Tarea 2

Cuerpo Docente

- Profesores: Andrés Abeliuk, Fabián Villena.
- Profesor Auxiliar: Martín Paredes

Instrucciones generales

- Grupos de máximo 4 personas.
- Esta prohibido compartir las respuestas con otros grupos.
- Indicios de copia serán penalizados con la nota mínima.
- Cualquier duda fuera del horario de clases al foro. Mensajes al equipo docente serán respondidos por este medio.
- Pueden usar cualquier material del curso que estimen conveniente, si utiliza material extra debe citarlo.

Integrantes

> POR FAVOR AGREGAR TODOS LOS NOMBRES DE LOS INTEGRANTES

Contexto

El discurso de odio es cualquier expresión que promueva o incite a la discriminación, la hostilidad o la violencia hacia una persona o grupo de personas en una relación asimétrica de poder, tal como la raza, la etnia, el género, la orientación sexual, la religión, la nacionalidad, una discapacidad u otra característica similar.

En cambio, la incivilidad se refiere a cualquier comportamiento o actitud que rompe las normas de respeto, cortesía y consideración en la interacción entre personas. Esta puede manifestarse de diversas formas, tal como insultos,

ataques personales, sarcasmo, desprecio, entre otras.

En esta tarea tendrán a su disposición un dataset de textos con las etiquetas `odio`, `incivilidad` o `normal`. La mayor parte de los datos se encuentra en español de Chile. Con estos datos, deberán entrenar un modelo que sea capaz de predecir la etiqueta de un texto dado.

El corpus para esta tarea se compone de 3 datasets:

- Multilingual Resources for Offensive Language Detection de Arango et al. (2022)
- Dataton UTFSM No To Hate (2022)
- Datos generados usando la API de GPT3 (modelo DaVinci 03).

Agradecimientos a los autores por compartir los datos y a David Miranda, Fabián Diaz, Santiago Maass y Jorge Ortiz por revisar y reetiquetar los datos en el contexto del curso "Taller de Desarrollo de Proyectos de IA" (CC6409), Departamento de Ciencias de la Computación, Universidad de Chile.

Tarea a resolver

Para esta tarea 2, buscaremos desarrollar un *benchmark* sobre una tarea de clasificación de NLP. Un benchmark es básicamente utilizar diferentes técnicas para resolver una misma tarea específica, en este caso seguiremos buscando alternativas para resolver el problema de clasificación de la tarea 1. Particularmente, se le pide:

- Implementar una arquitectura en RNN utilizando PyTorch.
- Utilizar transformers para revolver el problema de clasificación, en especifico utilizar BETO.
- Utilizar algún LLM utilizando Zero y Few short learning para resolver el problema de clasificación.

Cargar el dataset

En esta sección, cargaremos el dataset desde el repositorio del módulo. Para ello ejecute las siguientes líneas:

```
import pandas as pd
```

Haz doble clic (o ingresa) para editar

```
# Dataset.
dataset_df = pd.read_csv("https://raw.githubusercontent.com/dccuchile/CC6205/master/assignments/ne
```

Analizar los datos

En esta sección analizaremos el balance de los datos. Para ello se imprime la cantidad de tweets de cada dataset agrupados por la intensidad de sentimiento.

```
dataset_df.sample(5)
```

```
dataset_df["clase"].value_counts()
```

```
target_classes = list(dataset_df['clase'].unique())
target_classes
```

Instalar librerias

Puede usar esta celda para instalar las librerías que estime necesario.

```
%%capture
```

Importar librerías

Importar librerías

En esta sección, importamos la liberías necesarias para el correcto desarrollo de esta tarea. Puede utilizar otras librerías que no se en encuentran aquí, pero debe citar su fuente.

```
!pip uninstall torch torchvision torchaudio

!pip install torch==2.3.0 --index-url https://download.pytorch.org/whl/cpu
!pip uninstall scipy
!pip install scipy==1.11.4
!pip install torchtext
!pip install scikit-plot
```

```
import nltk
import numpy as np

from nltk import word_tokenize

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.base import BaseEstimator, TransformerMixin

# importe aquí sus clasificadores

import matplotlib.pyplot as plt

# word2vec
from gensim.models import Word2Vec, KeyedVectors
from gensim.models.phrases import Phrases, Phraser

# Pytorch imports
import torch
from torchtext.data import get_tokenizer
```

```
from torchtext.vocab import build_vocab_from_iterator

from torch.utils.data import DataLoader
from torchtext.data.functional import to_map_style_dataset

from torch import nn
from torch.nn import functional as F

from tqdm import tqdm
from sklearn.metrics import accuracy_score
import gc

from torch.optim import Adam
```

Crear un clasificador basado en RNN

En esta parte de le pide definir un clasificador utilizando `PyTorch` con alguna arquitectura en Redes Recurrentes. Para ello debe realizar todos los pasos vistos en el tutorial 5, por lo que se recomienda revisarlo. Importante, no puede replicar ningún ejemplo de los del tutorial 5, debe proponer sus propias arquitecturas. Se le recomienda leer como utilizar variaciones de la RNN, como la LSTM o GRU en `Pytorch`.

