An Application of Clustering and Cluster Update Methods to Boiler Sensor Prediction and Case-Based-Reasoning to Boiler Repair

Timothy Edward Rooney
University of Wisconsin-Milwaukee

AN APPLICATION OF CLUSTERING AND CLUSTER UPDATE METHODS TO BOILER SENSOR PREDICTION AND CASE-BASED-REASONING TO BOILER REPAIR

by

Timothy Rooney

A Thesis Submitted in

Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in Engineering

at

The University of Wisconsin–Milwaukee

December 2019

ABSTRACT

AN APPLICATION OF CLUSTERING AND CLUSTER UPDATE METHODS TO BOILER SENSOR PREDICTION AND CASE-BASED-REASONING TO BOILER REPAIR

by

Timothy Rooney

The University of Wisconsin–Milwaukee, 2019
Under the Supervision of Professor Amol Mali

Driven by demand from both consumers and manufacturers alike, Internet of Things (IoT) capabilities are being built into more products. Consumers want more control and access to their devices, while manufacturers can find data gathered from IoT-capable products invaluable. In this thesis, we use data from a growing fleet of IoT-connected boilers in the residential, light-commercial, and medium-commercial ranges to demonstrate a framework for cluster initialization and updating. We compare two methods of dynamically updating clusters: a sequential method inspired by sequential K-means clustering and a cohesion-based method called DYNC. A predictive artificial neural network system demonstrates the effectiveness of the clustering methods.

In a secondary topic, a multi-tiered case-based reasoning system (CBR) is created based on boiler problem and repair support cases. Word embeddings are extracted from case comments and used to predict potential solutions to problems and problem categories using user selection and input. The primary tier uses information about actions taken involving specific parts, along with comments fed through the word embedding model, to predict the correct next step. The secondary tier uses only case comments to provide categories of likely symptoms and solutions. The third tier is a pure probability fall-back model.

To my wife.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CBR	Case-Based Reasoning
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DHW	Domestic Hot Water
EMA	Exponential Moving Average
IP	Interaction Power
MAE	Mean Average Error
MLE	Maximum-Likelihood Estimation
MLP	Multi-Layer Perceptron
MSRE	Mean Squared Ranking Error
NLP	Natural Language Processing
NN	Neural Network
OTR	Outdoor Temperature Reset
RMSE	Root Mean Square Error
SVM	Support Vector Machine
SVC	Support Vector Classifier
UW	Unit Weight

ACKNOWLEDGEMENTS

I would like to sincerely thank Professor Amol Mali for advising me throughout the completion of this work. His guidance and motivation were essential and very much appreciated.

I also owe Aron Mebrahtu at Lochinvar and Yaojun Wang at A.O. Smith a great deal of thanks for helping me to understand the format, parameter meaning, and encoding methods of the connected boiler data.

I INTRODUCTION

This thesis is split into five main sections. In 1. Introduction, we provide a brief background on boiler operation and literature in the areas of clustering and case-based reasoning and its applications to product repair. In 2. Boiler Clustering Method, we fully introduce the boiler data model, the motivation behind clustering, and the implementation of a clustering and updating framework that is graded in effectiveness by a neural network predictive model. In 3. Case-Based Reasoning, we introduce the case comments data surrounding boiler repair and describe the method behind the multi-tiered CBR approach. In IV. Results and Discussion present the results of the various clustering and update methods and accuracy metrics of the CBR system. Finally, V. Conclusion points key findings from this thesis and gives directions for future work.

1 Boilers

There are three main categories of boilers: water tube, fire tube, and electric [1]. In a basic water tube boiler, hot exhaust gas from natural gas, propane, or another combustible source surrounds the area a tube or tubes or water. Figure 1 shows a simple example of this. Heat from this hot gas is transferred to the water, especially on the lower part of the tube(s), through a combination of conductive and convective heat transfer. As the water near the bottom of the tube(s) increases in temperature, it becomes less dense a steam bubbles are created. Cooler, more dense water travels down the downcomers the hotter, less dense steam and water mixture rises on the other side. It is this circulation that keeps the boiler operating as steam leaves through a screen at the top of the drum and feedwater enters the drum to be eventually sent through the water tube.

Fire tube boilers operate by sending hot combustion gases through tubes that are surrounded by water. As the inverse of water tube boilers, heat transfers from the hot gas inside of the tube

FIG. C1.2-2 STEAM DRUM WITH TUBES

(Courtesy of Combustion Engineering, Inc.)

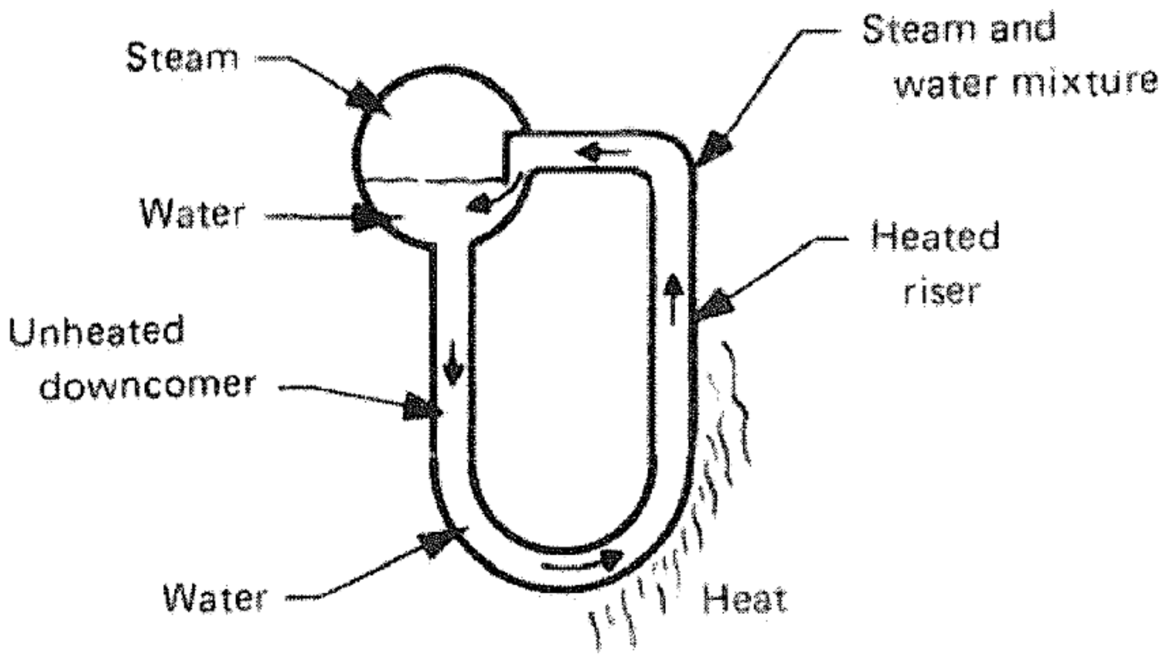


Figure 1: Simple Water Tube Boiler Diagram [1]

to the cooler water outside of the tube. Bubbles of steam break off of the tubes and rise to the surface of the water. Water flows around from the outer walls of the chamber to replace the volume occupied by the bubbles, and thus natural convection occurs. Relative to water tube boilers, fire tube boilers are typically cheaper to manufacture, have a larger footprint, and take longer to reach a useful state. Once they are fully started, they are better able to respond to changing demand than water tube boilers. They also have bigger turndown ratios, which is the ratio between its maximum allowed input and lowest allowed input.

None of the boilers in the connected boiler system and ticket datasets contain them, but they are mentioned here for completeness. In summary, there are two main types of electric boilers. The simpler, less common type uses electric resistance elements to boil the water, much like an electric water heater but with a higher setpoint. The more common type conducts electricity directly through the water using electrodes with voltages of up to 16,000V [1].

Outside of heating methods, another distinction between types of boilers is whether they are non-condensing or condensing. When the burner of a non-condensing water tube or fire tube boiler is on, all of the combustion products are released from the system through the exhaust in the form of gas. One component of these exhaust gases is water in the form of water vapor. There is latent heat stored in this water vapor that can be extracted by condensing it back into its liquid form. By using additional tubes between the burner and exhaust flue, condensing boilers extract this latent heat, discharge the resulting liquid water, and transfer this heat to the drum using a secondary heat exchanger.

2 Literature Review

2.1 Clustering

There are two main categories of clustering algorithms: *partitioning* and *hierarchical*. In their most basic forms, partitioning methods are based on the concept of first assigning the set of points to be clustered arbitrarily into a pre-determined number of clusters and then iteratively moving points between clusters to either maximize some measure of accuracy or minimize some measure of error [2]. They generally have an advantage over hierarchical clustering algorithms in time and resource usage. However, they are best applied to situations where the clusters are *hyperspherical* [3]. This means that the data falls into a shape approximating a sphere (or hypersphere) across its dimensions. A 2D example would be data in an approximately circular shape around a point, and a 3D example would be data arranged approximately spherically around a point.

Hierarchical methods, instead, are based on the concept of building-up or dividing-down to create a hierarchy of clusters. In agglomerative clustering methods, all data points begin as members of their own cluster. Two clusters are combined by some metric. Next, another two clusters - that may or may not contain the newly-formed cluster from before - are joined by the same metric. This process continues until all clusters are members of one large cluster with a hierarchy defined by the used metric. Divisive clustering works in the opposite direction. All points start as a member of a single cluster, and some metric is used to recursively divide the cluster until all points are a member of their own cluster.

A common and popular method of partitioning clustering is known as K-means clustering [4]. It is shown in Algorithm 1.

If n is the number of points in M, members of M are in $\mathbf{R}^d$, and k is the number of clusters in K, then the K-means algorithm has a runtime complexity of $O(n^{dk+1})$ [5]. K-means struggles in

```
Input: $M$: set of points
Input: $K$: arbitrarily-initialized cluster locations
1 foreach point $m \in M$ do
2     Find the closest cluster, $K_1(m)$ to $m$
3     Assign point $m$ to cluster $K_1(m)$
4 end
5 for each cluster $k$ in $K$ do
6     Update the cluster center of $K$ to be the average of the points assigned to it
7 end
8 Go back to step 1 until the cluster members no longer change.
```

Algorithm 1: K-means algorithm

areas where the true clusters vary greatly in size. Modified versions have been made to address this issue at the expense of complexity exist [3, 6], but such algorithms are still ideally applied to hyperspherical data.

Literally named, Dynamic Clustering (DC) is a family of algorithms based on the concept of gravity [7]. In it, points are initialized in space according to their assigned values, where, as this is an agglomerative clustering algorithm, each represents their own, one-member cluster. Through each timestep (ΔT) of predefined length, the accelerations and velocities of each point are determined through a gravitational model based on Newton's law of gravity (Equation 1, where m is the mass of a cluster, x is the location of a cluster, G is a pre-determined gravitational constant, and F is the simulated force applied to a cluster.). When the Euclidean distance between two points is lower than a predefined threshold (Δx), the points are merged into a cluster and then considered to be one point. The process continues until In one member of the DC family, called interaction power (IP), gravitational pull is heavier from clusters with more points. In the alternative, called unit weight (UW), the gravitational pull from each cluster is the same, regardless of the number of points it contains.

$$F_{ij} = G\frac{m_i m_j (x_j - x_i)}{||x_j - x_i||^3}, \tag{1}$$

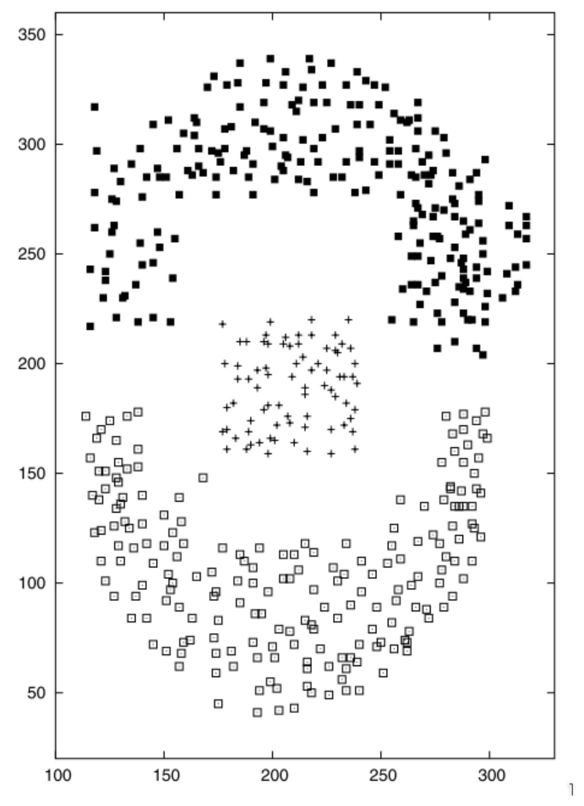


Figure 2: DC-UW Clustering Results

These algorithms (DC-IP and DC-UW) perform strongly in situations where clusters surround other clusters [7]. For example, it successfully clusters the three apparent groups of points in Figure 2, compared to the standard Ward's agglomerative clustering method [8], which mis-classified 29% of the points. However, while it is strong in successfully clustering groups of abnormal shapes, there is a large computational cost. This is effectively an application of the N-body problem, which is calculated in $O(n)$ for 2D problems, $O(n^2)$ for 3D problems, and $O(n^{m-1})$ for m-dimensional problems [9].

An alternative clustering method is known as *density-based* clustering algorithms. A popular algorithm is this family is density-based spatial clustering of applications with noise (DBSCAN) [10]. Where n is the number of points in $\mathbf{R}^d$ in a database, the runtime complexity of DBSCAN is $\Theta(n^2 d)$ [11]. Algorithm 2 shows its standard sequential implementation.

```
Input: $DB$: Database
Input: $epsilon$: Radius
Input: $minPts$: Density threshold
Input: $dist$: Distance function
Data: $label$: Point labels, initially $undefined$
foreach $point\ p \in DB$ do
    if $label(p) \neq undefined$ then continue;
    Neighbors $N \leftarrow$ RangeQuery$(DB, dist, p, \epsilon)$
    if $|N| < minPts$ then
        label$(p) \leftarrow$ Noise
        continue
    end
    $c \leftarrow$ next cluster label
    label$(p) \leftarrow c$
    Seed set $S \leftarrow N \setminus p$
    foreach $q \in S$ do
        if $label(q) = Noise$ then label$(q) \leftarrow c$;
        if $label(q) \neq undefined$ then continue;
        $N \leftarrow$ RangeQuery$(DB, dist, q, \epsilon)$
        label$(q) \leftarrow c$
        if $|N| < minPts$ then continue;
        $S \leftarrow S \cup N$
    end
end
```

Algorithm 2: DBSCAN algorithm [11]

It has two hyperparameters: ϵ defines the maximum radius that a point can exist within from another point to be considered one of its neighbors, while $minPts$ defines the minimum number of neighbors that a point must have to be considered a *core point*. A distance function, $dist$, is used to determine the distance between two points. An abstract function, RANGEQUERY, takes a selected point, p, compares its distances as calculated by $dist$ between all of points in the database, and returns those points that have distances less than ϵ as neighbors. Any point that has at least as many neighbors as $minPts$ is labeled as a core point, while those that do not are labeled as noise. When a core point is found, its neighbors are recursively expanded and labeled as members of the same cluster as part of a graph. In future passes of the algorithm, all points that have already been assigned to a cluster are ignored. In the end, all points either belong to a cluster or are labeled as noise.

