SETTING A GOLD STANDARD FOR THE PREPROCESSING OF STUTTERING DATA

AIMS

The aim of this paper would be produce a gold standard for the features to include in stuttering classification models with a principled and easily interpreted process pipeline. Benchmark this preprocessing method using simple hyper-banded models featuring a varied number of layers, thus mapping the landscape of these models. Subsequent papers are to deal with more complicated and specialised models which we would hope stand a chance of producing field leading performance.

PREPROCESSING

The preprocessing pipeline involved a-priori selection of base speech features which are subsequently transformed by Kalman Filters and Taylor Series expansion followed by concrete auto-encoder filtration to orthogonalise and restrict the dataset. This will maximise the variance accounted for whilst minimising non-orthogonality and the number of features.

A-PRIORI SELECTION OF BASE SPEECH FEATURES

A-priori selection of base speech features was carried out to target features of speech which could be reliably extracted from the time series of recorded audio data. The criterion were used to select these features:

* Previous contribution of these features to machine learning algorithms capable of categorising features of speech.
* Features which capture the phonological characteristic of particular categories of stutters or of stuttering in general, or under-lie these. These features may be products or corollaries of stutters with an established physiological basis.
* A high degree of signal noise separation for the phenomenon as a whole and its constituent parts.
* Appropriate representation of the whole phenomenon – as opposed to a part - as would be necessary to predict all five categories of labelled stutters.

KALMAN FILTERS AND TAYLOR EXPANSION

In lieu of an approximation of a biophysical forward model of speech we constructed Kalman filters and Taylor expansion approximations to capture dynamic qualities of the select speech features and to perform a PCA - allowing separation of constituent factors. The result of the creation of these new features is a proliferation of the number of features and the introduction of a predictable degree of non-linearity which makes selecting between them or transforming them sensible.

CONCRETE AUTO-ENCODER FILTRATION

The purpose of the concrete variational auto-encoder is to select features which i) are optimally orthogonal to each other ii) whilst representing the maximal amount of variance and iii) minimise the number of features so as to reduce parameters and maximise model parsimony.

The point of this section of the pipeline is to first train a decoder network with hyper-banding followed by training and then to substitute this into a decoder-selector complex which reads the number of features from a control file and then to run concrete variational auto-encoder selection of these features. This method of feature selection in parallel will be put to the test by a subsequent phase of modelling. Chiefly this will concern multi-layer perception testing of feature subsets.

*1) Concrete variational decoder hyper-banding*

In this first phase, the decoder of the concrete variational auto-encoder is trained using hyper-banding, an additional soft-max layer is used to train this four layer network in order to produce a score-able output. The numbers of nodes and activation functions are hyper-banded, as are the optimizer float norm and the optimizer clip norm to allow for SGD fitting.

*2) Concrete variational decoder parametrise*

The concrete variational auto-encoder hyper-banding .phase will lead to the adoption of optimal parameters for training in a subsequent phase of training. In this phase of training

the hyper-banded network will be trained with a soft-max layer as before.

*3) Concrete variational selector control file*

The concrete variational auto-encoder selector control file will be generated which specifies the number of features to extract. This will merely generate a list of numbers allowing parallel feature extraction of the number of features that has been specified

*4) Concrete variational selector*

The concrete variational auto-encoder needs to be assembled. The soft-max layer will be stripped out and the decoder will be entered into a decoder-selector complex. The pre-trained decoder-encoder will then be used to extract a set number of features determined by the concrete auto-encoder control file. Parallel processing will then be used to extract these features. The concrete variational auto-encoder complete with model weights will then be saved. Importantly the features extracted are recorded and saved.

With the recorded features extracted for each concrete variational selector and saved in a csv file we will be able to build up a profile of the most salient features to enter into a phase of training. It will be informative how restricting the number of features to extract may impact the specific features selected.

