Ion Channels project - Final report

Berenice Dethier

Summary The goal of this Kaggle competition was to predict the number of open channels based on an electrophysiological signal measured by bioassay. In this document, we discuss how we explored and cleaned the signal, built models with ensemble algorithms and neural networks, and how our predictions performed in the competition. We ranked 1220 out of 2618, in the top 47%.

1 *Overview*

The Kaggle competition hosted by the University of Liverpool was a good fit for my next project: a deep learning approach is possible, it involves signal processing, is related to biochemistry, and has enough incentives to keep me motivated. The format was excellent for learning, as more experienced scientists were pitching in and sharing some of their findings, opening topics I wouldn't have had time to familiarize myself with. The goal of the competition was to predict the number of open channels based on the electric signal measured by a bioassay. An example of the signal and the number of open channels is presented Figure 1. The relevance of identifying the number of open channels is summarized by the host on the competition page:

Details about the competition can be found on Kaggle - Ion Switching Competition.

> "Many diseases, including cancer, are believed to have a contributing factor in common. Ion channels are pore-forming proteins present in animals and plants. They encode learning and memory, help fight infections, enable pain signals, and stimulate muscle contraction. If scientists could better study ion channels, which may be possible with the aid of machine learning, it could have a far-reaching impact.
>
> When ion channels open, they pass electric currents. Existing methods of detecting these state changes are slow and laborious. Humans must supervise the analysis, which imparts considerable bias, in addition to being tedious. These difficulties limit the volume of ion channel current analysis that can be used in research. Scientists hope that technology could enable rapid automatic detection of ion channel current events in raw data.
>
> The University of Liverpool's Institute of Ageing and Chronic Disease is working to advance ion channel research. Their team of scientists have asked for your help. In this competition, you'll use ion channel data to better model automatic identification methods. If successful, you'll be able to detect individual ion channel events in noisy raw signals. The data is simulated and injected with real world noise to emulate what scientists observe in laboratory experiments.
>
> Technology to analyze electrical data in cells has not changed significantly over the past 20 years. If we better understand ion channel activity, the research could impact many areas related to cell health and migration."

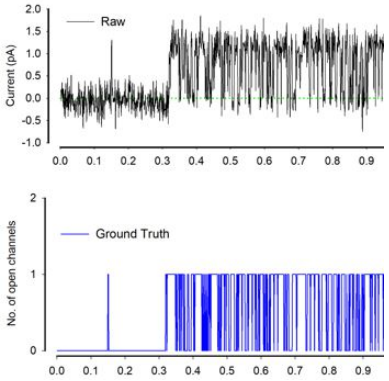

Figure 1: Example of a signal and number of open channels provided by the competition host, the University of Liverpool.

We started the exploratory data analysis (section 2) by looking at the signal and number of open channels in a few different situations, then took steps to clean the signal (section 3). Most of the work was performed with Jupyter Notebook IDE and Python 3.7, but the Kalman filter tuning was done with R (R Studio IDE).

I submitted 96 propositions during the competition. This report includes the most significant models and their performances (section 6). The best predictions were achieved with Catboost on a set of 139 features engineered from the signal, and with a Wavenet (type of CNN). Our highest macro F1 score was 0.9387 for Catboost models, and 0.9375 for Wavenets. This ranked us in the top 47% of the competitors. In comparison, the winners scored 0.9851 and the highest ranking team who did not exploit a leak in the data scored 0.9457. The project notebooks and other information can be found in the project GitHub repository.

2 *Exploratory Data Analysis*

Two data sets are intitially available, a training set consisting of 5,000,000 observations of the variables 'time', 'signal', and 'open channels', and a public test set consisting of 2,000,000 observations of 'time' and 'signal' only. A private test set is to be released on 05/18/2020. The code for the EDA was inspired by this notebook from Chris Deotte, and this notebook from Eunho Lee.

The documentation states:

> " While the time series appears continuous, the data is from discrete batches of 50 seconds long 10 kHz samples (500,000 rows per batch). In other words, the data from 0.0001 - 50.0000 is a different batch than 50.0001 - 100.0000, and thus discontinuous between 50.0000 and 50.0001."

We will therefore look at the batches as separate experiments.

Figure 2: The signal for the train set consists of 10 batches of 50 s measurements. The sampling rate is 10 kHz, so each batch contains 500,000 observations. When compared to the number of open channels (Figure 3-6), we notice the signal is correlated with the number of open channels. Batches have different amplitudes and baselines. There is drift in multiple batches.

When the signal and number of open channels are plotted over time (Figure 2), we notice the following things:

- Four batches have a small amplitude and small baseline (batches 1-3 and 7), four batches have an intermediate amplitude and baseline (batches 4, 6, 8 and 9), and two have a larger amplitude and a baseline slightly larger, too (batches 5 and 10). These caracteristics of the signal are correlated with the number of open channels: 0-1 open channels for batches 1, 2, 3 and 7, 0-3 open channels for batches 4 and 8, 1-5 open channels for batches 6 and 9, and 2-10 open channels for batches 5 and 10.
- Among the batches with 0/1 open channels, two have a low frequency of opening (batches 1 and 2), and two have a high frequency of opening (batches 3 and 7).
- There is drift in five of the ten batches: batch 2 (first part only), and 7-10 (parabolic drift).
- There seems to be noise in the form of a thick baseline when no channels are open (batch 1, for example) and in the form of spikes in signal (see btach 8, for example).

We can identify 5 behaviors in ion channels opening thanks to these observations:

1. **Type 1s**: batch 1 and batch 2. There is up to one open channel, and the rate for opening and closing is slow (Figure 3).
2. **Type 1f**: batch 3 and batch 7. There is up to one open channel, and the rate for opening and closing is fast (Figure 4).
3. **Type 3**: batch 4 and batch 8. There are between 0 and 3 channels open (Figure 5).
4. **Type 5**: batch 6 and batch 9. There are between 1 and 5 channels open (Figure 6, left).
5. **Type 10**: batch 5 and batch 10. There are between 2 and 10 channels open (Figure 6, right).

Figure 3: Example of signal and number of open channels typical of type **1s** signal.

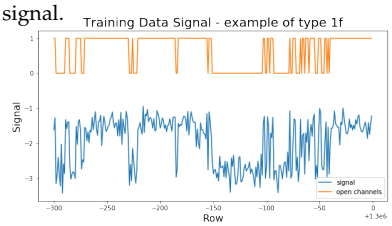

Figure 4: Example of a signal and number of open channels typical of **1f** signal.

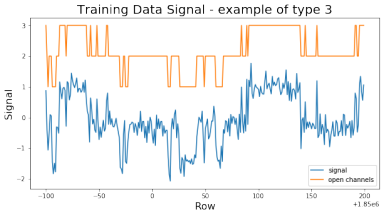

Figure 5: Example of a signal and number of open channels typical of type **3** signal.

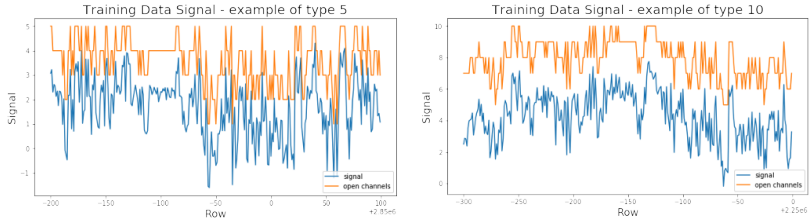

Figure 6: Example of a signal and number of open channels typical of type **5** signal (left) and of type **10** signal (right).

