Homework 2, FMNLP Fall 2022

You have ten days for this assignment, please submit as PDF(File>Print>Save as PDF). 100 points total.

If you have any further questions or concerns, contact the TA via email: borhanifard.zeinab@gmail.com or borhanifardz@ut.ac.ir

Telegram: @borhanifard

LINK: https://colab.research.google.com/drive/1yYuVjbAlUcor9SIsJmW6nWJ5GbC7-PgG?usp=sharing

---

IMPORTANT: After copying this notebook to your Google Drive, please paste a link to it below. To get a publicly-accessible link, hit the *Share* button at the top right, then click "Get shareable link" and copy over the result. If you fail to do this, you will receive no credit for this homework!

LINK: https://colab.research.google.com/drive/1yYuVjbAlUcor9SIsJmW6nWJ5GbC7-PgG?usp=sharing

---

How to submit this problem set:

- Write all the answers in this Colab notebook. Once you are finished, generate a PDF via (File -> Print -> Save as PDF) and upload it to Elearn (elearn.ut.ac.ir).
- **Important:** check your PDF before you submit to Elearn to make sure it exported correctly. If Colab gets confused about your syntax, it will sometimes terminate the PDF creation routine early.
- **Important:** on Elearn, please make sure that you tag each page with the corresponding question(s). This makes it significantly easier for our graders to grade submissions, especially with the long outputs of many of these cells. We will take off points for submissions that are not tagged.
- When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. One handy way to do this is by clicking `Runtime -> Run All` in the notebook menu.

Academic honesty

- We will audit the Colab notebooks from a set number of students, chosen at random. The audits will check that the code you wrote actually generates the answers in your PDF. If you turn in correct answers on your PDF without code that actually generates those answers, we will consider this a serious case of cheating. See the course page for honesty policies.
- We will also run automatic checks of Colab notebooks for plagiarism. Copying code from others is also considered a serious case of cheating.

Part 1: Annotation

In this homework, you will first collect a labeled dataset of 360 **Persian sentences** for a text classification task of your choice. You can do this in teams of size three. (Note that you can build teams of one or two as well but then you need to ask two or one other annotators to help you.) The process will include:

1. *Data collection*: Collect 360 Persian sentences from any source you find interesting (e.g., literature, Tweets, news articles, reviews, etc.)
2. *Task design*: Come up with a binary (i.e., only two labels) sentence-level classification task that you would like to perform on your sentences. Be creative, and make sure your task isn't too easy (e.g., perhaps the labels have some degree of subjectivity to them, or the task is otherwise complex for humans). Write up annotator guidelines/instructions on how you would like people to label your data
3. On your dataset, every example must be annotated by three persons. In order to get everything done on time, you need to complete the following steps:
    - Build annotation guideline

- Annotate the data (e.g., a spreadsheet or Google form)
- Collect the labeled data from each of the three annotators.
- Sanity check the data for basic cleanliness (are all examples annotated? are all labels allowable ones?)

4. Collect feedback from all the team members about the task including annotation time and obstacles encountered (e.g., maybe your guidelines were confusing! or maybe some sentences were particularly hard to annotate!)
5. Calculate and report inter-annotator agreement.
6. Aggregate output from three annotators to create final dataset.
7. Perform NLP experiments on your new dataset!

Question 1.1 (10 points):

Describe the source of your unlabeled data, why you chose it, and what kind of sentence selection process you used (if any) to choose 120 sentences for annotation. Also briefly describe the text classification task that you will be collecting labels for in the next section.

WRITE YOUR ANSWER HERE

> The source of our data is crawling multiple websites which had philosophical/religious sentences from authors such as Socrates and the prophet. (like this) It contains sentences from famous people and we randomly crawled 120*3 sentences.
>
> To make the task less objective and more subjective, we chose the binary classification task of whether a sentence is philosophical or religious. This task is challenging because the border between religious and philosophical concepts is not perfectly clear and might change based on the annotator's perspective. Here the guideline plays a vital role in preventing annotation discrepancy.

Question 1.2 (25 points):

Copy the annotation guidelines that you provided to your classmates below. We expect that these guidelines will be very detailed (and as such could be fairly long). You must include:

- The two categories for your binary classification problem, including the exact strings you want annotators to use while labeling.
- Descriptions of the categories and what they mean.
- Representative examples of each category (i.e., sentences from outside your dataset that you have manually labeled to give annotators an idea of how to perform the task)
- A discussion of tricky corner cases, and criteria to help the annotator decide them. If you look at the data and think about how an annotator could do the task, you will likely find a bunch of these!

