THE UNIVERSITY OF ILLINOIS AT CHICAGO

UIC

IE 594: BIG DATA ANALYTICS FOR CPMS PREDICTIVE MAINTENANCE OF HARD DISK DRIVES

AMIT YADAV | DARSHAN MA | MAYUR MISHRA

DATE: 6th December 2017

Table of Contents

INTRODUCTION

PROBLEM DEFINITION

Hard disk drives(HDD) are one of the most common devices in storage systems, and most of the information in the world is being stored on hard drives. While the failure of a single hard drive might be rare, a system with thousands of hard drives will often experience failures and even simultaneous failures, resulting in service unavailability, and even permanent data loss. Therefore, reliability is one of the biggest concerns in storage systems.

According to Schroeder and Gibson, the mean time to failure of HDDs ranges from 1,000,000 to 1,500,000 h. A report given by Samsung also reveals that the mean time between failures of their HDDs is as high as 1,200,000 h, meaning that the annual failure rate is about 0.7% if an exponential distribution is assumed. On the other hand, the rapid evolution of HDD manufacturing techniques imposes a stringent time constraint on development of HDDs of the latest vintages. Typically, the time allowed for the production process ranges from 3 to 9 months. A production process includes concept development, design, pilot manufacturing and life testing. Therefore, the time allowed for the life test of a new design is often very short, despite the high reliability of the new vintage. How to carry out the life test within a reasonable period and how to analyze the test data have become important issues for HDD reliability engineers.

Given the importance of users' data, operators insist on high levels of reliability, and typically either store multiple copies of data throughout the system (increasing storage costs), or store data using erasure code (which consumes resources during reconstruction). On top of this, disk failure prediction is used to further increase reliability via automatic backing up of data on at-risk disks. However, for disk failure prediction to be useful, we require accurate predictions.

In this paper, we explore the ability of decision trees, K Nearest neighbors, Neural Networks and XGBoost techniques to predict hard drive failure based on SMART (Self-Monitoring, Analysis and Reporting Technology) attributes, which continuously reports attributes relating to drive reliability. Besides high prediction accuracy, it has the advantage of giving humanly understandable prediction results (i.e., interpretability). Users can pinpoint the most significant attributes correlated with drive failure by analyzing the output regulations of the tree. The SMART data includes the number of hours the drive has been running, the temperature of the drive, whether sectors have gone bad, and many more things.

The rest of the paper is organized as follows. Section 2 describes the objective of the paper followed by Section 3: the detailed discussion on the techniques used. Section 4 compares the results of all the techniques & Section 5 concludes the best method to predict the hard disk failure.

STATEMENT OF OBJECTIVES

The objective of this project is to explore the abilities of some of the best prediction models based on SMART attributes. We will be using Alteryx, Python and RapidMiner to process & analyze the data. The drives used are different models from **Hitachi** and data is recorded at Backblaze data center.

OVERVIEW OF THE HARD DRIVE DATA

For more than a year, Backblaze has released periodic updates on its hard drive failure rates, acquisition strategies, and its periodic refresh of its drive pod infrastructure. This is rare data, since virtually no companies release such information.

The data includes basic drive information along with the S.M.A.R.T. statistics reported by that drive. The daily snapshot of one drive is one record or row of data. All the drive snapshots for a given day are collected into a file consisting of a row for each active hard drive. The format of this file is a "csv" (Comma Separated Values) file

Each day, all the drive “snapshots” are processed and written to a new daily stats file. Each daily stats file has one row for every drive operational in the data center that day. For example, there are 365 daily stats files in the 2013 data package with each file containing a “snapshot” for each drive operational on any given day.

The first row of each file contains the column names, the remaining rows are the actual data. The columns are as follows:

- Date – The date of the file in yyyy-mm-dd format.
- Serial Number – The manufacturer-assigned serial number of the drive.
- Model – The manufacturer(Hitachi) assigned model number of the drive.
- Capacity – The drive capacity in bytes.
- Failure – Contains a “NOT FAIL” if the drive is OK. Contains a “FAIL” if this is the last day the drive was operational before failing.
- 2013 SMART Stats –Raw and Normalized values for 5 different SMART stats as reported by the given drive. Each value is the number reported by the drive.
- Temperature - The column gives the temperature of the drive in Degree Celsius.
- Duration - The run duration of the drive-in hours.
- RPM - Spinning speed of the hard drive in RPM (Revolutions per Minute)

Data Source: The data is downloaded from the official website of Backblaze, a data storage provider: **https://www.backblaze.com/b2/hard-drive-test-data.html**

DATA PREPROCESSING:

The data package contains the daily observations from 2013 Q2 to Q4. Each day observation is contained in a separate '.csv' file. Combining so many files together was a challenge. We used Alteryx, a tool for data blending and advanced data analytics to combine all the files together. The output was a file with 1 million observations. For the ease of computational time we reduced the observations to 16,528 data points using random sampling (1 in 170 chance for each record) feature in **Alteryx.** The sampling gave us a good mix of both 'FAIL' & 'NOT FAIL' data points.

