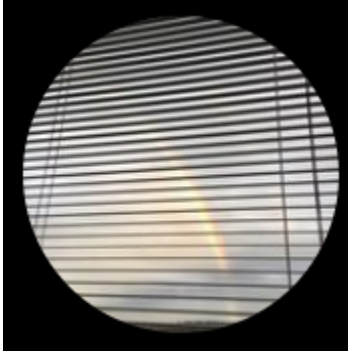



I. Basic information for winner announcement

Please provide your preferred information for use in announcing the winners of the competition.

<p>Name: iwyw (YWT) Hometown: Hong Kong Profile picture:</p> 	<p>Name: Tang, Chung Wai Hometown: Hong Kong Profile picture:</p> 
--	--

II. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

Team name: R4G (Research4Good)

YWT is a computer scientist by training and currently works as a researcher developing methods based on computer vision, machine learning, and deep learning for applications in public health. During free times, she volunteers as a mentor for junior research trainees.

Chung Wai Tang is a graduate student researcher from the University of Nottingham. He is passionate in writing, gerontology and computer applications in healthcare. Outside of his graduate studies, he is a full-time secondary school instructor and leader of the "Business and Economics" department, with over ten years of experience from two different internationalized schools.

2. What motivated you to compete in this challenge?

We are passionate in gerontology and love to contribute in every minor way to advance knowledge that helps promote the health and wellness of older adults. This is our second team competition/ experience. Compared to our previous competition experience in a different platform, the series of challenge deadlines and prizes offered in this competition strongly motivated us to place our first baby steps on the research field of natural language processing, a completely foreign territory of ours. We especially appreciated the thoughtfulness behind planning this competition, such as the live (and recorded) information sessions with the host/ domain-knowledge experts.

3. High level summary of your approach: what did you do and why?

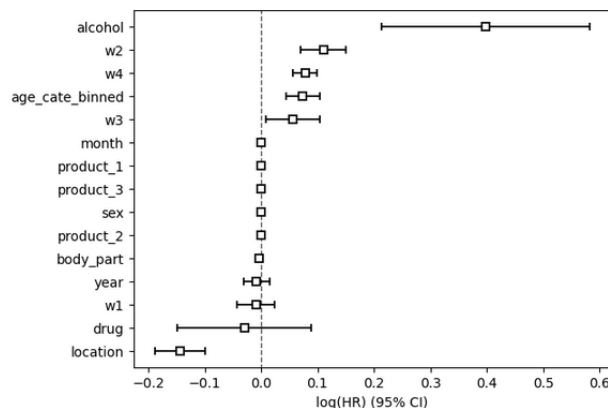
Research statement: We investigated the potentials of various survival model candidates that were trained to map data collected at baseline to patients' future outcomes, as measured using disposition categories, as a surrogate to patients' severity levels. If successful, these models, when deployed on a web application, for instance, will help assess the necessity of hospital visits at baseline.

Details on methods: In representing the textual narratives, we explored various large language models. In creating the outcome definitions, which involved the use of survival objects, i.e. each outcome represented by an event indicator and corresponding event time (e.g. number of hours from fall to hospital visit, or treatment at the hospital), we extracted time to event data from the textual narratives via keyword searches that account for typos. To aid the secondary analysis of seasonal trends in falls that may help the prediction results, we also extracted month and year from the "treatment_date" baseline variable.

To interpret how different language models generate different word embeddings, we used the UMAP algorithm to reduce the dimensionality of each word embedding and then plotted them on two-dimensional plots and coloured each sample based on the value of a categorical demographic variable (e.g. race and sex). In terms of baseline categorical variables to exclude in the prediction task, we dropped "fire involvement" due to its sparsity in order to bypass degenerate mathematical formulation. We also omitted "diagnosis" as they correlated with disposition (likewise, other diagnostic information in the narratives were also stripped out before calculating the word embeddings).

