ACARIS: Attempting to Improve Conversational AI and Human Social Skills using User Embeddings

Simon Slamka
OngakkenAI

June 21, 2023

Abstract

In this paper, we propose ACARIS, the Advanced Communication Augmentor and Relational Insights System, a system utilizing a novel method to analyze emotional state, intent, and interest of text communication parties. ACARIS is being built with the goal of improving social skills of humans, while also improving the performance of human-facing AI systems. We go over our initial approach, including the initialization of user embeddings from message features, concatenation of user embeddings with word embeddings, modifications of the DistilBERT architecture, and the training and evaluation processes. We also go over the results of our experiments. Presently, we are not able to achieve any statistically nor practically significant improvement over the current state-of-the-art. However, we actively continue to work on improving our approach by iterating over different architectures, hyperparameters, and user embedding forms. In addition, the work currently only handles sentiment analysis, as other tasks are still in heavy development. As such, the target variable of the model is a sentiment label, and as such, the model is a classifier.

1 Introduction

1.1 Keywords

ACARIS, Conversational AI, Social Skills, User Embeddings, BERT, DistilBERT, Sentiment Analysis, Intent Classification, Emotion Recognition, Interest Recognition

1.2 Definitions

- **ACARIS** - the Advanced Communication Augmentor and Relational Insights System

- **NLP** - Natural Language Processing - a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between machines and human language, in particular how to program computers to process and analyze large amounts of natural language data
- **Deep learning** - a class of machine learning algorithms that uses multiple layers to progressively extract higher-level features from the raw input
- **Paired t-test** - a statistical procedure used to determine whether the mean difference between two sets of observations is zero
- **Shapiro-Wilk test** - a test of normality in frequentist statistics
- **p-value** - the probability of obtaining results as extreme as the observed results of a statistical hypothesis test, assuming that the null hypothesis is correct
- **Wilcoxon signed-rank test** - a non-parametric statistical hypothesis test used when comparing two related samples, matched samples, or repeated measurements on a single sample to assess whether their population mean ranks differ
- **Decision boundary** - a hypersurface that partitions the underlying vector space into two sets, one for each class
- **model accuracy score** - the fraction of predictions a model got right
- **model precision score** - the fraction of positive predictions a model got right
- **model recall score** - the fraction of positive cases a model got right
- **model support score** - the number of occurrences of each class in y_true
- **model F1 score** - the harmonic mean of precision and recall
- **ROC curve** - the Receiver Operating Characteristic curve
- **model AUC score** - the area under the ROC curve - a plot of the true positive rate against the false positive rate
- **SVM** - Support Vector Machine - a supervised machine learning model that uses classification algorithms for two-group classification problems
- **DT** - Decision Tree - a decision support tool that uses a tree-like model of decisions and their possible consequences
- **LogReg** - Logistic Regression - a statistical model that in its basic form uses a logistic function to model a binary dependent variable, although many more complex extensions exist

- **RF** - Random Forest - an ensemble learning method for classification, regression and other tasks that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

- **TF-IDF** - Term Frequency - Inverse Document Frequency - a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus, calculated as the product of its term frequency and its inverse document frequency. Has no sense of word order

- **the Transformer** - a deep learning architecture introduced in 2017, used primarily in the field of NLP[1]

- **BERT** - Bidirectional Encoder Representations from Transformers - a transformer-based machine learning model for natural language processing pre-training developed by Google[2]

- 🤗 - Hugging Face - a company that develops and maintains open-source libraries for natural language processing

- **DistilBERT** - DistilBERT is a smaller, faster, cheaper version of BERT developed by 🤗[3]

- **Vector** - a quantity that has both magnitude and direction

- **Vector space** - a collection of vectors, which may be added together and multiplied ("scaled") by numbers, called scalars

- **Embedding space** - a vector space with a coordinate for each word in the vocabulary, such that words that share common contexts in the corpus are located close to one another in the space

- **Word embedding** - A vector representation of a word's meaning

- **User embedding** - A vector representation of a user's personality, emotional state, intent, and interest

- **FFNN** - Feed-Forward Neural Network - a neural network where connections between units do not form a cycle

- **ReLU** - Rectified Linear Unit - an activation function that returns the input value if it is positive, otherwise it returns zero

- **Loss** - a number indicating how bad the model's prediction was on a single example

- **Cross-entropy loss** - negative logarithmic likelihood loss - a loss function used for classification problems

