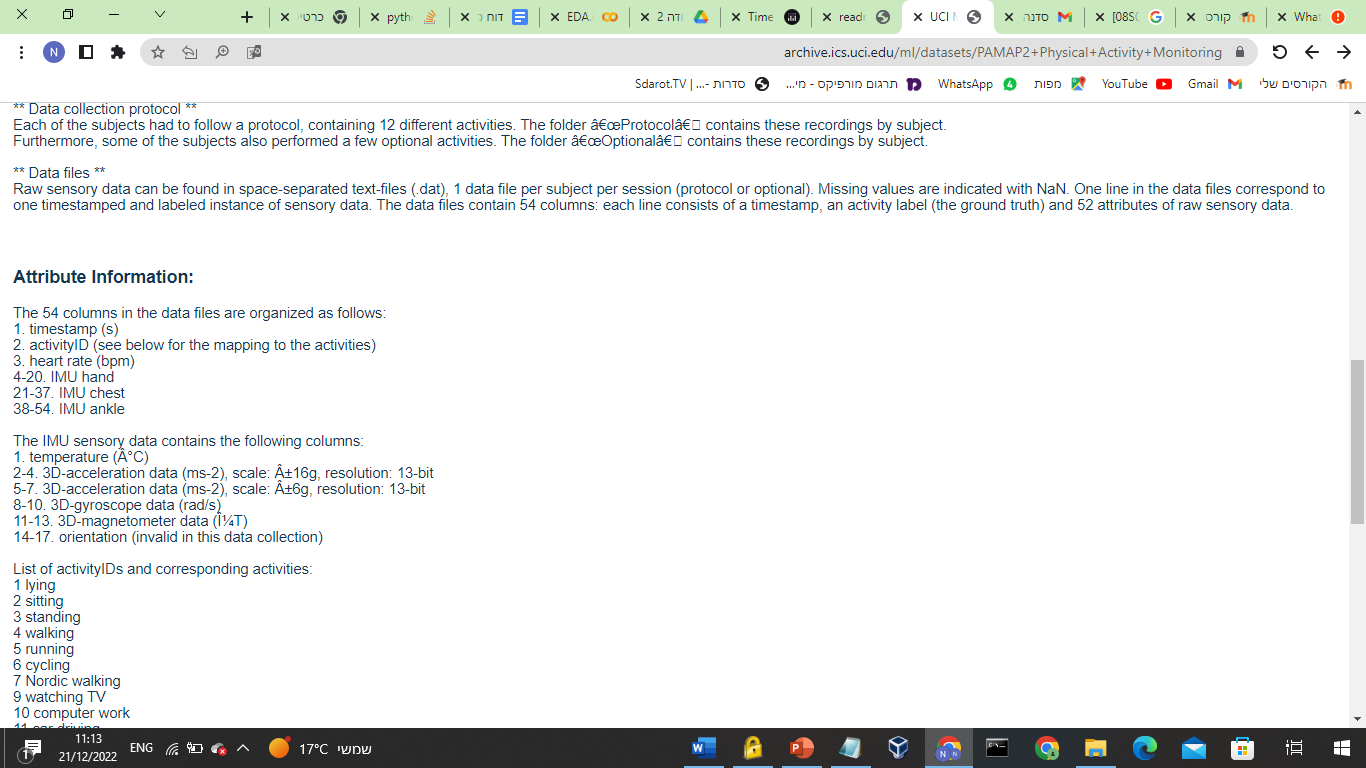
**EDA:**

First of all, let's try to see what kind of data we are looking at:



So our whole data set is made of 9 patients, and for each one of them, we have 54 columns that describe our data about the patient. In general, Our data is made of different IMU features, bpm, timestamp, and some activity. Our task is to determine what activity the patient is doing when given observation.

After loading the one of patients we can see that there are a lot of missing values in some of the features:

timestamp 0

activityID 0

bpm 342028

IMU hand1 1454

IMU hand2 1454

IMU hand3 1454

IMU hand4 1454

IMU hand5 1454

IMU hand6 1454

IMU hand7 1454

IMU hand8 1454

IMU hand9 1454

IMU hand10 1454

IMU hand11 1454

IMU hand12 1454

IMU hand13 1454

IMU hand14 1454

IMU hand15 1454

IMU hand16 1454

IMU hand17 1454

IMU chest1 509

IMU chest2 509

IMU chest3 509

IMU chest4 509

IMU chest5 509

IMU chest6 509

IMU chest7 509

IMU chest8 509

IMU chest9 509

IMU chest10 509

IMU chest11 509

IMU chest12 509

IMU chest13 509

IMU chest14 509

IMU chest15 509

IMU chest16 509

IMU chest17 509

IMU ancle1 1327

IMU ancle2 1327

IMU ancle3 1327

IMU ancle4 1327

IMU ancle5 1327

IMU ancle6 1327

IMU ancle7 1327

IMU ancle8 1327

IMU ancle9 1327

IMU ancle10 1327

IMU ancle11 1327

IMU ancle12 1327

IMU ancle13 1327

IMU ancle14 1327

IMU ancle15 1327

IMU ancle16 1327

IMU ancle17 1327

We can see that we handle a lot of missing data. In order to avoid removing missing data, we will use **imputation** to fill the missing gaps in our data. We can also see in the documentation of the dataset that features 14-17 are invalid data so we will remove those features. This leaves 41 features and 1 target.

