# Defect Learning in Manufacturing

[redacted] Raven Pillmann [redacted] [redacted]

Coauthor Name Redacted for Privacy

Coauthor Name Redacted for Privacy

Coauthor Name Redacted for Privacy

## Abstract

Defect detection in manufacturing is an important task to achieve efficiently and accurately. Leveraging machine learning models to detect defects has been done in a variety of ways previously, achieving acceptably high levels of accuracy. This project aims to apply machine learning to the task of defect detection and evaluate the results of the implemented models. We use Decision trees, naive Bayes, Word2Vec, and LSTM techniques on real-world data to explore and determine the effectiveness of each approach with the F1 score metric. Processing the data into useful feature sets for input into various models turned into one of the primary focal points of the project. It is shown that with specific models we can attain high F1 scores, such as 0.970 for Decision Trees, which suggests high feasibility for potential implementation in real-world applications.

## Introduction

One of the biggest goals of manufacturing is to have zero defects. In competitive environments, having zero defects is the primary metric because it has a direct impact on time to market and profit margins. In complex manufacturing settings, defect detection is just as critical as defect correction.

There are multiple safety nets and quality control steps that are established to ensure that products go through the line robust and defect free. One of the control steps is the standard statistical process charts that force the lots to stop processing if there is a deviation (defect) detected. Once a lot deviates from the main path, appropriate steps are taken to address the issues causing the deviation, and then it goes back to a particular station, so it can resume its main path. The decision as to where to position the lot in the line is up to the human operators. This decision is prone to errors and causes unnecessary detours and sometimes rework or even scrap.

We approached this problem as a fault diagnosis problem. Fault diagnosis can be viewed as a supervised learning task whose target is to classify a new testing observations to one of the existing classes (Zhao et al., 2018). In this paper, we attempt to learn the defect decisions that are effective by understanding and optimizing the key variables that cause bad decisions. Four well-known approaches were studied: Decision Trees, Naive Bayes, Word2Vec, and LSTM.

We wanted to start with **Decision Trees** because it has been known to be a powerful technique to discover a tree model for future event prediction. Decision trees are among the simplest machine learning algorithms, and are commonly used as base classifiers in ensemble methods. Furthermore, decision trees are robust to noise, can easily handle redundant features, and are easily interpretable, which is useful for real-world diagnostics.

We chose **Naive Bayes** because is a generative model that tends to have good results as a classifier. Additionally it converges quickly, as a simple model it could be trained and run quickly compared to other methods. Approaching naive Bayes as a candidate is an evaluation of another simple model.

Because we had a large volume of human comments, we also chose to study the corpus-based semantic representations of those dialogues. These semantic representations are called embeddings, and they rely on the idea that words with similar meanings tend to occur in similar contexts. We chose a skip-gram neural network language model in **Word2Vec** to learn the word embeddings of the comments that were made within the context of defects detection and resolution. This was to analyze and understand if there were semantic differences between defects comments where people ended up making right decision versus incorrect ones. The application of this would be to predict if a particular decision would be good or bad based on the comments that were being made on the deviation.

Each row of data is information on a sequence of events in the process that led to the decision as good or bad. In looking to explore a deep learning model that would handle sequential data one sample at a time, we decided to look at recurrent neural networks and its variants. RNN architectures have shown state-of-the-art performance on a wide range of complicated sequential problems including signal processing, speech classification and video captioning. Long short-term Memory **(LSTM)** is a variant of RNNs which was designed to better handle cases where the output at time t depends on much earlier inputs (Hochreiter and Schmidhuber, 1997).

In the work, we are essentially finding items that do not belong, often in the context of consistency or quality. This can be accomplished in a variety of ways, but when the examples of defective parts are numerous enough to train a classifier in a traditional supervised way, it is not really necessary to think of this problem as a specialty case. In fact, the currently curated dataset that is used for the experiments consists of 16,865 values, with 5,551 being defects with bad decisions. This produces a good to bad decision ratio of approximately 2:1 (2.038:1). Though Chandola et al. (2009) specify that there are instances where this holds true in anomaly detection, generally the case is one of limited outlier data. Accurately determining anomalous behavior is desired, but other metrics to consider are false alarms and missed detection. Ultimately, the goal is to have high accuracy in detection and low false alarms.

## Problem Statement

Yield is a standard metric that is used to quantify defects and could be affected by variations in the product components, processes, and tools. Production lines are rigorously monitored with statistical process controls throughout the production line. These are human-intensive, expensive, mostly univariate and not practical if you want full coverage of the entire process line. We seek to explore an architecture that enables a more economical and robust way of defect detection by applying advanced techniques for pattern recognition.

One of the challenges with fault prediction in an industrial process is the nature of data. The data is normally captured at different steps in the process and can contain extremely large number of data points. Pre-processing will require curating this data into a set of engineered features required to achieve acceptable performance.

The data contains deviant operations in a manufacturing assembly line. Each observation in the data denotes a defect in the manufacturing process which causes the widget to be removed from the assembly line so it can be investigated by the domain experts. There are 3 possible paths that the widget can take when it is taken off its normal process: 1) the experts can say the problem is not big enough and put it back in the assembly line, 2) they can say that particular operation needs to be redone, or 3) it can be determined that the widget is no longer fit for use. Once the decision to proceed is made, operators physically have to move the widget to the location so the widget can continue its life in the assembly line. It has been reported that there are a lot of mistakes made during this process, that is, operators are putting widgets back in the wrong part of the assembly. This mistake takes a trained eye to identify, and this could mean that the widget is manufactured inaccurately. This delays shipments of widgets, causes wasteful runs in the tools which could otherwise have been processing the correct widgets, and also results in scrapping widgets down the line because they are not built to design.