- **Adam** - an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights

1.3 Motivation

I've always had an issue with interpersonal relationships. I could never fully understand the way people work on a social level. I understand the technical and biological fundamentals, and superficially, how the mind works, but not how all this becomes so much more complex when actually talking and dealing with people. Sometimes, I feel that another person and I broadcast on different frequencies, because we are unable to either understand one another or maintain a (romantic) relationship for an extended period of time. I have tried for many years to find a technical solution to this problem, but only after coming to Denmark, I acquired the core knowledge to try and accomplish this ambitious goal by really getting into it and self-studying as much as I could.
I firmly believe that all things in nature are governed by rules and these rules can be observed, measured, and then, based on that data, predicted. This includes human behavior and emotions. We're nothing more than complex electrobiochemical machines driven by electrical impulses and hormones. Apart from that, even if we don't understand our own minds yet, we still have many, many years of knowledge about how human personalities work and how we make decisions based on what happens to or around us. On a fundamental level, this doesn't change. Sure, they say that we're all different. However, the core remains. We all share many attributes that make us human. I believe that we, given enough data, can use these attributes to predict human behavior, reactions, emotional state, and intent.

1.4 Hypotheses

1.4.1 H_1

Given enough conversational data per person, human behavior (personality, emotional state, intent, and interest rate) in text communication can be predicted with a high level of confidence ($> 80\%$) due to the fact that humans are, on a fundamental level, very similar to one another.

Additionally, we posit that person-specific performance improvements can be achieved by individualizing predictions by using a unique vector representation of a person's personality, emotional state, intent, and interest rate, which we call a user embedding. The postulate stems in the core concept of neural networks being universal function approximators. Therefore, in theory, a neural network should be able to learn to associate a user embedding with a person's behavior in text communication and use that information to improve its predictions.

1.4.2 H_0

The performance of a human behavior analysis model trained using a user embedding is not significantly different from the performance of a human behavior analysis model trained purely with input sequences as the model's input.

1.4.3 Statistical significance

To test the statistical significance between the ACARIS and the baseline models, we use paired t-tests or, if the metrics are not normally distributed, Wilcoxon signed-rank tests. A p-value of < 0.05 is considered statistically significant.

1.4.4 Practical significance

To test the practical significance between the ACARIS and the baseline models, we define a decision boundary of practical significance as a model performance improvement of the ACARIS model over the baseline model of $>= 5\%$ across all calculated model performance metrics, simultaneously. Namely, these are accuracy, precision, recall, F1, and AUC.

1.5 Premise

Interpersonal communication has always been an integral part of human lives and is critical from the moment we're born. With the rise of the Internet and, subsequently, social media and other forms of online text communication, the manner in which we talk has changed dramatically. This has led to a drop in social skills in humans, particularly those of the last generation. ACARIS attempts to adjust for this by providing a way to analyze the emotional state, intent, and interest of text communication parties, providing them with a way to improve their social skills, while also improving the performance of conversational AI systems, which are becoming increasingly prevalent in our society, and their ability to understand human emotions, intent, and interest is becoming more and more important, especially in AI systems that directly interact with humans, such as digital assistants, chatbots, and others.

1.6 Literature Review and Related Work

1.6.1 Studied Literature

- **Machine Learning with PyTorch and Scikit-Learn**[4] by Yuxi Liu, Vahid Mirjalili, Sebastian Raschka
- **Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow**[5] by Aurélien Géron
- **Transformers for Natural Language Processing**[6] by Denis Rothman, Antonio Gulli
- **Transformers for Machine Learning: A Deep Dive**[7] by Uday Kamath, Kenneth L. Graham, Wael Emara

1.6.2 Related Work

- **Human-AI Collaboration Enables More Empathic Conversations in Text-based Peer-to-Peer Mental Health Support**[8]

- **Context Matters: Recovering Human Semantic Structure from Machine Learning Analysis of Large-Scale Text Corpora**[9]

1.7 Initial attempts

Our initial attempts of implementing ACARIS included the use of SVMs, DTs, LogReg, and RFs, each paired with TF-IDF in the preprocessing stage. None of these methods have proven to be accurate enough (barely reaching 50% accuracy) for large amounts of complex language, especially casual language that often includes slang, sarcasm, emoji, and similar elements. The primary issue was the lack of these methods' ability to capture semantic and contextual aspects of our dataset. SVMs, DTs, LogReg, and RFs weren't built specifically for sequential data. For sequential data, RNNs and CNNs were classically used. However, due to the long-range dependency problem of RNNs, we made the decision to not use them, as we believed that they would not be able to capture the long-range dependencies in our dataset. Additionally, we weren't able to guarantee that during inference, there would be no long-range dependencies. Another issue was TF-IDF's lack of word order and semantic meaning understanding. Knowing that the Transformer[1] architecture is capable of capturing these aspects due to its positional encodings and self-attention mechanism, we decided to use it as the basis for ACARIS.

