Import here all the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
# import plotly.express as px
import random
from keras.utils import np_utils
from scipy.stats import multivariate_normal as mvn

import string
import re #regular expressions
import nltk
from nltk import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import WordNetLemmatizer
from __future__ import absolute_import, division, print_function, unicode_literals
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('punkt')

!pip install -q -U "tensorflow-text==2.8.*"
!pip install -q tf-models-official==2.7.0
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
from tensorflow import keras
import tensorflow_text as text
from official.nlp import optimization

#Import .py file of general algorithms
# from google.colab import files
# files.upload()
# from general import accuracy
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
#Mount Google Drive Folders
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

Load dataset

```
def numLabel(data):
  if data == 'commissive' or data == 'directive': #Regular chat
    label = 0
  elif data == 'inform':
    label = 1
  else: #Question
    label = 2

  return label
```

```
df_train = pd.read_csv('/content/drive/MyDrive/Enhance It/Training Projects/NLP/train.csv')
df_train.drop(['Emotion','Dialogue_ID'], inplace=True, axis=1)
df_train['Dialogue_Act'] = df_train['Dialogue_Act'].apply(numLabel)
df_train.head()
```

	Utterance	Dialogue_Act
0	say , jim , how about going for a few beers af...	0
1	you know that is tempting but is really not go...	0
2	what do you mean ? it will help us to relax .	2
3	do you really think so ? i don't . it will jus...	2
4	i guess you are right.but what shall we do ? i...	2

```
df_val = pd.read_csv('/content/drive/MyDrive/Enhance It/Training Projects/NLP/dev.csv')
df_val.drop(['Emotion','Dialogue_ID'], inplace=True, axis=1)
df_val['Dialogue_Act'] = df_val['Dialogue_Act'].apply(numLabel)
df_val.head()
```

	Utterance	Dialogue_Act

```
df_test = pd.read_csv('/content/drive/MyDrive/Enhance It/Training Projects/NLP/test.csv')
df_test.drop(['Emotion','Dialogue_ID'], inplace=True, axis=1)
df_test['Dialogue_Act'] = df_test['Dialogue_Act'].apply(numLabel)
df_test.head()
```

	Utterance	Dialogue_Act
0	hey man , you wanna buy some weed ?	0
1	some what ?	2
2	weed ! you know ? pot , ganja , mary jane some...	0
3	oh , umm , no thanks .	0
4	i also have blow if you prefer to do a few lin...	0

```
#Check the size of the data
print(f"Size of training set: {df_train.shape}")
print(f"Size of validation set: {df_val.shape}")
print(f"Size of test set: {df_test.shape}\n")

#Check and delete for duplicates
print(f"Number of duplicate rows in training: {df_train[df_train.duplicated()].shape}")
df_train.drop_duplicates(subset=None, keep="first", inplace=True)
print(f"Number of duplicate rows in validation: {df_val[df_val.duplicated()].shape}")
df_val.drop_duplicates(subset=None, keep="first", inplace=True)
print(f"Number of duplicate rows in test: {df_test[df_test.duplicated()].shape}\n")
df_test.drop_duplicates(subset=None, keep="first", inplace=True)

print(f"Final size of train: {df_train.shape}")
print(f"Final size of validation: {df_val.shape}")
print(f"Final size of test: {df_test.shape}")

df_train = df_train.reset_index()
df_val = df_val.reset_index()
df_test = df_test.reset_index()
```

```
Size of training set: (87170, 2)
Size of validation set: (8069, 2)
Size of test set: (7740, 2)

Number of duplicate rows in training: (14779, 2)
Number of duplicate rows in validation: (387, 2)
Number of duplicate rows in test: (271, 2)

Final size of train: (72391, 2)
Final size of validation: (7682, 2)
Final size of test: (7469, 2)
```

