Relatório do Projeto

Disciplina: Mineração de Texto
Projeto: Tradução de texto

Grupo:
Bruno Simões
Daniel Freire
Elymar Alves
Mateus Fittipaldi

1. Tutorial

Antes de começar o projeto é necessário abrir a base de dados contendo várias frases em inglês com tradução em português. Esses dados serão utilizados para treinar uma rede neural para fazer a tradução de frases em inglês para português.

```
file =
open('/content/projeto-de-mineracao-20192-traducao-de-texto/datas
ets/por.txt', mode='rt', encoding='utf-8')
data = file.read()
file.close()
```

1.1. Pré-processamento

Com a base de dados armazenada, agora precisamos tratar as frases, pois nós podemos entender o que há nelas, mas o computador não. Portanto, precisamos codificar as frases através de um pré-processamento para então podermos 'ensinar' ao computador o que deve ser feito.

1.1.1. Separar linhas e frases

Como primeiro passo do pré-processamento, precisamos separar cada linha da base de dados assim como as frases em inglês e português. Iremos separar as linhas pelo "\n" (nova linha) e armazená-las num array, em seguida iremos separar essas mesmas linhas pelo "\t" (tabulação) e armazenar o resultado da separação num outro array (tendo assim um array de arrays).

```
lines = data.strip().split('\n')
pairs = [line.split('\t') for line in lines]
```

1.1.2. Limpeza das linhas

Em seguida precisamos fazer uma "limpeza" nas linhas para facilitar a codificação. Iremos percorrer todas as linhas e normalizar os caracteres, ignorando os espaços em branco, convertendo todos os caracteres para minúsculas, removendo a pontuação, removendo caracteres que o computador não reconhece (passagem de Unicode para ASCII), removendo os números e, por fim, salvando as frases como strings.

```
cleaned = list()
re_print = re.compile('[^%s]' %
re.escape(string.printable))
table = str.maketrans('', '', string.punctuation)
for pair in pairs:
  clean_pair = list()
  for line in pair:
    # normalize unicode characters
    line = normalize('NFD', line).encode('ascii',
'ignore')
    line = line.decode('UTF-8')
    # tokenize on white space
    line = line.split()
    # convert to lowercase
    line = [word.lower() for word in line]
    # remove punctuation from each token
    line = [word.translate(table) for word in line]
    # remove non-printable chars form each token
    line = [re_print.sub('', w) for w in line]
    # remove tokens with numbers in them
    line = [word for word in line if word.isalpha()]
    # store as string
    clean_pair.append(' '.join(line))
  cleaned.append(clean_pair)

pairs_cleaned = array(cleaned)
```

- Para uma maior comodidade nas etapas seguintes, os dados pré processados até então são salvos em um arquivo a parte.

1.1.3. Separação de grupo de treino e de teste

Agora que temos todas as frases limpas e salvas, temos que separar o que será usado para treino do que será usado para testes. Essa separação é feita extraindo um recorte da nossa base de dados, aplicando uma randomização na ordem das entradas e separando esse resultado em mais duas partes(uma contendo 90% das entradas, sendo usada para treino e uma com os 10% restantes sendo usados para testes).

```
n_sentences = 30000
dataset = pairs_cleaned[:n_sentences, :]
# random shuffle
shuffle(dataset)
# split into train/test
train, test = dataset[:27000], dataset[27000:]
# save
save_data(dataset, 'eng-por-both.pkl')
save_data(train, 'eng-por-train.pkl')
save_data(test, 'eng-por-test.pkl')
```

- Uma limitação encontrada nesta etapa refere-se à quantidade de sentenças máximas que foi possível processar, definida como 30000 sentenças escolhidas aleatoriamente.

1.2. Tokenização

A seguir será aplicada a tokenização tanto para as palavras em inglês, quanto as palavras em português, no caso de traduções de outras línguas, lembre-se de utilizar a tokenização em ambos os lados.

```
def create_tokenizer(lines):
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(lines)
  return tokenizer

def max_length(lines):
  return max(len(line.split()) for line in lines)

# prepare english tokenizer
eng_tokenizer = create_tokenizer(dataset[:, 0])
eng_vocab_size = len(eng_tokenizer.word_index) + 1
eng_length = max_length(dataset[:, 0])
print('English Vocabulary Size: %d' % eng_vocab_size)
print('English Max Length: %d' % (eng_length))
# prepare portuguese tokenizer
por_tokenizer = create_tokenizer(dataset[:, 1])
por_vocab_size = len(por_tokenizer.word_index) + 1
```

```
por_length = max_length(dataset[:, 1])
print('Portuguese Vocabulary Size: %d' % por_vocab_size)
print('Portuguese Max Length: %d' % (por_length))
```

