Machine Learning in Predicting Hard Drive Failure

Written by CS3244 Group 20:
Benedict Halim (A0216056L), Han Wei Lun (A0216311X), Lim Jun Heng, Edward (A0222728E), Quan Teng Foong (A0223929X), Wong Zhi Heng Zac (A0216174J)

1 Introduction

With our increasing dependence on technology, storage of data has become increasingly important. Individuals store their data on personal hard drives and corporations utilize large data centers for data storage, most of which are implemented with hard drives. It can thus be said that our world is now very dependent on hard drives. However, hard drives are not free from fault and nowadays with the average personal hard drive holding 1-2 TB of data, the failure of a single drive can lead to catastrophic outcomes.

One common way to tackle this problem is by using RAID (Redundant Array of Independent Disks) configurations to ensure all data is backed up in multiple drives in the event of one single drive's failure. This solution, while useful and widely adopted, comes with its own set of caveats. For one, the large amount of redundancy means a waste of valuable storage space that could otherwise be used to store even more information. Additionally, this solution depends highly on the low probability of all drives failing at once, which while unlikely, is still a possibility. This is a costly solution for many individuals. With a vast majority of Singaporean citizens owning hard drives which store essential data, our application aims to provide them an alternative layer of security in protecting them from a loss of data, by warning them of potential hard drive failure.

Our group would like to propose a Machine Learning (ML) approach to tackle the issue of failure of SMART drives (Self-Monitoring, Analysis and Reporting Technology) (En 2022). Nowadays, most hard drives are SMART drives and SMART serves as a method for detecting and reporting indicators of drive reliability.

1.1 Problem Statement

In our project, we explored the idea of predictive maintenance for SMART hard drives. Our goal is to utilize an ML model to make accurate predictions to answer the question: "Is my hard drive expected to fail in the next 7 days?".

With the prediction results, we built an application capable of warning users in advance if their hard drive is expected to fail in a week, as well as providing further details on potential causes for the failure.

1.2 Dataset

We used the Backblaze hard drive dataset (Backblaze 2022) to train our ML models. Backblaze has been publishing statistics and insights on hard drives in their data centers since 2013. After cleaning, the dataset contained 32,000 rows and 24 input attributes. Although the dataset is not from Singapore, the results can be applied to Singapore's context as hard drives all share the same SMART attributes.

Our data is highly imbalanced as hard drive failures are uncommon. Having an imbalanced dataset affected the training of most of our models and most models completely ignored the minority class and always predicted hard drives to not fail. To tackle this problem, we combined the techniques of oversampling and undersampling. This technique applies a modest amount of oversampling to the minority class, which improves the bias towards it, at the same time, the majority class experiences a modest amount of undersampling which reduces the bias on it. This reduces the effect of overfitting by oversampling and loss of information by undersampling.

We split our dataset into Train (70%), Validation (15%), and Test (15%). To deal with the imbalanced data, RandomOverSampling and RandomUnderSampling were used in the Train set such that we had an equal number of positive and negative examples.

1.3 Metric

To evaluate the performance of our ML model on the test set, we decided to focus on the recall and precision values. Recall measures the proportion of successfully identified hard drive failures. Precision tells us out of all the hard drives that were predicted to fail, what proportion of them actually fails.

Formulas for calculating the 2 are as follows:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$
$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Naturally, we aim to predict all upcoming failures successfully, so that we can warn users in advance if their hard drive is about to fail. This means maximizing recall is our main priority for us, as we need to prevent situations whereby users using the application face a hard drive failure without warning.

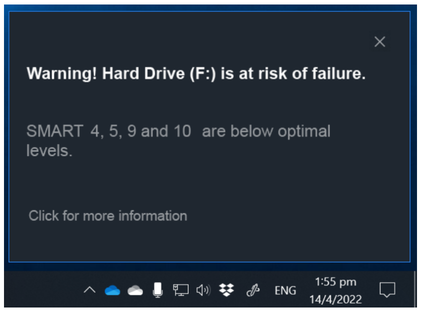
However, it is also necessary to ensure that our model does not constantly output false alarms, as it will cause annoyance to the user. This is why we also accounted for the precision, which represents the degree of confidence our prediction has that a hard disk is faulty. These metrics balance the feasibility of deploying our application in the real world to prevent loss of data due to hard disk failure while also considering real-world users' tolerance to false failure prediction.