1.8 Short intro on Transformers

The Transformer is a deep learning model architecture proposed in 2017 by a team of Google Brain/Google Research researchers. It is mainly used in NLP, but it has also found its way into numerous vision projects (Latent Diffusion[10]/Stable Diffusion/LLaVA[11], to name a few). The architecture aims to replace RNNs by using attention mechanisms (self-attention, in particular) and positional encodings to model long-range dependencies in sequential data.

2 Methodology

In this section, we provide a detailed description of our approach, including the computation of user embeddings, concatenation with word embeddings, modifications to the model architecture, and the training and evaluation processes.

2.1 Dataset

To build our dataset, we collected 65 thousand messages from our Discord server, classified them into 3 classes (pos, neg, neu), and augmented the dataset with the following additional datasets: GoEmotions[12] dataset, which contains 58 thousand messages classified into 27 classes. We squashed the 27 classes into 3 classes (pos, neg, neu); Twitter Sentiment and Emotion Analysis[13] containing 6032 rows of tweets labeled positive, negative, and neutral; Twitter-Sentiment

Data[14] containing 1975 rows of tweets labeled into the same three classes. To minimize error caused by input data, we removed all users with less than 25 messages, URLs, markdown code blocks, Discord mentions and custom emoji, and attachments. We also removed all messages shorter than 25 characters. The resulting dataset was split into 75% training, 17% validation, and 8% test sets, and contains 24769, 5504, and 2753 rows, respectively (filtered). The discrepancy most likely stems in our splitting algorithm's static seed. The dataset is not publicly available at the moment due to the sensitive nature of the data.

2.2 User Embeddings Initialization

For each user u, we loop through all messages m sent by u and extract these features:

- **Mean word count (per all** m**)** - $\overline{w} = \frac{1}{N}\sum_{i=1}^{N} w_i$, where w_i is the word count of i-th message and N is the total number of messages
- **Vocabulary richness (per all** m**)** - $r = \frac{U}{W}$, where U is the number of unique words and W is the total number of words
- **Mean emoji count (per all** m**)** - $\overline{e} = \frac{1}{N}\sum_{i=1}^{N} e_i$, where e_i is the emoji count of i-th message and N is the total number of messages
- **Mean emoticon count (per all** m**)** - $\overline{em} = \frac{1}{N}\sum_{i=1}^{N} em_i$, where e_i is the emoticon count of i-th message and N is the total number of messages
- **Mean punctuation count (per all** m**)** - $\overline{p} = \frac{1}{N}\sum_{i=1}^{N} p_i$, where p_i is the punctuation count of i-th message and N is the total number of messages
- **Mean sentiment score (per all** m**)** - $\overline{s} = \frac{1}{N}\sum_{i=1}^{N} s_i$, where s_i is the sentiment score of i-th message and N is the total number of messages
- ~~Dominant/prevalent topics (per all m) - the most common topics in all messages~~
- ~~Mean response time (per all m~~ - $\overline{r} = \frac{1}{N}\sum_{i=1}^{N} r_i$, where r_i is the response time of i-th message and N is the total number of messages
- **Mean message count (per day)** - $\overline{m} = \frac{1}{D}\sum_{i=1}^{D} m_i$, where m_i is the message count of i-th day and D is the total number of days
- **Mean links (per all** m**)** - $\overline{l} = \frac{1}{N}\sum_{i=1}^{N} l_i$, where l_i is the link count of i-th message and N is the total number of messages
- ~~Mean markdown code snippet sections (per all m) - $\overline{c} = \frac{1}{N}\sum_{i=1}^{N} c_i$, where c_i is the markdown code snippet section count of i-th message and N is the total number of messages~~
- **Mean abbreviations and acronyms (per all** m**)** - $\overline{a} = \frac{1}{N}\sum_{i=1}^{N} a_i$, where a_i is the abbreviation and acronym count of i-th message and N is the total number of messages

