Predicting the Formation of a Thermal Asperity Detector

Machine Learning Analysis on Manufacturing and Electrical Test Data

Erik J. Hutchinson
SEIS
University of St. Thomas
St. Paul, MN, USA
hutc7210@stthomas.edu

Monique Dargis
SEIS
University of St. Thomas
St. Paul, MN, USA
monique.dargis@stthomas.edu

Connor Mills
SEIS
University of St. Thomas
St. Paul, MN, USA
connor.mills@stthomas.edu

Derek Synan
SEIS
University of St. Thomas
St. Paul, MN, USA
syna4710@stthomas.edu

ABSTRACT

A read/write transducer is a mechanism that reads and writes magnetic information within a spinning magnetic hard disk. A read/write transducer can take upwards of one year to manufacture and within it is a thermal asperity detector that measures electrical resistance. The need to predict the detector's final electrical resistance in relation to its target is critical to the manufacturer's ability to produce a viable product.

This project explores various models that predict the final resistance of a thermal asperity detector within a read/write transducer. The dataset provided contained over 100,000 instances and a numerical target, which was also classified into a binary pass/fail outcome. The features of the dataset were comprised of both numerical and categorical variables. One hot encoding was applied to the categorical variables and resulted in a dataset with a very large number of features. Because of this, several dimensionality reduction techniques were attempted, including forms of both feature selection and feature reduction. Models predicting both numerical values for resistance and pass/fail success rate for resistance were explored.

In general, the analysis models all required a significant number of features to achieve relatively high model accuracy. The number of features significantly hampers the analysis and a further reduced set of features would need to be considered based upon engineer knowledge.

KEYWORDS

Thermal Asperity Detector, Machine Learning, Principal Component Analysis, Linear Regression, Naïve-Bayes, Decision Tree Regression, Random Forrest Regression, Logistical Regression

ACM Reference format:

Erik J. Hutchinson, Monique Dargis, Connor Mills, and Derek Synan. 2019. Predicting the Formation of a Thermal Asperity Detector: Machine Learning Analysis on Manufacturing and Electrical Test Data. In *SEIS763 Class Presentation*, *University of St. Thomas, Graduate Program Studies, St. Paul, MN, USA.*

1. INTRODUCTION

Inside of a spinning magnetic hard drive is a read/write transducer that is responsible for reading and writing information. As the read sensor passes over the disk in a particular location at a specified distance above the disk, the senor reads magnetic bits by sensing the magnetic pole of the bit. A one or zero is recorded for the differing magnetic poles. Writing magnetic bits occurs when the magnetic writer pulses a magnetic field strong enough to overcome the disk's bit magnetization. Again, a one or zero is written depending on the pulse.

A read/write transducer must fly a specified distance above a magnetic disk to have its optimal electrical and magnetic performance. The spacing between the read/write transducer and the magnetic hard disk is less than one nanometer in most hard drives today. This spacing is controlled by a process called, "Active Fly," whereby heat is provided within the transducer that causes the read/write transducer to fly closer to the hard disk surface. Consequently, the reliability of the read/write transducer is also paramount. During the magnetic hard disk manufacture, it is possible for thermal asperities to be formed on the disk surface. These asperities can be anywhere from 10 Angstroms to seven nanometers tall. These asperities must be avoided if all possible otherwise they can damage the read/write transducer and make it inoperable. Thus, a device called a "Thermal Asperity Detector," was created (also called TAD).

The second primary function of the thermal asperity detector is to determine how fast the read/write transducer approaches the disk surface and when the read/write transducer contacts the hard disk surface. Knowing how fast the transducer approaches the hard disk surface allows the drive code to determine how much power is required to maintain a specified target distance. Additionally, the drive code is able to know that the read/write transducer is at zero fly height as its reference point.

A thermal asperity detector (TAD) is a simple electrical resistance device that depends on its metal sensor's thermal coefficient of resistivity (TCR). When the sensor contacts the disk surface, a friction event occurs which subsequently causes the sensor's

resistance to rise. On the other hand, as the sensor is moved towards the disk surface by Active Fly, its resistance lowers as the air passing over it compresses (convection event). These two behaviors tell the drive code how fast the transducer is approaching the disk surface and when it makes contact. Thus, the targeted resistance of the thermal asperity detector is important since it forms the starting point for the operations.

With these key pieces of information in mind, the manufacture of a TAD within a read/write transducer takes months to build from start to finish. Tens of thousands of read/write transducers are manufactured on a single wafer. The TAD's sensor is formed with a single metal with a relatively high TCR value and it is printed with a single photolithographic mask and ion mill operation. However, its final dimension is not initially complete until the wafer is divided into bars and then into the final read/write transducer. Along the way, several measurements and different manufacturing tools are used. At the final point of manufacturing, the TAD resistance is measured and compared against the required specification. The goal thus is to potentially predict the final resistance and best manufacturing path to obtain a pass/fail condition for the thermal asperity detector.

1.1 Experimental Design

A sample of twelve weeks' worth of electrical test data was obtained from Seagate Technology. This data was joined with its manufacturing measurement and tool data that spans up to one year before the electrical test data. This data was obtained doing SQL queries and converted into CSV files. A single CSV file was created for the sample dataset.

The data has been altered to a degree to protect the exact manufacturing process of the TAD. Not all of the full manufacturing data was included in the sample data set. However, enough data was obtained to provide an initial dataset of 107,000 instances across 28 columns.