As a consequence, the following steps will need to be applied to the signal of the train and test set:

1. Remove the drift (areas to be processed will have to be identified visually)

2. Try applying filters to reduce the noise.

The distribution of observations depending on the number of open channels (Figure 7) indicates that zero open channels is the most common configuration (24.8% of the observations in that configuration) , followed by 1 (19.7%), 3 (13.4%), 2 (11.1%), 4 (8.1%), 5 (5.6%), 7 (5.3%), 6 (3.8%), 8 (4.9%), 9 (2.7%) then 10 open channels (0.7%).

We investigated the openings and closings of channels. A change in number of open channels will be called "event", and the number of timesteps the system remains in a certain state between two events will be called "interval". The distribution of intervals per number of open channels is summarized Figure 8. Three (3) open channels is the most common type of interval, then 2, 4, 7, 8, 1, 5, 6, 9, 0 and 10.

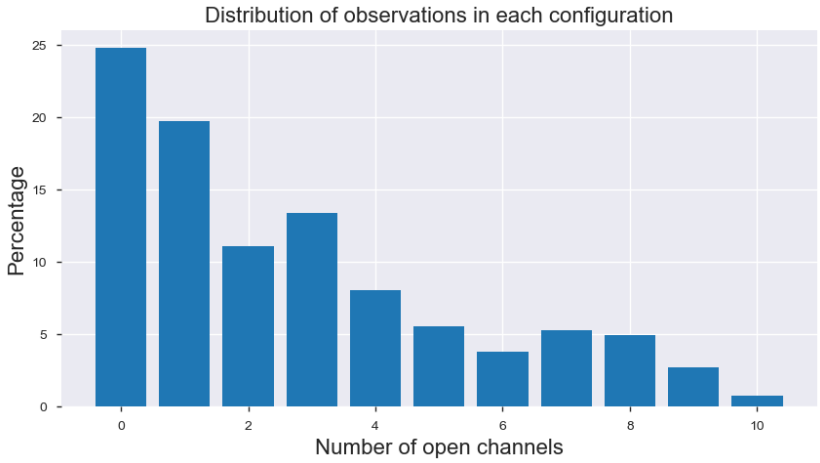

Figure 7: Zero open channels is the most common state, then 1, 3, 2, 4, 5, 7, 6, 8, 9, then 10 open channels.

To understand how the intervals are distributed, we plotted the count for each interval length on Figure 9. The ECDF is presented Figure 10.

Then, we looked at the boxplots (Figure 11). We see that 75% of intervals last between one and 5 timesteps (0 open channels), between one and 4 timesteps (1 open channel), between one and 3 timesteps (2-4 open channels), or between one and 2 timesteps (5-10 open channels). There are numerous outliers, with relatively high values (especially for 0 open channels: intervals were as long as 24000 timesteps).

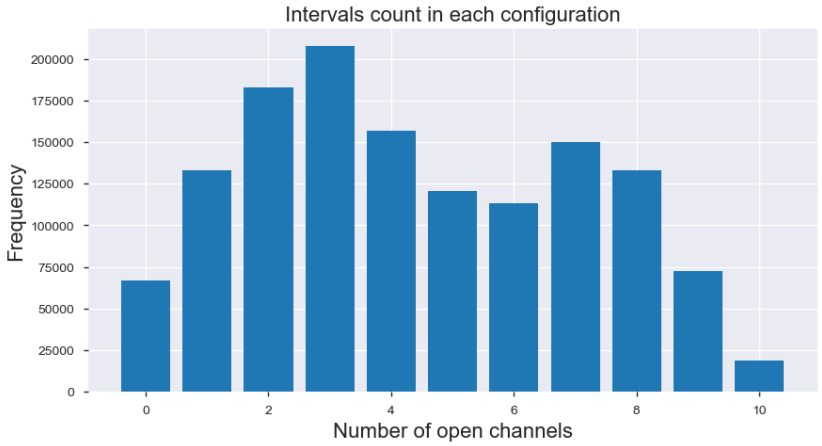

Figure 8: Three (3) open channels has the highest count of intervals, followed by 2, 4, 7, 8, then 1 open channels. There are less intervals for configurations when 5, 6, 9, 0 then 10 channels are open.

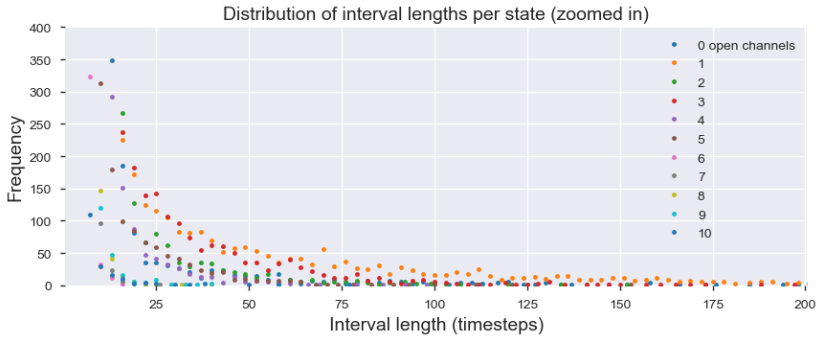

Figure 9: Most of the time, the system remains in one state for a short time (higher frequency for the short durations). There is more variability and longer intervals in the two following states: one channel open (orange trace) and three channels open (red trace).

Other statistics regarding the intervals are presented Table 1. These observations are consistent with the profiles of each of the five batches presented Figure 3-6. Batches 1s, 1f and 3, with low numbers of open channels, have long intervals. Batches 5 and 10, with the higher numbers of open channels, have shorter intervals/more frequent events.

Finally, the events type and frequency were plotted (Figure 12) to see which transitions were most common. Most timesteps (72.8%) are not events (difference in state is zero), and the distribution is very symetrical: -1 and 1 transitions each represent 11.6% of the timesteps, -2 and 2 represent 1.7%, and so on. A jump of 3, 4, and 5 number of open channels (whether positive or negative) occur in 0.2, 0.02, 0.002%, respectively.

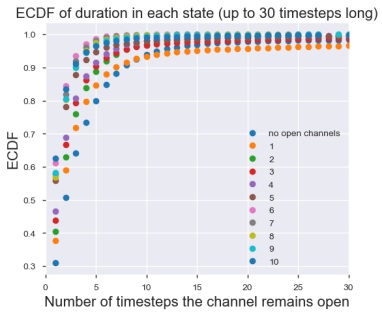

Figure 10: The observations are the same as for Figure 9: usually the intervals are short, and there are longer intervals for systems with one channel open (orange trace) and three channels open (red trace).

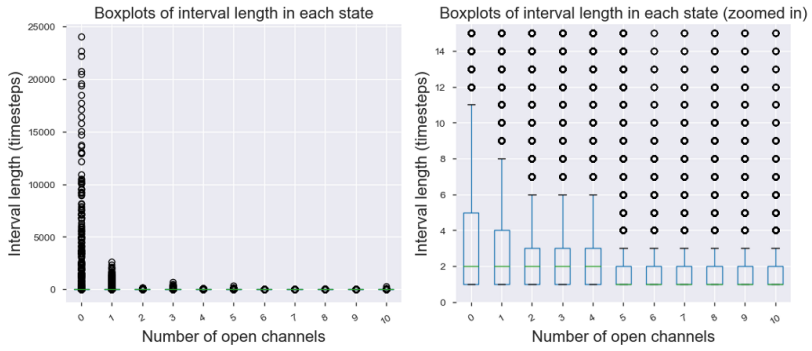

Figure 11: Boxplots of the intervals distribution per configuration.