VALIDATION OF FEATURES SELECTED BY CONCRETE AUTO-ENCODER COMPLEX

In order to validate the features selected by our concrete auto-encoder complex, we constructed multi-layer perceptrons that were hyper-banded and then trained/tested our algorithms across the dataset to moderate convergence. We therefore mapped the landscape of accuracy delivered by these algorithms as well as parameter weighted indices like BIC and AIC. This was done to benchmark the feature selection process. We would hope to validate the basic pipeline and feature selection process in order to produce models which score highly. We anticipate that these models are competitive accounts of the data, although SeqGAN and SeqVAE models which have proven so effective in speech and music, can be expected to account for more variance. We would hope that these specialised models, which will be covered in a separate paper, which deals with recurrent processing will lead the field.

FEATURE EXTRACTION BY AN AUTOMATED SCRIPT

Existing toolboxes will account for stock feature processing and extraction as well as the transformation of these features into the specific Taylor series and Kalman Filter variations required where applicable.

CONVERSION OF SCRIPTS INTO PYTORCH

It might be necessary to clean up our code and retain cutting edge validity by converting it from the tensorflow package that it is currently written in into pytorch, which is emerging as the leading package in industry and in academia and ensuring is written to a high standard.

TIME FRAME FOR PREPARATION:

*Week 1: May 31st to June 7th*

Feature selection with various numbers of features specified between 5 and 40.

*Weeks 2-5: June 7th to July 5th*  
Testing of features selected using multiple configurations of dense layer networks arrived at through hyper-banding. Speech scientists to analyse and interpret this data as it is produced, so that all aspects of it can be considered in a timely manner.

*Weeks 5+: July 5th*

Whilst paper is being prepared over a month or so, we will need to convert the code involved in the production of this paper into the emergent commercial and academic standard which is Pytorch and clean it up to academic standards as well as to get the feature selection script up and running. I’d also like to ensure that C++ support is available. I would hope that this process would be completed in a month.

Realistically, I would hope that we can submit this paper within three months time, and that it will achieve moderate impact by setting a gold standard for preparation of stuttering datasets for analysis which has been appropriately benchmarked using standard models which are easily interpretable. Subsequent papers will deal with more specialised models and will introduce recurrent process.

Setting a new gold-standard for stuttering classification features in machine learning based on concrete auto-encoder competition following theory independent forward modelling

**Abstract:**

In this paper we first described audio signal containing fluent speech and stutters in terms of phonological features that account for various aspects of speech production. A high degree of redundancy and theory independence in this description was apparent across the 32 features selected. We selected, as a basis, common features which we believed could help describe and classify stutters. Subsequently, we ran these features through a series of Kalman Filters and Taylor series approximations. For the Kalman Filters gain was varied between one and four time-steps in order to create a high pass filter across frequency. By way of analogy, Taylor series approximations of varying degrees (one to four) were used to separate out aspects of the signal by attending to lower to higher frequency windowed signal. Since Kalman filters hold memory of previous time-steps and Taylor series approximation account for prior and subsequent time-steps in calculating present values these algorithms hold implicit forward models that are theory independent. This stage of preprocessing resulted in proliferation of multi-collinear variables capturing dynamic aspects of the original variables from which they are derived. In the third and final stage of feature design and selection we allowed a concrete auto-encoder complex to select orthogonal subsets of features. Verification of this methodology was achieved by logistic regression and multi-layer perceptron modelling used to plot the landscape of features to be best included (whether base functions or their derivatives) to available maximise orthogonal variance in the dataset whilst minimising the total number of features. The theory-independent forward modelling and broad selection of basis features lends the profile of features selected to interpretation based on theory. [Report models: Our peak performing model computed across [x,y,z datasets] produced accuracy etc. …] A subsequent paper will report recurrent models starting with simple multi-layer-perceptrons prior to current bleeding-edge SeqVAE and SeqGAN designs. For convenience, script is included to preprocess audio signal consistent with our standard for features.

**Methodology**

We used a three step methodology to allow for competition between features.

