Title
Prediction of Manufacturing Data using several machine learning approaches

Author
Li, Luxi

Publication Date
2018

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA
Los Angeles

Prediction of Manufacturing Data using several machine learning approaches

A thesis submitted in partial satisfaction
of the requirements for the degree
Master of Applied Statistics

by

Luxi Li

2018

ABSTRACT OF THE THESIS

Prediction of Manufacturing Data
using several machine learning approaches

by

Luxi Li
Master of Applied Statistics
University of California, Los Angeles, 2018
Professor Ying Nian Wu, Chair

This thesis is to meet the needs of developing automation process on defective testing in the hard disk drives production process, focusing on machine learning and artificial intelligence. The objective is to to predict the defectives and improve the accuracy rate by using the tree based algorithm and neural networks. The powerful models can help manufacturing process improving time and labor efficiency.

Keywords Manufacturing data, binary classification, machine learning, TensorFlow, XGBoost, H2O, supervised learning, random forest, gradient boosting machines, residual network.

The thesis of Luxi Li is approved.

Frederic P. Schoenberg

Nicolas Christou

Ying Nian Wu, Committee Chair

University of California, Los Angeles

2018

To my parents . . .

who always understand me deeply

and care about my feelings

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my supervisor, Dr. Ying Nian Wu. Without his guidance, I can not complete the project as smooth as I planned. He has been generous and always encouraging with me. Thank you for the dedication of time and efforts to this project.

I would also like to thank the committee member, Dr. Nicolas Christou and Dr. Frederick Schoenberg, who has made valuable comments and helped me to refine the report.

CHAPTER 1

Introduction

1.1 Background and motivation

Hard disk drive, simply as HDD, is the dominant storage device used for data storage and information retrieve. It is made up of magnitude materials, and its information retrieving processes go through several platters rotations. There are hundreds of reading points inside the drive so that large capacity of information can be saved. Ideally, the factory wants to guarantee that the disks are fully functioned, but in reality of the manufacturing production process, defectives are included on the production line with rare possibilities. To avoid defectives deliver to customers, the process of testing product defective or not, simply as fail or pass, is important. However, manually doing the tests is costly and labor-consuming. In fact, from the past two decades, automation of testing products becomes a new trend on manufacturing process – manufacturing is on the cusp of a new automation era: rapid advances in robotics, artificial intelligence, and machine learning are enabling machines to match or outperforms humans in a range of work activities such as testing.

To achieve the cost reduction and improve the efficiency of labor work, artificial intelligence can make the testing process automated, that is, establishing the models using previous data of products can predict a new product as passed or failed. Given a large scale data of produced HDDs, the prediction to the new disk becomes possible by applying machine learning algorithms. But how the predictive methods work and successful? In this thesis, the main objectives are trying several approaches to do the predictive analysis on the HDDs data in order to get precised prediction. The analysis is established by theoretical point of views.

1.2 Data description

The data is source from Western Digital Corporation, the leading company of computer data storage and one of the largest HDD manufactures in the world. Data is constructed from HDD reliability statistics, consisting 99,164 product test data with total of 1035 columns including 1032 testing points, time feature, id number and passfails. Each feature is a testing point inside the disk with a numeric number and the pass indicates as 0, while 1 means fail. Data details are shown on table 1.1

Category	Detail
Date Time	time and date
Id	serial number that manufacturer assigned
GIDs	different testing points on HDDs.
Pass Fail	0 means pass, 1 means Fails.

Table 1.1: HDDs Data

Data can be separated into four categories. First, the unique serial number that each disk is assigned, which is useful for identification. The date and time feature shows when the disks are produced. Third category as testing points which range vary on different points, simply as GIDs, from 0 to 1 or over a hundred. Depending on the algorithms, normalization of data is required. The last category is the indicators presenting the disk passed or failed, there are 79% of passes and 21% of fails. More detailed analysis are described on the later section.

1.3 Outline

In this thesis, training and optimization of models with different machine learning algorithms and open-source packages is the main focus. Another focus is how the model accuracy can be improved. The paper is organized as follows: Chapter 2 sets up for the literature reviews of papers with the algorithms and package libraries that are used on the experiments. Chapter

3 establishes the theoretical basics for the machine learning methods. Chapter 4 describes the details of packages, and their usage and advantages on training process. Chapter 5 analyzes and evaluates the model and performances. Chapter 6 discusses the results and makes some recommendation for the future analysis.

CHAPTER 2

Literature reviews

In the statistical modeling, Breiman[6] provided two different way of thinking: one is modeling the data by understanding about the relationship between input variables and outcomes, and applying a stochastic model on the data generating process. Another is the algorithmic modeling that applying unknown mechanism to do optimization, which is an imitation process to get close to the true mechanism as good as possible.

Breiman[7] established random forests which originally conceived as a way of combining several CART [5] decision trees using bagging [4]. With Biau's paper[3], the algorithm is formalized on keeping the consistency of several randomized ensemble classifiers. Then, on Denil's experiment[10], it is proven to the consistency of model based on Breiman's theory. Kulkarni and Sinha [21] presented approaches of improving performance of random forest classifier in terms of accuracy by using dynamic programming. Then Louppe[23] took insights of variable importance with the theoretical characterization of the mean decrease of impurity.

When data became large, MapReduce clusters were applied on training models. According to Wakayama[26], large data with parallel distributed processing is more promising for solving the problem. MapReduce[26] has two procedures. One is that mapper performs in-parallel tasks on sub node, and reducer receives the result from mapper with a summary return to the master node. When random forest is applied, mapper is used to set up trees and reducer is used to generate the created trees and find the final forest.

Random forest is one of the common approach to build a single strong predictive model, which rely on simple averaging of models in ensemble. Boosting is using different strategy that adding new weak base-leaner model at each partition with respect to the error of whole ensemble. The initial work on boosting algorithms, Freund and Schapire proposed

adabptive boosing algorithm, simply as AdaBoost [11]. The main idea is to add weights to the data instead of randomly sub-sampling the data. Then, the more robust version of boosting introudced by Friedman[12] called gradient boosting machine which is that the learning procedure consecutively fits new models to provide a more accurate estimation to the outcomes. The gradient boosting machine, simply GBM, can highly customize to any particular data due to the high flexibility. An improved version of GBM on speed of training model is extreme gradient boosting, simply as XGBoost, which implemented by Chen[9]. Ensemble Learning is the approach of taking advantage of those ensemble machine learning algorithms to improve the model performance demonstrated by LeDell[22] on his dissertation that when single algorithm cannot approximate the true prediction function, super learner algorithm can combine multiple diverse learning algorithm together to create more powerful model, which can be applied through the open source H2O.