Data Preprocessing

```
def preprocess(sentence,lemma=True):
  #Remove url links
  proc_sent = re.sub(r'https?:\/\/.*[\r\n]*','',sentence)

  #Delete non-ASCII values
  proc_sent = str(proc_sent.encode("ascii","ignore"))[1:]

  #Remove punctuation
  proc_sent = ''.join([char for char in proc_sent if char not in string.punctuation])

  #Parse to lower case
  proc_sent = proc_sent.lower()

  #Tokenize and remove stop words
  stop_words = stopwords.words('english')
  proc_sent = word_tokenize(proc_sent)
  proc_sent = [word for word in proc_sent if word not in stop_words]

  #Remove single letters
  proc_sent = [word for word in proc_sent if len(word) != 1]

  #Stem or lemmatize (just 1)
  if lemma:
    lemmatizer = WordNetLemmatizer()
    proc_sent = [lemmatizer.lemmatize(word) for word in proc_sent]
  else:
    porter = PorterStemmer()
    proc_sent = [porter.stem(word) for word in proc_sent] #Stemming

  return proc_sent


# df_train['Utterance'] = df_train['Utterance'].apply(preprocess)
print(f"{len(df_train)} train observations...")
# df_val['Utterance'] = df_val['Utterance'].apply(preprocess)
print(f"{len(df_val)} validation observations...")
# df_test['Utterance'] = df_test['Utterance'].apply(preprocess)
print(f"{len(df_test)} test observations...")
```

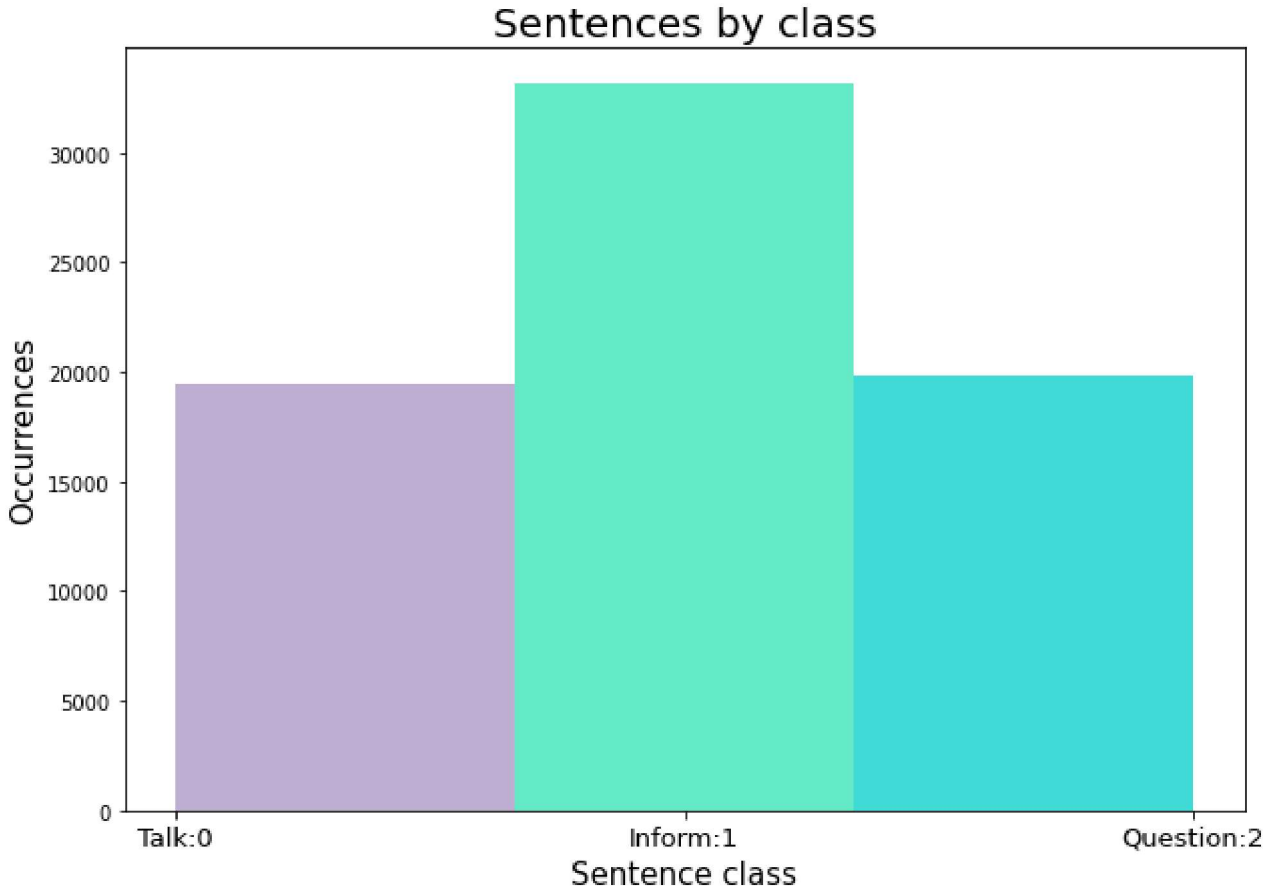
```
72391 train observations...
7682 validation observations...
7469 test observations...
```

```
df_train.drop('index', inplace = True, axis = 1)
df_val.drop('index', inplace = True, axis = 1)
df_test.drop('index', inplace = True, axis = 1)
```

```
plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
N, bins, patches = plt.hist(df_train['Dialogue_Act'], bins=3)
ax.set_xticks(np.arange(3))
ax.set_xticklabels(['Talk:0','Inform:1','Question:2'], size=13)
for i in range(len(N)):
  patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))

plt.ylabel("Occurrences", size=15)
plt.xlabel("Sentence class", size=15)
plt.title("Sentences by class", size=20)
```

```
Text(0.5, 1.0, 'Sentences by class')
<Figure size 432x288 with 0 Axes>
```