Blank fields: Although the dataset contains 80 different SMART attributes, most of them were blank fields. Most drives do not report values for all SMART stats. Also, different drives may report different stats based on their model. We removed all the blank attributes and left with only 9 attributes namely: Capacity, smart_1_raw, smart_2_raw, smart_3_raw, smart_4_raw, smart_5_raw, Temperature, Duration & RPM. The target attribute being 'Failure'.

Missing Data: We checked for missing data in the file and found none. The dataset is complete.

DATA MINING TOOLS

Python

Python is a high-level programming language which is used for general purpose programming. The language reads like English, and takes off stress from people who are beginners in coding. Python is very flexible there are no defined rules to build features. The language comes with a set of predefined functions which would be difficult to implement from the user's end. Scripts can be written easily and tested. The standard library offers all kinds of implementation required for coding.

RapidMiner

RapidMiner is a software platform which is used for data preparation, machine learning, deep learning, text mining, data mining, and predictive analysis. This software is developed on a open core model. It is used for research, education, business, training, and application development. Machine Learning processes such as data preparation, visualization, model validation and optimization can be performed on this platform. This is one of the advanced analytical software used for data science research.

MODELS

Classification models are a good fit for predictive maintenance where we want to predict whether the hard drive has failed or not. We chose the following models and compared their performance for maximum accuracy

1. Decision tree: Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The result is a tree with decision nodes and leaf nodes. A decision node has two or more branches. Leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor is called root node.

Tool Used: RAPIDMINER

Decision Tree Parameters	
Maximal Depth	20
Criterion	Gain_Ratio
Confidence	0.25
Minimal Gain	0.1
Minimal Leaf Size	2
Apply Pruning	Yes
Minimal Leaf Size for Split	4

2. Random Forest: Random Forest operates by creating multiple of decision trees during training and outputting the class that is, node of the classes or mean prediction of the individual trees. Each tree grows to the largest extent possible without pruning. This method is efficient on large data sets. Capabilities can be extended to unlabeled data, supervised clustering, data views and outlier detection. Arbitrary Forest shortcomings are that when utilized for relapse they can't anticipate past the range in the preparation information, and that they may over-fit informational indexes that are especially boisterous.

Tool Used: RAPIDMINER

Random Forest Parameters	
Number of Trees	20
Criterion	Gain_Ratio
Maximal Depth	20
Confidence	0.25
Minimal Gain	0.1
Minimal Leaf Size	2

3. Feed-Forward Neural Network: Neural Network is a powerful data-driven, self-adaptive & flexible computational tool. An NN is based on a collection of connected units called artificial neurons. Each connection between neurons can transmit a signal to another neuron. The receiving neuron can process the signal(s) and then signal downstream neurons connected to it. Neurons and the connections may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream.

Tool Used: PYTHON

Neural Network Parameters:

In [25]:
```
model = Sequential()
model.add(Dense(64, input_dim=8, kernel_initializer='normal', activation='relu'))
model.add(Dense(32, kernel_initializer='normal', activation='relu'))
model.add(Dense(1, kernel_initializer='normal'))
model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])
```

In [28]:
```
print(model.summary())
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 64)                576
_________________________________________________________________
dense_5 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 33
=================================================================
Total params: 2,689
Trainable params: 2,689
Non-trainable params: 0
_________________________________________________________________
None
```

In [29]:
```
kfold = KFold(n_splits=10, shuffle=True, random_state=123)
cvresults = []
for train, test in kfold.split(X, y):
    model.fit(X[train], y[train], epochs=100, batch_size=30, verbose=1)
    results = model.evaluate(X[test], y[test], verbose=1)
    cvresults.append(results)
```

4. XGBoost: XGBoost is short for 'Extreme Gradient Boosting. This is an open source software library which provides boosting framework for many programming languages including python. XGBoost aims to provide scalable, portable, and Distributed gradient boosting. XGBoost has a speed advantage over sklearn. This has been to be very reliable for a prediction model. The XGBoost model is used for supervised learning problems where the data is trained to predict a target value. We used Hyperparameter tuning as discussed below to find the best hyperparameters for our XGBoost model.

