UNIVERSIDADE FEDERAL DE GOIÁS

ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E COMPUTAÇÃO

LABORATÓRIO DE INOVAÇÃO E AUTOMAÇÃO 1
ENGENHARIA DE COMPUTAÇÃO

PROJETO FINAL

ANÁLISE DE DADOS DA INTERNET PARA ACOMPANHAMENTO DAS ELEIÇÕES PRESIDENCIAIS BRASILEIRAS DE 2018

Docente: ADRIANO CÉSAR SANTANA
Discente: GIULIA BORGES DE OLIVEIRA (201703674)
Discente: MOACIR BATISTA TAVARES (201708988)
Discente: NIKHERSON BRENNO SOUZA SILVA (201703691)

Goiânia, 13 de AGOSTO de 2023.

1. Descrição do Projeto

Para o projeto final da disciplina de laboratório de inovação e automação 1 decidiu-se que o trabalho seguiria na abordagem da área de Processamento de Linguagem Natural com foco na Análise de Sentimentos para a predição das eleições presidenciais brasileiras em 2018 utilizando dados da rede social *Twitter* e o modelo *BERT* baseado na rede neural *Transformer*.

2. Passos Envolvidos:

2.1 Preparação dos dados

Os dados utilizados foram disponibilizados por André Cristiani [1] em seu *github*. O *dataset* conta com 600 *tweets* rotulados com sentimentos positivos e negativos. Ao observar o conjunto de dados percebeu-se a presença de *tweets* repetidos, portanto, removeu-se os *tweets* duplicados resultando em 410 *tweets* para o treinamento e validação.

Os textos devem passar por um pré-processamento para limpeza dos dados e remoção de informações que não são úteis para o contexto do texto. Através de experimentos, observou-se que o melhor resultado para o *BERT*, com os dados de treinamento utilizados, foi obtido realizando etapas de normalização, a remoção de ruído (*links*, *urls*, *emojis*, *hashtags*) e a remoção de caracteres especiais.

Após o pré-processamento, os textos foram separados em tokens individuais e convertidos em uma entrada numérica para servir de entrada para o modelo, utilizando a biblioteca *BertTokenizer* [2]. Os sentimentos também foram convertidos para dados numéricos, sendo 0 para negativo e 1 para positivo.

2.2 Construção do modelo

O modelo consiste em 3 camadas:

1. A camada do *BERT* é o modelo *BERT* pré-treinado, que recebe como entrada os *tokens* gerados pelo tokenizador e realiza o trabalho de classificação propriamente dito.

2. A camada de *dropout* busca evitar o *overfitting*, generalizando o aprendizado e evitando que o modelo se ajuste demais aos dados de treinamento.

3. A camada linear é a camada que mapeia a saída gerada pelo *BERT* para as classes do experimento, 0 para sentimento negativo e 1 para sentimento positivo.

2.3 Ajuste fino do modelo

O *BERT* é um modelo pré-treinado. Ele é treinado com um grande volume de dados, onde aprende a reconhecer o contexto e relações da linguagem no qual foi treinado. O modelo pré-treinado é então ajustado, realizando um ajuste fino para realizar tarefas específicas, nesse caso, análise de sentimentos de *tweets* no contexto político.

O modelo utilizado para esse trabalho foi o *BERTimbau* [3], modelo *BERT* pré-treinado com dados em portugês, realizou-se o ajuste fino para que ele compreendesse o sentimento dos *tweets* com contexto das eleições presidenciais de 2018. Os seguintes parâmetros foram considerados:

Batch size	Tamanho máximo da entrada	Taxa de aprendizado
8	200	2e-5

2.4 Treinamento do modelo

Os dados rotulados são divididos aleatoriamente em 80% para treinamento e 20% para validação. O modelo foi treinado por 5 épocas. A cada iteração, o modelo é treinado e validado. A acurácia e loss de validação é usada para verificar a performance do modelo. Caso o modelo se mostre o melhor até então, ele é salvo para ser usado na previsão.

