CYBERTEC
DATA SCIENCE & POSTGRESQL

Using a predictive maintenance model of hard drive disks failure to back up data in a cost-effective way.

Kevin Speyer[1] and Hernesnizky

Juni 26, 2019

[1] Kevin.speyer@cybertec.at

Abstract

We use real data from the Self-Monitoring, Analysis and Reporting Technology (S.M.A.R.T) of Hard Drive Disks to train a machine learning model and predict the probability of failure of a HDD. After comparing various models, including Random Forest, Naive Bayes, Nearest Neighbors among others, we came up with a high accuracy (area under the ROC curve = 0.78 ± 0.04) model, which is fast to train and to predict with. The data is particularly challenging, given that there is a high imbalance between positive and negative classes (positive/negative < 10−4), and there are few positive cases. A dimensionality reduction method is developed to retain only the useful attributes and avoid unnecessary calculations, shortening the training and testing times.

We propose a smart implementation of the model to drastically reduce the number of HDD used to backup data, with a small increase in the risk of losing data. The model could be implemented with the help of PostgreSQL, to keep track of the S.M.A.R.T. metrics of the disks and trigger the necessary actions. Applying the model developed in this work, it is possible to lessen the number of backup disks by 50%, with a probability increase of only 9% to have a failure on non backed-up disks.

S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology) is a monitoring system included in computer hard disk drives (HDDs) and solid-state drives (SSDs). Its primary function is to measure and save various indicators of drive reliability with the intent of anticipating imminent hardware failures. Several attributes are evaluated to keep track of the state of the disks. Some of this include: power cycle count, temperature, distance the disk has shifted relative to the spindle, resistance caused by friction in mechanical parts while operating, etc. It is not evident which categories indicate a imminent failure of the HDD, and which ones are uncorrelated with the health status of the disks. Unfortunately, the S.M.A.R.T attributes are not comparable among different manufacturers, or even different models from the same manufacturer. Although there is a standard for this attributes, implementations by different manufacturers still differ and in some cases may lack some basic features or only include a few selected attributes. Another common inconvenience of the S.M.A.R.T. technology is the difficulty to access the data, which depends on the hardware compatibility with the HDD and the operating system used.

Typically, the data is organized in databases and PostgreSQL is becoming a very popular engine choice due to its advanced features and thanks to being an open source project. Postgres would be the natural choice to help implement this ML model, where the S.M.A.R.T. measurements can be stored, and it can easily interact with the python program developed in this work. To get the S.M.A.R.T. data out of the disks, a specific software must be used. For more information, see https://en.wikipedia.org/wiki/Comparison_of_S.M.A.R.T._tools.

In this work, we developed a statistical model from a large dataset to identify the disks that fail. The data was taken from: https://www.backblaze.com/b2/hard-drive-test-data.html, and the problem was inspired by a kaggle post: https://www.kaggle.com/backblaze/hard-drive-test-data. The data is 1.2GB, has 90 columns (attributes) for 3179296 instances. Aside from the S.M.A.R.T data, there is available information such as date, serial number, model, capacity, and of course failure status. A value of 1 in the field “failure status” means a failed disk, while 0 is a healthy disk.

To load the data from the comma separated file into a database, these are the steps to follow. First, connect to the running PostgreSQL server and then create the database where the data will live.

```
# CREATE DATABASE hdd_smart ;
CREATE DATABASE

# \connect hdd_smart ;
You are now connected to database "hdd_smart" as user "my_user".

# CREATE TABLE sma r t _me t r i c s

        (
        id serial NOT NULL ,
        date date ,
        serial_number varchar(100) ,
        model varchar(100) ,
        capacity_bytes numeric ,
        failure int ,
        smart_1_normalized int ,
        smart_1_raw numeric ,
        ...
        smart_255_raw numeric
        );

CREATE TABLE
```

Then we have to fill the table with the data in the file. This can be easily achieved using COPY from the database:

```
# COPY smart_metrics(date,serial_number,model,capacity_bytes,...)
FROM ' path/to/file.csv #
        DELIMITER ' , '
        CSV HEADER
        NULL AS 'NA' ;
COPY 3179295
```

The NULL AS 'NA' allows to import the missing values tagged with the string NA in the file, and convert them into NULL. Now the data is loaded into our Postgres database, and we can access it from within. For example, we ask which models have the most failed disks.

```
# SELECT distinct count( * ) as failed_count , model
FROM smart_metrics
WHERE failure=1
GROUP BY model ORDER BY failed_count DESC
limit 5 ;

failed_count | model
-------------------------------------
         139 | ST4000DM000
          15 | ST320LT007
          13 | Hi t a c h i HDS722020ALA330
           6 | WDC WD30EFRX
           6 | WDC WD800AAJS

(5 rows )
```