Tool Used: RAPIDMINER

XGBoost Hyperparameters	
Maximum Depth	6
Minimum Child Weight	5
Subsample	0.8
Colsample	0.8
Learning Rate	0.3
No. of Estimators	300

HYPERPARAMTER TUNING

Hyperparameter tuning works by running multiple *trials* in a single training job. Each trial is a complete execution of your training application with values for your chosen hyperparameters set within limits you specify. Hyperparameter tuning optimizes a single target variable (also called the hyperparameter metric) that you specify. The accuracy of the model, as calculated from an evaluation pass, is a common metric. The metric must be a numeric value, and you can specify whether you want to tune your model to maximize or minimize your metric. When you start a job with hyperparameter tuning, you establish the name of your hyperparameter metric. This is the name you assign to the scalar summary that you add to your trainer.

The most popular approach is **Grid Search,** and this is what we used for our XGBoost model.

In grid search, we try a set of configurations of hyperparameters and train the algorithm accordingly, choosing the hyperparameter configuration that gives the best performance. In practice, we specify the bounds and steps between values of the hyperparameters, so that it forms a *grid* of configurations. We typically start with a limited grid with relatively large steps between parameter values, then extend or make the grid finer at the best configuration and continue searching on the new grid.

The final hyperparameters that we used for our model are as follows:

In [66]:
```
params = { 'max_depth':6,'min_child_weight': 1, 'learning_rate', 'subsample': 1,
          'colsample_bytree': 1, 'objective':'reg:linear'} # Parameters that we are going to tune

params['eval_metric'] = "log_loss"

gridsearch_params = [
    (max_depth, min_child_weight)
    for max_depth in range(4,18)
    for min_child_weight in range(5,8)]

gridsearch_params = [
    (subsample, colsample)
    for subsample in [i/10. for i in range(7,11)]
    for colsample in [i/10. for i in range(7,11)]]
```

RESULTS:

Neural Network

Neural Network with one input, hidden and output layer each was very efficient in classifying our label for both the classes. We ran the model for 100 epochs as our loss function stopped reducing after about 50 epochs.

Accuracy: 95.51 %

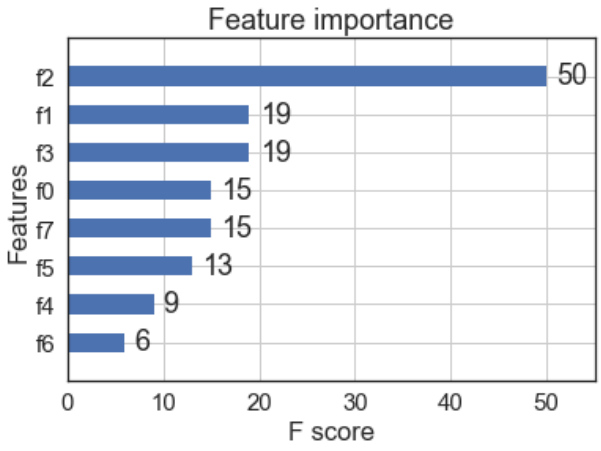
```
[[11252     1]
[  510    18]]
                  precision    recall  f1-score   support

    Class 1(FAIL)       0.96      1.00      0.98     11253
Class 0(NOT FAIL)       0.95      0.03      0.07       528

      avg / total       0.96      0.96      0.94     11781
```

XGBoost

XGBoost with hyperparameter tuning gave us the best accuracy for the classifying our label. We tuned it to get the best parameters with loss function being logarithmic which penalizes the false classifications.




```
Classification Report:
             precision    recall  f1-score   support

          0       0.96      1.00      0.98     11253
          1       0.80      0.19      0.31       528

avg / total       0.96      0.96      0.95     11781

Confusion Matrix:
 [[11228    25]
 [  426   102]]

Accuracy 0.961718020542
```

Accuracy: 96.17%

Decision Trees

Decision Tree is generally a good model for classification but in our case, it performed less accurately than the other two.

We got an **Accuracy of 84.72%** the decision criterion being gain ratio. We even varied the depth of the tree to check for all cases but accuracy was always in the 70-80 range. Thus, this model is a poor fit for our project.

Random Forest

Random Forest also is a generally good model for classification for categorical values. We even tried increasing the number of trees as the classifier won't overfit the model but still ended up with an **Accuracy of 86.38%**

MODEL	ACCURACY
Decision Tree	84.72%
Random Forest	86.38%
XGBoost	96.17%
Neural Network	95.51%

CONCLUSION:

Thus, XGBoost and Neural Network performed the best for our project with an accuracy more than 95%. This will help the hard disk manufactures to improve the features which contribute more towards the failure rate. This approach promises cost savings over routine or time-based preventive maintenance.

REFERENCES

[1] L. Breiman. *Random forests. Maching Learning*, 45(1):5{32, Oct. 2001.