A version of DBSCAN, called ρ-Double-Approximate DBSCAN, has been developed with the goal of creating a clustering algorithm that maintains clusters when a dataset has points that can be removed, added, or updated [12]. It makes two changes, both of which are based on a new clustering-precision hyperparameter, ρ. If a point p has at least $minPts$ neighbors with distances less than ϵ, then it is definitely a core point. If has less than $minPts$ neighbors with distances less than $(1+\rho)\epsilon$, then it is not a core point. Otherwise, it can be considered as either. Secondly, it creates a d-dimensional grid with side-length $\frac{\epsilon}{\sqrt{d}}$ over the data space $\mathbf{R}^d$. It uses the concept of cell denseness, where cells with at least $minPts$ are considered dense, to minimize runtime complexity. Because of the side-length definition, all points in a dense cell are known to be core points without using distance calculations. Graph edges are determined between cells, not points, further reducing runtime complexity. Thus, when points removed, added, or moved, the cost of iteratively updating all affected points in the graph is minimized. Figure 3 demonstrates this concept.

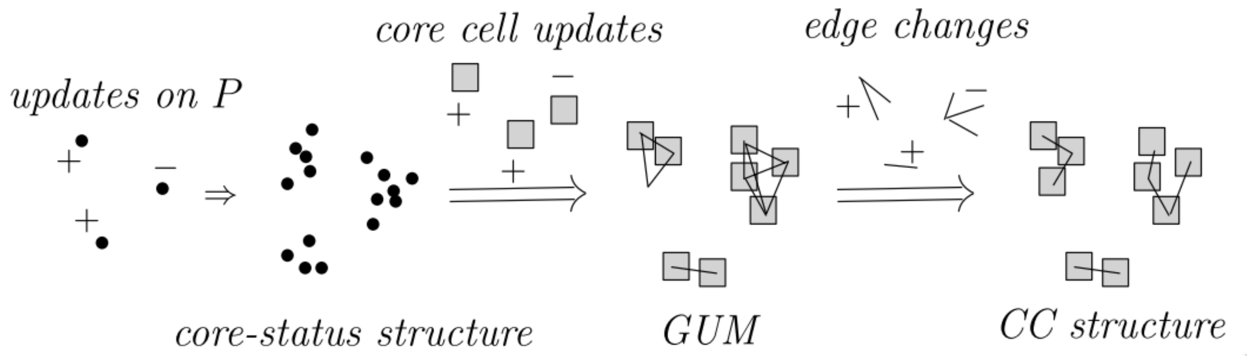


Figure 3: ρ-Double Approximate DBSCAN cell updating mechanism [12]

2.2 Case-Based Reasoning

At its core, Case-Based Reasoning (CBR) is the using known solutions to previous problems to identify or create a solution to a new problem [13]. In this defining work, CBR is defined to be based off of the way that expert humans integrate old experiences to solve new problems. Experts:

- depend on previous experiences when solving new problems.
- evaluate new problems in relation to these previous experiences.
  - recall old experiences that are related to a new problem.
  - interpret a new problem in relation to previous experiences.
- synthesize new solutions through adaptation.
- evaluate the effectiveness of new solutions.
  - repair their knowledge base after determining a new solution's effectiveness.

The interaction between these items are modeled in Figure 4 in a process known as the *Case-Based Reasoning cycle* [13]. Alternative definitions have been proposed [14, 15], but they all in some way define the steps of *retrieving* similar cases, *reusing* knowledge from similar cases to create a new solution, *revising* the new solution when needed, and *retaining* and evaluating the new solution and its results. It is important to keep in mind that CBR is an abstract methodology and framework for problem solving, not as discrete algorithm, and thus the technology used in each of these main steps can vary significantly. The retrievals steps can range from simple methods such as SQL queries and hard rules to complex statistical techniques and neural networks while the revision steps can likewise vary considerably in complexity [14].

CBR has been successfully adapted to many physical and language problem domains. In the physical domain, it has been used to identify sensor readings that indicate potential failures in an oil

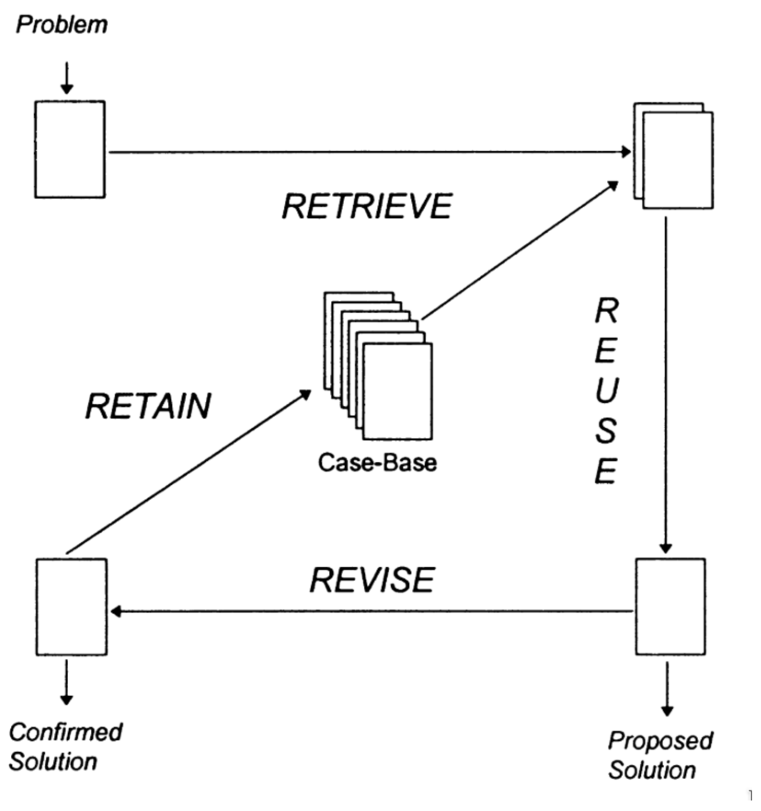


Figure 4: The CBR Cycle [15]

drilling application [16], office object placement [17], and fall detection using in-home IoT devices [18]. In the language domain, it has been used for sentiment analysis and recommendation [19, 20, 21], with artificial neural networks (ANNs) for internet domain name value and price estimation [22], and automated story generation [23]. Additionally, the topics of CBR and clustering have been used together, where hierarchical clustering algorithms can be used to improve information retrieval for CBR [24]

There are three main categories of CBR:

Substitution/Null

These seek to find an exact case match if possible, otherwise they find the closest previously-used solution [25] using a nearest-neighbor lookup. Then, try the solution and rely on the executor/human to make any adaptations as needed. This is useful in cases problem x is always solved by solution y and most problems are stored in the case-base. If a new problem is observed, the executor simply fixes the (probably) incorrect proposed solution and the next time the problem is seen the previous solution is re-applied. A classic example is IT help, where the vast majority of problems have an already-known, singular solution.

Transformation

These use previously-used solutions to generate the best guess at a new solution [25]. They are applicable to scenarios where there are broad categories of solutions but each case has unique details in a combination that may not have been observed before. They can be used in mechanical maintenance and repair [26, 27], medicine, and other applications where the the probability of solution$_y$ given problem$_x$ is not always 1.

Generative

Generative CBR systems create completely new multi-step solutions for given cases [25]. These are very academic - they are often mentioned in literature as an option but then a transformation or substitution system is used instead because of their practicality.

II BOILER CLUSTERING METHOD

1 Problem

1.1 Boilers

There are several boiler models connected to a cloud-based data collection system. These are:

- Crest
  - Single burner
  - Double burner
- Power Fin
- FTXL
- Knight
- Shield

These are described in detail below.

Crest boilers are a series of fire tube condensing boilers targeted at the commercial segment. They have input ratings between 750kBTU/hour and 6MBTU/hour, and their condensing nature helps them to get a thermal efficiency of up to 99%. Common installation locations include schools, hospitals, and hotels. They can be used to supply hydronic heating for a building. In addition, as shown in Figure 5, they can also be used to provide domestic hot water (DHW) for a building through the use of a hot water generator. Such hot water generators transfer heat from the non-potable boiled water to potable supply water through a heat exchanger. Typically, these heat exchangers use either coil-type or shell-and-tube heat exchangers [28, 29].

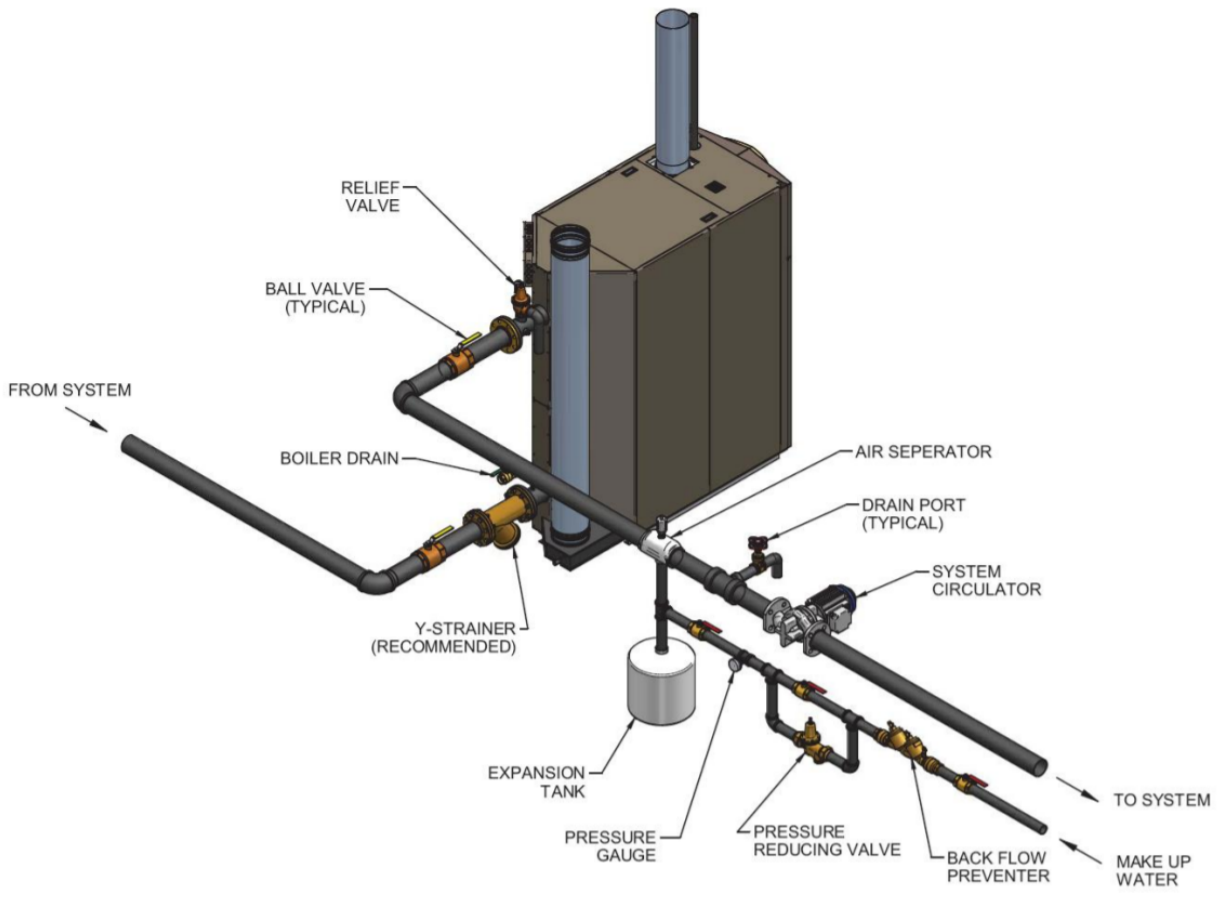


Figure 5: Crest Piping Diagram with Water Generator [29]

Table 1: Summary of Boiler Models

Series	Segment	Family	Models	Input Rating	Max Eff.	Turndown
Crest	Com	Fire Tube	12	750-6000kBTU	99%	25:1
Power Fin	Com	Water Tube	7	500-2000kBTU	85%	5:1
FTXL	Com and Res	Fire Tube	5	400-850kBTU	98%	10:1
Knight	Com	Fire Tube	5	80-285kBTU	99%	5:1
Shield	Com	Water Heater	8	125-500kBTU	96%	5:1

The Power Fin series of boilers are water tube non-condensing boilers that, like the Crest line, are targeted at commercial users. They range from inputs of 500kBTU/hour to 5MBTU/hour. Being non-condensing, they have a smaller footprint than Crest boilers with similar input ratings. In exchange, their thermal efficiencies are up to 87%. As a commercial boiler, major applications include hydronic heating and providing DHW through a hot water generator.

The FTXL series is made up of fire tube condensing boilers for residential and light commercial uses. With input ratings between 400kBTU and 850kBTU, they are used for hydronic heating and DHW supply in apartments, houses, and light-duty commercial locations like offices. As they are condensing, like the Crest, they achieve thermal efficiencies of up to 98%.

Smaller still, the Knight boilers have water tube input ratings between 80kBTU and 285kBTU. Because of their lower input ratings, they are typically used in residential settings for hydronic heating and indirect DHW supply through the use of water generators.

Although they are commercial water heaters, the Shield units are still part of the CON·X·US system. They have maximum input ratings between 125kBTU and 500kBTU. Like the boilers, they have modulating burners. As water heaters, their use is limited to DHW applications and their installation is simpler.

A summary of these model specs is shown in Table 1. Figure 6 shows a comparison of the available input ranges for models of each of the main boiler types described above..