2.5 Modelo aplicado

O melhor modelo salvo é carregado e utilizado para a análise de sentimentos dos *tweets* a serem classificados. Primeiro são removidos dados não relacionados aos candidatos e às eleições, em seguida o modelo realiza a análise de sentimento nos dados, que são classificados em Negativo ou Positivo. Em seguida separa-se o dataset por candidato, buscando nos textos menções ou palavras relacionadas a cada candidato. Em seguida são analisadas as predições feitas pelo modelo.

3. Código do Projeto

```
# -*- coding: utf-8 -*-
"""lia.ipynb

Automatically generated by Colaboratory.

"""

import torch
import matplotlib.pyplot as plt
import pandas as pd
from collections import defaultdict

from nltk import TweetTokenizer
from torch import nn
from transformers import BertModel
from torch.utils.data import Dataset
import torch.nn.functional as F

from transformers import BertTokenizer

import logging
logging.getLogger("transformers.modeling_utils").setLevel(logging.ERROR)

pd.set_option('display.max_colwidth', None)

"""## Configurações do BERT

Habilitando GPU
"""

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)

"""Tokenizer"""

PRE_TRAINED_MODEL_NAME = 'neuralmind/bert-base-portuguese-cased'
tokenizer_bert = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)

"""Exemplo com o BertTokenizer"""

sample_txt = "Mano eu não gosto do Bolsonaro, mas amo meu país"

tokens_basic = tokenizer_bert.tokenize(sample_txt)
tokens = tokenizer_bert.encode_plus(
```

```
    sample_txt,
    add_special_tokens=True,
    max_length=200,
    return_token_type_ids=False,
    padding=True,
    truncation=True,
    return_attention_mask=True,
    return_tensors='pt',
)

print(tokens_basic)
print(tokens)

"""### Modelo"""

class TweetDataset(Dataset):

    def __init__(self, texts, clean_texts, sentiments, tokenizer, max_len):
        self.texts = texts
        self.clean_texts = clean_texts
        self.sentiments = sentiments
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, item):
        text = str(self.texts[item])
        clean_text = str(self.clean_texts[item])
        target = self.sentiments[item] if self.sentiments is not None else None

        encoding = self.tokenizer.encode_plus(
            clean_text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding=True,
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'text': text,
            'clean_text': clean_text,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
```

```
            'sentiment': torch.tensor(target, dtype=torch.long) if target is not None else None
        }

from torch.utils.data import DataLoader

def create_data_loader(dataframe, tokenizer, max_len, batch_size):
    ds = TweetDataset(
        texts=dataframe.text.to_numpy(),
        clean_texts = dataframe.clean_text.to_numpy(),
          sentiments= dataframe.sentiment.to_numpy() if 'sentiment' in dataframe.columns  else
None,
        tokenizer=tokenizer,
        max_len=max_len
    )

    return DataLoader(
        ds,
        batch_size=batch_size,
        num_workers=4
    )

class SentimentClassifier(nn.Module):

    def __init__(self, n_classes):
        super(SentimentClassifier, self).__init__()
                        self.bert    =    BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME,
return_dict=False)
        self.drop = nn.Dropout(p=0.2)
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

    def forward(self, input_ids, attention_mask):
        _, pooled_output = self.bert(
            input_ids=input_ids.unsqueeze(0),
            attention_mask=attention_mask.unsqueeze(0)
        )
        output = self.drop(pooled_output)
        return self.out(output)

"""Criação do modelo"""

class_names = ['Negativo', 'Positivo']

sentiment_model = SentimentClassifier(len(class_names))
sentiment_model = sentiment_model.to(device)

"""## Métodos de treino e validação"""

EPOCHS = 5
```

```
BATCH_SIZE = 8
MAX_LEN = 200
LR = 2e-5

def train_epoch(
        model,
        data_loader,
        loss_fn,
        optimizer,
        device,
        scheduler,