COPY YOUR ANNOTATION GUIDELINES HERE. Please format them nicely so it's easy for us to read / grade :)

> **Description**
>
> Our binary classification task is to infer if a sentence is philosophical or religous. The two categories and the exact labels you should provide for each sentence are

"philosophical" or "religious". (To make this easier for you, the provided Google Sheet has Data Validation for labels and icnludes these two labels. You just need to select one from the drop down for each sentence. Anything other than these two labels will be rejected with the message "There was a problem. The data you entered in cell X violates the data validation rules set on this cell. Please enter one of the following values: philosophical, religious.")

Philosophical:

This category includes sentences relating or devoted to the study of the fundamental nature of knowledge, reality, and existence. Here are some philosophical examples in Farsi:

- مرا برای دزدیدن تکه نانی به زندان انداختند و پانزده سال در آنجا نان مجانی خوردم! این دیگر چه دنیایی است؟
- زندگی سراسر تجربه است. هرچه بیشتر تجربه کنید، بهتر است.
- در آن لحظه که به این نتیجه می رسید که همه چیز درست خواهد شد، حقیقتا نیز چنان خواهد شد.
- آدمی را آزمودن به کردار باید کرد نه به گفتار؛ چه بیشتر مردم، زشت کردار و نیکو گفتارند.

Religious:

This category includes sentences relating to a religion. Most prevalent characteristic of this category is being related to god and obedience to godly rules. Here are some religious examples in Farsi:

- خدایا مرا به اندازه یک چشم بر هم زدن، به خودم وامگذار.
- هرکس بر خدا توکل کند،خدا برایش کافی خواهد بود
- کسی که عبادت های خالصانه خود را به سوی خدا فرستد، پروردگار بزرگ برترین مصلحت را به سویش فرو خواهد فرستاد.
- آن کسی که نفسش را محاسبه کند، سود برده است و آن کسی که از محاسبه نفس غافل بماند، زیان دیده است.
- عاجزترین مردم کسی است که نتواند دعا کند.
- کوشنده ترین مردم کسی است که گناهان را رها سازد.

Corner Cases:

As you probably guessed, these two categories can sometimes be tricky to distinguish—especially when their concepts collide, or the main trait of a category is not present in a sentence.

For religious sentences, first, you should find the words about god. If it was not present, obedience, eternal life, praying, and deterministic promises of heaven and hell can be good signs.

Talking about life, less deterministic speaking and more doubt, discussing experiences, and learning from life can be good signs for philosophical sentences.

However, a philosophical sentence can also discuss death, life after death, or religious concepts.

- حتی مرگ نیز بر شیرینی طعم حیات می افزاید زیرا ما را نسبت به زندگانی وابسته و علاقمند می سازد.

What can help you in these sentences is the goal of the sentence. For instance, in this specific example, we know that religious concepts usually condemn life and promote the afterlife. In contrast, this sentence is clearly the opposite which is a sign of being philosophical.

Furthermore, a religious sentence can also discuss philosophical concepts such as wealth.

- همانا دینار و درهم پیشینیان شما را به هلاکت رساند و همین دو نیز هلاک کننده شماست.

What can make you decide better is your knowledge of religion learned at school, which we encourage you to review. In this specific example, the historical warning signifies a religious sentence.

Example of the provided sheet with the auto-validated labels.

برچسب	جملات
philosophical	علم در اصل چیزی جز پویش منظّم در طریق معرفت نیست...
philosophical	از شاعر نخواهید خودش را شرح دهد، نمی تواند
philosophical	شهامت استفاده از دلیل خود را داشته باشید. این شعار روشنگری است.
philosophical	من همان کار مادرم را می کنم. این حقایقی که من می گویم ، در ذهن شما هست ، من کمک می کنم که شما این نوزادهای موجود در زهدان ذهن خودتان را به دنیا بیاورید
philosophical	هر کس لطف بهار میخواهد باید دشواری زمستان را تحمل کند
philosophical	برای آدکه بتوانیم کسی را دوست داشته باشیم به زمان و آگاهی نیاز داریم . برای دوست داشتن ابتدا باید داوری کرد و برای برتری دادن ،باید نخست سنجید
philosophical	قدرتمندترین فردی کسی است که توان کنترل خودش را داشته باشد
religious	تا عقل کسی را نیازموده اید، به اسلام آوردن او وقعی نگذارید.
philosophical	مرگ دگم ، تولد اخلاق است.
religious	کارها چنان در سیطره تقدیر است که چاره اندیشی به مرگ می انجامد
religious	خداوند، مؤمن صاحب حرفه را دوست دارد.