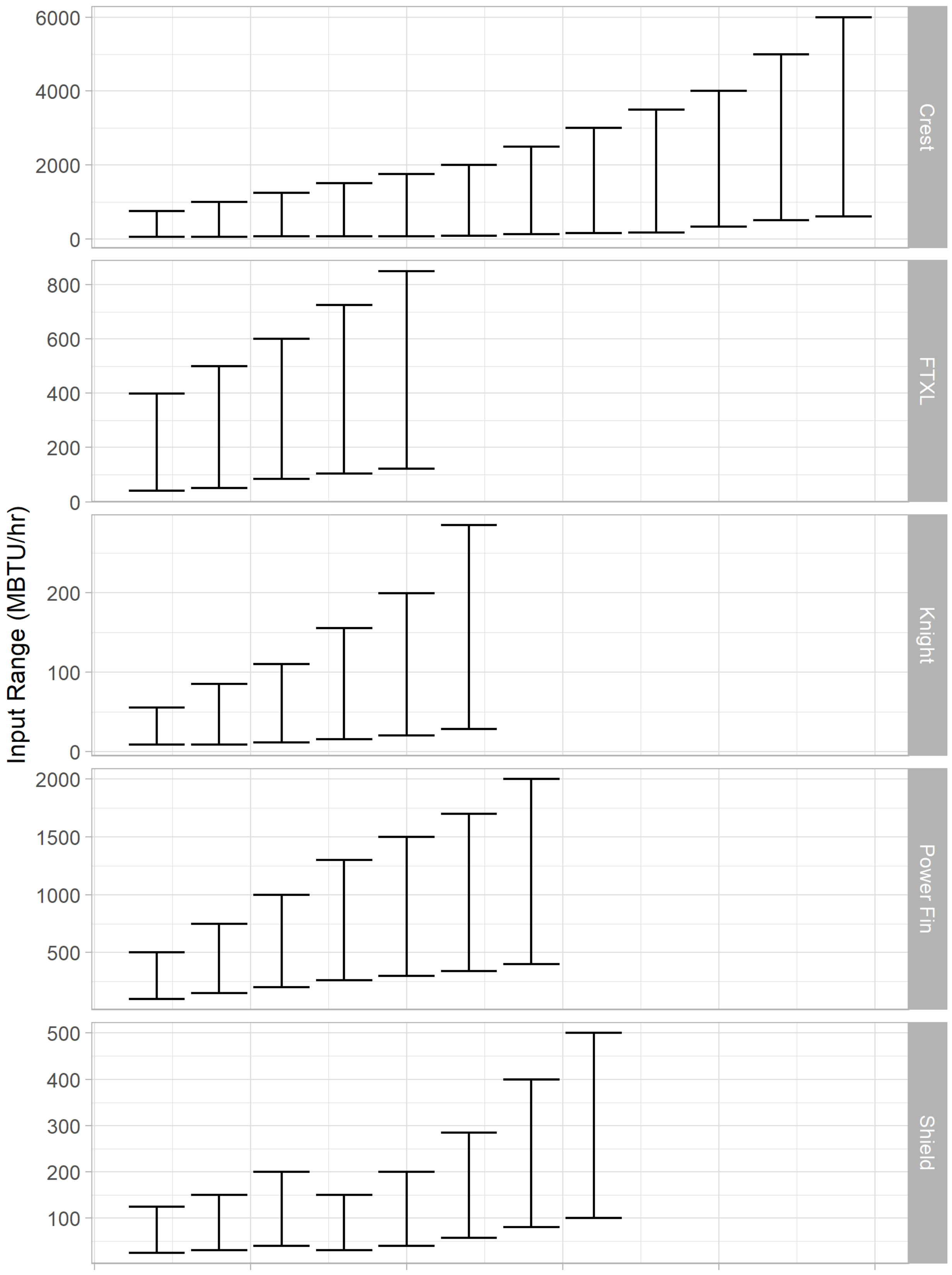


Figure 6: Boiler Input Ranges

Table 2: Simplified format of raw data from boilers

Item	Description
`oem_model`	The board model – either Herald or Page
`dsn`	The boiler serial number
`property_name`	The name of the property
`base_type`	The type of the property
`created_at`	The time that the property is uploaded by the boiler
`discarded`	Whether a property recording is considered discarded
`val_int`	An integer value
`val_decimal`	A decimal value
`val_float`	A floating point value
`val_boolean`	A boolean value
`val_string`	A string value

1.2 Data

Each day, the connected boilers send one of 271 possible parameters at irregular intervals for a total of around 500MB of data per day. Some of these parameters may be sent every few seconds, while others might only be sent once per week. These parameters vary from performance and sensor data - such as fan speed, outlet temperature, and flue temperature, to user-configurable parameters like outdoor reset temperatures. These parameters are received at uneven time intervals in a key-value paradigm. A simplified summary of the format of the data as it arrives is shown in Table 2. This is a subset of the full data format shown in Table 12 in APPENDIX A.

Of the 271 possible parameters, 239 of them can be handled numerically. Integers, decimals, and floating point numbers are easily handled numerically, and boolean values can be changed from TRUE/FALSE to 1 and 0. The remaining string values are made up of boiler model names, contact information, and location information. The model names are matched up with one of the 5 model families described before (the distribution of these is shown in Figure 7), and the location information is used to find local weather data for each boiler.

The modeling approach is split into two main sections. First, we develop clusters of boilers based on the full set of numeric data, their model families, and their weather data. Second,

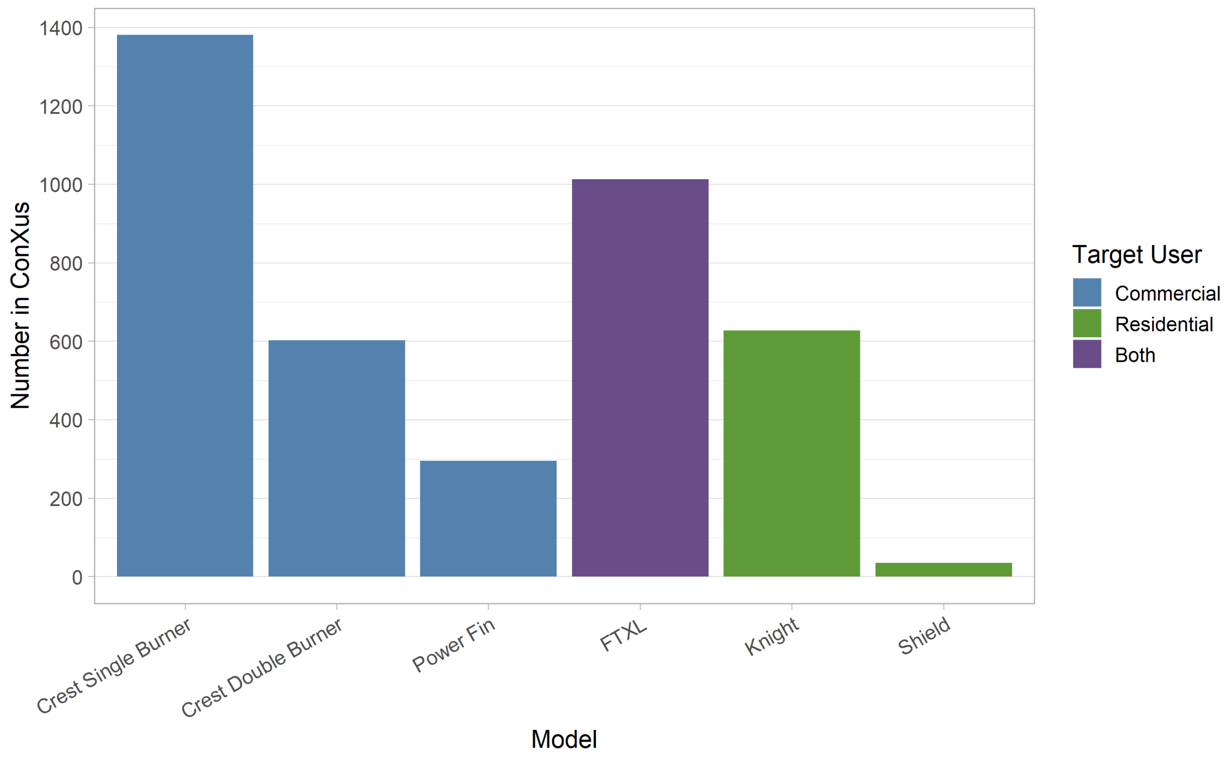


Figure 7: Distribution of unit model types

we use these clusters to inform models that predict future performance parameters based on previously-seen data. While the generated clusters could be used for market research or learning behavior patterns in the way that installers and users configure their boilers, here we demonstrate the usefulness of various clustering methods through the accuracy of their associated predictive models. The assumption is that models using well-clustered groups of boilers will be more accurate at predicting than those with poorly-clustered groups or without clusters entirely.

2 Implementation

One of the pieces of location information is the boiler IP address. In order to get the approximate location of a boiler, in terms of latitude and longitude, we use an API provided by ipstack [30] (https://ipstack.com/). This API allows up to 10,000 IP addresses to be converted to latitude and longitude per month through knowledge of the ISP and their IP designation patterns. With this information, the location of each boiler is indicated by a dot in Figure 8.

Next, we need a way to access weather data - specifically the minimum and maximum temperature of each day - for each boiler. For this, a small Python library was developed that interfaces with the NOAA's NCDC weather API [31] (https://www.ncdc.noaa.gov/cdo-web). Given a date, latitude, and longitude, it is able to collect the maximum and minimum temperature seen by the closest weather station.

One more limitation is needed prior to the clustering and analysis. Not all of the boilers in the system regularly send data for all parameters. To prevent building a system with artificially-inflated accuracy from predicting empty or filler values, we only allow boilers that have report unique values for at least 150 of the 239 numeric parameters. Over a testing period from April 01, 2018 to December 31, 2018, Figure 9 shows the number of these allowed boilers against the total number of boilers in the system.

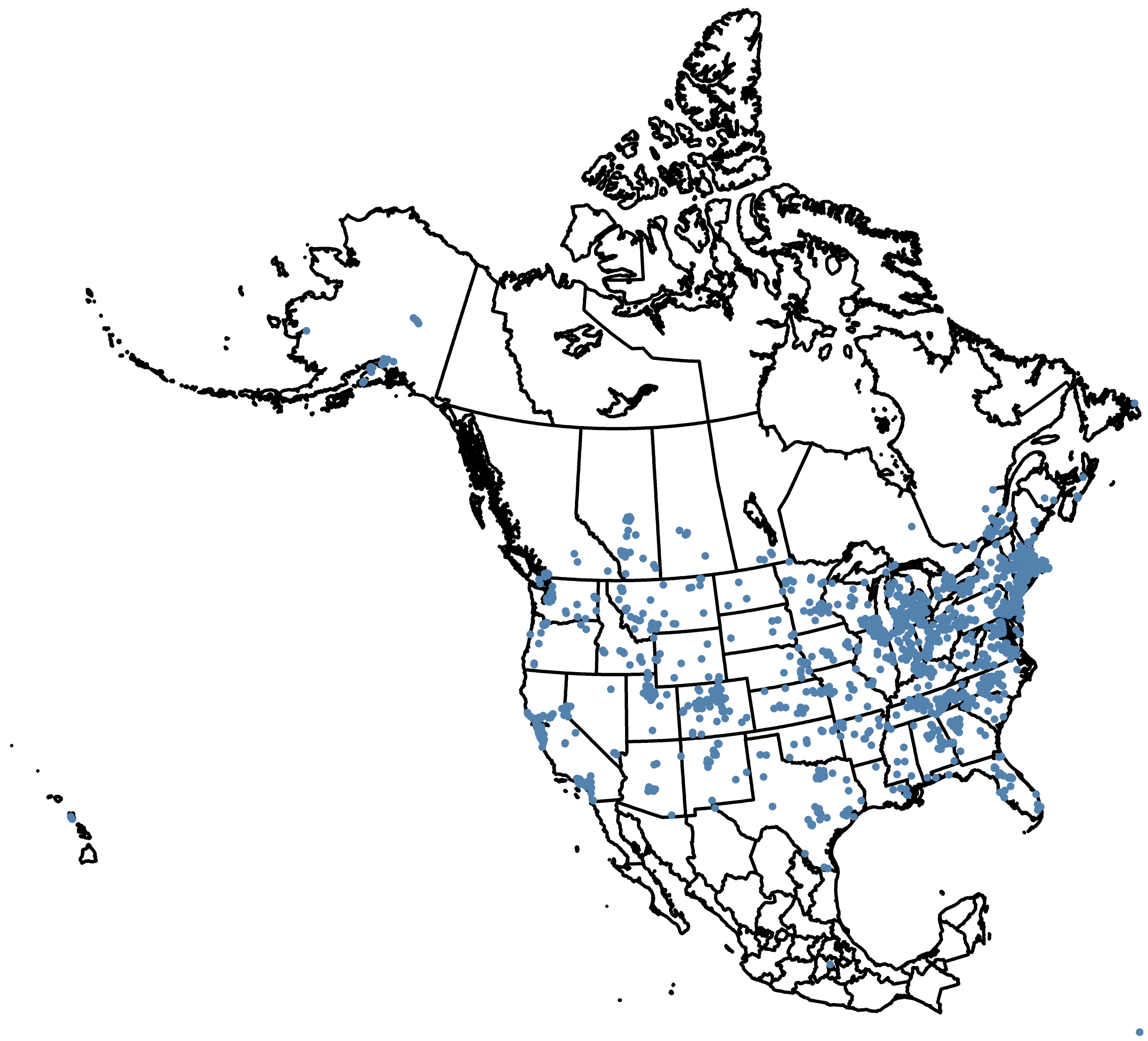


Figure 8: Map of unit locations in North America

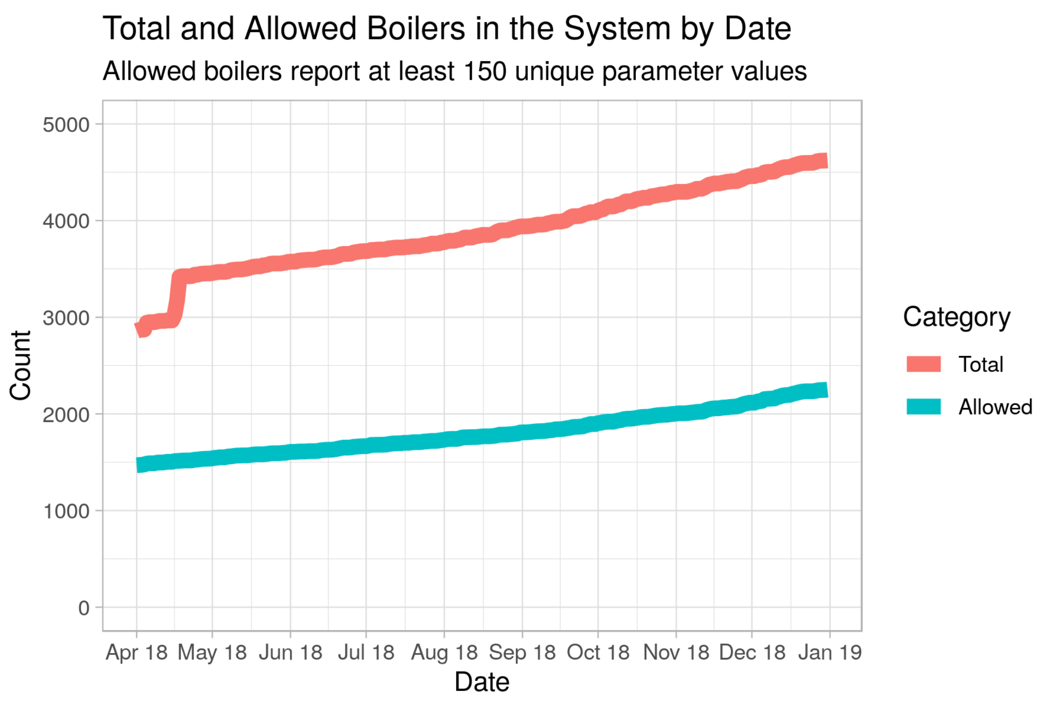


Figure 9: Total and Allowed Boilers in the System by Date

2.1 Time Series Analysis

Different properties can occur at very different frequencies. For example, sensor readings like flue temperatures, fan speeds, and outlet temperatures are sent very frequently, while other properties like outdoor reset temperatures are only sent when they are updated by a user or the building management system (BMS). Because the dataset is very large, a way to simplify it into per-day statistics is useful.