If we are already tracking the SMART data of the disks into the database, we need to load it into python to build the model first. The psycopg2 module is very useful to connect to the database and interact with it. The data can be fetched like this:

```
def read_disk_data( ) :
        " " " Read data, keep just 1 model and return
                2 sets diskriminated by label " " "
        import psycopg2
        import pandas as pd

        # Connect t o d a t a b a s e
        conn = psycopg2 . c o n n e c t ( f " h o s t ={ h o s t _ u r l } dbname={db_name}
        \
                " + f " user={my_user} password ={ passwd } " )

        # Get all data
                sql_query = f ' select_from { table_name } limit 10000 '
                all_data = pd . read_sql_query ( sql_query , conn )

        # Get most used model
                mos t_used_model = ( ( all_data . model . value _ counts ( )
        / all_data . shape [ 0 ] ) [ 0 : 1 ] ) . index [ 0 ]

        # Get data only for most used model
        ST400_data = all_data [ all_data . model == most_used_model ]
        neg_data = ST400_data [ ST400_data . failure == 0 ]
        pos_data = ST400_data [ ST400_data . failure == 1 ]

        return pos_data , neg_data
```

As mentioned before, the S.M.A.R.T features are not comparable between different models or manufacturers, so it is convenient to stay with just one model. We chose to analyze the model of HDD with the highest number of rows, in order to have the highest number of instances possible. After filtering the desired model, we discriminate the data by the failure label. Positive data is the one that is labeled as 1 in the “failure” field, and negative data is the one labeled with 0 in the same field.

Now we can explore the differences between both classes. First, we can check the ratio of instances that wehave in each class:

```
>>> pos_data . shape [ 0 ] / float ( neg_data . shape [ 0 ] )
8.27e-5
```

You don’t get more imbalanced classes than this! For every failed disk data, there are around _ 10, 000 instances of healthy HDD. To make the situation worse, there are only 146 failed disk instances to work with. It is evident that we will need to make some effort to get a highly accurate model from this dataset.

There is no way to know beforehand which attributes are relevant to the problem. Keeping all attributes just in case is a lazy strategy that will not pay off, as it will slow down calculations. The dimensionality of the problem may be unnecessarily high, and

we need to identify the features that behave differently between classes. There are a lot of possibilities on how to achieve this, but in this case we went with a classic: the t-test. This is a statistical test that gives the likelihood of two samples having the same mean value. A low value of this test (e.g.: $p < 0.05$) strongly suggests that both groups of data come from distributions with different mean values. By performing this kind of test for each attribute and dividing the groups by the failure label, we can identify the features that have a different mean value depending on the status of the disk. Welch's t-test is suitable for this cases, where there is no a priori information about the variance of the distributions of the attributes.

```
def get_relevant_att ( pos_data , neg_data , p_thr = 0.05 ) :
        " " " Returns a list with the relevant attributes
        T-test is performed with a significance of 0.05 " " "
        From scipy . stats import ttest_ind
        n_pos = pos_data . shape [ 0 ]
        n_neg = neg_data . shape [ 0 ]
        rel_att = [ ]
        for i_att in range ( 5 , pos_data . shape [ 1 ] -5 ) :
                if np .mod ( i_att , 2 ) == 0 : #skip RAW data
                    continue
                        p_val = ttest_ind ( neg_data . iloc [ : , i_att ] , \
                pos_data . iloc [ : , i_att ] , equal _var =False , \
                nan_policy = ' omit ' ) [ 1 ]
                if  p_ val < p_thr :
                        rel_att . append ( i_att )
        return rel_att
```

Another test that can extract the defining features of the data is the F-test, which can be used to check whether two groups of data have the same variance or not.

As the proverb says "A picture is worth a thousand words", and it can not be highlighted enough. It is always good practice to present data in nice colored plots. First, let's see the probability distribution functions of some of the relevant attributes, discriminated by class. In Figures 2 and 1, we present the probability distribution of two attributes named "smart_3_normalized" and "smart_9_normalized", for the positive (green "+" symbols) and the negative (red circles) classes. We can observe from these plots that statistically, the values adopted by this attributes are different depending on the status of the disk. For example, failed disks tend to have lower values of the feature "smart_3_normalized" than healthy disks, and can be used as an indicator of the disk's status.

