Proceedings of the ASME 2020
International Design Engineering Technical Conferences
and Computers and Information in Engineering Conference
IDETC/CIE2020
August 17-19, 2020, Virtual, Online

DETC2020-22346

OPTIMIZING AN ALGORITHM FOR DATA MINING A DESIGN REPOSITORY TO AUTOMATE FUNCTIONAL MODELING

Alex Mikes, Katherine Edmonds, Robert B. Stone, Bryony DuPont*
Design Engineering Laboratory
School of Mechanical, Industrial &
Manufacturing Engineering
Oregon State University
Corvallis, Oregon 97331
Email: MikesA@oregonstate.edu
Email: EdmondKa@oregonstate.edu
Email: Rob.Stone@oregonstate.edu
Email: Bryony.DuPont@oregonstate.edu*

ABSTRACT

The purpose of this research is to find the optimum values for threshold variables used in a data mining and prediction algorithm. We also minimize and stratify a training set to find the optimum size based on how well it represents the whole dataset. Our specific focus is automating functional models, but the method can be applied to any dataset with a similar structure. We iterate through different values for two of the threshold variables in this process and cross-validate to calculate the average accuracy and find the optimum values for each variable. We optimize the training set by reducing the size by 78% and stratifying the data, whereby we achieve an accuracy that is 96% as good as the whole training set and takes 50% less time. These optimum values can be used to better predict the functions and flows of any future product based on its constituent components, which can be used to generate a complete functional model.

1 INTRODUCTION

There are many tools to help design engineers in the early phases of modern product design [1]. Functional decomposition is one of the tools that takes considerable investment to master but offers key insights during the concept generation phase. This is a well-known technique that discretely connects the subfunctions and flows in a product that can be formatted in a graphical representation known as a functional model.

Functional models are helpful in the early design phase because they allow the designer to visualize the subfunctions and flows throughout a product and ensure adherence to the customer requirements [2] [3]. However, they may be highly subjective, time-consuming, and difficult to make, which is why they are often omitted from the design process [4]. Previous work has helped to standardize the words used for functions and flows to increase consistency in the language, known as the Functional Basis terms [5] [6]. There are also Component Basis terms, which help increase consistency in the language of product components [7]. Despite this work, functional models are still highly subjective and variable in terms of format and accuracy [8]. Automating the process of making functional models will increase the use of this helpful tool in the early design phases by decreasing the amount of time required to make them and increasing the accuracy and consistency between different designers as data is entered into the repository.

The ultimate goal of this research is to automate the generation of functional models to serve two purposes: 1) ease the

*Corresponding author.

workload associated with archiving products digitally; 2) increase the likelihood that designers will utilize a functional decomposition approach during product design. This paper primarily reports on effects that align with purpose 1. In order to accomplish this, we first automate the process of finding the functions and flows that a product's constituent components usually perform. After finding the most common functions and flows, we can order them based on simple guidelines known as grammar rules, which dictate the order of functions and flows within a component [8] [9] [10]. If our algorithm found the most common functions and flows of an *electric cord* component to be *export electrical energy, transfer electrical energy, and import electrical energy*, an example of grammar rules would be that *import* must come before *export* and *transfer* must come before *export* for the same flow. The linear functional chain for the flow of *electrical energy* would then be: *import electrical energy* $\rightarrow$ *transfer electrical energy* $\rightarrow$ *export electrical energy*. After finding the linear functional chains of every component in a product, we can then combine them all into a functional model based on the connections of components within the product.

In this process of automating the functional chains of components, there are several variables in the algorithm that determine which functions and flows are predicted to be in future components. This work focuses on optimizing these variables to find the values that give the highest accuracy with the least amount of time spent on computation.

In our previous work, we hypothesized that we could increase the accuracy of the automated generation of functional models if we only found the functions and flows for components in products that shared a similar component that was central to the functionality of the products [11]. For example, one of our datasets only contained products with a blade component, such as jigsaws, lawnmowers, and food slicers. We found that restricting the dataset based on a single component does not increase the accuracy of the automation. This work explores this idea further by developing the overall similarity between products that encompass more than a single component and tests if learning from highly similar products is any better at predicting functions and flows than somewhat similar products.

Our previous work also showed that increasing the size of the dataset used for data mining increases the accuracy of our automation algorithm. When restricting the dataset based on similar components, we were essentially restricting the extent of that data from which the algorithm could learn. This work explores this idea further to find if there is a limit to the increase in accuracy with an increase in size.

2 BACKGROUND

A design repository is a database that stores product data that can be used to inform data-driven design decisions [12]. In this work, we create an algorithm that mines data from a design repository that stores component-centric design information about products. This multi-decade project is housed at Oregon State University and it is simply called The Design Repository[1] [10] [12] [13] [14] [15] .

Similarity in design has been studied to increase consistency and understand the overlap between products in a product family within the portfolio of a company [16]. If a company can make a modular base of designs, it can quickly adapt to changes in the market with negligible changes in design and manufacturing processes, allowing them to put products on the market cheaper and faster [17]. There are many ways to measure the similarity between products [18]. This work measures the similarity between products as the percentage of components that they have in common. The similarity metric is calculated for all products in the dataset and then used for creating subsets of the data in the optimization, but any similarity metric could be substituted into the algorithm.

2.1 Data Mining, Classifiers, and Cross-Validation

Data mining is the process of examining information and recording patterns to discover knowledge about data [19]. A common use of data mining is to create a tool called a classifier, which uses the patterns found within data to predict if a new observation belongs to a class [20]. An example would be a classifier that predicts if an email is *spam* or *not spam* and moves them either to the spam folder or the regular inbox [21]. The classifier is "trained" by mining a dataset that contains examples of emails that are labeled as either *spam* or *not spam*, it finds the patterns within the data, and uses that information to predict if any other email is *spam* or *not spam* [22]. The accuracy can be quantified by making predictions for emails that are already labeled and recording the number of correct and incorrect predictions.

