Definition

Domain Background

Epilepsy is a chronic neurological condition that causes recurring, unprovoked seizures. Seizures can be extremely dangerous, and early warnings can mean the difference between life or death. Being able to detect electrical activity that corresponds to an oncoming seizure, before symptoms begin, would allow for preparations ahead of time.

Similar methodologies (RNN Classifiers on EEG Data):
https://techxplore.com/news/2019-12-deep-eeg-based-emotion-recognition.html
https://towardsdatascience.com/sleep-stage-classification-from-single-channel-eeg-using-convolutional-neural-networks-5c710d92d38e
https://medium.com/@justlv/using-ai-to-read-your-thoughts-with-keras-and-an-eeg-sensor-167ace32e84a
https://github.com/topics/eeg-classification

Specifically for Epilepsy:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6004942/

Datasets and Inputs

I will be using the Epileptic Seizure Recognition Data Set from here: https://archive.ics.uci.edu/ml/datasets/Epileptic+Seizure+Recognition The data was acquired from 500 23-second scans. This was then broken up by other researchers into 1-second chunks (consisting of 178 data points) each. Some of these contain seizure activity, and some do not. Only about 1/5th of the readings contain a seizure, so I chose a metric (F2) that will balance Precision & Recall.

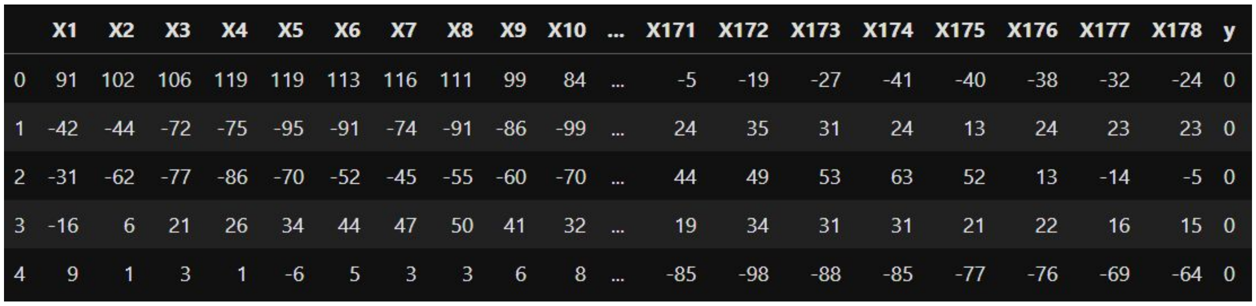
In addition, F2 is also desirable because for medical things, in general we'd rather have a False Positive than a False Negative and so it makes more sense to use an F-beta metric with a beta greater than 1 (the higher the number, the greater weight is placed on Recall).

Problem Statement

Electrical activity in the brain is highly nonlinear, and so is an excellent candidate for analysis via Deep Learning. I'll be attempting to categorize whether a stretch of electrical activity (recorded via EEG) corresponds to a seizure. I'll also try to use methods for analyzing panel data after generating some summary features.

Analysis

Each row is a time series of 178 points, with a class of Seizure or Non-seizure. Some sample rows:

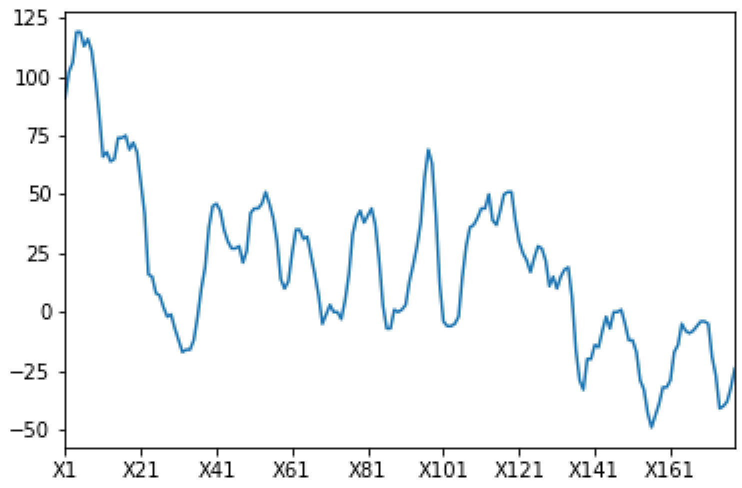


	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	...	X171	X172	X173	X174	X175	X176	X177	X178	y
0	91	102	106	119	119	113	116	111	99	84	...	-5	-19	-27	-41	-40	-38	-32	-24	0
1	-42	-44	-72	-75	-95	-91	-74	-91	-86	-99	...	24	35	31	24	13	24	23	23	0
2	-31	-62	-77	-86	-70	-52	-45	-55	-60	-70	...	44	49	53	63	52	13	-14	-5	0
3	-16	6	21	26	34	44	47	50	41	32	...	19	34	31	31	21	22	16	15	0
4	9	1	3	1	-6	5	3	3	6	8	...	-85	-98	-88	-85	-77	-76	-69	-64	0

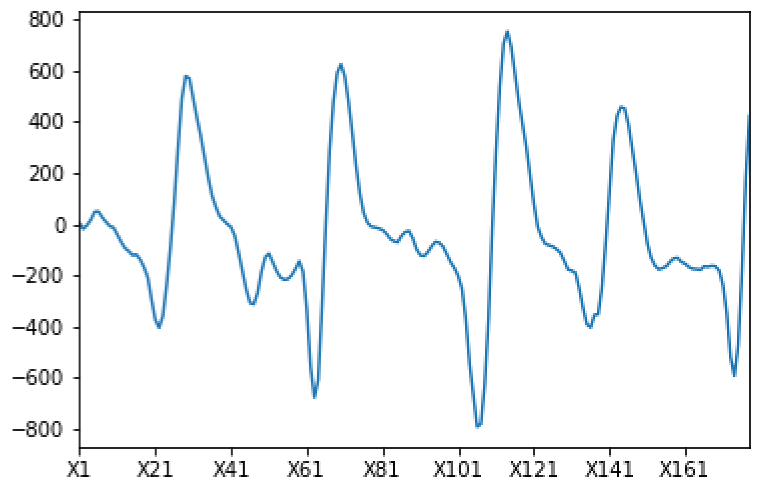
I saw from literature review that seizure activity is generally characterized by a greater amplitude, greater weight in the high-frequency range, as well as a loss of overall complexity. Converting a time series to the frequency domain is fairly straightforward, but complexity is non-trivial - I explored several feature extraction methods to look at signal complexity.

Visually, this is actually relatively easy to inspect. In seizure trials, peaks tend to be higher, and there is a noticeably more-regular structure to the overall time series.

Example of a non-seizure measurement



Example of a seizure measurement



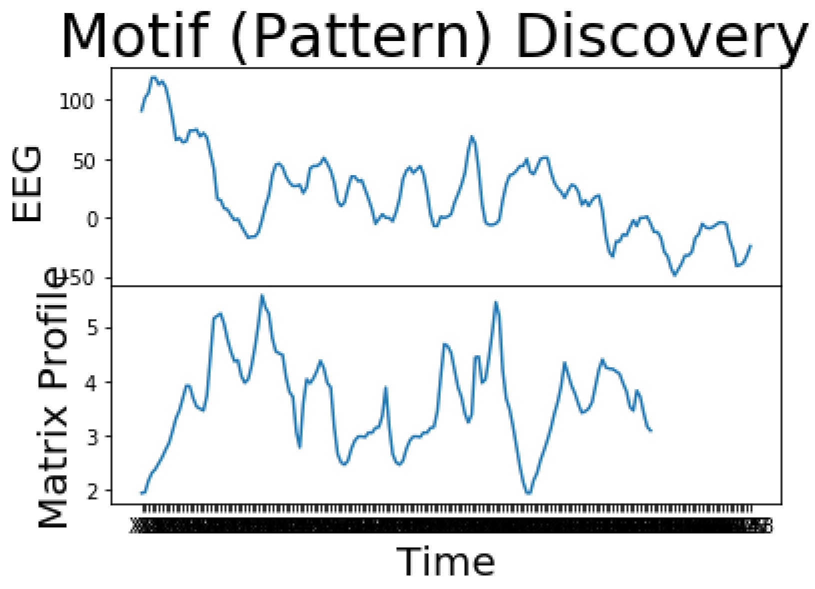
(all units are micro Volts)