        n_examples
):
    model = model.train()

    losses = []
    correct_predictions = 0

    for d in data_loader.dataset:
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        targets = d["sentiment"].to(device)

        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )

        _, preds = torch.max(outputs, dim=1)

        loss = loss_fn(outputs[0], targets)

        correct_predictions += torch.sum(preds == targets)
        losses.append(loss.item())

        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

    return correct_predictions.double() / n_examples, np.mean(losses)


def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()

    losses = []
```

```
    correct_predictions = 0

    with torch.no_grad():
        for d in data_loader.dataset:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["sentiment"].to(device)

            outputs = model(
                input_ids=input_ids,
                attention_mask=attention_mask
            )
            _, preds = torch.max(outputs, dim=1)

            loss = loss_fn(outputs.squeeze(0), targets)

            correct_predictions += torch.sum(preds == targets)
            losses.append(loss.item())

    return correct_predictions.double() / n_examples, np.mean(losses)

from sklearn.model_selection import train_test_split

RANDOM_SEED = 42

def sep_data(dataframe: pd.DataFrame, test_size = 0.2) :
    dataframe = dataframe.sample(frac=1)
                    df_train,     df_test     =     train_test_split(dataframe,     test_size=test_size,
random_state=RANDOM_SEED)

    return df_train, df_test

def create_data_loaders(df_train, df_test):
     train_data_loader: DataLoader = create_data_loader(df_train, tokenizer_bert, MAX_LEN,
BATCH_SIZE)
    test_data_loader = create_data_loader(df_test, tokenizer_bert, MAX_LEN, BATCH_SIZE)

    return train_data_loader, test_data_loader

from transformers import get_linear_schedule_with_warmup

def create_train_conf(len_df_train):
    optimizer = torch.optim.AdamW(sentiment_model.parameters(), lr=LR)
    total_steps = len_df_train * EPOCHS

    scheduler = get_linear_schedule_with_warmup(
        optimizer,
        num_warmup_steps=0,
```

```
        num_training_steps=total_steps
    )

    loss_fnr = nn.CrossEntropyLoss().to(device)

    return optimizer, scheduler, loss_fnr

"""## Dados

### Pré-processamento
"""

def normalize(text: pd.Series)-> pd.Series:
    return text.str.lower()

def remove_hashtags(df: pd.DataFrame):
    df.clean_text.replace(r'#\w+', ' ', regex=True, inplace=True)

def remove_link(df: pd.DataFrame):
    df.clean_text.replace(r'((?=(https://|http://))\S+)', ' ', regex=True, inplace=True)

def remove_tags(df: pd.DataFrame):
    # Remove marcações com exceção das marcações às contas oficiais dos candidatos
            df.clean_text.replace(r'@(?!(lulaoficial|jairbolsonaro|haddad_fernando))[^    ]*',    '    ',
regex=True, inplace=True)

def sub_linebreak(df: pd.DataFrame):
    df.clean_text.replace(r'\n', ' ', regex=True, inplace=True)

def remove_special_ch(df: pd.DataFrame):
    # Remoção de caracteres especiais e saltos de linha
    df.clean_text.replace(r'[^\w ]', ' ', regex=True, inplace=True)

def remove_rt(df: pd.DataFrame):
    df.clean_text.replace(r'(?=(rt |RT ))[^ ]*', '', regex=True, inplace=True)

def remove_empty(df: pd.DataFrame):
    df.drop(df[df.clean_text.str.fullmatch(r'\s*')].index, inplace=True)
    df.reset_index(drop=True, inplace=True)

def remove_len1(df: pd.DataFrame):
    indexes = list()
    for index, row in df.iterrows():
        words = TweetTokenizer().tokenize(row['clean_text'])
        if len(words) == 1:
            indexes.append(index)
    df.drop(indexes, inplace=True)
```

```
def clean_whitespaces(df: pd.DataFrame):
    df.clean_text.replace(r'  +', ' ', inplace=True, regex=True)
    df.clean_text.replace(r'^ ', '', inplace=True, regex=True)

def preprocess(df_param: pd.DataFrame):
    df = df_param.copy()
    df['clean_text'] = normalize(df.text)
    sub_linebreak(df)
    remove_hashtags(df)
    remove_link(df)
