



Ministério da Educação
UNIVERSIDADE FEDERAL DO PARANÁ
SEPT – Setor de Educação Profissional e Tecnológica
Especialização em Inteligência Artificial Aplicada



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

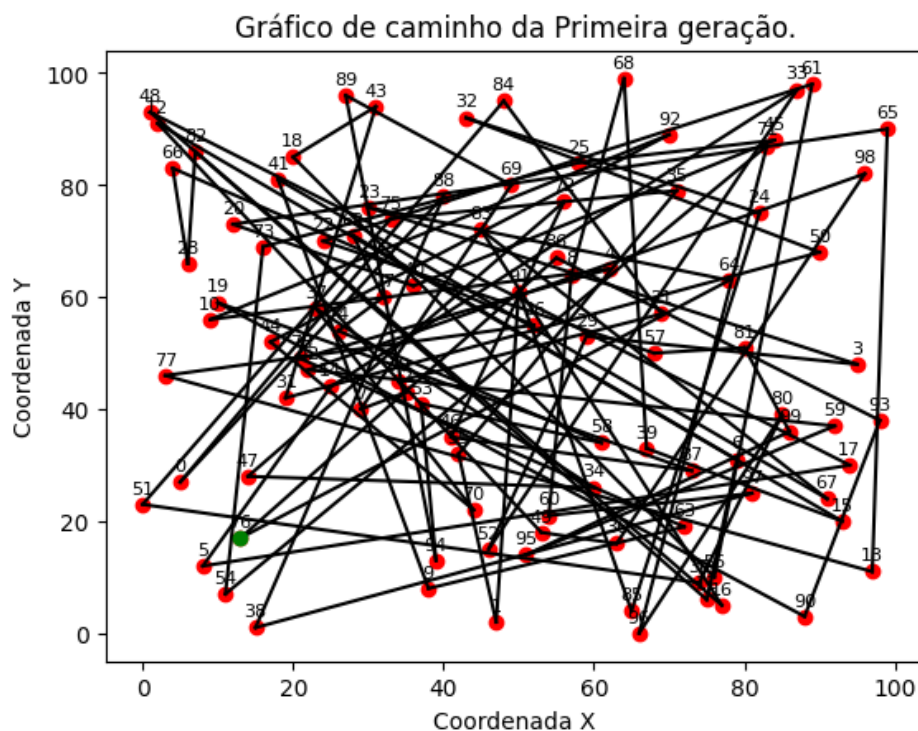
1 – Algoritmo Genético

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%.

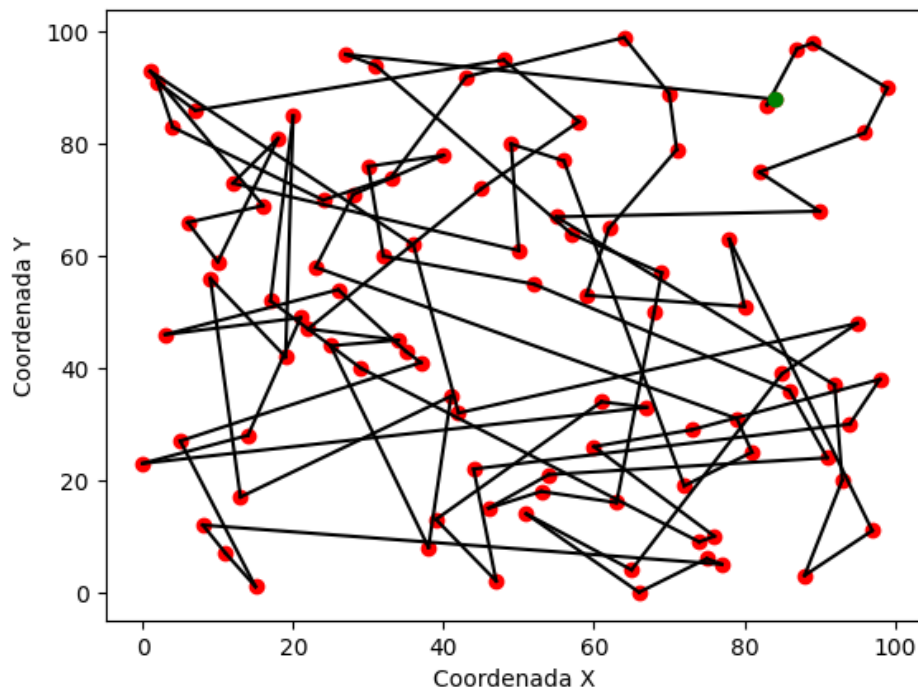
Melhor indivíduo da primeira geração:



Distância total percorrida: 5569.9 pixels

Melhor indivíduo da última geração:

Gráfico de caminho encontrado.



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.