1.4 Usage

Our model is intended to make a prediction based on the SMART attributes of a given hard drive. The application will warn users of possible failure and flag out certain SMART values which are causing the model to predict a failure value.

Overview of the proposed application:

1. Once the computer is turned on, the application will run in the background. If the model were to predict a hard drive failure, a warning notification will appear.



Warning! Hard Drive (F:) is at risk of failure.

SMART 4, 5, 9 and 10 are below optimal levels.

Click for more information

Figure 1: Warning message

2. Upon clicking the notification, the application will show the user a breakdown of all the SMART values and an indication of which values are causing the failure.

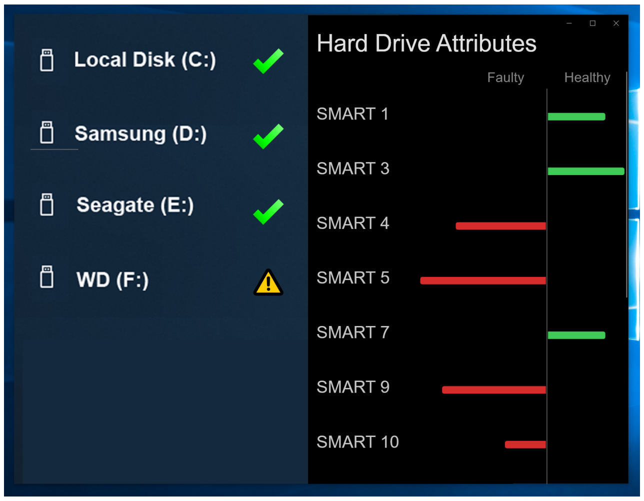


Figure 2: SMART Attributes Breakdown

Overall, the model adds value to the user as it reduces the probability of a hard drive failing without warning. The user can then make more informed decisions based on the prediction made by the model. If the model predicts failure, the user should either consider hard drive maintenance services or back up data into a separate hard drive as soon as possible to prevent data loss.

2 Selection of Model

Upon analysis, based on Pearson correlation, none of the variables have high correlation, implying no multicollinearity. After preparing the data, we do not have much noise in our dataset. These properties are helpful for considering which model to choose later on.

Using our data, we require a model capable of performing binary classification, based on numerical input attributes. We chose 3 ML models which we hypothesized would be appropriate for our intended purpose: Neural Networks, Bagging, and XGBoost.

Neural Networks (Marton, Lüdtke, and Bartelt 2022) have been proven to be powerful classifiers for complex problems, capable of finding underlying patterns in data while being robust against noise. At the same time, a trained Neural Network outputs predictions extremely quickly, which is well suited to our needs. With respect to our problem, Classification Neural Networks seem to be a very powerful tool we can leverage on. To implement a Neural Network model, we used the Multi-Layer Perceptron Classifier (MLPClassifier) from scikit-learn (sklearn), with relu activation function and stochastic gradient-based optimizer for weight optimization.

Bagging (Bühlmann 2012) stands for Bootstrap Aggregation. The algorithm utilizes the bootstrap sampling method to sample a subset of features within the dataset, then grows the largest decision tree possible using the sampled features. This is repeated to grow an ensemble of decision trees.

To make the prediction, each tree makes a prediction and their outputs are aggregated. Bootstrapping reduces overfitting since each subset tree has no knowledge of the other features, giving each tree some added flexibility. However, since the final prediction is based on the mean predictions from the subset trees, the classification result may not be precise.

Extreme Gradient Boosting (XGBoost) (Chen and Guestrin 2016) algorithm is widely considered to be a state-of-the-art ML technique amongst many data science communities. XGBoost is known to be well suited for dealing with numerical input attributes and high dimension datasets, features fast training speed, and has many parameters to prevent overfitting the model. These advantages align nicely with our problem and the constraints of this project.

2.1 Deciding on the Main Model

When comparing the models, Neural Networks produced weak predictions with low precision, which can be attributed to having insufficient data and time to fully exploit the potential of Neural Networks, due to the constraints of this project. Bagging was also severely outperformed by XGBoost in both metrics and was thus ruled out. One possibility for this could be that the Bagging algorithm forms its decision trees randomly, so the mean predictions of all the trees will not give precise values.

On the other hand, in XGBoost, each tree is formed to address the misclassifications made by the previous tree, making the model more precise with each iteration. In the context of our application, XGBoost will output predictions with low computational cost, reducing the strain on a user's hardware. Lastly, the large number of easily understandable hyperparameters in XGBoost means that we can easily tune the model to achieve the best performances. Based on these considerations, we therefore chose XGBoost for our work. Our approach is validated by our testing results elaborated in section 3.