We can look at the graphs and try to understand by changes of some parameters how the activityID changes in correlation to the IMUs sensors.





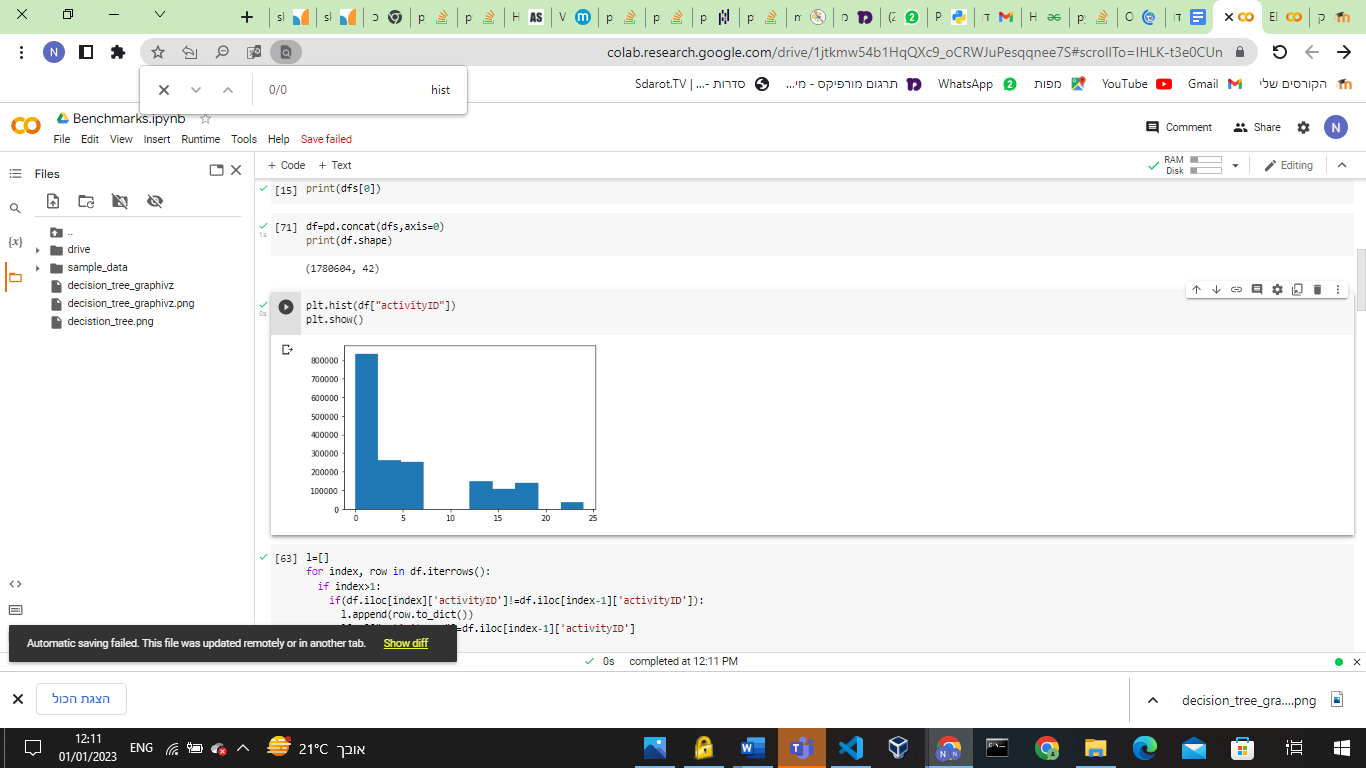
For example, we can see from the two graphs above, that By the IMU1 of hand, chest, and ankle we are not able to determine whether will be a change or not, but by the IMU5 We can see that after the descent there is a change in activityID from 0 to one so we might assume that after this kind of pattern in IMU5s, will be changing from 0 to 1 in activityID, our target. But we can't use this kind of method to extract real evaluations matrix, but only for the EDA and intuition.

Before we will implement models on the data here some points that we want to discuss:  
  
- Test set will contain patients 107 and 108, Validation set will be patient 109 and the rest will be in our Train data.

- Here some self-supervised tasks that we will like to try to pre-train our ML network:

1. what will be the % of the difference of specific feature/s given the other features (e.g given all the data (except activityID), predict the difference in % of every value in the chest IMUS vector).
2. Will the next value of features be the maximum value of the feature so far (e.g for the series [1,2,3,2,1]->[1,1,1,0,0]).
3. We will try to calculate the std of features given the other features (e.g given all the data (except activityID), predict the std of every value in the chest IMUS vector).