Andreas Eckner has developed a framework for the statistical analysis of unevenly-spaced time series [32]. The time-varying exponential moving average (EMA) method developed by him is shown in Equation 2. A psuedocode implementation of this is shown in Algorithm 3, which generates an exponentially-weighted moving average for each recorded value. We use this to assist with the clustering method. For each of the boilers with enough unique values to be considered and for each day, we calculate the 7-day EMA of the frequently-reported sensor values using Algorithm

3. We take the average of this 7-day EMA as the single data point considered for clustering for that parameter for each boiler. For the slower, non-sensor parameters, we simply consider the most-recently seen value. These parameters will only be entered into the system when changed, so even if the last-seen value is from months ago, it is still valid.

$$\text{EMA}(X,\tau)_t = \begin{cases} X_{t_1} & t = t_1 \\ \left(1 - e^{\frac{-\Delta t_n}{\tau}}\right) X_{t_{n-1}} + e^{\frac{-\Delta t_n}{\tau}} \text{EMA}(X,\tau)_{t_{n-1}} & t = t_n > t_1 \end{cases} \tag{2}$$

```
**Data:** $Time[]$ and $Value[]$ where a $Time$ element corresponds to the $Value$ element of the
    same index
**Output:** $EMA[]$
$EMA[0] \leftarrow Values[0]$;
**for** $i \leftarrow 1$ **to** $length(Time) - 1$ **do**
    $\alpha \leftarrow \exp\left(-(Time[i] - Time[i-1])/\tau\right)$;
    $EMA[i] \leftarrow (1-\alpha) * Value[i-1] + \alpha * EMA[i-1]$;
**end**
```

Algorithm 3: Pseudocode implementation of Equation 2

As an example of this process, let's use example sensor data from the outlet temperatures of two boilers during May of 2018. The points in Figure 10 show actual measured values from the first example boiler. The pink line shows the 7-day EMA calculated with Algorithm 3, while the blue line shows the daily average of the 7-day EMA that is used for clustering. The same process is applied to the second boiler in Figure 11, which features a more period usage pattern.

2.2 Clustering

Agglomerative clustering is a clustering method that is part of the hierarchical clustering family, which is itself a part of the larger clustering algorithm family. Clustering algorithms can be divided into two main groups: hierarchical methods and partitioning methods. [2] Partitioning methods are used in situations where intra-cluster relationships are not important.

Most agglomerative clustering methods assume that the data is *stationary*. Such data is assumed

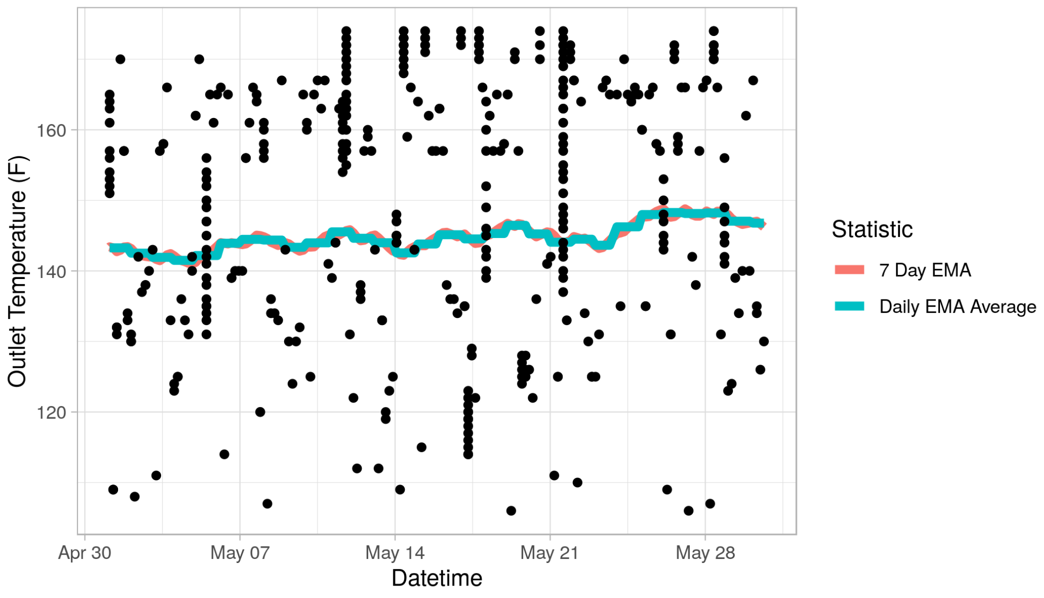


Figure 10: Averaging Method - Boiler Example 1

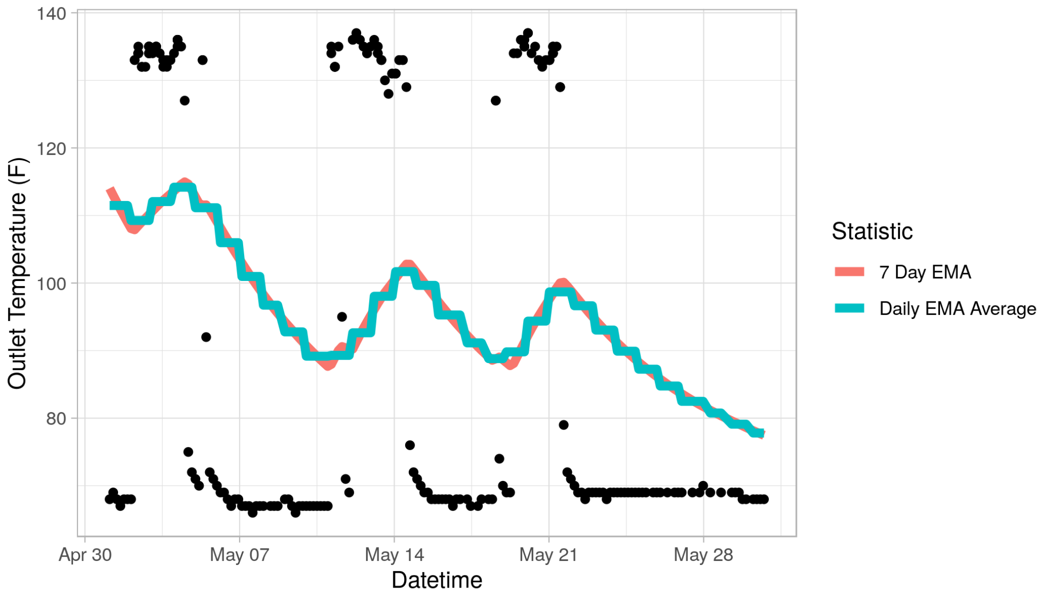


Figure 11: Averaging Method - Boiler Example 2

Table 3: Recommended ROC method kernels

Name	Function	Robust
Radial basis function (RBF)	$K(x,y) = e^{\frac{-\sum_i \lvert x_i - y_i \rvert^b}{\sigma^2}}$	Yes
Gaussian kernel (GK)	$K(x,y) = e^{\frac{-\sum_i \lvert x_i - y_i \rvert^2}{\sigma^2}}$	Yes
Polynomial kernel (PK)	$K(x,y) = \left(x^T y + 1\right)^d$	No

to have a mean, variance, and autocorrelation that does not change over time. [33] It does not show trends or seasonality. This can present a problem in the case of the boiler data because the boiler configuration parameters and sensor values certainly can exhibit trends and seasonality. A building management system (BMS) might change some of the configuration parameters throughout the year, and installers and contractors can make instantaneous step changes to configuration parameters.

Guedalia *et al.* proposed a on-line agglomerative clustering method for non-stationary data called AddC [34]. In the context of clustering methods, *on-line* refers to its ability to factor in temporal information. It does not refer to the traditional definition of on-line with respective to algorithms of a process that is intended to be continually updated over time.

Although AddC successfully manages to cluster non-stationary data, it is not robust to noise. In response, Zhang *et al.* proposed a new method, called Robust Online Clustering (ROC) [35]. Where AddC is based on an on-line k-means method, ROC is a kernel method. Zhang *et al.* propose replacing the standard agglomerative clustering distance function with a kernel-based function. This is defined in Equation 3, where $K(x, y)$ is the kernel function. Although the kernel function can be nearly anything, they suggest using one of the kernel functions shown in Table 3. If the Gaussian Kernel is used, Equation 3 can be reduced to Equation 4. Once a kernel is selected, the ROC algorithm is used as is shown in Algorithm 4

$$d(x,y) = \sqrt{K(x,x) - 2K(x,y) + K(y,y)} \tag{3}$$

$$d(x,y) = \sqrt{2 - 2K(x,y)} \tag{4}$$

```
   Input: A threshold $\epsilon$, an initial $N = 0$, and $N_{max}$
   Data: x[]
1  for $i \leftarrow 0$ to $length(x) - 1$ do
2      winner = $\min(d(x[i], y_j))$
3      Update the winner prototype $y_{winner}$ and its weight $c_{winner}$ where
           $c_{winner} = c_{winner} + K(x, y_{winner})$
           $y_{winner} = y_{winner} + \frac{x - y_{winner}}{c_{winner}}$
4      if $N < N_{max}$ then
5          $\delta = N$
6      else
7          Find the two closest prototypes, $\gamma$ and $\delta$, through $\min(d(y_\gamma, y_\delta))$ Merge these
            prototypes with
               $y_\gamma = \frac{y_\gamma c_\gamma + y_\delta c_\delta}{c_\gamma + c_\delta}$
               $c_\gamma = c_\gamma + c_\delta$
8      end
9      Create a new prototype $y_\gamma$ with $x[i]$ where $y_\gamma = x[i]$ and $c_\gamma = 0$
10 end
11 Remove all clusters with a negligible weight as defined by $c_j < \epsilon$ where $j$ is from 1 to $N$
```

Algorithm 4: ROC agglomerative clustering algorithm from Zhang *et al.*

As a point of comparison, we also use alternative agglomerative clustering algorithms, the partitioning K-means method described previously in Algorithm 1, and DBSCAN. For the agglomerative clustering algorithms, the general implementation is the same. Given a distance function, $d(\mathbf{x}, \mathbf{y})$, and a group of sets, or clusters, we use a *linkage criteria* to determine which clusters to connect next. To start, each point begins as a member of its own cluster. Then, at each step, the two closest clusters as determined by d and the linkage criteria are linked. This process continues until all of the points and their parent clusters are part of one single supercluster.

One example is the *centroid* agglomerative clustering method. If $C(A)$ is a function that returns the centroid of set A, then the linkage criteria that returns the next two clusters to join is $\text{argmin}_{A,B,A\neq B} d(C(A), C(B))$ for each combination of clusters A and B remaining. All members of B are added to A and the process recurses until only one cluster exists. More formally, an implementation of the centroid-finding function C is given by Equation 5, and thus d is given by Equation 6 [36].

$$C(A) = \frac{1}{|A|} \sum_{\mathbf{p} \in A} \mathbf{p} \tag{5}$$

$$d(A, B) = \frac{1}{|A||B|} \sum_{\mathbf{p} \in A} \sum_{\mathbf{q} \in B} \mathbf{p} \cdot \mathbf{q} \tag{6}$$

Ward's method operates similarly, but instead merges clusters in a way that minimizes the variance of the newly-created cluster. To accomplish this, the distance function in Equation 7 is used [37].

$$d(A, B) = \frac{|A||B|}{|A| + |B|} ||A - B||^2 \tag{7}$$

The recursive, linking nature of agglomerative clustering lends itself to the creation of *dendrograms*. These are visual representations of the process that an agglomerative clustering algorithm used. As an example, let us use part of a standard clustering benchmark dataset named "Unbalance" [38]. When clustered using Ward's method, it looks like Figure 12. The process of generating the clusters creates the dendrogram shown in Figure 13. The y-axis indicates the distance at which two clusters were joined, while the x-axis represents each individual point in the set of clusters. This dendrogram can also be used to demonstrate another feature of such agglomerative clustering

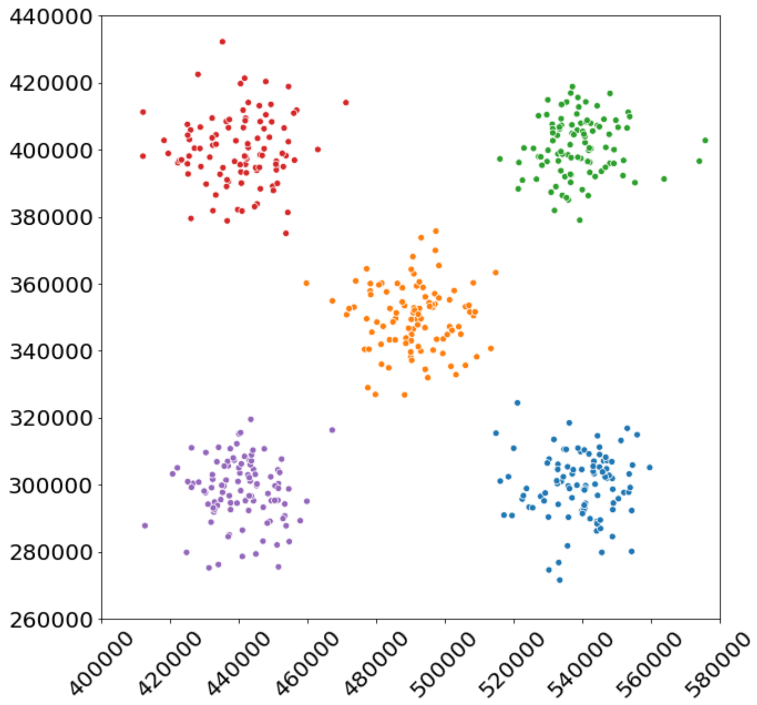


Figure 12: ”Unbalance” Clustered by Ward’s Method

algorithms. By their nature, agglomerative clustering algorithms do not have a defined number of output clusters. We can artificially generate these in one of two ways: first, we can define a cutoff distance below which clusters are considered to be independent. One can imagine horizontal line at y=200,000 on Figure 13 to demonstrate this. Alternatively, we can demand *n* independent clusters, which is the inverse of the first method. One can imagine a horizontal line starting at y=0. Its position increases along the y-axis until there are *n* independent clusters below it. In the case of Figure 13, this occurs at around y=120,000. By moving this line up and down, the number of clusters used can be changed while using the generated hierarchy.

