INSPER

CIÊNCIAS DA COMPUTAÇÃO

SPRINT SESSION

GRUPO ESMERALDA

REPORTE TÉCNICO DO TREINAMENTO DO CLASSIFICADOR

São Paulo, SP
2024

Sumário

1 Modelo de Classificação 1
1.1 Bases do modelo 1
1.2 Vetorização de Textos 1
1.3 O classificador 3
1.3.1 Treinamento do modelo 4

Referências 6

1 Modelo de Classificação

1.1 Bases do modelo

Este relatório apresenta o desenvolvimento e a implementação de um modelo de classificação baseado em aprendizado de máquina para classificar dúvidas entre os assuntos "Sobre o Ismart", "Processo Seletivo", "Ismart Online"e "Programa de Bolsas", ou seja, dado uma dúvida de um candidato, o modelo dirá sobre o que é essa dúvida.

Como qualquer aplicação baseada em aprendizagem de máquina, a quantidade e a qualidade dos dados fornecidos para treinamento é fundamental para o bom funcionamento do modelo. Como os dados fornecidos pela Ismart eram em relativa pouca quantidade e volume, utilizamos de técnicas de RAG para gerar um prompt para o GPT-4 gerar novas perguntas de acordo com cada classificação definida, realizando assim, um *data augmentation*, como mostrado em [Figura 1.1].

```
messages = [
    SystemMessage(content = 'Você é um analista de processos
    seletivos e me ajudará a escrever novas perguntas sintéticas
    relacionadas aos temas requeridos.'),
    HumanMessage(content = f'Com base no tema "Ismart online", e
    tomando como contexto as partes mais relevantes do edital:
    {most_similar("Ismart online")}, gere uma lista de 100
    perguntas sintéticas sobre esse tema.'),
]

chat = gpt(messages)
messages.append(chat)
chat
```

Figura 1.1 – Script utilizado para pedir novas perguntas ao GPT-4

Agora, com um banco de dados mais denso e robusto, separamos as perguntas por assuntos: "Sobre o Ismart", "Processo Seletivo", "Programa de Bolsas"e "Ismart online". Separamos as perguntas por PDFs, cada PDF de cada assunto possui todas as perguntas relacionadas àquele assunto. Fazendo uso do *Text Splitter* recursivo do *Langchain*, dividimos cada documento em chunks menores, para maior eficiência do processamento desses textos, demonstrado em [Figura 1.2].

1.2 Vetorização de Textos

O treinamento desse modelo é feito com base em representações vetorizadas de textos. É possível vetorizar chunks de textos, com cada chunk vetorizado gerando um vetor de 1536

```python
splitter = RecursiveCharacterTextSplitter(
    chunk_size = 50,
    chunk_overlap = 10,
    length_function = len
)
```

Python

```python
sobrechunks = splitter.create_documents([x.page_content for x in
sobre])
onlinechunks = splitter.create_documents([x.page_content for x in
online])
processoseletivochunks = splitter.create_documents([x.page_content
for x in processoseletivo])
bolsaschunks = splitter.create_documents([x.page_content for x in
bolsas])
```

Figura 1.2 – Separação dos documentos em chunks

dimensões, em que cada dimensão captura de certa forma um significado semântico do texto, como por exemplo, nesse espaço vetorial, a subtração de um vetor que representa a palavra "rei"por um vetor que representa a palavra "homem"e somando com um vetor que representa a palavra "mulher"resulta em um vetor que representa a palavra "rainha".

Essa técnica de vetorização também auxilia em tarefas de classificação. Outro exemplo é que um livro de matemática vetorizado estaria em um espaço totalmente diferente de um livro de português vetorizado dentro desse espaço. A primeira tentativa para classificar as dúvidas vetorizadas foi fazer uso de uma técnica de redução de dimensionalidade, o TSNE, para plotar a distribuição das dúvidas nesse espaço vetorial [Figura 1.3].