Another way of building a strong model is using neural networks, simply as NNs. The basic idea of NNs, the perceptron model, was constructed by Rosenblatt[24], and the back-propagation algorithm was fully appreciated until Rumelhart's paper [25]. Paper described that compared to several neural network, the backpropagation works far faster, and make it possible to use neural nets to solve problems. Hansen and Salamon[16] introduced the neural network ensembles which applied cross validation as a tool of optimizing network parameters and invoked ensembles of similar networks by reducing the total errors. Recent contribution in designing architectures for deep networks has accelerated the development of networks. The residual networks, simply ResNets, is wisely used on computer vision applications[18]. As He[18] mentioned, ResNets are obtained by stacking simple residual blocks to learn network parameters θ. Not only the broadening applications on real computer problems, ResNets by Xie[28] successfully focused on improving accuracy and stability by Harber[17] on the training process. Xie[28] introduced ResNxt, a homogeneous and multi-branch architecture to increase the prediction accuracy. And Stochastic depth by Huang [20] reduced the training time at the meantime increasing accuracy by randomly dropping a subset of layers and passing them to the identity function. Those successful improvement on NNs proved that NN is good approaches to train models. To make NNs applicable, TensorFlow

was explored by Google[9], and widely used recent years.

CHAPTER 3

Methodology

In this chapter, theoretical explanations of several machine learning algorithms are given. Two categories of methods are explained, one is tree-based algorithms of random forest and gradient boosting machines, another is neural network algorithm. Since the predictor, passfails, has only two indicators 0 or 1, the algorithms suitable for binary classification were used for training. Tree-based algorithms involve segmenting the predictor space into a number of simple regions which has the formula below:

$$f(x) = \sum_{n=1}^{N} w_n I(x \in Region_n)$$

On the other hand, the goodness of neural networks performs depend on the data. Originally Neural network is inspired by the brain, which consists of a network of neurons. Mathematically, NN is a concatenation of weighted functions. As the data becomes large, the neurons connections become stronger and ideally better model can be constructed.

3.1 Tree based algorithm

3.1.1 Random forest

Random Forest is initially mentioned by Breiman[7]. Repeatedly taking a bootstrap sample from the whole data with replacement to fit a classification tree. At each node, select several features at random out of all possible features, ant they are independent at each node. According to some objective function, the feature which can provide the best split is used to do a binary split on that node. Once the best split point is found on the selected variables, the tree is grown bigger. Then, at the next node, choose another features with same number

at random from all and repeat the same process. Breiman[7] suggested the possible values as square root of the number of features, half of square-rooted number of features or twice of the square-rooted number. Because of the randomness, Random Forest can avoid overfitting on most of the cases.

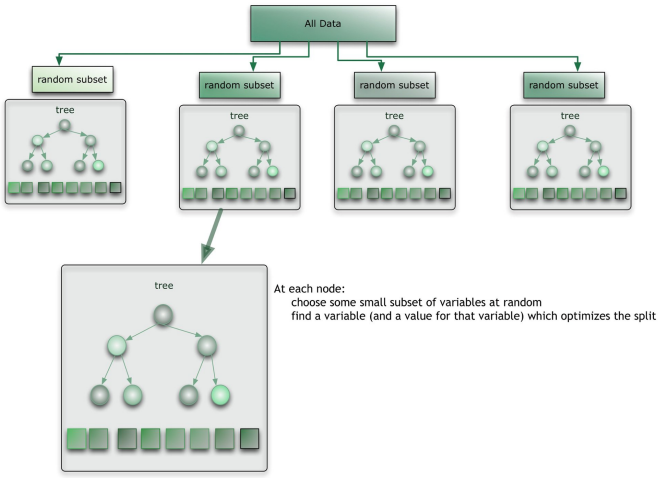

Figure 3.1: Random Forest [2]

When data is complex and voluminous, computer machines cannot handle to run a model in a short efficient time because the large number of trees needed to be built in order to output a result. In this case, an acceleration on training process is required. Distributed random forest has the strategy of computing the forests faster by building trees parallel on subsets of data. Starting with randomization partition of data to the nodes and building trees parallel, the training process goes through on multiple nodes, and each node builds a subset of forest at the same time, trees are independently constructed.

3.1.2 Gradient boosting

Gradient boosting came from the AdaBoost[8] that weak learner can be modified to become stronger, decision trees with a single split. Weak learner is defined as a classifier that is

slightly correlated to true classification, but slightly better than random probability.. According to Friedman[12], each step consists weighted minimization and recomputation of classifier and weight. Generalization of AdaBoost then becomes gradient boosting. According to Friedman's gradient boosting algorithm[12], on each iteration, the gradient descent is first computed in order to fit a new base learner function. Once the best gradient descent step size is found, the function estimation is updated. Here is the formula of gradient descent:

$$\theta_{n+1} = \theta_n - \eta \text{Ł}'(\theta_n)$$

where η is the step size or learning rate, and $\text{Ł}'(\theta)$ is the gradient.

The new model is fitted using information of errors from the previous model. The resulting predictors are the combination of all models. GBMs build trees at the same time, and new trees help to correct errors made by previous trees. Typically, number of trees, depth of trees and learning rate are the most important parameters. The recommendation is to choose smaller trees rather than larger ones. Other regularization method includes stochastic gradient boosting which is the boosting with sub sampling per split, and regularized gradient boosting which is the boosting with $L1$ or $L2$ regularization. At each iteration, the data are sampled into without replacement and the tree is built only using the sample of data.

Compared to random forest, GBM tends to have overfitting, but at the same time, if the hyperparameter setting is correct, GBM can have strong prediction to the data.

3.2 Neural network

Neural network is the third approach of training the model, inspired from biology that enables computer to learn from observational data. Perceptron model, the artificial neuron, were developed by Rosenblatt[24]. It is a deterministic version of the logistic regression, and the basic idea is to take several binary inputs and produces a single binary output. When the data is more complicated, the perceptron can weight up more complex network to make suitable outputs. Thinking of two layer network, the first layer weights the inputs and the second layer weights up the results from the first layer to make new outputs. The second

layer can make precise outputs on a more complex and more abstract level than perceptrons in the first layer. In this way, the third, forth layers can be added to a network. A multi-layer network of perceptrons can engage in more sophisticated data.