One method I used was the tsfresh package (from here: https://tsfresh.readthedocs.io/en/latest/text/introduction.html). It calculates hundreds of different summary statistics on a set of time series - including things like frequency, different variations of autocorrelation, and traditional descriptive statistics. I then trained a simple XGBoost model on these features, and looked at the SHAP values for feature importances (https://github.com/slundberg/shap). The most influential ones were Standard Deviation, auto-correlation at a few different lags, and another that compared the Standard Error from a calculated linear trend.

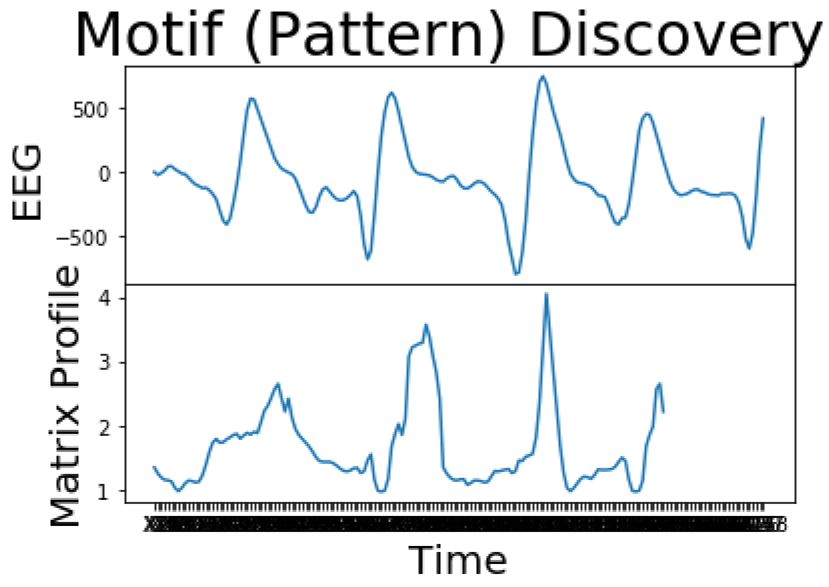
I also used the Matrix Profile to characterize different time series, using the STUMPY package (from here: https://stumpy.readthedocs.io/en/latest/). Matrix Profile is a powerful data mining technique that was recently discovered. It essentially takes every snippet of a certain window size, and then calculates the distance in the time series to that snippet's "nearest neighbor". It allows for a very powerful means of detecting motifs and anomalies.

I calculated it for the samples, with a Window Size of 30.