First we specified basis features sufficient in breadth to describe stuttering classification. Second in lieu of a forward model we allowed these features to be sampled by Kalman filters and Taylor series approximations. Third we allowed competition between features to both maximise orthogonality and variance explained using a sophisticated concrete auto-encoder complex. This was a variational auto-encoder whose decoder had been pre-trained across the over-specified set of basis and transformed variables. Subsequently we tested these data-sets using simple and easy to understand modelling in which logistic regression and two to four layer neural networks were used with hyper-banding.

1) Over-specification of problem in terms of basis features

We arrived at our 32 literature informed basis features as a result of our groups extensive experience in stuttering (from both machine-learning and clinical standpoints). We can be optimistic that these features should contain information sufficient to specify stutters in a phonological sense. This is the case given that we selected common features tested across various permutations in the machine learning for stuttering classification literature, aimed for completeness in the variety of features selected, and took a theory independent and over-arching account of features. We also prioritised features likely to be cardinal in describing particular categories of stutter relative to each other and used the five labelled stutter-types available in the UCLASS transcripts. A high degree of redundancy in stuttering description was likely built in due to the inclusion of these various overlapping theoretical account. In addition to this, indexing variables were also used to keep track of speaker and position in speaker utterance.

2) Forward modelling using Kalman filters and Taylor series approximations

We borrowed Kalman filters from engineering and Taylor series from physics instead of relying on any particular forward model designed to mimic speech production. The resultant transforms of our base variables capture aspects of dynamics and split these into their constituent parts by filtering these for frequency. A result of this allows for combined formulations of features that increase maximised orthogonal variance. Essentially forward modelling increases flexibility going into the concrete auto-encoder competition phase.

3) Concrete auto-encoder competition to select features to preserve with a pre-trained decoder layer

Our decoder was pre-trained with an additional soft-max layer that was stripped off prior to joining this decoder with a concrete layer encoder to form the variational auto-encoder complex. A simulated annealing process meant that combinations of a requested number of features could then be dropped out, feature-by-feature, over time. This process was run in parallel for a range of numbers of features to select allowing us to build up a landscape of selected features.

Testing the various data-sets resulting from each number of features to extract was performed by training hyper-banded 2, 3 and 4 layer neural networks and logistic regression algorithms following hyper-banding. In addition to this information, theoretical approaches were used to estimate the information present in these feature sets and its orthogonality. In order to estimate data we took random samples of the dataset based on those features included and tested these for their mutual information against each other to build an average of the data contained in the abridged dataset. Likewise dot products were computed within batches in order to assess orthogonality. This latter method is a simple mathematical way of checking for orthogonality.

**Folders to include from the github**

Preprocessing phase one - Includes Kalman filter and Taylor series transforms

Preprocessing phase two – Includes variational concrete auto-encoder scripts

Summary statistics – Testing of orthogonality and mutual information (to assess captured variance)

Verification and benchmarking of preprocessing – Modelling logistic regressions and multi-layer perceptron approaches

Drafting of documents folder – Work on the paper in progress

Preprocessing scripts – A variety of scripts sufficient to produce the winning combination of features from raw audio signal for the convenience of researchers attempting to recapitulate and test our gold standard

*Importantly it seems that Tensorflow and emergent PyTorch approaches to this software should be made available.*

**SUMMARY OF REQUIRED PIPELINE PACKAGES**

The following packages need to be installed in a condas virtual environment.

**PYTHON ENVIRONMENT:**

python==3.8.0

**LIST OF PERTINENT PACKAGES:**

tensorflow==2.11.0

keras==2.11.0

pandas==1.5.3

kerastuner-tensorboard-logger==0.2.3

Additional documentation is available at my personal website, which links to my personal github, <https://www.mark-alfred-griffiths.tech/>.

**Gold-standard stuttering classification feature-set selection**

This paper is the second piece of original research in the series of machine learning papers on stuttering classification published by our lab. The key objectives of this paper are:

• To advance preprocessing through the introduction of a concrete auto-encoder designed to`` maximise explained orthogonal variance.