    remove_tags(df)
    remove_special_ch(df)
    remove_rt(df)

    clean_whitespaces(df)
    remove_empty(df)
    remove_len1(df)
    return df

def to_sentiment(polaridade: str):
    if polaridade == 'Negativo':
        return 0
    if polaridade == 'Positivo':
        return 1

"""### Carregando dados"""

tweets = pd.read_csv('tweets.csv')
tweets['sentiment'] = tweets.polaridade.apply(to_sentiment)
tweets.drop_duplicates(subset=['text'], inplace=True)
tweets = preprocess(tweets)

"""### Treinamento"""
history = defaultdict(list)
best_accuracy = 0
loss_for_best_acc = 20

df_train, df_val = sep_data(tweets)

train_data_loader, val_data_loader = create_data_loaders(df_train, df_val)
optimizer, scheduler, loss_fn = create_train_conf(len(df_train))

for epoch in range(EPOCHS):

    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 10)

    train_acc, train_loss = train_epoch(
```

```
        sentiment_model,
        train_data_loader,
        loss_fn,
        optimizer,
        device,
        scheduler,
        len(df_train)
    )

    print(f'Train loss {train_loss} accuracy {train_acc}')

    val_acc, val_loss = eval_model(
        sentiment_model,
        val_data_loader,
        loss_fn,
        device,
        len(df_val)
    )

    print(f'Val   loss {val_loss} accuracy {val_acc}')
    print()

    history['train_acc'].append(train_acc.cpu())
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc.cpu())
    history['val_loss'].append(val_loss)

        if val_acc > best_accuracy or (val_acc == best_accuracy and val_loss < loss_for_best_acc):
        torch.save(sentiment_model.state_dict(), 'best_model.bin')
        best_accuracy = val_acc
        loss_for_best_acc = val_loss

"""## Métricas

**Acurácia**
"""

plt.plot(history['train_acc'], label='train accuracy')
plt.plot(history['val_acc'], label='validation accuracy')

plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1]);

"""**Loss**"""
```

```
plt.plot(history['train_loss'], label='train loss')
plt.plot(history['val_loss'], label='validation loss')

plt.title('Training history')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()

"""## Métricas"""

model_val = SentimentClassifier(len(class_names))
model_val.load_state_dict(torch.load('best_model.bin'))
model_val = model_val.to(device)

def get_predictions(model, data_loader):
  model = model.eval()

  review_texts = []
  predictions = []
  prediction_probs = []
  real_values = []

  with torch.no_grad():
    for d in data_loader.dataset:
      texts = d["clean_text"]
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)
      targets = d["sentiment"].to(device)

      outputs = model(input_ids=input_ids,
                      attention_mask=attention_mask)

      _, preds = torch.max(outputs, dim=1)

      probs = F.softmax(outputs, dim=1)

      review_texts.append(texts)
      predictions.extend(preds)
      prediction_probs.extend(probs)
      real_values.extend(targets.unsqueeze(0))

  predictions = torch.stack(predictions).cpu()
  prediction_probs = torch.stack(prediction_probs).cpu()
  real_values = torch.stack(real_values).cpu()
  return review_texts, predictions, prediction_probs, real_values

y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(model_val, val_data_loader)
```

```
import numpy as np
from sklearn.metrics import confusion_matrix
def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
    plt.ylabel('Sentimento verdadeiro')
    plt.xlabel('Sentimento previsto')

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred, target_names=class_names))

"""## Previsão

Carregando o modelo
"""

model = SentimentClassifier(len(class_names))
model.load_state_dict(torch.load('best_model.bin'))
model = model.to(device)

"""Carregando os dados para previsão"""

tweets_prev = pd.read_csv('2Turno.csv', dtype= {'Coordenadas': 'str'})
tweets_prev.drop(columns=['Usuario', 'Localizacao', 'Coordenadas'], inplace=True)
tweets_prev.rename({'Mensagem': 'text'}, axis=1, inplace=True)

tweets_prev.dropna(subset=['text'], inplace=True)

tweets_prev = preprocess(tweets_prev)