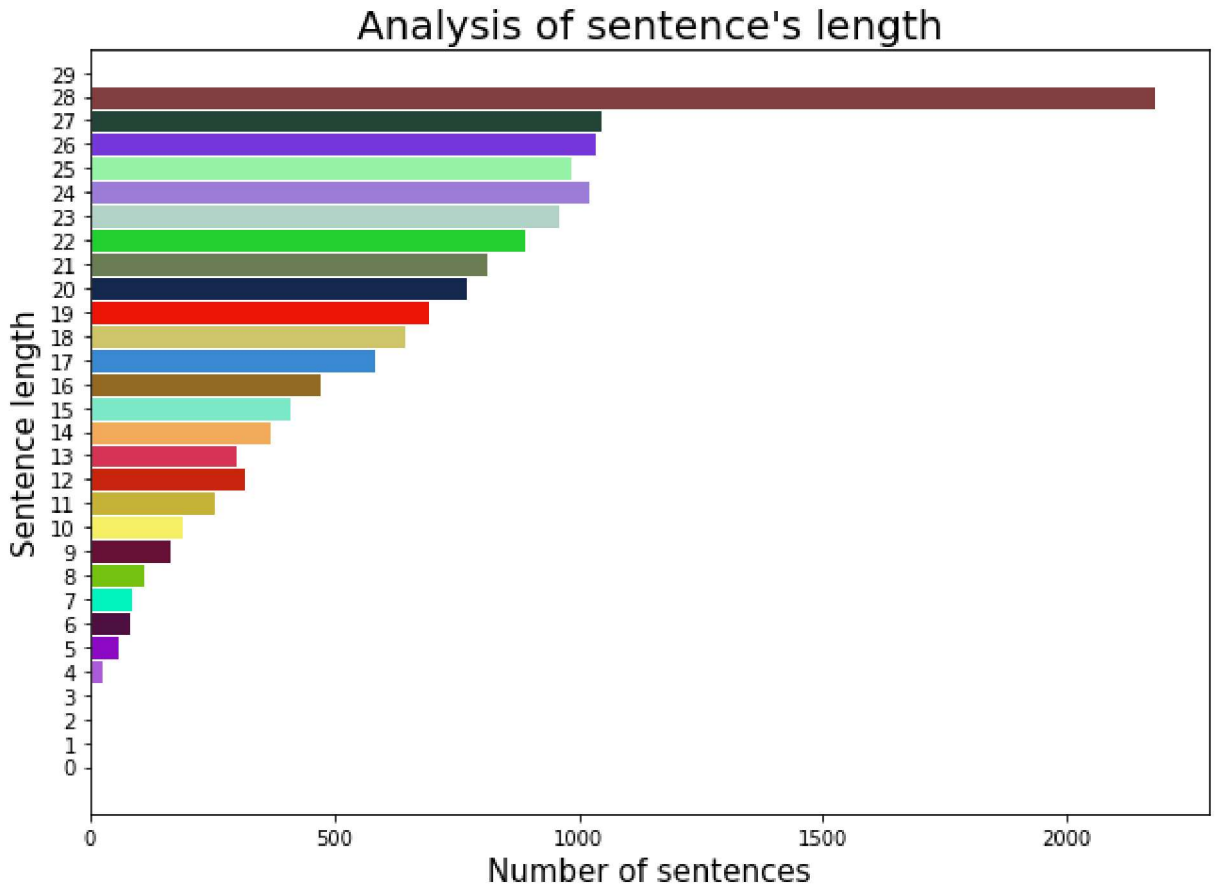



```
lengths = [len(i) for i in df_train['Utterance']]
uniques = np.unique(lengths)
bins = [i for i in range(30)]

plt.figure()
fig, ax = plt.subplots(figsize=(10,7))
N, bins, patches = plt.hist(lengths, bins=bins, orientation='horizontal', edgecolor='white',
ax.set_yticks(np.arange(30))
# ax.set_yticklabels(uniques, size=10)
for i in range(len(N)):
  patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))
```

```
plt.ylabel("Sentence length", size=15)
plt.xlabel("Number of sentences", size=15)
plt.title("Analysis of sentence's length", size=20)
```

```
Text(0.5, 1.0, "Analysis of sentence's length")
<Figure size 432x288 with 0 Axes>
```




```
# #Remove short sentences
# size = int(np.mean(np.arange(2,16)))

# for i in range(len(data['string'])):
#   #Save as is if the size is between the margins
#   #Clip the long sentences to make them shorter
#   if len(data['string'][i]) > 16:
#     data['string'][i] = data['string'][i][:size]
#   elif len(data['string'][i]) < 2:
#     data.drop([i], axis=0, inplace=True)

# data = data.reset_index()
# print(f"{len(data)} observations...")
# print(f"Mean size for clipping sentences: {size}")


# lengths = [len(i) for i in data['string']]
# uniques = np.unique(lengths)
# print(uniques)
```