Here's a non-seizure EEG recording and its Matrix Profile graph

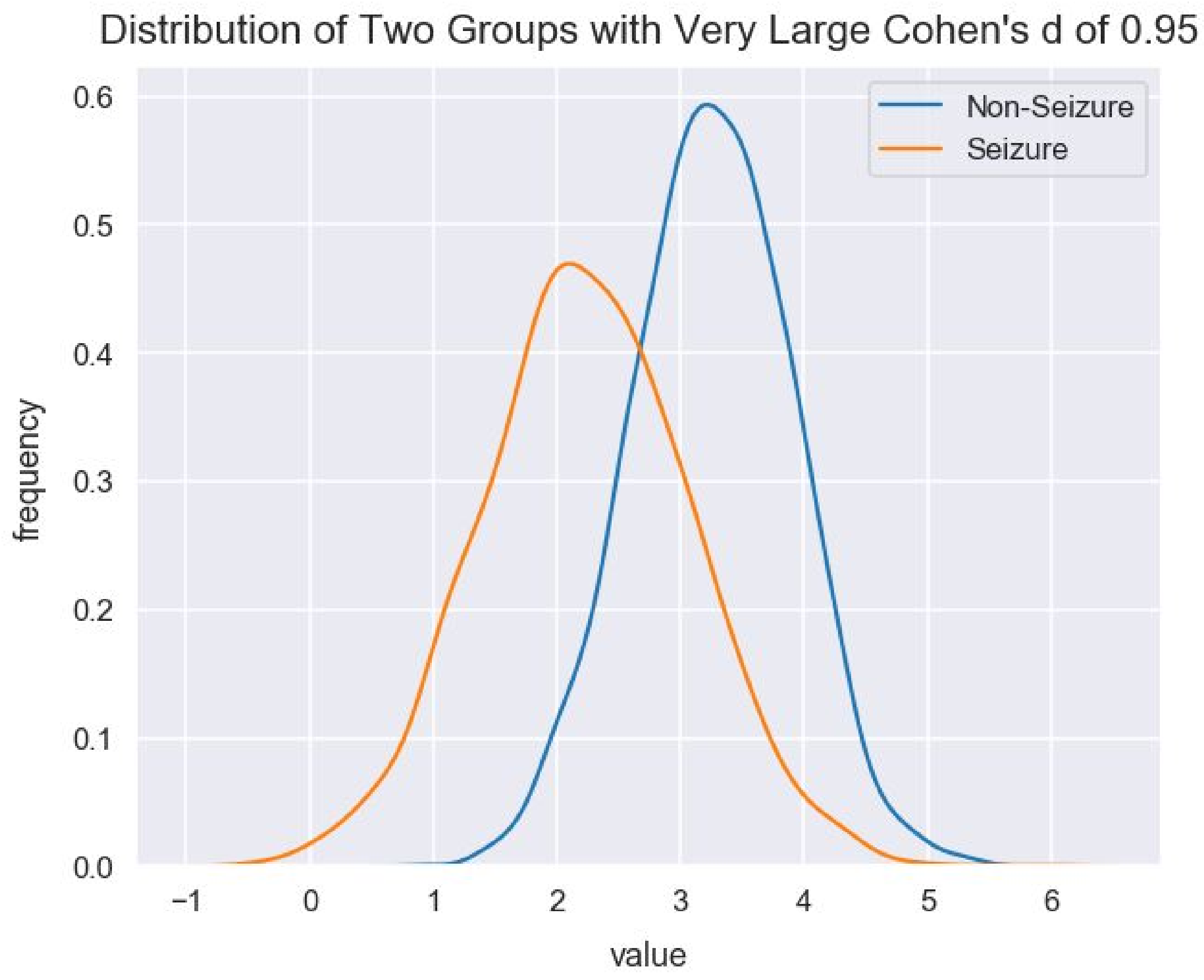


While here’s a recording with seizure activity

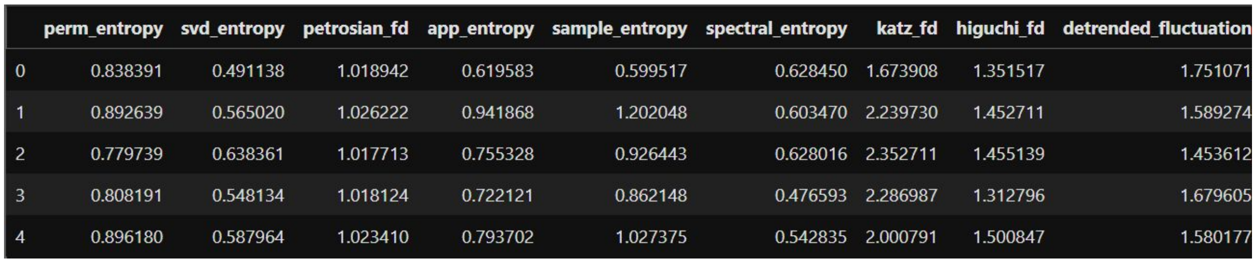


Note that the peaks are considerably lower for the Matrix Profile of the more-regular seizure recording.

Matrix Profile also allows us to summarize a given recording more easily. Taking the mean of a recording’s Matrix Profile gives us an idea of its overall regularity. I took a t-test on the means for the Seizure and non-Seizure recordings, and got a Cohen’s D of 0.95. Mean for the non-seizure group was 3.27, and 2.26 for the seizure group - so for any given 30-unit subsequence, the seizure group’s “nearest neighbor” was significantly closer.