Para completa esta parte deberá replicar el flujo de trabajo de como utilizar `PyTorch`. Esta esctrictamente prohibido utilizar variaciones que resuelvan directamente este problema, como `PyTorch Lightning` o `TensorFlow`. Los pasos a completar son los siguientes:

Cargar el dataset.

Haz doble clic (o ingresa) para editar

```
Comienza a programar o generar con IA.
```

Definir el vocabulario.

```
Comienza a programar o generar con IA.
```

Cargar el `DataLoader`

Recuerde que podría necesitar una función intermedia para procesar cada batch durante el entrenamiento, pero no es obligatorio hacerlo.

```
Comienza a programar o generar con IA.
```

Definir la red recurrente.

Recuerde que debe difirnir los hyperparametros que estime conveniente.

```
class RNNClassifier(nn.Module):
    def __init__(self):
        super(RNNClassifier, self).__init__()
        pass
    def forward(self, X_batch):
        pass
```

Funciones de entrenamiento y evaluación.

Para esta parte, puede utilizar las funciones vista en el tutorial directamente. Pero es su reponsabilidad ajustarlas a su código.

```
def CalcValLossAndAccuracy(model, loss_fn, val_loader):
    with torch.no_grad():
```

```
        Y_shuffled, Y_preds, losses = [],[],[]
        for X, Y in val_loader:
            preds = model(X)
            loss = loss_fn(preds, Y)
            losses.append(loss.item())

            Y_shuffled.append(Y)
            Y_preds.append(preds.argmax(dim=-1))

        Y_shuffled = torch.cat(Y_shuffled)
        Y_preds = torch.cat(Y_preds)

        print("Valid Loss : {:.3f}".format(torch.tensor(losses).mean()))
        print("Valid Acc  : {:.3f}".format(accuracy_score(Y_shuffled.detach().numpy(), Y_preds.det


def TrainModel(model, loss_fn, optimizer, train_loader, val_loader, epochs=10):
    for i in range(1, epochs+1):
        losses = []
        for X, Y in tqdm(train_loader):
            Y_preds = model(X)

            loss = loss_fn(Y_preds, Y)
            losses.append(loss.item())

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        print("Train Loss : {:.3f}".format(torch.tensor(losses).mean()))
        CalcValLossAndAccuracy(model, loss_fn, val_loader)

def MakePredictions(model, loader):
    Y_shuffled, Y_preds = [], []
    for X, Y in loader:
        preds = model(X)
        Y_preds.append(preds)
        Y_shuffled.append(Y)
```

```
        gc.collect()
        Y_preds, Y_shuffled = torch.cat(Y_preds), torch.cat(Y_shuffled)

        return Y_shuffled.detach().numpy(), F.softmax(Y_preds, dim=-1).argmax(dim=-1).detach().numpy()
```

Entrenamiento del modelo

Ejecute el entrenamiento de su modelo propuesto.

```
epochs = ...
learning_rate = ...

loss_fn = nn.CrossEntropyLoss()
rnn_classifier = RNNClassifier()
optimizer = Adam(rnn_classifier.parameters(), lr=learning_rate)

TrainModel(rnn_classifier, loss_fn, optimizer, train_loader, test_loader, epochs)
```

Evaluacion del modelo

Ejecute la evaluación de su modelo, y genere un reporte de evaluación similar al de la pregunta anterior.

```
Y_actual, Y_preds = MakePredictions(rnn_classifier, test_loader)
```

```
print("Test Accuracy : {}".format(accuracy_score(Y_actual, Y_preds)))
print("\nClassification Report : ")
print(classification_report(Y_actual, Y_preds, target_names=target_classes))
print("\nConfusion Matrix : ")
print(confusion_matrix(Y_actual, Y_preds))
```

Finalmente, análice sus resultados. ¿Porqué cree que obtuvo esos resultados?. ¿Es mejor que sólo utilizar Word

Finalmente, analice sus resultados, ¿por qué cree que obtuvo esos resultados?, ¿Es mejor que solo utilizar Word Embeddings, porque?. Justique.

Transformers BERT.

Para esta tarea se le piden crear una representación de texto usando la arquitectura basada en transformers, BERT:

```
!pip install transformers
```

Import BETO

Para esto debe importar el tokenizador y el modelo BETO.

```
from transformers import AutoTokenizer, AutoModelForMaskedLM
```

Ejemplo de extracción de features.

