Personality Detection on Persian Dataset

Phase 2

Mohammadjavad Mehditabar - 98522049

NLP Phase 2

July 12, 2023

Contents

1 Commands and Project Structure

Please refer to GitHub repository of this project, and follow guidelines on phase 2 to run this project. I have also explained how each sub module works.

2 Word2Vec

2.1 Implementation Approach

I developed a word2vec model that works based on skip-gram method and use negative sampling loss to update its weights. vectors are 10 dimensional and context window is 5. number of iteration was 60.
model has been trained on each 4 traits that each trait has 2 label individually, thus at the end of training we have 8 models. models are saved in *models* directory under name convention *"{triat}_{num_trait}.word2vec.npy"*.
during training it was also needed to keep tokens because after that we need to query *token* dictionary and extract the idx of each token. tokens of this 8 available models are saved into *"experiments/word2vec/{triat}_{num_trait}_tokens.json"*.
And in the end for further uses I generated a model that was trained on whole dataset. this model will be queried as feature extractor for a word. this model name is *"all.word2vec.npy"*.

2.2 results

we will discuss about "E" and "I" trait and consider them representing our other models. we will dig through what biases our model has and how are the common words are differed from each other by their vector differences in these 2 traits. word vector differences are measured by *cosine similarity* metric.

2.2.1 word vector differences

0	
-0.25134	شما
0.07524	خونه
-0.04506	اتفاق
-0.07823	هیجان
-0.02406	تنهایی

The figure above shows cosine similarty between word vectors *I, E* for each selected word. as we see the word "خونه" which doesnt matter for Intorvert person or Extrovert person is being more similar to their context and they use this word in the same context. but the words "شما" is the most dissimialr word that means this word doesnt happens to be in the same context for Introvert and Extrovert person. which is reasonable because Introvert person would rather to use this word more and contact other person with this word but Extrovert person prefer to use the word "تو" more.

Other words such as "اتفاق ، هیجان" are clearly used in a different way for these 2 trait. Intorvert would less use this word but Extrovert rather to use more.

2.2.2 checking bias between two traits

زن-ترس	مرد-ترس	
0.1274	-0.30583	I
0.21861	0.3789	E

The figure above shows *cosine similarity* between words "مرد ، ترس" and words "زن و ترس" in both traits *I, E*. As we can see there is high dissimialrity in *I* trait between the word "مرد" and the word "ترس". that this represent that as our common knowledge the word *"fear"* and the word *"man"* happens to be less relatable but women are more similar with the word *"fear"*. But in the *E* trait surprisingly this fact doesn't hold that, in fact the contrary of this assumption is being shown. Therefore we can deduce that in E trait we don't have bias in word vecotrs or bias happens in an opposite way.

3 tokenization

3.1 Implementation Approach

tokenization process has been implemented by *SentencePiece* tokenizer which is imporved version of *WordPiece* tokenizer. In fact this tokenizer achieved more accuracy since languages which don't use *space* in their sentences to separate words are hard to be tokenized. So *SentencePeice* uses *"_"* character to represent spaces and act with total context in the same way.

Training this tokenizer has implemented with their official library. And for becoming more accurate we used *KFold Cross Validation* and we also considered different *vocab size* to make a total conclusion which model's performance is better. Model performances are assessed by computing ratio of *"UNK"* token on each fold and vocab size separately. models are saved into *"models/fold_{num_fold}_v_{vocab_size}"*.

3.2 Results

results based on each fold and vocab size separately. I have also computed average on all folds for better decision making.

	vocab-size-1000	vocab-size-2000	vocab-size-3000	vocab-size-4000
fold-1	9.867	9.732	9.595	9.459
fold-2	9.867	9.732	9.595	9.46
fold-3	9.866	9.731	9.593	9.457
fold-4	9.87	9.739	9.606	9.473
fold-5	9.867	9.732	9.595	9.46
all-average	9.867	9.733	9.597	9.462

we see that as vocab size grows, small improvement on *UNK* tokens happens. and this is reasonable because if vocab size increases, it can accomodate more vocab inside it so less *UNK* tokens appear.
we can also take a look at the average on all fold in a scatter plot shown below:

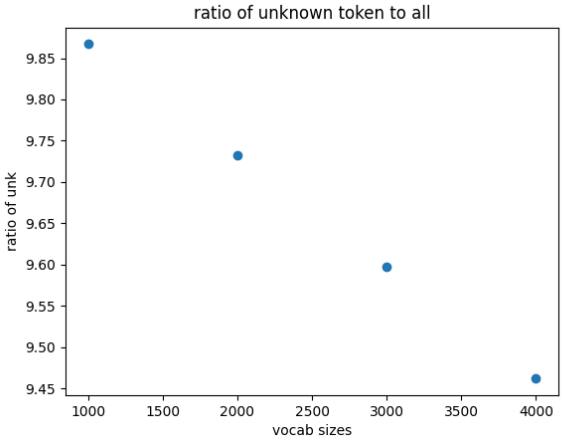


Figure 1: UNK token ratio

4 Language Model

4.1 Implementation Approach

This section tries to fine tune our dataset on *"GPT2-Medium"* and after fine tuning models will be saved. The saved model is then used to generate a text on each labels.
At first I was continuing with default *GPT2-Medium* but it didn't created a meaningful sentence then I tried to use some persian model that I came up with GPT2-Persian that would result in much better generation. models are saved into *models/[IE].language_model*.I will just cover the first trait *I,E* with more details.

4.2 Results

The examples of each labels are generated and then saved under the *stats/language_model* folder. So you can check their meaningfulness. I will get through the *I_example.txt* and *E_example.txt*. As we know recgonzing personality detection is such a hard task for even psychologist. for example with one or five sentences nobody can tell whether this person is Intorvert or Extrovert. But as much as I can, I'll check the generated sentences. we know that Extrovert people are more sociable and they tend to more interact with other people on the other hand Introvert people tend to be more isolated and don't like to communicate with other people and they are shy person.