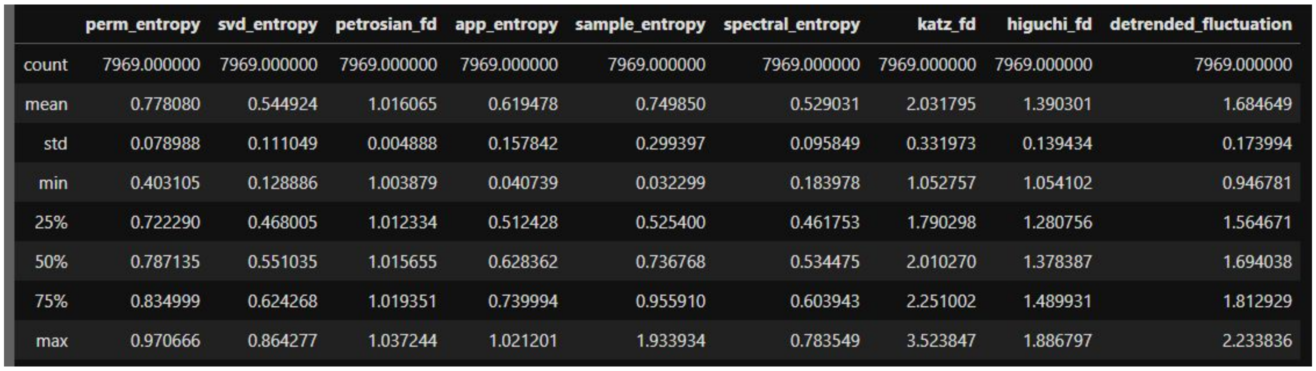


I also created a set of features for measuring EEG complexity from this package: https://github.com/raphaelvallat/entropy. These wound up being used in one of the final models. I discuss them in more depth below (when talking about Feature Importances), but here are some sample values.



	perm_entropy	svd_entropy	petrosian_fd	app_entropy	sample_entropy	spectral_entropy	katz_fd	higuchi_fd	detrended_fluctuation
0	0.838391	0.491138	1.018942	0.619583	0.599517	0.628450	1.673908	1.351517	1.751071
1	0.892639	0.565020	1.026222	0.941868	1.202048	0.603470	2.239730	1.452711	1.589274
2	0.779739	0.638361	1.017713	0.755328	0.926443	0.628016	2.352711	1.455139	1.453612
3	0.808191	0.548134	1.018124	0.722121	0.862148	0.476593	2.286987	1.312796	1.679605
4	0.896180	0.587964	1.023410	0.793702	1.027375	0.542835	2.000791	1.500847	1.580177

And some descriptives for those converted features:

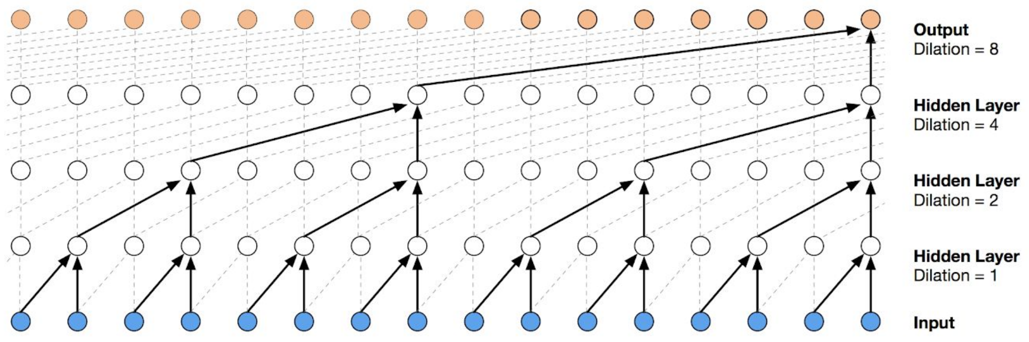


	perm_entropy	svd_entropy	petrosian_fd	app_entropy	sample_entropy	spectral_entropy	katz_fd	higuchi_fd	detrended_fluctuation
count	7969.000000	7969.000000	7969.000000	7969.000000	7969.000000	7969.000000	7969.000000	7969.000000	7969.000000
mean	0.778080	0.544924	1.016065	0.619478	0.749850	0.529031	2.031795	1.390301	1.684649
std	0.078988	0.111049	0.004888	0.157842	0.299397	0.095849	0.331973	0.139434	0.173994
min	0.403105	0.128886	1.003879	0.040739	0.032299	0.183978	1.052757	1.054102	0.946781
25%	0.722290	0.468005	1.012334	0.512428	0.525400	0.461753	1.790298	1.280756	1.564671
50%	0.787135	0.551035	1.015655	0.628362	0.736768	0.534475	2.010270	1.378387	1.694038
75%	0.834999	0.624268	1.019351	0.739994	0.955910	0.603943	2.251002	1.489931	1.812929
max	0.970666	0.864277	1.037244	1.021201	1.933934	0.783549	3.523847	1.886797	2.233836

Algorithms and Techniques
Ultimately I settled on two candidates for the final model. I explored XGBoost models on various summary statistics, and I look at Deep Learning models on the raw data itself as well as on Matrix Profile. I settled on XGBoost because, all other things being equal, I find it's the best choice for panel data. And Deep Learning has many useful architectures for sequence processing - especially when it's a noisy, fussy signal as one extracts from biological processes.

XGBoost, or Extreme Gradient Boosting, is a subset of Decision Tree models. Decision Trees essentially try to find the best set of questions to ask the data in order to determine the final answer - not unlike a game of 20 Questions. Decision Tree models are popular, among other reasons, because they require very little pre-processing (and every "data cleaning" step is a modeling decision where some information is lost). They're also able to find feature interactions in an organic way. Decision Trees tend to be very "path-dependent", however, so in practical usage these models tend to use ensembles of trees, each run on different subsets of features & data. There are two kinds - "Boosting" and "Bagging". XGBoost is a subset of the "boosting" models - where trees get trained sequentially and each new tree tries to get better performance on the trees that the previous set did poorly on. XGBoost in specific is a very refined version. Boosting trees used to be much slower to train compared to Bagging models (such as Random Forest), as Bagging models can be parallelized much more easily.

Temporal Convolutional Networks are a Deep Learning model that use layers of Convolutional Neural Networks to process time series data. Convolutional Neural Networks essentially "chunk" a set of data by moving along it with overlapping windows of a prespecified length, with the chunk size generally getting bigger as they're fed to higher layers. This allows them to process low-level motifs and then infer higher-order patterns from it. CNNs are generally used more for processing visual data, but there's nothing that stops it from being used with a 1D array. And, in 2016, Google's WaveNet (a special case of TCNs) turned out to perform excellently with complex tasks such as speech synthesis. They also have several advantages over Recurrent Neural Networks, which are what are traditionally used for processing sequences. For one thing, they are much better at handling long sequences. Recurrent Neural Networks generally have some method of "remembering" a certain set of previous states, as well as a mechanism for "forgetting" them. TCNs consider the structure of the whole sequence, however. In addition, CNNs are generally faster to train and benefit more from parallelism and especially GPUs. A diagram of the original WaveNet architecture (from here: https://github.com/philipperemy/keras-tcn)



Benchmark Model
This recent study's model https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6004942/ had an Accuracy of 97.75%, Sensitivity (Recall) of 90%, and Specificity (Precision) of 89.31%.   From these I calculate an F2 score of 0.8986.

Methodology

For the XGBoost model, I found that results were best with no preprocessing before extracting features. I tried XGBoost with several different sets of extracted features - tsfresh, frequency-domain, Matrix Profile summary statistics, and a set of metrics specifically designed to summarize EEG complexity from the EntroPy (https://github.com/raphaelvallat/entropy/tree/master/entropy) package. Of these, EntroPy did far and away the best. In addition, its transformations were relatively fast due to JIT-optimized code through the Numba package.