The ”Unbalance” data example gives the opportunity to describe a second concept in the clustering. Referring back to Figure 9, there is a continuously-increasing number of boilers to be clustered. To handle introducing new boilers to clusters and moving old boilers between clusters,

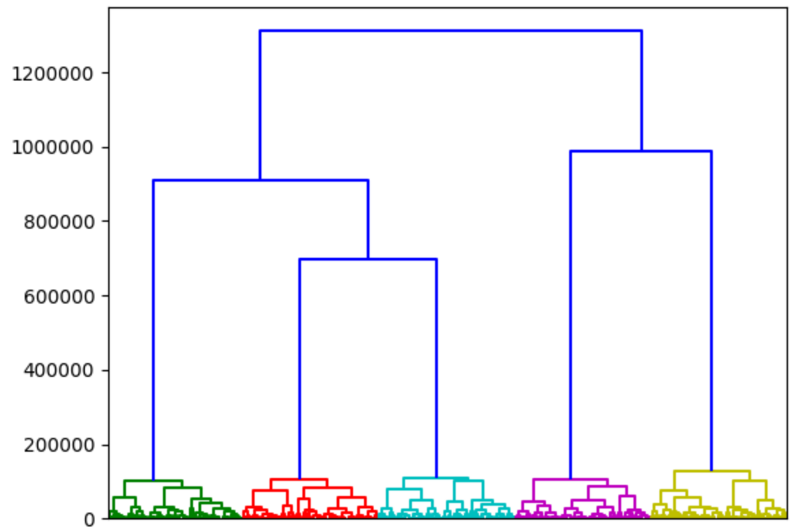


Figure 13: ”Unbalance” Dendrogram Clustered by Ward’s Method

we use two methods: a sequential method based off of sequential K-means, and a dynamic cohesion method named DYNC (or dynamic cohesion). We can also use a common framework, shown in Algorithm 5 to implement these two update mechanisms with the previously-mentioned clustering methods.

Φ represents one of the possible clustering methods described before. Θ and Δ take on one of two main flavors: the sequential method and DYNC. The sequential method borrows from the sequential K-means update procedure, shown in Algorithm 6 [39]. Algorithm 7 shows the modified process for Θ, while Δ is shown in Algorithm 8. For DYNC, we need a method for calculating the centroid of a cluster. Even if the original clustering method made use of kernels and non-Euclidean space, DYNC stays in Euclidean space. Therefore, we simply use centroid $= \frac{\sum c_i}{|\mathbf{c}|}$, where $\mathbf{c}$ is a cluster of points when the centroid is needed. The new point method Θ is given by Algorithm 9 and the update method Δ is given by Algorithm 10. As part of its cohesion-checking process, DYNC uses a cohesion limit ϵ. If a cohesion value μ for a cluster falls below this limit, the cluster is removed and old members re-assigned new clusters using Algorithm 9.

”Unbalance” can be used to demonstrate how these two methods work within the general

```
Data: $D_{initial}$, the initial dataset
Data: $D_{streaming}$, continuously-streaming data
Input: $\Phi(D_{initial})$, a cluster initialization function taking an initial dataset $D_{initial}$ as an input
        and returning a set of clusters, $C$, where such clusters contain a list of their elements
Input: $\Theta(p, \mathbf{C})$, a point addition function taking a new point, $p$, and a set of clusters, $\mathbf{C}$ and
        returning $\mathbf{C_{new}}$
Input: $\Delta(p, \mathbf{C})$, a point update function a point to be moved, a set of clusters, and returning
        $\mathbf{C_{new}}$
Initializating the clusters
$\mathbf{C} = \Phi(D_{initial})$
Update the clusters while new data is appearing
while $p \leftarrow D_{streaming}$ do
    if $p \in \mathbf{C}$ then
        $\mathbf{C} = \Delta(p, \mathbf{C})$
    else
        $\mathbf{C} = \Theta(p, \mathbf{C})$
    end
end
```

Algorithm 5: Clustering and Update Framework

```
Data: $\mathbf{C}$, a set of cluster centroids corresponding to K-means clusters
Data: $\mathbf{N}$, the number of items in each cluster
Data: $D_{streaming}$, continuously-streaming data
while $\mathbf{p} \leftarrow D_{streaming}$ do
    $\mathbf{c}_i$ = closest cluster in $\mathbf{C}$ to $\mathbf{p}$
    $N_i = N_i + 1$
    $\mathbf{c} = \frac{1}{N_i} \cdot (\mathbf{p} - \mathbf{c}_i)$
end
```

Algorithm 6: Sequential K-means Update Method

```
Input: $\mathbf{p}$, a new data point
Data: $\mathbf{C}$, the clusters in the dataset, containing member points, the number of members, and
       its current location
$\mathbf{c}, n, \ell \leftarrow$ the closest cluster in $\mathbf{C}$ to $\mathbf{p}$, its number of members, and its location vector
$n = n + 1$
$\ell = \frac{1}{n} \cdot (\mathbf{p} - \ell)$
Store the new $\mathbf{c}$, $n$, and $\ell$ in $\mathbf{C}$
```

Algorithm 7: Sequential Method - New Point

```
**Input:** p, a data point to be updated
**Data:** C, the clusters in the dataset, containing member points, the number of members, and
        its current location
$\mathbf{c}, n, \ell \leftarrow$ the closest cluster in C to p, its number of members, and its location vector
if p is already in c then
    $\ell = \frac{1}{n}(\mathbf{p} - \ell)$
else
    $\mathbf{c}_{old}, n_{old} \leftarrow$ the cluster that p is currently a member of and its number of members
    $n_{old} = n_{old} - 1$
    Remove p from $\mathbf{c}_{old}$ $n = n + 1$
    $\ell = \frac{1}{n} \cdot (\mathbf{p} - \ell)$
    Store c, $n$, and $\ell$ in C
end
```

Algorithm 8: Sequential Method - Updated Point

```
**Input:** p, a new data point
**Input:** $\epsilon$, a threshold cohesion value
**Data:** C, the clusters in the dataset, containing member points, number of members of each
        cluster, and their centroids
$\mathbf{c}, n, \mathbf{y} \leftarrow$ the closest cluster in C to p, its number of members, and its centroid
Add p to c
$n = n + 1$
$\mu = 1/\left(\frac{\sum ||c_i - \mathbf{y}||}{n}\right)$
if $\mu < \epsilon$ then
    Free the points in c and assign them to new remaining clusters in C following this new
      point method
end
```

Algorithm 9: DYNC - New Point

```
Input: p, a new data point
Input: $\epsilon$, a threshold cohesion value
Data: C, the clusters in the dataset, containing member points, number of members of each
      cluster, and their centroids
$\mathbf{c}, n, \mathbf{y} \leftarrow$ the closest cluster in C to p, its number of members, and its centroid
if p is not in c then
    $\mathbf{c}_{old}, n_{old}, \mathbf{y}_{old} \leftarrow$ the cluster that p is currently a member of, its number of members, and
     its centroid
    Remove p from $\mathbf{c}_{old}$
    $n_{old} = n_{old} - 1$
    $\mu_{old} = 1/\left(\frac{\sum ||c_{old,i} - \mathbf{y}_{old}||}{n_{old}}\right)$
    if $\mu_{old} < \epsilon$ then
        Free the points in $\mathbf{c}_{old}$ and assign them to new remaining clusters in C following the
         new point method
    end
    Add p to c
    $n = n + 1$
else
    Update p's location in c
end
$\mu_{new} = 1/\left(\frac{\sum ||c_i - \mathbf{y}||}{n}\right)$
if $\mu_{new} < \epsilon$ then
    Free the points in c and assign them to new remaining clusters in C following the new
     point method
end
```

Algorithm 10: DYNC - Updated Point

Table 4: Summary of Example Cluster Movements

Name	Description	Figure	Video Link
Rotation	Demonstrates the four outside clusters rotating about the center cluster	Figure 14	https://youtu.be/gvpAsV2d4Qg
Crossing	Demonstrates two cluster collisions	Figure 15	https://youtu.be/oQzQSHfqjAk
Splitting	Demonstrates splitting the center cluster	Figure 16	https://youtu.be/_Z-ndzRHcSo

framework of Algorithm 5. Starting with the points clustered as shown in Figure 12, we can use three examples of points moving. All three examples were initialized with Ward's method, but the dataset is simple enough that any non-specialized clustering algorithm should be effective in identifying the 5 initial groups. To create point movement in these examples, at each timestep, each point has a 25% chance of moving a semi-random distance along the direction indicated. While this is certainly very simplified compared to the potential for point and cluster movement in the boiler data, these examples illustrate a general intuition for the update methods. Table 4 summarizes these examples. In all of them, the cohesion threshold was $\epsilon = 60$.

In the **Rotation** example, there is not much difference seen between the two methods. When passing the center cluster, a few points flip between the outside and center cluster in the sequential example. In **Crossing**, we see large differences. With DYNC, the clusters crossing another cluster remain together, because at no point does their cohesion fall below the threshold. With the sequential method, the points travel through each other as before, but the clusters bounce off of each other. At the end, the points of the crossing clusters have switched the clusters that they are a part of. If one used some method after clustering that relied on historical information about a cluster, this could lead to problems. In the **Splitting** example, we show what happens when the center cluster splits apart and joins each of the outer clusters. The sequential method eventually assigns three of the four groups into their correct new cluster. The fourth split forms the basis of a new cluster that is touching the old cluster in the upper left. This is because the sequential method