2.2 Technical Understanding of the XGBoost Algorithm

XGBoost is a decision-tree based ensemble model that uses gradient descent to train a model to make predictions (Chen and Guestrin 2016).

Let the given dataset of n examples and m features be $D = \{(\boldsymbol{x_i}, y_i)\}(|D| = n, \boldsymbol{x_i} \in \mathbb{R}^m, y_i \in \mathbb{R})$. The objective to be minimised in XGBoost is

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \tag{1}$$

$$\text{where } \Omega(f) = \gamma T + \tfrac{1}{2}\lambda ||w||^2$$

and l is a differentiable convex loss function that measures the difference between the prediction $\hat{y}_i$ and the target y_i. Here $\Omega(f_k)$ acts as a regularization term where γ will determine whether a tree should be pruned at each split, as seen in the formula for $\mathcal{L}_{split}$ below and λ will lower the score of a split as it is in the denominator of the scoring function shown below. Both of these help prevent the tree from overfitting to the data.

As the objective includes functions as parameters, traditional optimization methods in Euclidean space cannot optimize the function. Instead, XGBoost greedily picks the tree that minimises the objective at each step, and the objective to be minimised becomes

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y_i}^{(t-1)} + f_t(\boldsymbol{x_i})) + \Omega(f_t) \tag{2}$$

Where $\hat{y}_i^{(t)}$ is the prediction of the i-th training instance at the t-th iteration. XGBoost uses a simplified approximation of the objective by using a Second-order taylor series approximation, giving us

$$\mathcal{L}^{(t)} \simeq \sum_{i=1}^{n} [l(y_i, \hat{y}^{(t-1)}) + g_i f_t(\boldsymbol{x_i}) + \frac{1}{2} h_i f_t^2(\boldsymbol{x_i})] + \Omega(f_t) \tag{3}$$

$$\text{where } g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$
$$\text{and } h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

Since g_i and h_i are constant for each iteration, they can be calculated once which decreases the computational burden. To Maximise $\mathcal{L}^{(t)}$, it is sufficient to maximise

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} [g_i f_t(\boldsymbol{x_i}) + \frac{1}{2} h_i f_t^2(\boldsymbol{x_i})] + \Omega(f_t) \tag{4}$$

as $l(y_i, \hat{y}^{(t-1)})$ is just a constant.
Define $I_j = \{i | q(\boldsymbol{x_i}) = j\}$ to be the instance set at leaf j, we can then expand $\tilde{\mathcal{L}}^{(t)}$.

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} &= \sum_{j=1}^{T} (\sum_{i\in I_j} g_i w_j + \sum_{i\in I_j} \frac{1}{2} h_i w_j^2) + \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T} [(\sum_{i\in I_j} g_i) w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda) w_j^2] + \gamma T\end{aligned}$$

where T is the number of leaves in the particular tree q.

Then to arrive at the minimum weight for a given tree $q(x)$ for the leaf j, we differentiate $\tilde{\mathcal{L}}^{(t)}$ with respect to w_j.

$$\frac{d\tilde{\mathcal{L}}^{(t)}}{dw_j} = (\sum_{i\in I_j} g_i) + (\sum_{i\in I_j} h_i + \lambda) w_j$$

$$\text{Let } 0 = (\sum_{i\in I_j} g_i) + (\sum_{i\in I_j} h_i + \lambda) w_j$$

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \lambda}$$

Substitute w_j^* into $\tilde{\mathcal{L}}^{(t)}$, we get

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2} \sum_{j=1}^{T} \frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

This can be used as the scoring function.
Instead of enumerating through all possible trees, q, XGBoost will greedily build trees by selecting the split with the lowest loss. At a given split, let I_L and I_R be the left and right instance set of the tree respectively, $I = I_L \cup I_R$, then at a given split, the reduction in loss can be calculated using the following:

$$\mathcal{L}_{split} = [\frac{1}{2}\frac{(\sum_{i\in I_L} g_i)^2}{\sum_{i\in I_L} h_i + \lambda} + \frac{(\sum_{i\in I_R} g_i)^2}{\sum_{i\in I_R} h_i + \lambda} - \frac{(\sum_{i\in I} g_i)^2}{\sum_{i\in I} h_i + \lambda}] - \gamma \tag{5}$$

This can be thought of as similar to calculating information gain described on page 16 of the Decision Tree lecture notes. To further improve the efficiency of the algorithm, instead of enumerating through all possible splits, XGBoost uses a weighted quantile sketch to split the data into quantiles and uses those as the potential split points.

