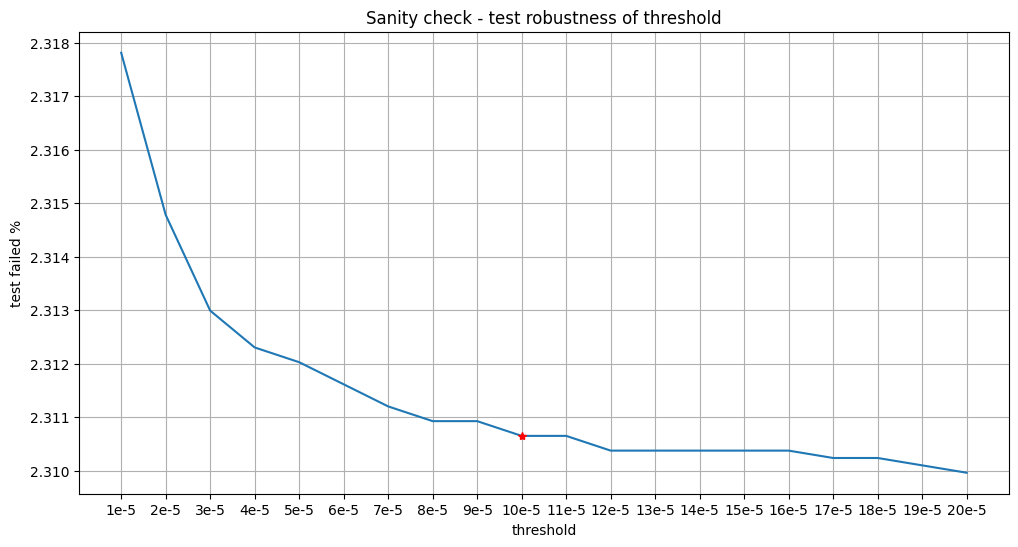
**Home Assignment**

The goal of this project is to predict the outcome of ‘TLJYWBE’ test.

Main assumption - we prefer a type II error (false negative) over a type I error (false positive) since it is more costly.

**Part 1 – Exploratory Data Analysis (EDA)**

The objective of this part is to clean the data, analyze it and prepare it as input to a prediction algorithm.

First, we observe the given threshold and close values for the target – and given the results we can see it is a robust threshold: 

Setting test past target label to 1 and test failed target label to 0 (treat this problem, given the robust threshold, as a classification problem) we can start with data cleaning:

We remove parts of the data that for sure won’t contribute to our final goal, such as duplicated rows, null columns, and columns containing large portion of the data with a single value.  
For single valued columns having 90% or more valid data (non-null), a considerable explanation for the 10% data missing is a network error of some kind, which enables us to assume all the values in the column should be the same, therefore we can drop it.

Next, we would like to drop columns missing a large portion of the data, but, in this case, given we only have 2.31% of false in our target, we can’t just drop columns with less than 10% of the data. We need to verify those columns are not good indicators of our target, therefore I checked the percentage of false values when looking at the non-null values for each of these columns. If it was similar to the overall target false percentage, and up to about twice the percentage of false values, the column was dropped. If the false value percentage was very high (about 4 times the original false value percentage) when examining the non-null value of one of these columns it was marked as important.

After removing columns with a small amount of data and keeping the ones with a higher false rate than the overall one, I checked for correlation within the important columns, if they are strongly correlated, we do not need them and can drop them as well.

Some further checks can be performed in this part, such as checking for correlation between binary columns (for example by using Jaccard score, or proportion agreement), nonlinear correlation, etc.

By this, we finished the basic data cleaning and can move forward to working based on column type. Given no characterization of the columns I separated them into the following categories:

* Indicators – single value columns
* Binary columns
* Categorial columns
* Numerical columns
* Non-numerical columns

For categorical columns, I set the bar at 200 categories.

Checking the number of unique values in each column gave almost the full list of numbers going up more than 500, but given no further knowledge about what each column represents, it is hard to determine an exact value at which we stop looking at it as categorical.

200 seems like a reasonable number of categories, for example, countries, target products, etc.

Determining what type of each column is I was able to start further analysis.

At this stage I had difficulty to keep analyzing the data using pandas (the kernel died due to a memory problem), I tried to work with the entire data (rows-wise) in SQL which also led to kernel crashing, hence I used a uniform sampling of the data and worked with 100,000 samples. Even with the resampling I had to use SQL to perform further analysis and prevent the kernel from crashing.