- ~~Mean hashtag count (per all m) - $\overline{h} = \frac{1}{N}\sum_{i=1}^{N} h_i$, where h_i is the hashtag count of i-th message and N is the total number of messages~~
- **Mean noun count (per all w in all m)** - $\overline{n} = \frac{1}{W}\sum_{i=1}^{W} n_i$, where n_i is the noun count of i-th word and W is the total number of words
- **Mean verb count (per all w in all m)** - $\overline{v} = \frac{1}{W}\sum_{i=1}^{W} v_i$, where v_i is the verb count of i-th word and W is the total number of words
- **Mean adjective count (per all w in all m)** - $\overline{adj} = \frac{1}{W}\sum_{i=1}^{W} adj_i$, where adj_i is the adjective count of i-th word and W is the total number of words

These features were selected based on what we thought would most accurately represent a user's text communication behavior. We projected that no two users would have the same or closely similar values for all of these features, which would allow us to differentiate between them.
Then, we concatenate all the features into a single vector e_u (of size d_e):

$$\begin{gathered}\texttt{from torch import cat}\\ e_u = \texttt{cat}([\overline{w}, r, \overline{e}, \overline{em}, \overline{p}, \overline{s}, \overline{m}, \overline{l}, \overline{a}, \overline{n}, \overline{v}, \overline{adj}], \text{dim} = -1)\end{gathered} \tag{1}$$

2.3 Concatenation of user and word embeddings

Given a message m with its word embedding e_w (of size d_m) from a BERT-like pre-trained model, which, in our case, was DistilBERT[3], we concatenate the user embedding e_u to the word embedding e_w to create a joint representation e_{joint} (of size $d_e + d_m$):

$$\begin{gathered}\texttt{from torch import cat}\\ e_{\text{joint}} = \texttt{cat}([e_u, e_w], \text{dim} = -1)\end{gathered} \tag{2}$$

2.4 Model architecture modification

To accommodate the concatenated embeddings, we create a custom FFNN, as such:

$$\begin{aligned}&\texttt{from torch.nn import Linear, ReLU, LayerNorm, Dropout}\\ &h_1 = \texttt{Linear}_{d_e,\ \texttt{512}}(e_u)\\ &h_1 = \texttt{ReLU}(h_1)\\ &h_2 = \texttt{Linear}_{\texttt{512},\ d_m}(h_1)\\ &h_2 = \texttt{LayerNorm}_{d_m}(h_2)\\ &h_2 = \texttt{Dropout}_{\texttt{0.1}}(h_2)\\ &h_3 = \texttt{Linear}_{d_m,\ \texttt{512}}(h_2)\\ &h_3 = \texttt{ReLU}(h_3)\\ &h_3 = \texttt{LayerNorm}_{\texttt{512}}(h_3)\\ &h_3 = \texttt{Dropout}_{\texttt{0.1}}(h_3)\\ &e'_u = \texttt{Linear}_{\texttt{512},\ d_m}(h_3)\end{aligned} \tag{3}$$

The FFNN contains 3 layers, each with a ReLU activation, a layer normalization, and dropout for regularization. The first linear (dense) layer is a function $y = xw + b$, where x is a user embedding vector that takes in the d_e-dimensional user embedding and learns to transform it to a dimensionality of 512. After the first linear transformation, ReLU will return 0 if its input is negative, and the value of the input if it's 0 or positive. This operation is applied element-wise and introduces non-linearity to the model. The second linear layer transforms its input from 512 to d_m, which is the dimensionality of our word embedding, and is of value 768. Layer normalization normalizes the features of the output of the linear layer to have a mean of 0 and a standard deviation of 1 to stabilize the learning process. Dropout will randomly set some of the activations to 0 (with a probability of 0.1), which helps with preventing overfitting by forcing the model to learn a more robust representation of the data. The third linear layer transforms its input from d_m back to 512. We, once again, apply a ReLU, layer normalization, and dropout. The final linear layer reduces its input dimensionality to d_m, which is 768. The output of the FFNN is the user embedding e'_u (of size d_m).