Question 1.3 (5 points):

Write down the names and emails of the two classmates who will be annotating your data below. Once they are finished annotating, create two .csv files (annotator1.csv and annotator2.csv) that contains each annotator's labels. The file should have two columns with headers **text** and **label**, respectively. You will include these files in an email to the instructors account when you're finished with this homework.

The tweets.csv file provided as an example in Part 2 below uses the same format.

WRITE CLASSMATE 1 NAME/EMAIL HERE: Samin Mehdizadeh (saminsani162@gmail.com)

WRITE CLASSMATE 2 NAME/EMAIL HERE: Sepehr Kamahi (sepehr.kamahi1997@gmail.com)

Question 1.4 (10 points):

After both annotators have finished labeling the 120 sentences you gave them, ask them for feedback about your task and the provided annotation guidelines. If you were to collect more labeled data for this task in the future, what would you change from your current setup? Why? Please include a summary of annotator feedback (with specific examples that they found challenging to label) in your answer.

WRITE ANSWER HERE

> **Samin Mehdizadeh:**
>
> - feedback:
>
> اهنمايي هاي گفته شده با جزئيات زيادي مطرح شده بود كه باعث مي شد انجام تسك راحت تر باشد. براي مثال براي جملات مذهبي اينكه به دنبال چه كلماتي باشيم هم مطرح شده بود كه بسيار كمك كننده بود.هم چنين وجود مثال هاي متعدد به همراه توضيح هر دسته امكان تمايز و مقايسه بين دو دسته را ايجاد كرده بود. در قسمت آخر نيز كه به اشتراكات اين دو دسته اشاره شده بود باعث شده بود هنگام برچسب زني به اين نكات هم توجه شود. مثلا گفته شده بود كه سخنان فلسفي هم شامل مفاهيمي مانند مرگ يا بعد از مرگ هستند با اين حال بايد به هدف .اين جملات توجه شود كه با ذكر مثال به طور دقيق توضيح داده شده بود
>
> - challenging examples:
>
> :با اين حال به دليل چالشي بودن اين تسك هم چنان مثال هايي وجود داشت كه دسته بندي آن ها مشكل بود
> راستگو باش تا استقامت داشته باشي
> باتوجه به اينكه اين جمله هم حالت قطعي داشت هم درمورد مفاهيم زندگي صحبت كرده بود تشخيص مذهبي يا
> فلسفي بودن آن مشكل بود
> رفتار انسان از سه منبع نيرو ميگيرد: ميل،هيجان و عقل
> با توجه به متن هاي مذهبي خوانده شده معمولا جملات با اين ساختار مذهبي هستند اگرچه محتوي آن كه مربوط
> به ميل و هيجان است معمولا مورد تاكيد متن هاي مذهبي نيست و به همين علت دسته بندي آن مشكل بود
>
> **Sepehr Kamahi:**
>
> - Feedback:
> The guide was crystal clear, Being able to select labels from google sheet was very useful. In the case of corner cases, stating multiple philosophical and religious concepts really helped us to distinguish between the sentences. I can't think of any way to make this guide even a little bit better.

- Challenging examples:

  پس آن کس که دنبال توهّم و کنجکاوی دروغین رفت به حق نرسید

  کسی که آرزوهایش طولانی است کردارش نیز ناپسند است

  صبر نیز بر چهار پایه قرار دارد. شوق، هراس، زهد و انتظار

  راستگو باش تا استقامت داشته باشی

  کارها چنان در سیطره تقدیر است که چاره اندیشی به مرگ می انجامد

What would I change?

First of all I would gather more data than what was asked in this assignment. As we know, we are living in the era of big models and big data. The more data we have, the better model we can train. Consequently, this bigger dataset will also be more diverse, crawled from many websites and authors, and maybe even from books to help the generalization of the model and prevent overfitiing to specific authors and writing styles.