1.2 Data Preparation

The sample data was prepared through a variety of simple techniques. As a first pass, any data rows that had missing data were eliminated. Secondly, any outliers greater than four standard deviations were eliminated for the various measurement techniques based upon the data source policies. This was done to ensure that mismeasurements did not adversely impact the final models. Finally, there were nine (9) features from the dataset that were ignored because they were included in the dataset for information purposes but do not have any impact on processing. Those columns were: WFR_X_UM, WFR_Y_UM, WFR_X_IN_CUBE, WFR_Y_IN_CUBE, BAR_ID, HEAD_ID, WAFER_ID, and PROD_CODE. The final instance count for the remaining columns after the data was prepared was greater than 88,000. The primary target for the data was HGA_RES. This column contained the measured electrical resistance of the TAD in the fully formed read/write transducer.

The data also contained a secondary categorical target, "HGA_PF." This column contained an integer value that designated if the "HGA_RES" passed or failed the design specification. If HGA_RES was less than 78 or greater than 100 ohms, then the instance was labeled "0", indicating a failure. Conversely, a passing instance with resistance between 78 and 100 ohms, inclusive, was labeled "1".

In the dataset, there were several columns that contained categorical variables. These columns were one hot encoded using the "get_dummies" command from the pandas API. The original dataset contained 26 features and two targets (numerical and binary pass/fail). After the original nine columns were eliminated, the dataset contained 17 feature columns and two targets. After one hot encoding, the feature count increased to 389.

Any other modifications to the data were dependent on the analysis to be performed. Each will be discussed in their relevant section.

1.3 Data Dictionary

Data File Column ID = Description - Data Type:

- WFR_X_UM = Transducer location on the Wafer in the X direction (unit is microns) – numerical
- WFR_Y_UM = Transducer location on the Wafer in the Y direction (unit is microns) - numerical
- WFR_X_IN_CUBE = Transducer column location within a photocube - ordinal
- WFR_Y_IN_CUBE = Transducer row location within a photocube - ordinal
- BAR_ID = Two digit bar id – character
- HEAD_ID = Two digit transducer head id – character
- WAFER_ID = Five digit wafer id – character
- PROD_CODE = Two digit wafer id family (wafer id's make up PROD) – character
- PHOTO_TOOL = Tool where photomask is applied – character
- PHOTOMASK = Eight digit id that identifies a photomask – character
- OVL-Y_TOOL = Vertical photomask alignment on the wafer tool – character
- OVL-Y_DELTA = Vertical photomask alignment measurement delta between measured and target value in microns– numerical
- SEM_TOOL = Linewidth measurement tool – character
- SEM_DELTA = Linewidth measurement delta between measured and target value in microns – numerical
- PROM_TOOL = Metal sheet resistance measurement tool – character
- PROM_DELTA = Metal sheet resistance measurement delta between measured and target value in ohms/square – numerical
- XRF_TOOL = Metal thickness measurement tool – character
- XRF_DELTA = Metal thickness measurement delta between measured and target value in nanometers – numerical

- MILL_TOOL = Ion mill tool – character
- WAFER_TAD_RES = Linewidth electrical resistance on the Wafer in ohms – numerical
- ISI_TESTER = Electrical test tool after Wafer is sliced into Bars – character
- ESTBP = Estimated lap distance in nanometers – numerical
- ELG_SH_DELTA = Lap resistance delta measured between lap guide and transducer in ohms – numerical
- LAP_TOOL = Lapping tool – character
- HGA_TESTER = Electrical tester for individual transducers as they fly above a magnetic disk – character
- HGA_RES = Final TAD resistance in ohms – numerical & the **target** value
- HGA_PF = HGA_RES in Pass/Fail Condition: 0 = Fail / 1 = Pass - character

2. MODEL TRAINING FOR NUMERIC TARGET

2.1 Multiple Linear Regression

2.1.1 Preparation for Multiple Linear Regression Because there is a numerical target value available for the TAD resistance value, several multiple linear regression models were trained on the data. Scikit-learn's Multiple Linear Regression object was used to train these models.

An attempt was made to train these models using scikit-learn's k-fold validation features in order to decrease the likelihood that the model might be influenced by the train/test split of the data. However, due to limitations within scikit-learn, k-fold validation failed to produce useable results. Instead, a simple train/test split was used to train the multiple linear regression model on 70% of the data and reserve 30% of the data for testing purposes. After a test/train split was performed, the features on both sides of the split were normalized to reduce compute time for the linear regression model.

Dimension Reduction Technique	Number Features/Components Included	Mean Squared Error
None	389	$8.38*10^{25}$
Stepwise Selection	171	11.78
PCA	335	17.18

Table 1. Results of Multiple Linear Regression models after various dimension reduction techniques .

2.1.2 Multiple Linear Regression with all features A multiple linear regression model (MLR) was first trained on all 389 features to establish a baseline Mean Squared Error scores Results of the model are reported in Table 1, and a visual representation of the predicted target value versus the actual target value can be found in Figure 1.

Because the results of the MLR model with all features included resulted in an extremely poor model for predicting the level of TAD resistance, attempts were made to reduce the number of features included in the model.

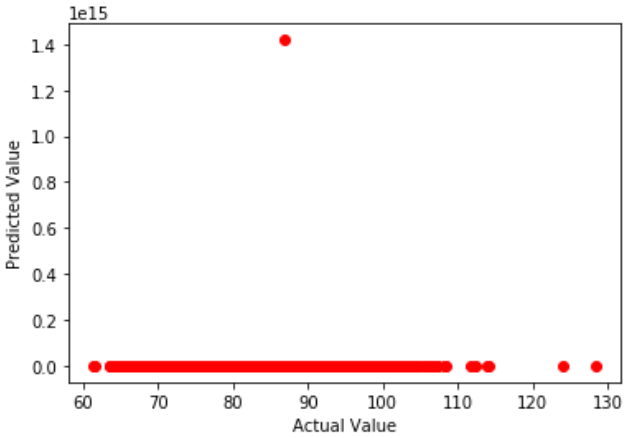


Figure 1. Visualization of Multiple Linear Regression model with all features included.