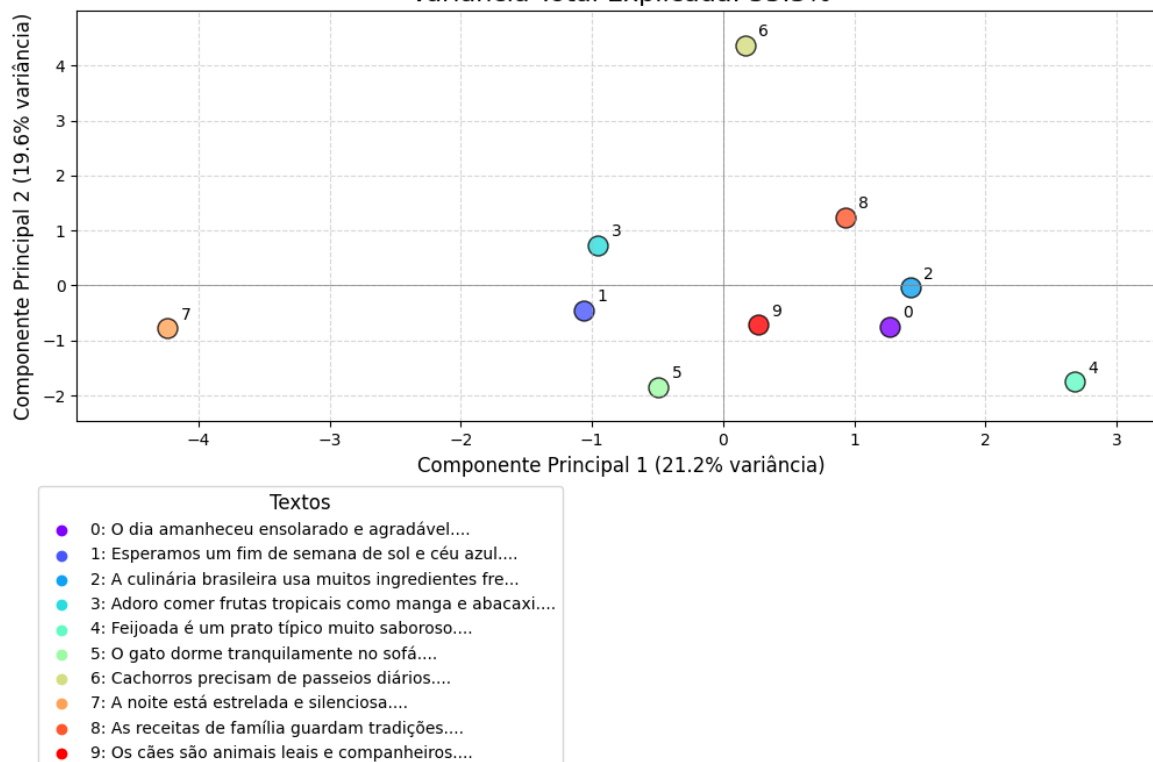
A representação vetorial foi obtida via Word2Vec, treinado com 100 épocas para capturar nuances semânticas em nosso pequeno corpus. A representação de cada sentença foi calculada pela média dos vetores das palavras, resultando em vetores de alta dimensionalidade.

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.

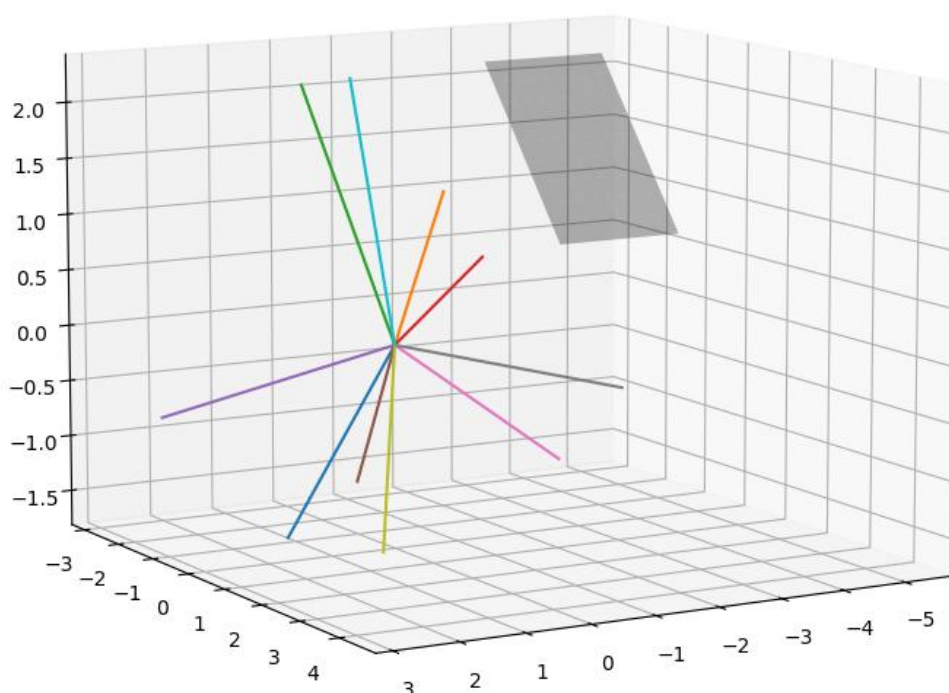
Frases utilizadas		relações
1	O dia amanheceu ensolarado e agradável.	Clima 1
2	Esperamos um fim de semana de sol e céu azul.	Clima 2 (similar a 1)
3	A culinária brasileira usa muitos ingredientes frescos.	Culinária 1
4	Adoro comer frutas tropicais como manga e abacaxi.	Culinária/frutas 2
5	Feijoada 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