## Significance and Impact

The onus of keeping the line defect free requires a huge amount of effort from engineers. There are engineers specifically hired to analyze the line yield and establish controls to prevent deviations that could cause unnecessary rework, or worse, scraps. In a sequential assembly line with over a thousand steps, the later defects that are not recoverable are found, the more efficiency and time to market loss may be incurred. There is a high need to find automated, cost-effective and timely ways of monitoring the defects because of the high urgency for time-to-market. A robust system that continuously monitors the processes and alerts or sends notifications and makes intervention in the factory floor such as shutting a tool down or putting suspicious process on hold could not only help streamline and make our processes more effective but also relieve the engineers from having to dedicate so much of their time in just identifying patterns in deviations. This could result in significant cost savings in terms of rework, human resources and scraps.

The current standard for alerting on anomalies in the factory is LOESS (Locally estimated scatterplot smoothing). LOESS, originally proposed by Cleveland (1979) and further developed by Cleveland and Devlin (1988), specifically denotes a method that is also known as locally weighted polynomial regression. At each point in the range of the dataset a low-degree polynomial is fitted to a subset of the data, with explanatory variable values near the point whose response is being estimated. However, we get a lot of false positives from it, which has led to the undesirable effect of the users being desensitized to the alarms. There are traditional statistical process charts to control for processes that go out of specification. However, due to the overhead required with it, these controls are set only for critical processes.

We hope that our contribution will be to explore and deploy an ensemble of models to learn and classify defects. In the real world, we would have a scoring mechanism that would give more weight to an alert if multiple models are predicting the same thing.

## Related Work

In high technology industries where the processes are highly complex and intellectually protected, it can be hard to benchmark on what kind of automatic defect detection systems have had successes in the industry at large. We seek to establish a repeatable process of defect detection by designing data structures most suitable for feature engineering and exploring various machine learning techniques to identify and understand which techniques work best for which kind of data and/or problem (Fox and Weisberg, 2018).

Detecting defects in semiconductor manufacturing was investigated in Huang et al. (2018) where the implementation of machine learning algorithms were shown to provide high levels of accuracy in predicting defects. The two implemented algorithms were k-nearest neighbors and random forests, with the random forest approach providing additional predictive capabilities for them as well. Random forests also outperformed k-nearest neighbors in this instance. Ye et al. (2018) implemented a neural net with five convolutional layers and two fully connected layers to classify defects based on captured images. The training was done on touch panel glass and back-coated mirrors and achieved accuracy above 96%.

Software is also prone to defects, and similar to our approach. Perreault et al. (2017) implemented multiple classifiers and evaluated the results based on F1 score as well as accuracy. The classifiers that were investigated were naive Bayes, neural networks, support vector machines, logistic regression, and k-nearest neighbor. The results indicated that most of the approaches produced good and comparable results, greater than 90% accuracy and greater than .90 F1 scores. Naive Bayes was, however, shown to produce slightly lower accuracy results.

Decision trees have been used in various capacities in regards to anomaly and defect detection. Usually, decision trees are used in hybrid systems. Jian et al. (2010) uses a hybrid model to detect defects in leather, where a neural network is first used to determine important features, which are then used to induce a decision tree. Aghdam et al. (2012) similarly used decision trees for detecting surface defects of steel. In particular, they first use a decision tree to classify whether an image contains a defect or not, and if so, a support vector machine is used to gain more insight into the type of defect. Both of these works demonstrated that stand-alone decision trees are sufficient classifiers to identify defects, and used additional models to enhance their predictions (for example, to specify the type of defect).

Naive Bayes classifiers have been seen a lot in the area of anomaly detection, particularly in Intrusion Detection Systems (IDS), either specifically as IDS or other security-based implementations. Muda et al. (2011) set out to use a hybrid approach of k-means clustering to provide the basis for classification with naive Bayes for determining anomalies. Another example is provided by Li and Li (2010) who used naive Bayes as the weak learner in an AdaBoost implementation for IDS with promising results.

Ahirwar et al. (2012) also implemented an IDS, relying on naive Bayes in addition to a radial basis function network that used evolutionary algorithms to optimize the network. This would be similar to our work, comparing established models for the task of defect detection in real-world data. The results of the comparison varied across several attack vectors, with naive Bayes yielding better results in two cases: User-to-Root attacks and normal behavior. In the experimental results naive Bayes had detection rates of 88.6% and 91.9% respectively.

On the industry front, EU-funded researchers are also developing advanced technology for the manufacturing industry, aiming for zero-defect production of everything from aircraft to machine tools (“Intelligent Approach…”, 2015).

There are many studies on application of LSTMs for fault detection and predictive maintenance for industrial and manufacturing processes. Extensive experimental results from work by Zho et al. (2018) for fault identification and analysis on a chemical process indicates that LSTMs can better separate different faults and provide more promising fault diagnosis performance. In the area of predictive maintenance, a study on developing an application to send an alert before an engine breaks down, has used LSTMs to predict current engine condition (Aydin and Guldamlasioglu, 2017). LSTMs were proposed for fault diagnosis in railway track circuits because of its ability to model long-term temporal dependencies that are characteristic of the faults in the track circuit (Bruin et al., 2017). There are also studies showing use of LSTMs as a way to predict failure based on log analysis, taking advantage of the models ability to work well with sequential data (Zhang et al., 2016).

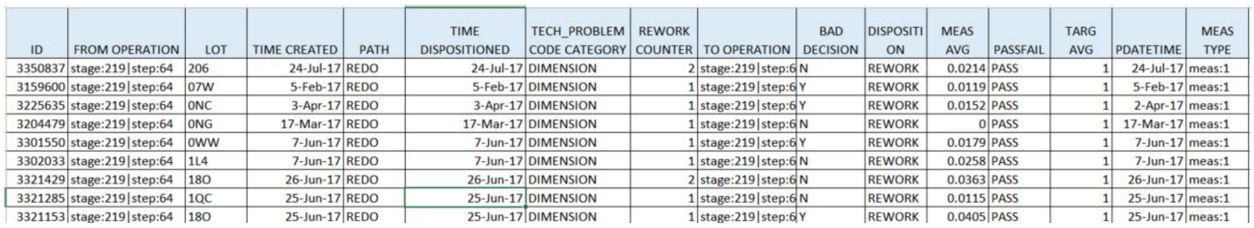
## Feature Extraction & Preprocessing

We used real data from an industrial process for our analysis. This presented us with two major problems that turned out to be more challenging than the team expected. First, the data had to be anonymized so it make it shareable, and second, it had to be preprocessed it in a way that it could be fed into the algorithms we chose.