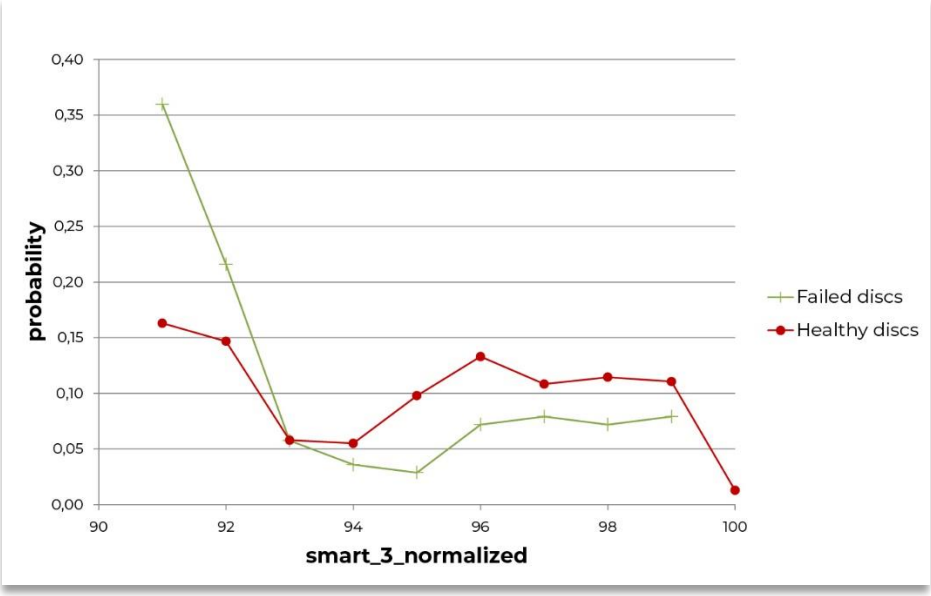

Figure 1: Probability distribution calculated for the attribute named "smart_3_normalized". The red circles correspond to the healthy disks, while the green plus symbols correspond to the failed disks. The healthy disks show, on average, different values than the failed disks.

After determining which attributes are relevant to the problem, we extract the columns that will be used to eventually fit a model. This can be done with a function like this one.

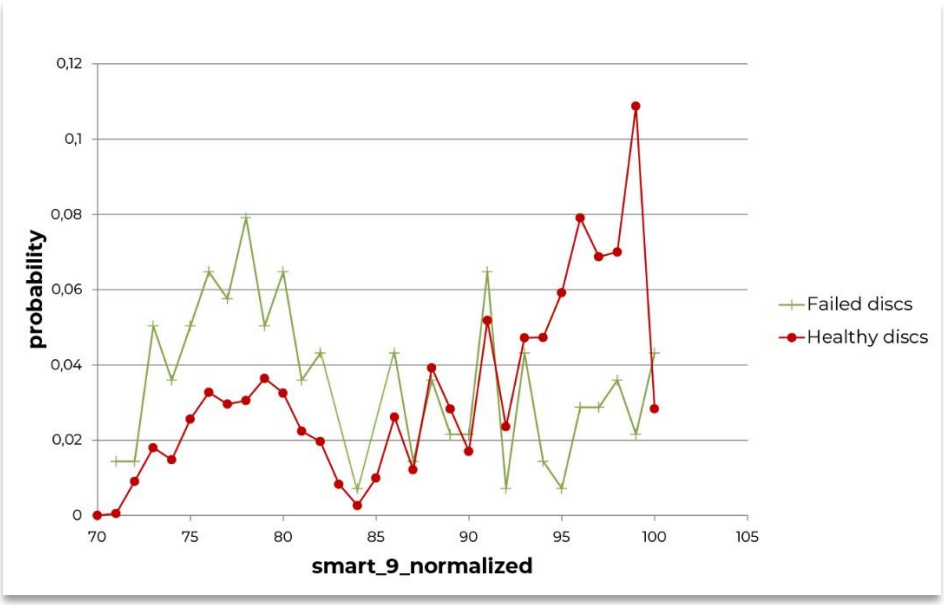

Figure 2: Probability density function calculated for an attribute named "smart_9_normalized". The red circles correspond to the healthy disks, while the green plus symbols correspond to the failed disks. The failed disks tend to have a lower value of this feature, and can be used as indicator of a risky HDD.

```
def get_rel_data ( pos_data , neg_data , rel_att ) :
        from sklearn . linear_model import LogisticRegression
        from sklearn import datasets
        from sklearn . preprocessing import StandardScaler

        pos_data_array = pos_data . iloc [ : , rel_att ] . values
        neg_data_array = neg_data . iloc [ : , rel_att ] . values

        n_neg_data = neg_data . shape [ 0 ]
        n_tot_data = n_neg_data + pos_data . shape [ 0 ]

        X_data = np . zeros ( [ n_tot_data , len (rel_att) ] )
        X_data [ 0 : n_neg_data , : ] = neg_data_array
        X_data [ n_neg_data : , : ] = pos_data_array
        Y_data = np . zeros ( n_tot_data )
        Y_data [ 0 : n _neg_data ] = 0
        Y_data [ n_neg_data : ] = 1

return X_data , Y_data
```