Visualização 2D dos Vetores de Sentenças (Word2Vec + PCA)
Variância Total Explicada: 55.3%



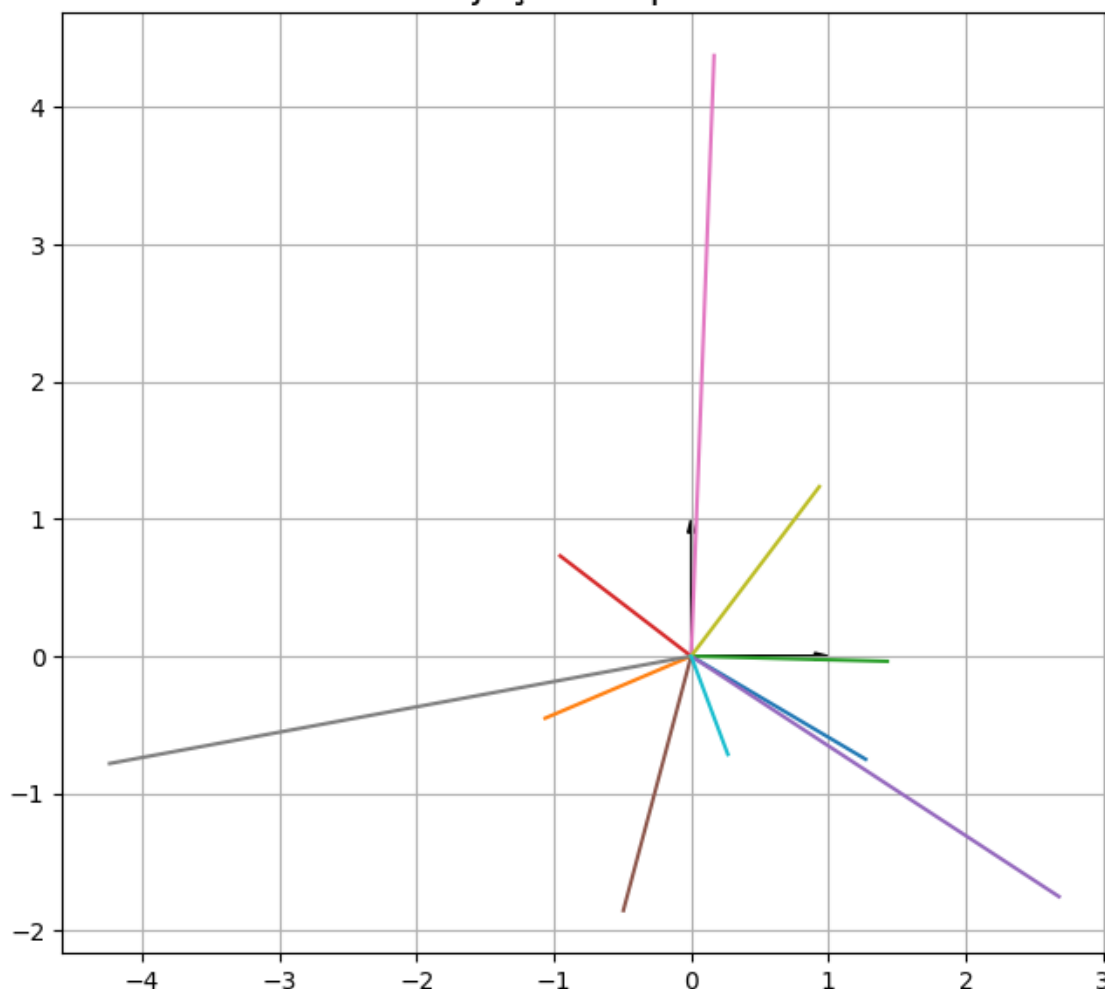
O gráfico acima mostra cada texto original como um ponto no espaço 2D. A posição de cada ponto é determinada pelo vetor da sentença (média dos vetores das palavras) após a redução com PCA. Textos com conteúdo semanticamente similar (segundo o modelo Word2Vec) devem aparecer mais próximos uns dos outros.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



Textos	
0:	O dia amanheceu ensolarado e agradável....
1:	Esperamos um fim de semana de sol e céu azul....
2:	A culinária brasileira usa muitos ingredientes fre...
3:	Adoro comer frutas tropicais como manga e abacaxi....
4:	Feijoada é um prato típico muito saboroso....
5:	O gato dorme tranquilamente no sofá....
6:	Cachorros precisam de passeios diários....
7:	A noite está estrelada e silenciosa....
8:	As receitas de família guardam tradições....
9:	Os cães são animais leais e companheiros....

