

Francisco Teixeira Rocha Aragão

2021031726

Tp1 - Processamento de Linguagem Natural

Objetivo: Testar diferentes variações de parametros para utilização do modelo Word2vec para a tarefa de verificação de analogias utilizando operações de subtração e adição de vetores.

Análise dos resultados

O presente trabalho busca testar diferentes variações de parâmetros para construção do modelo Word2Vec. Dessa forma, o foco é testar o impacto em alterar alguns parâmetros importantes, como:

- Implementação de Skip-gram ou CBOW
- Tamanho da janela de contexto
- Tamanho dos embeddings
- Número de iterações (epochs)

Desse modo, foi usado o arquivo 'text8' para avaliação do modelo Word2Vec, que foi treinado com o arquivo 'questions-words.txt'. O arquivo 'questions-words.txt' contém analogias do tipo "a é para b, assim como c é para d", e o objetivo é verificar se o modelo consegue encontrar a palavra d, dado a, b e c (executando assim operações vetoriais de soma e subtração para encontrar a palavra d).

Os resultados obtidos podem ser vistos ao final do notebook para diferentes iterações, testando uma combinação de diferentes parâmetros. dito isso, o melhor resultado obtido foram com os parâmetros:

- Implementação de Skip-gram
- 10 épocas
- Tamanho dos embeddings como 3
- Tamanho da janela de contexto como 25
- Média de erro de 0.114

Percebe-se que de modo geral para todos os conjuntos de instâncias realizadas, a implementação do Skip-Gram obteve melhores resultados em comparação ao Cbow. O mesmo vale para o tamanho dos embeddings em que conforme o tamanho diminua, melhor ficava o valor do erro encontrado. Isso é válido pensando que o corpus utilizado para treino é relativamente pequeno, e portanto, embeddings menores são mais eficazes visto que menos informação precisa ser codificada. Também é válido dizer que como o corpus apresenta informações de diferentes contextos, com menores tamanhos de embedding é possível generalizar melhor a informação coletada.

Sobre a janela de contexto, percebe-se no geral que melhores resultados foram obtidos com valores mais altos. Isso é válido na tarefa de classificação de analogias em

que é importante observar o uso de conjunto de palavras para entender o contexto em que a palavra está inserida. Mostrando assim os bons resultados vistos na prática. Sobre o número de épocas pequeno, como o tamanho do corpus é pequeno, não é necessário muitas épocas para treinar o modelo, visto que rapidamente é possível aprender boas representações de palavras.

Analogamente aos resultados obtidos, é possível de se comentar sobre as execuções negativas também:

- Implementação de Cbow
- 3 épocas
- Tamanho dos embeddings como 50
- Tamanho da janela de contexto como 5
- Média de erro de 0.90

O mesmo vale para outros testes feitos.

- Implementação de Cbow
- 5 épocas
- Tamanho dos embeddings como 200
- Tamanho da janela de contexto como 5
- Média de erro de 0.8978691275613163