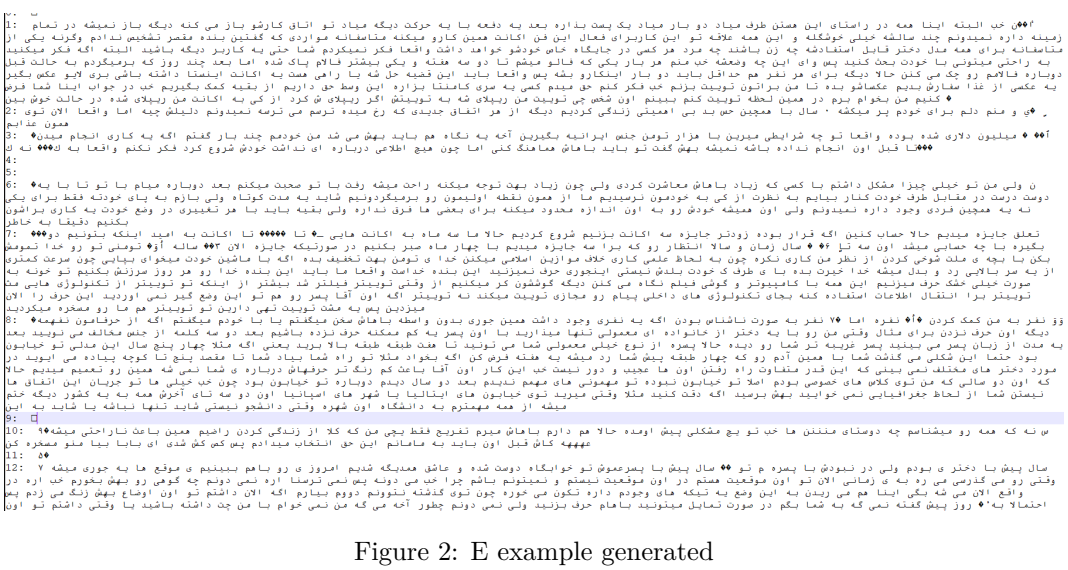


Figure 2: E example generated

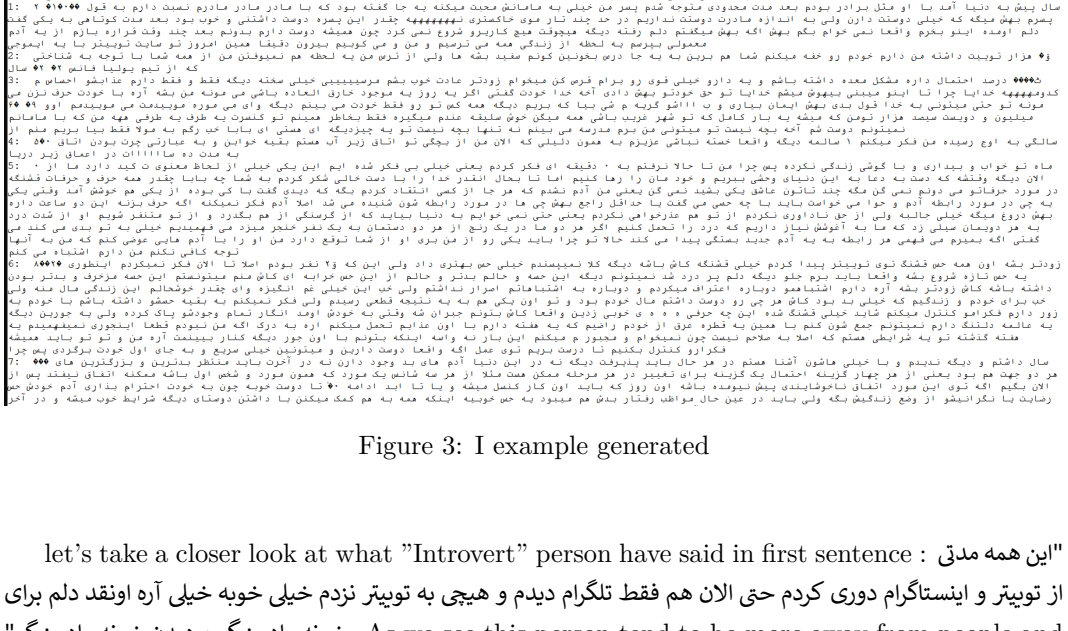


Figure 3: I example generated

let's take a closer look at what "Introvert" person have said in first sentence : "این همه مدتی از توییتر و اینستاگرام دوری کردم حتی الان هم فقط تلگرام دیدم و هیچی به توییتر نزدم خیلی خوبه خیلی آره اونقد دلم برای خونه مادربزرگ و دیدن خونه مادربزرگم". As we see this person tend to be more away from people and likes her past moments and don't want to socialize with other people. But if we examine a 8th sentence of "Extrovert" person : "اگه یه نفری وجود داشت همین جوری بدون واسطه باهاش سخن میگفتم". so this behavior is only seen in Extrovert people because they like to talk and they have capability to talk to other people even if they dont know them.

5 Feature Engineering

5.1 Implementation Approach

In this part our aim was to compare diffenret features and to conclude if we feed other feature besies our text data to our model, would model generalize and make decision better or not?
So I defined a Neural Network model which can accept all of the features that I mention in next part. my model features was:

- model architecture :
  - a input size of 768 nuerons(for sake of generalizing like BERT).
  - hidden layer of size 1000 with relu activation and dropout layer.
  - hidden layer of size 200 with relu activation and dropout layer.
  - output layer with one neuron and sigmoid activation
- model settings:
  - loss : Binary Cross Entropy
  - optimizer : Adam with learning rate = 0.001
  - epochs : 60
  - batch size : 32

inputs were fed in *"cuda"* format for utilization from gpu and training faster.
the features used in this experiments are exaplained below and I will get through the details of each input to be feedable to our model:

- sentence length : an array that first number in array is sentence length and the array was padded with zero to be in length 768.
- word length : an array that fills word length through 768 elements of it. if tweets has more word than 768 it will be truncated and if tweets has less length than 768 will be padded with zero to be in length 768
- words : assigning unique number to each word. I first created a *token dict* that each word has its unique number. And then tweets are tokenized each tokenized word are then mapped to their indexes. matching the dimension criterion was applied like previous approach.
- words-bigrams : just like previous feature but every consecutive two tokens are concatenated to represent one element of our array. array was filled until to get length 768. otherwise it will be padded by zero.
- word2vec : trained *all.word2vec.npy* model were used to extract each token vectors from it. and then vectors extracted are summed to be in same direction and then it was fed into model.