```
# bins = [i for i in range(20)]

# plt.figure()
# fig, ax = plt.subplots(figsize=(10,7))
# N, bins, patches = plt.hist(lengths, bins=bins, orientation='horizontal', edgecolor='white'
# ax.set_yticks(np.arange(18))
# for i in range(len(N)):
#   patches[i].set_facecolor("#" + ''.join(random.choices("ABCDEF" + string.digits, k=6)))

# plt.ylabel("Sentence length", size=15)
# plt.xlabel("Number of sentences", size=15)
# plt.title("Analysis of stemmed sentences", size=20)


def pd_to_tensor_data (data):
  new_tensor_data = tf.data.Dataset.from_tensor_slices(
    (
        tf.cast(data["Utterance"].values, tf.string),
        tf.cast(data["Dialogue_Act"].values, tf.int64)
        )
  ).batch(1)
  return new_tensor_data


# def join_text (text):
#   return " ".join(text)


# df_train['Utterance'] = df_train['Utterance'].apply(join_text)
# df_val['Utterance'] = df_val['Utterance'].apply(join_text)
# df_test['Utterance'] = df_test['Utterance'].apply(join_text)


tensor_train = pd_to_tensor_data(df_train)
tensor_validation = pd_to_tensor_data(df_val)
tensor_test = pd_to_tensor_data(df_test)
```