The next step is to remove outliers:

For numerical columns we can remove outliers – values greater than 75th percentile + 1.5IQR and values smaller than 25th percentile – 1.5IQR  
Another option for removing outliers is using z score, harder to perform in SQLite, as well as taking values greater than the mean + 3\*sigma.

These options can be optimized with a deeper knowledge of the data – based on the distribution of each column, etc.

Next, based on the column type I performed imputation – for numerical columns average, for categorical (and binary) the common.

In this stage, we could use learning to have better imputation.

From here we move on to the next part – Model testing and evaluation.

**Part 2 – Model Testing and Evaluation**

I chose to test 3 models – light GBM, CatBoost, and XGBoost, models which are known to deal with large data efficiently and yield good results on unbalanced data.

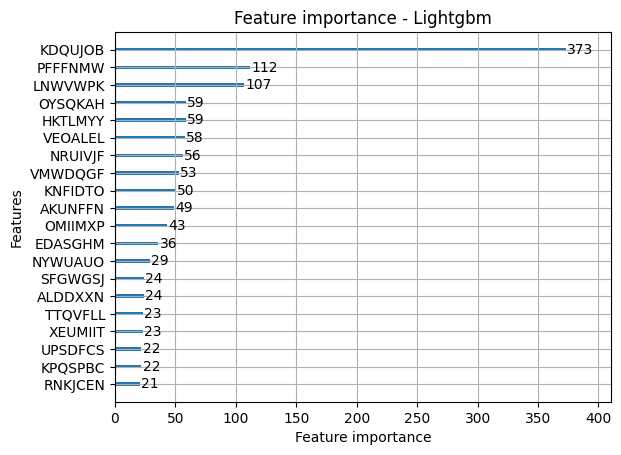
The data was splitted into 90% train and 10% test (90,000 samples and 10,000 accordingly).

Here are the results of the test set for each of the algorithms:



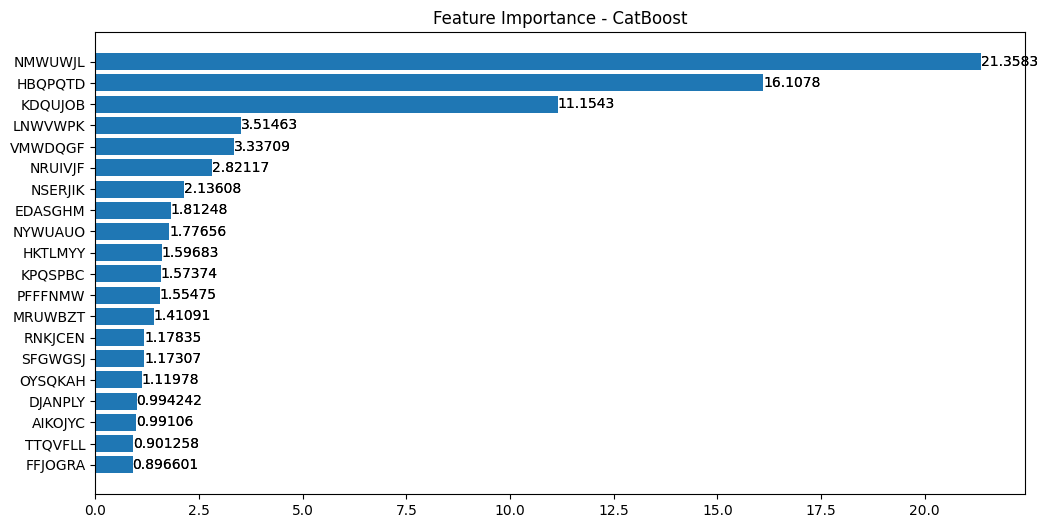
From this table we can deduce the best model for the job is CatBoost, since it has the lowest FPR and highest specificity, assuming we prefer to have type II error over type I error (as a business decision it makes more sense to have a flawless product be labeled as faulty than a faulty product labeled as flawless).