The following is the model definition (shared with permission from OngakkenAI):

```
class DistilBertForMulticlassSequenceClassification(
    DistilBertForSequenceClassification):
def __init__(self, config):
        super().__init__(config)
        self.userEmbSize = 12
        self.pre_classifier = nn.Linear(config.dim * 2, config.dim) #
            '2' because of the concat, which doubles the input size to
            that of the pre_classifier
        self.classifier = nn.Linear(config.dim, 3) # '3' for our 3
            classes
        self.userEmb = nn.Sequential(
                nn.Linear(self.userEmbSize, 512),
                nn.ReLU(),
                nn.Linear(512, config.dim), # config.dim.shape = (768,)
                nn.LayerNorm(config.dim),
                nn.Dropout(0.1),
                nn.Linear(config.dim, 512),
                nn.ReLU(),
                nn.LayerNorm(512),
                nn.Dropout(0.1),
                nn.Linear(512, config.dim)
        )

def forward(self, input_ids=None, attention_mask=None, userEmbs=None,
    head_mask=None, inputs_embeds=None, labels=None, output_attentions=
    None, output_hidden_states=None, return_dict=None):
        return_dict = return_dict if return_dict is not None else self.
            config.use_return_dict

        outputs = self.distilbert(input_ids, attention_mask=
            attention_mask, head_mask=head_mask, inputs_embeds=
            inputs_embeds, output_attentions=output_attentions,
            output_hidden_states=output_hidden_states, return_dict=
            return_dict)

        hidden_state = outputs[0]

        userEmbs = self.userEmb(userEmbs)
        userEmbsRep = userEmbs.unsqueeze(1).repeat(1, hidden_state.size
            (1), 1)
        hidden_state = torch.cat((hidden_state, userEmbsRep), dim=-1)
        pooled_output = hidden_state[:, 0]
        pooled_output = self.pre_classifier(pooled_output)
        pooled_output = nn.ReLU()(pooled_output)
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        loss = None
        if labels is not None:
                lossFct = nn.CrossEntropyLoss()
                loss = lossFct(logits.view(-1, self.num_labels), labels
                    .view(-1))

        if not return_dict:
                output = (logits,) + outputs[2:]
                return ((loss,) + output) if loss is not None else
                    output

        return SequenceClassifierOutput(loss=loss, logits=logits,
            hidden_states=outputs.hidden_states, attentions=outputs.
            attentions)
```

The *userEmbsRep* variable is created to associate a user embedding with each token in an input sequence. Initially, the user embedding is a matrix of shape $(batchSize, userEmbSize)$. To associate the user embedding with each token, we need to repeat it along a third dimension, which we need to add, making the user embedding a 3D tensor. We first add a singleton dimension to the position 1 using the unsqueeze() method, changing the shape to $(batchSize, 1, userEmbSize)$, and subsequently, repeat the user embedding along this dimension using the repeat() method, changing the second dimension from 1 to *seqLen* (input sequence length), as such $(batchSize, seqLen, userEmbSize)$. The unsqueeze() method doesn't affect the data inside the tensor, only its shape. This resulting user embedding is then concatenated with the model's hidden state, which represents word embeddings for all tokens in an input sequence, along the last dimension.

2.5 Training with user embeddings

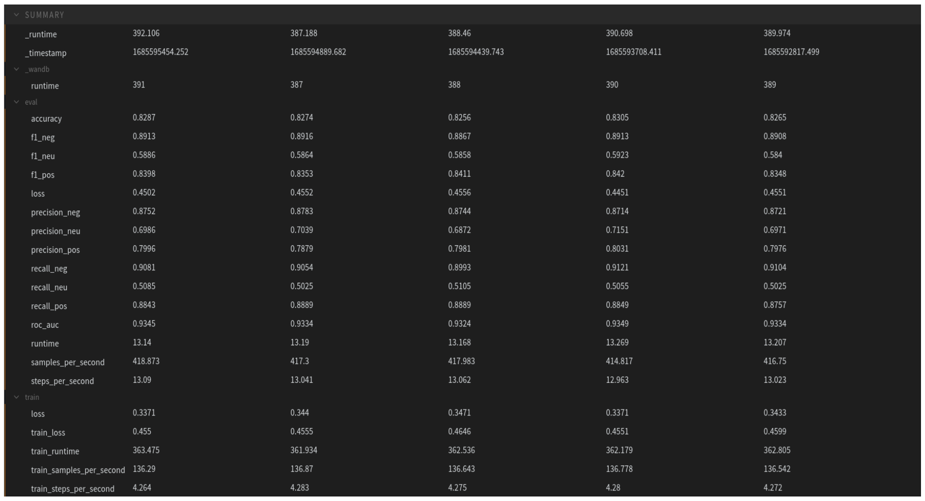
For each training step, we retrieve the underlying DistilBERT model's output at position 0 and store it in $hidden_state$. We then pass the user embedding e_u to the user embedding FFNN, which returns the user embedding's e'_u. Subsequently, we initialize a variable *userEmbsRep*, as explained above. After concatenating the user embeddings e'_u with the hidden state, we supply the resulting $pooled_output$

to the final classifier, returning logits. Having logits, we can calculate loss, which we do using the negative logaritmic likelihood loss function (CrossEntropyLoss). Lastly, we return the loss and logits or a SequenceClassifierOutput object (depending on the value of return_dict).