A number of steps were taken to anonymize the data and mask the real sequence of events. Not all fields in the database were shared with the team. Instead, a carefully picked set was picked based on the experience of the researcher sharing the data.

The products, operations, and measurement categorical types in the data were anonymized. We left the actual measurement values as they were because they would be used for our analysis. Separate mapping tables were created to map the real value with an encrypted value. The data used was sourced from multiple tables, and was combined using standard SQL queries.

Sample Data shown below. Data explanation Appendix 1a



| ID | FROM OPERATION | LOT | TIME CREATED | PATH | TIME DISPOSITIONED | TECH_PROBLEM CODE CATEGORY | REWORK COUNTER | TO OPERATION | BAD DECISION | DISPOSITI ON | MEAS AVG | PASSFAIL | TARG AVG | PDATETIME | MEAS TYPE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 3350837 | stage:219\|step:64 | 206 | 24-Jul-17 | REDO | 24-Jul-17 | DIMENSION | 2 | stage:219\|step:6 | N | REWORK | 0.0214 | PASS | 1 | 24-Jul-17 | meas:1 |
| 3159600 | stage:219\|step:64 | 07W | 5-Feb-17 | REDO | 5-Feb-17 | DIMENSION | 1 | stage:219\|step:6 | Y | REWORK | 0.0119 | PASS | 1 | 5-Feb-17 | meas:1 |
| 3225635 | stage:219\|step:64 | 0NC | 3-Apr-17 | REDO | 3-Apr-17 | DIMENSION | 1 | stage:219\|step:6 | Y | REWORK | 0.0152 | PASS | 1 | 2-Apr-17 | meas:1 |
| 3204479 | stage:219\|step:64 | 0NG | 17-Mar-17 | REDO | 17-Mar-17 | DIMENSION | 1 | stage:219\|step:6 | N | REWORK | 0 | PASS | 1 | 17-Mar-17 | meas:1 |
| 3301550 | stage:219\|step:64 | 0WW | 7-Jun-17 | REDO | 7-Jun-17 | DIMENSION | 1 | stage:219\|step:6 | Y | REWORK | 0.0179 | PASS | 1 | 7-Jun-17 | meas:1 |
| 3302033 | stage:219\|step:64 | 1L4 | 7-Jun-17 | REDO | 7-Jun-17 | DIMENSION | 1 | stage:219\|step:6 | N | REWORK | 0.0258 | PASS | 1 | 7-Jun-17 | meas:1 |
| 3321429 | stage:219\|step:64 | 18O | 26-Jun-17 | REDO | 26-Jun-17 | DIMENSION | 2 | stage:219\|step:6 | N | REWORK | 0.0363 | PASS | 1 | 26-Jun-17 | meas:1 |
| 3321285 | stage:219\|step:64 | 1QC | 25-Jun-17 | REDO | 25-Jun-17 | DIMENSION | 1 | stage:219\|step:6 | N | REWORK | 0.0115 | PASS | 1 | 25-Jun-17 | meas:1 |
| 3321153 | stage:219\|step:64 | 18O | 25-Jun-17 | REDO | 25-Jun-17 | DIMENSION | 1 | stage:219\|step:6 | Y | REWORK | 0.0405 | PASS | 1 | 25-Jun-17 | meas:1 |

We used a simple anonymization technique where the operations were shuffled and then a random number was assigned to each operation with the corresponding original value of the operation. Similar techniques were applied to anonymize the measurement types.

Most fields were categorical, but there were measurement values and their targets which were continuous in nature. These values were dependent on the lot being processed. Some lots had high numbers for their average measurements and target values and others were much lower. It was decided to compute the difference between the target value and the average measurement and normalize the variation to a scale of 0 to 1. There were a number of missing values for this feature, which were populated with a simple average of the feature value after normalization.

The data spanned across multiple days, and the time between processes were recorded in the date format. It was decided to drop time stamps at different steps and create new categorical features that showed the time taken between processes.

## Experiments

Because we used real world data, it was messy and due to the confidential nature of the data that is used, there was considerable amount of work done to anonymize the data. Further work was done to refine the data for processing within the various models.

Decision trees are supervised learning models that classify records by separating them via various feature thresholds. They are among the easiest machine learning models to implement, and can be found as the basis for a number of ensemble methods. Decision trees are sensitive to unbalanced data, favoring the majority class over the minority class (Almas et al., 2012).

Ensemble methods are introduced as a way of building more powerful classifiers from simpler models. Several ensemble methods exist, and in this project we explored Random Forests, AdaBoost, and Bagging, with decision trees as base classifiers.

We used scikit learn to implement these models, and hyperparameters were chosen with cross-validation. For Decision trees, entropy was chosen as the purity measure, and training continues until leaves are reached either by having only data points of the same class or there are less than 2 data points in a node. For the Random Forest, 200 decision trees were created with entropy as the impurity measure and a maximum of 10 features per tree. For Adaboost, 500 decision trees were trained with a maximum depth of 3 nodes, and entropy as the impurity measure. Finally, for the bagging scheme, 40 decision trees were trained, with entropy as the impurity measure.

Naive Bayes classifiers provide an approach with simple calculations based on assuming feature independence and applying Bayes' rule: $P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$. The assumption that the features are independent allows simple calculations based on Gaussian distributions within each feature. We can classify based on the equation: $y = arg\,max_y\, P(y) \prod_{i=1}^{n} P(x_i|\,y)$, for some class $y$. If dependencies among features do exist or if the distributions are not Gaussian, this could cause worse results. The possible point of concern in the data is the abundance of labeled data, even with conversion to numerical data, an analysis of the distribution was not done prior.