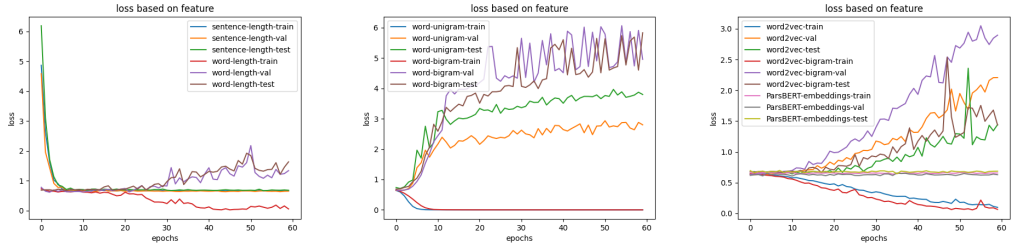
- word2vec-bigram : treated like before but for each two consecutive word two vectors are concatenated and then all these vectors are summed.
- ParsBERT : used bert-fa-base-uncased model to tokenize and then feed each tokenized input to extract vectors. like word2vec all these token vectors are summed to be in same dimension as model input.

models are saved into *models/feat_engineering_{feature_name}.pth*.

5.2 Results

since the whole plot wouldn't be located in one figure it has been plotted in multi figure.

5.2.1 loss

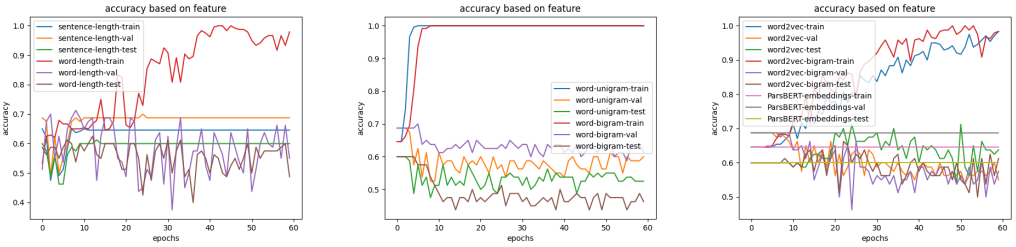


(a) loss A (b) loss B (c) loss C

Figure 4: loss over 7 trained model on 3 parts train / val / test

Figure 1 : As we see sentence length model on each 3 parts have been covnerged well. but word length model just perform fantastic in train part but on test and val doesn't generalize well. so it shows there is over-fit word length model.
Figure 2 : former behaviour is also seen in word-unigram and word-bigram model, so these models also overfit to just train data.
Figure 3 : both word2vec unigram and bi-gram are overfit to train data and they perform well on training data. best model seen so far can be ParsBERT model with a low loss and without any effect of overfitting.

5.2.2 accuracy



(a) acc A (b) acc B (c) acc C

Figure 5: accuracy over 7 trained model on 3 parts train / val / test

If we look at three figures closely we'll see effect of overfitting on 5 models from our 7 models. the only two models that don't overfit are "sentence length" and "ParsBERT" that ParsBERT with higher accuracy will defeat sentence length.