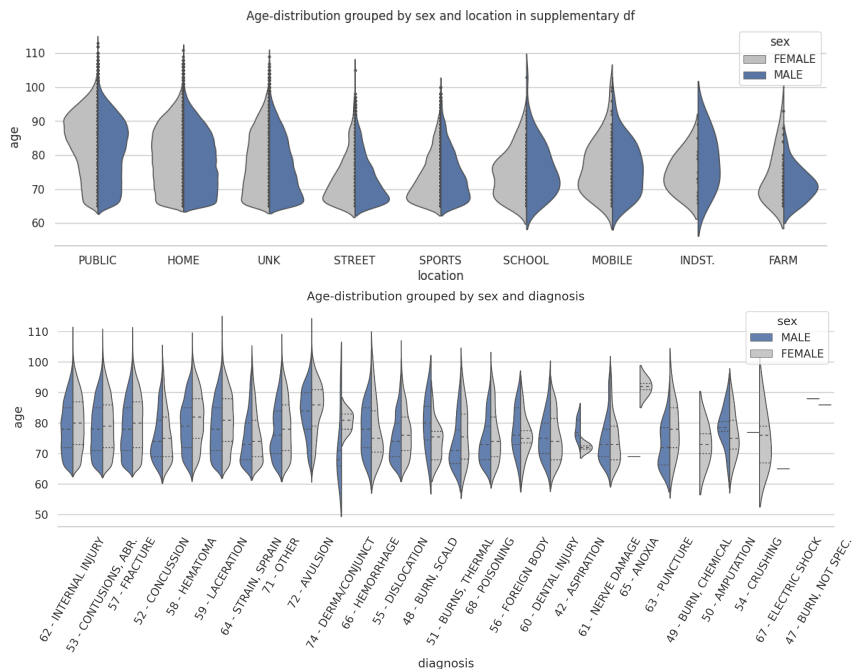
Finally, we fitted different model candidates using different combinations of input data (e.g. using a single set of word embedding alone, using their dimensionality reduced versions, combining all types of word embeddings with select demographic variables).

In summary, we observed that using all *compressed* feature representations have led to the best prognostic accuracies of survival models that were fitted using the eXtreme Gradient Boosting (XGB) models. To compare with a reference approach, we employed Cox's regression, which is a commonly adopted baseline for survival data analysis. The log of the hazard ratios and their confidence intervals are shown in the figure below.

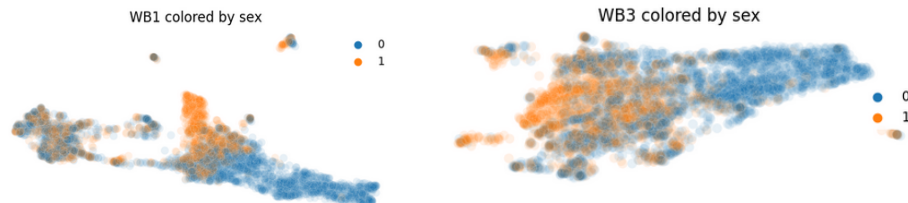


4. Do you have any useful charts, graphs, or visualizations from the process?

During the exploratory data analysis phase, we generated various violin plots to understand the characteristics of the cohort, e.g.:



We also employed dimensionality reduction algorithm (UMAP) to visualize the text embeddings. We were surprised by the clusters generated by two types of embeddings that seem to differentiate sex as shown below.



5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

% =====

Example 1: The code snippet below uses multiprocessing to speed up the extraction of word embeddings.
It also shows different usage of language models that others could expand on.

```
import multiprocessing as mp
emb = 4
TF_MODEL = 0
if emb<4:
    try:
        from sentence_transformers import SentenceTransformer
    except:
```

```

!pip install -U sentence-transformers
from sentence_transformers import SentenceTransformer
if emb==1:# 768
    model = SentenceTransformer('all-mpnet-base-v2')
elif emb==2: # 384
    model = SentenceTransformer('all-MiniLM-L6-v2')
elif emb==3: #768
    model = SentenceTransformer('paraphrase-mpnet-base-v2')
elif emb==4: # 512
    TF_MODEL = 1
    import tensorflow_hub as hub

    # Cer et al. "Universal Sentence Encoder". arXiv:1803.11175, 2018.
    module_url = "https://tfhub.dev/google/universal-sentence-encoder/4"
    model = hub.load(module_url)

elif emb==5:
    # Feng et al. Language-agnostic BERT Sentence Embedding. July 2020
    import tensorflow_hub as hub
    try:
        import tensorflow_text as text
    except:
        !pip install tensorflow_text
        import tensorflow_text as text
    from transformers import pipeline
    url='https://tfhub.dev/google/universal-sentence-encoder-cmlm/multilingual-preprocess/2'
    tokenizer = hub.KerasLayer(url)
    model = hub.KerasLayer('https://tfhub.dev/google/LaBSE/2')

def compute( inp ):
    res= model( tokenizer( inp ) )
    r=normalization( res['default'] )
    return np.array(r)

mp_context = NoDaemonContext()
nprocs = mp.cpu_count()
print( nprocs, 'cores' )
with mp_context.Pool( nprocs ) as pool:
    async_r = [pool.apply_async( compute, ii ).get() for i in tqdm(range(size_n)) for ii in inp[i] ]
    print( end='.' )

elif emb>=6:

    import torch
    from transformers import pipeline
    from torch.multiprocessing import Pool, Process, set_start_method