2. Criação da Máquina Neural de Tradução

2.1. Preparação dos dados de teste e treino

Aqui é aplicado o tokenizer definido anteriormente, usando como apoio duas funções de codificação, tanto no dataset de treino como no de teste.

```
def encode_sequences(tokenizer, length, lines):
  X = tokenizer.texts_to_sequences(lines)
  X = pad_sequences(X, maxlen=length, padding='post')
  return X

def encode_output(sequences, vocab_size):
  ylist = list()
  for sequence in sequences:
    encoded = to_categorical(sequence,
num_classes=vocab_size)
    ylist.append(encoded)
  y = array(ylist)
  y = y.reshape(sequences.shape[0], sequences.shape[1],
vocab_size)
  return y

trainX = encode_sequences(eng_tokenizer, eng_length,
train[:,0])
trainY = encode_sequences(por_tokenizer, por_length,
train[:,1])
trainY = encode_output(trainY, por_vocab_size)

testX = encode_sequences(eng_tokenizer, eng_length,
test[:,0])
testY = encode_sequences(por_tokenizer, por_length,
test[:,1])
testY = encode_output(testY, por_vocab_size)
```

2.2. Definição do modelo de rede neural a ser usado

Nosso modelo tem uma estrutura sequencial composta da seguinte forma:
camada de embedding (transformação de valores inteiros em vetores), camada LSTM (versão aprimorada de uma rede neural recorrente), camada de repeatvector (cria cópias da entrada atual), mais uma camada LSTM e uma camada timedistributed (para aplicação da função desejada em slices da entrada atual)
Por fim, após a definição, o modelo é compilado usando o algoritmo de otimização 'adam'.

```
def define_model(src_vocab, tar_vocab, src_timesteps,
tar_timesteps, n_units):
  model = Sequential()
  model.add(Embedding(src_vocab, n_units,
input_length=src_timesteps, mask_zero=True))
  model.add(LSTM(n_units))
  model.add(RepeatVector(tar_timesteps))
  model.add(LSTM(n_units, return_sequences=True))
  model.add(TimeDistributed(Dense(tar_vocab,
activation='softmax')))
  return model

model = define_model(eng_vocab_size, por_vocab_size,
eng_length, por_length, 256)
model.compile(optimizer='adam',loss='categorical_crossentrop
y')
```

2.3. Treino do modelo

Durante 30 epochs de treino, nosso modelo é rodado para verificação de possíveis melhorias automáticas e validação da diminuição da taxa de erro.

```
Checkpoint = ModelCheckpoint(filename, monitor='val_loss',
verbose=1, save_best_only=True, mode='min')
     model.fit(trainX, trainY, epochs=30, batch_size=64,
validation_data=(testX, testY), callbacks=[checkpoint],
verbose=2)
```

2.4. Avaliação do modelo

Definimos funções de avaliação baseados nos valores de BLEU e em uma ordem de pesos decrescentes porém incrementais em número. A avaliação leva em consideração a similaridade do resultado do algoritmo e o resultado esperado (escrito na base de dados).

```
def evaluate_model(model, tokenizer, sources, raw_dataset):
      actual, predicted = list(), list()
      for i, source in enumerate(sources):
            # translate encoded source text
            source = source.reshape((1, source.shape[0]))
            translation = predict_sequence(model, tokenizer,
source)
            raw_target, raw_src = raw_dataset[i,1],
raw_dataset[i,0]
            if i < 10:
                  print('src=[%s], target=[%s],
predicted=[%s]' % (raw_src, raw_target, translation))
            actual.append([raw_target.split()])
            predicted.append(translation.split())
      # calculate BLEU score
      print('BLEU-1: %f' % corpus_bleu(actual, predicted,
weights=(1.0, 0, 0, 0)))
      print('BLEU-2: %f' % corpus_bleu(actual, predicted,
weights=(0.5, 0.5, 0, 0)))
      print('BLEU-3: %f' % corpus_bleu(actual, predicted,
weights=(0.3, 0.3, 0.3, 0)))
      print('BLEU-4: %f' % corpus_bleu(actual, predicted,
weights=(0.25, 0.25, 0.25, 0.25)))

model = load_model('model.h5')
evaluate_model(model, por_tokenizer, trainX, train)
evaluate_model(model, por_tokenizer, testX, test)
```