Implementing a naive Bayes classifier on the dataset seemed possible after preprocessing the data. Upon receiving the preprocessed data, and implementing the same data read process as the Decision Trees approach, cross-validation was implemented via scikit-learn's built in tools. The number of folds chosen was 10 to provide a larger number of values to analyze. With the cross-validation it was possible to look at accuracy and F1 scores per train-test split as well as see averages and standard deviations. This simple implementation should provide insight into whether naive Bayes is appropriate for the processed data or not. Utilizing existing tools to calculate F1 scores and accuracy are deemed appropriate in doing so.

The raw dataset came with large volume of comments[1] from domain experts working on defects. We felt that it was important to process and analyze them to understand if we could get additional insights. We experimented on word2vec and latent semantic analysis. Given that

[1] Analysed but not anonymized and shared due to information gain and confidentiality reasons respectively

both are unsupervised, the only way we could check the accuracy of these models is by a domain expert looking at the results and determining if the results made contextual sense. This complicated the project because all supervised learning was done with anonymized data while the topic modeling had to be done with raw data, so that a domain expert could validate it.
Long short-term memory (LSTM) was explored as a deep learning model to classify the decision. The hypothesis was that the information in the sequence of information on the process flow represented by the features had a direct impact on the outcome – the decision.

An LSTM network is  a variant of RNN, whose architecture is designed to use predicted values at time t in order to predict values at time t+1. The implication here is that LSTM has to remember values over time. The ability to have a memory of previous predictions makes it possible to capture repeating structure over time or in a sequence of events.

LSTMs are trained using back propagation through time. LSTM networks have memory units which are connected through its different layers. There are different components in the memory unit that control how values flow through the network.

The input sequence is put through a memory unit and gates inside it are triggered conditionally by the activation functions in order to regulate how the values of the sequence are used. A cell component store values over time.

In LSTMs gates are a way to optionally let information through. They are composed of a sigmoid neural net layer and a pointwise multiplication operation.

There are three types of gates.

- The forget gate controls which information to keep and which to throw away from the cell state.
- The input gate has two layers. The input layer decides which values will be updated and the tanh layer creates a vector of new candidate values that could be added to the state.
- The output gate regulates the extent to which the input and memory is used to compute the output based on the cell state.

There are many variants to LSTMs like adding ‘peephole’ connections (Gers and Schmidhuber, 2000), or the Gated Recurrent Unit (GRU) that combines the forget and input gate into a single update gate (Cho. Et al, 2014).

A vanilla LSTM was implemented using Tensorflow api. The data was normalized to the input format that was required for ingest to the Tensorflow. The LSTM needs data with the format of [samples, time steps and features]. In this implementation, LSTM was unrolled through 6 steps and had 100 hidden units. It has a learning rate of 0.001. Tensorflow’s native implementations of the Adam optimizer is used with softmax cross entropy loss to define the network’s optimization and loss (graph in Appendix).

## Results

Accuracy, when used with unbalanced data sets, is often misleading. In our manufacturing data set, it would be possible to build a classifier that always predicts non-defects and still obtain an accuracy of 67%. In more extreme cases of unbalanced data, this accuracy could become much higher, without the classifier being informative at all.

Precision and recall as common evaluation metrics. Precision measures the percentage of classified objects that are correctly classified. Recall measures the percentage of objects of a certain class that are categorized as that class. F1 score is the harmonic mean of precision and recall: in order to achieve a high F1 score, both precision and recall need to be high. Therefore, F1 score is a suitable metric for evaluating classification with unbalanced data sets.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F1\ Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

We averaged F1 scores and accuracy across 10 training runs.

| **Classifier** | **F1 Score** | **Accuracy** |
|---|---|---|
| Decision Tree | 0.970 | 0.981 |
| Random Forest | 0.978 | 0.986 |
| AdaBoost | 0.984 | 0.989 |
| Bagging | 0.972 | 0.982 |
| Naive Bayes | 0.225 | 0.660 |
| Word2Vec | Contextual Interpretation | Contextual Interpretation |
| LSTM | 0.720 | 0.830 |

Decision Trees and ensemble methods achieved top F1 scores, indicating their good fit for defect detection in manufacturing. This is aligned with results achieved in other studies. In general, decision trees are good at ignoring irrelevant and redundant features: with a raw data set, where we may not know exactly what makes a decision bad or good, decision trees and decision tree based ensemble methods are simple, effective models to use. Additionally, they are exceptionally useful for this particular use case because of their expressiveness and interpretability, making them understandable for those supervising the manufacturing plant.

Ensemble methods also performed relatively well, with F1 scores at or above the decision tree. Out of all of the ensemble methods, AdaBoost performed the best. Originally, a simple decision stump was used as the base classifier, which achieved a low F1 score of 0.627. However, with max depth 3 decision trees as base classifiers, we achieved F1 scores of 0.984. Additionally, random forests and bagging performed on par or better when compared to single decision trees.

For naive Bayes, scikit's cross-validation tools were utilized to run 10-fold cross-validation on the data and gather the accuracies and F1 Scores. The cross-validation splits the data into 10 folds and iteratively fits the classifier to each subset in a 9:1 train-test ratio. This should allow full utilization of the data to gain insight into the classifier's ability to handle the data. The average F1 score for the Gaussian naive Bayes classifier implemented is 0.225 with average accuracy of 66%. The prior work suggested that naive Bayes would not provide the accuracy that the other classifiers could, but this is significantly lower than (Perreault et al., 2017) where reaching over 80% accuracy and over 0.90 in F1 scores were found in all the studied cases for software defect detection. It seems unlikely that naive Bayes would be recommended for this application, though perhaps different preprocessing approaches might yield better results.

We used word embeddings to understand and analyse the free form user comments in the dataset. The 2 techniques used were Word2Vec and Latent Semantic Analysis. The quality of the results was verified by the domain expert. The results were promising in that the Word2Vec model predicted the similarity of the operations and defect types well. Latent Semantic Analysis also did a good job of differentiating the various concepts(topics), although we feel there is more future work and refinement required to make it applicable for our problem space.

The model results were compressed into 2D space and TSNE was used to generate a manifold. Zooming into a subsection of the manifold, we can see the words being drawn with your closest neighbors.

