Code Presentation

IA368DD_2023S1: Deep Learning aplicado a Sistemas de Buscas

Student: Marcus Vinícius Borela de Castro

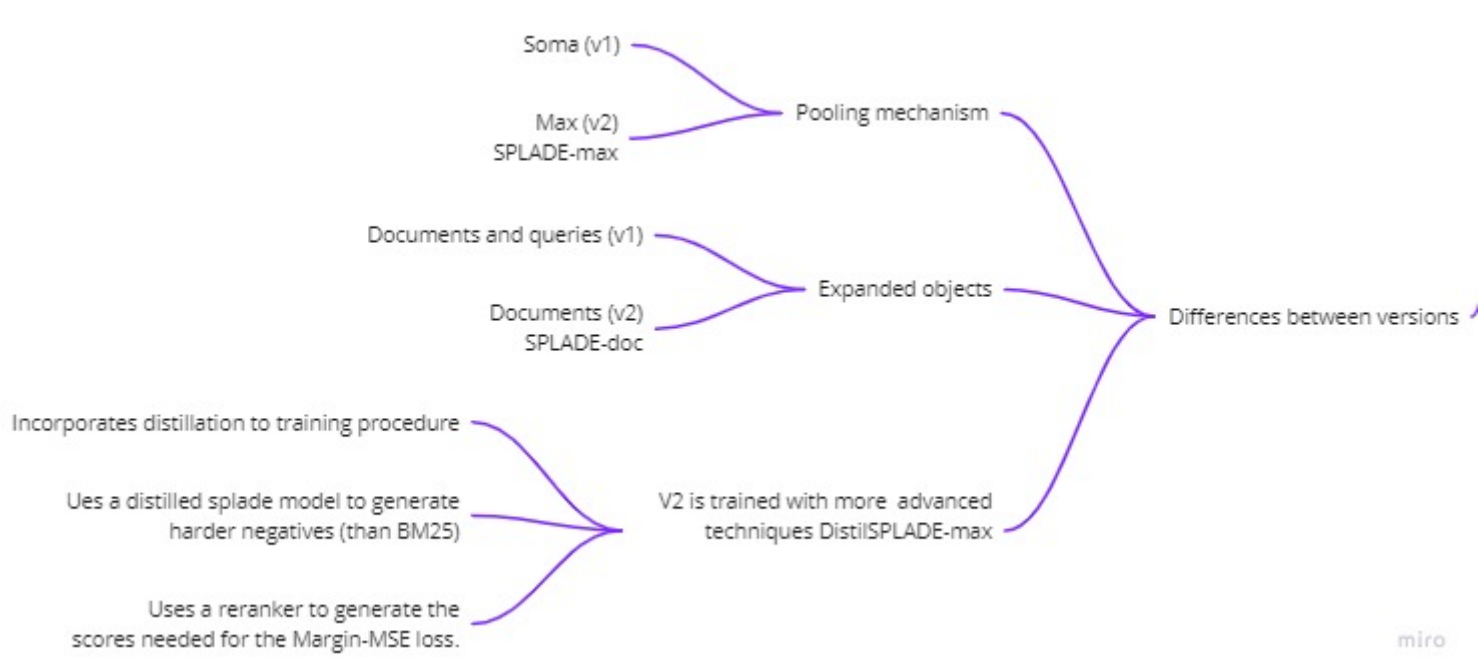
SPLADE

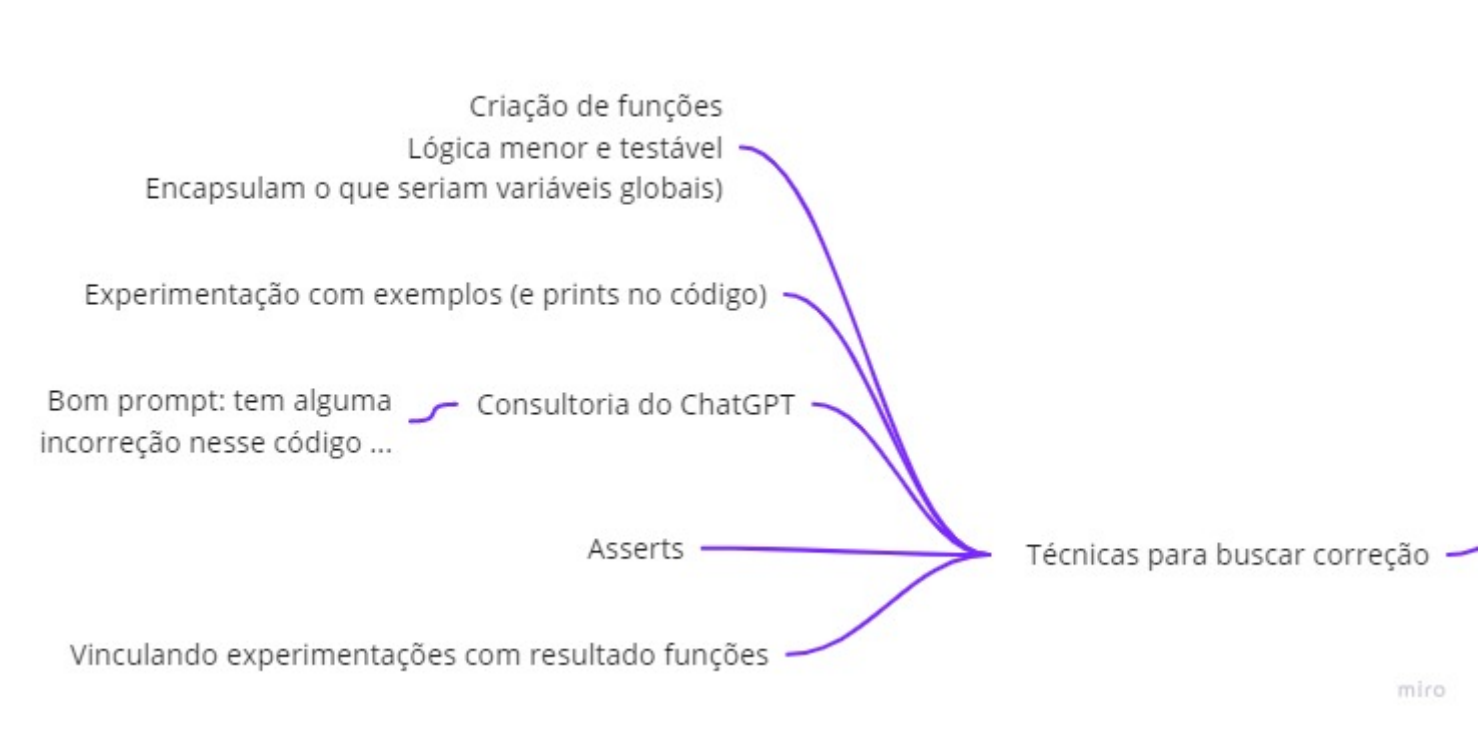
A model that both learns expansion and compression in an end-to-end manner.

The model generates a sparse BOW (the size of the vocabulary) for a text with expansion of important terms (synonyms and related, for example) and excludes unimportant terms (stop words, for example)

SPLADE
SParse **L**exical **A**n**D** **E**xpansion Model for Information Retrieval

Conceitos


Soma (v1)
Max (v2)
SPLADE-max
Pooling mechanism
Documents and queries (v1)
Documents (v2)
SPLADE-doc
Expanded objects
Differences between versions
Incorporates distillation to training procedure
Ues a distilled splade model to generate harder negatives (than BM25)
Uses a reranker to generate the scores needed for the Margin-MSE loss.
V2 is trained with more advanced techniques DistilSPLADE-max
miro


Criação de funções
Lógica menor e testável
Encapsulam o que seriam variáveis globais)
Experimentação com exemplos (e prints no código)
Bom prompt: tem alguma incorreção nesse código ...
Consultoria do ChatGPT
Asserts
Vinculando experimentações com resultado funções
Técnicas para buscar correção
miro

