2 = sorted(zip(avaliacaoNovaGeracao, novaGeracao), reverse=True)
253     geracao = [x for _, x in novaGeracao2]
254
255
256
257   geracao = novaGeracao.copy()
258   novaGeracao = []
259   preservaMelhor(geracao, novaGeracao)
260
261   plt.title("Gráfico de caminho encontrado.")
262   plt.xlabel("Coordenada X")
263   plt.ylabel("Coordenada Y")
264   plt.plot(x_points, y_points, 'ro')
265
266   print(f'Distância percorrida: {FuncaoDeAvaliacaoV2(novaGeracao[0])}')
267   print(f'Duplicidade? {has_duplicates(novaGeracao[0])}')
268
269   for i in range(0, len(novaGeracao[0]) - 1):
270     plt.plot([x_points[novaGeracao[0][i]], x_points[novaGeracao[0][i + 1]]], [y_points[
        novaGeracao[0][i]], y_points[novaGeracao[0][i + 1]]], 'k-')
271     if i == (len(novaGeracao[0]) - 2):
272       plt.plot([x_points[novaGeracao[0][i + 1]], x_points[novaGeracao[0][0]]], [y_points
          [novaGeracao[0][i + 1]], y_points[novaGeracao[0][0]]], 'k-')
273
```

Anexo II – Código Fonte do Exercício 2

Rua Dr. Alcides Vieira Arcoverde, 1225
CEP 81520-260 – Jardim das Américas
Curitiba/PR - Brasil

```python
# -*- coding: utf-8 -*-
"""Comparação modelos vetoriais.ipynb

Automatically generated by Colab.

Original file is located at

    https://colab.research.google.com/github/Equipe16-IAA/IAA015/blob/main/Compara%C3%A7%C3%A3o%20modelos%20vetoriais.ipynb

# 2) Compare a representação de dois modelos vetoriais

## Utilizando word2vec
"""

#Gensim do Colab não funciona tem que instalar e reiniciar
!pip install gensim nltk scikit-learn matplotlib numpy

# --- 0. Instalação e Imports (Necessário no Colab se não estiver pré-instalado) ---
# Descomente e execute as linhas !pip se receber erro de módulo não encontrado
# !pip install gensim nltk scikit-learn matplotlib numpy

import re # Para expressões regulares (limpeza de texto)
import numpy as np
from gensim.models import Word2Vec
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import matplotlib.cm as cm # Para gerar cores distintas para os pontos
import nltk
import math
from warnings import simplefilter
from scipy.cluster.hierarchy import dendrogram, linkage, ClusterWarning

#pacotes necessários para o tokenizador
nltk.download('punkt')

# Baixar o tokenizador 'punkt' do NLTK (necessário na primeira execução no ambiente)
try:
    nltk.data.find('tokenizers/punkt')
except nltk.downloader.DownloadError:
    print("Baixando o tokenizador 'punkt' do NLTK...")
    nltk.download('punkt', quiet=True)
    print("Download concluído.")

print("Bibliotecas importadas e NLTK pronto.")
print("-" * 40)

# --- 1. Definição dos Textos ---
# Mínimo 6 textos, com pelo menos 2 pares similares.
# Limitando a ~10-12 para clareza na visualização.
texts = [
    "O dia amanheceu ensolarado e agradável.",             # Clima 1
    "Esperamos um fim de semana de sol e céu azul.",       # Clima 2 (similar ao 1)
    "A culinária brasileira usa muitos ingredientes frescos.", # Culinária 1
    "Adoro comer frutas tropicais como manga e abacaxi.",  # Culinária/Frutas 2
    "Feijoada é um prato típico muito saboroso.",          # Culinária 3 (similar ao 3)
    "O gato dorme tranquilamente no sofá.",                # Animais 1
    "Cachorros precisam de passeios diários.",             # Animais 2
    "A noite está estrelada e silenciosa.",                # Clima/Noite 3 (ligeiramente similar ao 1/2)
    "As receitas de família guardam tradições.",           # Culinária 4 (similar ao 3/5)
    "Os cães são animais leais e companheiros.",           # Animais 3 (similar ao 7)
]