Desse modo, um possível fator impactante foi o algoritmo Cbow, presente em todos os piores resultados. Mesmo que se observe a tendência do Cbow performar melhor com mais épocas e menores tamanho de vetor, o impacto ainda foi grande de se utilizar o algoritmo, sendo essa combinação de parâmetros não performática para a tarefa em questão.

```
In [1]: from gensim.models import Word2Vec
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.model_selection import ParameterGrid
```

```
In [12]: # abrindo arquivo de corpus
with open('text8') as f:
    data = f.read()
```

```
In [13]: print(data[0:12])

anarchism o
```

```
In [14]: # organizando entrada, tokenizando em sentenças de tamanho 50 (valor arbitrario)
data_sentences = []
sentences = []
for i in data.split():
    sentences.append(i)
```

```
if len(sentences) == 50:
    data_sentences.append(sentences)
    sentences = []
```

```
In [15]: print(data_sentences[0:5])
```

```
[['anarchism', 'originated', 'as', 'a', 'term', 'of', 'abuse', 'first', 'used', 'against', 'early', 'working', 'class', 'radicals', 'including', 'the', 'diggers', 'of', 'the', 'english', 'revolution', 'and', 'the', 'sans', 'culottes', 'of', 'the', 'french', 'revolution', 'whilst', 'the', 'term', 'is', 'still', 'used', 'in', 'a', 'pejorative', 'way', 'to', 'describe', 'any', 'act', 'that', 'used', 'violent', 'means', 'to', 'destroy', 'the'], ['organization', 'of', 'society', 'it', 'has', 'also', 'been', 'taken', 'up', 'as', 'a', 'positive', 'label', 'by', 'self', 'defined', 'anarchists', 'the', 'word', 'anarchism', 'is', 'derived', 'from', 'the', 'greek', 'without', 'archons', 'ruler', 'chief', 'king', 'anarchism', 'as', 'a', 'political', 'philosophy', 'is', 'the', 'belief', 'that', 'rulers', 'are', 'unnecessary', 'and', 'should', 'be', 'abolished', 'although', 'there', 'are', 'differing'], ['interpretations', 'of', 'what', 'this', 'means', 'anarchism', 'also', 'refers', 'to', 'related', 'social', 'movements', 'that', 'advocate', 'the', 'elimination', 'of', 'authoritarian', 'institutions', 'particularly', 'the', 'state', 'the', 'word', 'anarchy', 'as', 'most', 'anarchists', 'use', 'it', 'does', 'not', 'imply', 'chaos', 'nihilism', 'or', 'anomie', 'but', 'rather', 'a', 'harmonious', 'anti', 'authoritarian', 'society', 'in', 'place', 'of', 'what', 'are', 'regarded'], ['as', 'authoritarian', 'political', 'structures', 'and', 'coercive', 'economic', 'institutions', 'anarchists', 'advocate', 'social', 'relations', 'based', 'upon', 'voluntary', 'association', 'of', 'autonomous', 'individuals', 'mutual', 'aid', 'and', 'self', 'governance', 'while', 'anarchism', 'is', 'most', 'easily', 'defined', 'by', 'what', 'it', 'is', 'against', 'anarchists', 'also', 'offer', 'positive', 'visions', 'of', 'what', 'they', 'believe', 'to', 'be', 'a', 'truly', 'free', 'society'], ['however', 'ideas', 'about', 'how', 'an', 'anarchist', 'society', 'might', 'work', 'vary', 'considerably', 'especially', 'with', 'respect', 'to', 'economics', 'there', 'is', 'also', 'disagreement', 'about', 'how', 'a', 'free', 'society', 'might', 'be', 'brought', 'about', 'origins', 'and', 'predecessors', 'kropotkin', 'and', 'others', 'argue', 'that', 'before', 'recorded', 'history', 'human', 'society', 'was', 'organized', 'on', 'anarchist', 'principles', 'most', 'anthropologists', 'follow']]
```

```
In [ ]: # preparando hiperparametros para serem usados (dados da última iteração para testes)
```

```
hiperparameters = {
    'vector_size': [3, 5, 7],
    'sg': [0, 1], # 1 = skip-gram, 0 = CBOW
    'window': [15, 20, 25],
    'epochs': [5, 8, 10],
}

grid = list(ParameterGrid(hiperparameters))
```

```
In [ ]: # preparando informações para teste que estão presentes no arquivo questions-words.txt
```

```
test_vectors = []
target_words = []
with open("questions-words.txt") as f:
    for line in f:
        if line.startswith(":"):
            continue

        line = line.strip().lower().split(' ')

        if len(line) != 4:
            continue

        # seleciono informações de teste e agrupo as palavras de analogias em listas
```

```

        test_vectors.append(line[0:3])
        target_words.append(line[3])

print(len(test_vectors))

```

19544

```

In [ ]: # treinando modelo word2vec com dados de entrada, variando os hiperparametros e calculando erro médio
# a melhor e pior configuração são salvas e impressas ao final
best_config = {}
min_avg_error = 50000

worst_config = {}
max_avg_error = 0

results = []

# dados estão no formato: palavra1 palavra2 palavra3 palavra4
# a ideia é que palavra1 - palavra2 + palavra3 = palavra4
# assim é calculada a similaridade de cosseno entre o resultado de palavra1 - palavra2 + palavra3 com a palavra4
# e assim o erro é calculado como 1 - similaridade
for parameter_configuration in grid: # testo todas as combinações de hiperparametros
    total_error = 0.0
    count_test_words_in_vocab = 0

    model = Word2Vec(sentences=data_sentences, vector_size=parameter_configuration['vector_size'], window=parameter_configuration['window'], sg=parameter_configuration['sg'])

    for idx in range(len(test_vectors)):

        word_a, word_b, word_c = test_vectors[idx]
        target = target_words[idx]

        if all(word in model.wv for word in [word_a, word_b, word_c, target]): # vejo se as palavras estão no vocabulário

            analogy_vector = model.wv[word_a] - model.wv[word_b] + model.wv[word_c]

            similarity = cosine_similarity([analogy_vector], [model.wv[target]])[0][0] # comparo o target com a palavra retornada pelo modelo

            # melhor erro é proximo de 0
            error = 1 - similarity
            total_error += error
            count_test_words_in_vocab += 1
            """ print(f"Analogy: {word_a} - {word_b} + {word_c} = {target}")
            print(f"Similarity with {target}: {similarity:.4f}, Error: {error:.4f}")
            print() """

    # calculo o erro médio de todas as analogias realizadas
    average_error = total_error / count_test_words_in_vocab