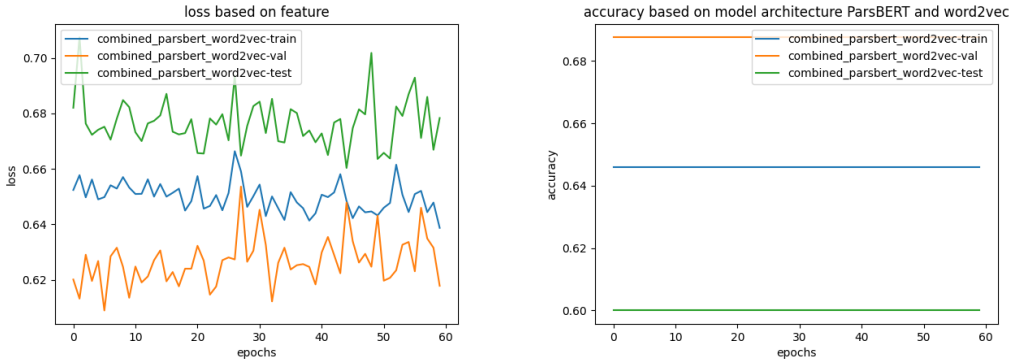
6 Model Architecture

6.1 Implementation Approach

As I see the progress on *transformer* models in recent years. I chose my three models have some usage of transformer in it. I will dig through the 3 model architecture I have implemented.

6.1.1 Word2Vec + ParsBERT

This model involve both Word2Vec and ParsBERT together. such that my base model is Neural Netwerk and the logics are just like as I explained in prevoius section but the input I fed into this model was vector of tokenized word on each model and then get average of these two different vecotrs and then feed it to the model. Since this model don't generlaize well(lack of sequence knowledge) I didnt try this model for other 3 traits.
Model is save into models are saved into *model_architecture_combined_parsbert_word2vec.pth* and the results of the models are convincing:



(a) loss

(b) acc

Figure 6: accuracy over 7 trained model on 3 parts train / val / test

6.1.2 BERT Truncated

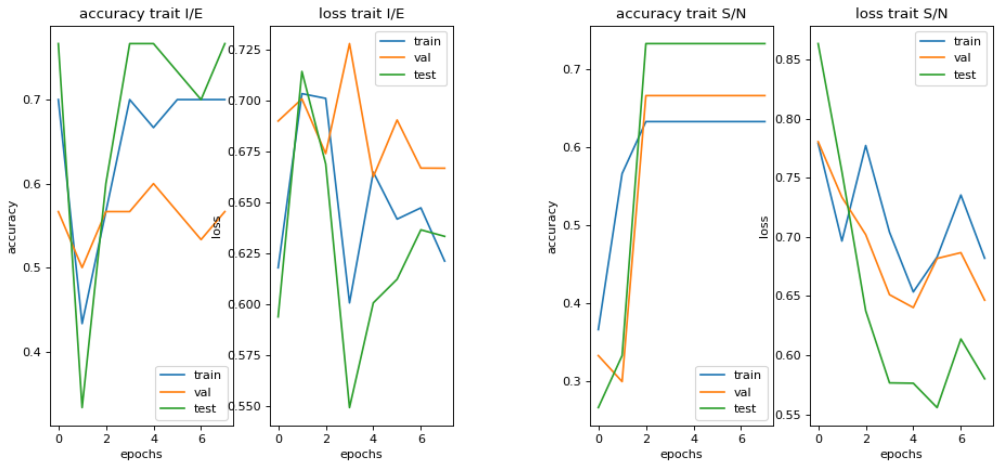
As in the name of section says, this model are feed tweets with truncated length to be not more than 512 tokens. during training this model, I have ran into serious GPU out of memory issue and I have tried many different solutions to overcome this problem. for example I used the built in *Trainer* object of pytroch for training or I decreased length of sentence which didn't lead to be escape way. At the end I tried to lowered *batch size* so that trade-off between batch size and model preserves. because lower batch size results in much more training time and higher batch size will result in GPU shortage. So I chose

batch size = 14 that happens to be the best batch size. through this batch size GPU usage was about 14GB. According to *colab* free account users don't have more than 15GB GPU.