**Naive Benchmarks:**



The most Naive and intuitive way is to see the majority rule.

We can see that most of our activities are zero, so a naive classifier will classify most of the data as 0. The accuracy will be 589097/1780604=0.33 on training

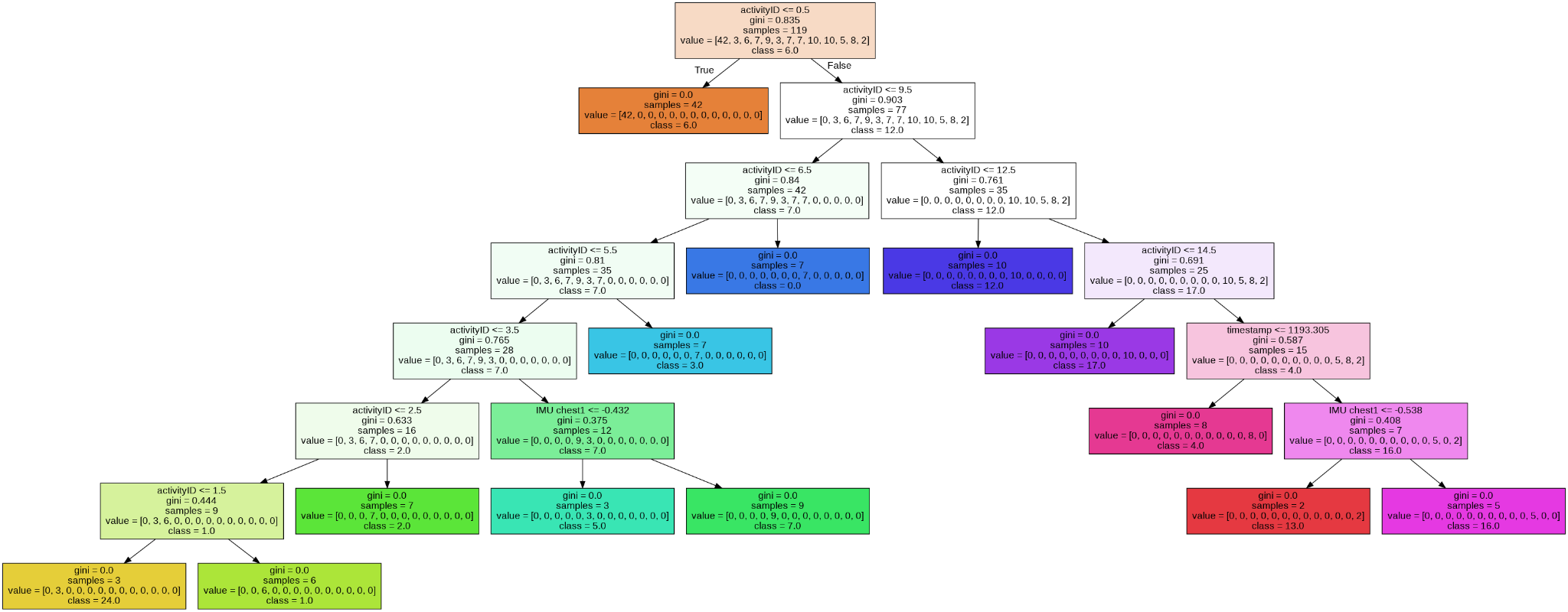
and 80822/313598=0.257 on the validation set.

We can improve our Naive classifier by extracting rules from our data.