2.1.3 Stepwise selection First, a stepwise selection algorithm was run on the features of the model. This algorithm was based on the statsmodel python package and iterated through a forward selection/backward elimination loop. Because of the large number of features, the threshold to be included in the model was set at $p < .01$. The threshold for features within the model to be eliminated was set at $p < .05$. This algorithm reduced the number of features to be used in the MLR model to only 171.

Using the same test/train split as before, an MLR model was trained using only the 171 features included after stepwise elimination. The coding of the stepwise selection model required that data in a Pandas dataframe be fed into the model. For this reason, the non-normalized training data was fed into the stepwise selection model. After feature reduction in both the training and test datasets, normalization was performed.

The Mean Squared Error score for the post-stepwise selection model can be seen in Table 1, and a visual representation of the predicted target value versus the actual target value can be found in Figure 2. Reducing the number of features to 171 did result in a much better model than when all features were included.

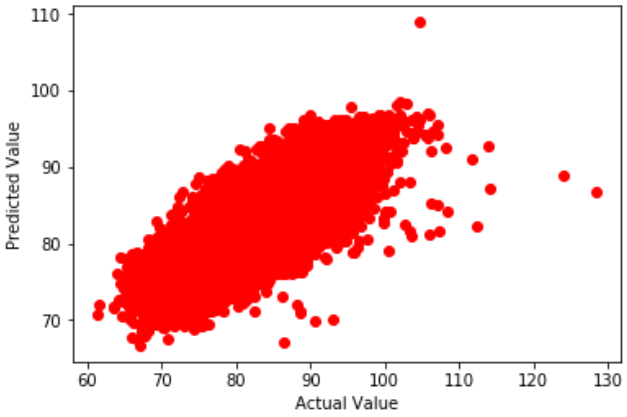


Figure 2. Visualization of Multiple Linear Regression model after number of features reduced through stepwise selection.

2.1.4 Principal Component Analysis Next, Principal Component Analysis (PCA) was used to transform the data and reduce the number of features included in the MLR model. As can be seen in Figure 3, no one principle component explained more than 2.5% of the variance in the dataset, and all but two of the components explained less than 1% of the variance.

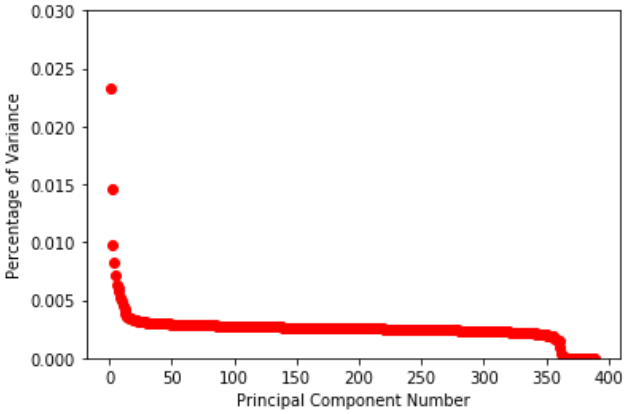


Figure 3. Visualization of Principle Component Analysis of all features in the dataset.

PCA revealed that the first 335 principal components explained 95% of the variance in the dataset, so this reduced set of components was used to train a new multiple linear regression model. The Mean Squared Error score for the post-PCA MLR model can be seen in Table 1, and a visual representation of the predicted target value versus the actual target value can be found in Figure 4. Reducing the number of transformed features to 335 improved the predictive power of the model versus using all of the features. However, the post-PCA MLR model still performed much worse than the post-stepwise selection MLR model.

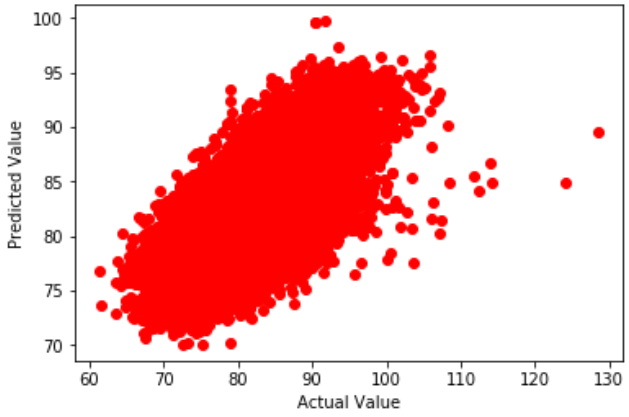


Figure 4. Visualization of Multiple Linear Regression model after number of features reduced using Principal Components Analysis.

2.1.5 Other forms of dimension reduction Kernel-PCA analysis was also attempted on both a local machine and a cloud-based virtual machine. However, because of the large size of the dataset, computing power was insufficient, and Kernel-PCA could not be performed.

2.1.6 Analysis From the results of the three multiple linear regression models generated, it is evident that reducing the number of features included in the model improves model accuracy. However, even after stepwise selection and with only 171 features included, the predictive power of the multiple linear regression model is relatively low, with an R2 value of only .48.