A classifier does not know the class to which the new observation belongs– instead it calculates a probability of class membership and the designer chooses the threshold of probability to switch between classes. This is known as a classification threshold, and it is directly linked to the accuracy of a classifier [23]. For example, if a designer sets the classification threshold for a spam email classifier at 99%, it would only label emails as *spam* that have a very high probability of being spam. This would not label very many emails as *spam* and there is a good chance that some spam emails would end up in the regular inbox. However, if a designer sets the classification threshold to 50% then there is a good chance that some emails that are *not spam* would be mis-classified as *spam* and moved to the spam folder, which is why some companies suggest you check your spam folder to find missing emails from them. The classification threshold determines the accuracy of the classifier and the optimum classifi-

[1] A basic web interface for The Design Repository is available at ftest.mime.oregonstate.edu/repo/browse

cation threshold changes for each dataset. This work data mines a dataset with product information, uses a classifier to learn the most common functions and flows for a component, and calculates the accuracy at different classification thresholds to find the best value for each dataset.

Optimizing a classification threshold is not a novel topic, and many researchers have shown effective methods for finding the best values for these numbers in imbalanced datasets such as ours (we describe how our data is imbalanced in the next section on the F1 score) [24]. Due to the unique structure of product data, it is not helpful to use these methods because they require analyzing all components as if they are product-agnostic. As designers, we know that the components of a product are intentionally symbiotic, and analyzing them independently of each other would be diminishing the work of the designers that created these products. A generalized example structure of our product data is shown in Table 1.

TABLE 1: EXAMPLE PRODUCT DATA STRUCTURE

Product ID	Component	Function-Flow
Product 1	Component 1	Function-Flow 1
		Function-Flow 2
		Function-Flow 3
	Component 2	Function-Flow 1
		Function-Flow 2
Product 2	Component 1	Function-Flow 1
	Component 2	Function-Flow 1
		Function-Flow 2
		Function-Flow 3

A common tool for calculating the accuracy of a classifier is known as cross-validation. This general term refers to any method that separates the original dataset into two parts: one used for training and one used for testing [25]. The training set is the information that the data-mining algorithm uses to find patterns and train the classifier. The testing set is the information used to test the classifier and calculate how well it performed. Testing with data from which the classifier did not learn is essential for reducing bias in the results [26]. Due to the variability in most datasets, cross-validation is often performed multiple times with different testing and training sets and averaged over all iterations. There are several methods of cross-validation, for this work we use leave-one-out cross-validation. As the name suggests, we pull a single product from the dataset to act as the testing data, and the remaining products constitute the training set [27].

2.2 Precision, Recall, and the F1 Score

Simple accuracy for a classifier is a ratio of the correct predictions to the total predictions, This number does not give a holistic representation of how well the classifier performs for most datasets, including ours. To account for the variety of ways in which a classifier can be correct and incorrect, we use the metrics of precision, recall, and the F1 score. The F1 score is the harmonic mean of precision and recall and is a replacement for simple accuracy.

A classifier learns from the training set to predict whether or not an item from the testing set is within a class, and that prediction is validated based on the known data from the testing set. Simply counting correct responses misses some of the additional ways in which the automation can be wrong. Precision, recall, and the F1 score account for these cases by using the confusion matrix shown in Table 2 to calculate ratios of the true positives, false positives, and false negatives [28].

TABLE 2: ACCURACY CONFUSION MATRIX

		Predicted?	
		Yes	No
Actual?	Yes	True Positive	False Negative
	No	False Positive	True Negative

Looking at the calculation for simple accuracy in Equation 1, if the value for True Negative was much larger than any other value, the accuracy would tend toward 100%. The abbreviations in the equations–*TP, TN, FP,* and *FN*–correspond to the values in the confusion matrix in Table 2 of True Positive, True Negative, False Positive, and False Negative, respectively. Consider a classifier that looks at medical data and predicts whether or not a patient has Parkinson's Disease. According to records, approximately one percent of people over the age of 60 have Parkinson's Disease [29]. If the classifier said that every patient over 60 did not have Parkinson's, it would be right 99% of the time because this is a True Negative and most people don't have Parkinson's. This high accuracy is misleading and not helpful. Our data is similarly imbalanced and simple accuracy is a misleading indicator of automation performance. Precision, recall, and the F1 score account for this imbalance and give a better understanding of the accuracy of a classifier.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{1}$$

Precision is the ratio of correct predictions to all predictions made by the classifier (Eqn. (2)). This number is the ratio of

predictions that were identified as being in the product that are actually in the product.

Recall is the ratio of correct predictions to all actual results made by the classifier (Eqn. (3)). This number is the ratio of the actual results that were correctly predicted. Recall is representative of the confidence that no positives have been missed and precision is representative of the confidence in the True Positives.

The F1 Score is the harmonic mean of precision and recall and equally balances the importance of the two metrics. The harmonic mean is defined as the reciprocal of the arithmetic mean of the reciprocals of the observations. For two numbers, this equation simplifies to the version shown in Eqn. (4) as the harmonic mean of Precision and Recall. This combination of values replaces the simple accuracy metric and provides more meaningful insight into the ability of an automation algorithm to predict results. Due to the multiplication in the numerator, very low values of precision or recall will have very low F1 scores.

$$Precision = \frac{TP}{TP+FP} \tag{2}$$

$$Recall = \frac{TP}{TP+FN} \tag{3}$$

$$F1 = \frac{2 * precision * recall}{precision + recall} \tag{4}$$

The F1 score is one variation of the general F score in which the values of precision and recall are equally weighted in the harmonic mean. The F1 score is preferred for datasets that have disproportionate values for True Negative results when the cost of a wrong prediction is not very high. This can be changed, for example, if the cost of a false positive is high (misdiagnosing a disease), then an F2 score can be used to weight the precision value differently in the harmonic mean.

This work uses the F1 score to calculate the accuracy of a classifier at different classification and similarity thresholds.