Model Creation

Create the embeddings

```
# embedding = "https://tfhub.dev/google/universal-sentence-encoder/4"
# hub_layer = hub.KerasLayer(embedding,
#                            input_shape=[],
#                            dtype=tf.string,
#                            trainable=True)


tfhub_handle_encoder = 'https://tfhub.dev/tensorflow/albert_en_base/2'
```

```
tfhub_handle_preprocess = 'https://tfhub.dev/tensorflow/albert_en_preprocess/3'

def build_classifier_model():
  text_input = tf.keras.layers.Input(shape=(),
                                     dtype=tf.string,
                                     name='text')
  preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess,
                                       name='preprocessing')
  encoder_inputs = preprocessing_layer(text_input)
  encoder = hub.KerasLayer(tfhub_handle_encoder,
                           trainable=False,
                           name = 'BERT_encoder')
  outputs = encoder(encoder_inputs)
  net = outputs['pooled_output']
  net = tf.keras.layers.Dropout(0.1)(net)
  net = tf.keras.layers.Dense(3,
                              activation = 'softmax',
                              name = 'classifier') (net)
  return tf.keras.Model(text_input, net)


classifier_model = build_classifier_model()


epochs = 20
steps_per_epoch = tf.data.experimental.cardinality(tensor_train).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 5e-4
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                          num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                          optimizer_type='adamw')


loss = tf.keras.losses.SparseCategoricalCrossentropy()
metrics = tf.metrics.SparseCategoricalAccuracy()


classifier_model.compile(optimizer=optimizer,
                         loss=loss,
                         metrics = metrics)


history = classifier_model.fit(x=tensor_train,
                               validation_data=tensor_validation,
                               epochs=epochs)
```

```
Epoch 1/20
72391/72391 [==============================] - 1191s 16ms/step - loss: 1.2716 - sparse_c
Epoch 2/20
72391/72391 [==============================] - 1184s 16ms/step - loss: 1.0415 - sparse_c
```

```
Epoch 3/20
72391/72391 [==============================] - 1189s 16ms/step - loss: 0.9645 - sparse_c
Epoch 4/20
72391/72391 [==============================] - 1192s 16ms/step - loss: 0.9390 - sparse_c
Epoch 5/20
72391/72391 [==============================] - 1186s 16ms/step - loss: 0.9255 - sparse_c
Epoch 6/20
72391/72391 [==============================] - 1182s 16ms/step - loss: 0.9138 - sparse_c
Epoch 7/20
62336/72391 [========================>.....] - ETA: 2:29 - loss: 0.9048 - sparse_categor
```

```
loss, accuracy = classifier_model.evaluate(tensor_test)
print(f"Loss: {loss} and Accuracy: {accuracy}")


# history_dict = history.history
# print(history_dict.keys())

# acc = history_dict['sparse_categorical_accuracy']
# val_acc = history_dict['val_sparse_categorical_accuracy']
# loss = history_dict['loss']
# val_loss = history_dict['val_loss']

# epochs = range(1, len(acc) + 1)
# fig = plt.figure(figsize=(10, 6))
# fig.tight_layout()

# plt.subplot(2, 1, 1)
# # r is for "solid red line"
# plt.plot(epochs, loss, 'r', label='Training loss')
# # b is for "solid blue line"
# plt.plot(epochs, val_loss, 'b', label='Validation loss')
# plt.title('Training and validation loss')
# # plt.xlabel('Epochs')
# plt.ylabel('Loss')
# plt.legend()

# plt.subplot(2, 1, 2)
# plt.plot(epochs, acc, 'r', label='Training acc')
# plt.plot(epochs, val_acc, 'b', label='Validation acc')
# plt.title('Training and validation accuracy')
# plt.xlabel('Epochs')
# plt.ylabel('Accuracy')
# plt.legend(loc='lower right')


dataset_name = 'silicone_dyda'
saved_model_path = '/content/drive/MyDrive/Enhance It/Training Projects/NLP/{}_bert_20'.forma

classifier_model.save(saved_model_path, include_optimizer=False)
```

```
# model = keras.Sequential()
# model.add(hub_layer)  #Embedding layer
# model.add(keras.layers.Dense(1024, activation='relu'))
# model.add(keras.layers.Dropout(.2))
# model.add(keras.layers.Dense(512, activation='tanh'))
# model.add(keras.layers.Dropout(.2))
# model.add(keras.layers.Dense(512, activation='relu'))
# model.add(keras.layers.Dropout(.2))
# model.add(keras.layers.Dense(256, activation='relu'))
# model.add(keras.layers.Dropout(.2))
# model.add(keras.layers.Dense(128, activation='relu'))
# model.add(keras.layers.Dropout(.1))
# model.add(keras.layers.Dense(20, activation='relu'))
# model.add(keras.layers.Dropout(.2))
# model.add(keras.layers.Dense(3, activation='softmax'))
# model.summary()


# model.compile(optimizer = 'Nadam',
#               loss = keras.losses.SparseCategoricalCrossentropy(),
#               metrics = ['accuracy'])


# history = model.fit(tensor_train.shuffle(10000).batch(512),
#                     epochs = 100,
#                     validation_data = tensor_validation.batch(512),
#                     verbose = 1)


# plt.figure(figsize=(10,7))
# plt.plot(history.history['accuracy'])
# plt.plot(history.history['val_accuracy'])
# plt.title('Model Accuracy')
# plt.ylabel('Accuracy')
# plt.xlabel('Epoch')
# plt.legend(['Train', 'Validation'], loc='upper left')
# plt.show()

# plt.figure(figsize=(10,7))
# plt.plot(history.history['loss'])
# plt.plot(history.history['val_loss'])
# plt.title('Model Loss')
# plt.ylabel('Loss')
# plt.xlabel('Epoch')
# plt.legend(['Train', 'Validation'], loc='upper right')
# plt.show()
```