Luego, debe cargar los modelos pre-entrenados:

```
tokenizer = AutoTokenizer.from_pretrained('dccuchile/bert-base-spanish-wwm-uncased')
model = AutoModelForMaskedLM.from_pretrained('dccuchile/bert-base-spanish-wwm-uncased', output_hid
```

Una vez cargado BETO, debe utilizarlo para extraer los embeddings asociados a la texto de su corpus. Se espera que usted realice lo siguiente:

Tokenizamos el texto para extraer los ids a cada palabra en el vocabulario interno de BETO, se recomienda utilizar el padding, trunctation, y max_length para considerar oraciones de diferentes tamaños.

Luego, debe verificar si cada uno de los ids extraídos se encuentran en la misma máquina donde fue cargado el

Luego, debe verificar si cada uno de los ids extraídos se encuentran en la misma maquina donde fue cargado el modelo (CPU o GPU), se recomienda dejar todo en GPU.

```
# oración
sentence = "Hola, que tal? me gusta mucho el curso de NLP"

# extraemos los ids de los tokens, se recomienda definir los valores de las variables:
#  padding, truncation, max_length debido a que las secuencia de texto puede tener diferentes larg
inputs = tokenizer(sentence, padding=True, truncation=True, return_tensors="pt", max_length=512)
# revisamos si cada de los ids, se encuentran en la misma máquina que el modelo (GPU o CPU)
inputs = {k: v.to(model.device) for k, v in inputs.items()}
inputs
```

Una vez, extraídos los ids, usted debe obtener los estados ocultos de las ultimas capas de BETO.

```
# Extraemos la features
outputs = model(**inputs)

# ahora `outputs` tendrá el atributo `hidden_states`
hidden_states = outputs.hidden_states
```

Finalmente, debe guardar los embeddings en CPU los embeddings del token [CLS] que será usados en la clasificación del análisis de sentimientos.

```
with torch.no_grad():
# Seleccionamos la última capa y obtenemos el primer token ([CLS]) para cada ejemplo
# Estos son los embeddings que normalmente se usan para tareas de clasificación.
  cls_embeddings = hidden_states[-1][:, 0, :].cpu().numpy()
```

Extraer features de todo el dataset

Considerando lo anterior, usted debe implementar la función `get_beto_features_in_batches` para extraer los features de BETO (los estados ocultos y los embeddings del token [CLS]).

Esta función recibe dos parámetros, el texto a vectorizar y un tamaño de batch, debido a que es extramadamente recomendable a que procesen el texto por lotes, ya que si cargan todos el modelo se les agotará la memoria RAM.

```
# Función para procesar los textos en lotes y obtener las características de BETO

def get_beto_features_in_batches(texts, batch_size):

  pass
```

Ahora extraíga los features, un punto importante es que la extracción de features podría tomar alrededor de una a dos horas dependiendo del tamaño del batch que utilicen.

```
train_embs = get_beto_features_in_batches(..., ...)
```

Clasificación

Una vez extraído los features de BETO, realice la clasificación de los embeddings obtenidos. Debe implementar dos clasificadores a su elección.

```
Comienza a programar o generar con IA.
```

```
Comienza a programar o generar con IA.
```

Reporte de evaluación

Una vez realizada la clasificación, realice el reporte de clasificación y el análsis de la matriz confusión para ambos clasificadores.

Recuerde que para hacer esto, debe extraer los features del dataset de testing.

```
Comienza a programar o generar con IA.
```

```
Comienza a programar o generar con IA.
```

Finalmente, que puede decir de los resultados obtenidos. ¿Se diferencia de los resultados obtenidos de la red RNN? ¿A que se debe esto? Justifique

Haz doble clic (o ingresa) para editar

Large Language Model

Zero Shot Learning

Utilizando la técnica de zero shot learning, utilice la API de `openai` para clasificar el texto del dataset.

Además, genere el reporte de clasificación usando `scikit-learn` como en las preguntas anteriores.

Recuerde solicitar al profesor auxiliar el TOKEN para hacer consultas al LLM.

```
Comienza a programar o generar con IA.
```

```
Comienza a programar o generar con IA.
```

Few Shot Learning

Utilizando la técnica de few shot learning, utilice la API de `openai` para clasificar el texto del dataset.

Además, genere el reporte de clasificación usando `scikit-learn` como en las preguntas anteriores.

Recuerde solicitar al profesor auxiliar el TOKEN para hacer consultas al LLM.

```
Comienza a programar o generar con IA.
```

```
Comienza a programar o generar con IA.
```