3 METHODS

While the ultimate goal of this research is to generate functional models for a product based on its constituent components, the intermediate step is to predict the functions and flows for each component individually. We accomplish this by finding the most common functions and flows for each component and assuming that future components will likely have some of the same results. This work and the following methods focus on optimizing two of the variables in this process that are used to tune the algorithm that generates a functional model, which are not the same steps that we use when strictly predicting functions and flows for the components of a product. The results of the optimization approach will help us increase the accuracy of the predictions and create better functional models.

We separate the dataset into two parts: the testing set and the training set. The testing set is a single product because the input for the automatic generation of functional models is a single product. The training set is some subset of the rest of the products that are above a threshold of similarity (the similarity threshold is expanded upon later). In this work, the dataset of all consumer products within the design repository consists of 139 products, so each testing set is one product and the training sets consist of some subset of the remaining 138 products. The testing set is comprised of products for which the functions and flows are known for each component, so that information is used to verify what the automation algorithm predicts, and we calculate how well the algorithm performed with the F1 score.

The overall algorithm is shown below. Each step is detailed in the following sections.

1. Query repository
2. Compute similarity between all products
3. Iterate through different values for two algorithm variables
4. Cross-validate dataset to find accuracy for each combination of variables
5. Find optimal values for variables

3.1 Query Repository and Find Similarity

We query for the product ID, component, subfunction, and inflow data for each consumer product in the repository. This data is imported to Python v3.7 for computation [2] [30]. For this work, the similarity between each product is determined by the percentage of components they have in common. The similarities are computed for each combination of products in an *n x n* matrix with the product IDs as the row and column headers. The main diagonal of this matrix consists of ones because every product is 100% similar to itself, but the matrix is not symmetric because each product can contain a different number of components. For example, consider a case where Product 1 has 20 components and Product 2 has 40 components. If they have 10 components in common, the similarity between Product 1 and Product 2 is 10/20 = 50%, but the similarity between Product 2 and Product 1 is 10/40 = 25%. The first product of the pair is known as the "generating" product. A 5x5 subset of the 139x139 similarity matrix is shown in Table 3 (The same matrix is in Appendix A with product names instead of product IDs).

[2]The software used for all of the computation in this and previous work is available at https://github.com/AlexMikes/AutoFunc

TABLE 3: 5x5 SUBSET OF SIMILARITY MATRIX

ID	286	203	206	288	591
286	1.000	0.294	0.192	0.667	0.313
203	0.263	1.000	0.269	0.556	0.313
206	0.263	0.412	1.000	0.778	0.438
288	0.316	0.294	0.269	1.000	0.188
591	0.263	0.294	0.269	0.333	1.000

3.2 Cross-Validate Dataset to Find Accuracy

As described in the background (section 2.1), we use leave-one-out cross-validation to calculate the accuracy of our classifier. At the beginning of each iteration, one product is extracted from the main dataset and it is used as the testing set. The rest of the products are *eligible* to be in the training set but are only used if they fall within the threshold of similarity with the product in the testing set (we explain the process of selecting products for the training set in the next subsection). Once training and testing sets have been established, we use the training set to find and predict the functions and flows for each of the components in the testing set. A graphical example of leave-one-out cross-validation is shown in Table 4.

TABLE 4: LEAVE-ONE-OUT CROSS-VALIDATION EXAMPLE

	Dataset				
	Product 1	Product 2	Product 3	Product 4	Product 5
Iteration #					
1	Product 1	Product 2	Product 3	Product 4	Product 5
2	Product 1	Product 2	Product 3	Product 4	Product 5
3	Product 1	Product 2	Product 3	Product 4	Product 5
4	Product 1	Product 2	Product 3	Product 4	Product 5
5	Product 1	Product 2	Product 3	Product 4	Product 5

Train Test

The testing set comes from the original data, so the actual functions and flows are known and can be used to validate whether or not the predicted functions and flows are correct. We use this information to calculate the precision, recall, and F1 scores 139 times so that each product is used for the testing set.

3.3 Iterate Through Classification and Similarity Thresholds

We find the F1 score while iterating through two of the automation algorithm variables for every combination of training and testing sets. For this work, we increment the similarity threshold from 0 to 100 in increments of ten and the classification threshold from 10 to 100 in increments of five. We chose these increments through experimenting to find where the resolution became too large to notice small changes. There are 18 values of classification thresholds (10, 15, 20, 25...95%) and 10 values of similarity thresholds (0, 10, 20...90%). There are 139 combinations of testing and training sets, the same number of products in the dataset.

After applying a similarity threshold, the training set decreases in size. There are 139 products in the whole dataset. One is used for the testing set and the other 138 are *eligible* for the training set. At each similarity threshold, the algorithm chooses the products that meet the similarity threshold and builds a training set from those products. If the similarity threshold is 0% then all 138 products are in the training set, but any similarity threshold higher than that will cause the training set to be smaller than 138.

The algorithm finds the frequency of each combination of component, function, and flow. For each component, these combinations sum to 100% to capture all of the functions and flows in the training set. Each component, function, and flow combination is an individual percentage that represents the frequency of each combination out of all of the functions and flows for that component. The classification threshold is a cutoff point for the sums of these individual frequencies per component. An example of some component, function, and flow combinations with their individual and summed frequencies for the component *Battery* is shown in Table 5. These combinations are sorted from largest to smallest individual frequency. The right column shows how a 70% threshold determines which combinations are predicted for future instances of the *Battery* component. The first two combinations have individual frequencies of 34% and 28% respectively, so they sum to 62% of the total functionality of that component. With a threshold of 50%, only these two functions and flows would be kept. With a threshold of 70%, more functions and flows would be added to the sum. The next function and flow combination has an individual frequency of 17%, which brings the sum to 79%, which satisfies the threshold and those top three functions and flows are predicted for future *Battery* components.