XGBoost also supports column or row subsetting, which means that only selected columns or rows respectively are used to train the model. This is another factor to prevent overfitting of the model. XGBoost also compresses the data by columns into ”blocks”, which allows for parallel computation. XGBoost has cache-aware access to reduce inefficiencies caused by cache miss or when the data does not fit in the cache. The technical properties discussed in this section thus help us understand the workings of the XGBoost model and why it can perform well for our use case.

2.3 Hyperparameters

XGBoost offers many hyperparameters that are interpretable by the user, and can be used to optimize their model. Such optimizations can be extremely useful in picking up patterns and regularities in the data, as well as tackling the issue of overfitting to our training data. The following are some hyperparameters we used to improve our model (Xgboost 2022).

$\underline{learning\_rate(\eta)}$ ensures that the weights at each iteration are shrunk so as to avoid overfitting. Higher learning rates result in faster training of the model but are more susceptible to overfitting.
$\underline{max\_depth}$ We limited this parameter to mitigate the possibility of the algorithm building complex trees, which helps reduce overfitting.
$\underline{min\_child\_weight}$ Minimum sum of instance weight needed in a child. The tree will not split if the sum of instance weight does not exceed the given threshold. Larger values will result in less complex algorithms, preventing overfitting.
$\underline{n\_estimators}$ indicates the number of trees (or rounds) used in the XGBoost model. Increasing the number of rounds helps increase the accuracy of the model since it means constantly using prior knowledge to build new trees before aggregating the results.
$\underline{gamma}$ As explained in section 2.2, larger values correspond to higher thresholds hence less complex algorithms. This parameter helps us to prevent overfitting by adjusting the penalization factor used in the computation of the regularized objective.
$\underline{reg\_alpha}$ adjusts the L1 regularization term on weight, penalizing the sum of the absolute value of the weights when misclassification is made. Larger values will result in less complex algorithms. This is used in our case where our data had fairly high dimensionality, thus allowing the algorithm to run faster when implemented and prevent overfitting.

3 Experimental Setup

We trained the models mentioned in section 2 with the train dataset described in section 1.2. We used the validation set to gauge the performance of the different models and to tune our hyperparameters for XGBoost. We tested the final model on unseen data (test set).

We found that XGBoost gave the best overall results in accordance with our intended metrics in section 1.3 as shown in Figure 3 below.

	Neural Networks	Bagging	XGBoost
Precision	0.175	0.216	0.562
Recall	0.736	0.573	0.660

Figure 3: Table of Metrics of Trained Models

This table indeed validates our earlier decision in Section 2.1 to focus on using XGBoost. Therefore, moving on, we aim to iteratively improve our model to acquire better results.

For our experiment, we used GridSearchCV, which exhaustively tunes the hyperparameters of XGBoost to improve our results. Additionally, with the ability to do cross-validation in tandem with testing the parameters, it ensures the model results are consistent.

The specific hyperparameter values used are:

Hyperparameter	Range of Values
learning rate, η (eta)	0.05,0.1,0.3,0.5,1
$gamma$	0,0.1,0.2,0.3,0.4
reg_alpha	10^{-5},10^{-2},0.1,1,100
max_depth	5, 7, 9
min_child_weight	1, 3, 5
$n\_estimators$	50,200,400,500

Figure 4: Hyperparameters Used

These values chosen were meant to represent an extensive range allowable by the size of our dataset and in accordance with the industry's best practices to prevent overfitting.

By iteratively tuning each hyperparameter and observing the results generated, our final model achieved a recall of 76.39% and a precision of 34.06% on the validation set, as shown below. (Note in section 1.3 we noted we will be more tolerable to lower precision).

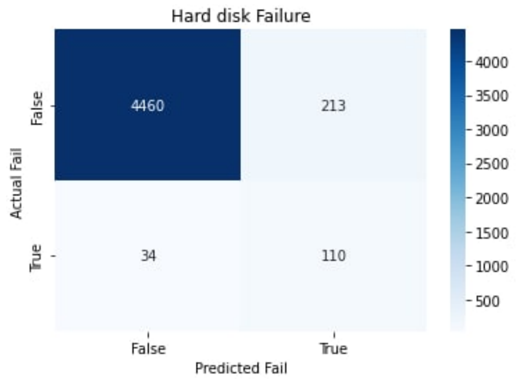


Figure 5: Confusion Matrix of Final XGBoost Predictions

Upon testing our model on the unseen data, thereby simulating real-life users inputting their hard drive attributes to the application to get a result, we obtained a relatively high recall of 80.88% and precision of 30.99%.