2.2 Decision Tree Regression

2.2.1 Preparation for Decision Tree Regression Running a decision tree regressor on the data was a straightforward process. A dataset that included columns resulting from one hot encoding on PHOTO_CODE was used for this model; however, the researchers had knowledge of the product and knew that PHOTO_CODE would not have predictive power in the model. Because of this, the decision was made to proceed with this larger dataset of 294 features. Scikit-learn’s train_test_split module was again used to separate 70% of the data for model training and 30% for testing. After the split, the test and train features of the data were normalized.

2.2.2 Decision Tree Regression without Dimension Reduction The continuous numerical target value was used to train the decision tree regression, and model performance was evaluated using mean squared error technique. Mean squared error (MSE) is computer by comparing the predicted target values for the test portion of the data to the actual target values for that same data. Most of the default values in the DecisionTreeRegressor module were left as-is, evaluating MSE as the criterion with no maximum features or leaf nodes. This model resulted in a mean squared error of 19.01 when test data predictions were evaluated.

2.2.3 Decision Tree Principal Component Analysis The next step in evaluating the decision tree models was to apply principal component analysis to determine whether this feature reduction model would improve the predictive power of the model. In order to do this, all of the same preprocessing steps were applied as the standard decision tree model. Then, the PCA module was imported and applied to the train and test datasets. 7 models with 10, 20, 30, 40, 50, 100, 200, and 394 components were run to evaluate the effect of number of features on mean squared error.

MSEs of the various post-PCA decision tree models can be found in Figure 6. The highest Mean Squared Error resulted from having only 20 features reduced from 394. This reduction resulted in a mean squared error value of 26.7, a worse result than the standard decision tree model run prior.

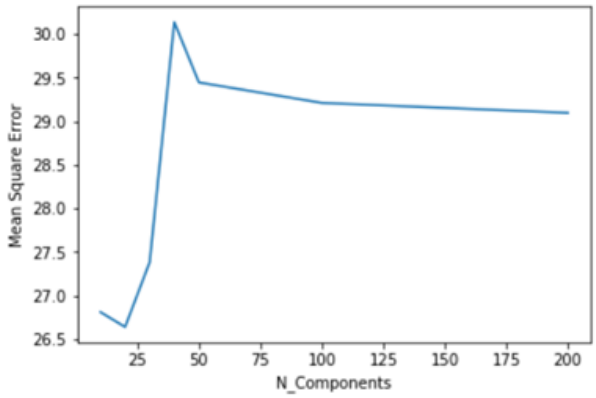


Figure 6. Visualization of the impact of n_components value on Mean Squared Error after Principal Components Analysis

2.2.4 Decision Tree Linear Discriminant Analysis In addition to principal component analysis, linear discriminant analysis was applied to the same data to evaluate the effect on Mean Squared Error. Just as with PCA, the hyperparameter that was chosen to fine tune the model was the n_components hyperparameter, and values used were again 10, 20, 30, 40, 50, 100, 200, and 394 components. Results can be seen in Figure 7.

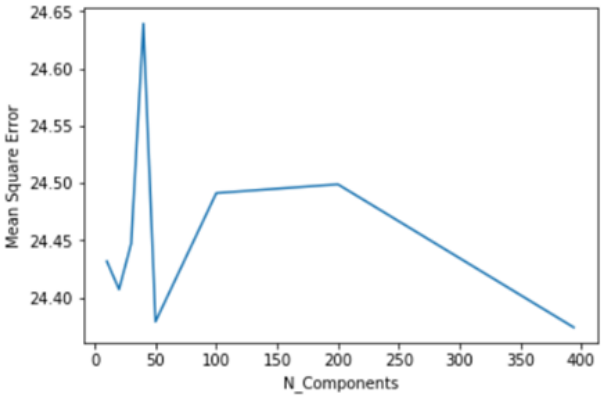


Figure 7. Visualization of effect of n_components on Mean Squared Error after Linear Discriminant Analysis

Of the post-LDA models, the model with 50 components resulted in the smallest Mean Squared Error value and is the ideal number of components to use with this data when applying LDA. However, with an MSE value of 24.35, the predictive power of the model is still less than that of a model run without performing LDA.

2.2.5 Analysis Table 2 tabulates the best results of the decision tree models generated after various feature extraction techniques were applied. Both principal component analysis and linear discriminant analysis resulted in models that were less accurate at predicting the target value than a simple Decision Tree regression with no feature extraction. It can be concluded that, for this dataset, performing feature extraction techniques before training a Decision Tree model provides no positive benefits.

Feature Extraction Technique	Number Features/Components included	Mean Squared Error
None	394	19.01
PCA	20	26.7
LDA	50	24.35

Table 2. Most accurate Decision Tree models after various feature extraction techniques are applied to the dataset.

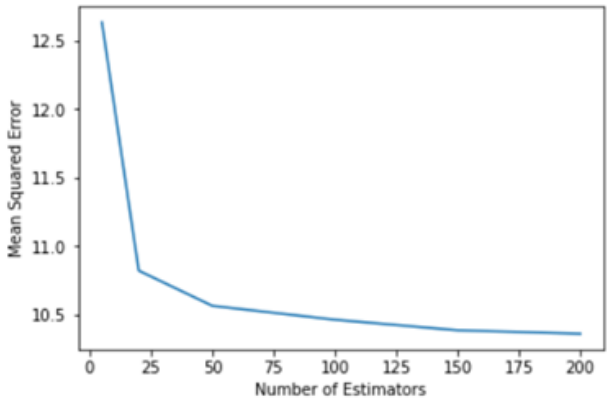
2.3 Random Forest Regression

2.3.1 Preparation for Random Forest Regression For the first random forest implementation, the procedures from the standard decision tree regression were replicated. This included using a dataset that included PHOTO_CODE, using scikit-learn's train/test split with 30% of data reserved for testing, and normalizing data after the test/train split. Scikitlearn's RandomForestRegressor module was then used to train a regression models, and mean squared error was used to evaluate the performance of these models.

