**Machine Breakdown Predictions**

**Harrison Howze, Sam Schmahl, Omar Abdelsalam, Daniel Byerly**

1. **Problem Statement and Background (15%)**

Our goal is to use a generated dataset to minimize costs and downtime of a business by predicting machine failures and finding the primary factors that contribute to those failures.

We found the initial dataset from Kaggle. The data includes rotational speed, air and process temp, torque of the machines, tool wear in minutes. There are 2 target variables, whether the machine fails and what specific failure type it is if it does fail. Our goal is to create a model of the data that can accurately predict when a machine is going to break before it does.

We believe that machine owners and operators would benefit from having a model like our intended one, as it would potentially reduce downtime and lost product due to misaligned, broken, or worn machine parts as we could better predict when the machine is to fail.

Should our work be successful, there could be practical application to the manufacturing industry as well as further incentive to collect accurate machine data to create a model like ours with real recorded data for specific machines and entire production lines.

As for related work, there have been several attempts to make predictive machine failure algorithms and of the ones we found, several were successful in predicting failures, giving us a basis to work from and potential future work in

1. **Data and Exploratory Analysis (15%)**

The dataset from Kaggle can be found at this link: [https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification/data](https://www.kaggle.com/datasets/shivamb/machine-predictive-maintenance-classification/data%20)

The dataset comes with a provided data dictionary, which can be found below:

The dataset consists of 10,000 data points stored as rows with 14 features in columns

* UID: unique identifier ranging from 1 to 10000
* productID: consisting of a letter L, M, or H for low (50% of all products), medium (30%), and high (20%) as product quality variants and a variant-specific serial number
* type: consisting of the letter prefix found in productID (L, M, H)
* air temperature [K]: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K
* process temperature [K]: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
* rotational speed [rpm]: calculated from power of 2860 W, overlaid with a normally distributed noise
* torque [Nm]: torque values are normally distributed around 40 Nm with an Ïƒ = 10 Nm and no negative values.
* tool wear [min]: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process. and a  
  'machine failure' label that indicates whether the machine has failed in this particular data point for any of the following failure modes are true.

Targets:

* Target: Failure or Not
* Failure Type: Type of Failure

We have potentially unnecessary columns, the UID and product ID which allow the algorithm to learn which items in the training set always fail, as such we removed them from the dataset for training and testing purposes. These could potentially lead to data leakage in which the model could find a pattern within the ID system to predict failures. UID is redundant for the same reason, as well as because of the *type* feature that already contains the relevant information – the product quality – that could be extracted from it. We also are not going to use failure type, as our focus is predicting whether the machine will fail or not regardless of type. Our focus at the start is binary classification, failure or not. There is no missing data, tool wear has a large amount of variance but is standardized. As for issues with the data, we examined the data thoroughly and searched for outliers, anomalies, and missing data by plotting all combinations of relations between numeric features and analyzing each one, all of which produced no issues. The visualization for that can be seen below:

A screenshot of a computer

Description automatically generated

1. **Methods (10%)**

Our initial approach to the problem statement involved building a scatterplot matrix to determine correlation between individual numerical variables, then using the *gghighlight* library to find initial trends in data when comparing 2 attributes against each other and finding patterns whether the machine breaks or not.

An issue we ran into was our data still has a categorical attribute in it alongside the rest of the numeric attributes, that being the machine type. Initially we discussed splitting the data into 3 types, one for each machine type, but we decided instead to convert from a categorical to a numeric, with L becoming 0, M becoming 1, and H becoming 2. This would eliminate the need to run multiple similar tests on three different datasets, which is more efficient for us.

We initially explored using linear or logistic regression for our model, however since we are using several attributes rather than a smaller number, we feel using a decision tree would be the superior approach to dealing with multiple attributes at once and finding the weights of each attribute on the machine failing. Decision trees are very easy to understand and interpret, and the actions of certain decision makers in the process can be considered more easily; we decided that, since we have multiple numerical variables that we are testing to see if we get a binary fail or not fail condition, we should use decision trees.

In the end we decided that our best method to approach our problem statement would be a classification-based decision tree, as it more flexible with nonlinear data than a regression model and can interpret several features at once into its decision matrix. Classification trees are used to predict a categorical value from a decision tree. We initially started using a regression-based tree to calculate our results, but we quickly realized that, even though the target value of our dataset is a numerical value (0 or 1), we are actually determining if the machine failed or not, which is a binary categorical value that is represented by a numerical 0 or 1. Classification trees also do not require us to scale our features (even though our data was already fairly standardized/normalized) and make classifications quickly. However, they do take longer to train, usually don’t work very well on small datasets, and are more difficult to extrapolate.

1. **Tools (10%)**

For our initial approach we used gghighlight and ggplot2 in order to get a cursory visualization of the data so we could see positive, neutral, or negative trends in the data while we discussed the potential uses of linear regression versus decision trees to solve our problem. We decided to go with decision trees as our approach to a solution of our problem statement for the benefits mentioned in the methods section. Initially we were going to use regressive decision trees on each of the three split segments of our data, but since we are trying to find a categorical value, we decided to use decision tree classifiers instead, as we are predicting whether or not a machine fails, which is a binary classification.