Question 1.5 (10 points):

Now, compute the inter-annotator agreement between your two annotators. Upload both .csv files to your Colab session (click the folder icon in the sidebar to the left of the screen). In the code cell below, read the data from the two files and compute both the raw agreement (% of examples for which both annotators agreed on the label) and the Cohen's Kappa. Feel free to use implementations in existing libraries (e.g., sklearn). After you're done, paste the numbers in the text cell that follows your code.

If you're curious, Cohen suggested the Kappa result be interpreted as follows: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

```
### WRITE CODE TO LOAD ANNOTATIONS AND
import pandas as pd
from sklearn import metrics
! gdown 1WnCqe5OStQ3K2v24w4XsIS47aZTsqJkg
! gdown 1XWixgfb9c8tzria2yrnAhUsbZIUcWDAb
annotator1 = pd.read_csv("./mohsen-(annotator-samin) - mohsen.csv")
annotator2 = pd.read_csv("./mohsen-(annotator-sepehr) - mohsen.csv")

### COMPUTE AGREEMENT + COHEN'S KAPPA HERE!
raw_agreement = (annotator1["label"] == annotator2["label"]).sum() / len(annotator1["lab
cohen_agreement = metrics.cohen_kappa_score(annotator1["label"], annotator2["label"])
print("Raw Agreement:".ljust(18), raw_agreement)
print("Cohan's Agreement:".ljust(18), cohen_agreement)
```

```
Downloading...
From: https://drive.google.com/uc?id=1WnCqe5OStQ3K2v24w4XsIS47aZTsqJkg
To: /content/mohsen-(annotator-samin) - mohsen.csv
100% 18.7k/18.7k [00:00<00:00, 20.2MB/s]
Downloading...
From: https://drive.google.com/uc?id=1XWixgfb9c8tzria2yrnAhUsbZIUcWDAb
To: /content/mohsen-(annotator-sepehr) - mohsen.csv
100% 18.6k/18.6k [00:00<00:00, 22.0MB/s]
Downloading...
From: https://drive.google.com/uc?id=1B3pWxmPGzshQNwNVDEixf-PpiRl--Mf2
To: /content/final_dataset.csv
100% 68.5k/68.5k [00:00<00:00, 57.4MB/s]
Raw Agreement:     0.85
Cohan's Agreement: 0.6995827538247565
```

RAW AGREEMENT:

> 0.85

COHEN'S KAPPA:

> 0.699
>
> This level of agreement falls in the category of substantial agreement which is good for this callenging and subjective dataset.

Question 1.6 (10 points):

To form your final dataset, you need to *aggregate* the annotations from both annotators (i.e., for cases where they disagree, you need to choose a single label). Use any method you like other than random label selection to perform this aggregation (e.g., have the two annotators discuss each disagreement and come to consensus, or choose the label you agree with the most). Upload your final dataset to the Colab session (in the same format as the other two files) as final_dataset.csv. Remember to include this file in your final email to us!

DESCRIBE YOUR AGGREGATION STRATEGY HERE

> I reviewed the disagreements and chose the label I agreed with the most. The final dataset is downloaded in the following cell. I used sheet capabilites and conditional formatting to easily spot the differences.

text	label	annotator1	annotator2	agreement
نمی توانم چیزی به دیگران بیاموزم ؛ فقط می توانم وادارشان کنم که بیندیشند	philosophical	philosophical	philosophical	TRUE
سگ ، همان روحی را دارد که فیلسوف	philosophical	philosophical	philosophical	TRUE
پس کسی که درست اندیشید به ژرفای دانش رسید و آن کس که به حقیقت دانش رسید، از چشمه زلال شریعت نوشید	religious	religious	religious	TRUE
بدانید که خدا سخت کیفر است	religious	religious	religious	TRUE
فتار انسان از سه منبع دیرو میگیرد: میل، هیجان، و عقل	philosophical	philosophical	religious	FALSE
آن قدر بر مال دنیا حریص مباش که از مفقود شدنش اندوهناک شوی	religious	religious	religious	TRUE
موفقیت در دل مردان ایجاد غرور و کوری میکند، در حالی که رنج به آن‌ها می‌آموزد که صبور و نیرومند باشند	philosophical	philosophical	philosophical	TRUE
انسان آزاد آفریده شده است اما همیشه در زنجیری است که خود باقته است	religious	philosophical	religious	FALSE
از دانسزا گفتن، به جز کینه مردم، سودی نمی بری	religious	religious	religious	TRUE
چالاک باش تا هوشیار باشی	philosophical	philosophical	philosophical	TRUE

```
! gdown 1B3pWxmPGzshQNwNVDEixf-PpiRl--Mf2
```

```
Downloading...
From: https://drive.google.com/uc?id=1B3pWxmPGzshQNwNVDEixf-PpiRl--Mf2
To: /content/final_dataset.csv
100% 68.5k/68.5k [00:00<00:00, 43.7MB/s]
```

Part 2: Text classification

Now we'll move onto fine-tuning pretrained language models specifically on your dataset. This part of the homework is meant to be an introduction to the HuggingFace library, and it contains code that will potentially be useful for your final projects. Since we're dealing with large models, the first step is to change to a GPU runtime.