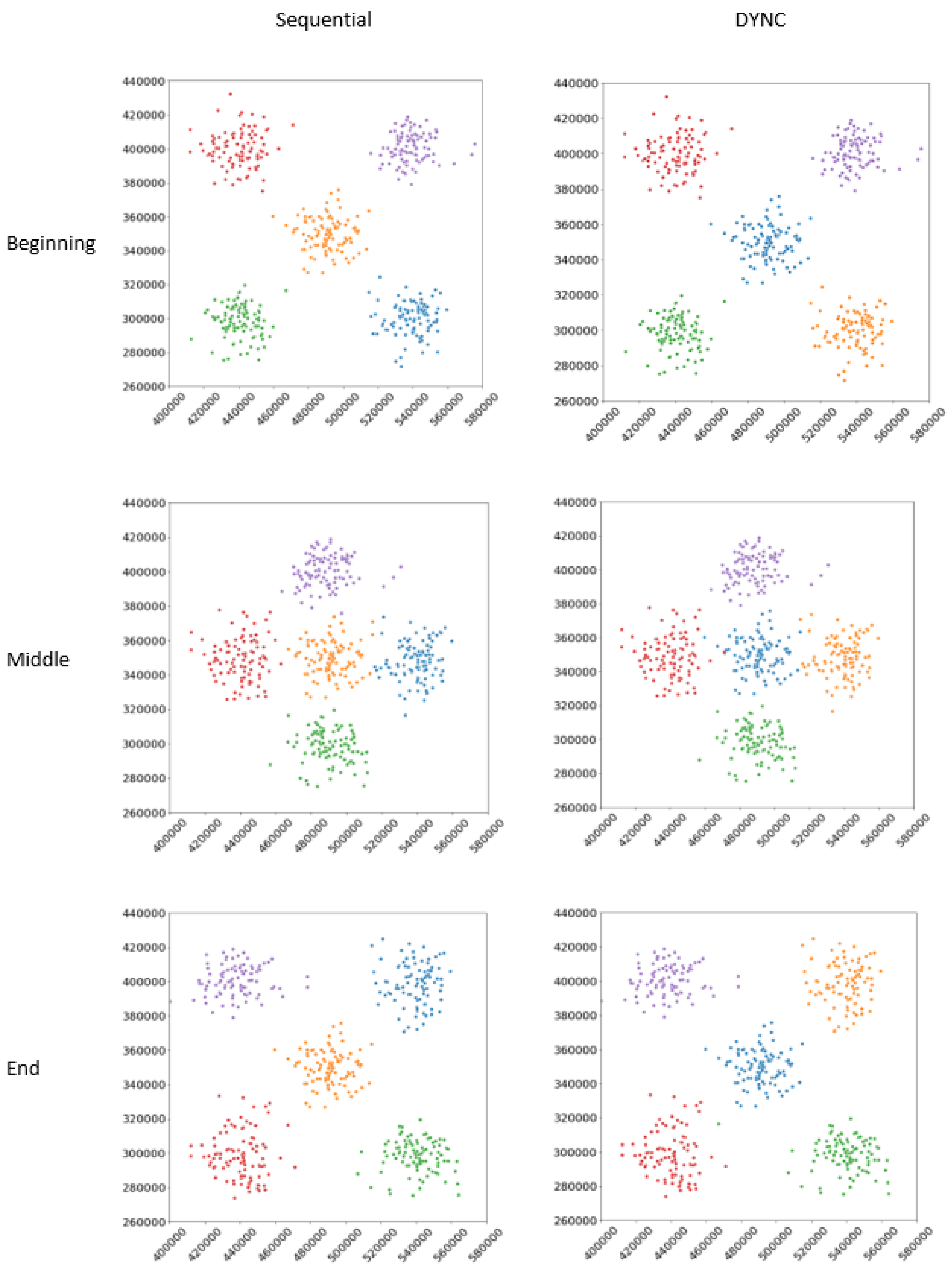


Figure 14: "Unbalance" Rotation Example for Sequential and DYNC Updating

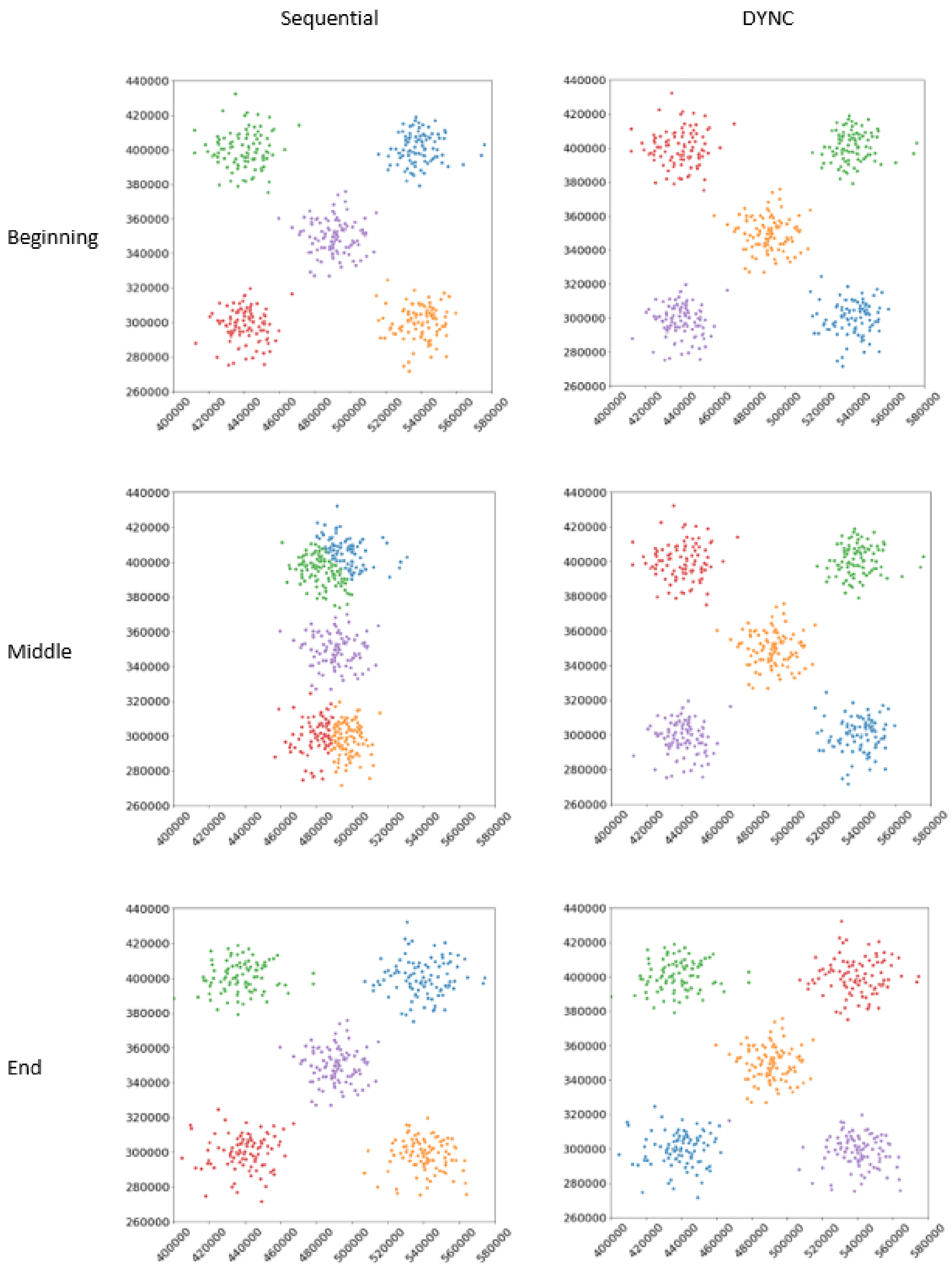


Figure 15: ”Unbalance” Crossing Example for Sequential and DYNC Updating

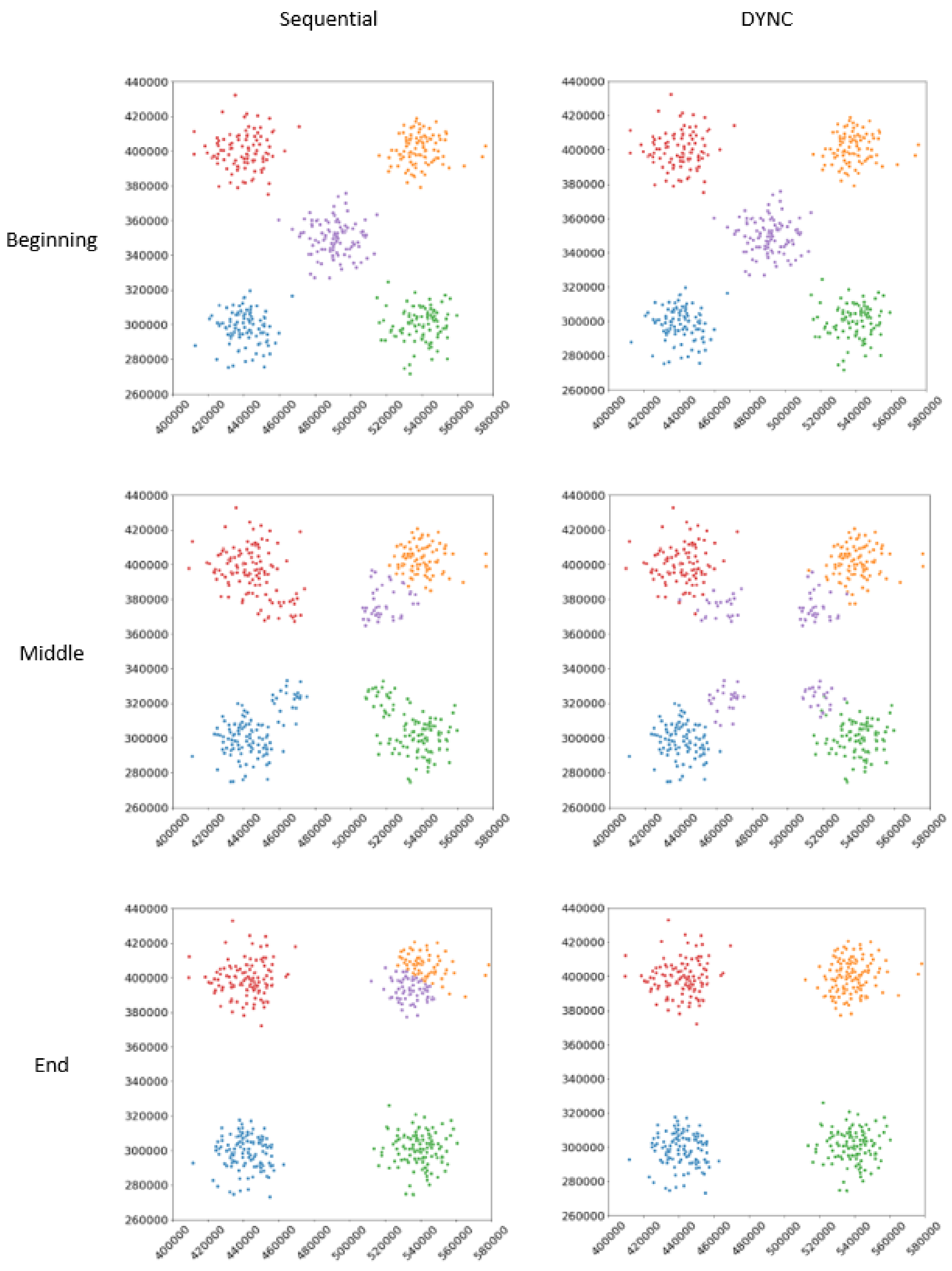


Figure 16: ”Unbalance” Splitting Example for Sequential and DYNC Updating

has no way to remove old clusters when they are not needed.

To handle clustering the boilers, we use the same methods as demonstrated on the ”Unbalance” dataset. The Kmeans, agglomerative clustering with Ward’s method and the centroid method, and DBSCAN use implementations provided by Scipy [40]. The inputs for each boiler for each day are the 239 numeric values, an indicator for which of the model families it belongs to, and the minimum and maximum outdoor temperatures seen in that day at that boiler from the NCDC weather data. The output is an integer that represents the cluster that the boiler belongs to on that day. All of the clustering algorithms are run from April 1st, 2018 to December 31st, 2018. A summary of the initialization methods, update methods, number of clusters, and other parameters are shown in Table 5. It also shows the amount of time that each method took to initialize and update for the 9-month period. There is also a modifications to the general Algorithm 9 and Algorithm 10 for the boiler clustering. The ϵ values are decreased by 0.01 each time a cluster is removed. This allows the system to settle into an (at least temporary) equilibrium. Without this, there were cases where the system was re-initializing several times per simulated day.

Comments on Centroid Clustering and DBSCAN for Boiler Data The centroid clustering method creates few independent clusters on the boiler data. In other words, when 16 clusters are requested, 15 clusters each with 1 boiler are returned along with a final cluster containing the remaining boilers. We can explore this by looking at Figure 17. There is a small group of individual boilers on the left-hand side, while the hierarchy of the remaining boilers is relatively flat. This shows that the centroid method has a hard time differentiating between the different boilers. Compare this to Figure 18. Ward’s method generates a clearly-defined hierarchy of boilers and clusters. Because the centroid method creates such unbalanced clusters, we do not apply the sequential and DYNC update methods to its results. DYNC immediately removes the large cluster of boilers because

Table 5: Initialization and Update Methods used for Boiler Parameter Clustering

Method, Φ	Clusters	Update Method, Θ, Δ	Initialization Time (s)	Update Time (s)
Kmeans	8	Sequential	0.511	749
Kmeans	16	Sequential	0.721	798
Kmeans	8	DYNC, $\epsilon = 0.5$	0.511	3989
Kmeans	16	DYNC, $\epsilon = 0.5$	0.721	2369
Ward	8	Sequential	0.297	476
Ward	16	Sequential	0.297	536
Ward	8	DYNC, $\epsilon = 0.5$	0.297	3774
Ward	16	DYNC, $\epsilon = 0.5$	0.297	2309
ROC, RBF kernel, $\epsilon = 100$	-	Sequential	8.059	823
ROC, RBF kernel, $\epsilon = 100$	-	DYNC, $\epsilon = 0.5$	8.059	2197
Centroid	8	-	0.280	-
Centroid	16	-	0.280	-
DBSCAN	-	Sequential	1.153	3829

its cohesion is very low. The sequential update method only adds new boilers to the large cluster because the singleton clusters are outliers.

DBSCAN presents some of its own problems. Because it focuses on finding non-hyperspherical clusters, the Euclidean distance measures used in both the sequential the DYNC methods show abnormal behavior. At the outset, most points immediately recluster under the sequential method, very likely to shuffle the non-hyperspherical clusters into approximately hyperspherical ones. DYNC experiences the same problems that it had with the centroid method, where adding new points with Euclidean distance measures rapidly decreases the cohesion of each cluster and decreases the number of clusters rapidly. A fix was attempted where DBSCAN would be allowed to recluster from all points again once the number of clusters fell below a certain threshold, but this simply created a rapid and very computationally-expensive loop of frequent reclustering.

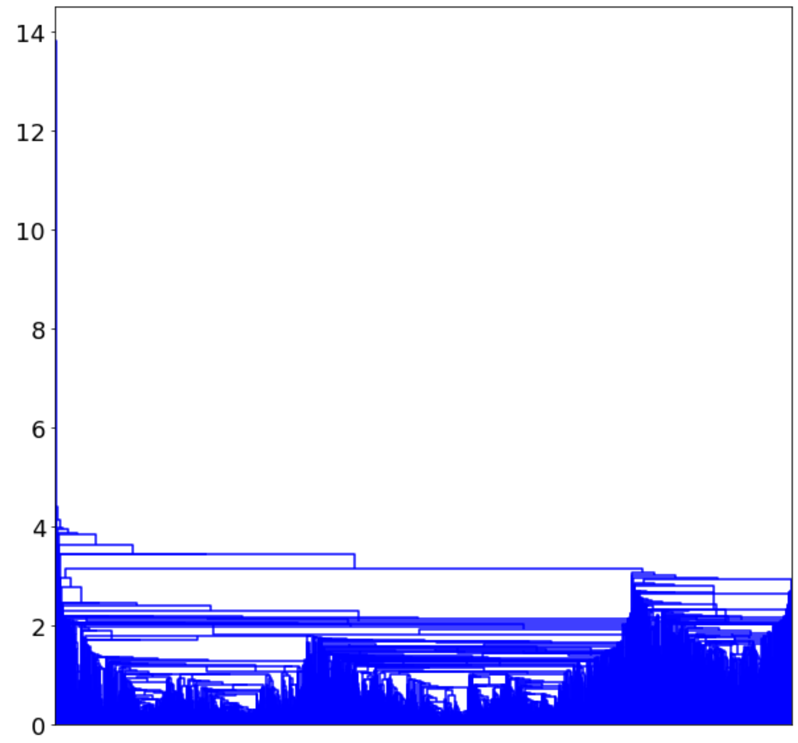


Figure 17: Boiler Dendrogram Clustered by the Centroid Method

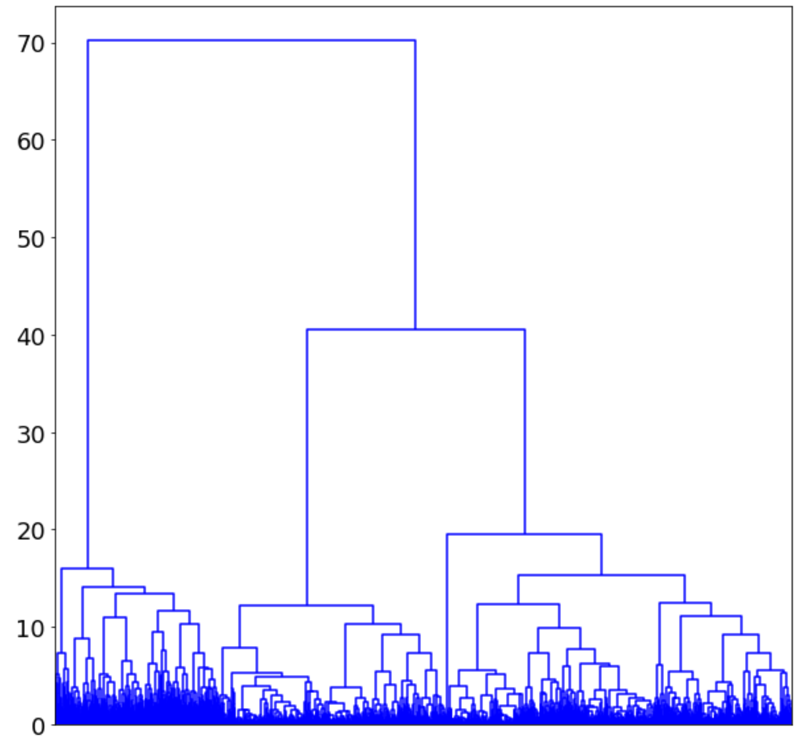


Figure 18: Boiler Dendrogram Clustered by Ward's Method

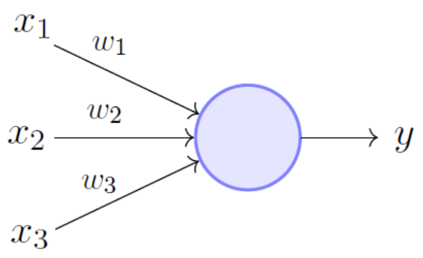


Figure 19: Example of a Perceptron [41]

2.3 Prediction

For predicting future values of sensor readings, we use an artificial neural network (ANN) system. This model (and all ANNs) are made up of layers of *perceptrons*. These are *artificial neurons* that are governed by simple rules. An example of one is shown in Figure 19. Each perceptron has vector of weights, $\vec{w}$, and a bias, b. Their output is governed by Equation 8. Multiple neurons, like these perceptrons, connected together in a network form a *neural network* (Figure 20) which creates a multi-layer perceptron (MLP) neural network. Each neural network has at least 3 layers: an *input layer*, an *output layer*, and at least 1 *hidden layer*.

$$y = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ \\ 0 & \text{if } \vec{w} \cdot \vec{x} + b <= 0 \end{cases} \tag{8}$$

In our case, the objective function (alternatively called the loss function) is the *log-loss function* shown in Equation 9 from before, In our central model, our output vector elements are binary as an indicator for each part, so $p_{i,j}$ is either 0 or 1. The Adam authors' recommended hyperparameter values of $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-8}$ are used.

$$\text{log loss} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{M} y_{i,j} \log(p_{i,j}) \tag{9}$$