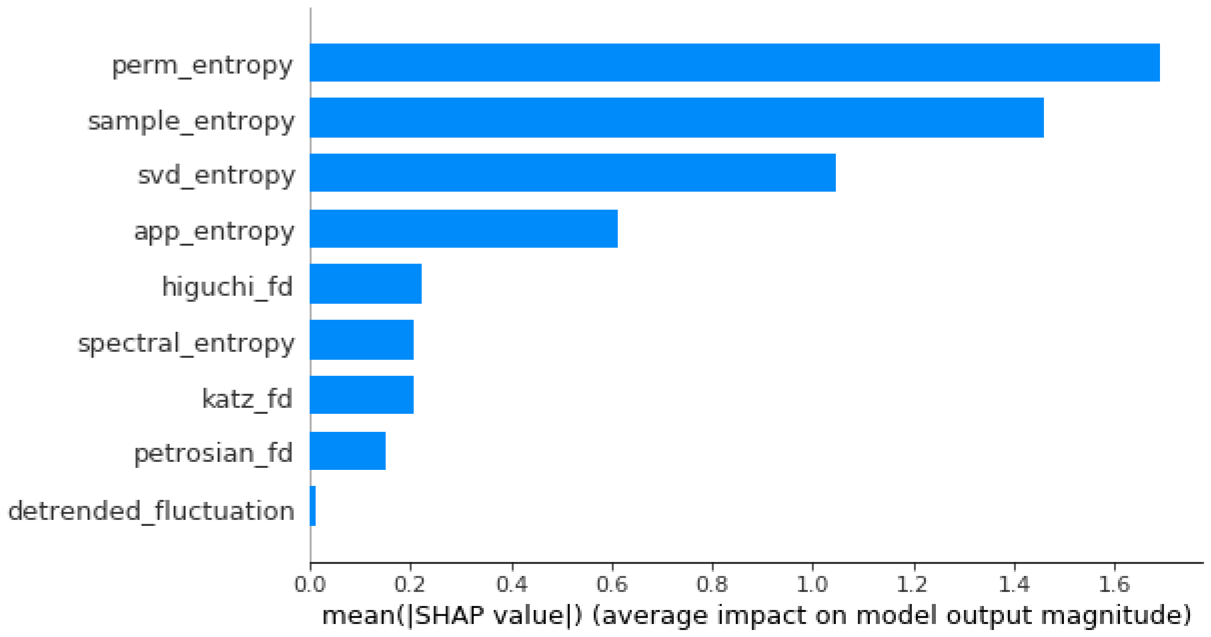
For the Deep Learning model, I experimented with several architectures. At first I used variations on Recurrent Neural Nets, but none of them did particularly well. I then looked into Temporal Convolutional Networks, starting with Google's famous WaveNet architecture. TCN which uses layers of 1D Convolutional Networks, each with a longer dilation, to extract time-dependent features of data. Lower levels extract smaller features and motifs, which then get processed by the higher layers. This performed by far the best, and was faster to train as well. I made it manually, but then found a package to make them in Keras that had slightly better performance (package, as well as a fantastic explanation of TCN, are here: https://github.com/philipperemy/keras-tcn). For these I did no preprocessing at all - I experimented with using several normalization techniques, as well as filtering out noise - but all reduced performance of the Validation set dramatically. This makes some theoretical sense - the dataset's instructions said to filter with a low-pass filter of 40Hz, but Convolutional Neural Networks can pick up signal from exactly that sort of seeming-noise.

Results
I’m choosing to report on both the XGBoost model with EntroPy features and the TCN model. The XGBoost model scored better (F2 of 0.8792), but that’s not the only factor - especially since this is for a Machine Learning Engineer nanodegree. The XGBoost model created its calculated features relatively quickly, but the fact that the TCN model required at all makes the cost/benefit tricky - especially for a model that might be used in a medical situation where every millisecond counts. For comparison’s sake, processing the Validation set (3416 records) took the XGBoost model 4.31 seconds, while the TCN model took 1.12 seconds.

	F2 Score	**Time to Process 3416 Records (s)**
XGBoost	0.8792	4.31
TCN	0.8323	1.12

For the XGBoost model, I used the SHAP package to examine feature importances. This uses a model from Game Theory to examine, for each prediction, how much each feature contributed to the outcome.