```

```

""" print(f"\nAverage Analogy Error: {average_error}") """

results.append((parameter_configuration, average_error))

if average_error < min_avg_error:
    best_config = parameter_configuration
    min_avg_error = average_error
    model.save("best_model")

if average_error > max_avg_error:
    worst_config = parameter_configuration
    max_avg_error = average_error

print(f"\nBest Configuration: {best_config}")
print(f"Min Average Error: {min_avg_error}")

print(f"\nWorst Configuration: {worst_config}")
print(f"Max Average Error: {max_avg_error}")

```

Best Configuration: {'epochs': 10, 'sg': 1, 'vector_size': 3, 'window': 25}
Min Average Error: 0.11449590156785465

Worst Configuration: {'epochs': 5, 'sg': 0, 'vector_size': 5, 'window': 15}
Max Average Error: 0.7583584528408859

```

In [9]: # salvando resultados em um arquivo de texto
results.sort(key=lambda x: x[1])

with open("results5.txt", "w") as f:
    for result in results:
        f.write(f"{result[0]} -> {result[1]}\n")

```

```
In [19]: # imprimindo os resultados -> piores resultados
with open("results1.txt", "r") as f:
    lines = f.readlines()

    # 5 melhores resultados
    print("Melhores resultados - Conjunto 1 de teste")
    for line in lines[:5]:
        print(line.strip())

    print()

    # 5 piores resultados
    print("Piores resultados - Conjunto 1 de teste")
    for line in lines[-5:]:
        print(line.strip())
```

Melhores resultados - Conjunto 1 de teste

```
{'epochs': 5, 'sg': 1, 'vector_size': 50, 'window': 10} -> 0.55538132695304
{'epochs': 10, 'sg': 1, 'vector_size': 50, 'window': 10} -> 0.5600422201176011
{'epochs': 20, 'sg': 1, 'vector_size': 50, 'window': 10} -> 0.5707512863907391
{'epochs': 5, 'sg': 1, 'vector_size': 50, 'window': 5} -> 0.5774646656031192
{'epochs': 10, 'sg': 1, 'vector_size': 50, 'window': 5} -> 0.584635035953901
```

Piores resultados - Conjunto 1 de teste

```
{'epochs': 5, 'sg': 0, 'vector_size': 200, 'window': 10} -> 0.8830664120799637
{'epochs': 5, 'sg': 0, 'vector_size': 200, 'window': 2} -> 0.8836903947134177
{'epochs': 5, 'sg': 0, 'vector_size': 50, 'window': 5} -> 0.8844702592285507
{'epochs': 5, 'sg': 0, 'vector_size': 100, 'window': 5} -> 0.8917970729349668
{'epochs': 5, 'sg': 0, 'vector_size': 200, 'window': 5} -> 0.8978691275613163
```

```
In [20]: # imprimindo os resultados
with open("results2.txt", "r") as f:
    lines = f.readlines()

    # 5 melhores resultados
    print("Melhores resultados - Conjunto 2 de teste")
    for line in lines[:5]:
        print(line.strip())

    print()

    # 5 piores resultados
    print("Piores resultados - Conjunto 2 de teste")
    for line in lines[-5:]:
        print(line.strip())
```

```
Melhores resultados - Conjunto 2 de teste
{'epochs': 5, 'sg': 1, 'vector_size': 15, 'window': 10} -> 0.38703091299607423
{'epochs': 8, 'sg': 1, 'vector_size': 15, 'window': 10} -> 0.38780395781142457
{'epochs': 3, 'sg': 1, 'vector_size': 15, 'window': 10} -> 0.39179346938399207
{'epochs': 8, 'sg': 1, 'vector_size': 15, 'window': 5} -> 0.40069795448117645
{'epochs': 5, 'sg': 1, 'vector_size': 15, 'window': 5} -> 0.4125249547295823
```

Piores resultados - Conjunto 2 de teste

```
{'epochs': 5, 'sg': 0, 'vector_size': 50, 'window': 5} -> 0.8870883486498904
{'epochs': 3, 'sg': 0, 'vector_size': 30, 'window': 10} -> 0.8881735312187637
{'epochs': 3, 'sg': 0, 'vector_size': 50, 'window': 10} -> 0.8969098171314653
{'epochs': 3, 'sg': 0, 'vector_size': 30, 'window': 5} -> 0.8986601603363565
{'epochs': 3, 'sg': 0, 'vector_size': 50, 'window': 5} -> 0.9022114387767982
```

```
In [21]: # imprimindo os resultados
with open("results3.txt", "r") as f:
    lines = f.readlines()

    # 5 melhores resultados
    print("Melhores resultados - Conjunto 3 de teste")
    for line in lines[:5]:
        print(line.strip())

    print()

    # 5 piores resultados
    print("Piores resultados - Conjunto 3 de teste")
    for line in lines[-5:]:
        print(line.strip())
```