• To create competing baseline models using hyper-banding to search for hyper-parameters. We will construct logistic regression, quasi-SVM, random-forest layer neural network and MLP models. A broad metric of comparative statistics including VC-shatter, AIC and BIC will be created. As such, this paper follows preliminary baseline work and proof of concept. We aim to provide a logistic regression baseline with more complex models showing enhanced accuracy and parameter weighted efficiency.

Preprocessing beyond initial speech science feature extraction had three objectives. Firstly, we sought to capture dynamic elements of the waveform. This was achieved by transforming the original speech features through ordinary difference equation Taylor series and Kalman filter abstractions in lieu of a biomechanical forward model of speech production. Secondly, we sought to select a subset of features that were orthogonalised with respect to each other. Thirdly, we aimed to maximise variance explained by a restricted number of features to promote high parameter weighted information criteria. The last two objectives were upheld using a concrete autoencoder to select twenty five features. The use of a concrete layer was arrived at as a way to select features commanding the greatest proportion of variance with respect to the dependent variable and orthogonal to each other.

Taylor series approximations were computed using the ordinary difference method with degrees from first order to fifth order inclusive across windowed time acquisitions. Higher order Taylor series approximations capture higher frequency information meaning that equation degree acts as a low pass filter which extends to block out successively higher frequencies over the range of its 1st to 5th order implementation. Kalman filter approximations modeling signal as a first order process and jointly tracking means and covariances across various time-step differences allow signal to be predicted with a resolution that decreases with time-step difference value. Higher time-step difference interval values cut out higher frequencies extending low-pass filter range.

We also looked at a Fourier transform of frequencies meaning that order of Taylor series and Kalman filter pitches in a non-linear way that sensitive to differences in successive formant frequency amplitudes. When low-pass filtration is selected the proximity of successive formant frequencies becomes more or less influential in determining their influence on each other. The fine detail on the pitch map is cut out by low order Taylor series approximations of the formant and (coarse grain, high time-step) Kalman filters.

The information available to the Kalman filter is only that from previous time-steps whilst the Taylor series approximation captures information from across the entire window – looking forward and backward. Both of these conditions are suitable to capture aspects of a forward model of speech production, and their contradistinction is useful when allowing factor competition in the concrete auto-encoder phase.

The feature selector variant of the variational-autoencoder known as the concrete auto-encoder was developed to combine features in the latent space and test reconstructions of them in combination allowing for permutation testing of combinations of features which maximized accounted for variance. This was carried out in a manner that started by selecting the single feature that accounted for the most variance, before selecting subsequent features one at a time in combination with the body of features selected prior to this. Therefore the second feature selected by the variational concrete auto-encoder may not be the feature that would account for the second most variance if a single feature were selected in isolation but will be the feature that accounts for the most variance orthogonal to the first feature. This process allows for conservation of the calculations performed in prior phases of the feature-set selection process through weights being updated. The variational capacity of the latent space to compress a variety of numbers of features in combination allows for training of an increasing number of features whilst upholding the encoder complexes selection of this number of features in a simulated annealing permutation context. The decoder weights are then trained in order to replicate the operations in this layer by a flexible process that integrates training across feature-sets of all numbers of features to select.

It is of note that in our concrete auto-encoder the decoding complex is per-trained across the entire dataset of basis speech features and their Kalman Filter and Taylor series transforms using a flexible hyper-banding process. [I will update this with more concrete auto-encoder details and justification of this method.]

To restate the problem at hand, preprocessing in this study sought to i) describe the data in terms of salient speech criteria influential in previous language studies, ii) provide expanded abstractions of these speech processing features amenable to processing of dynamic data, iii) select orthogonalised features to maximise explained variance from within this expanded feature-set using a concrete auto-encoder.