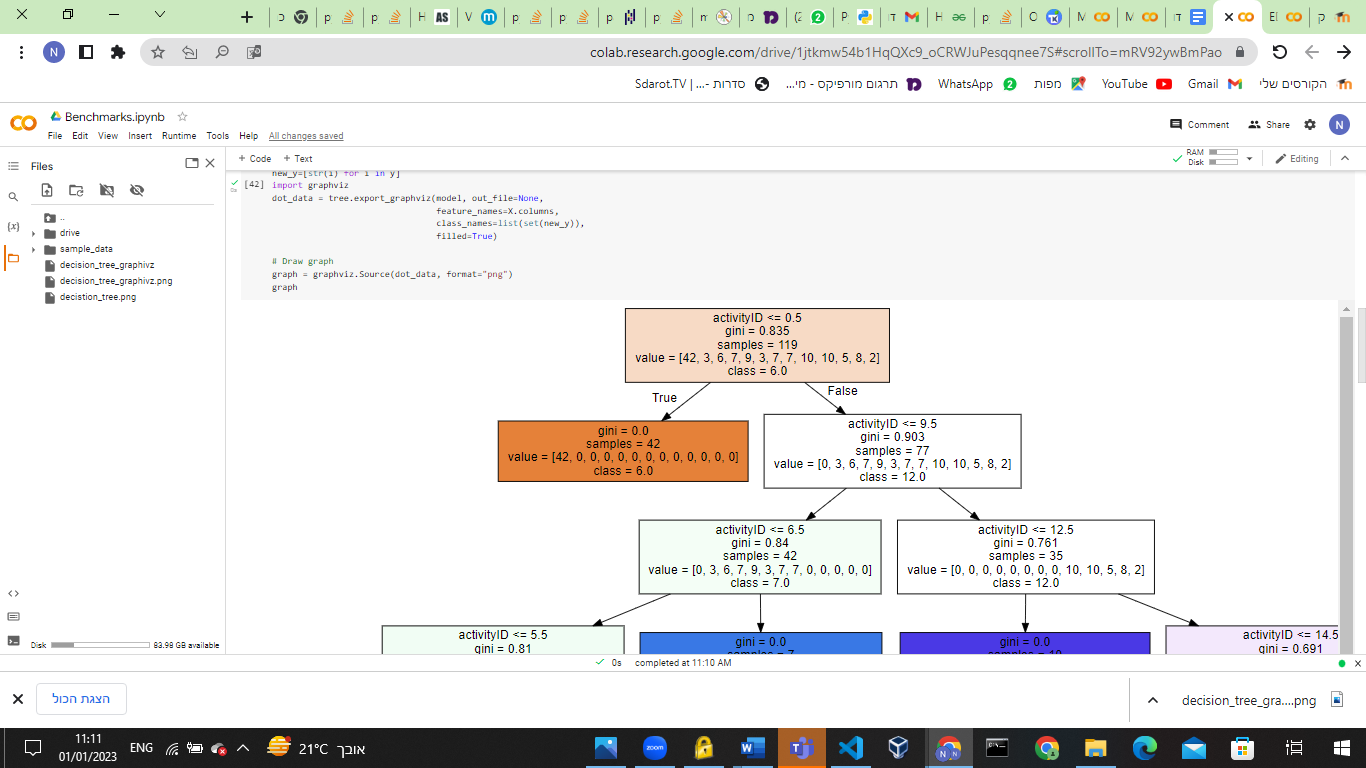
Let's try to extract rules from our dataset!

We will see what the values are before and after a change of activityID and create a tree out of it.

By this tree we can extract naive rules about the change between values:



for example, We can see that after activityID of 6, we are likely to have activityID of 6 or 12