# Garantir que não exceda um limite razoável para o plot
max_texts_to_plot = 12
if len(texts) > max_texts_to_plot:
    print(f"Limitando a {max_texts_to_plot} textos para visualização.")
    texts = texts[:max_texts_to_plot]

```

```python
print(f"Textos de entrada ({len(texts)}):")
for i, t in enumerate(texts):
    print(f"{i}: {t}")
print("-" * 40)

nltk.download('punkt_tab')

# --- 2. Pré-processamento ---
def preprocess_text(text):
    """Limpa e tokeniza o texto."""
    # Converte para minúsculas
    text = text.lower()
    # Remove pontuações básicas e números (opcional, mas ajuda a focar nas palavras)
    text = re.sub(r'[^\w\s]', '', text) # Remove pontuação
    text = re.sub(r'\d+', '', text)      # Remove números
    # Tokeniza (divide em palavras)
    tokens = nltk.word_tokenize(text, language='portuguese') # Especificar idioma ajuda
    # Remove palavras vazias (stopwords) - Opcional, pode ou não ajudar com poucos dados
    from nltk.corpus import stopwords
    nltk.download('stopwords') # Se for usar
    stop_words = set(stopwords.words('portuguese'))
    tokens = [word for word in tokens if word not in stop_words and len(word) > 1] # Remove stopwords e palavras de 1 letra
    tokens = [word for word in tokens if len(word) > 1] # Apenas remove palavras de 1 letra
    return tokens

tokenized_texts = [preprocess_text(t) for t in texts]

print("Textos Tokenizados (Exemplo do primeiro):")
print(tokenized_texts[0])
print("-" * 40)

# --- 3. Treinamento do Modelo Word2Vec ---
# Parâmetros importantes para datasets pequenos:
# min_count=1: Inclui todas as palavras. Essencial aqui.
# epochs: Aumentar o número de épocas de treino.
# vector_size: Dimensão dos vetores (100 é comum, mas 50 pode ser suficiente aqui).
# window: Contexto de palavras vizinhas.
# sg=1: Usa Skip-gram (geralmente melhor para datasets menores).

vector_dim = 50  # Dimensão dos vetores de palavras
w2v_model = Word2Vec(sentences=tokenized_texts,
                     vector_size=vector_dim,
                     window=3,         # Janela menor pode ser melhor para frases curtas
                     min_count=1,      # Crucial para datasets pequenos
                     workers=4,        # Número de threads (ajuste conforme CPU disponível no Colab)
                     sg=1,             # Skip-gram
                     epochs=100)       # Mais épocas para compensar poucos dados

print("Modelo Word2Vec treinado.")
print(f"Tamanho do vocabulário: {len(w2v_model.wv.index_to_key)}")
# Exemplo: Palavras mais similares a 'sol' (se 'sol' estiver no vocabulário)
try:
    similar_words = w2v_model.wv.most_similar('sol', topn=3)
    print(f"Palavras mais similares a 'sol': {similar_words}")
except KeyError:
    print("Palavra 'sol' não encontrada no vocabulário (corpus muito pequeno).")
print("-" * 40)

# --- 4. Cálculo dos Vetores de Sentenças ---
# Estratégia: Média dos vetores das palavras na sentença

def get_sentence_vector(tokens, model, vector_size):
    """Calcula o vetor médio para uma lista de tokens."""
    vector = np.zeros(vector_size)
    count = 0
    for word in tokens:
```