	Number of open channels				
	0	**1**	**2**	**3**	**4**
Count	66626	133244	183147	207870	157010
Mean	18.6	7.4	3.0	3.2	2.6
Std	381.7	43.7	4.6	7.5	3.2
Min	1	1	1	1	1
25%	1	1	1	1	1
50%	2	2	2	2	2
75%	5	4	3	3	3
Max	24024	2596	168	685	101

	Number of open channels					
	5	**6**	**7**	**8**	**9**	**10**
Count	120620	113144	150272	133275	72645	18800
Mean	2.3	1.7	1.8	1.8	1.9	1.9
Std	4.1	1.1	1.3	1.4	1.7	3.2
Min	1	1	1	1	1	1
25%	1	1	1	1	1	1
50%	1	1	1	1	1	1
75%	2	2	2	2	2	2
Max	350	19	32	67	45	263

Table 1: Statistics about the interval lengths per number of open channels

The public test set is presented Figure 13. The subbatches are coded in different colors. There is also drift, and signal from all 5 groups described above, distributed as follows: 1s, 3, 5, 1s, 1f, 10, 5, 10, 1s, 3, 1s, 1s. In terms of timesteps for each group, the test set is composed of 65% of 1s data, 5% of 1f data, and 10% of data from each of the other groups. This imbalanced distribution will not impact the score however, as the competition is scored with macro F1 score, the unweighted average of the F1 scores of each class.

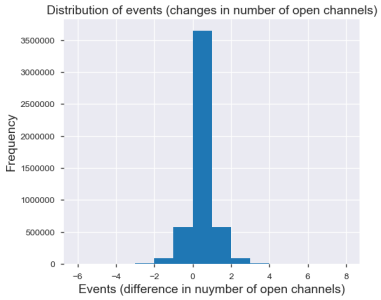

Figure 12: Changes in configuration are called events. They range from -6 (6 fewer open channels compared to the previous timestep) to 8 (8 extra open channels compared to the previous timestep).

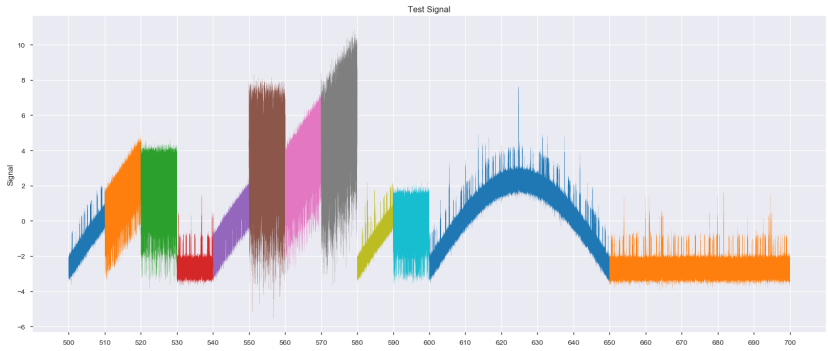

Figure 13: The test set consists of 4 batches, but the amplitude and drift are not uniform within each batch. The signal has been manually divided into subbatches color-coded on the figure.

3 *Signal Preprocessing*

The first step to clean the signal was the removal of the drift. This was done beautifully by Chris Deotte and is explained in details in this discussion. The drift had been added and uses a sine function, sometimes truncated (for example batch 2 of the training signal). The train and test signal with and without drift are presented Figure 15 and 16, respectively.

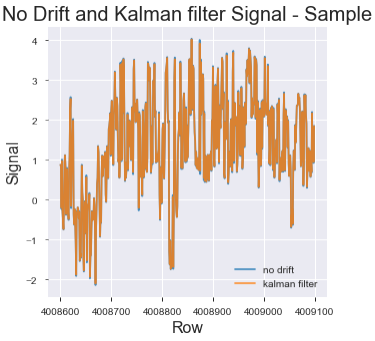

Figure 14: Sample of signal with and without Klaman filter (filter applied by michaln available here). The two signals are hard to distinguish, but the blue trace has a broader amplitude.

Figure 15: Plots of the train set signal with and without drift.

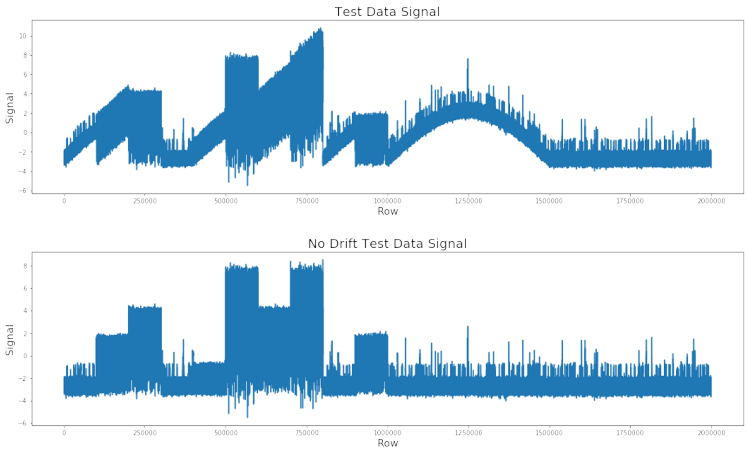

Figure 16: Plots of the test set signal with and without drift.

In addition, the signal is noisy. Applying a Kalman filter helped getting a cleaner signal. Kaggle user michaln shared a filtered signal here. Improvement due to the Kalman filter is not noticeable at full scale, see Figure 14 for a sample of the signal.

We tried to improve the signal by applying Kalman filters with different parameters in RStudio (functions SSModel and KFS from package KFAS, parameters: Q and H as arguments of SSModel). Examples of filtered signals are presented Figure 17 and 18. We tried various combinations of H and Q without investigating the theory. The level of noise is different, but it is hard to tell if the filtered signal will be a better feature than the raw signal, especially for higher numbers of open channels (signal not pictured). We will try multiple options during the modelling part of the project.

We looked at the Fast Fourier transform (FFT) of the signal to detect patterns and tune filter(s). There is a peak at 50 Hz, which corresponds to the line noise. To remove it, we created a notch filter, which takes the FFT of the signal, zeroes out the Fourier coefficients at/around 50 Hz, then takes the inverse FFT. FFT before and after notch filter are presented Figure 19. The small size of the peak at 50 Hz made us decide agaisnt applying the transformation after: we don't want to lose information around 50 Hz for this small source of noise. This portion of the work was also perform with R.

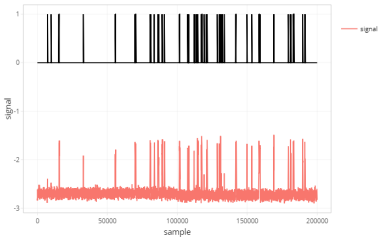

Figure 17: Sample of signal filtered by Klaman filter with parameters H =0.1 and Q = 0.001.