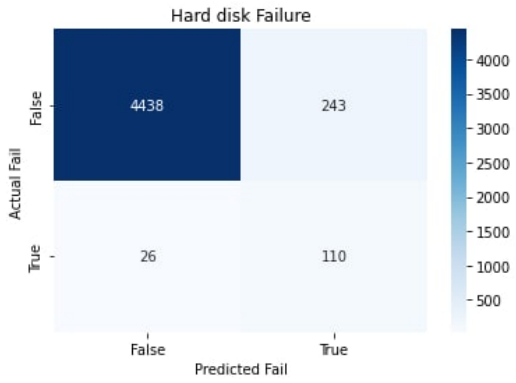


Figure 6: Confusion Matrix of Final XGBoost Predictions on Unseen Test Data

3.1 Discussion of Findings

Our test results tell us that a vast majority of all potential failures of a user's hard drive will be caught by our application, and 1 in 3 times a warning is put out, the hard drive is guaranteed to fail. Additionally, the low False Positive Rate (FPR) of 5.19% indicates our application will very rarely flag out a working hard drive as faulty. This is significant for our application as it shows confidence that our application will produce meaningful predictions. Since the application is run when detecting a connection to a hard drive, the application will be frequently executed, allowing us to better capture potential failures of hard drives through a probabilistic approach.

In fact, these results were not as ideal as we hoped for at the start. Our initial aim was to accurately warn users about their failing hard disk all the time, but we realized during the training phase that such an optimistic aim was not really possible without compromising the number of wrongly predicted failures. However, this factor is also important since we do not want to annoy our users with constant false alarm predictions. Therefore, in finding a good balance between the two, we were only able to predict a hard drive failure most of the time.

Nevertheless, this result still fits in nicely with our intended application's aim to consistently and accurately warn users whenever their hard drive is expected to fail in the next 7 days. This helps to prevent an unexpected hard drive failure that leads to unexpected data loss.

3.2 LIME (Local Interpretable Model-Agnostic Explanations)

Next, we used LIME (Ribeiro, Singh, and Guestrin 2016) to explore what are the features that contributed to the prediction for each of the failed hard drives. LIME allows us to focus on each individual failure which allows us to inform the users what SMART attributes are the cause of failure, leading to easier verification of the problem. This may reveal new findings about attributes that people do not usually associate with hard disk failure.

$$\xi(x) = \underset{g\in G}{argmin}\ \mathcal{L}(f, g, \pi_x) + \Omega(g) \tag{6}$$

Legend	
f	Original XGBoost model
G	All classes of simple interpretable model
g	Instance of simple interpretable model
π_x	Proximity measure between an instance to x
$\Omega(g)$	Complexity measure of chosen model

The formula above indicates that for each input instance x, we look for the best approximation of our original XGBoost model f by a simple model g in the neighborhood of input instance x while trying to stay as simple as possible.

This is done by minimizing the loss terms $\mathcal{L}(f, g, \pi_x)$ and $\Omega(g)$ functions. The process to minimize $\mathcal{L}(f, g, \pi_x)$ is by drawing random samples of training instances that are weighted by π_x, so as to consider only the instances locale to x. Then, we obtain the labels of these instances based on predictions of the original XGBoost model f. And lastly, we choose g that is best able to minimize $\mathcal{L}(f, g, \pi_x)$.

$\Omega(g)$ is simply minimized based by reducing the complexity of the simple model selected g.

By running LIME on one of the failure instances, we can get a clear picture of which attributes at which corresponding levels were giving rise to failure predictions (Figure 7). Larger positive weights (orange) indicate a higher contribution towards a positive failure prediction.