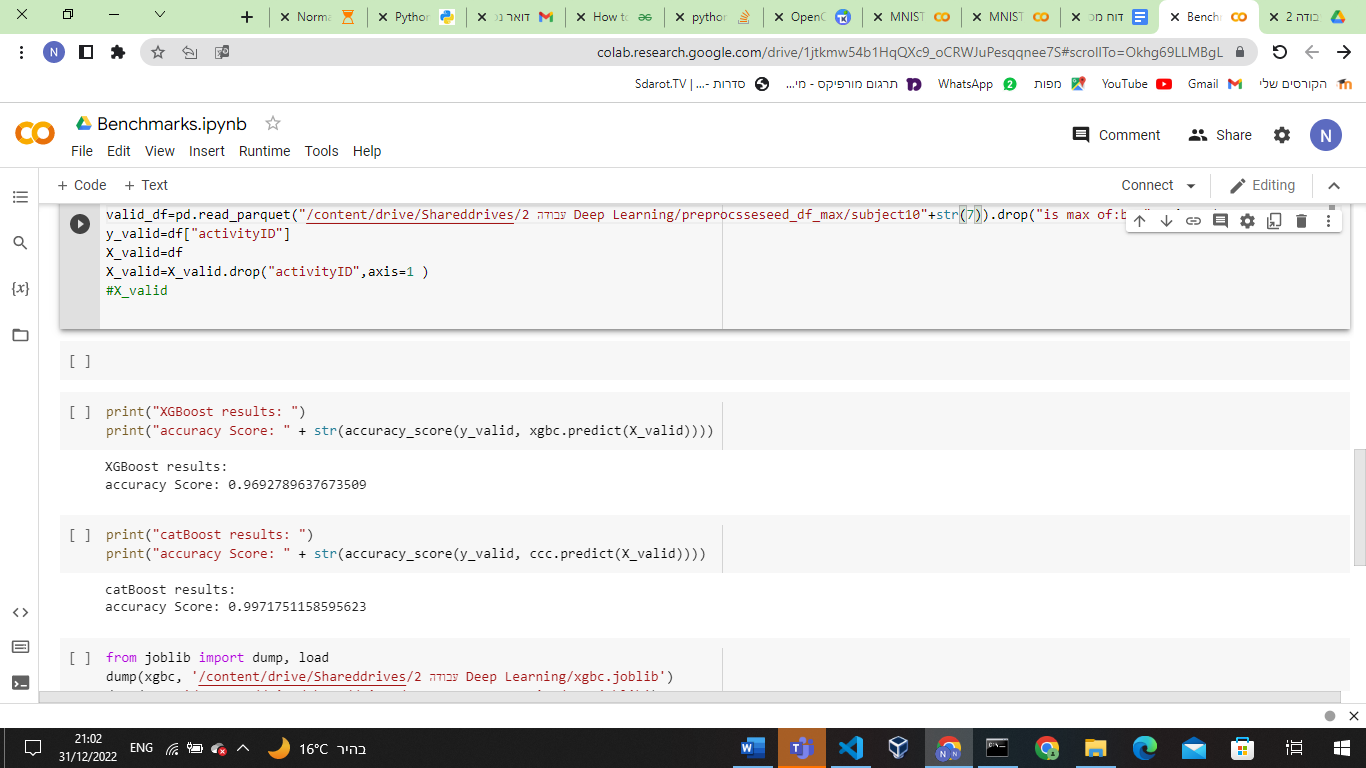


But this tree is only based on 119 samples wherever there is a change between activities. But we can assume that if there is no change so the activity is the same as it was.

The decision tree accuracy on validation is **0.475**.

**ML Benchmarks:**

By many Data scientists as written [here](https://m.facebook.com/groups/MDLI1/permalink/2326778337486201/?mibextid=Nif5oz), boosting algorithms are considered the state of the art on tabular datasets. So we will use catBoost and XGBoost as benchmarks on our datasets.



Here is an example of the accuracy score on the validation dataset (patient 7) of catBoost and XGBoost.

But, here we are not using a sliding window as input and output. so, let's assume that the input value is only one record and the output is a prediction for 10 records. We can see that in this way, the probability of being right with catBoost will be: 0.997\*\*10=0.97 while XGBoost 0.969\*\*10=0.72.

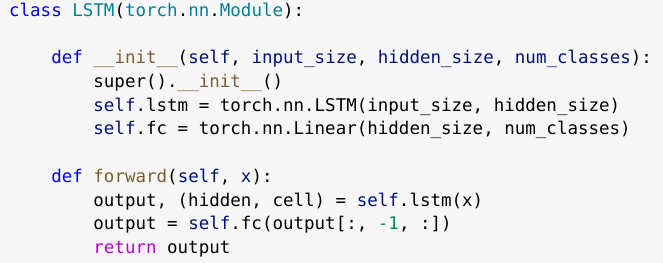
If we increase the sliding window even more, the accuracy will continue to decrease.

**RNN Benchmarks:**

In this section, we try to construct our own RNN network that will classify windows that contain 200 samples.

Each model that we will represent in the next section was trained with the following parameters:

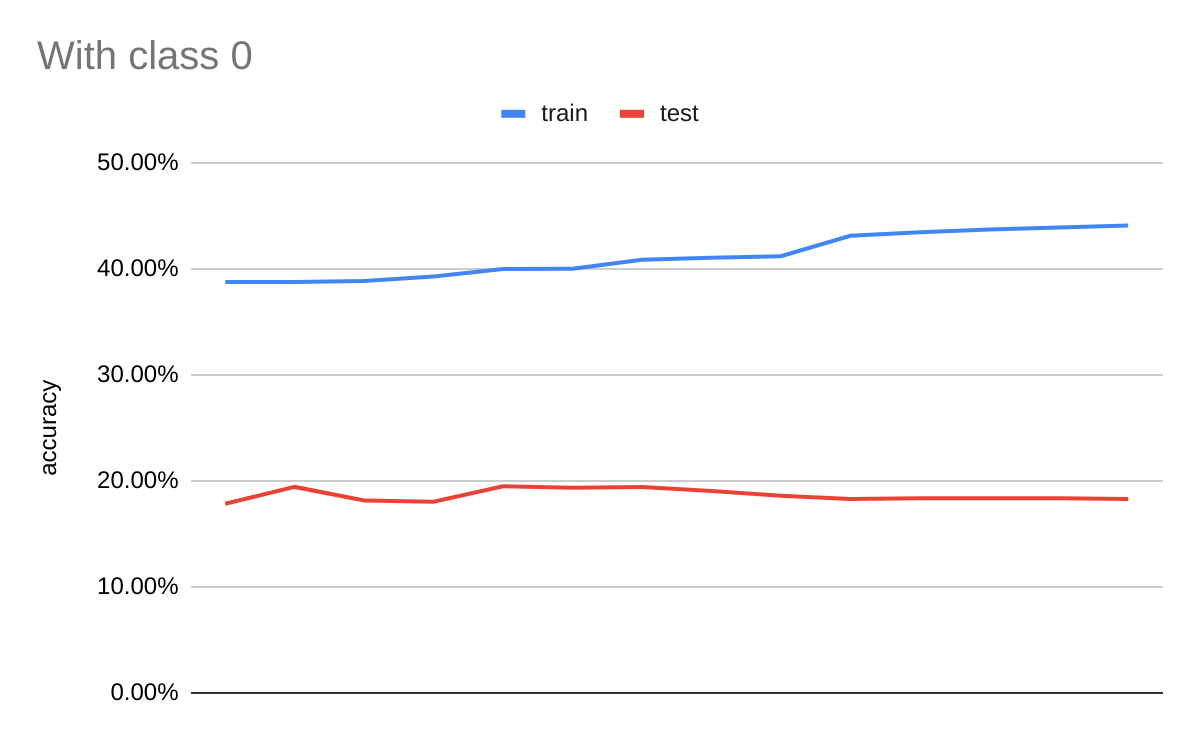
* The optimizer was Adam because when using fewer epochs than 100 it gets better accuracy than the SGD optimizer.
* The batch size is 32 because of two things - in lower batch size we can get better accuracy and the ram size of most GPUs can’t handle a big batch size of our data.