TABLE 5: EXAMPLE OF CLASSIFICATION THRESHOLD APPLICATION

Battery	Individual frequency	Running sum of frequencies	70% Threshold
Store electrical	0.34	0.34	Keep
Supply electrical	0.28	0.62	Keep
Transfer electrical	0.17	0.79	Keep
Import electrical	0.12	0.91	Reject
Position solid	0.09	1	Reject

Our algorithm iterates through the testing set product in an outermost loop, the similarity threshold for the second loop, and the classification threshold for the third loop. For example, suppose we are using the Delta jigsaw as a testing product. We find all products that fall within the similarity threshold, then find the F1 scores at all 18 classification thresholds (10, 15, 20, 25…95%). We increment the similarity threshold by ten percent and again find the F1 scores for all 18 classification thresholds. We repeat this process for each similarity threshold, then change testing products and repeat the entire process. We perform 180 calculations for each testing/training set (10 similarity thresholds x 18 classification thresholds), which is repeated 139 times (once for each product as the testing set).

3.4 Find Optima

The accuracy of the automation algorithm is captured by the F1 scores that are calculated for each combination of input variables for each combination of testing and training sets. We average the 139 F1 scores for each combination of similarity and classification thresholds to find the combination that yields the highest average F1 score. This is unique for each dataset, so this cross-validation method can be performed to find the optimum values before using it to predict any results. After calculating all of the F1 scores, we find the optimum values and use them for our classifier to make more accurate future predictions.

4 RESULTS

The initial objective of this research was to find the optimum values for classification and similarity thresholds for a dataset used for automating functional modeling. All of the previous methodologies address this objective and the results are following in Section 4.1. In the process of developing those methodologies and analyzing the results, we explored the idea of creating an optimized training set. This is a second optimization that was inspired by the interesting observations from the first optimization. The methods and results for the optimized training set are discussed in Section 4.2.

4.1 Optimizing Thresholds

The highest average of 139 F1 scores for this dataset was 0.445 with a **similarity threshold of 0.2** and a **classification threshold of 0.55**. Figure 1 shows a 3D scatter plot of all data points for the calculation of these numbers. Each dot represents an average of 139 numbers.

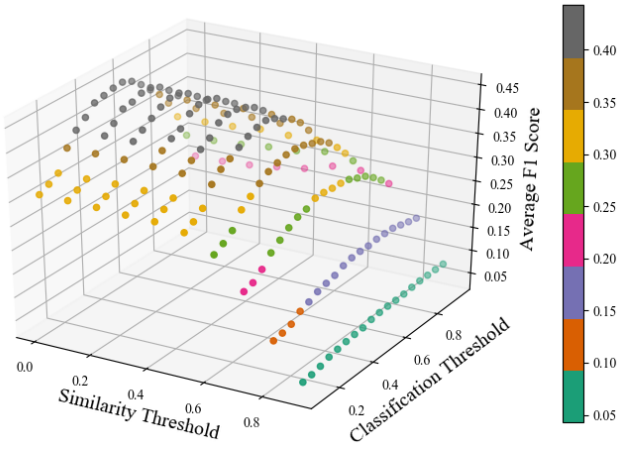

FIGURE 1: 3D SCATTER PLOT OF SIMILARITY THRESHOLD, CLASSIFICATION THRESHOLD, AND F1 SCORE

Figure 2 shows a plot of the average F1 scores against the classification threshold for the optimum similarity threshold of 0.2. This line can also be seen on the 3D scatter plot and clearly shows a maximum value of the F1 score at the optimum classification threshold.

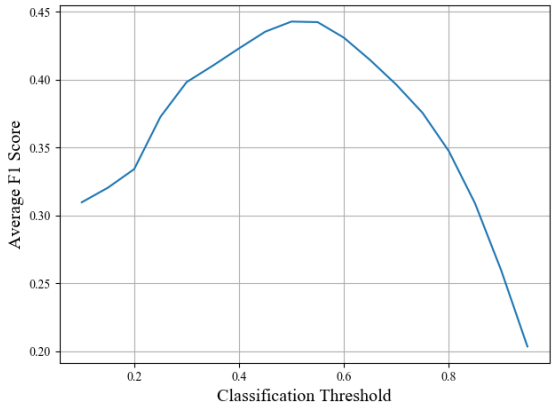

FIGURE 2: AVERAGE F1 SCORES VS. CLASSIFICATION THRESHOLD AT 0.2 SIMILARITY THRESHOLD

Adjusting the similarity threshold affects the number of products in the training set. Figure 3 shows the number of products in the training set decrease as the similarity threshold increases. Without a similarity threshold (0%), all 138 products are in the training set. With a similarity threshold of 50%, there are 28 products in the training set.

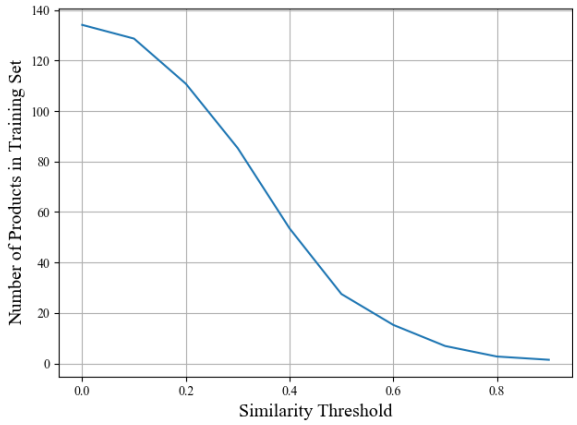

FIGURE 3: NUMBER OF PRODUCTS VS. SIMILARITY THRESHOLD

Figure 4 shows how the average F1 score changes with the number of products in the training set. There is a clear plateau around 30 products where adding additional data has very little effect on the accuracy. The similarity threshold is the independent variable that we are optimizing and the number of products in the training set is the dependent variable that changes as the result of changing the similarity threshold.

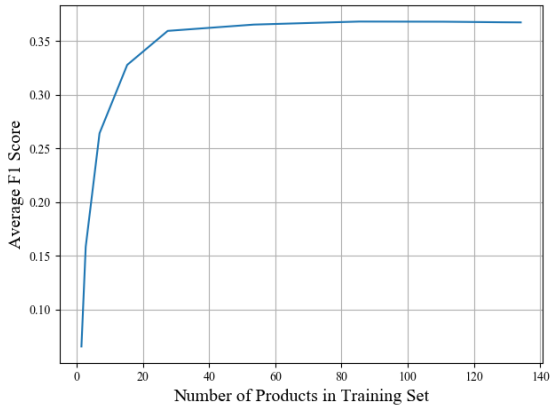

FIGURE 4: AVERAGE F1 SCORE VS. NUMBER OF PRODUCTS

Figure 5 shows how the average F1 score changes with the similarity threshold at the optimum 0.55 classification threshold. There is little change from 0% to 50% and then a sharp decrease in accuracy with increasing similarity threshold.