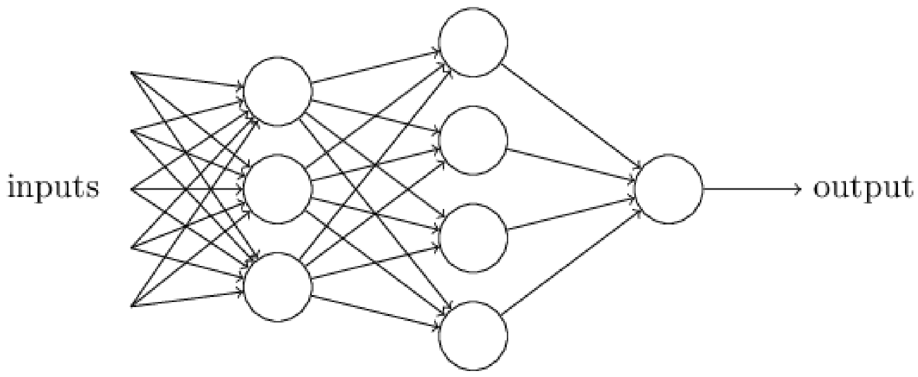


Figure 20: Example of a Simple Neural Network [42]

```
**Input:** $\alpha$: The step size
**Input:** $\beta_1, \beta_2 \in [0, 1)$: The exponential decay rates
**Input:** $f(\theta)$: The stochastic objective function
**Input:** $\theta_0$: The initial parameter vector
**Output:** $\theta_t$
$\vec{m}_0 \leftarrow 0$ Initialize the first moment vector
$\vec{v}_0 \leftarrow 0$ Initialize the second moment vector
$t \leftarrow 0$ Initialize the timestep
**while** $\theta_t$ *not converged* **do**
    $t \leftarrow t + 1$
    $\vec{g}_t \leftarrow \Delta_\theta f(\theta_{t-1})$
    $\vec{m}_t \leftarrow \beta_1 \cdot \vec{m_{t-1}} + (1 - \beta_1) \cdot \vec{g}_t$
    $\vec{v}_t \leftarrow \beta_2 \cdot \vec{v_{t-1}} + (1 - \beta_2) \cdot \vec{g}_t \odot \vec{g}_t$
    $\hat{m}_t \leftarrow \vec{m}_t / (1 - \beta_1^t)$
    $\hat{v}_t \leftarrow \vec{v}_t / (1 - \beta_2^t)$
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$
**end**
```

Algorithm 11: Pseudocode implementation of Adam [43]

Table 6: Input and Output Prediction Parameters for ANN Model

Inputs	Outputs
Boiler Status	Fan Speed
Max Fan Speed	Power
Min Fan Speed	Pump Input Voltage
Hot Water Generator Max Fan Speed	Pump Output Voltage
Power	Boiler Pump Status
Pump Input Voltage	DHW Pump Status
Pump Output Voltage	System Pump Status
Boiler Pump Status	Flame Current
DHW Pump Status	Flue Temperature
System Pump Status	HWG Temperature
Flame Current	Inlet Temperature
Flue Temperature	Outlet Temperature
HWG Temperature	
Inlet Temperature	
Outlet Temperature	
Setpoint	
HWG Setpoint	
Minimum Outdoor Temperature	
Maximum Outdoor Temperature	

The symbol $\odot$ indicates the *Hadarmard product*, an operator commonly used in neural network training algorithms. It simply means elementwise multiplication of vector elements. For example, $\begin{bmatrix} 5 \\ 6 \end{bmatrix} \odot \begin{bmatrix} 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 35 \\ 48 \end{bmatrix}$. The Adam algorithm continues until the model reaches *convergence*. This occurs when the magnitude in the change in θ_{t-1} to θ_t falls below a defined threshold.

With this method in place, we need a set of inputs and outputs to train and test against. As inputs, for each boiler, we look at the previous 7 days of each of the input parameters in Table 6 and calculate their mean. For outputs, we simply use the current day's average value of the output parameters. To make use of the clusters, we use the process shown in Figure 21 for each of our desired outputs. The submodel predictions are graded on mean average error (MAE) and root mean square error (RMSE).

The ANN model used is a feedforward MLP neural network described above. It has two hidden

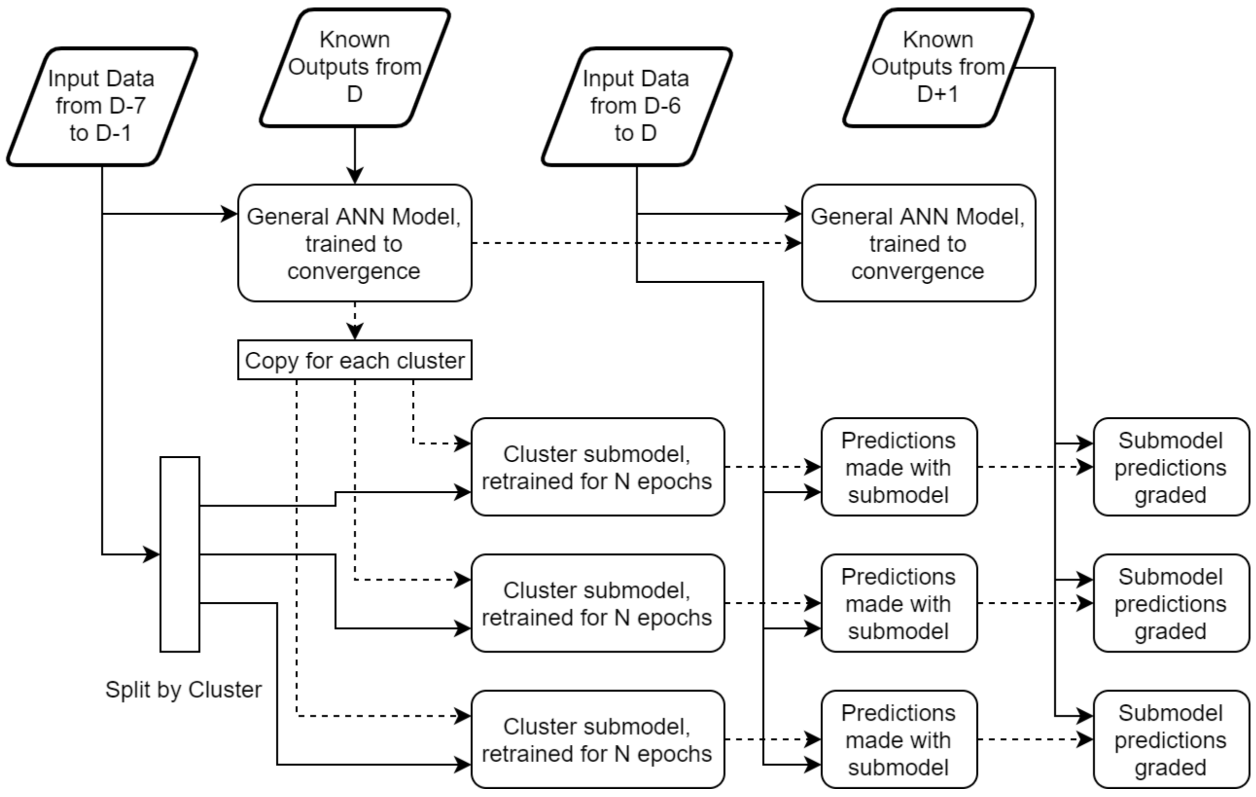

Input Data from D-7 to D-1
Known Outputs from D
Input Data from D-6 to D
Known Outputs from D+1
General ANN Model, trained to convergence
General ANN Model, trained to convergence
Copy for each cluster
Split by Cluster
Cluster submodel, retrained for N epochs
Cluster submodel, retrained for N epochs
Cluster submodel, retrained for N epochs
Predictions made with submodel
Predictions made with submodel
Predictions made with submodel
Submodel predictions graded
Submodel predictions graded
Submodel predictions graded

Figure 21: Cluster-Based ANN Modeling System

layers - the first has 80 neurons and the second has 40. When this general model is trained to convergence, we use a standard threshold of 0.0001. When this general model is copied for each cluster on each date, the neural network is only trained for another $N = 100$ times, or *epochs*, through the system. Submodels trained in this way dn not reach convergence on their specific cluster data. This is intentionally done to prevent *catastrophic overfitting*. Ideally, the submodels should have some mix of knowledge from both the general, unclustered data and from its specific cluster. If they are trained to convergence, they forget about patterns they observed in the whole dataset and then perform very poorly at the grading stage. Ideally, they retain general knowledge while favoring information that they saw in the cluster-specific data.

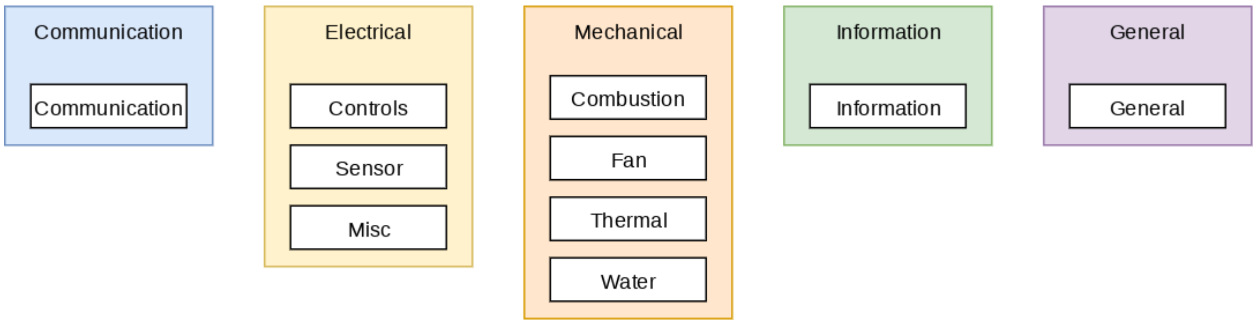


Figure 22: Symptom Groups and Types

III CASE-BASED REASONING

1 Problem

We have a set of 19,769 customer support cases related to boilers. This set includes boiler serial numbers, dates, a symptom code, a solution code, and some case comments as shown in Table 7. From these cases we seek to build a case-based reasoning (CBR) model to assist with solving new cases. Such a system would allow users to input information about the current case - including selecting symptom types, groups, or specific symptoms or through entering text comments.

In the data, there are 92 possible symptom codes and 40 possible solution codes. In order to support grouping of the symptom codes, we further categorize each into one of the 10 possible symptom groups shown in Figure 22. Each of these symptom groups then is assigned to one of 5 symptom types, also shown in Figure 22. Figure 23 shows the distribution of the cases within the symptom groups and symptom types.

2 Method

The full CBR system is shown in Figure 24. It is made up of 3 main models or methods: a predictive CBR system, a comment-based model, and a pure probability model. We will go through

Table 7: Example Technician Case Data

Serial Number	Date/Time Opened	Symptom Code	Solution Code	Case Comments
1621102968787	4/23/2018 9:55 AM	Display Blank	Parameter explanation	Cycled power and display still did not come on. Explained to tech to check cables and if necessary replace display communication board.
1743107965587	12/19/2017 8:51 AM	Flame Failure Ignition	Part info Provided	Unit is not lighting, 7″ static, 24 volts to gas valve but no dynamic drop. Based on what tech told me I advised him to replace gas valve
I06H00190891	7/25/2018 11:00 AM	Hard Starts	Cleaning or Adjustment	Recommended cleaning the burners & Hex
L13C20287408	9/18/2017 8:40 AM	Outdoor Sensor Fault	Part Replaced	Said the heater doesn't reset when he presses the reset button. Told would need to check the resistance across the sensor and check connections to the sensor. Wanted to know which sensor this was referring to. Told the outlet sensor. Customer said was going to purchase one through the local distributor.
1526101697898	7/19/2018 10:11 AM	Flame Fail Running	Combustion adjustments	Unit has been cleaned, experiencing low flame signal, explained how to check and adjust combustion

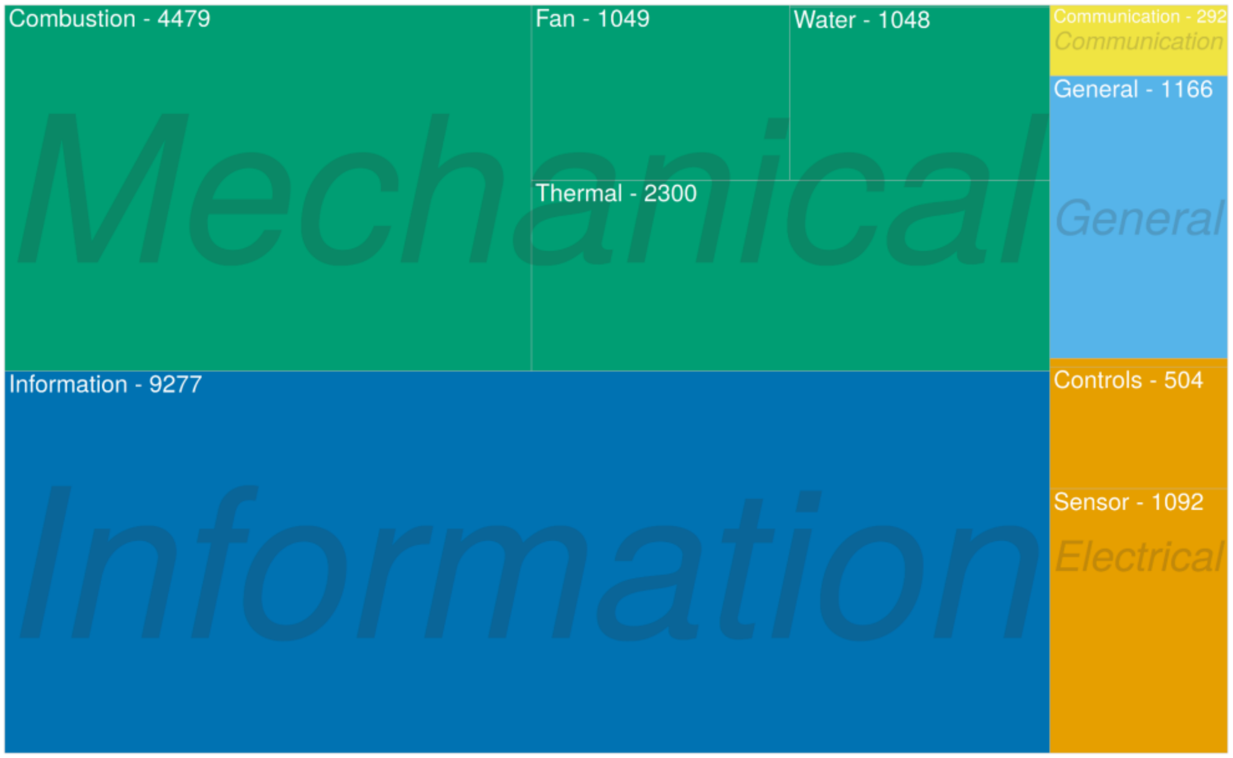


Figure 23: Case Data Distributions

each of these in turn, starting with the pure-probability model.