https://huggingface.co/datasets/silicone

https://huggingface.co/datasets/silicone/tree/main/dummy/dyda_da/1.0.0

Model implementation

Load the dataset

```
path = '/content/drive/MyDrive/Enhance It/Training Projects/NLP/my_model.h5'

load_model = keras.models.load_model((path), custom_objects={'KerasLayer': hub.KerasLayer})
```

```
path = '/content/drive/MyDrive/Enhance It/Training Projects/NLP/silicone_dyda_bert_20-2022061
reloaded_model = tf.keras.models.load_model(path)
```

```
WARNING:tensorflow:No training configuration found in save file, so the model was *not*
```

```
tf.keras.models.save_model(reloaded_model,"Albert_model.h5")
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be bu
```

```
path = '/content/Albert_model.h5'

load_model = keras.models.load_model((path), custom_objects={'KerasLayer': hub.KerasLayer})
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *r
WARNING:tensorflow:No training configuration found in the save file, so the model was *r
```

```
examples = "How old are you?"
reloaded_results = tf.nn.softmax(reloaded_model(tf.constant([examples])))
print(reloaded_results)
```

```
tf.Tensor([[0.21325038 0.21428524 0.5724644 ]], shape=(1, 3), dtype=float32)
```

```
from numpy.ma.core import argmax
phrase = 'How old are you?'

probs = load_model.predict(np.array([phrase]))
print(load_model.predict(np.array([phrase])))
```

```
[[0.00255847 0.00739947 0.9900421 ]]
```