When small changes in the weights or bias happens on any single perceptron in a network, the output of that perceptron can sometimes completely change. To avoid this problem, the artificial neuron, sigmoid neuron is introduced. Sigmoid neurons are similar to perceptrons but can modify the small changes in the weights or bias can cause only small changes on their output. Same to a perceptron, the sigmoid neuron has weights for each input, $w_1, w_2, ...$, and a bias b. The outputs become as sigmoid function:

$$\sigma(wx + b) = \frac{1}{1 + \exp\left(-\sum_{j=1} w_j x_j - b\right)}$$

Sigmoid funciton can smooth step function of 0 and 1 by outputting any real number between 0 and 1. Rectified linear unit, simply as ReLU, is another important function for the perceptrons.

$$\sigma(wx + b) = max(0, \sum_{j=1} w_j x_j + b)$$

The major benefit of ReLU is the reduction on likelihood of vanishing gradient. When $\sum_{j=1} w_j x_j + b > 0$, the gradient will have a constant value. In contrast, gradient of the sigmoid function is increasing less when x increases as the figure 3.2 , and hence the constant gradient of ReLU causes faster learning compared to sigmoid.

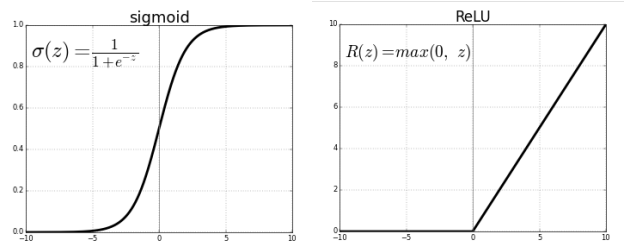

Figure 3.2: left: sigmoid function, right: ReLU function.

Sigmoid function saturates at two edges, which causes gradient to vanish, while ReLU

does not. When the input is large in absolute value, the gradient on sigmoid function becomes smaller. Since the derivative of sigmoid funciton is always smaller than one, when multiple layers exist, the gradients of those multiplication can quickly converges to 0. Another benefit is that ReLU can add more sparsity to the input data since when $\sum_{j=1} w_j x_j + b < 0$ arises, ReLU can make those units to 0, while sigmoid introduces some negative values.

The figure 3.3 shows basic NNs flow, starting from the leftmost layer called input layer. The middle layer is called a hidden layer which can have more than one layer, and the rightmost contains output neurons. The output from one layer is used as input to the next layer. Such network is called feedforward neural networks, which means no loops in the network.

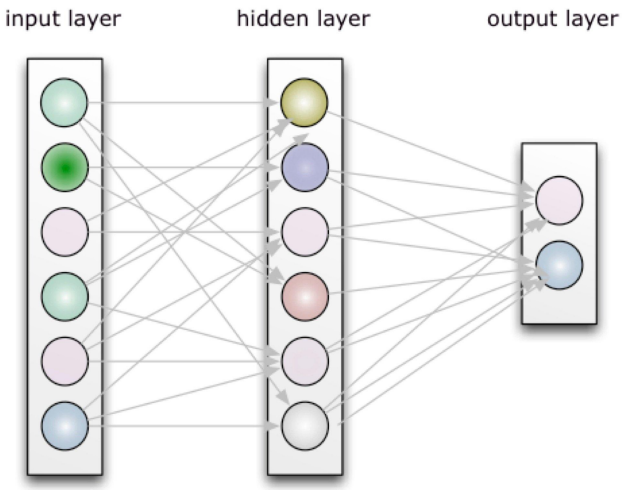

Figure 3.3: NNs flow plot [2]

In order to calculate the gradient of the loss function, back propagation algorithm[25] is used. The main idea is that the partial derivative of cost function L with respect to any weight w or bias b in the network, as $\frac{\partial L}{\partial w}$ or $\frac{\partial L}{\partial b}$. The derivatives can explain how changing the weights and biases changes the overall behavior of the network. Now, the network includes two phases, propagation and weight updates. And the whole process becomes: input vector is propagated forward to the network to several layers and finally to the output layer. The output from the network compare to the true output by using loss function. The resulting

errors are then propagated from the output back through the network until each neuron has an associated error values. Then, those error values are used to calculate the gradient of the loss function, and this gradient can update the weights in order to minimize the loss function.

When the network is trained by back propagation, the outputs from hidden layer keep changing due to the parameter changing, which causes a change in the distribution of the network activations, called inverse covariate shift. To stabilize, one layer of batch normalization is added to normalize the output of previous layer, by subtracting the batch mean and divide by the batch standard derivation, thereby eliminating the vanishing gradient problem.

There are variants of gradient descent. The traditional gradient descent calculates the gradient of whole data but perform one update at last, hence it is very slow when the data is large. Stochastic gradient descent, simply as SGD, on the other hand performs one parameter update for each training example, which is faster. The frequent updates can have high variance and causes loss function to fluctuate differently. Mini batch gradient descent performs an update for every batch with some training example in each batch, which can avoid the problem of SGD that producing high variance parameter and traditional gradient descent that can be very slow. To furthur optimize gradient descent, momentum and adaptive moment estimation, simply as Adam, can be used on training process. Since SGD causes high variance and makes it hard to reach convergence, momentum can accelerate SGD by adding a fraction number of the update vector of last step to the current vector, which can be faster and stable on convergence. Adam is to compute the adaptive learning rates for each parameter, and keeps an exponentially decaying average of the previous gradient, which is similar to momentum.

Residual neural network, simply as ResNet, is constructed by adding several residual blocks to the original NNs [18]. The difference between residual blocks and ordinary NN blocks is that residual blocks can optimize the residual mapping rather than just using the original mapping, therefore, a small changes from 5.0 to 5.1 will have the rate lower than 5%, but the residual changes can be 100%.

3.3 Performance measures

To explore the model, training data is used, and to evaluate the model strong or not, the testing data is used. Since the model prediction of HDDs data is in the form of class label 0 or 1, the confusion matrix is used for checking whether the predictors are truly classified into correct classes. Four main results are on confusion matrix: True Negative, simply as TN, True Positive as TP, and False Negative as FN, False Positive as FP. TP means correct classification to positive class, while FP means incorrect classification. The general confusion matrix form is in the table 3.1.

	Predicted 0	Predicted 1
True 0	TN	FP
True 1	FN	TP

Table 3.1: General confusion matrix for binary classification

Many evaluation measures of binary classification are derived from the confusion matrix. And some important measures derived from confusion matrix are listed on the table 3.2.