```

```

from transformers import BertModel, BertTokenizerFast
names={}
names[6] = "setu4993/LEALLA-small"
names[7] = "setu4993/LEALLA-base"
names[8] = "setu4993/LEALLA-large"

def get_pipe():
    tokenizer = BertTokenizerFast.from_pretrained(names[emb])
    model = BertModel.from_pretrained(names[emb])
    model = model.eval()
    return tokenizer, model

def compute( inp ):
    tokenizer, model = get_pipe()
    english_inputs = tokenizer(inp, return_tensors="pt", padding=True)

    with torch.no_grad():
        english_outputs = model(**english_inputs)
        r = np.array( english_outputs.pooler_output )
    return r

multi_pool = Pool(processes=3)
predictions = multi_pool.map( compute, inp )
multi_pool.close()
multi_pool.join()
Embeddings[emb]=np.array(predictions).squeeze()

if emb<5:
    Embeddings[emb] = embed( model, inp, TF_MODEL )

% =====

# Example 2: To optimize the hyperparameters of the XGB model, we employed Bayesian optimization via the
# Optuna package. The code below shows how to set up the objective function that Optuna relies on when
# determining the next set of hyperparameters to set for the XGB candidate model, which can also be replaced with
# other models such as random survival forest.

def tuner(trial):
    params = {'learning_rate': trial.suggest_float('learning_rate', 0.001, 1.0), 'aft_loss_distribution':
    trial.suggest_categorical('aft_loss_distribution', ['normal', 'logistic', 'extreme']),
    'aft_loss_distribution_scale': trial.suggest_float('aft_loss_distribution_scale', 0.1, 10.0), 'max_depth':
    trial.suggest_int('max_depth', 3, 10), 'booster': trial.suggest_categorical('booster', ['gbtree', 'dart', ''],
    'scale_pos_weight': trial.suggest_float('scale_pos_weight', pos_ratio*0.1, 10*pos_ratio ),
    'alpha': trial.suggest_float('alpha', 1e-8, 10 ), # L1 reg on weights
    'lambda': trial.suggest_float('lambda', 1e-8, 10 ), # L2 reg on weights
    'eta': trial.suggest_float('eta', 0, 1.0), # step size

```

```

'sampling_method': trial.suggest_categorical('sampling_method', samp_choices ),
'subsample': trial.suggest_float('subsample', 0.01, 1 ),
'gamma': trial.suggest_float('gamma', 1e-8, 10) # the larger the more conservative; minimum loss
reduction required to get a leaf
    }

```

```

params.update(base_params)
pruning_callback = optuna.integration.XGBoostPruningCallback(trial, 'valid-aft-nloglik')
bst = xgb.train(params, ds['trn1'], num_boost_round=10000,
                evals=[(ds['trn1'], 'train'), (ds['trn2'], 'valid')],
                early_stopping_rounds=50, verbose_eval=False, callbacks=[pruning_callback])
if bst.best_iteration >= 25:
    return bst.best_score
else:
    return np.inf

```

```

# Initialize Optuna (hyperparameter search) study object
study = optuna.create_study(direction='minimize')

```

```

# Begin the search; will take some time
study.optimize( tuner, n_trials= n_trials )
params = {}
params.update(base_params)
params.update(study.best_trial.params)

```

```

# Refit using the best parameters found by Optuna
bst = xgb.train(params, ds['trn1'], num_boost_round=10000,
                verbose_eval=False, evals=[(ds['trn1'], 'train'), (ds['trn2'], 'valid')], early_stopping_rounds=50)