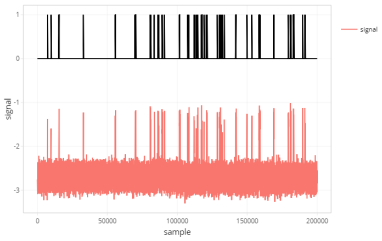

Figure 18: Sample of signal filtered by Klaman filter with parameters H =0.1 and Q = 0.1.

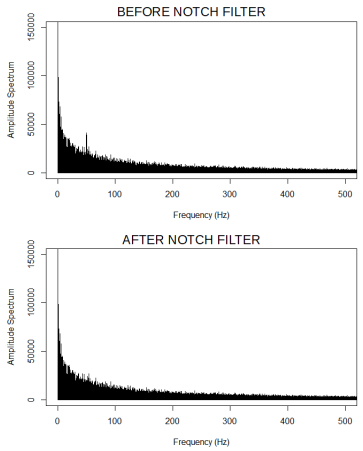

Figure 19: Notch filter applied on the line noise (50 Hz) of the signal: FFT of signal before (top) and after (bottom) transformation.

4 *Problem definition*

The labels of this datare integers between 0 and 10. We can treat it as a regression problem, especially since the events are mostly jumps of one number of open channels, or a classification. Some classes are underrepresented (6, 8-10 open channels), which might pose a

problem because the score for the competition is macro F1 score: the average of the F1 score for each class, unweighted. Wrong predictions for the less likely intervals are going to cost as much as other wrong predictions, even though less observations are available for training.

The signal without drift and processed with Kalman filter is the preferred candidate for modelling, but we have other options if the results are not good.

5 *Feature engineering*

The dataset provided by Kaggle has a single feature, the signal. We are going to generate other features derived from the signal and feed the collection of features to the algorithm.

5.1 *Rolling statistics and lag*

Since the signalk is a sequence of measurements, it seems interesting to work on a time window and extract statistics that might be relevant to predict the number of open channels. We computed seven (7) statistics for the centered window and the right-bordered window. Figure 20 illustrates the concept for $timestep = 1299750x10^{-4}$ s, with a centered and right window, respectively. Windows of 5, 15, 45, 135, 405, and 1215 timesteps were selected initially to provide a wide variety of window sizes.

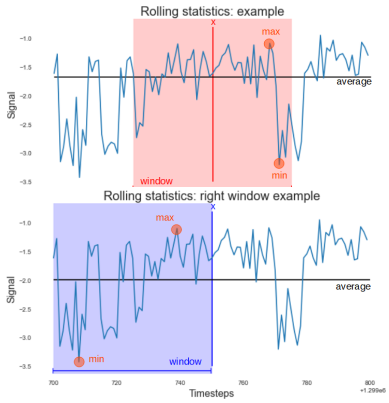

Figure 20: Example of a 49 timestep centered window (red) and right window (blue) for timestep 129.9750 s.

The following statistics were computed for each rolliong window:

1. Average
2. Standard deviation
3. Minimum
4. Maximum
5. Skew
6. Kurtosis
7. Weighted average (more weight given to signal of timepoints closer to x)

The statistics represented on the schema are the average, minimum and maximum. In addition, calculations on these statistics were also computed:

1. Difference between maximum and minimum
2. Ratio between minimum and maximum
3. Ratio of difference between maximum and minimum and window average

Similarly, it seems useful to provide previous and upcoming timesteps to the model. These are called lag variables, and for this project we generated variables with signal shifted by:

1. -1 timepoints (lag/previous timepoint)
2. -3 timepoints
3. -5 timepoints
4. +1 timepoints (next timepoint)
5. +2 timepoints

These intervals have been chosen arbitrarily, and based on other competitors' results.

5.2 *Other features*

Finally, we generated the following features:

1. x^2 (named power2)
2. x^3 (named power3)
3. Square root of x
4. Derivative of x
5. Lags of derivative (1, 3, -1 and -3)
6. Integral over last 10 points

Derivative The derivative seemed helpful for batches 1s and 1f because the spikes correspond exactly to opening/closing of channels (Figure 21). The derivative is less useful when the number of open channels goes up and down like for the other batches. We included it anyway.

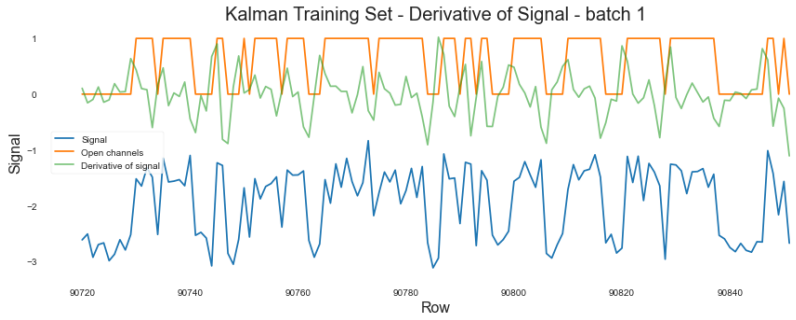

Figure 21: Example of signal, number of open channels and derivative of the signal for batch 1s. The green trace (derivative) spikes when a channel opens and dips when it closes.

Integral We looked into integration, which in this case is a cumulative sum of the signal. It was important to normalize the signal (by subtracting the mean), or the cumulative sum would have kept going down (the baseline signal is negative, at around -2.5). Figure 22 shows that there is a drop every time a channel opens. We thought of ways to use the integral to improve predictions, and decided to look at a short interval before x instead of the whole sequence.

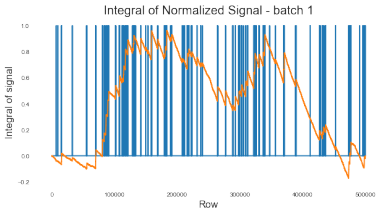

Figure 22: Integral/cumulative sum of the signal for batch 1s.

Figure 23: Example of signal, number of open channels and integral of the normalized signal over the last 10 timesteps for batch 1s and 5. The green trace (integral10) is similar to the signal with less noise.

After some tweaking, we found that calculating the integral over the previous x time points lead to a much smoother signal which followed the opening of channels closely. We arbitrarily decided to use 10 timesteps because it appeared to maximize the smoothing effect. Examples of the result are presented for batch 1 and for batch 5 (Figure 23). The signal had been normalized by batches.

Peak detected Finally, we uses `scipy` function `find_peaks` to detect peaks. The output is a bool, true if a peak is detected at the time-point, false if not. We called the variable peak_detected. The function takes multiple arguments to tune peak detection. Figure 24 and 25 give an idea of how the function works. We tweaked the parameters to flag open channels only, but to have as little false negative as possible. We want a high precision, that is to flag all the peaks that are open channel, even if we also flag a lot of peaks that are not open channels. We obtained a precision of 96.9%, but the recall was then 13.0%. We see that the tool could be useful for signal with one open channel (24), but less so with more open channels (25). We found out later that due to the nature of the signal this variable wasn't informative: it doesn't flag non-gaussian peaks, and our peaks are broad with a flat top.Instead, it labeled small variations as peaks.

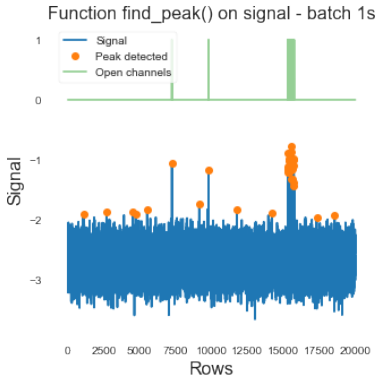

Figure 24: Function find_peak() applied to signal with 0/1 open channels: the peaks usually correspond to an open channel.