Measure	Formula
True Positive Rate (TPR)	$\frac{TP}{TP+FN}$
False Positive Rate (FPR)	$\frac{FP}{FP+TN}$
True Negative Rate (TNR)	$\frac{TN}{FP+TN}$
False Negative Rate (FNR)	$\frac{FN}{FP+N}$
Positive Predictive Value (PPV)	$\frac{TP}{TP+FP}$

Table 3.2: Performance measures on binary classification

TPR, called sensitivity or recall, is the proportion of true positive classification against all true classification. When the predictor has unbalanced class, and positive class is a minority, TPR becomes important since the extreme case is that when 99% of negative classes exists, model with prediction of all outputs as negative class can reach the accuracy to 99%, but TPR is 0% since no positive class is classified, which shows the model performance

is really bad, and hence TPR is one measure of true classification of predictor. Improving TPR becomes important on model optimization. TNR is specificity that proportion of true positive classification against all true classification. PPV, called precision, is useful measures as the proportion of correct predicted classes and all positives.

An method of evaluating performance is finding optimal threshold if the model predictions are in the form of real numbers or probability, called ROC analysis. This method is based on dividing the prediction using all possible thresholds and calculating the specific measures for each threshold and plot the graph for each measures. The ROC plot shows the dependency between specificity and sensitivity. Resulting curve starts from (0,0) and ends at (1,1) position. The better a model is, the closer the curve reaches to (0,1). The result of ROC analysis is a performance measure called Area Under Curve, simply as AUC. When AUC is 1, then it is perfect prediction. Ideally, higher AUC shows the model performs better.

CHAPTER 4

Analysis of packages

In this chapter, analysis of packages is given. The comparison between the packages used for experiments, advantages and disadvantages are listed. For the HDDs data, open-source application, H2O and XGBoost are chosen to train the tree-based algorithm model, while Tensorflow is used for NNs approach.

4.1 H2O

Known R packages for random forest and gradient boosting are *Random Forest*, *e*1071, *gbm* and *caret*. Python has *Random Forest* and *scikit learn* which is available for almost all ensemble machine learning approaches. However, regarding to the HDDs data, since features are over a thousand, those packages can take very long time on training models if applied on usual computer machines, which is time-consuming. Since the experiments went through on traditional CPU machine, the packages with implementation of fast computing is comparatively useful.

H2O.ai is one of the powerful open source of AI machine to both individuals and enterprises. It is a in-memory, distributed, fast and scalable machine learning and predictive analytic platform to build models on big data[13]. The main difference of H2O to other know packages is that it used distributed MapReduce framework and utilization on implementation in Java. For instance, data on *read.csv()* function in R is read row by row, while H2O can force the reading in a parallel way, save the column-format data across the clusters. H2O compresses the data reading and improves the speed. Since H2O is good at exploiting multi-clusters, training process becomes faster and efficient. H2Os GBM sequentially builds

regression trees on all the features of the data in a fully distributed way - each tree is built in parallel. In addition, H2O can handle the missing values, imbalanced features and drop the bad columns such as constant columns out during training. The data without cleaning can also be trained to a model with good performance. H2O can support training the model at any time, which means that disruption on modeling without losing information of trained model can be accomplished and continuously modeling is available from the part where the training stopped.

4.2 XGBoost

An improved faster computation on GBM is extreme gradient boosting, simply as XGBoost with implementation of gradient boosting decision trees. Compared to the *scikit learn*, XGBoost is extreme faster even though both of them are implemented on the same principle of gradient boosting. XGBoost uses sparse matrix with sparsity aware algorithms. When categorical feature with almost 0, XGBoost can find out the best splitting points on non-zero data and no need to look at whole entries. Moreover, similar as H2O, XGBoost does distributed computing for training very large models using a cluster of machines, and does parallel tree learning to quickly find the best split by using all of the CPU cores during training, and support multi-core processing, which reduces the total training time. Automatically handling the missing data is also a feature that similar to H2O. Another feature is that XGBoost can support continuing modeling from an already fitted model by streaming new data into the further boost constructions, and that's makes life easier on production process on HDDs as new data always produce from the factory. In order to make a consistent and high performance model, large volume of data is needed, and XGBoost can perfectly supports this process by saving the data matrix, models, or reloading it later, and possible to keep the model updated.

4.3 TensorFlow

The last open source package used for training process is TensorFlow, which is created by Google Brain[1]. The motivation of creating TensorFlow is to explore the use of large scale machine learning models. This library is used for numerical computations with the help of the data flow graphs. The main difference between ordinary machine learning packages and TensorFlow is that TensorFlow has higher flexibility on building machine learning models using a set of simple operators like *add* and *matmul*, in other words, it is a comparatively low level library, while other packages like *scikit learn* has already implemented the algorithms and few lines of codes is enough. TensorFlow is designed for deep learning, and hence representation learning is used instead of feature engineering, which means that TensorFlow can automatically extract important features. To establish deep network, TensorFlow is more useful than traditional machine learning packages. For the experiment, TensorFlow was used to construct NNs.

CHAPTER 5

Predictive analysis and evaluation

This chapter is about the experiments made with HDDs data. The measure statistics of models was evaluated on each approach with model improvements by tuning parameters or constructing complicated network. With the evaluation of the results, the comparisons and discussions are given.

The best models of each algorithm were selected according to the selected metrics, AUC and TPR, and all of the models were compared using ROC curves. Moreover, feature importance of optimized model is listed by each algorithm.

5.1 Basic data cleaning and analysis

Before moving to the modeling part, basic understanding of data is required. The original data includes 99,164 rows with 1035 columns. To understand the data better, some basic statistics are given on figure 5.1, figure 5.2 and table 5.1. At first, the missing values are taken into consideration. According to the data, missing values range from 0 to 460 per column, and total of 742 rows contains one or more missing values, which means that 0.7% of rows has the non-informative values and can be excluded from the original data. If H2O or XGBoost is applied, missing values are not necessary for removal, since H2O can handle the missing values by itself. The histograms are some testing points features, called $GID380$, $GID179$. Both of features distribute randomly and skew left or right, and their ranges differ. The table 5.1 is basic summary of some GIDs that have different ranges, some are negative numbers and some are more than 400. Thus, when necessary, standardization of features is required.