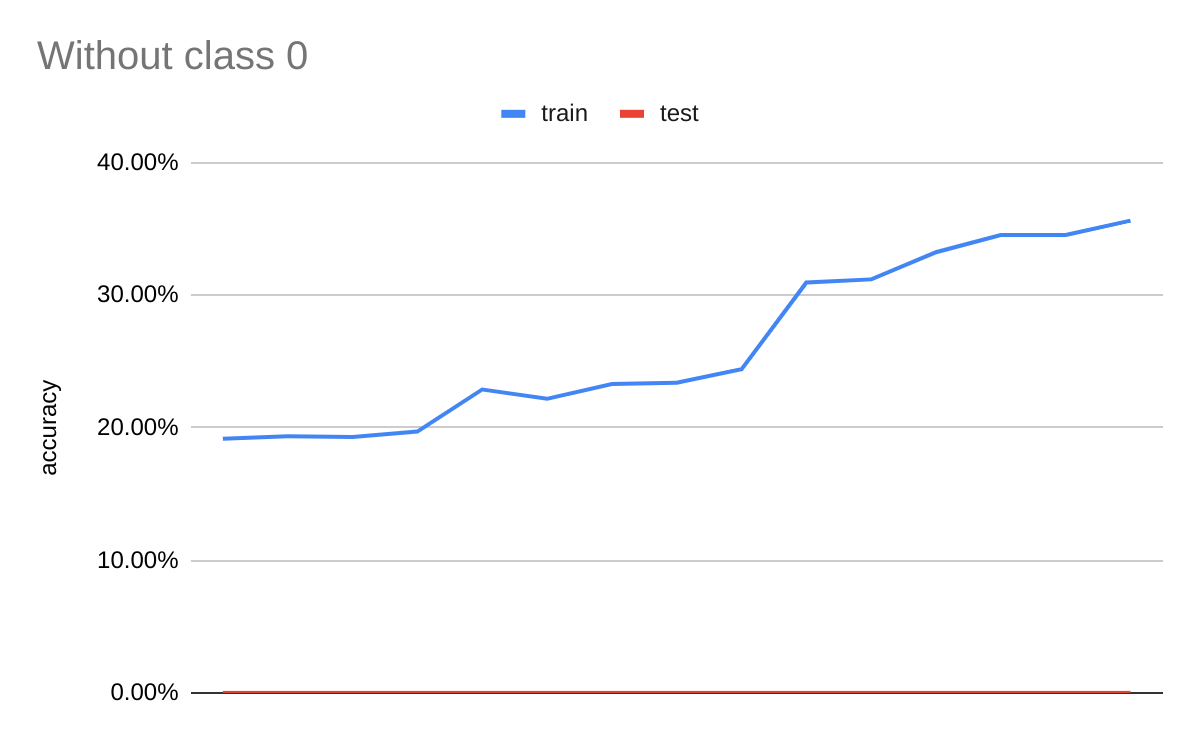




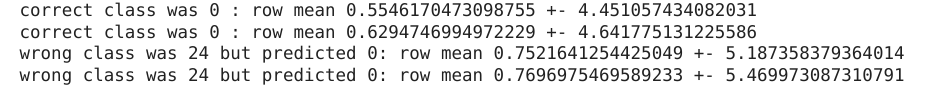
We tried to fit 2 models to the data - one that will get that data that contains the class 0 and another that will not.

With stable learning rate of 1e-4 and 15 epochs the result are:

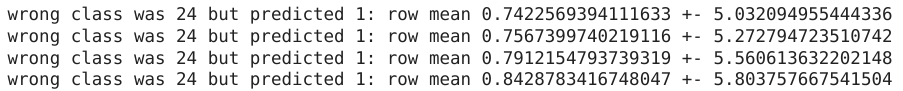




We can see that the model that got the class 0 started stable and could predict with an accuracy of ~20% the validation data. On the other hand, the model that didn’t get this class failed to predict any of the validation data. In total on the test data, we can see that the difference isn’t that huge with 41.79% accuracy for the first and 35.62% for the other.

First, let's analyze the model that got the class 0. We started to think about what can lead to that odd results and we could see that mean of the records may lead the model to predict differently:  
  


We can see that the records may have the ~same mean and can lead to the same class, so we think giving the model more time to learn may help it.