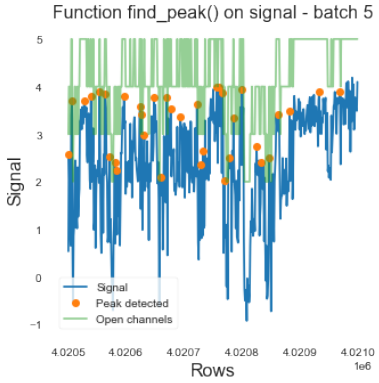

Figure 25: Function find_peak() applied to signal with more than one open channels: the peaks don't correspond to a change in open channels.

5.3 Importance of processing per batch

The signal is provided as a single table, although we know there are 10 separated batches of 50 s chained to another. With this in mind, we had to generate the features per batch: a rolling window at the edge of a batch would not have a correct value, because it would compute the statistics with signal from a different batch. We made sure to process each batch separately when generating features, then we concatenated them back together.

5.4 Handling of missing values

The signal itself does not contain missing values. Some features we engineered did, and here is how we avoided them:

1. Square root

   Negative signals would result in NAs. To avoid that, we extracted the signal and computed the square root of the absolute value of the signal. The latter is then multiplied by the sign.

   ```
   df['sqroot'] = np.sqrt(np.abs(df.signal)) * np.sign(df.signal)
   ```

2. Edges of rolling windows

   When we compute rolling statistics, if a window contains missing values, the result is also a missing value. It creates problem at the edges, especially for largers right windows such as 1215 timesteps, which would return 1214 NA values before being able to return correct statistics. To avoid this problem, `pandas` offers the argument `min_periods`. If set to 1, it will compute on a window as small as 1. For example, to compute the mean of various window sizes on regular (aligned to the right) windows, the code for computing the mean is:

   ```
   name = 'mean_'+str(window)+'_r'
   df[name] = df.signal.rolling(window = window, min_periods = 1).mean()
   ```

5.5 Final set of features

We ended up with a dataset containing 139 features. As we will see below, some will be removed based on how much they contribute to the predictions.

6 Machine learning

Now that we created our features, we need a way to assess our models' performances. The metric chose by the competition host is the

macro F1, which is an unweighted average of the F1 scores of all classes. It means that misassigning a label for a class that is under-represented has the same weight as a mistake made in classes where the algorithm had more opportunities to learn. Macro F1 is available in `scikit-learn`.

Some competitors chose to split the data into the five batches and train a different model for each batch. Because we are not sure that the test data have the exact same distribution of open channels per batch (i.e. there could be a batch with 1-10 open channels or any other variation), we decided against this strategy.

Model number	Submission score	Description	Type	Algorithm	Hyperparameters tuning	Dataset	Feature Engineering	Feature selection
1	0.129	Simple logistic regression on batch 0	Classifier	sk-learn logistic regression	none	No drift - Chris Deotte	None	none
2	0.625	Gaussian NB	Classifier	sklearn GaussianNB	none	No drift - Chris Deotte	None	none
3	0.363	Catboost regression with 92 features	Regressor	catboost	learning_rate=0.1 and default	No drift - Chris Deotte	No drift - 1st set	none
4	0.047	Gaussian NB with 92 features	Classifier	sklearn GaussianNB	none	No drift - Chris Deotte	No drift - 1st set	see note
5	0.167	Simple logistic regression on all batches - Kalman data	Classifier	sk-learn logistic regression	none	Kalman filter - michaln	None	none
6	0.634	Gaussian NB - Kalman filter	Classifier	sklearn GaussianNB	none	Kalman filter - michaln	Kalman - 2nd set	none
7	0.817	Catboost as Multiclassifier with 92 features - Kalman filter	Classifier	catboost	learning_rate=0.1 and default	Kalman filter - michaln	Kalman - 2nd set	95% importance
8	0.829	Catboost as Multiclassifier with 77 features - Kalman filter - optimized 95% importance	Classifier	catboost	iterations=5000, learning_rate=0.1, depth= 6, l2_leaf_reg= 5, random_strength= 20	Kalman filter - michaln	Kalman - 2nd set	95% importance
9	0.797	Catboost as Multiclassifier with 49features - Kalman filter - 80% importance	Classifier	catboost		Kalman filter - michaln	Kalman - 2nd set	80% importance
10	0.814	Catboost as Multiclassifier with 65 features 90% importance	Classifier	catboost		Kalman filter - michaln	Kalman - 2nd set	90% importance
11	0.938	Catboost as Multiclassifier with 122 features - Kalman filter	Classifier	catboost	learning_rate=0.1 and default	Kalman filter - michaln	Kalman - 3rd set	none
12	0.938	Catboost as Multiclassifier with 103 features - Kalman filter - optimized 95% importance	Classifier	catboost	iterations=5000, learning_rate=0.05 , depth= 10, l2_leaf_reg= 9, random_strength= 10	Kalman filter - michaln	Kalman - 3rd set	95% importance
13	0.938	Catboost as Multiclassifier with 60 features - Kalman filter - 80% importance	Classifier	catboost		Kalman filter - michaln	Kalman - 3rd set	80% importance
14	0.938	Catboost as Multiclassifier with 89 features 90% importance	Classifier	catboost		Kalman filter - michaln	Kalman - 3rd set	90% importance

Figure 26: Example of how I kept track of the many submissions.

Ninety-six (96) submissions were made for this competition. We kept track of each submission in a master table, a small portion is presented Figure 26. The first, very simple models (logistic regression, Naive Bayes) did not produce high leaderboard scores (although Naive Bayes was surprisingly good for a model that trained in a few seconds, with no feature engineering!). We moved to Catboost and CNNs, and these two approaches lead to public leaderboard[1] scores of 0.940.

[1] during the competition, the predictions submitted by competitiors are evaluated, and the score is calculated on a portion (30%) of the test set only. This score is called "public leaderboard". At the end of the competition, the score calculate on the entire test set is released. The latter is called "private leaderboard score". Sometimes, models overfit on the public portion of the test set, and the score goes down once calculated on the entire set. Competitors have to select two models for final submission without knowing the private scores.

6.1 Regression vs. classification

Two approaches were used to solve the problem: a regression moldel (the number of open channel is a number between 0 and 10), or a classification model (each configuration is then a class). A seen Figure 12, some transitions are of more than one channel opening/closing, which indicates a non-continuous dependent variable and possi-

bly poor prediction with a regression. We tried the two approaches, but the regression did not lead to good results. Catboost regression model had a macro F1 score of 0.36, in sharp contrast with Catboost first multi-classification results (0.83, same number of features). The classification method was selected for the rest of the competition.

6.2 Choosing a cleaned signal

We started using the signal cleaned by Chris Deotte, then used the signal with a Kalman filter provided by michaln. As described in section 3, we applied Kalman filters ourselves to try and improve noise removal. The submission score for our cleaned signal was 0.861 and 0.874, a setback from 0.940 (at the time) on michaln's signal. We therefore used michaln's signal for the rest of the project.

6.3 Decision tree models

I had a great experience with Catboost in a previous project and started off with Catboost. XGBoost was also used, as well as regular Random Forest. We often looked at the features sorted by importance and selected the top 80-95% contributing features. No group of features stood out (not all/lots of features from a certain window, or a ceratin type of statistics), except the log which was removed in later implementations. Had anything been moticed, we could have used the information to engineer better features.