2.6 Updating user embeddings

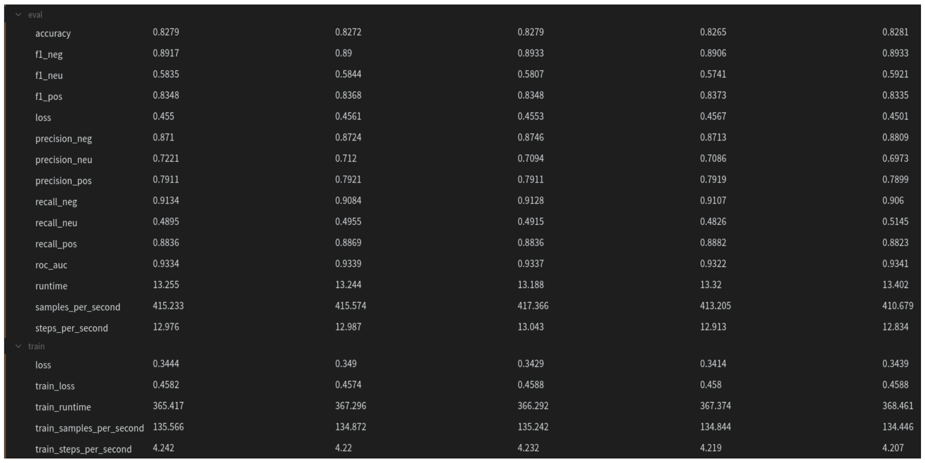
The user embeddings e_u are not updated during training. Instead, they are pre-computed during data preprocessing and stored in a .pt file, one file per user. As such, we can reuse them later, without the need to run feature extraction again. We repeat training every week, using new data from the previous week to update the user embeddings e_u. The reason for this decision is that we want the user embeddings e_u in a static state to prevent any potential issues with losses. We considered updating the user embeddings e_u during training, but we decided against it because we believe that it would be too computationally expensive and would not provide any significant benefits. It could actually worsen the model's performance, as the user embeddings e_u would be constantly changing, which could lead to the model being unable to learn to associate messages m with user embeddings e_u.

3 Evaluation



SUMMARY					
_runtime	392.106	387.188	388.46	390.698	389.974
_timestamp	1685595454.252	1685594889.682	1685594439.743	1685593708.411	1685592817.499
_wandb					
runtime	391	387	388	390	389
eval					
accuracy	0.8287	0.8274	0.8256	0.8305	0.8265
f1_neg	0.8913	0.8916	0.8867	0.8913	0.8908
f1_neu	0.5886	0.5864	0.5858	0.5923	0.584
f1_pos	0.8398	0.8353	0.8411	0.842	0.8348
loss	0.4502	0.4552	0.4556	0.4451	0.4551
precision_neg	0.8752	0.8783	0.8744	0.8714	0.8721
precision_neu	0.6986	0.7039	0.6872	0.7151	0.6971
precision_pos	0.7996	0.7879	0.7981	0.8031	0.7976
recall_neg	0.9081	0.9054	0.8993	0.9121	0.9104
recall_neu	0.5085	0.5025	0.5105	0.5055	0.5025
recall_pos	0.8843	0.8889	0.8889	0.8849	0.8757
roc_auc	0.9345	0.9334	0.9324	0.9349	0.9334
runtime	13.14	13.19	13.168	13.269	13.207
samples_per_second	418.873	417.3	417.983	414.817	416.75
steps_per_second	13.09	13.041	13.062	12.963	13.023
train					
loss	0.3371	0.344	0.3471	0.3371	0.3433
train_loss	0.455	0.4555	0.4646	0.4551	0.4599
train_runtime	363.475	361.934	362.536	362.179	362.805
train_samples_per_second	136.29	136.87	136.643	136.778	136.542
train_steps_per_second	4.264	4.283	4.275	4.28	4.272