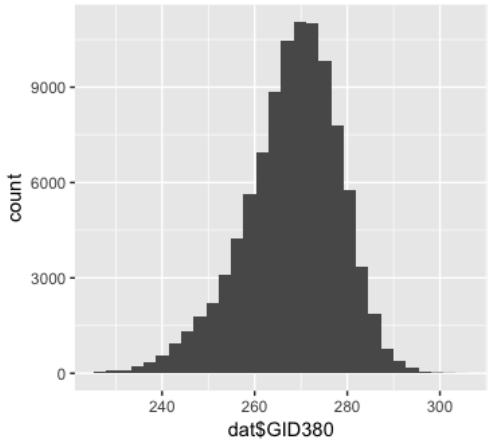

Figure 5.1: Feature GID380.

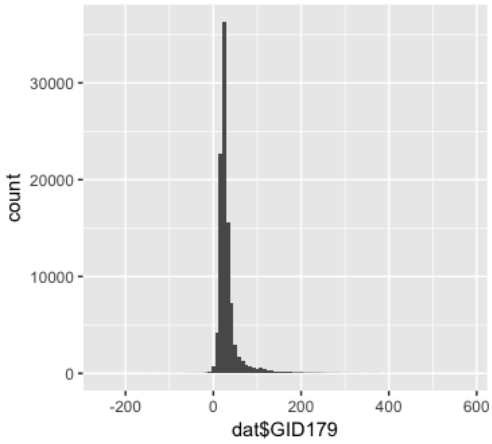

Figure 5.2: Feature GID179.

	GID89	GID997	GID981	GID1013	GID192	GID851	GID453	GID764
Min	42.0	0	0	-98	-4362	-4582	0	-3
Median	479.0	0	5	-7	32	0	2	-1
Mean	479.2	2.74e-04	2.939	-7.42	27.67	-21.44	2.289	-0.8029
Max	663.0	1.10e+01	5.000	36.00	571.00	6567.00	63.000	3.0000

Table 5.1: Summary statistics for some features

Another component needs to be considered is date and time feature. The time range is from January 4th to 18th, containing two weeks, and time interval is randomly distributed without any pattern, which shows that the testing sample disks are randomly chosen from the production line. In order to avoid the loss of information, the time feature is kept on the modeling process, but it is converted from day-time base to time base. The reason is that time impacts to the disks has much more information than date, which can be a case that the disks produce in the morning may have more defectives compare to the night production.

The predictor, passfail, is a binary feature that only includes 0 or 1. 0 means the product is passed, while 1 is failed. The figure 5.3 shows the barplot of how passfail is distributed. It is obvious that 0s have twice or more than 1s. In fact, there are 78187 for 0s and 20235 for 1s, which is unbalanced. For the further analysis of modeling, this unbalance has the risk of

causing a high prediction accuracy by predicting all of outputs as 0s. To solve this problem, the sampling techniques can be used. However, either H2O or XGBoost can deal with the unbalance of predictor by sampling the minority classes.

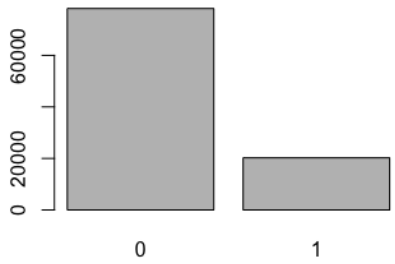

Figure 5.3: Barplot for passfail

5.2 Model analysis and evaluation

The baseline of the prediction is to use the worst scenario that a model predicts most of outputs as majority class or predicts all of outputs as minority class. The result confusion matrix of the baseline cases is in the table 5.2. In those cases, the accuracy is 79.05% and 20.94%. Both of the cases have comparatively low accuracy.

	Predicted 0	Predicted 1
True 0	19459	0
True 1	5155	0

	Predicted 0	Predicted 1
True 0	0	19459
True 1	0	5155

Table 5.2: left: all prediction to 0 right: all prediction to 1

Then, the predictive modeling is the next step. The selection of predictive approaches are: random forest, GBMs, and ResNets to perform as good as possible. Since random forest and GBMs are based on the decision trees, they are the good approaches to the unbalanced data in general. NN can be used for variety of data. Since HDDs data has high dimensionality,

NN is also a good approach of modeling in a complicated network.

H2O library is applied for training tree-based models. The implementations are distributed random forest, simply as DRF, and GBMs. First experiment was training a model with default parameters setting to see how the algorithm predicts. The default settings are described on the H2O documentation[14] [15]. Then, grid search were used for parameter tunning in order to find the best models.

Regarding to random forest, the hyperparameters for grid search included the number of trees selected from 10 to 200 at an interval of 10, and the maximum depth of 5 to 150 at an interval of 5. The figure 5.4 shows ROC analysis of the default setting model, and the table 5.3 shows the confusion matrix. The accuracy for training data is 97.7%, and testing data is 94.41%, AUC is 96.62% and TP rate is 85.8%. Those are very good statistics.

	Predicted 0	Predicted 1
True 0	18791	668
True 1	732	4423

Table 5.3: Confusion matrix base RF

Figure 5.4: ROC for base RF

The best model for DRF is when the number of trees is 100 and maximal tree depth is 100, with table 5.4 and figure 5.5. The AUC is 97.4%, accuracy is 95.32% and TPR is 87.82% for testing data. It shows a slightly improvement of the prediction accuracy. The figure 5.6 shows the variable importance. $GID98$ and $GID84$ are the most important features since the scaled importance shows only these two features have the importance over 50%.

	Predicted 0	Predicted 1
True 0	18927	532
True 1	628	4527

Table 5.4: Confusion matrix best RF

Figure 5.5: ROC for best RF

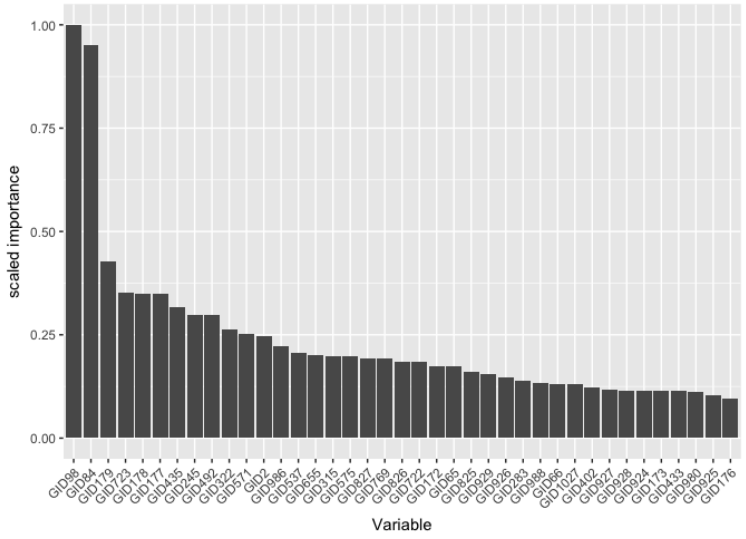

Figure 5.6: Variable importance for the best model trained on DRF

Regarding to GBMs, XGBoost and H2O were used. When H2O was used, the set of default parameters which is described in documentation [15] is the first model. The figure 5.7 and table 5.5 shows the model with AUC of 96.92%, accuracy of 95.44% and TPR of 87.18%.