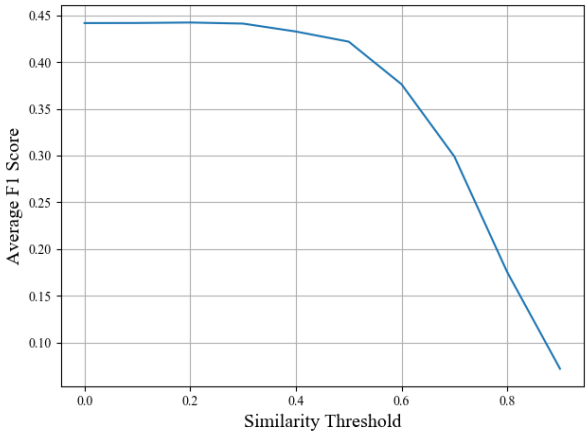

FIGURE 5: F1 SCORE VS. SIMILARITY THRESHOLD

Figure 6 shows the computation time change with the number of products in the training set. Training sets with 30 products and 138 products have roughly the same accuracy, but computation time is roughly doubled. Our previous work showed that using more data in the training set yields better results, but this work showed that idea is true to a point, after which there is no benefit to the accuracy despite an increase in computation time.

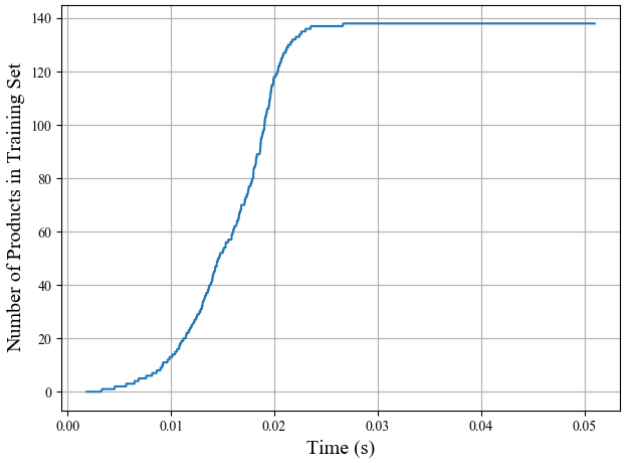

FIGURE 6: NUMBER OF PRODUCTS IN TRAINING SET VS. COMPUTATION TIME

Combining the results from Figures 1-6, we find that for our data, we obtain roughly the same *average* accuracy by training from 30 products and 130 products. However, using less than 20 products results in a steep decline in average accuracy. If we only used 30 random products in our training sets, we would not have the same accuracy for each testing set because the products would change and some would be better or worse than the average. Despite finding some optimum number of products that maximizes average accuracy and minimizes computation time, using all available data will increase the likelihood of being able to predict the functions and flows for a future product at the expense of computation time.

4.2 Optimizing Training Set

To further explore this idea of an optimal dataset, we looked at which training sets had the best performance with high average F1 scores and low numbers of products. To ensure that the smaller training set was still representing all of the data, we filtered training sets by the ratio of unique component basis terms that are covered by the products in the training set. We calculated the total number of unique component basis terms in the entire dataset, and for each training set, we calculated the ratio of unique component basis terms covered by the products in the training set.

The results in Figure 4 show the accuracy increasing as products are added to the training set, and then a plateau after which there is little change as more products are added. We used this data to plot the *change* in the average F1 score versus the number of products. When this line goes to zero, adding more products to the training set does not change the average F1 scores. We overlaid the data showing how the ratio of component basis terms in the training set changes with the number of products in the dataset. At the point where the F1 score stops changing, there are an average of 28 products and a ratio of 75% of component basis terms in the training sets. These results are shown in Figure 7.

We used the results from Figure 7 to find the products that appear most often in the optimal training sets and keep those as a static training set to cross-validate the rest of the data. Figure 8 shows the ratio of component basis terms covered vs. the number of products in the training sets. Each dot in the scatter plot represents a training set. The lines are thresholds used for highlighting the "best" training sets, with a ratio of component basis terms greater than 70% and less than 40 products. The optima were found around 75% and 30 products, so we used a buffer of 10 products and a five percent ratio of component basis terms for this analysis.

After finding the optimal training sets, we looked at which products most often appear in these sets. While each product has its own ratio of component basis terms, they are relatively low individual numbers. The benefit of a high ratio of component