```

```

# Explore hyperparameter search space
fig = optuna.visualization.plot_param_importances(study)
fig.show()

```

```

% =====

```

Example 3: The code below allows us to run Cox's regression analysis to interpret the relative predictive value of each input variable and visualize these values. In particular, the log of hazard ratio of each variable and their 95% confidence intervals are plotted as shown below. Variables whose confidence intervals contain 0 suggest they may be of less relevance for a given outcome.

```

import sksurv
import matplotlib.pyplot as plt
from sksurv.util import Surv
from lifelines import CoxPHFitter

```

X is a dataframe containing all input variables and variables defining the outcome, i.e. "time2hosp" and "severity"

```
penalty = np.ones( X.shape[1]-2 )*.1
```

PH stands for “proportional hazard”, which is an assumption that the Cox fitter relies on

```
cph = CoxPHFitter(penalizer=penalty, l1_ratio=.1)
```

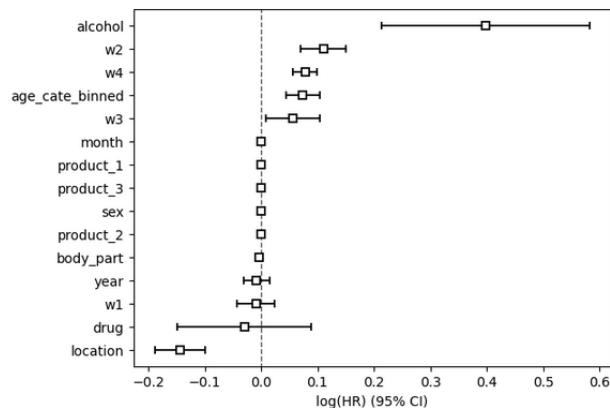
```
cph.fit(X, 'time2hosp', 'severity')
```

```
cph.print_summary()
```

```
cph.plot()
```

% =====

Example visual output from the Cox fitter:



6. Please provide the machine specs and time you used to run your model.

Some of the statistics below were based on:

<https://www.kaggle.com/code/lukicdarkoo/kaggle-machine-specification-cpu-gpu-ram-os>

- CPU (model):
 - o Intel(R) Xeon(R) CPU @ 2.30GHz
 - o CPU MHz: 2300.000
 - o CPU cores: 16
- GPU (model or N/A): GPU P100
- Memory (GB): 118619724 kb; hard drive storage space of 20 Gigabytes
- OS: Unix
- Train duration: >30 hours in total, including calculation of word embedding and model fitting
- Inference duration: roughly 3 minutes when tested in the same environment described above

7. Anything we should watch out for or be aware of in using your notebook (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

- Local Python scripts n*.py are being called in the notebook via the exec() function:


```
exec( open('n1.py','r').read() )
```
- Random seeds are set in an effort to generate reproducible processes, however, the actual reproducibility of the entire pipeline has not been evaluated.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

No

9. How did you evaluate the quality of your analysis and insights, if at all?

We conducted evaluation using unseen samples (from Secondary.csv) and computed evaluation metrics, including concordance index, Brier's score, and Pearson's correlation between the predicted and actual time to events. We compared the performance of different model candidates and input combinations to draw preliminary conclusions.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

Other variants of the LEALLA model (e.g. large) have not been evaluated in the current study. In future work, we would compare the performances more systematically, as well as the hyperparameters of the LEALLA network.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

The processing of the narratives could be optimized. The interpretability of the textual embeddings are crucial for downstream application. Other ideas from the initial (midpoint submission) thought processes that we would like to return to in the future are:

- We also raised various hypotheses during exploratory analysis; we observed various factors common to a fall: patient was multitasking, moving (up or down the stairs), patient fell for reasons beyond their control (response to external factors such as a pet or grandkid running across the street unexpectedly). We ran out of time to continue these explorations but would plan on quantifying the prevalence of these factors.
- Quantification on the effects of polypharmacy
- Quantification on their inability to delegate tasks to other family members (e.g. removing Christmas decorations themselves rather than having a family / friend helping out)

12. What simplifications could be made to run your solution faster without sacrificing performance?

The extraction of word embeddings may be parallelized differently, e.g. extract all embeddings at the same time in one single for-loop. However, the current approach allows us to expand the list of language models to test more easily.