Adding a hardware accelerator

Please go to the menu and add a GPU as follows:

```
Edit > Notebook Settings > Hardware accelerator > (GPU)
```

Run the following cell to confirm that the GPU is detected.

```
import torch

# Confirm that the GPU is detected

assert torch.cuda.is_available()

# Get the GPU device name.
device_name = torch.cuda.get_device_name()
n_gpu = torch.cuda.device_count()
print(f"Found device: {device_name}, n_gpu: {n_gpu}")
device = torch.device("cuda")
```

```
Found device: Tesla T4, n_gpu: 1
```

Installing Hugging Face's Transformers library

We will use Hugging Face's Transformers (https://github.com/huggingface/transformers), an open-source library that provides general-purpose architectures for natural language understanding and generation with a collection of various pretrained models made by the NLP community. This library will allow us to easily use pretrained models like `BERT` and perform experiments on top of them. We can use these models to solve downstream target tasks, such as text classification, question answering, and sequence labeling.

Run the following cell to install Hugging Face's Transformers library and download a sample data file called tweets.csv that contains tweets about airlines along with a negative, neutral, or positive sentiment rating. Note that you will be asked to link with your Google Drive account to download some of these files. If you're concerned about security risks (there have not been any issues in previous semesters), feel free to make a new Google account and use it for this homework!

```
!pip install transformers
!pip install -U -q PyDrive

from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
print('success!')

import os
import zipfile

# Download helper functions file
helper_file = drive.CreateFile({'id': '16HW-z9Y1tM3gZ_vFpJAuwUDohz91Aac-'})
helper_file.GetContentFile('helpers.py')
print('helper file downloaded! (helpers.py)')

# Download sample file of tweets
data_file = drive.CreateFile({'id': '1QcoAmjOYRtsMX7njjQTYooIbJHPc6Ese'})
data_file.GetContentFile('tweets.csv')
print('sample tweets downloaded! (tweets.csv)')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/pub
Requirement already satisfied: transformers in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-pack
```

```
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/pyth
Requirement already satisfied: huggingface-hub<1.0,>=0.10.0 in /usr/local/lib/python3.7
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/d
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dis
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/li
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packa
success!
helper file downloaded! (helpers.py)
sample tweets downloaded! (tweets.csv)
```

The cell below imports some helper functions we wrote to demonstrate the task on the sample tweet dataset.

```
1 from helpers import tokenize_and_format, flat_accuracy
```

Part 1: Data Prep and Model Specifications

Upload your data using the file explorer to the left. We have provided a function below to tokenize and format your data as BERT requires. Make sure that your csv file, titled final_data.csv, has one column "text" and another column "labels" containing integers.