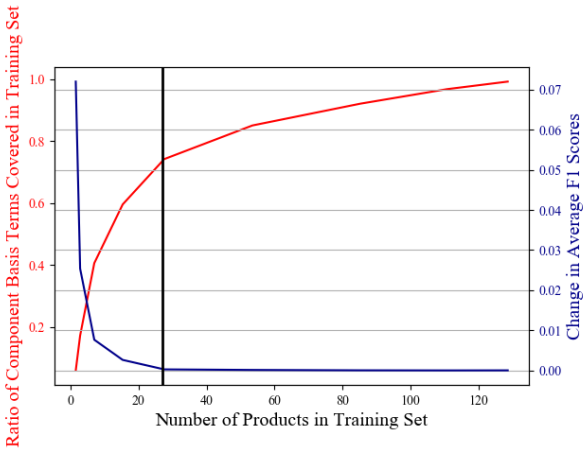

FIGURE 7: F1 SCORE AND RATIO OF COMPONENT BASIS TERMS VS. NUMBER OF PRODUCTS IN TRAINING SET

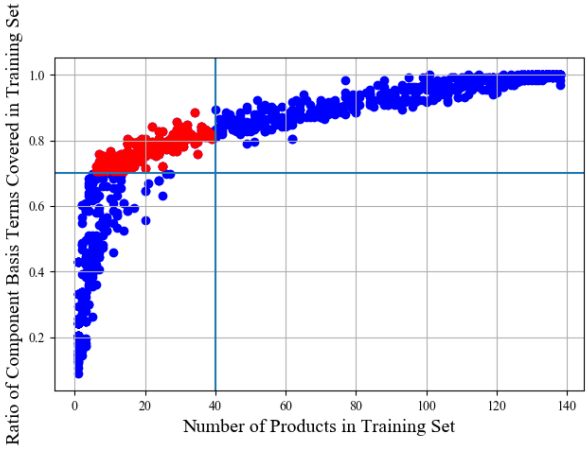

FIGURE 8: RATIO OF COMPONENT BASIS TERMS COVERED VS. NUMBER OF PRODUCTS IN TRAINING SET WITH THRESHOLDS APPLIED TO HIGHLIGHT THE BEST TRAINING SETS

basis terms in the training set comes from combining products that each cover different component basis terms to give a high overall ratio. Figure 9 shows the top 20 products that appear most often in the "best" training sets (The bottom 10 were removed for figure clarity; the full 30 can be seen in Figure 11 in Appendix B). The most frequent product in these training sets is a Datsun truck, which is more complex and contains more components than many of the other products in the dataset. The Datsun truck has 205 appearances in the 217 optimal training sets, but only has a 12% individual ratio of component basis terms covered.

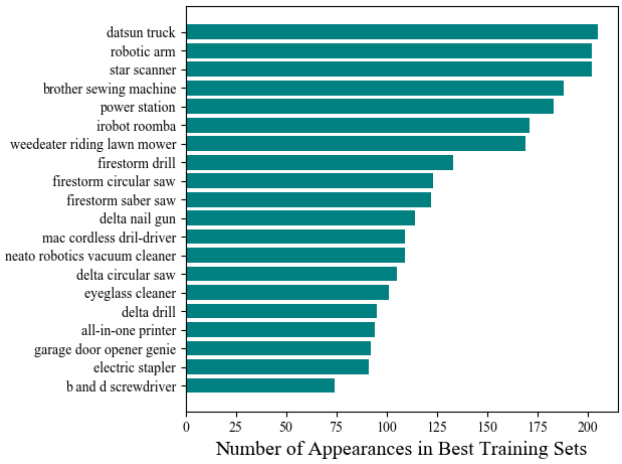

FIGURE 9: FREQUENCY OF PRODUCTS IN THE BEST TRAINING SETS

We used the 30 products that are most often found in the optimal training sets to make a static training set that we used to cross-validate the rest of the dataset (we will call these the 30 "best" products). Similar to the method explained previously, we used every other product in the dataset as a testing set with the 30 "best" products as a training set and found average F1 scores through each classification threshold. The reduced training set is 78% smaller than a full training set (30 products instead of 138 products). Figure 10 shows the same data from Figure 2 with the average F1 scores from the entire dataset alongside the average F1 scores from the reduced dataset of the 30 "best" products.

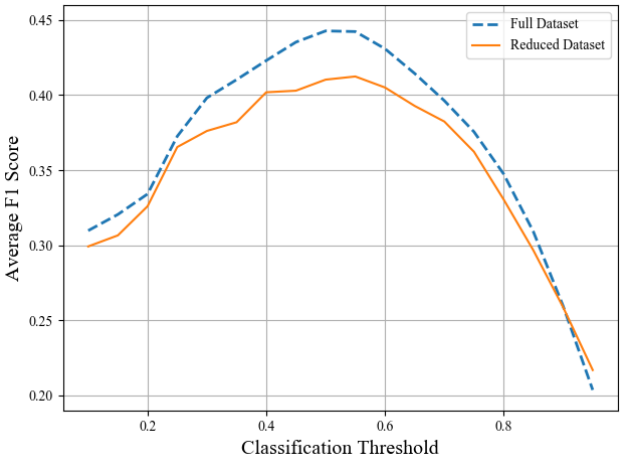

FIGURE 10: AVERAGE F1 SCORES VS. CLASSIFICATION THRESHOLDS FOR THE FULL AND REDUCED DATASETS

Using the optimized and reduced dataset gave us similar results as using the whole dataset. If the goal is to find the absolute best accuracy possible with little concern for data size, computation power, or time requirements, then restricting the dataset would not be necessary. If the goal is to get good accuracy while reducing data size, computation power, or time requirements, then using a subset of data that represents the whole dataset is ideal. The difference in average accuracies between the full dataset and the reduced dataset is the largest at the optimum classification threshold of 0.5. For this point, the F1 score for the reduced dataset is 0.4103 and for the full dataset is 0.4428. At this point of its worst performance, the reduced dataset is still 93% as accurate as the full dataset. At its best, the reduced dataset is 6% better than the full dataset at a classification threshold of 0.95. On average, the reduced dataset is 96% as accurate as the full dataset.

The computation time in Figure 6 shows that a training set of 30 products takes 0.012 seconds and a training set of 138 products takes 0.024 seconds, which means our optimized dataset takes 50% less computation time.

This method of reducing a dataset based on the ratio of the component basis terms in the training set is similar to stratification in cross-validation, which ensures a ratio of each class in a training set. Our method differs from stratified cross-validation because it does not ensure a ratio of class membership, but instead a ratio of representation of a type of data. The structure of product data has three levels, as shown previously in Table 1. In the hierarchy of **Product** → **Component** → **Function-Flow**, we are stratifying the ratio of the representation of the **Component** level. This entire method can be generalized to many applications of automation and data mining classifiers, especially those with a similar structure to product data. Our results show that reducing the data and stratifying the second level of the data hierarchy can yield results with an average of 96% accuracy in 50% of the time when compared to the full dataset.