In order to validate our preprocessing methodology we created two classes of models – i) those that hinge on logistic regression and ii) MLP r neural networks. For each approach we first used batched hyper-banding to converge on an optimised set of hyperparameters. Subsequently, we created a model based on these cost-function-minimising parameters and trained it over the entire data-set. Each model class is treated separately below.

For logistic regression a custom layer was created. In order to prevent hyperbanding from resulting in redundancies and multiple compilation errors we then split hyperbanding into l1 and l2 kernel and SGD, Adam and RMSProp factorial and ran these as separate hyperbanding sessions. We read out the top model from each of these and allowed these to compete.

For our neural networks we used hyperband training over a single network and alternated the final layer between a softmax for the basis MLP, a random fourier features layer hinged loss for the (quasi-SVM) NN-SVM, and a random-forest layer for the NN-RF implementation. The key thing here is that the initial hyperbanding work that we carried out for the basis MLP is used to likewise inform the NN-SVM and NN-RF.

The parameters that we iterated over are listed below:

Logistic regression:

*Compiler:*

Compiler:

- Adam

- SGD

- RMSProp

*Kernel:*

Glorot uniform initializer

Kernel regularizer:

- Kernel regularizer L1 value [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

- Kernel regularizer L2 value [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

*Bias:*

Bias initializer set to zeros

Bias regularizer

-L1

-L2

Bias regularizer value [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

*Activity:*

Activation: Sigmoid

Activation Regularizer:

- L1

- L2

Activity Regularizer Value [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

*If Adam:*

*-* learning rate [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

- beta 1 [min\_value=0.1, max\_value=0.9, step=0.1]

- beta 2 [min\_value=0.001, max\_value=0.999, step=0.001]

- beta 2 [min\_value=0.1, max\_value=0.9, step=0.1]

- epsilon [1e-05, 1e-06, 1e-07]

- amsgrad [True, False]

*If SGD:*

*-* learning rate [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

- momentum [min\_value=0.1, max\_value=0.9, step=0.1]

- nesterov [True, False]

*If RMSProp:*

*-* learning rate [1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

*-* rho [min\_value=0.1, max\_value=0.9, step=0.1]

- momentum [min\_value=0.1, max\_value=0.9, step=0.1)]

- centred [True, False]

*Neural Network approaches:*

*Pure-MLP with softmax*

ni\_neurons [min\_value=32, max\_value=128, step=32]  
nj\_neurons [min\_value=32, max\_value=128, step=32]  
nk\_neurons [min\_value=32, max\_value=128, step=32]  
nl\_neurons [min\_value=32, max\_value=128, step=32]  
  
ni\_neurons\_activation ['relu', 'tanh']  
nj\_neurons\_activation ['relu', 'tanh']  
nk\_neurons\_activation ['relu', 'tanh']  
nl\_neurons\_activation ['relu', 'tanh']

ni\_neurons\_dropout\_bool [True, False]  
nj\_neurons\_dropout\_bool [True, False]  
nk\_neurons\_dropout\_bool [True, False]  
nl\_neurons\_dropout\_bool [True, False]

ni\_neurons\_dropout\_value [min\_value=0.1, max\_value=0.8, step=0.1]  
nj\_neurons\_dropout\_value [min\_value=0.1, max\_value=0.8, step=0.1]

nk\_neurons\_dropout\_value [min\_value=0.1, max\_value=0.8, step=0.1]

nl\_neurons\_dropout\_value [min\_value=0.1, max\_value=0.8, step=0.1]

number\_of\_layers [3, 4]

ni\_neurons\_batch\_normalisation\_momentum [min\_value=0.1, max\_value=0.9, step=0.1]  
ni\_neurons\_batch\_normalisation\_epsilon [min\_value=0.1, max\_value=0.9, step=0.1] ni\_neurons\_batch\_normalisation\_epsilon [1e-3, 1e-4]

nj\_neurons\_batch\_normalisation\_momentum [min\_value=0.1, max\_value=0.9, step=0.1]  
nj\_neurons\_batch\_normalisation\_epsilon [min\_value=0.1, max\_value=0.9, step=0.1] nj\_neurons\_batch\_normalisation\_epsilon [1e-3, 1e-4]