Model Explaination

1. first of all I concatenated all tweets of user with space and this makes our input to be in string format instead of list.
2. after that I make 4 individual trait for training 4 four different models. I meant that "I,E" are together and then binary classified, "S, N" are together and then binary classified and etc.
3. then I splitted my dataset into 3 parts train/validation/test.
4. I defined Custom model which uses BERT CLS embeddings which stands for classification. and with this embeddings I managed to build a fully connected layer which maps (768, 1). as we know our output neuron should be one because we have binary classification and use sigmoid as activation function. and *BCELoss* for loss function and AdamW for optimizer. And in the end model was trained on 8 epochs.
5. I make use of pretrained ParsBERT model as my pretrained BERT embeddings.
6. Fine-tuning phase is applied on all layer even BERT layer.
7. The model is saved into *models/BERT_TRUNCATED_MODEL*

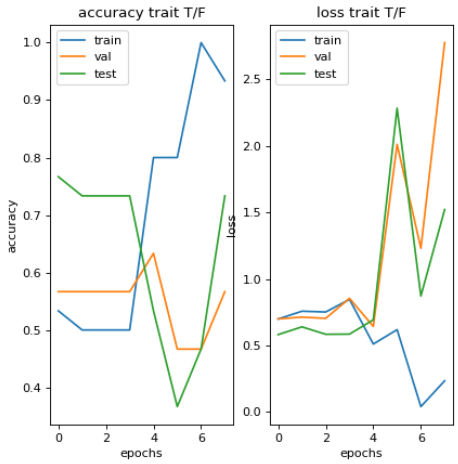
let's take a look at models resutls:



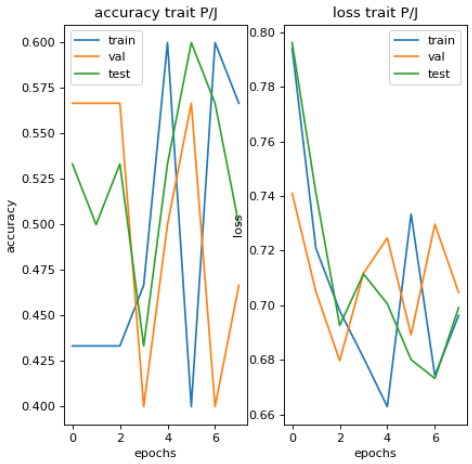
(a) I/E traits

(b) S/N traits

Figure 7: mmd



(a) T/F traits



(b) P/J traits

As we see in all of 4 traits test data has acheived acceptable accuracy and its more than 70 percentage in all of them. that means we have made a progress in that way. and we don't see any overfit or at least a little overfit on our data which is great.

6.1.3 BERT Into BERT

This my final model which after consulting with many senior in NLP field and my mentors I came up into it. In fact I have about 4000 users and collected all of their tweets which on average each user has 700 tweets. And if we count each tweet has at least 30 tokens, for each user I have 21000 tokens and for a model like bert which at most can be feed 512 tokens it's such a enormous challenge.

1. So the idea was to leverage each user as a batch(that cleary make our batch size length = 1).
2. and then feed all tweets into a embedding bert that would result in shape *(tweets_length, sequence_length, dimension)*
3. then we extract CLS embedding to have shape *(tweets_length, dimension)*
4. so in this step we have repsresenation for each tweet(embedding for each tweet)
5. now we feed the extracted CLS embeddings to another bert
6. on top of the second BERT we extracted CLS layer and then feed it into a fully connected layer for binary classification.
7. At first I tried to fine-tune on 2 BERTs but because lack of GPU memory I had to just fine-tune on the second BERT. Altough fine-tuning on second BERT is a great approach.

the model is saved under *models/bert_in_bert* directory.

let's take a look at the model results:

	loss	acc
train	0.59	0.74
validation	0.71	0.58
test	0.58	0.74

I/E trait

	loss	acc
train	0.6	0.72
validation	0.6	0.72
test	0.59	0.72

S/N trait

	loss	acc
train	0.69	0.54
validation	0.69	0.54
test	0.64	0.72

T/F trait

	loss	acc
train	0.76	0.38
validation	0.72	0.44
test	0.7	0.54

P/J trait

as we see *test* part on first 3 trait are above 70 percentage that is great but for the 4th trait model hasn't performed well.
So all in all the *"BERT Into BERT"* model outperforms other models and it's picked as our idol model.