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:

Projeção no plano



Textos

- 0: O dia amanheceu ensolarado e agradável....
- 1: Esperamos um fim de semana de sol e céu azul....
- 2: A culinária brasileira usa muitos ingredientes fre...
- 3: Adoro comer frutas tropicais como manga e abacaxi....
- 4: Feijoada é um prato típico muito saboroso....
- 5: O gato dorme tranquilamente no sofá....
- 6: Cachorros precisam de passeios diários....
- 7: A noite está estrelada e silenciosa....
- 8: As receitas de família guardam tradições....
- 9: Os cães são animais leais e companheiros....



Ministério da Educação
UNIVERSIDADE FEDERAL DO PARANÁ
SEPT – Setor de Educação Profissional e Tecnológica
Especialização em Inteligência Artificial Aplicada



Anexo I – Código Fonte do Exercício 1


```
1  # -*- coding: utf-8 -*-
2  """IAA015_Caixaiero_Viajante_GA_v2.ipynb
3
4  Automatically generated by Colab.
5
6  Original file is located at
7
8      https://colab.research.google.com/github/Equipel6-IAA/IAA015/blob/main/IAA015\_Caixaiero\_Viajante\_GA\_v2.ipynb
9  """
10 import numpy as np
11 from random import randint, random
12 import random
13 import matplotlib.pyplot as plt
14 import math
15 import copy
16
17 """#Criando as funções principais do Algoritmo Genético
18
19 ## Criando as coordenadas das 100 cidades
20 """
21
22 NRO_CIDADES = 100
23
24 x_points = list(range(NRO_CIDADES))
25 y_points = list(range(NRO_CIDADES))
26
27 random.shuffle(x_points)
28 random.shuffle(y_points)
29
30 plt.title("Gráfico de cidades geradas.")
31 plt.xlabel("Coordenada X")
32 plt.ylabel("Coordenada Y")
33 plt.plot(x_points, y_points, 'ro')
34 plt.show()
35
36 """###População Inicial"""
37
38 def populacaoInicial(tamanho):
39     percursos = []
40     percurso = []
41     for i in range(0, tamanho):
42         cidades = list(range(0, NRO_CIDADES))
43         listaCidades = cidades.copy()
44         random.shuffle(listaCidades)
45         for cidade in cidades:
46             cidadePercurso = random.choice(listaCidades)
47             listaCidades.remove(cidadePercurso)
48             percurso.append(cidadePercurso)
49         percursos.append(percurso)
50         percurso = []
51         cidades = []
52     return percursos;
53
54 ##DEBUG função População Inicial
55
56 percursos = populacaoInicial(2)
57
58 for percurso in percursos:
59     print(percurso)
60
61 """###Verifica Elementos Duplicados"""
62
63 def has_duplicates(lst):
64     n = max(set(lst), key=lst.count)
65     return lst.count(n) > 1
66
67 ##DEBUG função duplicados
68 listaDuplicados = [2, 1, 2, 5, 2]
69 print(has_duplicates(listaDuplicados))
70
```

```

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