Feature importances:

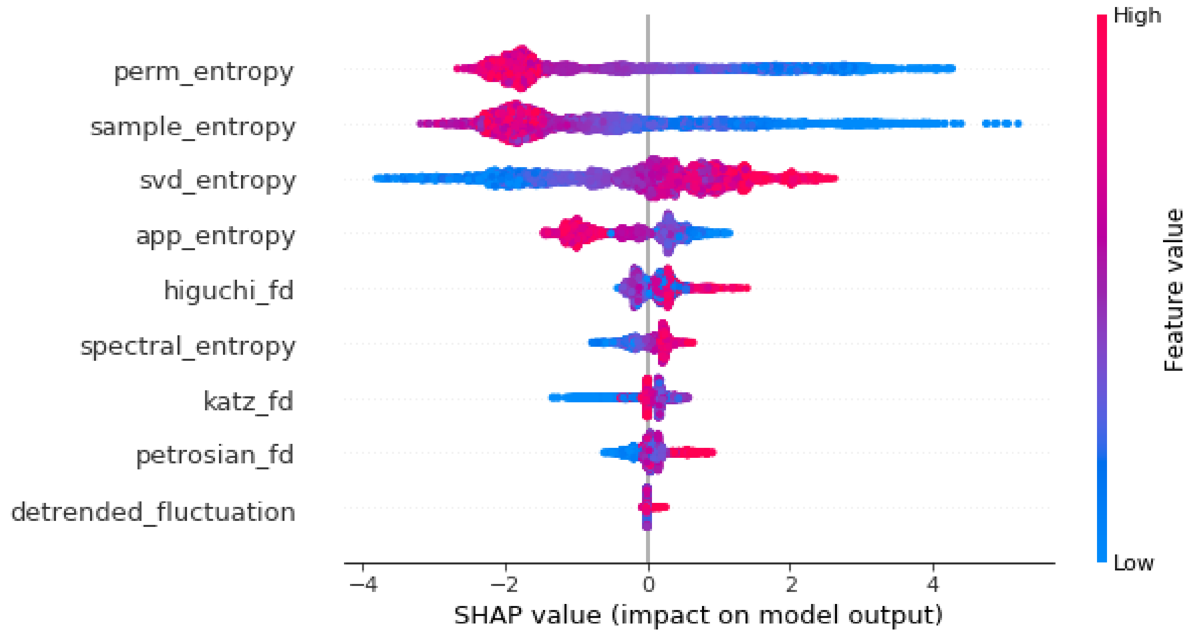


The EntroPy variables basically fall into two categories. There are Entropy variables - which seemed to be the most important here, particularly Permutation Entropy. Permutation Entropy basically takes every subsequence of a given length (3 in this case, based on default), and then bins them according to relative size of the number. For instance, the sequence “1, 6, 5” would be classified as a “0, 2, 1” permuation since the first element is the smallest, the second is the largest, and the third is in the middle. It then looks at how many of each type are present, and calculates the time series.

The other category is various forms of one-dimensional Fractal Dimension. These are measures of how self-similar a sequence is. These were relatively uninfluential compared to the Entropy variables.

SHAP also allows to not only see feature importance, but directionality - ie, whether a high value makes you more or less likely to be classified as seizure activity.

For the most part, lower Entropies mean more regularity - mostly the same “motifs” repeating. SVD Entropy is different, however - high SVD Entropy is related to higher likelihood of seizure. SVD Entropy effectively winds up measuring frequency complexity - more variation in the frequency means a higher entropy (more here: https://www.researchgate.net/publication/320341528_Time-Varying_Time-Frequency_Complexity_Measures_for_Epileptic_EEG_Data_Analysis).



My initial “handmade” TCN model eventually was replaced with a premade one, that had both faster training time and better performance. Hyperparameter optimization did not yield much in the way of significant improvement over defaults, though it might if I take more time to experiment with it - the number of “stacks” for TCN in particular was something I thought I could go somewhere with. In addition, the various calculated features for the XGBoost models all have parameters of their own that I could play with, but for the purposes of this project proved to be too many moving parts.