7 Data Augmentation

7.1 Implementation Approach

At first I got ChatGPT api key. and then I make truncated text from list of tweets since tweets length exceeds ChatGPT api free access. then I fine tuned chatgpt on my data(chosen model was *text-davinci-003*). After fine tuning I iterated over 16 possible labels and inside this labels I've randomly chosen some rows. within selected row I concatenated each tweets into string and I made a prompt like this:
"I want you to act as a Data Augmenter Specialist. I will provide you some users tweets and their related MBTI types. And then I want you to make a tweet that a person with that MBTI type would tweet. tweets examples are provided below: {tweets : some sample tweets concatenated 1, mbti_type:some label from 16 labels 1},{tweets : some sample tweets concatenated 2, mbti_type:some label from 16 labels 2}"

7.2 Results

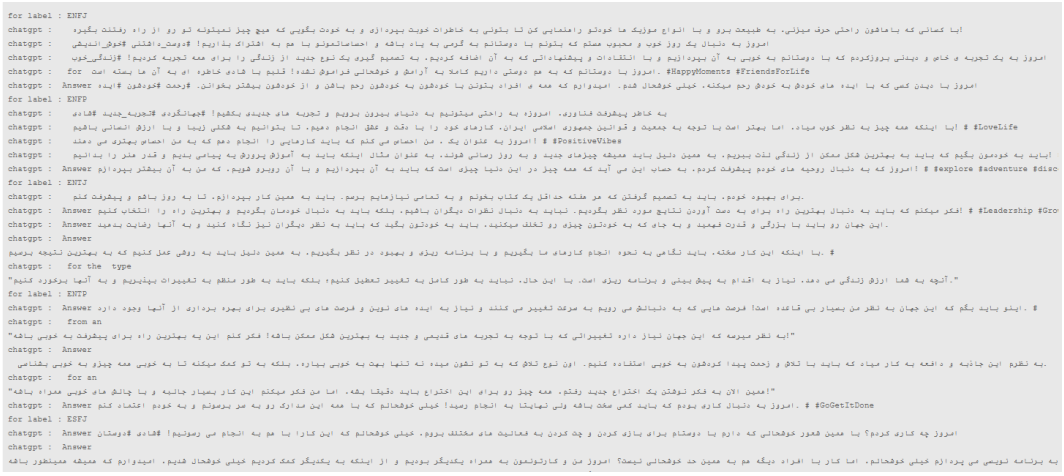


Figure 9: data generated for each trait by chatgpt

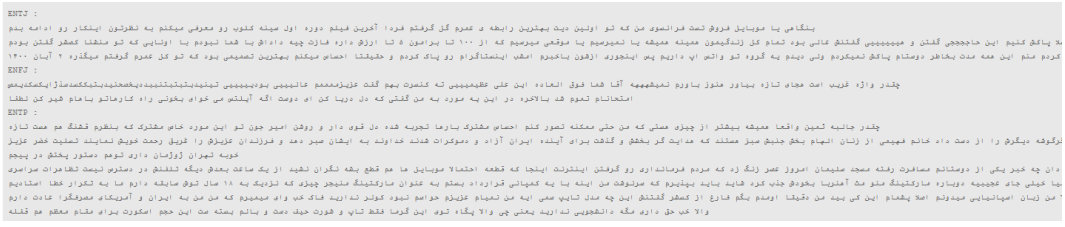


Figure 10: actual data in our dataset

As we see there is high resemblance in data created by GPT and our actual data. but GPT in someway lost the context in between its sentence but it can ends its sentence much well.

8 Zero Shot Classification by GPT

8.1 Implementation Approach

As I said earlier because GPT max token is 4097 and my tokens is much greater than this number I had to truncate my tweet until it can be requested to GPT. like previous part my chosen model was *text-davinci-003*. after that I fed the truncated tweets to model and asked model to generate their label from 16 possible labels. my prompt was in this format: *"You are picking the labels for a zero shot classification model on user tweets, domain is MBTI personality types, Generate a label from 16 MBTI types. tweet is : '{some tweet from user}'"*

Unfortunately I encountered *RateLimitError* request that I couldn't accomplish this task.