	Predicted 0	Predicted 1
True 0	18998	461
True 1	661	4494

Table 5.5: Confusion matrix base GBM

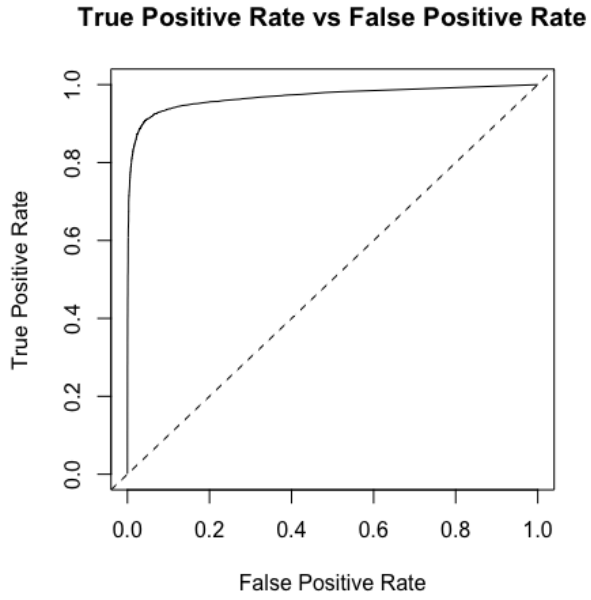

Figure 5.7: ROC for base GBM

As the hyperparameters for grid search, the selection of parameters were number of trees from 15 to 150 at an interval of 10, the maximum depth of 50, 80 or 100, and the learning rate from 0.01 to 0.1 at an interval of 0.01. The best model is when learning rate is 0.07, maximum depth is 80, and number of tree is 135. The AUC becomes 97.43%, accuracy is 96.06%, and TPR is 87.7%. Figure 5.8 and table 5.6 shows the result. Figure 5.9 is the variable importance for the GBM model. Again, $GID84$ has the highest scaled importance, and $GID179$ is secondly important variable which ranked thirdly on DRF model.

	Predicted 0	Predicted 1
True 0	19126	333
True 1	638	4517

Table 5.6: Confusion matrix best GBM

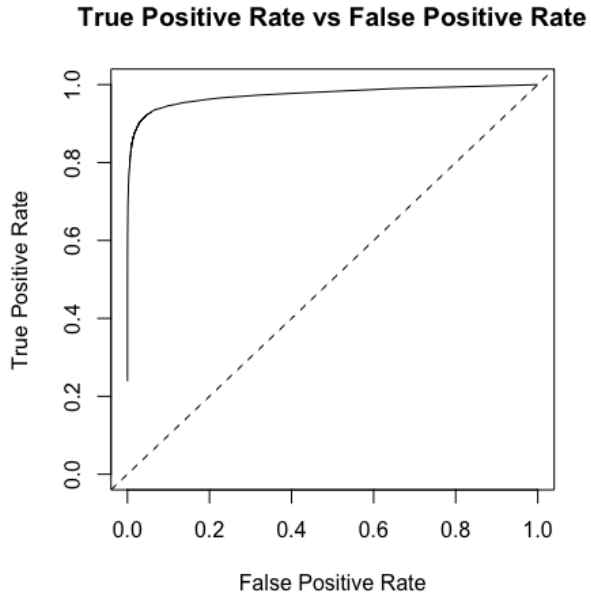

Figure 5.8: ROC for best GBM

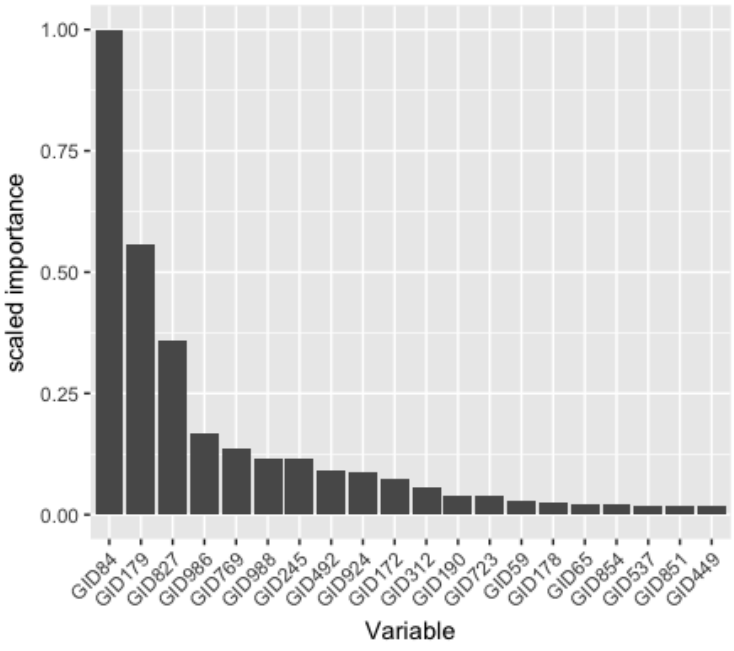

Figure 5.9: Variable importance for the best model trained on DRF

XGBoost is another package used for GBM. The initial model set the parameter as the

default setting which is described on [27]. The AUC is 90.5% with the accuracy of 85.8%. Then, the grid search for tuning parameters was performed by experimenting the learning rate from 0.01 to 0.1 at an interval of 0.01, and maximum depth from 50 to 100 at an interval of 10 with number of round as 100 and early stopping of 10 rounds. Early stopping is defined as when the model performance does not improve on N rounds, the model will stop training further and keep the best number of boosting rounds. Moreover, number of thread is defined as 5, which is the number of parallel threads used to run XGBoost. Finally, the optimized model was the set of learning rate of 0.08 and maximum depth of 80 on the 36 rounds. The summary evaluation statistics is the training data has AUC of 99.4%, testing data has AUC of 97.5%, the accuracy 95.62% and TPR of 85.37%. The figure 5.11 shows the feature $GID244$, $GID222$, $GID83$ are important.