2.3.2 Random Forest Regression without Dimension Reduction In order to refine the model as much as possible, a grid search technique was applied on the n_estimators hyperparameter within RandomForestRegressor. An array of n_estimators containing n = [5, 20, 50, 100, 150, 200] was defined and a "for" loop was created to re-run the random forest regression with the chosen estimators. The same loop then saved the estimator value used and the results of a Mean Squared Error calculation into separate arrays.

These saved arrays were used to visualize the results of the grid search and to choose a suitable n_estimators value for running the model. Figure 8 shows the values of the Mean Squared Error as a function of the number of estimators, and it is worth noting that

the range of difference between the highest and lowest MSE



generated is a little over 2.0.

Figure 8. Visualization of Estimator vs MSE on a random forest regression model

Moving from 5 estimators to 20 showed the largest improvement over any other shift, which makes sense as the complexity of the model grows exponentially as the estimator count is increased. As the estimator value was raised, the model continued to show signs of improvement until Mean Squared Error reached 10.36 with 200 estimators. However, diminishing returns became an issue as marginal error improvements and more expensive computational requirements came into play. Given the information discovered through this search, it is reasonable to conclude that 50 is the ideal number of estimators for this model. With 50 estimators, there was a suitable balance of model complexity and fitment runtime and low MSE of 10.57.

Although there are several other hyperparameters that can be evaluated to further tune random forest decision tree model, the relatively low Mean Squared values and the relative success of the random forest regression model over the other regression models led the researchers to move away from hyperparameter selection. Instead, effort was put toward dimension reduction with the hope of further improving the accuracy of the model.

2.3.3 Random Forest after Principal Component Analysis When applying PCA to the random forest model, there were two hyperparameters to evaluate. The first was number of components for the principal component analysis object to reduce our original data into. The second, was number of estimators in the random forest model. A nested for loop was implemented to evaluate a variety of possible combinations of component and estimator count. This took a fair amount of processing time as there were 7 values in each loop which resulted in the fitment of (7x7) 49 models total.

Once those models were run, the parameters and results were stored in array form. By extracting the index location of the minimum value in the Mean Squared Error array, it is possible to pull the values for the parameters used on that specific model. This method turned out to be very computationally expensive and ran for almost two hours before completing. Results can be found in Figures 9 and 10.

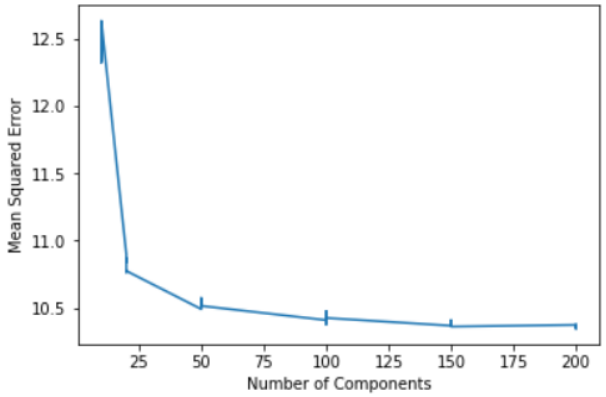


Figure 9. Visualization of Number of Components vs MSE when applying PCA on random forest regression model

After running all 49 models, the model with the lowest Mean Squared Error included 200 components and used 50 estimators in the random forest. The Mean Squared Error of that model was 10.34, which is better than any previously-generated regression model.

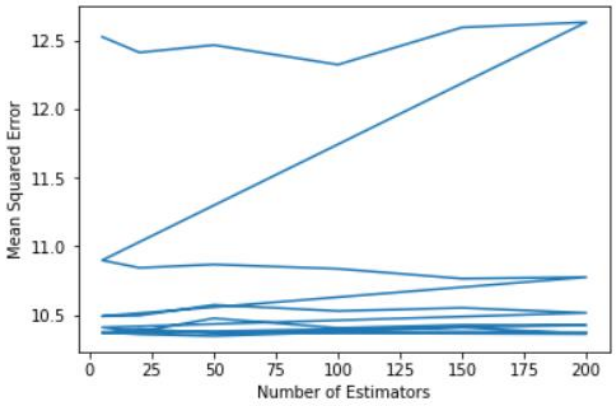


Figure 10. Visualization of Number of Estimator vs MSE when applying PCA on random forest regression model

2.3.4 Random Forest after Linear Discriminant Analysis The last random forest model attempted was a random forest model with Linear Discriminant Analysis implemented. Mean Squared Error scores displayed as a function of the number of components can be found in Figure 11, and MSE scores as a function of the number of estimators in the model can be found in Figure 12.

The best model when applying LDA included 100 components, used only 5 estimators, and resulted in an MSE of 11.58, lower than the best PCA random forest regression models. The LDA random forest regression model was similar to the PCA implementation in that it was extremely computationally

expensive to run. For this reason, the components parameter tested was capped at 100. However, results show that as the number of components used increases, the MSE appears to decrease. It is possible that, were greater computing power available and a larger number of components were used, the MSE might approach the MSE of the PCA random forest model.