Melhores resultados - Conjunto 3 de teste

```
{'epochs': 8, 'sg': 1, 'vector_size': 10, 'window': 15} -> 0.2954936634862853
{'epochs': 5, 'sg': 1, 'vector_size': 10, 'window': 15} -> 0.30072306892771716
{'epochs': 3, 'sg': 1, 'vector_size': 10, 'window': 15} -> 0.31008412199108054
{'epochs': 3, 'sg': 1, 'vector_size': 10, 'window': 10} -> 0.31038375133059615
{'epochs': 5, 'sg': 1, 'vector_size': 10, 'window': 10} -> 0.3192339427661957
```

Piores resultados - Conjunto 3 de teste

```
{'epochs': 5, 'sg': 0, 'vector_size': 30, 'window': 10} -> 0.8634177259757336
{'epochs': 5, 'sg': 0, 'vector_size': 30, 'window': 5} -> 0.8660574117078195
{'epochs': 3, 'sg': 0, 'vector_size': 30, 'window': 15} -> 0.8798409598996422
{'epochs': 3, 'sg': 0, 'vector_size': 30, 'window': 10} -> 0.8860237199068711
{'epochs': 3, 'sg': 0, 'vector_size': 30, 'window': 5} -> 0.8976718970943462
```

```
In [22]: # imprimindo os resultados
with open("results4.txt", "r") as f:
    lines = f.readlines()
```

```

# 5 melhores resultados
print("Melhores resultados - Conjunto 4 de teste")
for line in lines[:5]:
    print(line.strip())

print()

# 5 piores resultados
print("Piores resultados - Conjunto 4 de teste")
for line in lines[-5:]:
    print(line.strip())

```

Melhores resultados - Conjunto 4 de teste

```

{'epochs': 8, 'sg': 1, 'vector_size': 5, 'window': 20} -> 0.19574313511324876
{'epochs': 5, 'sg': 1, 'vector_size': 5, 'window': 20} -> 0.19791436021546593
{'epochs': 3, 'sg': 1, 'vector_size': 5, 'window': 20} -> 0.199327464784689
{'epochs': 5, 'sg': 1, 'vector_size': 5, 'window': 15} -> 0.20131631473280664
{'epochs': 8, 'sg': 1, 'vector_size': 5, 'window': 15} -> 0.20434490671463792

```

Piores resultados - Conjunto 4 de teste

```

{'epochs': 3, 'sg': 0, 'vector_size': 5, 'window': 10} -> 0.8056594325398947
{'epochs': 3, 'sg': 0, 'vector_size': 10, 'window': 20} -> 0.8077876065773347
{'epochs': 3, 'sg': 0, 'vector_size': 7, 'window': 15} -> 0.8118513876709923
{'epochs': 3, 'sg': 0, 'vector_size': 10, 'window': 15} -> 0.8121803097646034
{'epochs': 3, 'sg': 0, 'vector_size': 10, 'window': 10} -> 0.8218598262022079

```

In [23]: *# imprimindo os resultados -> melhores resultados*

```

with open("results5.txt", "r") as f:
    lines = f.readlines()

# 5 melhores resultados
print("Melhores resultados - Conjunto 5 de teste")
for line in lines[:5]:
    print(line.strip())

print()

# 5 piores resultados
print("Piores resultados - Conjunto 5 de teste")
for line in lines[-5:]:
    print(line.strip())

```


Melhores resultados - Conjunto 5 de teste

```
{'epochs': 10, 'sg': 1, 'vector_size': 3, 'window': 25} -> 0.11449590156785465  
{'epochs': 8, 'sg': 1, 'vector_size': 3, 'window': 25} -> 0.12627258985665643  
{'epochs': 10, 'sg': 1, 'vector_size': 3, 'window': 20} -> 0.12849591586262585  
{'epochs': 8, 'sg': 1, 'vector_size': 3, 'window': 20} -> 0.13624663163420458  
{'epochs': 5, 'sg': 1, 'vector_size': 3, 'window': 25} -> 0.14409597154196624
```

Piores resultados - Conjunto 5 de teste

```
{'epochs': 5, 'sg': 0, 'vector_size': 7, 'window': 25} -> 0.737519417538554  
{'epochs': 5, 'sg': 0, 'vector_size': 7, 'window': 20} -> 0.7466540246473996  
{'epochs': 5, 'sg': 0, 'vector_size': 5, 'window': 20} -> 0.7484880196498578  
{'epochs': 5, 'sg': 0, 'vector_size': 7, 'window': 15} -> 0.7554935015382345  
{'epochs': 5, 'sg': 0, 'vector_size': 5, 'window': 15} -> 0.7583584528408859
```