Another interesting visualization that allows to gain some insight into the problem is the attribute vs attribute plot, discriminated by the class. In Figure 3, we can see such a plot, were the attributes are named “smart_189_normalized” and “smart_7_normalized” and the negative data (healthy disks) is shown in red circles, while the positive data (failed disks) are plus symbols in green.

After playing around with the data we can proceed to searching for the best model to predict the status of HDD.We used the scikit-learn library for python, which facilitates the implementation of the most widely used models. Some of the models studied in this work are: Random Forest, AdaBoost, Logistic Regression, and Naive Bayes.

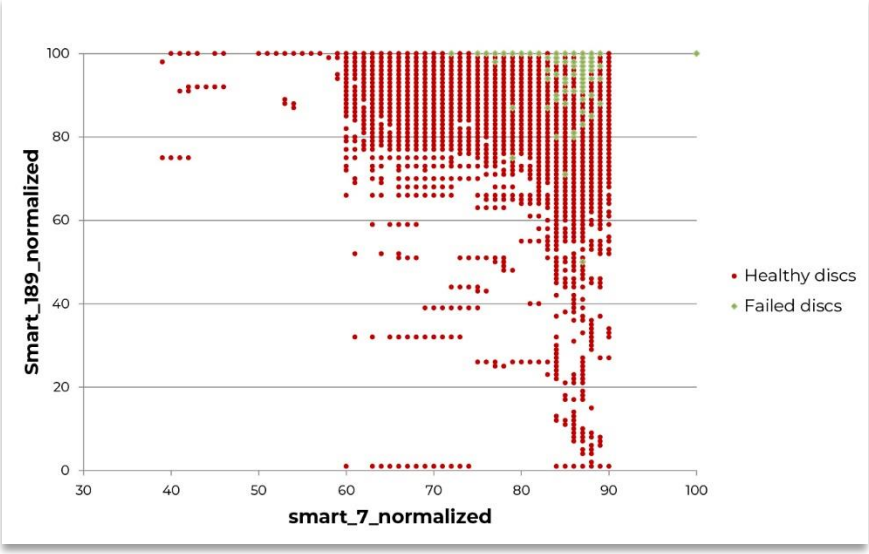

Figure 3: Data discriminated by fail status is plotted in an attribute vs attribute plot. These plots allow to see on plain sight correlation between features, and visualize which models are better suited for the problem. Red circles represent data of healthy disks and the green plus symbols correspond to failed disks.

Before comparing the models, it is important to look at a subject that is frequently ove looked: Model Selection. Some of the models mentioned have hyperparameters that affect the score of the model. It is a wise idea to test several of these hyperparameters to find an optimum and get the most of the model, but it is important to do it performing nested cross validation. To show an example, lets focus on AdaBoost, which has the learning rate as a hyperparameter. It is tempting to use the entire dataset to find an optimum value of the learning rate, but that would lead obviously to overfitting. The correct way to find the optimum value for the learning rate is to perform training/test partitions of the data for each value of the learning rate, and perform cross validation. In Figure 4, we present the Area Under the Curve (AUC) of the receiver operating characteristic (ROC) curve, with nested cross validation, as a function of the learning rate. The optimum value lies around 0.125. The lines are a linear fit of part of the data, and it does look strange, because the horizontal axis is in logarithmic scale.