	Predicted 0	Predicted 1
True 0	19203	339
True 1	742	4330

Table 5.7: Confusion matrix best XGBOOST

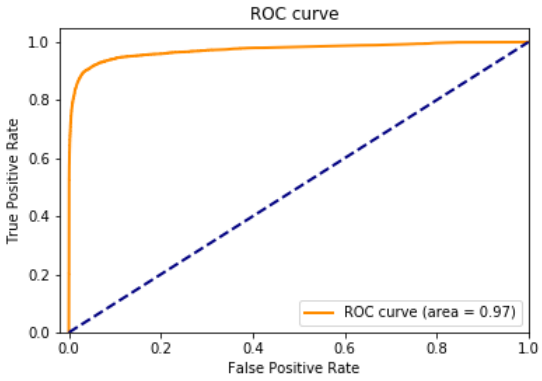

Figure 5.10: ROC for best XGBOOST

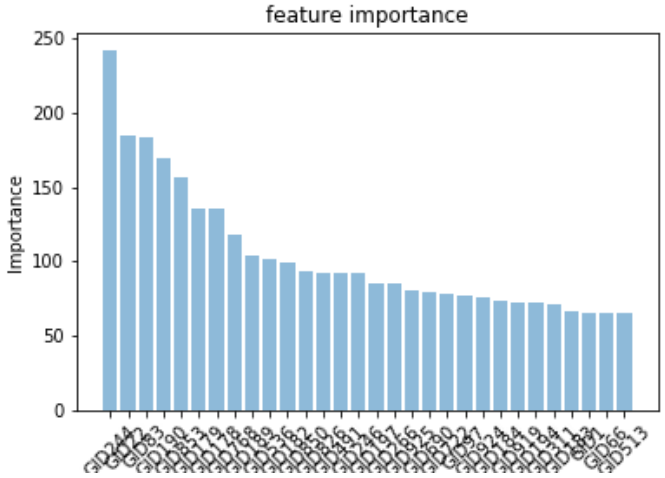

Figure 5.11: Variable importance for the best model trained on XGBOOST

Last approach, modeling of NNs, were built under Tensorflow. The process of building model is different, because the models are constructed layer by layer, and the best structure with best parameter settings are explored together. The first model was built by 3 hidden layers without batch normalization. On each layer, Sigmoid function or ReLu function was used. To use a better optimizer, SGD, SGD with momentum and Adam optimizer is used under the same parameter settings to see which one has higher evaluation statistics. The table 5.8 shows that SGD with momentum works best, hence the further modeling depends on the momentum optimizer. Then, to get higher accuracy, more layers and batch normalization were added. The accuracy for testing data could reach to 89.5%. When 10 layers, the accuracy becomes 91.5%.

Optimizer	Accuracy
Gradient Descent	85.86%
SGD Mometum	89.5%
Adam	88.73%

Table 5.8: Table of optimizer with 3 layers network

When changing some parameters, such as starting learning rate from 0.001, 10 hidden

layers, batch normalization, SGD momentum optimizer and total epochs of 200, the AUC reached to 95.3% with the accuracy of 93.6%, TPR becomes 81.1%.

	Predicted 0	Predicted 1
True 0	18842	617
True 1	938	4217

Table 5.9: Confusion matrix ResNet

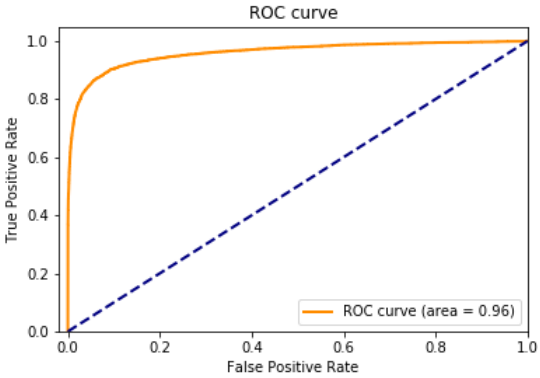

Figure 5.12: ROC for ResNet

The figure 5.13 shows the feature importance on the model. $GID84$, $GID98$, $GID769$ are the most important features, which is the combination result of DRF and GBM.

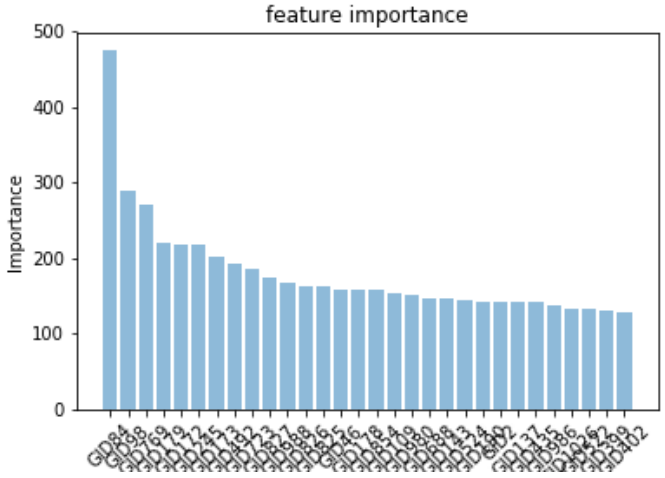

Figure 5.13: Variable importance trained on ResNet

The experiments using different algorithms and packages with finalized model is described. Comparing those models, the model using GBM reaches slightly higher statistics on TPR, accuracy and AUC on testing data, and either H2O and XGBoost can achieve the

comparatively similar result. It is not surprising to see that ResNet can improve a network with higher accuracy from simple network.

5.3 Analysis summary

In the previous section, the results of data process and examined experiments are presented, which the objectives were satisfied, but some limits existed.

For the section of data cleaning and analysis, the main emphasis was placed to get basic understanding of different features, and their distribution. The data were split into training, validation and test data.

In the experimental section, three main algorithms were used. DRF, GBM and ResNet. The experiments were separated into four main part – DRF on H2O, GBM on H2O, GBM on XGBoost, and ResNet on Tensorflow. The analysis exposed that the performance of optimized model on each algorithm are close to each other. Comparatively, GBM performed better. The best models are significantly better than the baseline models on accuracy. The feature importance is displayed on each optimized model. $GID84$ is the important feature that rank on the top of each algorithm. On each optimized model, around 30 to 50 features are selected as important feature. Even the data has 1033 features, the very important features can be restricted to less than 100.

