Desafio de IA Celero - Classificador de Reviews

Marcela Ribeiro de Oliveira

14 de janeiro de 2020

Abstract

Este relatório explica o desenvolvimento do desafio de IA da Celero, que consiste em criar um classificador de reviews. Este relatório explica de forma breve os passos de implementação da solução bem como os resultados obtidos.

1 Solução do desafio

A solução do desafio foi dividida em 3 passos principais:

- Pré-processamento (*pre-processing)*;
- Extração de representação/ características (feature extraction);
- Classificação (classification);

1.1 Pré-processamento

O pré processamento é um dos componentes chave de um framework de classificação de texto. No pré processamento transformações são aplicadas no texto de forma a prepará-lo para que possa ser adequadamente representado e processado por um algoritmo.

Para este desafio as etapas de pré processamento adotadas foram as seguintes:

Conversão para letras minúsculas (*Lowercase conversion*): é o processo de converter todas as letras maiúsculas para letras minúsculas.

Tokenização (*Tokenization*): é o procedimento de dividir um texto ou uma frase em palavras ou outras partes significativas, como *tokens*.

Remoção de *stop words*: *Stop words* são palavras frequentes que não contêm informações, como preposições e conjunções de pronomes. Alguns exemplos de tais palavras no inglês incluem *'the', 'of', 'and', 'to'*.

Remoção de ruído: esse processo inclui remover pontuação, caracteres especiais e caracteres não alfabéticos.

Stemming: esse processo visa obter o radical das palavras derivadas. A hipótese por trás do *stemming* é que palavras com o mesmo radical geralmente descrevem os mesmos conceitos ou conceitos relativamente próximos no texto.

1.1.1 Bibliotecas utilizadas

Para auxiliar o pré-processamento foram utilizadas funções contidas na biblioteca **nltk** (Natural Language Toolkit)[1].

1.2 Extração de representação

Para extrair a representação de cada review foi utilizado o algoritmo Paragraph Vector. Este algoritmo aprende representações (vetores) de tamanho fixo, a partir de textos de tamanho variável. O Paragraph Vector for proposto por Tomas Mikolov e Quoc Le.

1.2.1 Bibliotecas utilizadas

Para a etapa de extração de características, foi utilizada uma implementação do paragraph vector disponível no **gensim**[2], o **doc2vec** (Doc2vec paragraph embeddings)[3].

O doc2vec possui vários parâmetros que precisam ser configurados de acordo com a representação desejada. Dentre eles alguns importantes são:

- **vector_size**: é o parâmetro que define o tamanho da representação dos textos, nos experimentos foi utilizado 100.
- **window**: é a distância máxima entre a palavra atual e a prevista em uma frase (tamanho do contexto), nos experimentos foi utilizado 10.
- **min_count**: número mínimo de vezes que uma palavra precisa ocorrer nos textos para ser considerada, nos experimentos foi utilizado 1 (todas as palavras são utilizadas).

Como o modelo de representação do doc2vec precisa ser treinado, quando o programa está em modo de treinamento, também é treinado e salvo o modelo do doc2vec para ser carregado e utilizado no modo de execução.

1.3 Classificação

Como classificador foi utilizado uma MLP (Multilayer Perceptron), uma rede neural composta de vários perceptrons. Uma MLP é composta de pelo menos três camadas: uma camada de entrada, uma camada escondida e uma camada de saída.

[1] https://www.nltk.org/
[2] https://pypi.org/project/gensim/
[3] https://radimrehurek.com/gensim/models/doc2vec.html

1.3.1 Bibliotecas utilizadas

Para implementação da MLP foi utilizada a biblioteca **Keras** (The Python Deep Learning library)[4].

Na solução implementada, foi utilizado o modelo sequencial. A rede neural é composta por uma camada de entrada do tipo *dense* com 200 neurônios e ativação *relu*. Em seguida, há outra camada do tipo *dense* e ativação *relu* seguida por uma camada de *dropout* com taxa 0.25. No fim da rede há uma camada do tipo *dense* com ativação *softmax*, que é onde é realizada a classificação.

O classificador é treinado por 64 épocas com tamanho do batch de 32.

2 Implementação

A implementação foi desenvolvida através de classes que separam responsabilidades, no total são 4 classes:

- DataLoader : responsável pela leitura dos dados (datasets).
- DataPreprocessing : responsável pelo tratamento e preparação dos dados.
- RepresentationHandler : responsável pela criação do modelo de representação.
- Classifier : responsável pela classificação.

O github foi utilizado para armazenamento e versionamento do código. Para esse problema além da branch *master*, duas outras branches foram utilizadas para separar o desenvolvimento das funcionalidades.

3 Resultados na base de teste IMDB

No modo de treinamento do programa, após ser construída a representação dos dados de treino e teste, foram utilizados 25000 reviews para treino do classificador e 25000 dados para teste. Além disso, foi feito o shuffle da base de dados para evitar o esquecimento catastrófico.

A acuraácia (taxa de acerto) na base de teste foi 82%.

[4] https://keras.io/