For the other model that didn’t get the class 0:  


we could see that he didn’t succeed in any class. That can be that choosing patient 109 as our validation led the model to try to predict the anomaly because it succeeded with 35.62% on the test set.

**Pre-Trained network:**

First, we have tried the second method but noticed the following distribution:

index of patient

Counter(0-># of no changes until point, 1->#, of maximums until point, -1 ->number of minimums until point )

0

Counter({0: 376330, 1: 59, -1: 27})

1

Counter({0: 446905, 1: 69, -1: 25})

2

Counter({0: 252761, 1: 57, -1: 14})

3

Counter({0: 329506, 1: 41, -1: 28})

4

Counter({0: 374689, 1: 70, -1: 23})

5

Counter({0: 361721, 1: 67, -1: 28})

6

Counter({0: 313519, 1: 55, -1: 24})

7

Counter({0: 407932, 1: 85, -1: 13})

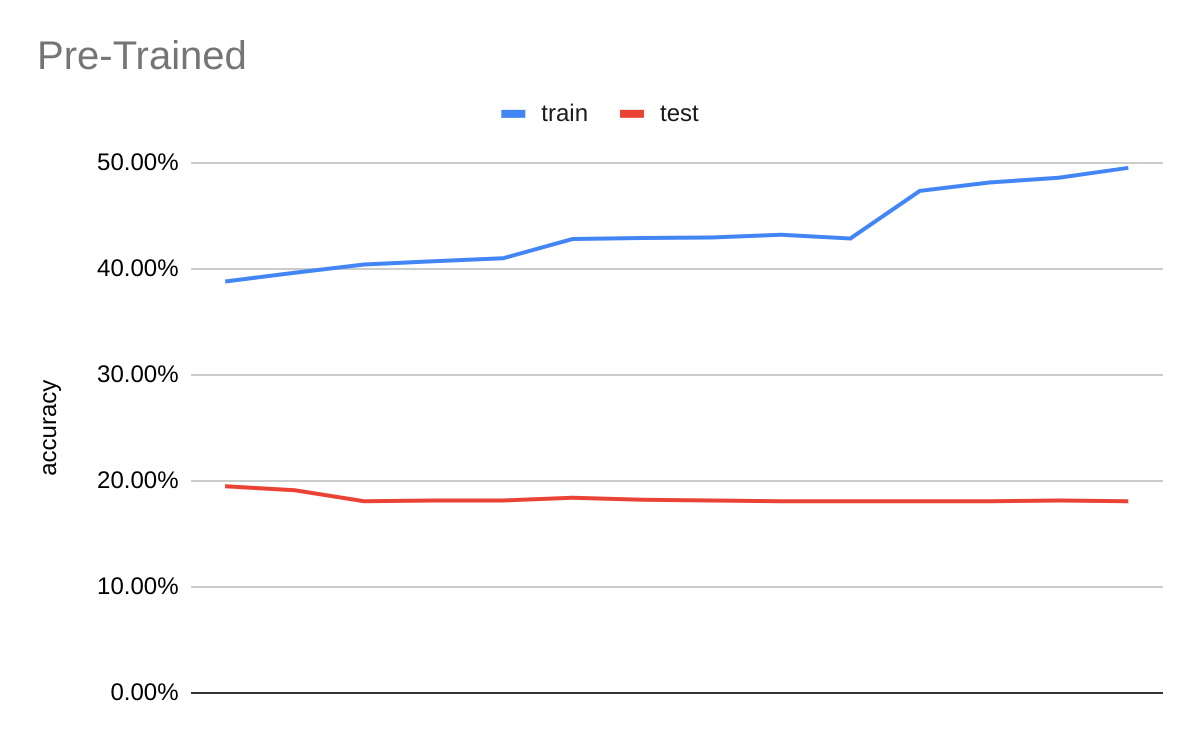
8

Counter({0: 8457, 1: 15, -1: 4})

We can easily see that there is a very low amount of 1,-1’s in the dataset so we decided to use the first metric in our research (% of 1,-1’s in the dataset 0.00245). which means that the most Naive model that gives an accuracy of 99.6% can give as an output only 0’s.

Then we tried to use the first method (compute the % of the change in other IMUS vectors) but a lot of the averages of a window were equal to 0. With our data, we were left with only 88 records of the train and that is a prediction so we choose to keep with the last method - calculate the average std of every value in the chest IMUS vector.

With stable learning rate of 1e-4 and 15 epochs on pre task and the task itself the result are:



We trained the pre-trained model on the real task with class 0 because it was the upper hand. We are not seeing any change in our validation data but a jump in train accuracy by 5.5%. In total on the test data, we can see that the model got 47.48% a 6% higher than the one that was not pre-trained. This can show that even basic pre-train can get us better results.

On the other hand we can see that the ML model catBoost beat both of the models on validation with 97% accuracy.