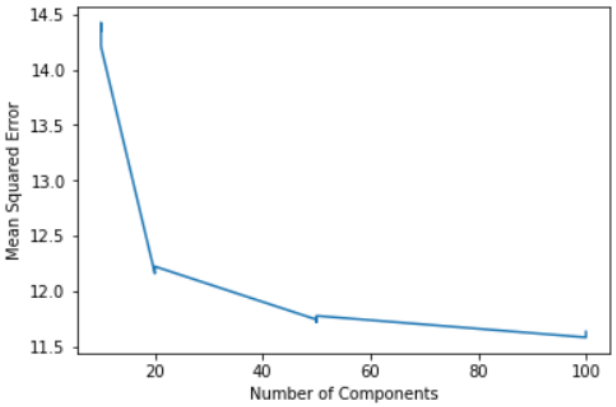


Figure 11. Visualization of Number of Components vs MSE when applying LDA on random forest regression model

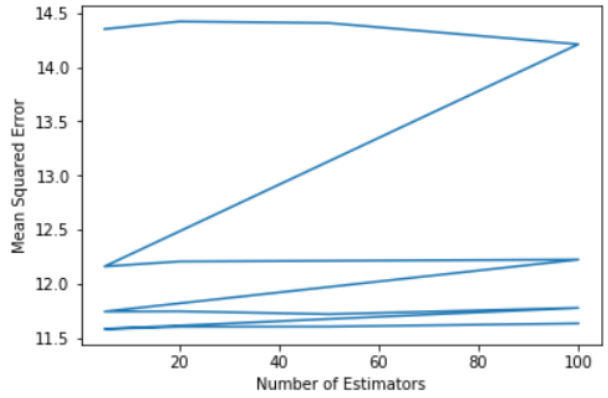


Figure 12. Visualization of Number of Estimators vs MSE when applying LDA on random forest regression model

2.3.5 Analysis Table 3 tabulates the best results of the random forest regression models generated after various feature extraction techniques were applied and components and estimators selected. The Mean Squared Error difference between the random forest regression model trained on all features and the best model after applying principal component analysis is so low as to be negligible. However, the model trained on all features required four times the estimators to obtain this result. When the MSE random forest regression model trained on the full set of features with only 50 estimators is taken into account (at an MSE score of 10.57), it is evident that applying PCA is necessary to reduce the number of estimators and thereby reduce computing costs.

Feature Extraction Technique	Number Features/Components included	Number of Estimators	Mean Squared Error
None	394	200	10.36
PCA	200	50	10.34
LDA	100	5	11.58

Table 3 Most accurate Random Forest regression models after various feature extraction techniques are applied to the dataset.

3. MODEL TRAINING FOR CATEGORICAL TARGET

3.1 Logistic Regression

3.1.1 Preparation for Logistic Regression Several logistic regression modeles were trained using the pass/fail target. The Logistic Regression object was imported from scikit-learn, and the k-fold cross validation feature of sci-kit learn was successfully used to minimize the risk that a randomly-selected training subsection of the data might result in an inaccurate model. All logistic regression models were run using normalized data and 10 folds, and model accuracy scores were generated based on the rate of true positive and true negative predictions. No separate test-train split was used; rather, the whole dataset was used to train and test the model using k-fold validation.

3.1.2 Logistic regression with all features As with the multiple linear regression model procedure, a logistic regression model was first fitted using all features of the dataset to provide a baseline accuracy rating. Features were normalized before model training to reduce compute time. Model accuracy mean and standard deviation can be found in Table 4 and show a relatively high model accuracy rate.

Dimension Reduction Technique	Number Features included	Model Accuracy Mean	Model Accuracy Standard Deviation
None	389	0.84	0.06
Stepwise Selection	171	0.85	0.05
PCA	336	0.82	0.06

Table 4. Results of Logistic Regression models after various dimension reduction techniques are applied to the dataset.

3.1.3 Stepwise selection Next, stepwise selection was performed on the dataset. The same algorithm as was used for multiple linear regression, with the same hyperparameters, was used to reduce the number of features. Because no test/train split was used for logistic regression, the entire dataset was fed into the stepwise selection algorithm. Again, non-normalized data was fed into the stepwise selection algorithm because of requirements of the code.

The result of stepwise selection was a set of 171 features to be used to train a logistic regression model. After reducing the number of features in the dataset, the data was normalized in preparation for training a logistic regression model using k-fold validation. Results of this model can be found in Table 2.

3.1.4 Principal Component Analysis Principal Component Analysis was again used to transform the data and reduce the number of features included in the logistic regression model. Whereas PCA was completed on only the training portion of the dataset for multiple linear regression, use of k-fold validation allowed PCA to be completed on the entire dataset for use with logistic regression.

Results of PCA on the entire dataset, which can be seen in Figure 13, were extremely similar to results of PCA performed on only a training subsection of the data. As before, only two components account for more than 1% of the variance in the dataset. The only slight variation in results is that 336 components accounted for 95% of the variance in the entire dataset, rather than 335 components in the training subset.

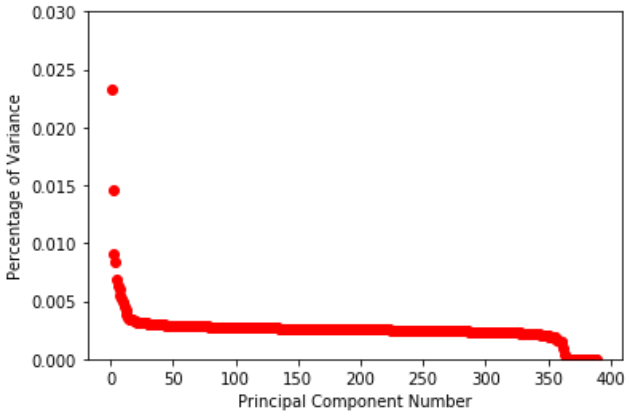


Figure 13 Visualization of Principle Component Analysis of all features in the dataset.