```

```

139     ponto1 = random.randrange(1, round((len(geracao[0]) - 1)))
140     #print(f'ponto de corte {ponto1}')
141
142     paiA = geracao[indA]
143     paiB = geracao[indB]
144
145     res1 = list(set(paiA[0:ponto1]) & set(paiB[ponto1:]))
146     res2 = list(set(paiB[0:ponto1]) & set(paiA[ponto1:]))
147
148     cromossomosA = paiA[0:ponto1]
149     cromossomosB = paiB[ponto1:]
150     for i in range(0, len(res1)):
151         cromossomosB.remove(res1[i])
152         cromossomosB.append(res2[i])
153
154     filho1 = cromossomosA
155     filho1.extend(cromossomosB)
156     #print(f'filho1 {filho1} duplicado? {has_duplicates(filho1)}')
157     nova.append(filho1)
158     break
159
160 ##DEBUG cruzamento
161 novaGeracao = []
162 geracao = populacaoInicial(5)
163 print(geracao)
164 #preservaMelhor(geracao, novaGeracao)
165 cruzamento(geracao, 4, novaGeracao)
166 print(geracao)
167 print(novaGeracao)
168
169 """##Mutação"""
170
171 #TODO: Implementar verificação de avaliação antes de inserir o individuo que sofreu
mutação na solução, incluir porcentagem para isso ocorrer
172
173 def mutacao(geracao, taxaMutacao):
174     if randint(1,100) <= taxaMutacao:
175         individuoMutacao = random.randrange(0, len(geracao))
176         print(f'Realizando mutacao no Indivíduo: {individuoMutacao}')
177         gene_1 = random.randrange(0, len(geracao[0]))
178         gene_2 = random.randrange(0, len(geracao[0]))
179         while gene_1 == gene_2:
180             gene_2 = random.randrange(0, len(geracao[0]))
181         copiaGeracao = copy.deepcopy(geracao)
182         temp = copiaGeracao[individuoMutacao][gene_1]
183         copiaGeracao[individuoMutacao][gene_1] = copiaGeracao[individuoMutacao][gene_2]
184         copiaGeracao[individuoMutacao][gene_2] = temp
185         #print(f'Indivíduo a ser mutado: {geracao[individuoMutacao]} //Resultado após
mutação: {copiaGeracao[individuoMutacao]}')
186         avaliacaoCopia = FuncaoDeAvaliacaoV2(copiaGeracao[individuoMutacao])
187         avaliacaoGeracao = FuncaoDeAvaliacaoV2(geracao[individuoMutacao])
188         #print(f'Avaliação antes da mutação: {avaliacaoGeracao}')
189         #print(f'Avaliação após a mutação: {avaliacaoCopia}')
190         ##30% de chances de verificar se a mutação criou individuo melhor antes de
executar a mutação
191         if random.randrange(0,100) < 101: ##Forçando a sempre entrar aqui para testes
192             print(f'Entrou nos 30% de chance de avaliação da Mutação - Avaliando ...')
193             if avaliacaoCopia < avaliacaoGeracao:
194                 print(f'Avaliação superior ao anterior, enviando para a mansão X')
195                 geracao[individuoMutacao] = copiaGeracao[individuoMutacao]
196             else:
197                 print(f'Avaliação abaixo do anterior, tentando uma nova mutacao ...')
198                 mutacao(geracao, 100)
199         else:
200             print(f'Mutação não considerou avaliação')
201             geracao[individuoMutacao] = copiaGeracao[individuoMutacao]
202
203 #Teste Mutacao
204
205 geracao = populacaoInicial(100)
206 print(geracao)
207

```

```

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
234         + 1]]], [y_points[melhorPrimeiraGeracao[0][i]], y_points[melhorPrimeiraGeracao[0][i
235             + 1]]], 'k-')
236
237 if i == (len(melhorPrimeiraGeracao[0]) - 2):
238     plt.plot([x_points[melhorPrimeiraGeracao[0][i + 1]], x_points[
239         melhorPrimeiraGeracao[0][0]]], [y_points[melhorPrimeiraGeracao[0][i + 1]],
240             y_points[melhorPrimeiraGeracao[0][0]]], 'k-')
241
242 plt.show()
243
244 avaliacaoP0 = FuncaoDeAvaliacao(p0)
245
246 geracao = copy.deepcopy(p0)
247 while numGeracoes > 0:
248
249     if numGeracoes % 100 == 0:
250         print(f'Geração {numGeracoes}')
251
252     novaGeracao = []
253     mutacao(geracao, taxaMutacao)
254     preservaMelhor(geracao, novaGeracao)
255     cruzamento(geracao, taxaCruzamento, novaGeracao)
256     numGeracoes = numGeracoes - 1
257     avaliacaoNovaGeracao = FuncaoDeAvaliacao(novaGeracao)
258     novaGeracao2 = sorted(zip(avaliacaoNovaGeracao, novaGeracao), reverse=True)
259     geracao = [x for _, x in novaGeracao2]
260
261
262 geracao = novaGeracao.copy()
263 novaGeracao = []
264 preservaMelhor(geracao, novaGeracao)
265
266 plt.title("Gráfico de caminho encontrado.")
267 plt.xlabel("Coordenada X")
268 plt.ylabel("Coordenada Y")
269 plt.plot(x_points, y_points, 'ro')
270
271 print(f'Distância percorrida: {FuncaoDeAvaliacaoV2(novaGeracao[0])}')
272 print(f'Duplicidade? {has_duplicates(novaGeracao[0])}')
273
274 for i in range(0, len(novaGeracao[0]) - 1):
275     plt.plot([x_points[novaGeracao[0][i]], x_points[novaGeracao[0][i + 1]]], [y_points[
276         novaGeracao[0][i]], y_points[novaGeracao[0][i + 1]]], 'k-')
277
278 if i == (len(novaGeracao[0]) - 2):
279     plt.plot([x_points[novaGeracao[0][i + 1]], x_points[novaGeracao[0][0]]], [y_points[
280         novaGeracao[0][i + 1]], y_points[novaGeracao[0][0]]], 'k-')

```

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



Ministério da Educação
UNIVERSIDADE FEDERAL DO PARANÁ
SEPT – Setor de Educação Profissional e Tecnológica
Especialização em Inteligência Artificial Aplicada