- BLEU (**Bi**L**ingual** **E**valuation **U**nderstudy) é um número entre 0 e 1 que compara a similaridade da tradução com uma referência de alta qualidade (neste caso o próprio dataset). O valor 0 significa que a tradução não coincide com a tradução de referência, sendo assim de baixa qualidade. O valor 1 significa que a tradução corresponde perfeitamente à referência, sendo assim de alta qualidade.
- Como parte do cálculo da pontuação BLEU, é utilizado n-grams para calcular a precisão, onde são utilizados unigramas, bigramas, trigramas e quadrigramas. Esse valor corresponde a

quantas palavras são comparadas por vez com a tradução de referência. Sendo assim, unigrama compara palavra por palavra, bigrama compara par por par e assim vai até quadrigrama comparando quarteto por quarteto de palavras. E o cálculo de cada um desses valores é cumulativo para o cálculo do BLEU, variando apenas o peso de cada n-gram. Por padrão, o cálculo vai progredindo, primeiro utiliza-se apenas unigramas para o BLEU-1, depois unigramas e bigramas para BLEU-2, trigramas entram no BLEU-3 e quadrigramas no BLEU-4.

3. Testes

3.1. Definição

A função de tradução recebe como entrada o modelo da rede neural após as etapas de treino, o texto base, o tokenizer do idioma origem, o tamanho da base do idioma origem e o tokenizer do idioma destino. Funciona basicamente verificando o resultado da predição informada pelo modelo após a codificação das sequências serem feitas.

```
def translate_text(model, src_text, src_tokenizer,
src_length, tar_tokenizer):
  ltext = list()
  ltext.append(src_text)
  enconded_text = encode_sequences(src_tokenizer,
src_length, ltext)
  translated_text = predict_sequence(model, tar_tokenizer,
enconded_text)
return translated_text
```

3.2. Resultados

Como são escolhidas frases aleatórias para o treino e teste do modelo, a pontuação BLEU nunca fica exatamente a mesma, mas ainda tem resultados próximos:

Treino:

BLEU-1: 0.74
BLEU-2: 0.63
BLEU-3: 0.56
BLEU-4: 0.39

Teste:

BLEU-1: 0.56
BLEU-2: 0.42
BLEU-3: 0.34
BLEU-4: 0.19

Os resultados do BLEU para o treino foram melhores que os do teste, pois são utilizadas frases que a rede neural já conhece, portanto os resultados do teste são mais importantes para sabermos como está a qualidade da tradução na prática.
Dependendo do BLEU a qualidade da tradução varia bastante, indo de “traduções de qualidade muito alta” para “difícil de compreender o sentido”. No geral, nossa tradução apresenta uma boa adequação ao conseguir traduzir as palavras (BLEU-1), porém não apresenta uma boa fluência, ocorrendo casos onde fica difícil compreender o significado da frase traduzida (BLEU-4).

4. Usando outras línguas

4.1. Inglês para outra língua com dataset parecido

Para utilizar este projeto para fazer a tradução de outras línguas além de inglês para português basta realizar algumas mudanças simples. A principal e mais importante mudança é o dataset, pois precisamos de referências de tradução para a(s) nova(s) língua(s). Felizmente, o site onde conseguimos o dataset Portuguese-English também possui datasets de 76 outras línguas para inglês.
É necessário primeiro baixar o dataset desejado e abrir o arquivo da forma desejada, como por exemplo pelo repositório git:

```
file =
open('/content/projeto-de-mineracao-20192-traducao-de-texto/
datasets/language.txt', mode='rt', encoding='utf-8')
data = file.read()
file.close()
```

- Entende-se que o arquivo já está no repositório.
- Deve-se substituir “language.txt” pelo arquivo do dataset da língua escolhida.

Dependendo da língua, pode ser necessário realizar mais algum passo no pré-processamento, mas da forma que está já é possível realizar a tradução.
Para facilitar a compreensão também é bom trocar o nome das variáveis com “por” e “eng”, ou pela abreviação da língua em questão ou por um nome genérico:

```
save_data(dataset, 'src-tar.pkl')
# prepare source tokenizer
src_tokenizer = create_tokenizer(dataset[:, 0])
src_vocab_size = len(src_tokenizer.word_index) + 1
src_length = max_length(dataset[:, 0])
print('Source Vocabulary Size: %d' % src_vocab_size)
print('Source Max Length: %d' % (src_length))
# prepare target tokenizer
tar_tokenizer = create_tokenizer(dataset[:, 1])
tar_vocab_size = len(tar_tokenizer.word_index) + 1
tar_length = max_length(dataset[:, 1])
print('Target Vocabulary Size: %d' % tar_vocab_size)
print('Target Max Length: %d' % (tar_length))
```

E com isso já pode treinar e avaliar a rede neural para se obter um novo tradutor de inglês para outra língua.