If you run the cell below without modifications, it will run on the tweets.csv example data we have provided. It imports some helper functions we wrote to demonstrate the task on the sample tweet dataset. You should first run all of the following cells with tweets.csv just to see how everything works. Then, once you understand the whole preprocessing / fine-tuning process, change the csv in the below cell to your final_data.csv file, add any extra preprocessing code you wish, and then run the cells again on your own data.

```
1 # REPLACE 'bert-base-uncased' in helpers.py with "HooshvareLab/bert-base-parsbert-uncase
2 # !!! REPLACE BEFORE RUNNING THIS !!! (Restart if you executed this beforehand)
3
4 from helpers import tokenize_and_format, flat_accuracy
5 import pandas as pd
6
7 ! gdown 1B3pWxmPGzshQNwNVDEixf-PpiRl--Mf2
8 df = pd.read_csv('final_data.csv')
```

```
df["label"] = pd.factorize(df["labels"])[0]
# df = pd.read_csv('tweets.csv')

df = df.sample(frac=1).reset_index(drop=True)

texts = df.text.values
labels = df.label.values

### tokenize_and_format() is a helper function provided in helpers.py ###
input_ids, attention_masks = tokenize_and_format(texts)

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Print sentence 0, now as a list of IDs.
print('Original: ', texts[0])
print('Token IDs:', input_ids[0])
```

```
Downloading...
From: https://drive.google.com/uc?id=1B3pWxmPGzshQNwNVDEixf-PpiRl--Mf2
To: /content/final_data.csv
100% 68.5k/68.5k [00:00<00:00, 73.3MB/s]
Original:  ی که با فاسقان دشمنی کند و برای خدا خشم گیرد، خدا هم برای او خشم آورد، و روز قیامت او را خشنود سازد
Token IDs: tensor([  101,  1298,  1293, 29824, 29837,  1293, 14157,  1271, 25573,  1291
        25573, 29824, 29834, 18511,  1278, 29825, 22192, 15915, 29837,  1293,
        15915, 15394,  1298,  1271, 17149, 25573, 29837,  1277, 15394, 25573,
         1277, 29825, 22192,  1305, 14498, 17149, 15394,  1268,  1277, 15394,
        25573,  1297, 22192,  1271, 17149, 25573, 29837,  1270, 29836,  1277,
        29825, 22192,  1270, 29836, 17149, 15394,  1268,  1298,  1280, 29836,
        29823,  1292, 14498,   102])
```

Create train/test/validation splits

Here we split your dataset into 3 parts: a training set, a validation set, and a testing set. Each item in your dataset will be a 3-tuple containing an input_id tensor, an attention_mask tensor, and a label tensor.

```

total = len(df)

num_train = int(total * .8)
num_val = int(total * .1)
num_test = total - num_train - num_val

# make lists of 3-tuples (already shuffled the dataframe in cell above)

train_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train)]
```

```
val_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train, num_v
test_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_val + num_t

train_text = [texts[i] for i in range(num_train)]
val_text = [texts[i] for i in range(num_train, num_val+num_train)]
test_text = [texts[i] for i in range(num_val + num_train, total)]
```

Here we choose the model we want to finetune from https://huggingface.co/transformers/pretrained_models.html. Because the task requires us to label sentences, we wil be using BertForSequenceClassification below. You may see a warning that states that `some weights of the model checkpoint at [model name] were not used when initializing. . .` This warning is expected and means that you should fine-tune your pre-trained model before using it on your downstream task. See here for more info.

```
from transformers import BertForSequenceClassification, AdamW, BertConfig

model = BertForSequenceClassification.from_pretrained(
    "HooshvareLab/bert-base-parsbert-uncased", # Use the 12-layer BERT model, with an un
    num_labels = 2, # The number of output labels.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()

```

```
Some weights of the model checkpoint at HooshvareLab/bert-base-parsbert-uncased were
- This IS expected if you are initializing BertForSequenceClassification from the che
- This IS NOT expected if you are initializing BertForSequenceClassification from the
Some weights of BertForSequenceClassification were not initialized from the model che
You should probably TRAIN this model on a down-stream task to be able to use it for p
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(100000, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
```

```
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (1): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
      (intermediate_act_fn): GELUActivation()
```

ACTION REQUIRED

Define your fine-tuning hyperparameters in the cell below (we have randomly picked some values to start with). We want you to experiment with different configurations to find the one that works best (i.e., highest accuracy) on your validation set. Feel free to also change pretrained models to others available in the HuggingFace library (you'll have to modify the cell above to do this). You might find papers on BERT fine-tuning stability (e.g., Mosbach et al., ICLR 2021) to be of interest.

```
batch_size = 16
optimizer = AdamW(model.parameters(),
                  lr = 1e-5, # args.learning_rate - default is 5e-5
                  eps = 1e-6 # args.adam_epsilon  - default is 1e-8
                )
epochs = 20
```

Fine-tune your model

Here we provide code for fine-tuning your model, monitoring the loss, and checking your validation accuracy. Rerun both of the below cells when you change your hyperparameters above.

```
import numpy as np
# function to get validation accuracy
def get_validation_performance(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0

    for i in range(num_batches):

      end_index = min(batch_size * (i+1), len(val_set))

      batch = val_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Tell pytorch not to bother with constructing the compute graph during
      # the forward pass, since this is only needed for backprop (training).
      with torch.no_grad():

        # Forward pass, calculate logit predictions.
        outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)
        loss = outputs.loss
        logits = outputs.logits