Anexo II – Código Fonte do Exercício 2

```

1  # -*- coding: utf-8 -*-
2  """Comparação modelos vetoriais.ipynb
3
4  Automatically generated by Colab.
5
6  Original file is located at
7
8      https://colab.research.google.com/github/Equipe16-IAA/IAA015/blob/ana/Compara%C3%A7%C
9      3%A3o%20modelos%20vetoriais.ipynb
10
11  # 2) Compare a representação de dois modelos vetoriais
12  ## Utilizando word2vec
13  """
14  #Gensim do Colab não funciona tem que instalar e reiniciar
15  !pip install gensim nltk scikit-learn matplotlib numpy
16
17  # --- 0. Instalação e Imports (Necessário no Colab se não estiver pré-instalado) ---
18  # Descomente e execute as linhas !pip se receber erro de módulo não encontrado
19  # !pip install gensim nltk scikit-learn matplotlib numpy
20
21  import re # Para expressões regulares (limpeza de texto)
22  import numpy as np
23  from gensim.models import Word2Vec
24  from sklearn.decomposition import PCA
25  import matplotlib.pyplot as plt
26  import matplotlib.cm as cm # Para gerar cores distintas para os pontos
27  import nltk
28  import math
29  from warnings import simplefilter
30  from scipy.cluster.hierarchy import dendrogram, linkage, ClusterWarning
31
32  #pacotes necessários para o tokenizador
33  nltk.download('punkt')
34
35  # Baixar o tokenizador 'punkt' do NLTK (necessário na primeira execução no ambiente)
36  try:
37      nltk.data.find('tokenizers/punkt')
38  except nltk.downloader.DownloadError:
39      print("Baixando o tokenizador 'punkt' do NLTK...")
40      nltk.download('punkt', quiet=True)
41      print("Download concluído.")
42
43  print("Bibliotecas importadas e NLTK pronto.")
44  print("-" * 40)
45
46  # --- 1. Definição dos Textos ---
47  # Mínimo 6 textos, com pelo menos 2 pares similares.
48  # Limitando a ~10-12 para clareza na visualização.
49  texts = [
50      "O dia amanheceu ensolarado e agradável.", # Clima 1
51      "Esperamos um fim de semana de sol e céu azul.", # Clima 2 (similar ao 1)
52      "A culinária brasileira usa muitos ingredientes frescos.", # Culinária 1
53      "Adoro comer frutas tropicais como manga e abacaxi.", # Culinária/Frutas 2
54      "Feijoada é um prato típico muito saboroso.", # Culinária 3 (similar ao 3)
55      "O gato dorme tranquilamente no sofá.", # Animais 1
56      "Cachorros precisam de passeios diários.", # Animais 2
57      "A noite está estrelada e silenciosa.", # Clima/Noite 3 (ligeiramente
58          similar ao 1/2)
59      "As receitas de família guardam tradições.", # Culinária 4 (similar ao 3/5)
60      "Os cães são animais leais e companheiros.", # Animais 3 (similar ao 7)
61  ]
62
63  # Garantir que não exceda um limite razoável para o plot
64  max_texts_to_plot = 12
65  if len(texts) > max_texts_to_plot:

```

```

65     print(f"Limitando a {max_texts_to_plot} textos para visualização.")
66     texts = texts[:max_texts_to_plot]
67
68     print(f"Textos de entrada ({len(texts)}):")
69     for i, t in enumerate(texts):
70         print(f"{i}: {t}")
71     print("-" * 40)
72
73     nltk.download('punkt_tab')
74
75     # --- 2. Pré-processamento ---
76     def preprocess_text(text):
77         """Limpa e tokeniza o texto."""
78         # Converte para minúsculas
79         text = text.lower()
80         # Remove pontuações básicas e números (opcional, mas ajuda a focar nas palavras)
81         text = re.sub(r'[^w\s]', '', text) # Remove pontuação
82         text = re.sub(r'\d+', '', text)     # Remove números
83         # Tokeniza (divide em palavras)
84         tokens = nltk.word_tokenize(text, language='portuguese') # Especificar idioma ajuda
85         # Remove palavras vazias (stopwords) - Opcional, pode ou não ajudar com poucos dados
86         from nltk.corpus import stopwords
87         nltk.download('stopwords') # Se for usar
88         stop_words = set(stopwords.words('portuguese'))
89         tokens = [word for word in tokens if word not in stop_words and len(word) > 1] #
90         # Remove stopwords e palavras de 1 letra
91         tokens = [word for word in tokens if len(word) > 1] # Apenas remove palavras de 1
92         # letra
93         return tokens
94
95     tokenized_texts = [preprocess_text(t) for t in texts]
96
97     print("Textos Tokenizados (Exemplo do primeiro):")
98     print(tokenized_texts[0])
99     print("-" * 40)
100
101     # --- 3. Treinamento do Modelo Word2Vec ---
102     # Parâmetros importantes para datasets pequenos:
103     # min_count=1: Inclui todas as palavras. Essencial aqui.
104     # epochs: Aumentar o número de épocas de treino.
105     # vector_size: Dimensão dos vetores (100 é comum, mas 50 pode ser suficiente aqui).
106     # window: Contexto de palavras vizinhas.
107     # sg=1: Usa Skip-gram (geralmente melhor para datasets menores).
108
109     vector_dim = 50 # Dimensão dos vetores de palavras
110     w2v_model = Word2Vec(sentences=tokenized_texts,
111                          vector_size=vector_dim,
112                          window=3,           # Janela menor pode ser melhor para frases curtas
113                          min_count=1,        # Crucial para datasets pequenos
114                          workers=4,          # Número de threads (ajuste conforme CPU
115                          disponível no Colab)
116                          sg=1,               # Skip-gram
117                          epochs=100)        # Mais épocas para compensar poucos dados
118
119     print("Modelo Word2Vec treinado.")
120     print(f"Tamanho do vocabulário: {len(w2v_model.wv.index_to_key)}")
121     # Exemplo: Palavras mais similares a 'sol' (se 'sol' estiver no vocabulário)
122     try:
123         similar_words = w2v_model.wv.most_similar('sol', topn=3)
124         print(f"Palavras mais similares a 'sol': {similar_words}")
125     except KeyError:
126         print("Palavra 'sol' não encontrada no vocabulário (corpus muito pequeno).")
127     print("-" * 40)
128
129     # --- 4. Cálculo dos Vetores de Sentenças ---
130     # Estratégia: Média dos vetores das palavras na sentença