CHAPTER 6

Conclusion and recommendation

6.1 Recommendation

Data is the basement of the analysis. Real data with high quality is more meaningful to explain the result. To make better analysis, the reduction of missing values and constant columns is required. Analysis of feature importance showed that the time feature does not have impact on the prediction analysis, and only few features are truly important for the modeling. The next step of analysis can get more data so that feature extraction can be used to get more information on the testing process, and removal of non-informative features is required, which can improve the modeling accuracy.

The analysis showed how the prediction can be properly match the expected result based on the right model or procedures. Even though GBM performed well on the experiment, it does not mean GBM is the only method to be used on training HDDs data. To get more powerful result, trying other algorithms or packages is another way of improving model.

6.2 Conclusion

This thesis was focused on prediction of HDDs data with emphasis on testing different approaches to get better statistics. In the first part of the thesis, theoretical background was introduced, and concepts of machine learning was described. The main goal was to understand how accurate those supervised learning approaches with HDDs data can achieve. After problem definition and data cleaning, the data is prepared to the modeling step.

Despite the machine for training models has limitation on speed, the result can be used to

understand and predict the HDDs passfail on testing process. The predictive models based on DRF, GBM and ResNet were trained and tested. Even though the experiments show the best approach is to use GBM, the tiny difference among the optimized models indicates that with any of approaches and correct optimization, the training models can be improved, especially the ResNet. TensorFlow used for NNs has more flexibility of customizing the structure of networks. H2O and XGBoost has already implemented very powerful machine learning algorithms, so the model improvements by parameter tunning may not be so obvious. Feature importance shows that the good prediction is due to some of the features and remaining feature are not important on the modeling process. To reduce the time for future model training, some feature can be removed.

As mentioned before, the data only includes two weeks of disk test data, and the next step of further research can be see whether the optimized models are stable under different time ranges and see any date and time patterns influences to the model performances. If data becomes voluminous, GPU or other way of acceleration on modeling would make the research more time efficient and powerful.

REFERENCES

[1] Abadi et al.(2015). TensorFlow: Large-scale machine learning on heterogeneous systems. *https://www.tensorflow.org/*.

[2] Benyamin, D.(2012). A Gentle introduction to random forests, ensembles, and performance metrics in a commercial system. *http://blog.citizennet.com/blog/2012/11/10/random-forests-ensembles-and-performance-metrics*. Copyright 2018 CitizenNet Inc.

[3] Biau, G.(2012). Analysis of a random forests model. *Journal of Machine Learning Research*, 13:10631095.

[4] Breiman, L.(1996). Bagging predictors. *Machine Learning*, 24(2): 123140.

[5] Breiman, L., Jerome H.F., Richard A.O., Charles L.S.(1984) Classification and regression trees. Belmont, CA: Wadsworth International Group.

[6] Breiman, L.(2001) Statistical modeling: The two cultures, 16 *Statistical Science*, 1999.

[7] Breiman, L.(2001). Random forests. *Machine Learning*, 45(1): 4-32.

[8] Breiman, L.(1999). Prediction games and arcing algorithms. *Journal Neural Computation*. 11(7). 1493 - 1517.

[9] Chen, T., Guestrin, C.(2016). XGBoost: A scalable tree boosting system. arXiv:160302754 [cs]. 785-94.

[10] Denil, M., Matheson, D., Freitas N. Narrowing the gap: random forests in theory and in practice. *https://arxiv.org/pdf/1310.1415.pdf*.

[11] Freund, Y., Schapire, R.(1994). A decision−theoretic generalization of on-line learning and application to boosting. *European Conference on Computational Learning Theory*.

[12] Friedman, J.(2000). Greedy function approximation: A gradient boosting machine *Annals of Statistics* 29: 1189–1232,

[13] H2O.ai. *http://docs.h2o.ai/h2o/latest-stable/h2o-docs/welcome.html* Copyright 2016-2017 H2O.ai.

[14] H2O.ai. random_forest.py. *https://github.com/h2oai/h2o-3/blob/master/h2o-py/h2o/estimators/random_forest.py*. Copyright 2017.

[15] H2O.ai. gbm.py. *https://github.com/h2oai/h2o-3/blob/master/h2o-py/h2o/estimators/gbm.py*. Copyright 2017.

[16] Hansen L., Salamon P.(1990). Neural network ensembls.*IEEE Transaction on Pattern Analysis and Machine Intelligence*, 12, 993-1001 10.1109/34.5887.

[17] Haber, E., Ruthotto, L.(2017). Stable architectures for deep neural networks. *arXiv preprint arxiv:1705.03341*.

[18] He, K., Zhang, X., Ren, S., Sun, J.(2016). Deep residual learning for image recognition. *Conference on Computer Vision and Pattern Recognition*.

[19] He, Z. M., Yang, H., Xie, M.(2012) Statistical modeling and analysis of hard disk drives (HDDs) failure. *APMRC, 2012 Digest*. 13228475.

[20] Huang, G., Sun, Y., Liu, Z., Sedra, D., Weinberger, K. Q.(2016). Deep networks with stochastic depth. *European Conference on Computer Vision*.

[21] Kulkarni, V., Sinha, P.(2014). Effective learning and classification using random forest algorithm. *International Journal of Engineering and Innovative Technology (IJEIT)* 3(11).

[22] LeDell, E. Scalable ensemble learning and computationally efficient variance estimation *https://www.stat.berkeley.edu/ ledell/papers/ledell-phd-thesis.pdf*.

[23] Louppe, G.(2014) Understanding random forests: from theory to practice. arXiv:1407.7502.

[24] Rosenblatt, F.(1958). The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*. 65 – 386.

[25] Rumelhart, D., Hinton, G., Williams, R.(1988). Learning representations by back-propagating errors. *Neurocomputing: foundation of research*. 696 – 699

[26] Wakayama et al. (2015). Distributed forests for MayReduce-based machine learning *Asian Conference on Pattern Recognition*.

[27] XGBoost. XGBoost Parameters *http://xgboost.readthedocs.io/en/latest /parameter.html*. Copyright 2015 – 2016.

[28] Xie, S., Girshick, R., Dollár, P., Tu,Z., and He,K.(2017). Aggregated residual transformations for deep neural network. *Conference on Computer Vision and Pattern Recognition*.