Methods details

Given that the language models have different architectures, they come with different (computation) space and time complexity, separate pipelines were developed:

GloVe: Global Vectors for Word Representation. GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus. [Pennington et al 2014]

- Word level similarity

1. Get pre-trained word vectors: Wikipedia2014 + Gigaword5 [glove.6B.300d.w2v.txt]
2. Embed the verbatim transcripts with the vectors
3. Compute the cosine similarity for the verbatim data frames. Algorithms include K2:10: the word-to-word variability at K inter-word distances, with K ranging from 2 to 10; MV5/10: average semantic similarity of each word in 5- or 10- words window
4. Exclude dysfluencies (partial, punctuation, repetition, unintelligible, noise, single letter, neologism, speech error, filler pauses) from the verbatim transcripts
5. Implement K2:10 and MV5/10 cosine similarity algorithms on the dysfluencies-excluded data frames.
6. Additionally exclude NLTK stop words
7. Implement K2:10 and MV5/10 cosine similarity algorithms on the dysfluencies-stopwords-excluded data frames.
8. Statistical analysis: calculate 5th percentile (Q5), 95th percentile (Q95), and interquartile range (IQR), aggregate over each individual participant
9. Clinical analysis: merge the output data frames with the clinical data frames, including diagnosis group, SSDvHC, demographics, TLC individual items rating, factor scores derived from TLC ratings
10. Clinical analysis: calculate group diagnosis effect sizes, TLC factors correlation coefficients, and correlation p-values

- Utterance level similarity

1. Calculate the verbatim utterance embeddings by averaging the verbatim word embeddings [sentence length > 4], the so-called *mean pooling*
2. Implement the similarity algorithms on the verbatim utterance embeddings. The algorithms include FOC: the first order cosine similarity of consecutive phrase vectors; SOC: second order cosine similarity between phrase separated by another intervening phrase
3. Calculate the dysfluencies-excluded utterance embeddings by averaging the dysfluencies-excluded word embeddings [sentence length > 4]
4. Implement FOC and SOC similarity algorithms on the dysfluencies-excluded utterance embeddings.
5. Calculate the dysfluencies-stopwords-excluded utterance embeddings by averaging the dysfluencies-stopwords-excluded word embeddings [sentence length > 4]
6. Implement FOC and SOC similarity algorithms on the dysfluencies-stopwords-excluded utterance embeddings.

7. Repeat steps 8-10: statistical analysis, clinical analysis

BERT: Bidirectional Encoder Representations from Transformers [Pogiatzis 2019; Devlin et al 2018]

- Word level similarity

1. Compile verbatim *sentence*-level aggregate data frames
2. Extract *verbatim* contextualized word embeddings from uncased BERT base model using Keras and TensorFlow [specifically, the last 4 layers from the 12 hidden encoders], with step 1 output as input representation
3. Compute K2:10 and MV5/10 cosine similarity for the verbatim word embeddings
4. Statistical analysis, clinical analysis
5. Repeat step 1-4 for the dysfluencies-excluded word embeddings
6. Take step 5 dysfluencies-excluded word embeddings output, filter out NLTK stopwords' word embedding vectors
7. Implement the K2:10 and the MV5/10 algorithms on the dysfluencies-stopwords-excluded word embeddings
8. Statistical analysis, clinical analysis

- Utterance level similarity

1. Compile verbatim *sentence* level aggregate data frames [sentence length > 4]
2. Encode sentences to get their embeddings using the stsb-roberta-large model
3. Compute FOC and SOC cosine similarity scores of sentence pairs
4. Statistical analysis, clinical analysis
5. Repeat step 1-4 for FOC/SOC similarity of the dysfluencies-excluded and the dysfluencies-stopwords-excluded data frames

T5: Text-to-text transfer transformer [Patil 2020; Raffel 2019]

- Word level similarity

1. Compile verbatim *sentence* level aggregate data frames [sentence length > 4]
2. Encode sentences to get word embeddings using the TensorFlow framework and the T5small model last_hidden_states function, with step 1 output as input representation. [specifically, initialize the T5Model class and only forward pass through its encoder. The first element of the returned tuple is the final hidden states]
3. Compute MV5/10 cosine similarity of verbatim word embeddings
4. Statistical analysis, clinical analysis
5. Repeat step 1-4 for MV5/10 cosine similarity of dysfluencies-excluded word embeddings and dysfluencies-stopwords-excluded word embeddings
6. Repeat step 1-4 for K2:10 cosine similarity of the verbatim and the dysfluencies-excluded word embeddings
7. Take step 6 dysfluencies-excluded word embeddings data frames, filter out NLTK stop words.
8. Compute K2:10 similarity for the dysfluencies-stopwords-excluded word emebeddings
9. Statistical analysis, clinical analysis

- Utterance level similarity

1. Compile verbatim *sentence* level aggregate data frames [sentence length > 4]
2. Encode sentences to get word embeddings using the PyTorch framework and the T5small model t5_tokenizer.decode function to get FOC and SOC similarity
3. Statistical analysis, clinical analysis
4. Repeat 1-3 for the dysfluencies-excluded FOC/SOC similarity and the dysfluencies-stopwords-excluded FOC/SOC similarity

GPT3: Generative Pre-trained Transformer 3 is an autoregressive language model that uses deep learning to produce human-like text [Brown et al 2020; OpenAI API]

- Word level similarity

1. Compile verbatim *word* level aggregate data frames [***NOTE***: Different from BERT and T5, for which we managed to encode sentence and return one embedding vector per token, GPT3 get_embedding seems to have *mean pooling* as a built-in helper function. It returns a single embedding vector for the whole sentence. Therefore, *word* level data frame is the input for step 2]
2. Get word embeddings from OpenAI API GPT3 using get_embedding function, engine text-similarity-babbage-00 [paid, can only be done locally]
3. Compute K2:10 and MV5/10 similarity for verbatim word data frames
4. Statistical analysis, clinical analysis
5. Repeat step 1-4 for the dysfluencies-excluded word embeddings and dysfluencies-stopwords-excluded word embeddings

- Sentence level similarity

1. Compile verbatim *sentence* level aggregate data frames [sentence length > 4]
2. Encode sentences using OpenAI API GPT3 using get_embedding function, engine text-similarity-babbage-00 [paid, can only be done locally]
3. Compute FOC and SOC similarity for the verbatim sentence data frames
4. Statistical analysis, clinical analysis
5. Repeat steps 1-4 for the dysfluencies-excluded sentence embeddings and dysfluencies-stopwords-excluded sentence embeddings