The decision tree classifier was deployed using the “rpart” library that is a part of the R package library; it is a library that is used for building regression and classification trees, and since we were using a classification tree as our model, this library was decided upon. We first split the data into a train and test set with a proportion value of 0.8 (80% train, 20% test). The line of code to set up the model is as follows:

model <- rpart(Target~., data=train\_data, method='class')

The first parameter Target ~ . is the formula that sets up the Target value as the value we want to find and all other variables in the training set as the values that are being used to determine the Target variable. It uses the training data set and is a classification tree (hence method=’class’). The minsplit parameter sets the minimum number of observations that must exist in the node in order for a split to be attempted to 2, which is also a common practice from what we’ve found. The decision tree model turned out really well, and from the picture below, we can clearly see the conditions by which the tree made its predictions. There are other parameters that can be influenced as well, including the maximum depth of the tree and the minimum number of observations that can be observed in any leaf node, but those have been left untouched for now, as they may or may not be useful to us yet.

Potential improvements to the model may lie in some of the extra function parameters that are built within the *rpart* function, as they may be able to tweak the model to a more specific and accurate prediction. We may evaluate some of these at a later date.

1. **Results (35%)**

*[Give a detailed summary of the results of your work. Here is where you specify the exact performance measures you used. Usually there will be some kind of accuracy or quality measure. There may also be a performance (runtime or throughput) measure. Please use visualizations whenever possible. Include links to interactive visualizations if you built them. You should attempt to evaluate a primary model and in addition a "baseline" model. The baseline is typically the simplest model that's applicable to that data problem, e.g. Naive Bayes for classification, or K-means on raw feature data for clustering. If there isn't a plausible automatic baseline model, you can e.g. compare with human performance by having someone hand-solve your problem on a small subset of data. You won’t expect to achieve this level of performance, but it establishes a scale by which to measure your project's performance. Compare the performance of your baseline model and primary model and explain the differences.]*

As you can see from the model below, the decision tree is an excellent measure for visualizing the results of the data and showing what the model did to determine its classification for each feature.

A diagram of a computer

Description automatically generated

(seed – 4893175)

For performance measurements we used a combination of precision, recall, and F1 score to evaluate the accuracy of the model, each of which being more accurate than expected with an F1 score around 98%. We first developed a confusion matrix based off the results of the predicted values of the test set, which was then used to help calculate our precision metrics. The confusion matrix helps us visualize the results of the performance of our model. Each row of the matrix represents the instances of the actual class, while the columns represent the predicted class. The confusion matrix that was generated with the above decision tree was the following:

> conf\_mat

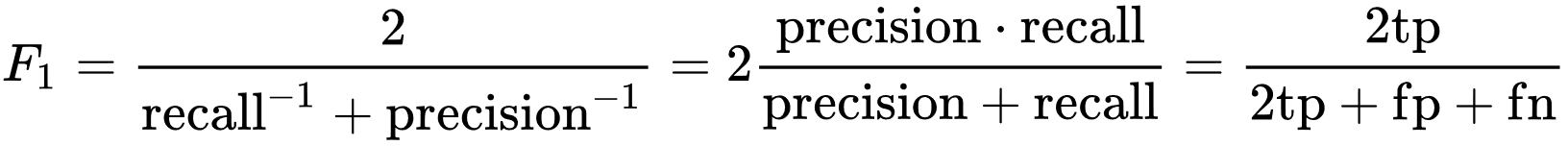
predict\_test

0 1

0 1928 6

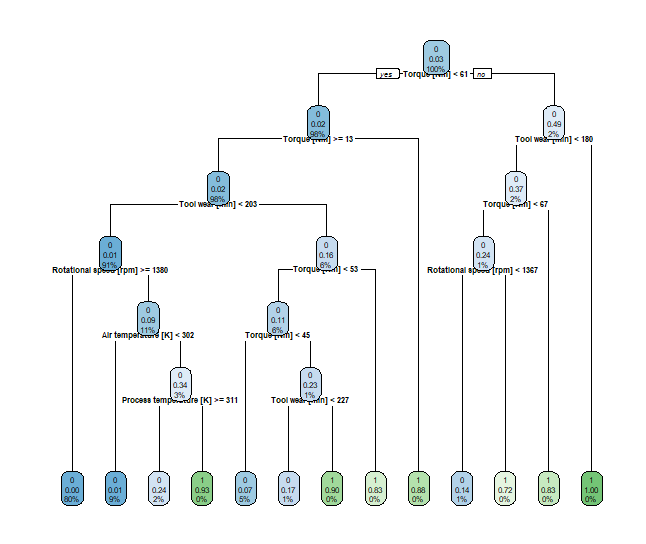
1 24 42

From this matrix we can see that the total results add up to 2000 (the total number of instances in our test set). 1,928 of the instances were predicted correctly to be a 0 (the machine did not fail), whereas 42 were instances that were predicted correctly to be a 1 (machine fail). We can already see that a vast portion of the training data is mostly made up of instances of machines of machine that didn’t fail; this will be important later for analyzing potential issues with our dataset. We decided upon using these measures as they are good performance metrics for analyzing classification systems. The precision metric finds what proportion of the identifications that were “positive” were actually correct, or in other words how many “true positives” were correctly identified out of all positive identifications. Precision is calculated by dividing the number of true positive identifications by the sum of all true positive and false positive identifications. The recall metric, also sometimes called the “sensitivity” metric, find the proportion of actual positives that were identified correctly; in other words, it calculates how many “correct” guesses were made out of all instances that were identified as “correct”. This metric is calculated by dividing the number of true positive identifications by the sum of all true positive and false negative identifications. We then used these two metrics to calculate the F1 score of our graph, which is a measure of the accuracy of both of those tests. The formula for calculating the F1 score is the harmonic mean of the precision and recall factors, and the formula can be seen here:



We also ran a general accuracy measure, which was calculated by the sum of all correctly identified instances (that is, all instances that were correctly predicted both true and false) divided by the sum of all values in the confusion matrix. With the baseline model, the precision, recall, f1-score, and accuracy are all around 98% to 99%, meaning that the accuracy of the model and its ability to predict correct measures is incredibly high, almost perfect in fact.

For the final primary model, we decided to dive into the extra parameters that are built into the *rpart*  function. One of those parameters is *maxdepth*, which is used to control the height of the tree. One thing of note from the baseline model is the back and forth between the “Process temperature [K]” and “Air temperature [K]” that occurs after a depth of 6. In order to trim the tree slightly, we set the *maxdepth* tree to 6 to see what happens if this sort of ping-pong effect between those two temperature variables is pruned from the tree. We thought about potentially decreasing the *minsplit* variable, which would limit the smallest number of observations in the parent node that could be split further, but since this could potentially lead to overfitting and our baseline model already seems fairly “accurate”, we decided not to change that. Here’s a visualization of this new decision tree with a maxdepth of 6:



As we can see from this decision tree visualization, the depth has been capped at a max of sixth; the only major change that was that the back-and-forth between the two temperature values has been cut from the tree. In terms of performance measures there was very little change between the primary and the base model: the percentage range never changed by a significant amount (the highest difference was between the recall metrics, which varied by approximately 0.006 between the two models, ). This shows that there is no significant difference between our baseline and primary model and that the default parameters of the *rpart* function fit the dataset fairly well as it currently stands. The confusion matrix for the primary model is as seen below:

> conf\_mat\_b

predict\_test\_b

0 1

0 1929 5

1 36 30

Interestingly enough, the false negative rate appears to have risen a much more significant amount: this may be due to the maximum depth of the tree being capped at 6, so the ability of the tree to correctly identify true positives and true negatives may have been limited. Its identification of 0 actually improved by correctly predicting one extra positive instance.

With such a high F1 score, there is the possibility of some error within the model. Our first though was the model was seeing the classification or some form of identifier whether it would fail or not that was outside of its parameters, however simple analysis of each layer of the decision tree ruled that out quickly. Our more likely culprits are a lack of negative results within the data and the model being overfit. Since out of the 10000 data points only around 300 came back with a 1 indicating a failure of the machine, our model could predict that every machine would work and still be right 96.6% of the time. With the fact that such a high percentage of this dataset is categorized as machines not failing, it is difficult to assess whether or not the machine is accurately predicting machine failure due to the small number of the data that categorizes a machine as failing; taking into account that the data is split into training and testing, with around 20% of the data being used as a test, that limits the number of machine failures even further when the model is making its predictions based on the new data. Since our examples of failures are limited, this leads to a bias towards declaring the machine functional and not failing, and the easiest solution to this issue for any future research would be to expand the dataset to add more failure cases for testing.

Performance Measures

* Precision
* Recall
* F1 score
* All above to help evaluate accuracy
* Accuracy score (Baseline: 98%)

Baseline measures

* No cap to depth
* We keep the industry standard 80-20 ratio
* Accuracy score: 98% (Potentially overfit?)

Primary

* Tinker with the tree, find best results
* Best result changes:
* Explanations:
  + 96.6% of the dataset involves non-failing machines, therefore our examples of machine failures are limited and not representative.

1. **Summary and Conclusions (10%)**

*[In this section give a high-level summary of your results. If the reader only reads one section of the report, this one should be it, and it should be self-contained. You can refer back to the "Results" section for elaborations. This section should be less than a page. Emphasize any results that were surprising.]*

* Show baseline tree with uncapped depth
  + Include its accuracy, prec/recall, f1 score
* Compare baseline with best fit that we found
  + Depth, accuracy, etc.
* What factors in the tree had the most influence? What had the least?
* Was the dataset entirely fit for this type of research (only ~3% failures)
* How will this help some company that this dataset could be based on?

1. **Appendix (5%)**

A link to the GitHub repository with our work can be found here:

<https://github.com/Hmhowze/Machine_Breakdown_Prediction>