Based on the results of PCA, a reduced set of 336 components was used to train a new logistic regression model. The accuracy results of the post-PCA logistic regression model can be seen in Table 2. Reducing the number of transformed features to 336 reduced the accuracy of the model as compared to both the model with all features included and the model generated after reducing features through stepwise selection.

3.1.5 Other forms of dimension reduction Linear discriminant analysis was attempted to reduce the number of features included in the logistic regression model. However, high collinearity of variables made linear discriminant analysis impossible.

As with multiple linear regression, kernel-PCA analysis was attempted on both a local machine and a cloud-based virtual machine. Again, computing power was insufficient for the size of the dataset.

3.1.6 Analysis The results of the three logistic regression models successfully generated show again that stepwise elimination yields the highest model accuracy. However, the difference in model accuracy between the worst model, the post-PCA model with an accuracy of 0.81, and the best model, the post-stepwise selection model with an accuracy of 0.84, is very small.

3.2 K-Nearest Neighbors

3.2.1 Preparation for K-Nearest Neighbors As with decision tree regression and random forest, regression, the 394-feature dataset that included PHOTO_CODE. Again, scikit-learn's train/test split was utilized with 30% of data reserved for testing. Data in both the triin and test splits were also normalized to avoid the magnitude of one tool's measurements dominating the other features and skewing the grouping of data points in this K-NN model.

With close to 400 features, it was determined that no model would be trained on all of the features of the dataset. Instead, dimension reduction was used to pare the number of components to a more manageable size before a K-NN model was trained.

3.2.2 Principal Component Analysis In order to work on narrowing down features in the K-NN model, Principal Component Analysis (PCA) dimension reduction was applied after normalizing the data but before cross validating different values of K. After reviewing the array of the explained variance ratios provided by the PCA function from the scikit-learn Python library, the PCA function was standardized to self-select the required number of components to be used in the model, based on a 95% threshold for the cumulative explained variance ration. Essentially, the PCA function included elements or features from the dataset into the model until 95% of the variance in the data was explained. In the case of K-NN, 333 of the original 389 features needed to be included to reach the 95% explained variance threshold set.

3.2.2 Determining K value The next step in the KNN model, was to determine the hyperparameter value of K to be used in the model. The hyperparameter value of K, sets the number of instances in the training data that are closest to the new instance (new data coming into the model) that will be evaluated when classifying that instance into a class. In the TAD dataset, the classification classes are 0 (not within expected range of values and fails the quality process) and 1 (within expected range of values and passes the quality process).

In order to narrow down the possible values of K to be used in the model, cross validation was performed using 4 different

possible values of K (5, 10, 15, 20) and then fitting the model with all of them. The cross validation was done using the GridSearchCV function from the Sklearn Python Library. The scoring provided by that function is accuracy (% of correct classifications on the training data). Based on the accuracies results of all four possible values of K, K=15 was ultimately used in the final model, as its accuracy was less than 0.01 different than K = 20 (see Table 5). Expanding the possible values of K beyond the four noted was not attempted, due to processing time and ultimately because the overall accuracy of the model at this point was satisfactory.

K Value	Model Accuracy	Splits
5	0.78	5
10	0.78	5
15	0.79	5
20	0.79	5

Table 5. GridSearchCV results for finding best K value to use in K-NN model

To complete the model and get an end accuracy result, K-Fold cross validation was completed on the training set of data. The K-Fold validation was completed by using the cross_val_score function from the scikit-learn Python Module. Within that function, the estimator (type of model) was set to a KNN classifier, where K=15, and the function was set to use a 5-fold cross-validation strategy. The resulting accuracy score from the KNN model described in all the previous steps was 0.79.

3.2.3 Other forms of dimension reduction Other methods of dimension reduction were explored, such as Linear Discriminant Analysis and Kernel PCA, however unique issues were encountered that prevented use of additional dimension reduction techniques. LDA could not be utilized as a dimension reduction methodology, because there was multicollinearity identified by the Python function. Due to the large number of features in our dataset, no further time was invested into LDA.

Kernel PCA could not be utilized because neither local machines nor virtual machines instances within Google cloud datalab (at an affordable processing power level) solution would complete or even start the Kernel PCA process. The errors received were due to memory shortage, presumably because of the number of features included in our dataset.

3.3 Support Vector Machine

3.3.1 Preparation for Support Vector Machine The preliminary data cleaning and preparatory steps in the SVM model were similar to the steps applied against the K-Nearest Neighbors (K-NN) model. This included using a dataset that included PHOTO_CODE, using scikit-learn's train/test split with 30% of data reserved for testing, and normalizing data after the test/train split.

As with K-NN, it was determined that Principal Component Analysis should be applied before any attempt was made to train an SVM. PCA results showed that to reach 95% of the variance explained, 333 of the original 389 features. This set of 333 principal components was used to train the SVM model.

3.3.2 SVM Kernel selection The SVM algorithm in scikit-learn (referred to within that module as SVC) function includes a hyperparameter for what type of kernel the algorithm should use. In order to validate the kernel best fit our modeling needs, a cross validation using GridSearch was performed on the kernel variable. The four kernel values used in cross validation were linear, poly, rbf, and sigmoid.

Model accuracy was used to determine the SVM kernel. During this phase of cross validation, both RBF and Linear kernels produced accuracies that differed by less than 0.01, as shown in Figure 12. The RBF Kernel was ultimately chosen for further model training.