To test the validity of the results, we picked a particular point STAGE220STEP1. Running similarity generated the following list.

This result was decrypted to understand if the similarities made sense. We found out that the operations that the word2Vec determined were the closest, were actually the operations that happened closest to STAGE220STEP1.

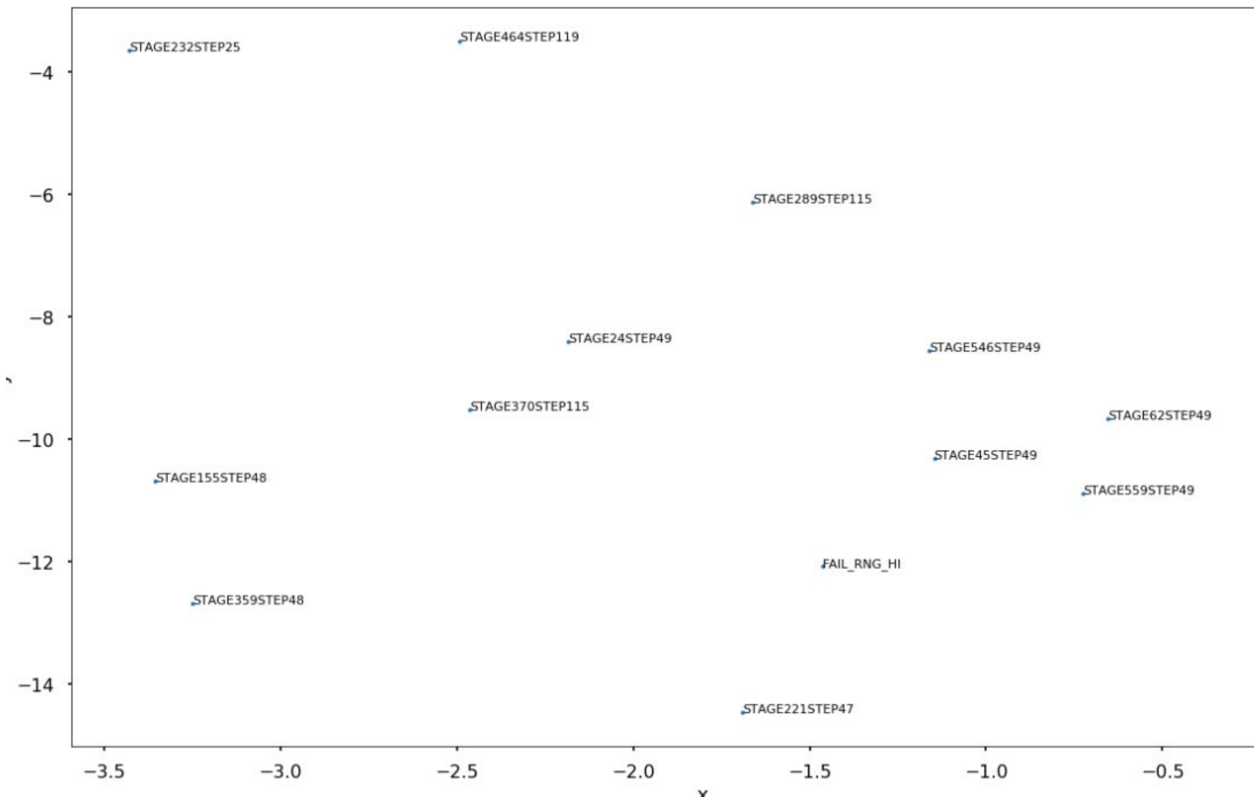


**Fig. 1. Zoomed in 2D projection of the similarities for a section of data**

| OPERATION | RANKING | ACTUAL STATION SEQUENCE |
|---|---|---|
| STAGE220STEP1 | -- | 1160 |
| STAGE220STEP3 | 0.9224976897239685 | 1158 |
| STAGE66STEP40 | 0.6536527872085571 | 1491 |
| STAGE66STEP3 | 0.6399881839752197 | 1492 |
| STAGE66STEP115 | 0.6377933025360107 | 1494 |
| STAGE437STEP25 | 0.6027792692184448 | 807 |
| STAGE300STEP48 | 0.5905027389526367 | 1083 |
| STAGE84STEP3 | 0.5790196657180786 | 985 |
| STAGE135STEP14 | 0.5726902484893799 | 203 |
| STAGE65STEP1 | 0.5703320503234863 | 1079 |

Intrepretation:

Looking more into the most similar operation, STAGE220STEP3, we can see that STAGE220STEP3 is the to-operation for the operation STAGE220STEP1. It makes sense that it ranked at .92 because it happens to be a different step of the same stage.

There were 21 instances in the entire dataset, where the from_operation was STAGE220STEP1, 11 of them went to STAGE220STEP1. The rest went to the same location, therefore would not show up in the model. It thus makes sense that the model caught it as the most similar.

Samples of sentences where STAGE220STEP1 occurs is shown below:

| |
|---|
| STAGE220STEP1 LAPBACK DIMENSION STAGE220STEP3 REWORK PASS |
| STAGE220STEP1 LAPBACK DIMENSION STAGE220STEP3 REWORK FAIL_RNG_HI |
| STAGE220STEP1 LAPBACK DIMENSION STAGE220STEP3 REWORK FAIL_AVG_LOW |
| STAGE220STEP1 LAPBACK DIMENSION STAGE220STEP3 REWORK FAIL_NUM_PTS |
| STAGE220STEP1 LAPBACK DIMENSION STAGE220STEP3 REWORK FAIL_AVG_HI |
| STAGE220STEP1 REDO DIMENSION STAGE220STEP3 REWORK PASS |
| STAGE220STEP1 REDO DIMENSION STAGE220STEP3 REWORK FAIL_AVG_HI |
| STAGE220STEP1 REDO PROBLEM_RUN STAGE220STEP3 REWORK FAIL_AVG_HI |
| STAGE220STEP1 REDO PROBLEM_RUN STAGE220STEP3 REWORK PASS |
| STAGE220STEP1 LAPBACK MISENG STAGE220STEP3 REWORK PASS |
| STAGE220STEP1 LAPBACK MISENG STAGE220STEP3 REWORK FAIL_AVG_HI |