nk\_neurons\_batch\_normalisation\_momentum [min\_value=0.1, max\_value=0.9, step=0.1]  
nk\_neurons\_batch\_normalisation\_epsilon [min\_value=0.1, max\_value=0.9, step=0.1] nk\_neurons\_batch\_normalisation\_epsilon [1e-3, 1e-4]

nl\_neurons\_batch\_normalisation\_momentum [min\_value=0.1, max\_value=0.9, step=0.1]  
nl\_neurons\_batch\_normalisation\_epsilon [min\_value=0.1, max\_value=0.9, step=0.1] nl\_neurons\_batch\_normalisation\_epsilon [1e-3, 1e-4]

optimizer\_momentum\_float\_value [min\_value=0.1, max\_value=0.9, step=0.1]  
optimizer\_clipnorm\_float\_value [min\_value=0.1, max\_value=1.0, step=0.1]

*MLP pretrain*

model.add(Dense(init\_sets.num\_cats, kernel\_initializer=glorot\_uniform\_initializer(), name='output\_layer'))

model.add(Activation('sofmax'))  
  
model.compile(optimizer=SGD(momentum=optimizer\_momentum\_float\_value,  
clipnorm=optimizer\_clipnorm\_float\_value), loss='categorical\_crossentropy', metrics=['accuracy'])

*Random-Forest*

*better\_default@v1:*

model.add(Dense(data.num\_cats,kernel\_initializer=glorot\_uniform\_initializer(), bias\_initializer='zeros'))

model.add(Activation('softmax'))

model.compile(optimizer=SGD(momentum=best\_hps.get('optimizer\_momentum\_float\_value'),clipnorm=best\_hps.get('optimizer\_clipnorm\_float\_value')), loss="categorical\_crossentropy",,metrics=["accuracy"])

model\_better\_default = tfdf.keras.RandomForestModel(preprocessing=nn\_headless\_model,hyperparameter\_template='better\_default@v1')

model.compile(optimizer=SGD(momentum=optimizer\_momentum\_float\_value,  
clipnorm=optimizer\_clipnorm\_float\_value), loss='categorical\_crossentropy', metrics=['accuracy'])

*Random-Forest*

*benchmark\_rank1@v1:*

model.add(Dense(data.num\_cats,kernel\_initializer=glorot\_uniform\_initializer(), bias\_initializer='zeros'))

model.add(Activation('softmax'))

model.compile(optimizer=SGD(momentum=best\_hps.get('optimizer\_momentum\_float\_value'),clipnorm=best\_hps.get('optimizer\_clipnorm\_float\_value')), loss="categorical\_crossentropy",,metrics=["accuracy"])

model\_better\_default = tfdf.keras.RandomForestModel(preprocessing=nn\_headless\_model,  
hyperparameter\_template='better\_default@v1’

*[Note that there are two random-forest implementations with the layer specifications defined by pre-sets that are overall winners in dealing with data.]*

*SVM*

# ADD RANDOM FOURIER FEATURES LAYER

model.add(random\_fourier\_features\_layer(params['output\_dim'], params['scale'], params['kernel\_initializer'], name='quasi\_svm\_layer'))

# ADD OUTPUT

model.add(Dense(units=data.num\_cats, name ='output\_layer')

# USE HINGE LOSS IN COMPILE

model.compile(optimizer=SGD(momentum=params['optimizer\_momentum\_float\_value'], clipnorm=params['optimizer\_clipnorm\_float\_value']), Loss=keras.losses.hinge, metrics=[keras.metrics.CategoricalAccuracy(name="accuracy")])

*[As subsequent phase of hyper-banding will be necessary following the NN-basis hyperbanding to arrive at the parameters above. This should be relatively trivial to implement, but it is necessary to note that hyperbanding for NN-SVM will be a two-stage process.]*