UNIVERSIDADE FEDERAL DA PARAÍBA
CENTRO DE INFORMÁTICA
CURSO DE CIÊNCIA DE DADOS E INTELIGÊNCIA ARTIFICIAL

LUÍS HENRIQUE AUGUSTO DE LIMA

PROJETO FINAL DA DISCIPLINA DE PROCESSAMENTO DE LINGUAGEM NATURAL

JOÃO PESSOA
2023

O reconhecimento de entidades nomeadas (NER - *Named Entity Recognition*) é uma tarefa da área de processamento de linguagem natural que envolve a identificação e classificação de entidades significativas em um texto, como nomes de pessoas, organizações, locais, datas, entre outros. O objetivo é extrair informações específicas do texto, identificando e categorizando entidades relevantes.

Algumas das várias aplicações do NER são recuperação de informações onde é usado para extrair informações relevantes de documentos e textos para melhorar a precisão e eficiência dos sistemas de recuperação de informações, análise de sentimentos, o qual pode ser utilizado para identificar as entidades mencionadas em um texto e analisar o sentimento associado a elas, em *chatbots* e assistentes virtuais, o NER é fundamental para a compreensão de comandos e solicitações feitas por usuários em linguagem natural. Ele permite identificar entidades mencionadas em uma conversa e fornecer respostas relevantes e personalizadas, entre outros.

No presente trabalho, serão utilizados dois modelos para o reconhecimento de entidades nomeadas. Um deles será através da biblioteca *spacy*, que oferece uma ampla gama de recursos e funcionalidades para lidar com tarefas relacionadas ao processamento de textos, como *tokenização*, marcação de partes do discurso, análise sintática, reconhecimento de entidades nomeadas e muito mais. E o outro será criada utilizando a biblioteca *tensorflow*.

O objetivo do presente trabalho é utilizar um modelo de NER para gerar palavras-chave, resumos e descobrir entidades em um texto. Além disso, o modelo criado também será capaz de gerar textos com um limite máximo de palavras especificado. Essas funcionalidades são úteis em várias aplicações, como indexação automática de documentos, geração de resumos automáticos, análise de conteúdo e muito mais.

Cada modelo possui suas limitações, mas, a partir da utilização, geraram resultados satisfatórios para o que foi proposto.

O modelo de rede neural criado através do *tensorflow*, utilizado neste trabalho, embora seja capaz de aprender padrões e gerar texto de forma razoável, ele pode não ser capaz de capturar completamente a precisão contextual e a semântica das palavras. Além disso, a qualidade e a diversidade dos dados de treinamento podem afetar significativamente o desempenho do modelo. Portanto, é necessário ter cuidado ao interpretar os resultados e considerar as limitações do modelo.

A ideia inicial dos dados utilizados para o treinamento, seria utilizar quatro textos distintos, cada um gerado através do *chatGPT*, com diferentes *prompts*, mas que em geral eram “gere um texto inglês sobre X (tema), com Y caracteres”. Porém, o pré-processamento de texto basicamente iria agrupar

todo conteúdo dos quatro textos e ao final do modelo, seria gerado um novo texto baseado nisso tudo. O problema se dá na crescente falta de contexto, a partir da utilização de textos muito distintos. Sendo assim, optou-se por utilizar um grande texto com cerca de vinte mil caracteres, que falasse de um único assunto, a fim de trazer resultados mais coesos.

O pré-processamento dos dados se dá a partir da função *preprocess_data()*, que recebe um caminho de arquivo como entrada, lê, converte todo o texto em letras minúsculas e retorna o texto pré-processado.

Para a criação do modelo de rede neural, primeiramente é definido um *tokenizador*, é gerado um vocabulário a partir do mesmo, cria-se um dicionário que mapeia cada palavra única no vocabulário para um índice inteiro. A lista *input_sequences* é utilizada para armazenar sequências de treinamento, onde o texto pré-processado é dividido em linhas, *tokenizado*, e cria-se uma sequência *n_gram* que será adicionada na lista anterior.

Ainda antes de ser criado o modelo, é realizado um *padding* na lista *input_sequence*, a fim de normalizar o comprimento das sequências de treinamento e a separação dos dados de entrada e saída, além de converter as *labels* em vetores *one-hot* (os quais são uma forma de representar categorias ou classes de forma binária, onde cada categoria é representada por um vetor de zeros com um único elemento igual a 1. Em um vetor *one-hot*, o valor 1 está presente na posição correspondente à categoria atual, e todas as outras posições contêm o valor 0).