4.2. Outra língua para inglês

Independente de qual seja a língua de origem o procedimento para fazer a tradução para inglês é o mesmo. Após realizar o procedimento do tópico anterior (4.1) precisamos trocar o índice do array do dataset, pois nos datasets em questão, inglês sempre fica na frente.

```
src_tokenizer = create_tokenizer(dataset[:, 1])
tar_tokenizer = create_tokenizer(dataset[:, 0])
```

Também é necessário trocar o índice dos arrays de treino e teste.

```
trainX = encode_sequences(src_tokenizer, src_length,
train[:,1])
trainY = encode_sequences(tar_tokenizer, tar_length,
train[:,0])
trainY = encode_output(trainY, tar_vocab_size)

testX = encode_sequences(src_tokenizer, src_length,
test[:,1])
testY = encode_sequences(tar_tokenizer, tar_length,
test[:,0])
testY = encode_output(testY, tar_vocab_size)
```

De forma geral, basta apenas trocar o índice de 0 para 1 e vice versa quando estiver utilizando o dataset.

4.3. Utilizando outro dataset

Se a ideia for fazer a tradução entre duas línguas que não sejam inglês ou simplesmente se quiser utilizar um dataset diferente, já muda um pouco o procedimento.
Primeiro, como sempre, precisamos abrir o(s) arquivo(s) com o dataset. Se for o caso de mais de um arquivo (um para cada língua), será necessário também criar um array com os dois datasets. Dependendo, talvez também seja necessário separar as frases manualmente, caso não estejam separadas por '\n', ou modificar o pré-processamento para fazer a separação e/ou tratar outras variações que o dataset possa ter.
O mais difícil seria o pré-processamento, uma vez que o array esteja limpo, contendo tanto as frases da língua original quanto as da língua traduzida, o projeto estará pronto para rodar com o(s) novo(s) dataset(s).

5. Considerações finais

5.1. Conclusão

O modelo criado mostrou-se capaz de retornar bons resultados quando as sentenças a serem traduzidas fazem parte integralmente da base de treino e também foi capaz de alcançar resultados compreensíveis, porém fora do contexto original nos demais casos.
Como forma de melhorar o desempenho do modelo de tradução, alguns pontos foram destacados:
• Melhorar/modificar o processo de limpeza dos dados;
• Refinar o vocabulário, fazendo por exemplo a remoção de palavras pouco utilizadas;
• Adicionar camadas de processamento ao modelo bem como extensão do tempo de treinamento do mesmo;
• Aumento da capacidade da memória, o que permitiria que um dataset maior fosse representado durante o processo;
• Alternar o método de mensuramento dos próprios resultados (usar outros além do BLEU).

5.2. Referências

“Tab-delimited Bilingual Sentence Pairs”. ManyThings.org. Disponível em: <http://www.manythings.org/anki/>.

BROWNLEE, Jason. “How to Develop a Neural Machine Translation System from Scratch”. Machine Learning Mastery, 2019. Disponível em: <https://machinelearningmastery.com/develop-neural-machine-translation-system-keras/>.

BROWNLEE, Jason. “A Gentle Introduction to Calculating the BLEU Score for Text in Python”. Machine Learning Mastery, 2019. Disponível em: <https://machinelearningmastery.com/calculate-bleu-score-for-text-python/>.

5.3. Artigos de referência

BROWNLEE, Jason. “Deep Learning for Natural Language Processing Develop: Deep Learning Models for your Natural Language Problems”. Machine Learning Mastery, 2017. Disponível em: <https://machinelearningmastery.com/deep-learning-for-nlp/>.

WANG, Kai, BABENKO, Boris, BELONGIE, Serge. “End-to-end scene text recognition.” Computer Vision” 2011 IEEE International Conference, 2011.

PRASAD, Manasa, BREINER, Theresa, VAN ESCH, Daan. “Mining Training Data for Language Modeling across the World’s Languages”. Proceedings of the 6th International Workshop on Spoken Language Technologies for Under-resourced Languages(SLTU 2018), 2018. Disponível em: <https://www.isca-speech.org/archive/SLTU_2018/pdfs/Manasa.pdf>.

BAHDANAU, Dzmitry, CHO, Kyunghyun, BENGIO, Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate. 2016. Disponível em: <https://arxiv.org/abs/1409.0473>.

VIJAYARANI, S., JANANI, R. “Text Mining: Open Source Tokenization Tools - An Analysis”. Advanced Computational Intelligence: An International Journal, 2016. Disponível em: <http://aircconline.com/acii/V3N1/3116acii04.pdf>.