```

```
        # Accumulate the validation loss.
        total_eval_loss += loss.item()

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the number of correctly labeled examples in batch
        pred_flat = np.argmax(logits, axis=1).flatten()
        labels_flat = label_ids.flatten()
        num_correct = np.sum(pred_flat == labels_flat)
        total_correct += num_correct

    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_correct / len(val_set)
    return avg_val_accuracy
```

```
import random

# training loop

# For each epoch...
for epoch_i in range(0, epochs):
    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode.
    model.train()

    # For each batch of training data...
    num_batches = int(len(train_set)/batch_size) + 1

    for i in range(num_batches):
      end_index = min(batch_size * (i+1), len(train_set))

      batch = train_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

```

```
      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Clear the previously calculated gradient
      model.zero_grad()

      # Perform a forward pass (evaluate the model on this training batch).
      outputs = model(b_input_ids,
                              token_type_ids=None,
                              attention_mask=b_input_mask,
                              labels=b_labels)
      loss = outputs.loss
      logits = outputs.logits

      total_train_loss += loss.item()

      # Perform a backward pass to calculate the gradients.
      loss.backward()

      # Update parameters and take a step using the computed gradient.
      optimizer.step()

    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set. Implement this function in the cell above.
    print(f"Total loss: {total_train_loss}")
    val_acc = get_validation_performance(val_set)
    print(f"Validation accuracy: {val_acc}")

print("")
print("Training complete!")

```

```
======== Epoch 1 / 20 ========
Training...
Total loss: 12.226588547229767
Validation accuracy: 0.75

======== Epoch 2 / 20 ========
Training...
Total loss: 10.503029853105545
Validation accuracy: 0.75

======== Epoch 3 / 20 ========
Training...
Total loss: 9.638869136571884
Validation accuracy: 0.7222222222222222
```

```
======== Epoch 4 / 20 ========
Training...
Total loss: 8.861823290586472
Validation accuracy: 0.7222222222222222

======== Epoch 5 / 20 ========
Training...
Total loss: 7.998718351125717
Validation accuracy: 0.7222222222222222

======== Epoch 6 / 20 ========
Training...
Total loss: 6.453018128871918
Validation accuracy: 0.75

======== Epoch 7 / 20 ========
Training...
Total loss: 5.191357970237732
Validation accuracy: 0.8333333333333334

======== Epoch 8 / 20 ========
Training...
Total loss: 4.074896186590195
Validation accuracy: 0.6944444444444444

======== Epoch 9 / 20 ========
Training...
Total loss: 3.0472105741500854
Validation accuracy: 0.6388888888888888

======== Epoch 10 / 20 ========
Training...
Total loss: 2.5034476220607758
Validation accuracy: 0.7222222222222222

======== Epoch 11 / 20 ========
Training...
Total loss: 1.929030615836382
Validation accuracy: 0.7777777777777778

======== Epoch 12 / 20 ========
Training...
```

Evaluate your model on the test set

After you're satisfied with your hyperparameters (i.e., you're unable to achieve higher validation accuracy by modifying them further), it's time to evaluate your model on the test set! Run the below cell to compute test set accuracy.

```
1 get_validation_performance(test_set)
```

```
0.75
```

Question 2.1 (10 points):

Congratulations! You've now gone through the entire fine-tuning process and created a model for your downstream task. Two more questions left :) First, describe your hyperparameter selection process in words. If you based your process on any research papers or websites, please reference them. Why do you think the hyperparameters you ended up choosing worked better than others? Also, is there a significant discrepancy between your test and validation accuracy? Why do you think this is the case?

WRITE YOUR ANSWER HERE

> Firstly, because the dataset is really small, we should use smaller batch size as well (I used it based on https://openreview.net/pdf?id=nzpLWnVAyah). Moreover, a smaller learning rate also is crucial for this small dataset for not overfiting to the data. For adam eps, the huggingface default value for it is 1e-6 (https://huggingface.co/docs/transformers/main_classes/optimizer_schedules). The epochs is the result of running multiple times and watching where the validation accuracy decreases as the training loss still going down. This is overfiting and here we stop that by manually changing epochs but the corerct way of doing this is to use early stopping mechanism.
>
> Sometimes there is a significant discrepancy between test and validation accuracy. It is due to small of size of the dataset and teherfore very small size of validation and test sets. By increasing the size of dataset, these discrepencies shall vanish and we could have more smooth results with less noise.