The model estimated that 21st most similar was word was operation:
STAGE229STEP48

Querying that dataset we can see the following  sentences:

| |
|---|
| STAGE229STEP48 REDO DIMENSION STAGE229STEP48 REWORK FAIL_NUM_PTS |
| STAGE229STEP48 DRYETCH_REWORK PROBLEM_RUN STAGE229STEP48 REWORK FAIL_NUM_PTS |
| STAGE229STEP48 REDO DIMENSION STAGE229STEP48 REWORK FAIL_AVG_LOW |
| STAGE229STEP48 REDO DIMENSION STAGE229STEP66 REWORK FAIL_NUM_PTS |
| STAGE229STEP48 REDO DIMENSION STAGE229STEP48 REWORK FAIL_AVG_HI |

| |
|---|
| STAGE229STEP48 REDO PROBLEM_RUN STAGE229STEP48 REWORK FAIL#ABORT |
| STAGE229STEP48 REDO DIMENSION STAGE229STEP48 REWORK PASS |
| STAGE229STEP48 REDO MISENG STAGE229STEP48 REWORK FAIL_NUM_PTS |

We can interpret that the 2 operations ‘STAGE220STEP1 ‘ and ‘STAGE229STEP48 ‘ tend to fail for the same reason and same measurement type.  The model estimated that 1601 most similar word to STAGE220STEP1  was DIMENSION.

What is interesting about this finding is that it demonstrates that the model is doing more than frequencies. For example, when we give it a word that means an operation, it is assigning higher similarities to operations than failure types or measurement types. If we were to ask an operator what an operation is most similar to, he is mostly likely going to decide another operation, and not a measurement type or a failure mode.

This was used using gensim’s top-N similar words, which uses the multiplicative combination objective proposed by Levy et al. (2015). Positive words still contribute positively towards the similarity, negative words negatively, but with less susceptibility to one large distance dominating the calculation.

In the common analogy-solving case, of two positive and one negative examples, this method is equivalent to the “3CosMul” objective (equation (4)) of Levy et al (2015).

$$cos(x,y) = \frac{\vec{v}_x \cdot \vec{v}_y}{\sqrt{\vec{v}_x \cdot \vec{v}_x}\sqrt{\vec{v}_y \cdot \vec{v}_y}} =$$

$$\frac{(\vec{w}_x + \vec{c}_x) \cdot (\vec{w}_y + \vec{c}_y)}{\sqrt{(\vec{w}_x + \vec{c}_x) \cdot (\vec{w}_x + \vec{c}_x)}\sqrt{(\vec{w}_y + \vec{c}_y) \cdot (\vec{w}_y + \vec{c}_y)}}$$

$$= \frac{\vec{w}_x \cdot \vec{w}_y + \vec{c}_x \cdot \vec{c}_y + \vec{w}_x \cdot \vec{c}_y + \vec{c}_x \cdot \vec{w}_y}{\sqrt{\vec{w}_x^2 + 2\vec{w}_x \cdot \vec{c}_x + \vec{c}_x^2}\sqrt{\vec{w}_y^2 + 2\vec{w}_y \cdot \vec{c}_y + \vec{c}_y^2}}$$

$$= \frac{\vec{w}_x \cdot \vec{w}_y + \vec{c}_x \cdot \vec{c}_y + \vec{w}_x \cdot \vec{c}_y + \vec{c}_x \cdot \vec{w}_y}{2\sqrt{\vec{w}_x \cdot \vec{c}_x + 1}\sqrt{\vec{w}_y \cdot \vec{c}_y + 1}} \qquad (4)$$

Fault detection with LSTM based models have shown promise in research as we saw in earlier discussion of related work. For this project, the LSTM was trained on 80% of the data and tested on 20%. After 800 iterations the model had an accuracy of 83%. Given the constraints of the data, the accuracy can be seen as an indication that tweeks to the model and training the model with a broader range of feature points on the process while maintaining the actual sequence of process steps is likely to increase accuracy.

Another possible enhancement is to explore each lot and it's associated label. This was not possible for the given dataset as there were 820 lots. Instead a richer dataset of fewer lots could help create a model that learns better and predicts with accuracy.

One of the challenges working with real world data was the time it took to pre-process the data and transform it to suit the ingest requirements for LSTMs. It should also be recalled that all the data points were not made available and the data did not follow the real process sequence in order to maintain the anonymity of the data. At under 13,492 rows the training dataset (80% of the data) was also relatively small.

One path to ensure that the data keeps it's sequence information is for the original owners and can still have value for researchers is to create synthetic data using a schema driven data generation process which has shown good results in sharing real world big data in practice (Friedman and Dunning, 2015).

Another technique could be use privacy preserving techniques like differential privacy to reveal sequential information about the process without compromising the privacy of the data.

## Discussion

The topic of defect detection can fall under various umbrellas of active machine learning research. We could think of this problem as an anomaly detection problem if we were to adhere to the broader definition of anomaly detection as the problem of finding patterns in data that do not conform to expected behavior (Chandola et al., 2009). However, we are hesitant to call it that because irregularities to us seem to be not as rare as we would typically see in the anomaly detection studies. Therefore, we settled on the title Defect Learning because in essence we are trying to understand what happens when the lot deviates and based on that predict where it should go in the line after the defect fixes are completed.

The dataset used for the project was a selective collection of data points to preserve the privacy of the data. In light of the experience we had working on this dataset, we would recommend that techniques be explored for schema driven synthetic data generation to preserve anonymity and yet make it possible to share data with researchers. Privacy preserving techniques like differential privacy could also be explored as a way to get a tradeoff between utility and privacy when sharing data

Naive Bayes as implemented in this project would not be feasible to implement in an actual system. Investigation into why the classifier does not work well with this data could provide further insight. More time was invested into getting the data into a usable state than could be allocated to further analysis of these results.

Future work might include looking into implementation of a system for defect detection. There is also room to expand on which models were used, refinements of the current models to predict the next step in the process and achieve better results, and investigating combinations of the approaches to enact some redundancy.