#Bolsonaro
bwords = ['jair', 'bolsonaro']

#Haddad
hwords = ['haddad', 'lula']

tweets_bolsonaro = tweets_prev[tweets_prev.clean_text.str.contains('|'.join(bwords))]
print('Quantidade de tweets que citam Bolsonaro:', len(tweets_bolsonaro))

tweets_haddad = tweets_prev[tweets_prev.clean_text.str.contains('|'.join(hwords))]
print('Quantidade de tweets que citam Haddad:', len(tweets_haddad))
```

```
indexes = set(tweets_bolsonaro.index)
indexes = indexes.union(tweets_haddad.index)
print('Quantidade de tweets que citam algum dos 2 candidatos:', len(indexes))

tweets_prev.drop(index=tweets_prev.index.difference(indexes), inplace=True)

tweets_prev

"""**Criando a estruturação para a previsão**"""

tweets_prev_data_loader = create_data_loader(tweets_prev, tokenizer_bert, MAX_LEN, BATCH_SIZE)

"""**Método para previsão dos dados**"""

def get_predictions(model, data_loader):
  model = model.eval()

  texts = []
  clean_texts = []
  predictions = []

  with torch.no_grad():
    for d in data_loader.dataset:
      text = d["text"]
      clean_text = d["clean_text"]
      input_ids = d["input_ids"].to(device)
      attention_mask = d["attention_mask"].to(device)

      outputs = model(input_ids=input_ids,
                      attention_mask=attention_mask)

      _, preds = torch.max(outputs, dim=1)

      texts.append(text)
      clean_texts.append(clean_text)
      predictions.extend(preds)

  predictions = torch.stack(predictions).cpu()

  return pd.DataFrame({'text': texts, 'clean_text': clean_texts, 'sentiment': predictions})

"""**Previsão**"""

preds = get_predictions(model, tweets_prev_data_loader)

def to_polarity(sentiment: int):
```

```
    if sentiment == 0:
        return 'Negativo'
    if sentiment == 1:
        return 'Positivo'

preds.to_csv('results.csv', index=False)

preds = pd.read_csv('results.csv')

df = preds.copy()
df.loc[df.clean_text.str.contains('|'.join(hwords)), 'label'] = 'Haddad'
df.loc[df.clean_text.str.contains('|'.join(bwords)), 'label'] = 'Bolsonaro'

df = df.drop(columns=['text', 'clean_text']).groupby('label').value_counts().reset_index(name
= 'amount')
df['Sentimento'] = df.sentiment.apply(to_polarity)

df

"""## Plot dos resultados

**Quantidades totais por sentimento**
"""

import seaborn as sns

sns.set_style("whitegrid")

green = '#00FF7F'
red = '#FF3333'
colors = sns.set_palette([red, green])

hue_order = ['Negativo', 'Positivo']

g = sns.catplot(data=df, x = 'label', y = 'amount', hue='Sentimento', hue_order=hue_order,
kind='bar', palette=colors)

g.ax.set(title = 'Tweets')

for p in g.ax.patches:
    txt = str(int(p.get_height()))
    txt_x = p.get_x() + p.get_width()/2
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt, ha = 'center')

"""**Porcentagem de positivos por candidato**"""

df = preds.copy()
```

```
df = df.loc[df.sentiment == 1]
df.loc[df.clean_text.str.contains('|'.join(hwords)), 'label'] = 'Haddad'
df.loc[df.clean_text.str.contains('|'.join(bwords)), 'label'] = 'Bolsonaro'

df = df.drop(columns=['text', 'clean_text']).groupby('sentiment').value_counts(normalize=True).reset_index(name = 'percent')
df['percent'] = df['percent'].apply(lambda x: x*100)

df

import seaborn as sns

sns.set_style("whitegrid")

blue = '#0000FF'
red = '#FF3333'
colors = sns.set_palette([blue, red])

g = sns.catplot(data=df, x = 'label', y = 'percent', kind='bar', palette=colors)
g.ax.set_ylim(0,100)

g.ax.set(title = 'Tweets')

for p in g.ax.patches:
    txt = str(round(p.get_height(), 1))
    txt_x = p.get_x() + p.get_width()/2
    txt_y = p.get_height()
    g.ax.text(txt_x,txt_y,txt, ha = 'center')
```

4. Referências

[1] - CRISTIANI, A.; LIEIRA, D.; CAMARGO, H. A Sentiment Analysis of Brazilian Elections Tweets. Disponível em: <https://sol.sbc.org.br/index.php/kdmile/article/view/11971/11836>. Acesso em: 15 ago. 2023.

[2] - Tokenizer. Disponível em: <https://huggingface.co/docs/transformers/main_classes/tokenizer>.

[3] - neuralmind/bert-base-portuguese-cased · Hugging Face. Disponível em: <https://huggingface.co/neuralmind/bert-base-portuguese-cased>.