Figure 1: Baseline model performance[15]



eval					
accuracy	0.8279	0.8272	0.8279	0.8265	0.8281
f1_neg	0.8917	0.89	0.8933	0.8906	0.8933
f1_neu	0.5835	0.5844	0.5807	0.5741	0.5921
f1_pos	0.8348	0.8368	0.8348	0.8373	0.8335
loss	0.455	0.4561	0.4553	0.4567	0.4501
precision_neg	0.871	0.8724	0.8746	0.8713	0.8809
precision_neu	0.7221	0.712	0.7094	0.7086	0.6973
precision_pos	0.7911	0.7921	0.7911	0.7919	0.7899
recall_neg	0.9134	0.9084	0.9128	0.9107	0.906
recall_neu	0.4895	0.4955	0.4915	0.4826	0.5145
recall_pos	0.8836	0.8869	0.8836	0.8882	0.8823
roc_auc	0.9334	0.9339	0.9337	0.9322	0.9341
runtime	13.255	13.244	13.188	13.32	13.402
samples_per_second	415.233	415.574	417.366	413.205	410.679
steps_per_second	12.976	12.987	13.043	12.913	12.834
train					
loss	0.3444	0.349	0.3429	0.3414	0.3439
train_loss	0.4582	0.4574	0.4588	0.458	0.4588
train_runtime	365.417	367.296	366.292	367.374	368.461
train_samples_per_second	135.566	134.872	135.242	134.844	134.446
train_steps_per_second	4.242	4.22	4.232	4.219	4.207

Figure 2: ACARIS model performance[15]

4 Validation

To validate our approach, we compare the performance of our model to the performance of a baseline model. We use the same model, DistilBERT, but without user embeddings e_u. We fine-tune both models on the same dataset and compare their performance on the same test set.

The base model performed as follows:

```
        Baseline metrics:
              precision    recall  f1-score   support

         neg       0.88      0.90      0.89      1514
         neu       0.63      0.47      0.54       461
         pos       0.81      0.89      0.85       778

    accuracy                           0.83      2753
   macro avg       0.77      0.75      0.76      2753
weighted avg       0.82      0.83      0.82      2753

[[1369   85   60]
 [ 146  217   98]
 [  45   44  689]]
ROC AUC: 0.9355478295192009
Log loss: 0.43026491940343037
```

The ACARIS model performed as follows:

```
        ACARIS metrics:
              precision    recall  f1-score   support

         neg       0.89      0.90      0.89      1514
         neu       0.62      0.50      0.55       461
         pos       0.81      0.88      0.85       778
```

```
    accuracy                           0.83      2753
   macro avg       0.77      0.76      0.77      2753
weighted avg       0.82      0.83      0.82      2753

[[1361   90   63]
 [ 134  232   95]
 [  36   54  688]]
ROC AUC: 0.9343263777788224
Log loss: 0.43922137166238784
```

5 Results

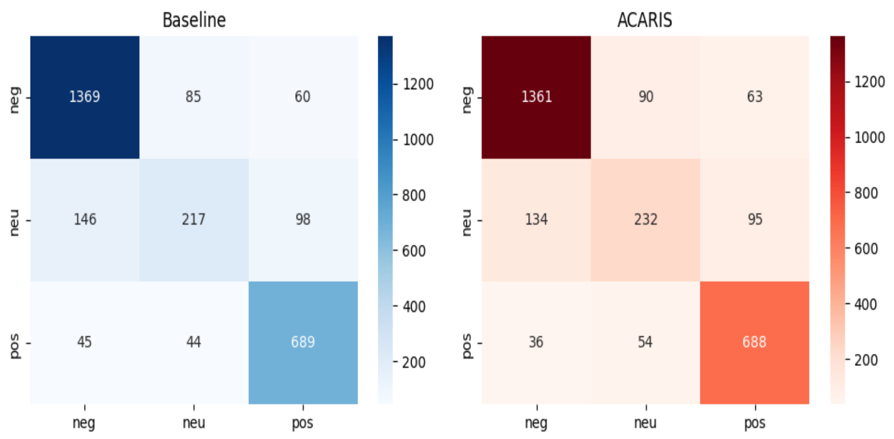


Figure 3: The confusion matrix of both models

Both models performed similarly.

6 Conclusion

Given all the information and performance metrics presently available as a result of the current implementation, we must conclude that we failed to reject the null hypothesis H_0, and as such, in this particular work, rejected the hypothesis H_1. The ACARIS model does not exhibit any statistically nor practically significant improvements over the baseline model.