Model number	Submission score	Description (all are Catboost Multiclassifiers)	Feature Engineering
7	0.817	Catboost Multiclassifier with 92 features	aggregation function on rolling windows, right and centered, window sizes: [5, 7, 11, 31, 101]
8	0.829	77 features - optimized 95% feature importance from model 7	
9	0.797	49 features - 80% feature importance from model 7	
10	0.814	65 features 90% feature importance from model 7	
11	0.938	Catboost Multiclassifier with 122 features	aggregation function on rolling windows, right and centered, window sizes: [5, 15, 45, 135, 405, 1215]
12	0.938	103 features – hyperparameters tuned - 95% feature importance from model 11	
13	0.938	60 features - 80% feature importance from model 11	
14	0.938	89 features - 90% feature importance from model 11	
15	0.839	Catboost Multiclassifier with 133 features	same as model 11 with the addition of lag and power features. The signal was shifted up to avoid negative signal.
16	0.938	Catboost Multiclassifier with 133 features	same as model 11, with the addition of lag and power features (no vertical shift of signal)
17	0.937	103 features - hyperparameters tuned - 95% feature importance from model 16	
18	0.938	Same as 17 with more iterations	
19-20	0.938	Catboost Multiclassifier with 138 features	same as model 16, with additional lag features (1 and 3 on both sides) and derivative instead of log
21	0.939	Added lag of the derivative to model 19	Same as model 19, with additional lag of the derivative features
23	0.939	114 features – 95% feature importance from model 21	
24	0.939	114 features – hyperparameters tuned – 95% feature importance from model 21	

Figure 27: Ensemble models: part 1/3.

We started with the rolling statistics (and signal) as the only features for submissions 7-14 (Figure 27). We tried to find the best window sizes, and we aimed to remove useless features. There was no real trend when we looked at the feature importance. We obtained scores of 0.938, which was extremely encouraging. Starting at model 15, we included lag, logarithm, power2, power3 and square root. We also tried to add the absolute minimum signal to the signal to shift all the values up and avoid negative signal. The goal was to avoid NA values for the square root, and to hopefully improve the score. The score went down, and model 16 and beyond use the non-shifted signal. We later included more lag features, the derivative of the signal (replacing the logarithm, starting at model 19), then lags of the derivative (staring at model 21). This last change improved the score to 0.939.

Model number	Submission score	Description (all are Catboost Multiclassifiers)	Feature Engineering
25	0.939	98 features – 90% feature importance from model 21 - peak_detected added	Additional categorical feature: peak_detected
26	0.940	Hyperparameters tuned – peak_detected added to model 25	
31	0.660	Catboost Multiclassifier with 141 features	Same as 21 (removed peak_detected) with normalized signal, integration and integration on previous 10 timesteps
32	0.682	Catboost Multiclassifier with 142 features	Same as 31with peak_detected
33	0.939	Catboost Multiclassifier with 142 features	Same as 31, no signal normalization
34	0.940	95% feature importance from model 33	
35	0.940	Model 34 – hyperparameters tuned	
36	0.940	Model 33 – hyperparameters tuned	
37	0.861	Catboost as Multiclassifier with 142 features	Kalman filters applied by us (not from michaln)
39	0.874	90% feature importance from model 37	
40	0.868	90% feature importance from model 37, new Kalman filter parameters	
41	0.843	90% feature importance from model 37, hyperparameters tuned	
42	0.584	Models stacking: Random Forest and Naive Bayes	Features identical to model 31
43	0.524	Models stacking: Naive Bayes and catboost	
44	0.935	Models stacking: optimized RF and catboost	
45	0.935	Models stacking: optimized RF (95% feature importance) and catboost	

Figure 28: Ensemble models: part 2/3.

For models 25-45, we already had a solid score with Catboost which we were trying to improve (Figure 28). We included the peak_detected feature discussed in section 5, which improved the score to 0.940 for the first time (model 26). We wanted to use the integral of the signal over the previous 10 timesteps (see section 5), but we had to normalize the signal to obtain a meaningful value. If all the features are generated on the normalized signal, the resulting model performs very poorly (model 31, 0.660). We went back to signal normalization starting at model 33, and the score went back to 0.940. Models 37-41 describe models built with signal processed through our own Kalman filters, which was unsuccessful (see above).

Finally, we tried to stack Catboost and other models: a random forest (RF) and the promising Naive Bayes model. Models 42 and 43 display strong overfitting on the test set, the predictions on the train set were much better. Besides overfitting, stacking did not improve the leaderboard score (models 44-45).

Model number	Submission score	Description (all are Catboost Multiclassifiers)	Feature Engineering
46	0.940	Catboost Multiclassifier with 139 features	Features identical to model 31, generated per batch (to avoid issues at batches edges)
47	0.940	Hyperparameters tuned for model 46	
48	0.940	Models stacking: RF and catboost with features generated per batch	
49	0.939	RF alone	
53	0.939	95% feature importance from model 47	
54	0.937	90% feature importance from model 47	
51	0.934	XGBoost on FE per batch	
52	0.939	XGBoost with softmax	
55	0.929	XGBoost with softmax - maximized	
56-57	0.929	XGBoost with softmax - minimized	
71	0.939	Catboost Multiclassifier with 140 features	Features identical to model 31, added is_jump
75	0.940	Catboost Multiclassifier with 140 features	Features identical to model 31, added high_catpred (prediction from a separate Catboost model on batch 5 and 10 only)
76	0.940	Hyperparameters tuned for model 75	
77	0.938	Hyperparameters tuned for model 75, using class imbalance	
82	0.940	Catboost Multiclassifier with 139 features	Features identical to model 31, improved square root to avoid NA values when negative

Figure 29: Ensemble models: part 3/3.

For the last series of ensemble models (models 46-82, Figure 29), we improved the features by generating them per batch (models 46-54, see section 5 for the rationale) and worked on missing values by calculating the square root on the absolute value of the signal then multiplying by the sign (model 82). We tried is_jump feature with poor results (model 71), and high_catpred (models 75-77) with more interesting results. We included the class imbalance parameter of catboost in model 77 with a decrease in prediction performance.

Finally, we tried XGBoost on the most promising set of features (models 51-57). The highest score with XGBoost after optimization was 0.939, very similar to Catboost.

The best score with Catboost on the public leaderboard was 0.940, and we had multiple models reaching this score. We selected the very last model, with features engineered by batches and the improve square root feature, for final submission (model 82). It scored 0.9372 on the private leaderboard score. Figure 30 summarizes the public and private scores for models having scores 0.938 and more. We had 16 models which scored higher than that on the private leaderboard, with the highest private score for model 35 (features not generated per batch, with feature "peak_detected", 95% of feature importance optimized model). If we had selected model 35 for final submission, we would have ranked 1120th instead of 1220th.