```



```

129 def get_sentence_vector(tokens, model, vector_size):
130     """Calcula o vetor médio para uma lista de tokens."""
131     vector = np.zeros(vector_size)
132     count = 0
133     for word in tokens:
134         if word in model.wv: # Checa se a palavra está no vocabulário do modelo
135             vector += model.wv[word]
136             count += 1
137     if count > 0:
138         vector /= count # Calcula a média
139     # else: # Opcional: Lidar com sentenças onde nenhuma palavra está no vocabulário
140     #     print(f"Aviso: Sentença com tokens '{tokens}' não possui palavras no
141     #           vocabulário.")
142     return vector
143 # Calcula o vetor para cada texto tokenizado
144 sentence_vectors = np.array([get_sentence_vector(tokens, w2v_model, vector_dim)
145                             for tokens in tokenized_texts])
146
147 print(f"Vetores das sentenças calculados. Shape: {sentence_vectors.shape}")
148 # Verificar se há vetores nulos (indicativo de problemas)
149 if np.any(np.all(sentence_vectors == 0, axis=1)):
150     print("AVISO: Uma ou mais sentenças resultaram em vetor nulo (nenhuma palavra no
151           vocabulário?).")
152 print("-" * 40)
153
154 # --- 5. Redução de Dimensionalidade com PCA ---
155 # Reduzir os vetores de 'vector_dim' dimensões para 2 dimensões para plotagem
156
157 pca = PCA(n_components=3, random_state=42) # random_state para reprodutibilidade
158 pca_result = pca.fit_transform(sentence_vectors)
159
160 print(f"Vetores reduzidos para 2D com PCA. Shape: {pca_result.shape}")
161 print("Componentes Principais (x, y, z) para cada sentença:")
162
163 pca_result = pca_result * 100 # Multiplicado por 100 para melhor visualização nos
164 # gráficos.
165
166 for i, (x, y, z) in enumerate(pca_result):
167     print(f"Texto {i}: ({x:.4f}, {y:.4f}, {z:.4f})")
168 print("-" * 40)
169
170 # --- 6. Visualização dos Vetores 2D ---
171 #
172 plt.figure(figsize=(12, 9)) # Ajuste o tamanho conforme necessário
173
174 # Gerar cores distintas para cada ponto usando um colormap
175 colors = cm.rainbow(np.linspace(0, 1, len(texts)))
176
177 # Plotar cada ponto (vetor 2D da sentença)
178 for i, (x, y, z) in enumerate(pca_result):
179     plt.scatter(x, y, color=colors[i], s=150, alpha=0.8, edgecolors='k')
180     # Adicionar anotação (número do texto) ligeiramente deslocado do ponto
181     plt.text(x + 0.1, y + 0.1, str(i), fontsize=10, ha='left', va='bottom')
182
183 # Variância explicada
184 explained_variance_ratio = pca.explained_variance_ratio_
185 total_explained_variance = explained_variance_ratio.sum() * 100
186
187 plt.title(f'Visualização 2D dos Vetores de Sentenças (Word2Vec + PCA)\nVariância Total
188 Explicada: {total_explained_variance:.1f}%', fontsize=16)
189 plt.xlabel(f'Componente Principal 1 ({explained_variance_ratio[0]*100:.1f}%
190 variância)', fontsize=12)
191 plt.ylabel(f'Componente Principal 2 ({explained_variance_ratio[1]*100:.1f}%
192 variância)', fontsize=12)
193 plt.grid(True, linestyle='--', alpha=0.5)
194
195 legend_handles = [plt.scatter([], [], color=colors[i], label=f'{i}: {texts[i][:50]}...')]