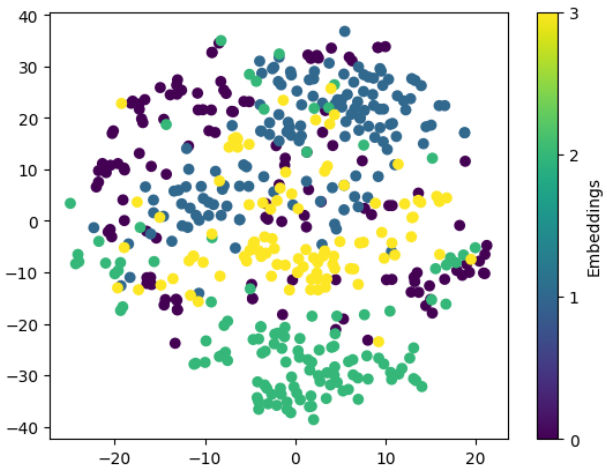

Figura 1.3 – Distribuição dos pontos dos textos vetorizados, com sua dimensão reduzida para 2 por meio de um TSNE. As cores representam os assuntos: roxo: "Sobre o Ismart"; azul: "Ismart Online"; verde: "Processo Seletivo"; amarelo: "Programa de Bolsas".

Primeiro, uma dúvida que é frequente é o por que de precisarmos utilizar uma técnica como o TSNE para visualizar essas vetorizações. Os vetores têm 1536 dimensões, então é impossível visualizar um plano com 1536 dimensões. A técnica de TSNE é feita com base na

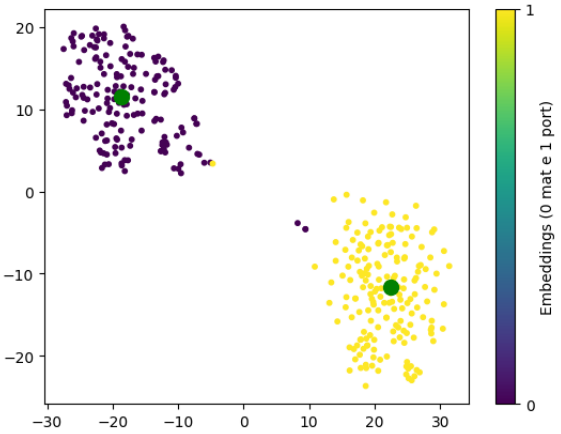

Figura 1.4 – Distribuição da vetorização de um livro de matemática e de português. Perceba que eles são bem afastados, por conta da diferença do assunto, o que não ocorre no caso das dúvidas.

análise dos eixos principais por meio de autos-vetores, diminuindo a dimensão desses pontos para 2, sendo possível assim, plotar esses vetores num plano 2D.

Saindo do adendo, é possível perceber que os pontos estão dispostos de forma não tão uniforme, não sendo possível observar um padrão. Para critérios de comparação, essa é a visualização do exemplo das vetorizações dos livros de matemática e português em [Figura 1.4].

Com isso, é possível perceber que para realizar essa classificação, será necessário treinar um modelo especificamente para essa tarefa.

1.3 O classificador

A estrutura utilizada para o classificador é uma stack de camadas Densas, que recebe como input a vetorização da pergunta realizada, e tem como output uma camada densa de 4 perceptrons com uma ativação softmax, definida pela Equação 1.1:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K \tag{1.1}$$

Que gera uma distribuição de probabilidade para cada classe de dúvida, que somadas sempre dão 1, e a classe de maior valor, é a que tem a maior probabilidade de ser a real.

Como otimizador do modelo, foi utilizado o adam, e para função de perda, utilizamos a entropia cruzadada, definida pela equação 1.2:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{1.2}$$

O objetivo dessa função é determinar a diferença entre duas distribuições de probabilidade: a gerada pela função softmax na camada de saída do modelo e a distribuição verdadeira. Por exemplo, considere duas classes, A e B, representadas pelos índices 0 e 1 de um vetor, respectivamente. Se a distribuição de saída do modelo for [0.3, 0.7] e a classe correta for B, a distribuição esperada será [0, 1], uma vez que a classe correta é sempre B.