## Conclusion

Our biggest learning in this paper has been the realization of the effort required in understanding and preprocessing the data prior to building machine learning models to solve specific questions. Several sessions were dedicated to just understanding and discussing the data so we could apply the right models. We ended up writing more code to anonymize and preprocess the data than to actually build and test the models.

Our contributions in light of our experiences and findings can be listed as follows:

- An automated and efficient way of randomizing and anonymizing the data. The implementation we came up with allows us to identify the columns to be anonymized, randomize the values and index them in a way that we can easily decrypt them needed with simple join conditions using SQL.
- We showed that on this data, decision trees and decision tree based ensemble methods provided high levels of performance, whereas naive Bayes is not a good fit. LSTMs may also be a promising approach. However, these models are supervised, and they may be generalized to other data sets only if those data sets are labeled. Conversely, Word2Vec may be a promising approach when applied to unlabeled data sets.
- A repeatable process flow for feature engineering for the specific data we were working with. We leave it to future work to generalize the concept so machine learning algorithms can have feature-sets that are more plug and play than vanilla datasets.

Given that we know the effectiveness of the current solutions that are implemented for defect learning in the world where the data came from, we feel our work has contributed to applying machine learning to a real world business problem. Researchers in the organization that shared data with us can take advantage of the results we have found and the models we have built. The in house team can apply the feature engineering techniques we have applied to a broader set of data points and apply the algorithms to aid quality assurance in reducing defects.

With relative ease of implementation and high performance scores, the machine learning algorithms that we have shown to be successful in this domain are potentially viable solutions in a real-world setting. The economical advantage to utilizing the machine learning models for identifying defects would lie in reducing the waste, both in terms of time and product.

## References

1. Aghdam, S. R., Amid, E., and Imani, M.F. “A Fast Method of Steel Surface Defect Detection Using Decision Trees Applied to LBP Based Features.” *2012 7th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, 2012, doi:10.1109/iciea.2012.6360951.

2. Ahirwar, D.K., Saxena, S.K., and Sisodia, M.S. “Anomaly Detection by Naive Bayes & RBF Network”. *International Journal of Advanced Research in Computer Science and Electronics Engineering 1,* 1 (2012), 14-18.
3. Almas, A., Farquad, M.AH, Avala, N.S, and Sultana, J. “Enhancing the Performance of Decision Tree: A Research Study of Dealing with Unbalanced Data”. *Seventh International Conference on Digital Information Management (ICDIM 2012)*, 2012, doi:10.1109/icdim.2012.6360115.
4. Aydin, O. and Guldamlasioglu, S. ‘‘Using LSTM networks to predict engine condition on large scale data processing framework,’’ in *Proc. Int. Conf. Elect. Electron. Eng*., pp. 281–285, 2017.
5. Chandola, V., Banerjee, A., and Kumar, V. “Anomaly detection: A survey.” *ACM Comput. Surv. 41,* 3 (July 2009), 15:1-15:58.
6. Cho, K., van Merrienboer, B., Gulcehre, C., Bougares, F., Schwenk, H. and Bengio, Y. 2014, Learning phrase representations using RNN encoder-decoder for statistical machine translation. in *Conference on Empirical Methods in Natural Language Processing* (EMNLP 2014).
7. de Bruin,T., Verbert, K. and Babuska, R. "Railway Track Circuit Fault Diagnosis Using Recurrent Neural Networks", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 3, pp. 523-533, March 2017.
8. Dunning, T and Friedman, E. (2015) available at https://www.oreilly.com/library/view/sharing-big-data/9781491953624/ (as seen on 12/15/2018).
9. Fox, A. and Weisberg, S. “Nonparametric Regression in R: An appendix to *An R Companion to Applied Regression,* third edition”. Available at https://socialsciences.mcmaster.ca/jfox/Books/Companion/appendices/Appendix-Nonparametric-Regression.pdf. 2018. https://en.wikipedia.org/wiki/Local_regression
10. Gers, F.A. and Schmidhuber, J. "Recurrent nets that time and count", *Proc. IJCNN'2000 Int. Joint Conf. Neural Networks*, 2000.
11. Hochreiter, S. and Schmidhuber, J. “Long short-term memory”. *Neural computation*, 9(8):1735–1780, 1997.
12. Huang, Y.J., Pan, C.L., Lin, S.C., and Guo, M.H. Machine-Learning Approach in Detection and Classification for Defects in TSV-Based 3-D IC. *IEEE Transactions on Components, Packaging and Manufacturing Technology 8,* 4 (2018), 699-706.
13. “Intelligent Approach to Zero-Defect Manufacturing - Horizon 2020 - European Commission.” *European Commission*, 26 Mar. 2015, ec.europa.eu/programmes/horizon2020/en/news/intelligent-approach-zero-defect-manufacturing.
14. Jian, L., Wei, H., and Bin, H. “Research on Inspection and Classification of Leather Surface Defects Based on Neural Network and Decision Tree.” *2010 International Conference On Computer Design and Applications*, 2010, doi:10.1109/iccda.2010.5541405.
15. Levy, O., Goldbeg, Y., and Dagan, I. “Improving Distributional Similarity with Lessons Learned from Word Embeddings”. 2015.

16. Li, W., and Li, Q.X. “Using naive Bayes with AdaBoost to enhance network anomaly intrusion detection”. *Proceedings - 3rd International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2010* (2010), 486-489.
17. Muda, Z., Yassin, W., Sulaiman, M.N., and Udzir, N.I. “Intrusion detection based on K-means clustering and Naive Bayes classification”. *Proceedings of the 2011 7th International Conference on Information Assurance and Security, IAS 2011,* July (2011), 192-197.
18. Perreault, L., Berardinelli, S., Izurieta, C., and Sheppard, J. Using Classifiers for Software Defect Detection. *26th International Conference on Software Engineering and Data Engineering.* 2017.
19. *Understanding LSTM Networks*. 27 Aug. 2015, web.stanford.edu/class/cs379c/class_messages_listing/content/Artificial_Neural_Network_Technology_Tutorials/OlahLSTM-NEURAL-NETWORK-TUTORIAL-15.pdf.
20. Ye, R., Pan, C.S., Chang, M., and Yu, Q. (2018). Intelligent defect classification system based on deep learning. *Advances in Mechanical Engineering*, 10(3):1-7.
21. Zhang, K., Xu, J., Min, M.R., Jiang, G., Pelechrinis, K., and Zhang, H. "Automated it system failure prediction: A deep learning approach", *IEEE International Conference on Big Data*, pp. 1291-1300, 2016.