When we look at the macro F1 scores per batch (example 2) we re-

Model number	Public score	Private score	Description
11	0.938	0.9365	Rolling statistics only
12	0.938	0.9368	Hyperparameters tuned - 95% feature importance
13	0.938	0.9369	80% feature importance
14	0.938	0.9367	90% feature importance
16	0.938	0.9366	Addition of lag and power features (no vertical shift of signal)
19-20	0.938	0.9381	Additional lag features (1 and 3 on both sides) and derivative instead of log
21	0.939	0.9383	Addition of lag of the derivative
23	0.939	0.9385	95% feature importance
24	0.939	0.9385	Hyperparameters tuned – 95% feature importance
25	0.939	0.9385	90% feature importance - peak_detected added
26	0.940	0.9380	Hyperparameters tuned – peak_detected added
33	0.939	0.9380	Addition of peak_detected, no normalization
34	0.940	0.9385	95% feature importance
35	0.940	0.9387	Hyperparameters tuned – 95% feature importance
36	0.940	0.9384	Hyperparameters tuned
46	0.940	0.9374	Features generated per batch
47	0.940	0.9377	Hyperparameters tuned
48	0.940	0.9379	Models stacking: RF and catboost, features generated per batch
49	0.939	0.9362	RF alone
53	0.939	0.9369	95% feature importance
52	0.939	0.9363	XGBoost with softmax optimization
75	0.940	0.9381	Addition of high_catpred (prediction from model trained on bacth 5 and 10 only)
76	0.940	0.9384	Hyperparameters tuned
77	0.938	0.9367	Hyperparameters tuned, using class imbalance
82	0.940	0.9372	Features not generated per batch, square root calculations to avoid NA

Figure 30: Private and public scores for Ensemble models performing 0.938 and above on the public leaderboard. Model 82 was the one submitted for the competition, and model 35 was the best performing model in hindsight. The two are highlighted in yellow. Hues of green in the score columns are proportional to the score (the darker the higher).

alize that the models perform remarkably well on batches 1s, 1f and 3, but not so well on batches 5 and 10. This observation had lead to the the idea of having one model for predicting high open channels numbers, which was then incorporated as a feature of models 75-77.

Batch type	Macro F1 score
1s	0.9975
1f	0.9971
3	0.9810
5	0.8364
10	0.8862

Table 2: Example of macro F1 score calculated by batches. These are the results for model 33, but all our models displayed the same trend: good predictions for batches 1-3 and poorer predictions for batches 5-10.

We tried to understand where the models were failing. Besides the F1 scores per batch, we calculated the percentage of correct predictions for each configuration (Table 3). The proportion of wrong predictions increases with the number of open channels, with configurations above 5 open channels having only 91% of correct predictions. We also plotted the difference between predicted and true number of open channels over time (Figure 31) and noticed the amplitude of the mistakes were small (1 unit of difference) except around 3.5-4 M timesteps, mistakes were up to three units large. We could have worked on reducing noise for that section of the train set.

Open channels	0	1	2	3	4	5
Percentage correct	99.8	99.4	98.3	98.4	97.7	95.8
Open channels	6	7	8	9	10	
Percentage correct	90.6	91.1	91.4	91.5	90.9	

Table 3: Example of model failure analysis for model 76. The percentage of correct predictions is higher for configurations below 6 open channels.

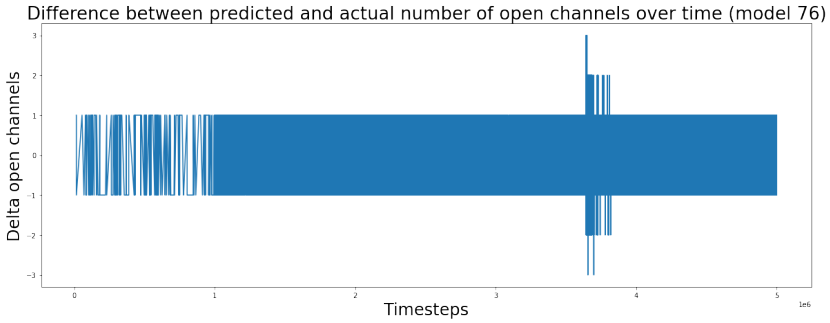

Figure 31: Difference between predicted and actual number of open channels over time. Most predictions were one unit off, except in the XX area.

6.4 *Neural networks*

In parallel, we started to work on neural networks, with the signal as the only feature. The following characteristics were common to all the networks:

- TensorFlow/Keras was used
- Label classes were one-hot encoded (sklearn LabelEncoder)
- The activation for the output layer was 'softmax'
- Optimizer was Adam
- We applied categorical cross-entropy loss
- Batch size was 32
- We ran for 120-500 epochs with a learning rate scheduler ranging from $1e^{-3}$ to $1e^{-4}$
- We used early stopping and model checkpoint callbacks
- The nets are trained with GPU acceleration

Models are summarized Figure 32 and 34. We started with fully connected and 1D CNN networks with terrible prediction scores (models 22, 27-29). Convolution seemed appropriate since the signal is a continuous variable.

Based on other competitors' experiences, we switched to Wavenets. Wavenets are explained by Joseph Eddy as follows:

Model number	Public score	Private score	Description (by default, signal is the only feature)	Activation
22	0.122	0.132	Fully connected (Dense – 100 nodes)	ReLu
27	0.060	0.048	1D CNN (2 Conv1D – 16 filters, 2 kernels; Dropout 0.5; Flatten; Dense 100 nodes)	ReLu
28	0.118	0.110	1D CNN (2 Conv1D – 32 filters, 5 kernels; Dropout 0.5; Flatten; Dense 100 nodes)	
29	0.131	0.129	1D CNN (4 Conv1D – 128 filters, 6 kernels; Dropout 0.5; MaxPooling 1; Flatten; Dense 100 nodes)	
58	0.939	0.938	Wavenet (4 dilations of [Conv1D – 16 filters, 1 kernel; WaveNetResidualConv1D – 16 filters, 3 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [12, 8, 4, 1]])	Sigmoid/ mish
59	0.937	0.936	Wavenet like model 58 with **3 features (signal, power2 and exp)**	
60	0.940	0.938	Wavenet like model 58 with **2 features (signal, catboost predictions)**	
61	0.934	0.934	Wavenet like model 58, data split sequentially	Sigmoid/ mish
62	0.937	0.936	Wavenet like model 61 starting with 32 filters instead of 16	
63	0.939	0.937	Wavenet like model 61 with 5 dilations instead of 4	
64	0.936	0.934	Wavenet like model 61 including improvements from 62 and 63	
72	0.904	0.904	Wavenet like model 63 with **2 features (signal, is_jump)**	
78	0.940	0.937	Wavenet like model 62 with **2 features (signal, catboost predictions)**	
79	0.935	0.934	Wavenet (4 dilations of [Conv1D – 32 filters, 3 kernels; WaveNetResidualConv1D – 32 filters, 3 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [12, 8, 4, 1]]) with **4 features (signal, high_catpred, power2, exp)**	Sigmoid/ mish
80	0.931	0.925	WaveNet like model 79, BatchNormalization before the output layer and 4 features	
81	0.213	0.180	WaveNet like model 79, BatchNormalization after each WaveNet and 4 features	

Figure 32: Neural Network models: part 1/2.

> At the heart of WaveNet's magic is the dilated causal convolution layer, which allows it to properly treat temporal order and handle long-term dependencies without an explosion in model complexity.

A comparison between causal CNN and Wavenet is presented Figure 33. Wavenets gave much better results, comparable to Catboost (models 58-81, and 85-92). We tried various architechtures, and used 1-3 important features (as determined by catboost) and/or catboost predictions to improve the models. Despite our best efforts, we could not go past 0.940 on the public leaderboard. The specifics of the dilation and mish activation can be found in the original author's notebook.