```
            if word in model.wv: # Checa se a palavra está no vocabulário do modelo
                vector += model.wv[word]
                count += 1
        if count > 0:
            vector /= count # Calcula a média
        # else: # Opcional: Lidar com sentenças onde nenhuma palavra está no vocabulário
        #     print(f"Aviso: Sentença com tokens '{tokens}' não possui palavras no vocabulário.")
        return vector
    # Calcula o vetor para cada texto tokenizado
    sentence_vectors = np.array([get_sentence_vector(tokens, w2v_model, vector_dim)
                                 for tokens in tokenized_texts])

    print(f"Vetores das sentenças calculados. Shape: {sentence_vectors.shape}")
    # Verificar se há vetores nulos (indicativo de problemas)
    if np.any(np.all(sentence_vectors == 0, axis=1)):
        print("AVISO: Uma ou mais sentenças resultaram em vetor nulo (nenhuma palavra no vocabulário?).")
    print("-" * 40)

    # --- 5. Redução de Dimensionalidade com PCA ---
    # Reduzir os vetores de 'vector_dim' dimensões para 2 dimensões para plotagem

    pca = PCA(n_components=3, random_state=42) # random_state para reprodutibilidade
    pca_result = pca.fit_transform(sentence_vectors)

    print(f"Vetores reduzidos para 2D com PCA. Shape: {pca_result.shape}")
    print("Componentes Principais (x, y, z) para cada sentença:")

    pca_result = pca_result * 100 #Multiplicado por 100 para melhor visualização nos gráficos.

    for i, (x, y, z) in enumerate(pca_result):
        print(f"Texto {i}: ({x:.4f}, {y:.4f}, {z:.4f})")
    print("-" * 40)

    # --- 6. Visualização dos Vetores 2D ---
    #
    plt.figure(figsize=(12, 9)) # Ajuste o tamanho conforme necessário

    # Gerar cores distintas para cada ponto usando um colormap
    colors = cm.rainbow(np.linspace(0, 1, len(texts)))

    # Plotar cada ponto (vetor 2D da sentença)
    for i, (x, y, z) in enumerate(pca_result):
        plt.scatter(x, y, color=colors[i], s=150, alpha=0.8, edgecolors='k')
        # Adicionar anotação (número do texto) ligeiramente deslocado do ponto
        plt.text(x + 0.1, y + 0.1, str(i), fontsize=10, ha='left', va='bottom')

    # Variância explicada
    explained_variance_ratio = pca.explained_variance_ratio_
    total_explained_variance = explained_variance_ratio.sum() * 100

    plt.title(f'Visualização 2D dos Vetores de Sentenças (Word2Vec + PCA)\nVariância Total Explicada: {total_explained_variance:.1f}%', fontsize=16)
    plt.xlabel(f'Componente Principal 1 ({explained_variance_ratio[0]*100:.1f}% variância)', fontsize=12)
    plt.ylabel(f'Componente Principal 2 ({explained_variance_ratio[1]*100:.1f}% variância)', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.5)

    legend_handles = [plt.scatter([], [], color=colors[i], label=f'{i}: {texts[i][:50]}...')
                      for i in range(len(texts))]
    plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(0.5, -0.5),
               fontsize='medium', title="Textos", title_fontsize='large', borderaxespad=1.)

    plt.margins(0.1)
    plt.axhline(0, color='grey', lw=0.5)
    plt.axvline(0, color='grey', lw=0.5)

```