## Appendix

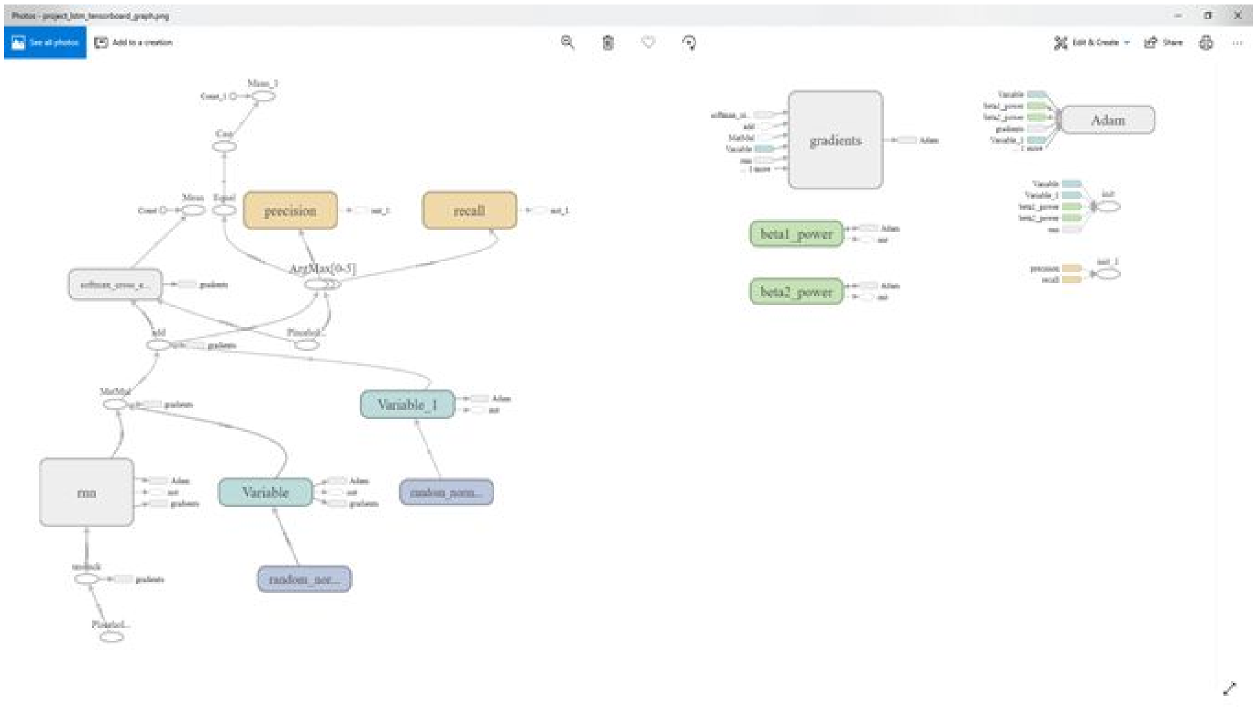
**1a.**

| | |
|---|---|
| ID | ID of the defect. A lot could have multiple defects, therefore multiple IDs |
| FROM OPERATION | The station where the problem was detected |
| LOT | Unique identifier for the widget |
| TIME CREATED | When the problem was identified and the lot was taken off the assembly line |
| PATH | The path that the lot took after defect was diagnosed |
| TIME DISPOSITIONED | Time the decision to either use-as-is, rework or scrap is made |
| TECH PROBLEM CODE CATEGORY | The category of the problem |
| REWORK COUNTER | The number of times the lot has already been reworked |
| TO OPERATION | The station where the lot is supposed to be repositioned after the decision |
| BAD DECISION | Y/N field that labels a decision as bad, which typically means ineffective |
| DISPOSITION | The decision that was made (Label) |
| MEAS AVG | Average value of the measurement taken at the operation for a lot |
| TARG AVG | The limit set by engineers for each measurement type |
| PASSFAIL | Whether the measurement pass or failed. If lot measurement is outside of the target average it tends to be an automatic failure |
| PDATETIME | Time that the measurement was taken |
| MEAS TYPE | Type of the measurement |
| TECH_COMMENTS | Free hand comments made by the technician regarding the problem |
| OPR_COMMENTS | Free hand comments made by the operator regarding the problem |

Frequency and Data Types of Non Time Columns

| Column Name | Frequency | Data Type |
|---|---|---|
| ID | 2249 | Categorical |
| FROM_OPERATION | 208 | Categorical |
| LOT | 820 | Categorical |
| PATH | 18 | Categorical |
| TECH_PROBLEM_CODE_CATEGORY | 9 | Categorical |
| REWORK_COUNTER | 11 | Categorical |
| TO_OPERATION | 245 | Categorical |
| BAD_DECISION | 2 | Boolean |
| DISPOSITION | 1 | |
| MEAS_AVG | 6927 | Continuous |
| PASSFAIL | 34 | Categorical |
| TARG_AVG | 398 | Continuous |
| MEAS_TYPE | 736 | Categorical |

LSTM - tensorboard graph


Photos - project_lstm_tensorboard_graph.png
See all photos
Add to a creation
Edit & Create
Share
Mean_1
Const_1
Cast
Const
Mean
Equal
precision
recall
softmax_cross_e...
ArgMax[0-5]
add
Placehol...
MatMul
Variable_1
rnn
Variable
random_norm...
random_nor...
unstack
Placehol...
gradients
Adam
init
init_1
beta1_power
beta2_power