4.3 FUTURE WORK

In future work, we will automate the ordering of functions within a flow based on existing grammar rules, as well as develop more grammar rules. We plan to treat each subfunction as potentially housing different input and output flows. For example, suppose the component *Electric motor* has the functions *Import electrical energy, Convert electrical energy,* and *Export mechanical energy*. We will update the algorithm such that the *Convert* subfunction accommodates different inflows and outflows, of *Electrical energy* and *Mechanical energy*, respectively.

Future work will also explore optimizing the classification threshold separately for individual components, as well as subassemblies or common groups of components. Additionally, while this work is a starting point that helps maintain accuracy at reduced computational cost, we intend to fully automate the

functional modeling of a product by building on this work.

5 CONCLUSION

Functional modeling is a helpful tool in the early design stages but is often omitted because of the time requirement. Even when functional models are used, the accuracy and consistency can vary between the designers that make them. Automating functional modeling will help standardize the format and syntax, decrease the time required to make functional models, and increase the prevalence and accuracy of functional models in design and design repositories. This work builds toward automating functional modeling by optimizing the variables for an algorithm that mines a design repository of product information and uses that to predict functions and flows of components. Optimizing these values will further increase the accuracy and consistency of automated functional models.

The accuracy of a classification algorithm depends on the values of threshold variables. The structure of product data requires iteration through these variables to determine their optimum values, as opposed to some types of problems that can be optimized analytically. This work uses a cross-validation method to optimize classification and similarity thresholds that are used for data mining a design repository to find the most common functions and flows for components as a step toward automating functional modeling.

To automate functional modeling, the designer will input a list of components and their connections within a product. We found that optimizing a classification threshold can increase the accuracy of this automation. We also found that enforcing a threshold of similarity between the input component and the products used in the training set can reduce computational expense while maintaining accuracy, but the results are variable and change with each product used for testing. Stratifying the reduced dataset helps minimize the variance and increase the average accuracy. We found that we could achieve 96% of the total accuracy with a stratified reduced dataset used for training. Optimizing the classification threshold but not enforcing the similarity threshold will maximize the number of products in the training set and maximize the overall accuracy when not cross-validating and evaluating averages. If reducing data size, computation power, or time requirements is necessary, then optimizing the classification threshold using a stratified subset of data can achieve similar results with somewhat reduced accuracy.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under Grant No. CMMI-1826469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Otto, K., and Wood, K., 2003. *Product design: techniques in reverse engineering and new product development*. Prentice Hall.

[2] Ullman, D. G., 1992. *The mechanical design process*, Vol. 2. McGraw-Hill New York.

[3] Pahl, G., and Beitz, W., 1984. "Engineering design: a systematic approach".

[4] Kurfman, M. A., Stock, M. E., Stone, R. B., Rajan, J., and Wood, K. L., 2003. "Experimental Studies Assessing the Repeatability of a Functional Modeling Derivation Method". *Journal of Mechanical Design,* ***125***, pp. 682–693.

[5] Stone, R. B., and Wood, K. L., 2000. "Development of a Functional Basis for Design". *Journal of Mechanical Design,* ***122***(4), dec, p. 359.

[6] Hirtz, J., Stone, R., McAdams, D., Szykman, S., and Wood, K., 2002. "A functional basis for engineering design: Reconciling and evolving previous efforts". *Research in Engineering Design,* ***13***, pp. 65–82.

[7] Kurtoglu, T., Campbell, M. I., Bryant, C. R., Stone, R. B., McAdams, D. A., et al., 2005. "Deriving a component basis for computational functional synthesis". In ICED 05: 15th International Conference on Engineering Design: Engineering Design and the Global Economy, Engineers Australia, p. 1687.

[8] Nagel, R. L., Vucovich, J. P., Stone, R. B., and McAdams, D. A., 2008. "A Signal Grammar to Guide Functional Modeling of Electromechanical Products". *Journal of Mechanical Design,* ***130***(5), may, p. 051101.

[9] Bohm, M. R., and Stone, R. B., 2010. "Form Follow Form - Fine Tuning Artificial Intelligence Methods". *Proceedings of the ASME 2010 International Design Engineering Technical Conferences and Computers and Information*, pp. 1–10.

[10] Sridharan, P., and Campbell, M. I., 2005. "A study on the grammatical construction of function structures". *Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM,* ***19***(3), pp. 139–160.

[11] Edmonds, K., Mikes, A., Stone, R. B., and DuPont, B., 2020. Data mining a design repository to generate linear functional chains: a step toward automating functional modeling. Under Review.

[12] Szykman, S., Sriram, R., Bochenek, C., Racz, J., and Senfaute, J., 2000. "Design repositories: engineering design's new knowledge base". *IEEE Intelligent Systems,* ***15***(3), may, pp. 48–55.

[13] Bohm, M. R., Stone, R. B., Simpson, T. W., and Steva, E. D., 2008. "Introduction of a data schema to support a design repository". *Computer-Aided Design,* ***40***(7), jul, pp. 801–811.

[14] Bryant, C. R., McAdams, D. A., Stone, R. B., Johnson, J.,

Rajagopalan, V., Kurtoglu, T., and Campbell, M. I., 2008. "Creation of Assembly Models to Support Automated Concept Generation". In ASME 2005 International Design Engineering Technical Conferences, pp. 259–266.

[15] Kurtoglu, T., Campbell, M. I., Bryant, C. R., Stone, R. B., and McAdams, D. A., 2005. "Deriving a Component Basis for Computational Functional Synthesis". *ICED 05: 15th International Conference on Engineering Design: Engineering Design and the Global Economy*, p. 4061.

[16] McAdams, D. A., Stone, R. B., and Wood, K. L., 1999. "Functional interdependence and product similarity based on customer needs". *Research in Engineering Design,* ***11***(1), pp. 1–19.

[17] Simpson, T. W., 2004. "Product platform design and customization: Status and promise". *Ai Edam,* ***18***(1), pp. 3–20.

[18] Thevenot, H. J., and Simpson, T. W., 2006. "Commonality indices for product family design: a detailed comparison". *Journal of Engineering Design,* ***17***(2), pp. 99–119.

[19] Fayyad, U. M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R., et al., 1996. *Advances in knowledge discovery and data mining*, Vol. 21. AAAI press Menlo Park.