Finalmente, é criado o modelo de rede neural utilizando *Keras/TensorFlow*. Inicialmente, é criado um objeto *Sequential*, que representa o tipo mais simples de modelo de rede neural sequencial, onde as camadas são empilhadas uma após a outra. A primeira camada adicionada ao modelo é a camada *Embedding*. Essa camada é responsável por mapear as palavras em vetores de *embedding*. Os parâmetros *total_words*, *embedding_dim* e *input_length* são passados para a camada. A segunda camada adicionada ao modelo é uma camada LSTM (*Long Short-Term Memory*), a qual ajuda a capturar dependências de longo prazo nas sequências de entrada. O parâmetro 150 especifica o número de unidades LSTM (ou seja, o tamanho do estado oculto da camada LSTM). A terceira camada adicionada é uma camada *Dense*, que é uma camada completamente conectada (*fully connected*). Essa camada possui um número de neurônios igual a *total_words* e usa a função de ativação *softmax* - usada para converter as saídas da camada em probabilidades, garantindo que a soma de todas as saídas seja igual a 1. Cada saída representa a probabilidade de uma palavra específica ser a próxima palavra na sequência. O método *compile*() é chamado para compilar o modelo, onde é passado os seguintes parâmetros: *loss='categorical_crossentropy'*, que é a função de perda utilizada para calcular a diferença entre as saídas previstas e

as saídas reais (*one-hot encoded*), e o parâmetro *optimizer='adam'*, que é o otimizador utilizado para ajustar os pesos do modelo durante o treinamento.

Em sequência, o modelo de rede neural é treinado usando os dados de entrada *xs* e saída esperada *ys*, um número de épocas de treinamento fixo em cem e o parâmetro *verbose=1* (que determina o nível de detalhes das informações exibidas durante o treinamento. O valor 1 indica que será exibido um progresso detalhado com informações sobre o número de épocas concluídas e a perda (*loss*) atual).

Durante o treinamento, o modelo ajusta os pesos das suas camadas para minimizar a perda (*loss*) calculada entre as saídas previstas e as saídas reais. A otimização é realizada usando o algoritmo de otimização especificado durante a compilação do modelo (no caso, o otimizador *'adam'*).

Ao final do treinamento, o modelo terá aprendido a mapear as sequências de palavras para prever a próxima palavra na sequência com base no contexto anterior.

Antes de exibir os resultados, são criadas algumas funções.

A função *generate_text()* irá gerar uma sequência de palavras a partir do modelo treinado. Em resumo, a função *generate_text()* gera texto a partir do modelo treinado, usando um texto inicial (*seed_text*) como ponto de partida e gerando um número específico de palavras (*next_words*) com base nas probabilidades previstas pelo modelo. Cada palavra gerada é adicionada ao texto inicial, e o texto resultante é retornado como a saída da função.

A função *preprocess_text()* executa uma série de etapas de pré-processamento no texto gerado pelo modelo. Isso inclui a divisão do texto em frases, *tokenização* das palavras, remoção de *stopwords* e atribuição de *POS tags* a cada palavra. O resultado é uma lista de palavras pré-processadas, que serão utilizadas para a tentativa de criar um resumo a partir do texto gerado.

A função *extract_keywords()* usa a técnica *TF-IDF* para identificar as palavras-chave mais relevantes em um texto. Isso é feito *tokenizando* o texto, removendo *stopwords* e calculando as pontuações *TF-IDF* para cada palavra. As palavras-chave são então selecionadas com base nas pontuações *TF-IDF* mais altas.

Por fim, a função *create_summary()* gera um resumo do texto fornecido com base nas frequências das palavras em cada frase. As frases com as pontuações mais altas são selecionadas e concatenadas para formar o resumo final.

Após a implementação das funções citadas, agora partimos para a geração do texto a partir do modelo criado, o pré-processamento do resultado anterior

gerado, extração das entidades nomeadas, das palavras-chaves e a tentativa de criar um resumo, o qual, não obteve o resultado esperado.

Como forma de comparação, foi passada a variável inicial, do documento *.txt.* A partir dele, é possível visualizar um resultado mais eficiente a partir da utilizações das funções explicadas anteriormente.