```

```

190         for i in range(len(texts))]
191 plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(-0.05, -0.5),
192           fontsize='medium', title="Textos", title_fontsize='large', borderaxespad=1.)
193
194 plt.margins(0.1)
195 plt.axhline(0, color='grey', lw=0.5)
196 plt.axvline(0, color='grey', lw=0.5)
197
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
204 das palavras) após a redução com PCA.")
205 print("Textos com conteúdo semanticamente similar (segundo o modelo Word2Vec) devem
206 aparecer mais próximos uns dos outros.")
207 print("Com um corpus pequeno, as relações capturadas pelo Word2Vec podem ser fracas ou
208 baseadas em coocorrências simples de palavras.")
209
210 centro = pca_result.mean(axis=0)
211 v_centro = pca_result - centro
212
213 print('centro', '\n', centro)
214 print( np.allclose(pca_result, v_centro+centro))
215
216 # obter o plano para projeção ortogonal
217 U,s,Vh = np.linalg.svd(v_centro)
218 print('Variancia: ', np.square(s) / np.square(s).sum() )
219 W2 = Vh.T[:, :2]
220 v1_plano = Vh.T[:, 0]
221 v2_plano = Vh.T[:, 1]
222 projetados2d = v_centro.dot(W2)
223 print(projetados2d)
224 # calculando o plano de projeção
225 menor = np.min(pca_result,axis=0)-1
226 maior = np.min(pca_result,axis=0)+1
227 x1s = np.linspace(menor[0], maior[0], 10)
228 y1s = np.linspace(menor[1], maior[1], 10)
229 z1s = np.linspace(menor[2], maior[2], 10)
230 C = Vh
231 R = C.T.dot(C)
232 x1, x2 = np.meshgrid(x1s, y1s)
233 z = (R[0, 2] * x1 + R[1, 2] * x2) / (1 - R[2, 2])
234 #
235 # tentando visualizar as projeções no espaço 3d
236 fig = plt.figure(figsize=(12, 9)) # Ajuste o tamanho conforme necessário
237 ax = fig.add_subplot(111, projection='3d')
238 ax.plot_surface(x1, x2, z, alpha=0.3, color="k")
239 ax.view_init(elev=10, azim=60)
240 # pontos originais
241 for v in pca_result:
242     xs = np.linspace(0, v[0], 10)
243     ys = np.linspace(0, v[1], 10)
244     zs = np.linspace(0, v[2], 10)
245     plt.plot(xs,ys,zs=zs)
246
247 plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(-0.05, -0.12),
248           fontsize='medium', title="Textos", title_fontsize='large', borderaxespad=1.)
249 plt.title('Vetores e plano calculado para projeção', fontsize=16)
250 plt.show()
251
252 fig = plt.figure(figsize=(12, 9))
253 ax = fig.add_subplot(111, aspect='equal')
254 # pontos projetados
255 for v in pca_result:
256     xs = np.linspace(0, v[0], 10)

```

```

254     ys = np.linspace(0, v[1], 10)
255     ax.plot(xs,ys)
256
257     ax.arrow(0, 0, 0, 1, head_width=0.05, \
258             length_includes_head=True, head_length=0.1, fc='k', ec='k')
259     ax.arrow(0, 0, 1, 0, head_width=0.05, \
260             length_includes_head=True, head_length=0.1, fc='k', ec='k')
261     ax.grid(True)
262     #plt.xscale('log')
263     #plt.yscale('log')
264     #plt.xlim(-0.05,0.05)
265     #plt.ylim(-0.05,0.05)
266     plt.legend(handles=legend_handles, loc='center left', bbox_to_anchor=(-0.05, -0.23),
267               fontsize='medium', title="Textos", title_fontsize='large', borderaxespad=1.)
268     plt.title('Projeção no plano', fontsize=16)
269     plt.show()
270
271     def vectorDistance(v1,v2):
272         soma = 0
273         for i in range(0,len(v1)):
274             d = v1[i]-v2[i]
275             soma = soma + d*d
276         return math.sqrt(soma)
277
278     def calcDistance(mat):
279         nrows = len(mat)
280         ncols = len(mat[0])
281         res = np.zeros( (nrows,nrows) )
282         for i in range(0,nrows):
283             m1 = mat[i]
284             for j in range(0,nrows):
285                 if i!=j:
286                     m2 = mat[j]
287                     res[i][j] = vectorDistance(m1,m2)
288                 else:
289                     res[i][j] = 0
290         return res
291
292     d = calcDistance(pca_result)
293     d1 = calcDistance(projetados2d)
294     print('original')
295     print(d)
296     print('projetada')
297     print(d1)
298
299
300     simplefilter("ignore", ClusterWarning)
301     Z1 = linkage(d, 'single')
302     fig = plt.figure()
303     dn = dendrogram(Z1)
304     plt.show()
305     Z2 = linkage(d1, 'single')
306     fig = plt.figure()
307     dn = dendrogram(Z2)
308     plt.show()

```