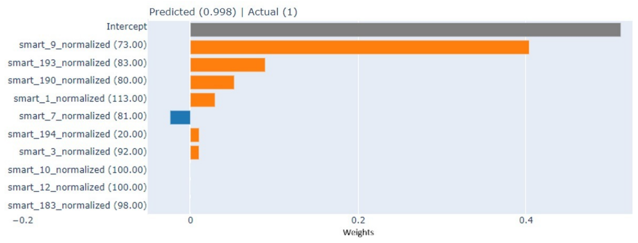


Figure 7: LIME Results on Particular Failure Chosen

Interestingly, before modeling, based on current domain knowledge, many experts cited metrics like SMART 5, SMART 187, SMART 188, SMART 197, SMART 198 which are currently used to indicate failure (Klein and Klein 2016). However, from this example, we see that there are actually other potentially influential factors like SMART 9, SMART 193, SMART 1 that indicate an imminent failure. This shows that ML is useful in improving our knowledge about hard drive failures since our model was able to find underlying patterns in data that is not easily identifiable.

We translated this easily interpretable information to a simple graph format as shown in our application interface (Figure 2), which gives users additional information on the aspects of why their hard drives may be failing. Through justifying our predictions, our application will be able to confidently answer 'Why Should I Trust You?', which is a prevalent topic in the ML community.

3.3 Recommendation for Further Research

With more data gathered over the years, Neural Networks can be more extensively trained, potentially outperforming other models. Stacking this with our current model might combine both their strengths and compensate for their weaknesses.

Additionally, based on our study, XGBoost uses a greedy algorithm to minimize the loss function. This is similar to how Greedy Best First Search (Stuart Russell 2012) is short-sighted in finding its goal. One possible novel improvement to XGBoost would be to take inspiration from A* Search and construct a consistent heuristic that is capable of making accurate predictions, XGBoost could potentially achieve a better ensemble of weak learners.

Overall, to expand on our current project, it's also possible to use the application to periodically fetch data, allowing the model to be constantly updated and improved periodically. Additionally, since XGBoost is able to handle sparse data independently, our model can still be trained as per normal even if new data provided has missing values.

4 Individual Findings

With this, we conclude our discussion on using ML to create an application to warn users of imminent hard disk failures. We are proud to say that all members contributed equally towards the completion of this project, including data cleaning, training and evaluating models, and the writing of this report. Exchanges of skills and knowledge about Computer and Data Science amongst members of the group made the project a meaningful experience for us. It was a challenge to understand the intricate underlying details of XGBoost that made it work well. Therefore, it was equally rewarding when we managed to get a good test score from the model, validating the use case of our application using XGBoost.

- Benedict: The hardest part for me was to qualitatively choose which models can help solve the problem. I also learn the purpose of the different hyperparameters of XGBoost and how to use them to our advantage.
- Teng Foong: I learned that different techniques exploit the different properties of the different processes to achieve desired outcomes so it is very important to select the best model for the given problem.
- Wei Lun: My takeaway is the necessity for mathematical derivations in understanding concepts of the ML models.
- Edward: I learned how to apply the knowledge that we have learned in lectures, like gradient descent, to understand machine learning papers.
- Zac: Many ML models stem from more simple algorithms, and understanding the technicalities of these algorithms allows us to better utilize them.

References

Backblaze. 2022. Backblaze Hard Drive Stats. *Backblaze Hard Drive Stats, Available at: https://www.backblaze.com/b2/hard-drive-test-data.html*, 14.

Bühlmann, P. 2012. Bagging, Boosting and Ensemble Methods. *Handbook of Computational Statistics*.

Chen, T.; and Guestrin, C. 2016. XGBoost: A Scalable Tree Boosting System, Available at: https://dl.acm.org/doi/pdf/10.1145/2939672.2939785. 785–794.

En. 2022. S.M.A. R. T. *Wikipedia, Available at: https://en.wikipedia.org/wiki/S.M.A.R.T.*, 14.

Klein, A.; and Klein, A. 2016. What SMART Hard Disk Errors Actually Tell Us. *Backblaze Blog — Cloud Storage and Cloud Backup, Available at: https://www.backblaze.com/blog/what-smart-stats-indicate-hard-drive-failures/*, 14.

Marton, S.; Lüdtke, S.; and Bartelt, C. 2022. Explanations for Neural Networks by Neural Networks. *Applied Sciences*, 12.

Ribeiro, M.; Singh, S.; and Guestrin, C. 2016. Arxiv.org. Available at: https://arxiv.org/pdf/1602.04938.pdf. In *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*, 1135–1144.

Stuart Russell, P. N. 2012. In *Artificial Intelligence A Modern Approach (4th Edition)*, 189, 202.

Xgboost. 2022. XGBoost Parameters — xgboost. *Xgboost.readthedocs.io. Available at: https://xgboost.readthedocs.io/en/stable/parameter.html*, 1.(5.).