```
198   plt.tight_layout(rect=[0, 0, 0.85, 1])
199   plt.show()
200
201   print("\n--- Interpretação do Gráfico ---")
202   print("O gráfico acima mostra cada texto original como um ponto no espaço 2D.")
203   print("A posição de cada ponto é determinada pelo vetor da sentença (média dos
      vetores das palavras) após a redução com PCA.")
204   print("Textos com conteúdo semanticamente similar (segundo o modelo Word2Vec) devem
      aparecer mais próximos uns dos outros.")
205   print("Com um corpus pequeno, as relações capturadas pelo Word2Vec podem ser fracas
      ou baseadas em coocorrências simples de palavras.")
206
207   centro = pca_result.mean(axis=0)
208   v_centro = pca_result - centro
209
210   print('centro','\n',centro)
211   print( np.allclose(pca_result, v_centro+centro))
212
213   # obter o plano para projeção ortogonal
214   U,s,Vh = np.linalg.svd(v_centro)
215   print('Variancia: ', np.square(s) / np.square(s).sum() )
216   W2 = Vh.T[:, :2]
217   v1_plano = Vh.T[:, 0]
218   v2_plano = Vh.T[:, 1]
219   projetados2d = v_centro.dot(W2)
220   print(projetados2d)
221   # calculando o plano de projeção
222   menor = np.min(pca_result,axis=0)-1
223   maior = np.min(pca_result,axis=0)+1
224   x1s = np.linspace(menor[0], maior[0], 10)
225   y1s = np.linspace(menor[1], maior[1], 10)
226   z1s = np.linspace(menor[2], maior[2], 10)
227   C = Vh
228   R = C.T.dot(C)
229   x1, x2 = np.meshgrid(x1s, y1s)
230   z = (R[0, 2] * x1 + R[1, 2] * x2) / (1 - R[2, 2])
231   #
232   # tentando visualizar as projeções no espaço 3d
233   fig = plt.figure(figsize=(12, 9)) # Ajuste o tamanho conforme necessário
234   ax = fig.add_subplot(111, projection='3d')
235   ax.plot_surface(x1, x2, z, alpha=0.2, color="k")
236   #ax.view_init(elev=180, azim=180)
237   # pontos originais
238   for v in pca_result:
239       xs = np.linspace(0, v[0], 10)
240       ys = np.linspace(0, v[1], 10)
241       zs = np.linspace(0, v[2], 10)
242       plt.plot(xs,ys,zs=zs)
243   plt.show()
244
245   fig = plt.figure(figsize=(12, 9))
246   ax = fig.add_subplot(111, aspect='equal')
247   # pontos projetados
248   for v in pca_result:
249       xs = np.linspace(0, v[0], 10)
250       ys = np.linspace(0, v[1], 10)
251       ax.plot(xs,ys)
252
253   ax.arrow(0, 0, 0, 1, head_width=0.05, \
254           length_includes_head=True, head_length=0.1, fc='k', ec='k')
255   ax.arrow(0, 0, 1, 0, head_width=0.05, \
256           length_includes_head=True, head_length=0.1, fc='k', ec='k')
257   ax.grid(True)
258   #plt.xscale('log')
259   #plt.yscale('log')
260   #plt.xlim(-0.05,0.05)
261   #plt.ylim(-0.05,0.05)
262   plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(0.5, -0.5),
263              fontsize='medium', title="Textos", title_fontsize='large', borderaxespad=1.
                )
264   plt.show()
265
```

```
266   def vectorDistance(v1,v2):
267       soma = 0
268       for i in range(0,len(v1)):
269           d = v1[i]-v2[i]
270           soma = soma + d*d
271       return math.sqrt(soma)
272
273   def calcDistance(mat):
274       nrows = len(mat)
275       ncols = len(mat[0])
276       res = np.zeros( (nrows,nrows) )
277       for i in range(0,nrows):
278           m1 = mat[i]
279           for j in range(0,nrows):
280               if i!=j:
281                   m2 = mat[j]
282                   res[i][j] = vectorDistance(m1,m2)
283               else:
284                   res[i][j] = 0
285       return res
286
287   d = calcDistance(pca_result)
288   d1 = calcDistance(projetados2d)
289   print('original')
290   print(d)
291   print('projetada')
292   print(d1)
293
294
295   simplefilter("ignore", ClusterWarning)
296   Z1 = linkage(d, 'single')
297   fig = plt.figure()
298   dn = dendrogram(Z1)
299   plt.show()
300   Z2 = linkage(d1, 'single')
301   fig = plt.figure()
302   dn = dendrogram(Z2)
303   plt.show()
```

```
274   plt.plot(x_points[novaGeracao[0][0]], y_points[novaGeracao[0][0]], 'go')
275   plt.show()
```