1.3.1 Treinamento do modelo

A estrutura do modelo foi definida em 1.5.

```
X_train, X_test, y_train, y_test = train_test_split(
    allvectors, y, test_size = 0.2, random_state = 42
)

X_train, X_val, y_train, y_val = train_test_split(
    X_train, y_train, test_size = 0.2, random_state = 42
)


model = Sequential([
    Dense(64, activation='relu', kernel_regularizer=tf.keras.
    regularizers.l2(0.005), input_shape=(allvectors.shape[1],)),
    Dropout(0.5),
    Dense(32, activation='relu', kernel_regularizer=tf.keras.
    regularizers.l2(0.005)),
    Dense(4, activation='softmax')
])
```

Figura 1.5 – Arquitetura do modelo utilizado. Perceba a separação de dados para treino e teste. Também foram utilizados regularizadores l2 nas camadas, fornecendo uma penalidade em caso de gradientes muito fortes, e uma camada de dropout, desativando aletaoriamente metade dos perceptrons da camada.

O modelo foi treinado com um callback de early stopping, o monitor baseado nos dados de validação, e com uma paciência definida em 5. Foram utilizadas 100 épocas e um batch size de 32.

Após 100 épocas de treinamento, plotamos as métricas de acurácia para os dados de treinamento e validação em [Figura 1.6].

E também as perdas em [Figura 1.7].

Testando o modelo, temos resultados satisfatórios, mostrados em [Figura 1.8].

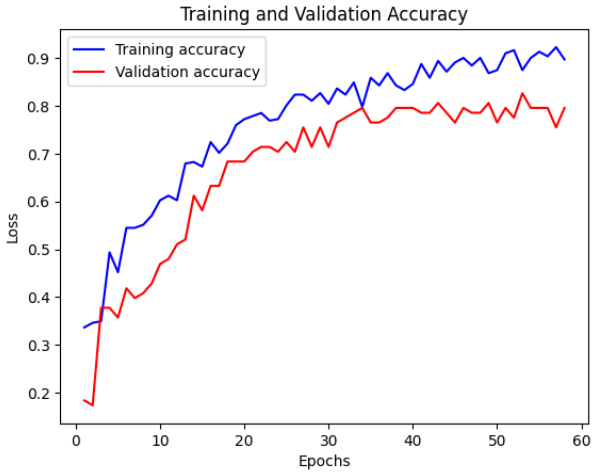

Figura 1.6 – Acurácia para os dados de treino(em azul) e validação(em vermelho).

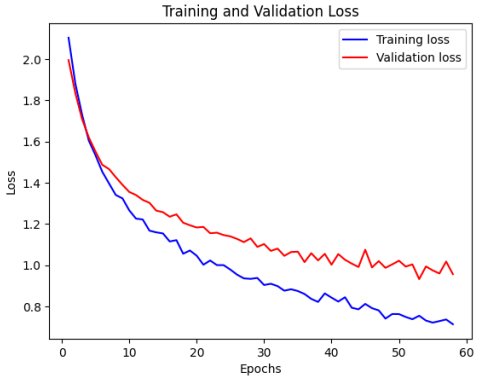

Figura 1.7 – Perdas(entropia cruzada) para os dados de treino(em azul) e validação(em vermelho).

```
inputto = np.array(embeddings.embed_documents(['ate quando posso
me inscrever ?']))
```

Python

```
prediction = model.predict(inputto, batch_size = 32)
idx = np.argmax(prediction, axis = 1)
tags[idx[0]]
```

Python

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 141ms/step
'processo seletivo'
```

Figura 1.8 – Resultado do modelo. Vetorizamos o texto da pergunta, e usamos como input para o modelo. Perceba que o output do modelo é uma distribuição de probabilidades, com cada número corresponde à probabilidade de ser à classe relacionada àquele índice. Para pegarmos o índice de maior elemento, usamos a função argmax.

Referências