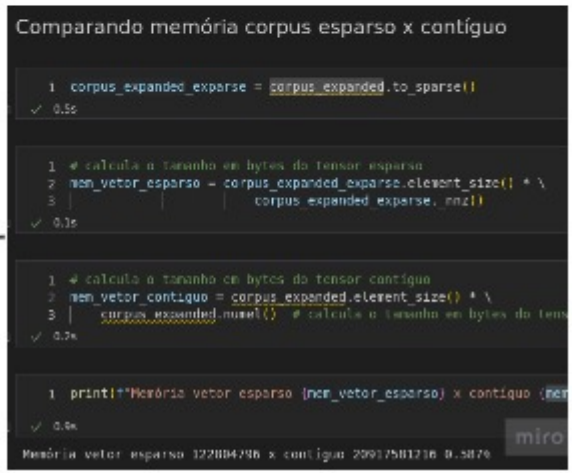
Truques do código

vetores esparsos no pytorch

Compreensão:

```
tensor(
    indices=tensor([[   0,   ...    0,          1,     ... 1]
                    [ 1996,  ...  189026,  1999, ..., 19782]]
    values=tensor([ 2.4e-02, ..., 1.1e+00,  6.2e-02, ...1.3e-01]),
    size=(2, 30522), nnz=123, layout=torch.sparse_coo)
```

economia de espaço



Problemas encontrados e soluções

Tempo inferência do corpus estava entre 2 e 3 horas

Movimentava para a cpu e realizava max na cpu.
Ao mudar para gpu o tempo caiu para a casa dos 20 minutos.

```
(...)
encoded_input_cuda = {key: value.to(device) for key, value in encoded_input.items()}
with torch.no_grad():
    model_output = parm_model(**{k:v for k,v in encoded_input_cuda.items() if k != 'special_tokens_mask'})
logits = model_output.logits.cpu()
if parm_ind_agg == 'max':
    wj, _ = torch.max(torch.log(1 + relu(logits)), dim=1)
elif parm_ind_agg == 'sum':
    wj = torch.sum(torch.log(1 + relu(logits)), dim=1)
```

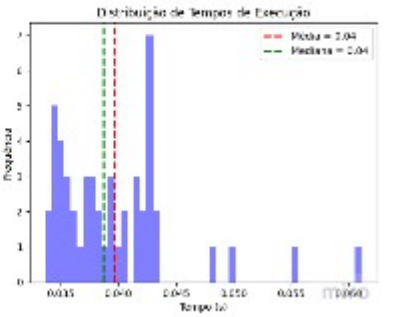
Os termos que compõem as expansões são os mesmos, seja usando max, seja usando sum (ver seção de experimentos)