Question 2.2 (20 points):

Finally, perform an *error analysis* on your model. This is good practice for your final project. Write some code in the below code cell to print out the text of up to five test set examples that your model gets **wrong**. If your model gets more than five test examples wrong, randomly choose five of them to analyze. If your model gets fewer than five examples wrong, please design five test examples that fool your model (i.e., *adversarial examples*). Then, in the following text cell, perform a qualitative analysis of these examples. See if you can figure out any reasons for errors that you observe, or if you have any informed guesses (e.g., common linguistic properties of these particular examples). Does this analysis suggest any possible future steps to improve your classifier?

```
## YOUR ERROR ANALYSIS CODE HERE
## print out up to 5 test set examples (or adversarial examples) that your model gets wr
def get_preds(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0
    preds = []
    for i in range(num_batches):

      end_index = min(batch_size * (i+1), len(val_set))

      batch = val_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Tell pytorch not to bother with constructing the compute graph during
      # the forward pass, since this is only needed for backprop (training).
      with torch.no_grad():

        # Forward pass, calculate logit predictions.
        outputs = model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)
        loss = outputs.loss
        logits = outputs.logits

        # Accumulate the validation loss.
        total_eval_loss += loss.item()

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the number of correctly labeled examples in batch
```

```
        pred_flat = np.argmax(logits, axis=1).flatten()
        labels_flat = label_ids.flatten()
        num_correct = np.sum(pred_flat == labels_flat)
        total_correct += num_correct
        preds.extend(pred_flat)

    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_correct / len(val_set)
    return preds

test_preds = get_preds(test_set)
test_df = df.iloc[num_val + num_train: total]
test_df["preds"] = test_preds
pd.set_option('display.max_colwidth', None)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:64: SettingWithCopyWarning
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
```

	id	text	labels	annotator1	annotator2	agreement	label	preds
328	172	دین همیشه به رسمیت شناختن همه وظایف و تکالیف ما از طریق کتب آسمانی منجر می شود.	religious	religious	religious	True	1	0
331	224	زبان عاقل در پشت قلب اوست، و قلب احمق در پشت زبانش قرار دارد	philosophical	philosophical	philosophical	True	0	1
335	105	همیشه به شهادت تعداد کمی نگاه کنید: هرگز صداها را حساب نکنید ، فقط ببینید وزن آنها چقدر ارزش دارد.	religious	philosophical	religious	False	1	0

DESCRIBE YOUR QUALITATIVE ANALYSIS OF THE ABOVE EXAMPLES HERE

> First of all we see that 3 of 9 (33%) of errors were also annotation disagreements. This is important as the raw disagreement was only 15%. this shows that model also struggles more where the annotators did not have agreement which is also seen in multiple research papers such as https://aclanthology.org/2020.emnlp-main.746.pdf These examples include:
>
> - همیشه به شهادت تعداد کمی نگاه کنید: هرگز صداها را حساب نکنید ، فقط ببینید وزن آنها چقدر ارزش دارد.
> - انتقام دلیل سبکی عقل و پستی روح است
> - دانش، میراثی گرانبها، و آداب، زیورهای همیشه تازه، و اندیشه، آیینه ای شفّاف است
>
> Investigating errors shows that some of them are really tricky. Like

- زبان عاقل در پشت قلب اوست، و قلب احمق در پشت زبانش قرار دارد

was labeld philosophical, however, its expression style shows determinism and also is similar to religion quotes. So here it looks like the model has done better than annotators and has discovered errors and noise in dataset. This idea has been actually researched about and our PLMs are able to make a map of dataset and determine easy, hard, and noisy examples. (https://aclanthology.org/2020.emnlp-main.746.pdf)

- زیبارویی که می داند زیبایی ماندنی نیست پرستیدنی ست

Another examples shows that the word "پرستیدنی" has made the model to mistakenly label religious, which shows that despite its good results, it still can struggle and get overfit to specific words.

This analysis shows that more data can help the model to better generalize. For example, if we had more philosophical samples including the word "پرستیدنی", the model would learn that this word needs more context to determine the category of the sentence. Small datasets usually come with these complications and overreaction to specific words and phrases, instead of really understanding the whole sentence.

Finished? Remember to upload the PDF file of this notebook and url of your colab and your three dataset files (annotator1.csv, annotator2.csv, and final_data.csv) to Elearn (elearn.ut.ac.ir)

Colab paid products - Cancel contracts here