Kernel	Model Accuracy Mean	Splits
RBF	0.84	5
Poly	0.82	5
Linear	0.84	5
Sigmoid	0.78	5

Figure 12. GridSearchCV results for finding best kernel to use in SVM model

3.3.3 Support Vector Machine To complete the SVM model and get an end accuracy result, K-Fold cross validation was completed on the training data. The K-Fold validation was completed by using the cross_val_score function from the Sklearn Python Module. Within that function, the estimator (type of model) was the SVM classifier algorithm, where kernel = RBF, a 5-fold cross-validation strategy was used. The resulting accuracy score from the SVM model described in all the previous steps was 0.84.

3.3.4 Other forms of dimension reduction As with the other attempts at dimension reduction used with K-Nearest Neighbors, LDA and Kernel PCA dimension reduction techniques could not be used due to collinearity of data and lack of computing power, respectively.

3.4 Naïve Bayes Classification

3.4.1 Preparation for Support Vector Machine The preliminary data cleaning and preparatory steps in the SVM model were similar to the steps applied to both KNN and SVM. This included using a dataset that included PHOTO_CODE, using scikit-learn's train/test split with 30% of data reserved for testing, and normalizing data after the test/train split.

Again, Principle Component Analysis (PCA) was applied. To maintain consistency with the PCA approach from K-NN and SVM models, a 95% variance threshold was set.

3.4.2 Naïve Bayes Classification The final accuracy result was generated by completing K-Fold cross validation on the training set of data. The K-Fold validation was completed by using the cross_val_score function from the Sklearn Python Module. Within that function, the estimator (type of model) was the GaussianNB algorithm and the function used a 5-fold cross-validation strategy. As stated above, the resulting accuracy score from the Naive Bayes model was 0.66.

4. CONCLUSION

The dataset discussed in this paper contained two types of targets – both a numeric value for resistance of the thermal asperity detector and as categorical pass/fail indicator. Because there were two different datatyped target, both prediction and classification models were trained.

Multiple linear regression, decision tree regression, and random forest regression models were trained to predict the numerical target value. For each type of analysis, multiple models were trained, starting with a model that used all features. Additionally, stepwise selection and principal component analysis were used to reduce the number of features and components used to train multiple linear regression models. For the decision tree regression and random forest regression models, principal component analysis and linear discriminant analysis were used before additional model training. In addition, for random forest regression, the hyperparameter of the number of estimators was selected based on modeling results. Limitations in scikit-learn prevented use of k-fold validation, and simple test-train splits were used to validate results.

Out of all of the regression models trained, random forest regression using the first 200 principal components and 50 estimators yielded the lowest Mean Squared Error, at 10.34. Random forest regression consistently had the lowest MSE of all the regression techniques used. Even more impressively, when linear discriminant analysis was applied, the random forest regression model needed only 100 components, lower than any other model, and only 5 estimators to generate an MSE under 12.0. This score was better than the MSE generated by most of the other regression models.

The only other regression model that came close to achieving a similar level of predictive power was the multiple linear regression model trained after stepwise selection of features, with an MSE of 11.78. However, the multiple linear regression model trained on all features was, by far, the worst model trained, with a Mean Squared Error off the charts. This performance appears to be due to a single extremely poor prediction made by the model. However, multiple linear regression models trained on different randomly selected test-train splits of data yielded similar results. The conclusion must be that training multiple linear regression on all features of this dataset is unwise.

On the classification side, logistic regression, k-nearest neighbors, support vector machine, and naïve bayes classification were all used to train models that could classify data into a binary pass/fail outcome. All models were trained after using principal component analysis, and in addition logistic regression was trained using both all features and a reduced set of features after stepwise selection. Grid searches were used to select hyperparameters for both k-nearest neighbors and support vector machine models. Unlike with the prediction models, k-fold validation was successfully used on all classification models.

Of all the classification models, logistic regression after stepwise selection yielded the best mean accuracy rating of 0.85. Except for Naïve Bayes, all classifier models trained had mean accuracy ratings of 0.78 and above. This is a remarkably consistent result in comparison to the variability of the mean squared error scores generated by the prediction models, which ranged from 10.34 to $8.38*10^{25}$. It can be concluded the dataset lends itself better to classifying outcomes into a pass/fail binary than it does to predicting the exact numeric value of a target.

From a business perspective, it is quite clear that some engineering judgment is required since the number of features expanded rapidly. However, in terms of a mean accuracy rating, a 0.85 for the logistic regression is quite acceptable for the time period outlined from processing to fully functional transducer testing. The results could potentially be better with a reduced set of features as stated before. It must be stated that it was unexpected that the logistic regression would work better than the linear regression methods. Transfer functions between manufacturing processes and electrical testing are typically reported as multiple linear regression models. However, in this case, using the pass/fail criteria worked better.

One of the biggest challenges faced in this project was the large size of the dataset. Not only were there over 100,000 instances in the initial dataset (reduced to 88,00+ after data cleaning), but there were a large number of features to contend with as well. One hot encoding resulted in a total feature count of almost 400 features, and the sheer volume of data prevented the use some types of dimension reduction techniques, such as Kernel Principal Components Analysis. When computations could be completed, they sometimes took several hours to complete, even when run on a cloud-based virtual machine. With additional computing resources, it would be interesting to see the results of Kernel PCA and Linear Discriminant Analysis for classification modeling.

ACKNOWLEDGMENTS

Thanks to our instructor, Dr. Manjeet Rege, for his tireless assistance and willingness to share his wealth of knowledge