Pois as tokens da expansão são as não zeradas do logits (após re

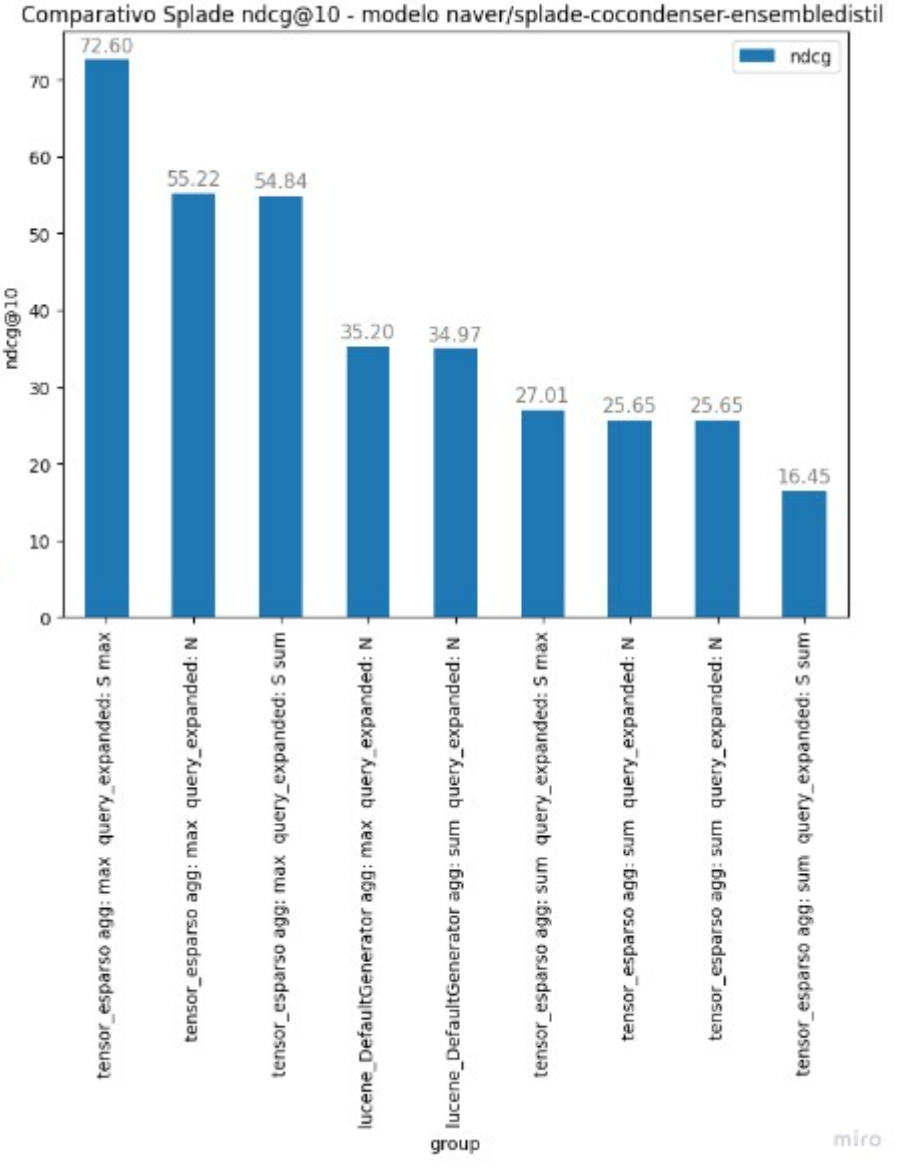
O que muda são os valores dos scores das tokens, o que pode in (score(sum) >> score(max)).

Resultados interessantes

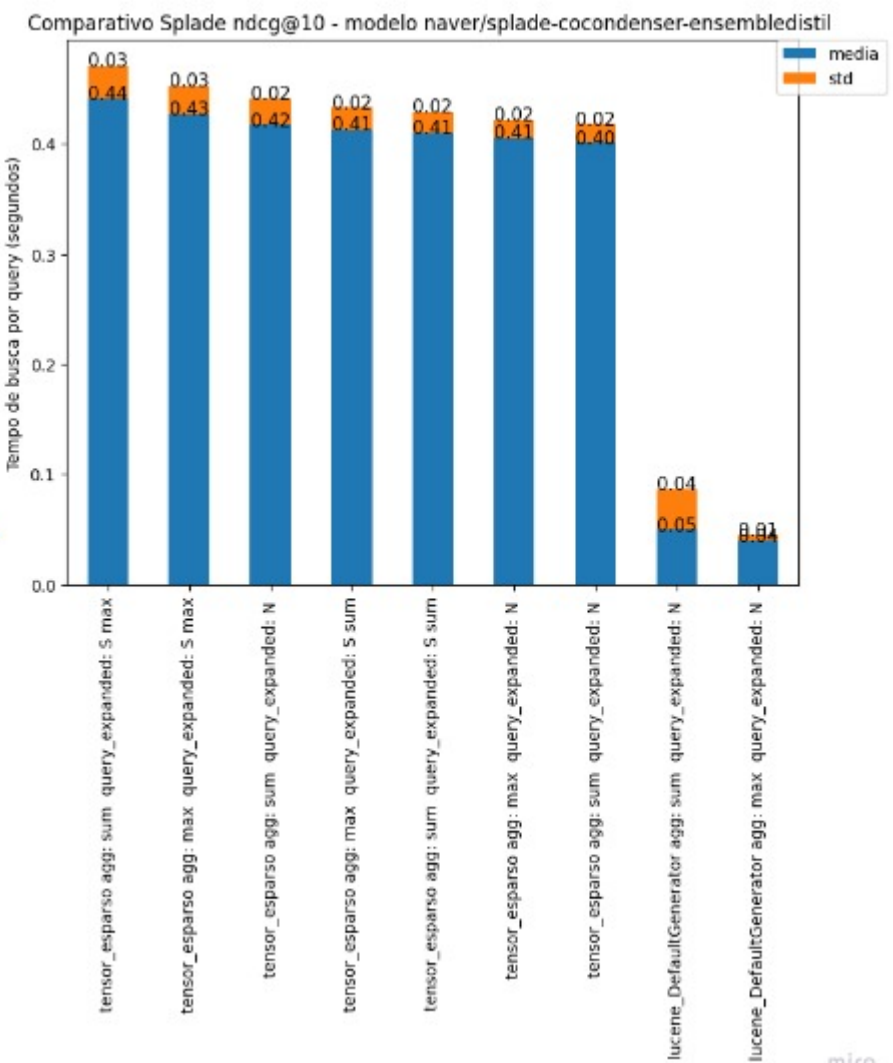
Latência muito menor usando pyserini/lucene