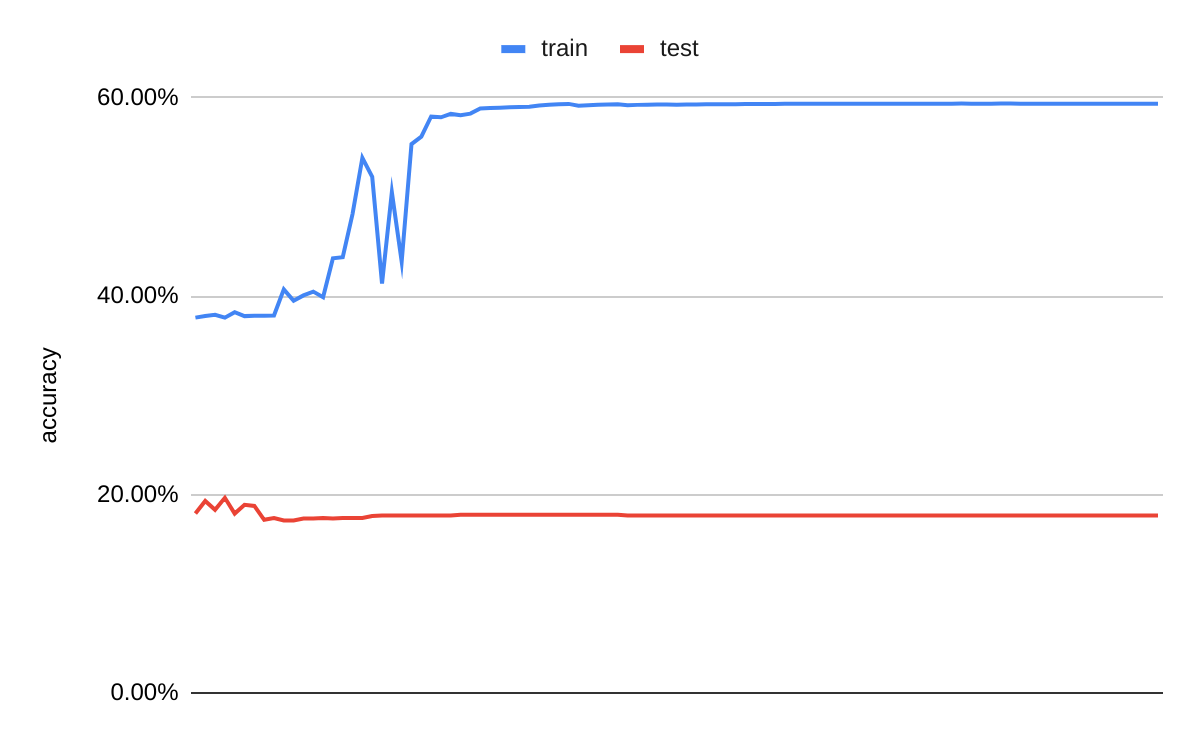
**Better RNN model:**

We saw that our first RNN model (with the class 0) did get a little better accuracy than the very basic benchmark model but we want to make the model better.

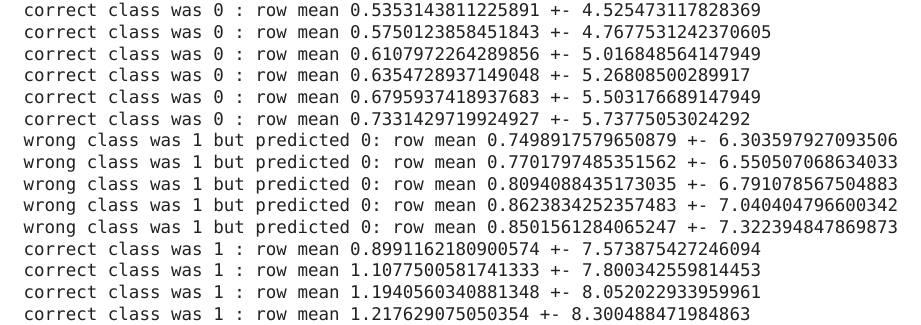
We think about these actions:

* our network architecture is not made well and simply for this task
* our learning rate is stable and not big enough at the start
* we need to train the model for a longer

So we increased the learning rate to 3e-4 and decrease it by half every 5 epochs, and train the net for 100 epochs now:



And we can see that we are getting better results. On the train set, we see a 50% jump in accuracy. Furthermore, we got a 25% jump in accuracy on the test set with a result of 56.64%.  
Even with this jump in performance, we can’t beat the accuracy of the ML model catBoost.

Let’s see if the model did generalize better than our basic RNN:  
  


We can see that we still have a range that we don’t know how to classify well but we are generalizing the problem better (and the accuracy says it).

In total we could learn that:

* Pre-Trained is beneficial to the model even if it seems useless.
* Longer runs can actually help the NN model to learn.
* We don’t need to rush to NN algorithms and we have “basic” other ML alternatives that can be better.

For next time - I think that using k-fold could help us with patient data like that, anomalies can impact the train badly, and choosing randomly isn't always a good option.