We tried a fully connected net (model 83) with the 139 features from our Catboost model, with a macro F1 score of 0.936 (public) and 0.852 (private). We also tried to solve the class imbalance by using SMOTE (as suggested by the work of the competition hosts' github repository), but that lead to huge overfit (model 84).

6.5 Final submission

In the end, Catboost was superior to all the neural networks (except neural networks using catboost predictions as a feature). At this point, we had numerous models performing at 0.940, but needed to pick only two for final submission. We selected what we thought was the most promising Catboost model, model 82, and averaged

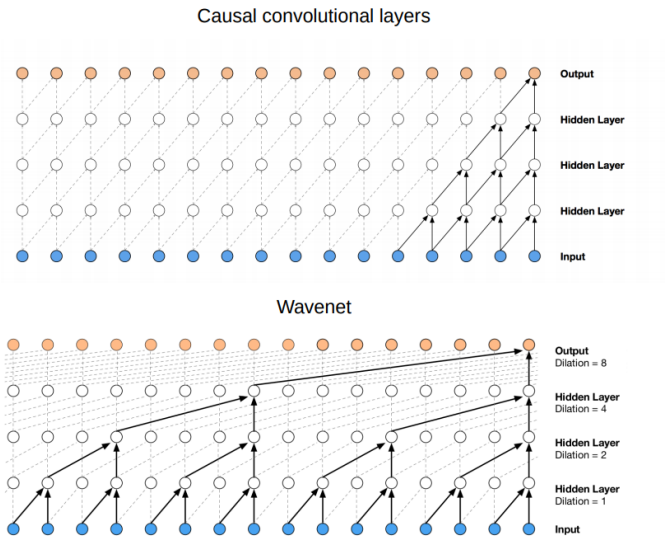

Figure 33: CNN and Wavenet schemes. From Oord et al., 2016, "WaveNet: A Generative Model for Raw Audio".

five top performing models for our second submission (model 96 of Figure 34): two wavenets (models 60 and 78) and three Catboost models (models 48, 76 and 82). The average resturned a public score of 0.94097 and private score of 0.93793, so slightly above model 82 alone.

7 *Results of the competition*

7.1 *Winners of the competition*

The competition hosts had made public that some of the data had been simulated. One way to win the competition was therefore to reverse engineer the simulated data. Two teams were abe to take advantage of a "leak" in the data. They realized that batch 10 had twice the variance of the other batches, and from there discovered that batch 10 in the test set was the result of adding signals from batch 5. Therefore, they scored very high on the private leaderboard (batch 10 data was in the private portion of the test set), with a macro F1 of 0.9851. The second team discovered some but not all of the trick and scored 0.9582. While it is disappointing to have been measured against non machine learning approaches, the predictions of these two teams had the highest scores, and the issue is the responsibility of the organizers, not the competitiors.

To compare my performances against teams who used strictly machine learning, we need to look at the 3rd team. They scored 0.9457, only 0.007 points away from my highest submission. They were able to reach this score by using hidden Markov modeling. In

Model number	Public score	Private score	Description (by default, signal is the only feature)	Activation
83	0.936	0.852	Fully connected (4 time [Dense -11*i nodes; Dropout 0.2] with I = [16, 8, 4, 2]) with **139 features**	ReLu
84	0.112	0.079	1D CNN using SMOTE with **5 features (signal, catboost predictions, exp, power2 and sqroot)**	
85	0.938	0.931	Wavenet (3 dilations of [Conv1D – 64 filters, 4 kernels; WaveNetResidualConv1D – 64 filters, 4 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [8, 4, 1]]) with 5 **features (signal, catboost predictions, exp, power2 and sqroot)**	
86	0.940	0.937	Wavenet like 85, BatchNormalization after 1st Conv1D	
87	0.939	0.929	Wavenet like 86, 400 epochs	
88	0.940	0.937	Wavenet like 86, 500 epochs	
89	0.939	0.830	Wavenet (3 dilations of [Conv1D – 64 filters, 1 kernels; WaveNetResidualConv1D – 64 filters, 1 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [8, 4, 1]]), BatchNormalization after 1st Conv1D, with **5 features (signal, catboost predictions, exp, power2 and sqroot)**	Sigmoid/ mish
90	0.939	0.932	Wavenet (3 dilations of [Conv1D – 64 filters, 2 kernels; WaveNetResidualConv1D – 64 filters, 2 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [8, 4, 1]]), BatchNormalization and MaxPooling after 1st Conv1D, with **5 features (signal, catboost predictions, exp, power2 and sqroot)**	
91	0.940	0.937	Wavenet like 90, 3 kernels instead of 2 and 500 epochs	
92	0.940	0.937	Wavenet (4 dilations of [Conv1D – 32 filters, 2 kernels; WaveNetResidualConv1D – 32 filters, 2 kernels, stacked layers] with filters*(2^i) at each iteration i and stacked layers = [12, 8, 4, 1]]), BatchNormalization after 1st Conv1D, **with 5 features (signal, catboost predictions, exp, power2 and sqroot)** 5000 timesteps per batch	
96	0.941	0.938	Average of 5 models: 2 Wavenets and 5 Catboosts	NA

Figure 34: Neural Network models: part 2/2.

their words:

> "One should approach this competition as an optimization problem with an element of machine learning, rather than a machine learning problem with an element of optimization. We found that pure optimization works really great for this synthetic data, and no serious overfitting has been observed. "

Other top performing models used the following implementations and had superior results:

- Removal of the noise coming from the power line at 50 Hz. We had tried to play with the signal and gotten lower F1 scores, but we didn't try the 50 Hz filter by itself (in combination with other Kalman filter changes).
- Removal of outliers in the 8th batch of data (they excluded these from the training set, see Helgi's solution)
- Stacking Keras models with no early stopping: the one with best f1 score, with lowest categorical cross entropy and the lates (still Helgi's solution)
- Grouping batch type 1s, 1f, 3 and 5 into a segment group 0 and batch type 10 into segment group 1 and used the segment group as a feature in modeling. This takes into account the different

nature of batch 10 with less risk of overfitting. This trick is also from Helgi's implementation.

- Using noise as a feature. The 5th team used an XGB Classifier to tune predictions from Wavenet (using signal, wavenet predictions and noise as features, see 5th team solution)

7.2 My results

The first few submissions were very promising. We quickly went from nothing to 0.9. After that, we stalled at 0.940, despite efforts to analyse where the models failed.One team who removed the noise in the 8th batch of data, which could probably have helped us given that our worse predictions were in at area (see Figure 31). The two models we selected for final submission scored 0.9387 and 0.9375 (Catboost on a set of 139 features engineered from the signal and Wavenet, respectively). The former placed us in the top 47% of the competitors.

Overall, our models were good at predicting batches of type 1-5, but performed poorly on type 10. After the leak was revealed, it made sense: type 10 was not the same at all and maybe should have required its own model.

As described Figure 30, 32 and 34, and discussed above, our best performing model on the private leaderboard was not one of the two selected for submission. Instead, it was model 35, a Catboost model using 112 features generated from the signal. That model would have been ranked 1120 out of 2618 (top 43%).

I had some good ideas, and I am overall very happy with my participation in this competition. I learned a lot about signal processing, HMM, noise and drift removal, and neural networks/CNN, and I came close to some of the best models. I particularly enjoyed the help and sharing of notebooks by experienced competitors early in the competition. The collaborative atmosphere allowed for some fantastic learning on my end!