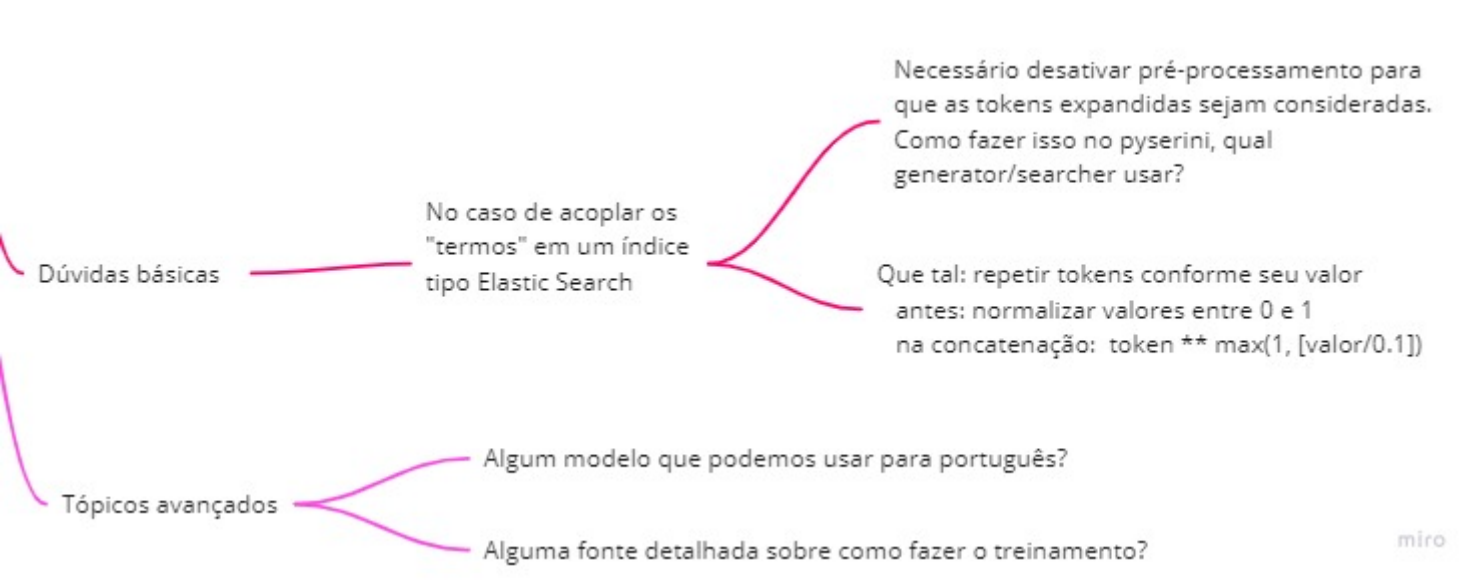


Resultado ndcg@10



Latência




Dúvidas básicas
No caso de acoplar os "termos" em um índice tipo Elastic Search
Necessário desativar pré-processamento para que as tokens expandidas sejam consideradas. Como fazer isso no pyserini, qual generator/searcher usar?
Que tal: repetir tokens conforme seu valor
antes: normalizar valores entre 0 e 1
na concatenação: token ** max(1, [valor/0.1])
Tópicos avançados
Algum modelo que podemos usar para português?
Alguma fonte detalhada sobre como fazer o treinamento?
miro