7 Future work

Generally speaking, we still believe that the first part of the hypothesis H_1 applies, and that individualizing model outputs to each user is the way to go. It's just a matter of data sufficiency and correct approach. We plan to continue experimenting with various pre-trained models, especially those released in the recent weeks, such as the Falcon-40B[16], which performs incredibly well on eval tests, such as the ARC (25-shot)[17], HellaSwag (10-shot)[18], and MMLU (5-shot)[19], and is presently the number one open-source model[20] in terms of its performance. The recent weeks have also seen numerous optimization techniques that enable more efficient training of larger language models, so even with our limited hardware, we are able to fine-tune models of 13B and more parameters. In addition, we also want to rethink our approach to user embeddings, and perhaps learn them during training, if we find a way to do so efficiently. Until then, we'll continue working with pre-computed user embeddings, but will expand them to add more complexity. Another area of improvement, in our opinion, is the concatenation operation. We believe that summation with word embeddings instead of concatenation may offer different results, and will try to implement it in the future. Finally, and this is the most important factor, is the lack of user-specific data. We don't presently have sufficient messages for many users to train a model that generalizes well, most likely due to our aggressive filtering procedure (all users with less than 25 messages and messages of less than 25 characters are discarded). We suspect that this is the main reason why our model failed to outperform the baseline model.

References

[1] Ashish Vaswani et al. *Attention Is All You Need.* 2017. arXiv: `1706.03762 [cs.CL]`.

[2] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: `1810.04805 [cs.CL]`.

[3] Victor Sanh et al. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.* 2020. arXiv: `1910.01108 [cs.CL]`.

[4] Sebastian Raschka et al. *Machine Learning with PyTorch and Scikit-Learn: Develop machine learning and deep learning models with Python.* Packt Publishing Ltd, 2022.

[5] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow.* " O'Reilly Media, Inc.", 2022.

[6] D. Rothman and A. Gulli. *Transformers for Natural Language Processing: Build, train, and fine-tune deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, and GPT-3.* Packt Publishing, 2022. ISBN: 9781803243481. URL: `https://books.google.de/books?id=u9FjEAAAQBAJ`.

[7] U. Kamath, K.L. Graham, and W. Emara. *Transformers for Machine Learning: A Deep Dive.* CRC Press, 2022. ISBN: 9781000587098. URL: `https://books.google.de/books?id=zENpEAAAQBAJ`.

[8] Ashish Sharma et al. *Human-AI Collaboration Enables More Empathic Conversations in Text-based Peer-to-Peer Mental Health Support.* 2022. arXiv: `2203.15144 [cs.CL]`.

[9] Marius Cătălin Iordan et al. *Context Matters: Recovering Human Semantic Structure from Machine Learning Analysis of Large-Scale Text Corpora.* 2022. DOI: `https://doi.org/10.1111/cogs.13085`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1111/cogs.13085`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.13085`.

[10] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* 2021. arXiv: `2112.10752 [cs.CV]`.

[11] Haotian Liu et al. *Visual Instruction Tuning.* 2023. arXiv: `2304.08485 [cs.CV]`.

[12] Dorottya Demszky et al. *GoEmotions: A Dataset of Fine-Grained Emotions.* 2020.

[13] Cyber Cop. *Tweet sentiment and emotion analysis.* Sept. 2021. URL: `https://www.kaggle.com/datasets/subhajournal/tweet-sentiment-and-emotion-analysis?select=all_tweets.csv`.

[14] Saurabh Kumar. *Twitter-sentiment data*. July 2022. URL: `https://www.kaggle.com/datasets/codersaurabh/twittersentiment-data?select=Tweet-Sentimental-Data.csv`.

[15] *Weights and Biases*. URL: `https://wandb.ai/site`.

[16] Ebtesam Almazrouei et al. "Falcon-40B: an open large language model with state-of-the-art performance". In: (2023).

[17] Peter Clark et al. *Think you have Solved Question Answering? Try ARC, the AI2 Reasoning Challenge*. 2018. arXiv: `1803.05457 [cs.AI]`.

[18] Rowan Zellers et al. *HellaSwag: Can a Machine Really Finish Your Sentence?* 2019. arXiv: `1905.07830 [cs.CL]`.

[19] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. 2021. arXiv: `2009.03300 [cs.CY]`.

[20] Edward Beeching et al. *Open LLM Leaderboard*. `https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard`. 2023.