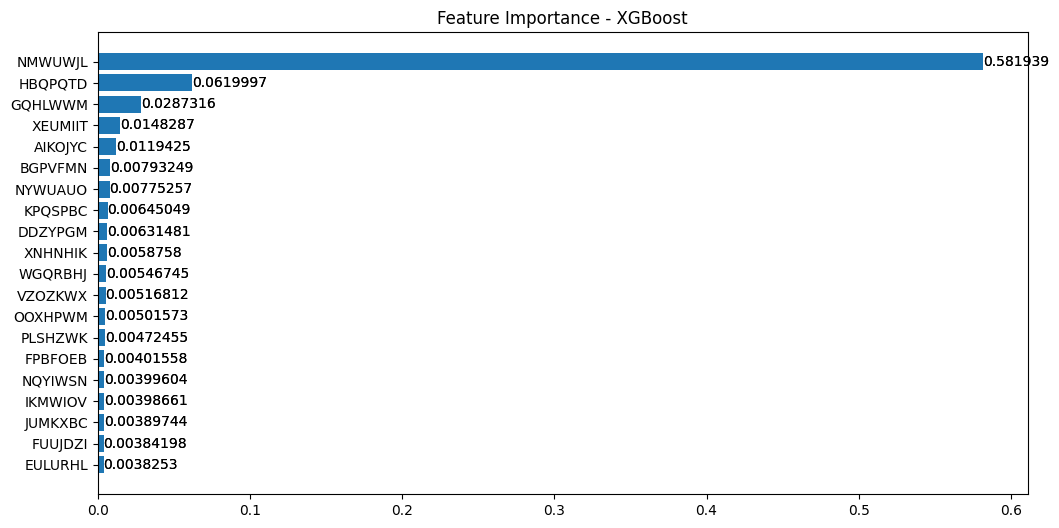
**Feature importance comparison**

****

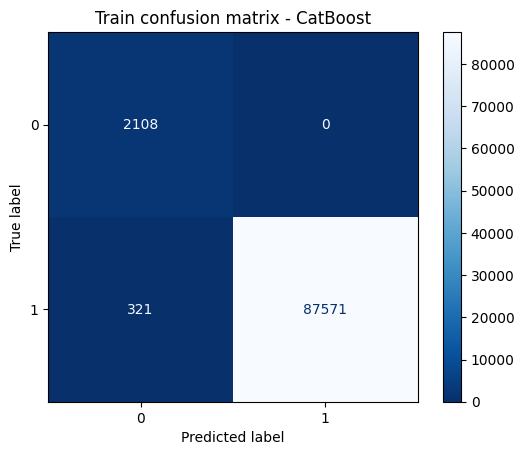
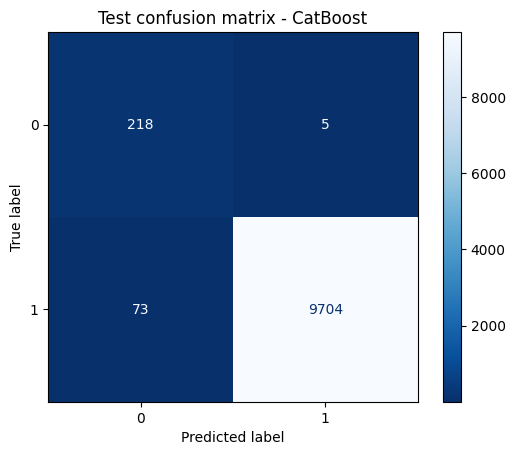
As we can see from the graphs below both CatBoost and XGBoost's first two important features are the same, while lightGBM uses different features as the most important ones.

A possible next step with this extra information we got from the models' important features would be to explore these features more thoroughly to see if we can get an even better model with fewer data passed to the model (reduce costs of data querying, etc)





**CatBoost results**

**** 

Those results were achieved by weighting the samples with a label of 1 with a 0.01 coefficient.

Before achieving those results, I tested model training with no weighting of the samples and some different coefficients for the positive label.

Due to a lack of resources (used my mac air), I tested those things manually instead of using automatic parameter tuning with cross-validation on the training set.

**Further remarks**

Another approach to handle the imbalanced data could be a weighted sampling of the data to get a higher percentage of false observations in the data passed to the model.

Furthermore, based on the small amount of false target percentage we could look at this problem as an anomaly detection problem – use an autoencoder to pass the data to a latent space (potentially less sparse due to lower dimensionality) and use clustering in the latent space.

Classical dimensionality reduction methods could be tested too, such as PCA, t-sne, etc.