[20] Weiss, S. M., and Kulikowski, C. A., 1991. *Computer systems that learn: classification and prediction methods from statistics, neural nets, machine learning, and expert systems*. Morgan Kaufmann Publishers Inc.

[21] Conway, D., and White, J. M., 2011. *Machine Learning for Email: Spam Filtering and Priority Inbox*. O'Reilly Media, Inc.

[22] Lanzi, P. L., 2000. *Learning classifier systems: from foundations to applications*. No. 1813. Springer Science & Business Media.

[23] Hastie, T., Tibshirani, R., and Friedman, J., 2009. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media.

[24] Bekkar, M., Djemaa, H. K., and Alitouche, T. A., 2013. "Evaluation measures for models assessment over imbalanced data sets". *J Inf Eng Appl,* ***3***(10).

[25] Geisser, S., 1975. "The predictive sample reuse method with applications". *Journal of the American statistical Association,* ***70***(350), pp. 320–328.

[26] Stone, M., 1974. "Cross-validatory choice and assessment of statistical predictions". *Journal of the Royal Statistical Society: Series B (Methodological),* ***36***(2), pp. 111–133.

[27] Duchesne, P., and Rémillard, B., 2005. *Statistical modeling and analysis for complex data problems*, Vol. 1. Springer Science & Business Media.

[28] Sokolova, M., Japkowicz, N., and Szpakowicz, S., 2006. "Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation". In Australasian joint conference on artificial intelligence, Springer, pp. 1015–1021.

[29] Reeve, A., Simcox, E., and Turnbull, D., 2014. "Ageing and parkinson's disease: why is advancing age the biggest risk factor?". *Ageing research reviews,* ***14***, pp. 19–30.

[30] AlexMikes, 2019. AlexMikes/AutoFunc: Automated Functional Representations, June.

Appendix A: 5x5 Similarity Matrix With Product Names

TABLE 6: 5x5 SUBSET OF SIMILARITY MATRIX WITH PRODUCT NAMES

Product	Air Hawg Toy Plane	Alcohawk Digital Alcohol Detector	Epson All-In-One Printer	Apple USB Mouse	Black & Decker Can Opener
Air Hawg Toy Plane	1.000	0.294	0.192	0.667	0.313
Alcohawk Digital Alcohol Detector	0.263	1.000	0.269	0.556	0.313
Epson All-In-One Printer	0.263	0.412	1.000	0.778	0.438
Apple USB Mouse	0.316	0.294	0.269	1.000	0.188
Black & Decker Can Opener	0.263	0.294	0.269	0.333	1.000

Appendix B: 30 Most Common Products in Top Training Sets

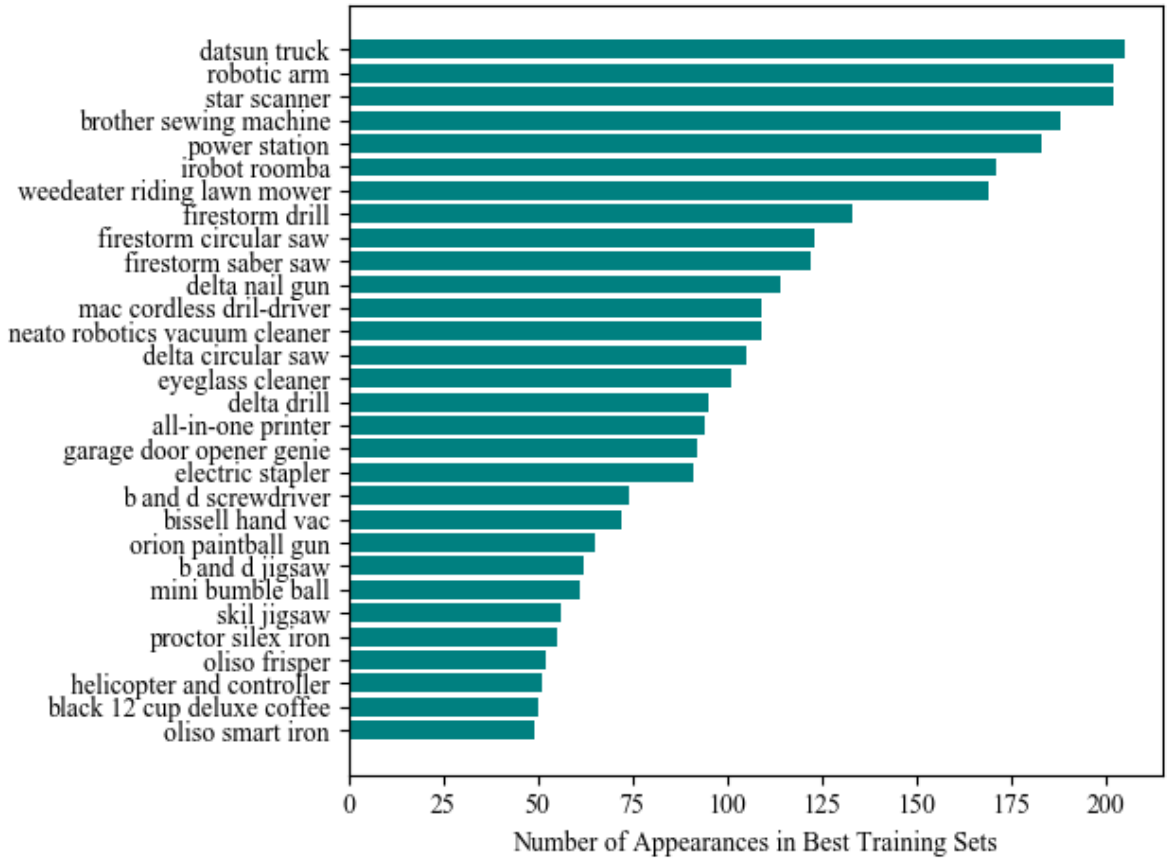

FIGURE 11: FREQUENCY OF PRODUCTS IN THE BEST TRAINING SETS