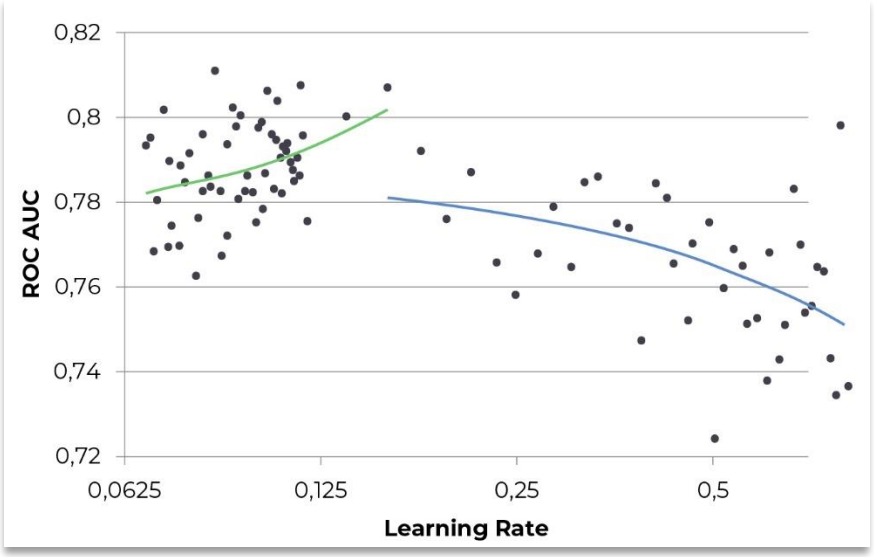

Figure 4: To find the optimum value of the hyperparameter Learning Rate of the AdaBoost model, we plot the Area Under the ROC curve as a function of this parameter. The green and blue lines are linear regressions of the data, for the data to the right and to the left of the maximum. Horizontal axis is in log-scale.

After fixing the hypermarameters of the models we can compare the models, by performing a new set of cross validations between models. For example, we can divide the data into 10 equally sized sets, and perform 10 tests with different training/testing values. It is important that we take new partitions, and not use the ones used to pick up the optimum hyperparameters of our models to avoid “leakage”. In Figure 5, we exhibit the ROC curve for 4 models trained in this work. It is possible to observe that there is no big separation between the score of the different models. In cases like this, other factors like the time taken to train the model or the speed to predict new data can tilt the decision towards one learning algorithm over another. In this case, logistic regression was chosen because it is very fast to train and to predict with, and has a AUC of 0.78 ± 0.04, which is in the same range of the best models used. To overcome the imbalance in the label distribution of the data for the logistic regression, we can use the option class_weight=’balanced’ when creating the model object from the LogisticRegression class of the scikit-learn module.

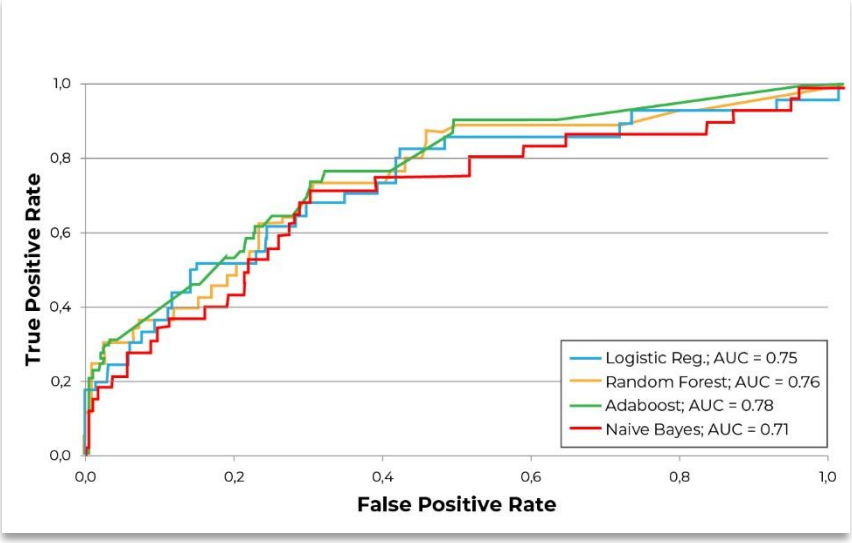

Figure 5: Receiver Operating Characteristic curve for some of the models trained in this work. The ROC curve is always evaluated in a test set, and not over the training data. The area under this curve (AUC) gives a good idea of the goodness of the fit.

This model could be implemented in databases to achieve a cost-effective intelligent backup system. For example, if we have 60 HDD with data, and we want a back up system to avoid data loss in an event of a disk failure, we would need in principle 120 disks (60 data disks + 60 backup disks). We can only lose data if a very improbable event takes place: two disks, containing the same data break in the same day. To keep the math simple, let's say that the probability of that happening is zero. Implementing the model developed in this
work, we could target the 50% of disks with the highest failure risk. This means that only 30 backup disks are needed. If you already have 60 backup disks, these could be used to store more data and expand the total memory of your system. The million dollar question now is: Using only half of the backup disks, what is the probability of losing data in a year? The answer is 8.6%. That is the probability that over the course of a year at least one HDD not backed up will fail. On the other hand, the probability that at least one HDD with backup will fail in the same year is 56%. This model could also be implemented to help in redundancy systems, which creates multiple copies of files across several hard drives. To sum it up, accepting a little risk it is possible to save large amounts of space designated to backup data.