[2] Tianqi Chen, Carlos Guestrin, XGBoost: *A Scalable Tree Boosting System*

[3] Tim P. Vogels, Kanaka Rajan, and L.F. Abbott, *Neural Network Dynamics*, Vol. 28:357-376

[4] KenjiDoya, DeLiangWang, *Fostering deep learning and beyond*, Volume 97, January 2018

[5] Jürgen Schmidhuber, Volume 61, *Deep learning in neural networks: An overview* January 2015

[6] JingLi Rebecca, J.Stones, GangWang, XiaoguangLiu, ZhongweiLi, MingXu, *Hard drive failure prediction using Decision Trees*, Reliability Engineering & System Safety Volume 164, August 2017

APPENDIX

PYTHON CODE FOR NEURAL NETWORK

```
import numpy as np

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler

from sklearn.cross_validation import train_test_split

import seaborn as sns

import matplotlib.pyplot as plt

df = pd.read_csv(r'C:\Users\Mayur\Downloads\Modified.csv')

df.head()

df_norm = df[['smart_1_raw','smart_5_raw','smart_9_raw', 'smart_194_raw',

'smart_197_raw','Temp','Duration','RPM']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))

df_norm.sample(n=5)

target = df[['failure']].replace(['NOT FAIL','FAIL'],[0,1])

target.sample(n=5)
```

```
df1 = pd.concat([df_norm, target], axis=1)

df1.sample(n=5)

X = df1.drop(['failure'], axis=1).values

y = df1['failure'].values

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=123)

model = Sequential()

model.add(Dense(64, input_dim=8, kernel_initializer='normal', activation='relu'))

model.add(Dense(32, kernel_initializer='normal', activation='relu'))

model.add(Dense(1, kernel_initializer='normal'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

print(model.summary())

model.fit(x_train, y_train, epochs=10, batch_size=50)

results = model.evaluate(x_test, y_test, verbose=1)

model.evaluate(x_test, y_test, verbose=1)

kfold = KFold(n_splits=3, shuffle=True, random_state=123)

cvresults = []

for train, test in kfold.split(X, y):

    model.fit(X[train], y[train], epochs=10, batch_size=30, verbose=1)

    results = model.evaluate(X[test], y[test], verbose=1)
```

```
    cvresults.append(results)

from sklearn.metrics import classification_report,confusion_matrix

y_pred = model.predict_classes(x_test)

target_names= ['Class 1(FAIL)', 'Class 0(NOT FAIL)']

print("Confusion Matrix: \n",confusion_matrix(y_test,y_pred))

print(classification_report(y_test, y_pred,target_names=target_names))



x = [i for i in range(len(y_test))]

plt.scatter(x, y_test, label='test', alpha=1)

plt.scatter(x, y_pred, label='test predictions', color='red', alpha=0.5)

plt.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3,

          ncol=2, mode="expand", borderaxespad=0.4)

plt.figure(figsize=(16, 16), dpi=80)

plt.show()
```

PYTHON CODE FOR XGBoost

```
import numpy as np

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from sklearn.model_selection import cross_val_score

from sklearn.model_selection import KFold

from sklearn.preprocessing import MinMaxScaler

from sklearn.cross_validation import train_test_split

import seaborn as sns

import matplotlib.pyplot as plt

from keras.layers.core import Flatten

df = pd.read_csv('Modified-Big data.csv')

df.head()



df_norm = df[['smart_1_raw','smart_5_raw','smart_9_raw', 'smart_194_raw',

'smart_197_raw','Temp','Duration','RPM']].apply(lambda x: (x - x.min()) / (x.max() - x.min()))

df_norm.sample(n=5)
```

```
target = df[['failure']].replace(['NOT FAIL','FAIL'],[0,1])

target.sample(n=5)



df1 = pd.concat([df_norm, target], axis=1)

df1.sample(n=5)



X = df1.drop(['failure'], axis=1).values

y = df1['failure'].values

x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.7, random_state=123)

model = Sequential()

model.add(Dense(32, input_dim=8, kernel_initializer='normal', activation='relu'))

model.add(Dense(16, kernel_initializer='normal', activation='relu'))

model.add(Dense(1, kernel_initializer='normal'))

model.compile(loss='binary_crossentropy', optimizer='adam',metrics=['accuracy'])

print(model.summary())

model.fit(x_train, y_train, epochs=, batch_size=50)

results = model.evaluate(x_test, y_test, verbose=1)

kfold = KFold(n_splits=3, shuffle=True, random_state=123)

cvresults = []
```

```
for train, test in kfold.split(X, y):
    model.fit(X[train], y[train], epochs=10, batch_size=30, verbose=1)
    results = model.evaluate(X[test], y[test], verbose=1)
    cvresults.append(results)
results = model.evaluate(x_test, y_test, verbose=1)


from sklearn.metrics import classification_report,confusion_matrix
y_pred = model.predict_classes(x_test)
target_names= ['Class 1(FAIL)', 'Class 0(NOT FAIL)']
print("\n",confusion_matrix(y_test,y_pred))
print(classification_report(y_test, y_pred,target_names=target_names))
```