2.1 Pure Probability Model

The pure probability model is very simple. For each case, it considers only the symptom types, groups, and codes and the solution codes. Initially, for each of the four categories, it recommends the most-seen option in the dataset. If a user selects a higher-level category, it recalculates the probability down-the-line based on previously-seen cases. For example, a user may select that they have a mechanical type of symptom but not yet be able to narrow it down further. The group, symptom, and solution probabilities will be redetermined based what was seen from cases that had mechanical issues. This is shown in steps 1 and 2 in Figure 10. Keep in mind, there are 5 symptom types, 10 symptom groups, 92 symptom codes, and 40 solution codes. We only show the top 3 most likely for each category to save space. For step 3, the user might select that the gas pressure switch is open. The system tells the user that, based on prior probabilities when the gas pressure switch was open, the most probable cause is an electrical problem. If it isn't that, then the part may need to be replaced or there might be a gas supply issue.

Figure 26 shows another example. In this case, the user immediately knows that the pump is not working. When this is selected, the model back-fills the parent categories of that selection to 100% probabilities before suggesting fixing an electrical problem, explaining a parameter, or providing part information are the three most-probable solutions.

It is important to note that this method is not optimized in any way for improved accuracy - it is purely based on showing the most common categories and solutions in previous cases. The intent of this model is to have a reasonable fall-back when no other information such as text comments are provided by a user. Additionally, you may notice the high probability of generic categories and solutions such as ”product information” and ”customer instruction.” In many cases, the text

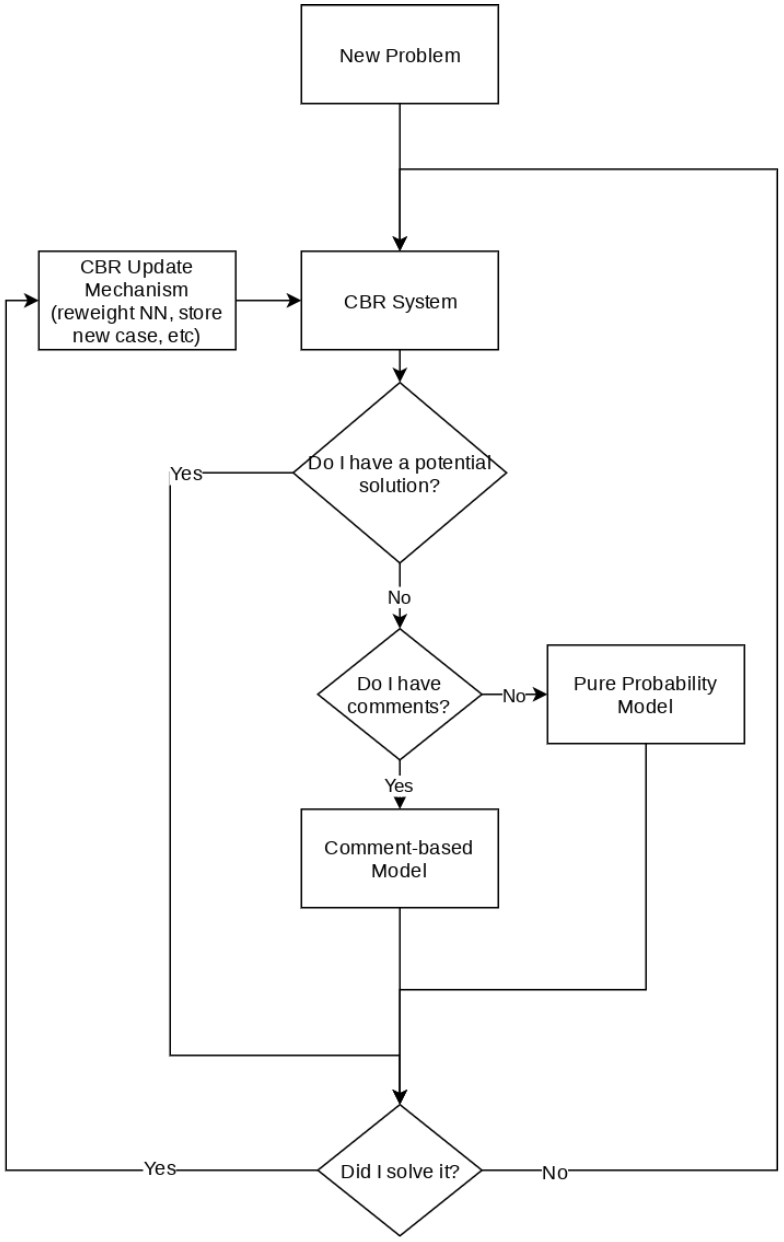
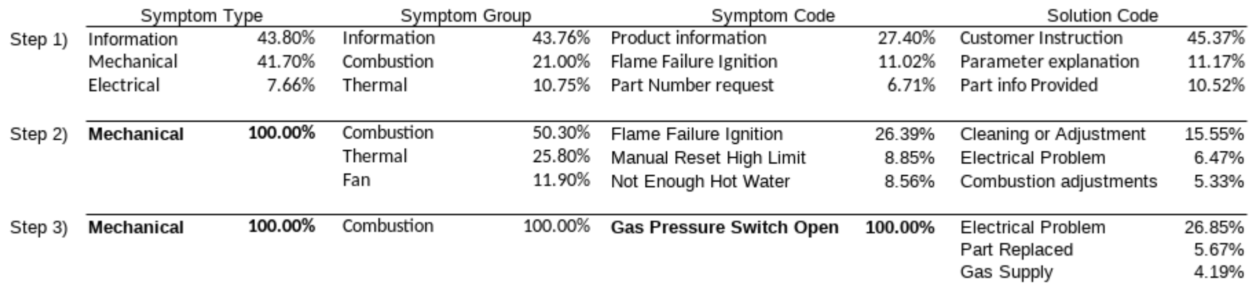
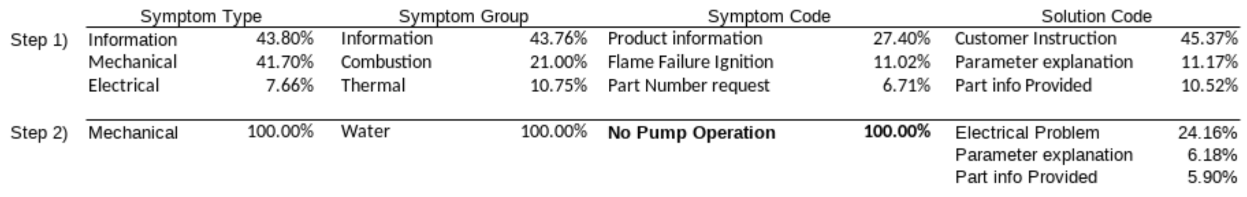


Figure 24: Full Case-Based Reasoning Method



	Symptom Type		Symptom Group		Symptom Code		Solution Code	
Step 1)	Information	43.80%	Information	43.76%	Product information	27.40%	Customer Instruction	45.37%
	Mechanical	41.70%	Combustion	21.00%	Flame Failure Ignition	11.02%	Parameter explanation	11.17%
	Electrical	7.66%	Thermal	10.75%	Part Number request	6.71%	Part info Provided	10.52%
Step 2)	**Mechanical**	**100.00%**	Combustion	50.30%	Flame Failure Ignition	26.39%	Cleaning or Adjustment	15.55%
			Thermal	25.80%	Manual Reset High Limit	8.85%	Electrical Problem	6.47%
			Fan	11.90%	Not Enough Hot Water	8.56%	Combustion adjustments	5.33%
Step 3)	**Mechanical**	**100.00%**	Combustion	100.00%	**Gas Pressure Switch Open**	**100.00%**	Electrical Problem	26.85%
							Part Replaced	5.67%
							Gas Supply	4.19%

Figure 25: Pure Probability Example #1



	Symptom Type		Symptom Group		Symptom Code		Solution Code	
Step 1)	Information	43.80%	Information	43.76%	Product information	27.40%	Customer Instruction	45.37%
	Mechanical	41.70%	Combustion	21.00%	Flame Failure Ignition	11.02%	Parameter explanation	11.17%
	Electrical	7.66%	Thermal	10.75%	Part Number request	6.71%	Part info Provided	10.52%
Step 2)	Mechanical	100.00%	Water	100.00%	**No Pump Operation**	**100.00%**	Electrical Problem	24.16%
							Parameter explanation	6.18%
							Part info Provided	5.90%

Figure 26: Pure Probability Example #2

comments provide additional information - for example, the technician may have entered ”part info provided” and then comment that part information for a gas valve was entered so that it can be replaced. Arguably, ”part replaced” would have been a better solution to select, but such problems are not handled until the predictive model.

2.2 Comment-Based Model

In order to construct a comment-based system around the technician call data, a natural language processing (NLP) tool is needed. Such a tool needs to be *adaptable*, where it is able to update its processing abilities as new information is collected in the CBR adaptation step. It also needs to be able to handle imperfect spelling, grammar, and sentences, as seen in the example entries in Table 7.

Another requirement is that the NLP method must be able to be fed into a higher-level machine learning method so that category predictions can be generated. One way to do this is to provide numeric vectors represent words, groups of words, or sentences. A commonly-used family of methods is known as n-gram modeling [44]. At its core, it is based on the assumption that the probability of a given word occurring next in a sequence of English words can be defined as a sequence of conditional probabilities. Formally, this is defined as $\Pr(w_k|w_1^{k-1})$, where w_1^k is defined with Equation 10 [44].

$$\Pr(w_1^k) = \Pr(w_1) \cdot \Pr(w_2|w_1) \cdots \Pr(w_k|w_1^{k-1}) \tag{10}$$

Because developing a system based fully off of Equation 10 would be incredibly computationally expensive and only provide useful predictions in situations where the model had seen the exact sentence used as an input before, n-gram modeling introduces a simplification. For the kth word in a sentence or phrase, $\Pr(w_k|w_{k-n+1}^{k-1}) \approx \Pr(w_k|w_1^{k-1})$. With approximation, only n conditional probabilities need to be calculated to predict a given word. With a given set of training data, these probabilities are calculated prior to model use using frequency analysis.

As a demonstration, let's use the final sentence 4th example sentence from Table 7: "Customer said he was going to purchase one through the local distributor." Without the n-gram simplification, predicting the final word, "distributor", from the sentence would be given by: $\Pr(\text{distributor}|\text{customer said he was going to purchase one through the local})$. Unless the exact sentence has been seen before, a language model would have no way of predicting "distributor" at the end of the sentence.

In a bigram (2-gram) model, the probability of the next word in a string of English words becomes [45]:

$$\Pr(w_n|w_1, w_2, \cdots w_{n-1}) = \Pr(w_1) \cdot \Pr(w_2|w_1) \cdots \Pr(w_n|w_{n-1}) \tag{11}$$

In a trigram (3-gram) model, it is [45]:

$$\Pr(w_n|w_1, w_2, w_3 \cdots w_{n-1}) = \Pr(w_1) \cdot \Pr(w_2|w_1) \cdot \Pr(w_3|w_2, w_1) \cdots \Pr(w_n|w_{n-1}, w_{n-2}) \tag{12}$$

In many cases, calculating these probabilities directly is computationally expensive, and a further simplification is used, called *maximum-likelihood estimation* (MLE) [44]. For bigrams, we first define $C(x, y)$ as number of times the bigram x, y occurs in a corpus. With this, we can determine the bigram frequency of a word within a corpus:

$$\Pr(w_n|w_{n-1}) = \frac{C(w_{n-1}, w_n)}{\sum_w C(w_{n-1}, w_n)} \tag{13}$$

For trigrams, we instead use:

$$\Pr(w_n|w_{n-1}, w_{n-2}) = \frac{C(w_{n-1}, w_{n-2}, w_n)}{\sum_w C(w_{n-2}, w_{n-1}, w_n)} \tag{14}$$

So far, this does not give us vector representations of words and sentences. For this, we use Word2Vec [46] and Doc2Vec [47], which are two methods built on top of n-gram modeling that convert word and sentence meaning to numeric vectors.

Instead of working directly with n-gram models, Word2Vec and related models use the *skipgram* architecture [46]. Where n-gram models seek to predict the probability of the next word following a sequence of words, skip-gram models predict the probability words surrounding a given word, both before and after. The extent to which this prediction is done is called the *window*, where, if n is the window size, $\frac{n-1}{2}$ words are predicted before the given word, and another $\frac{n-1}{2}$ words are predicted after. This is represented by Figure 27.

The key to extracting vector representations of words takes place in the *projection* part of the skip-gram model. While in most machine learning techniques we create and train a model in this phase to be used for some end, in this case the weights of the generated model is the end itself. The basic principal behind Word2Vec is, knowing both the input words and their associate skip-gram output words, training a model that can accurately predict the output word probabilities, extracting the model weights as a vector representation of the word, and discarding the model. This model is typically a feedforward neural network [46], which is also the type of model we use within the CBR system.

To begin the process, input and output vectors are generated for each word. The input vector has one element for each of the unique words in the corpus. For a given word, it is a one-hot vector where the element representing the word is 1 and the rest of the elements are 0. The output vector has the same length as the input vector, but each element holds the probability that its associated word is seen within the window of the input word. A model (usually a neural network) is created next. This model takes the input and output vectors for each word and trains on them using standard or optimized neural network training methods, and the vector representation of the word is defined as the weights of the hidden layer neurons corresponding to the one-hot vector element for a word. Therefore, the size of the hidden layer is the length of the vector that will represent each word. Mikolov, et. al's original paper does not give a suggested neural network optimization method and backpropogation algorithm when generating Word2Vec vectors with this technique. While general optimization methods such as Adam [43] (Algorithm 11) will work, Rong has developed a computationally-efficient method for Word2Vec neural network optimization [48]. We use a Python implementation of this method from the GENSIM library [49]. Doc2Vec works very similarly, except when training the input vector is a concatenation of the one-hot word vector and a one-hot vector of the length of the number of documents indicating the document that the current

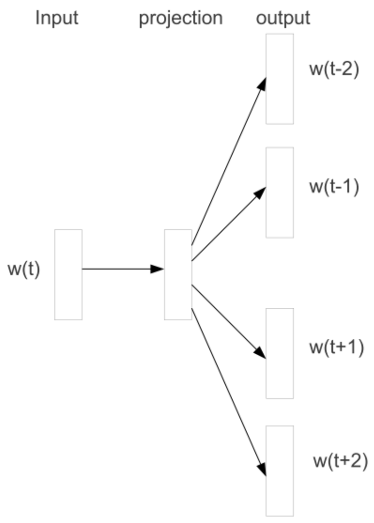


Figure 27: The Skip-Gram Architecture [46]

word sample comes from. The intent of this is to create a model that is aware of both the likelihood of occurrence of surrounding words, but also the semantic meaning of the input documents. In the end, we don't classify these semantic meanings into human terms, like how we discard the neural network model when the vectors are extracted.

Now that we have a method of extracting vectors from the case comments, we need a way of using this to predict four categories of symptom and solution classes: symptom type, symptom group, symptom code, and solution code. To help with this, various corpa in Table 8 were used when training the Word2Vec and Doc2Vec models. The intent of training the word vector models both with the case comments and with general-purpose corpa is to provide more general examples of English for the model to learn from than are found exclusively in the case comments. The first step is to create Word2Vec and Doc2Vec models using various combinations of corpa. These models are generated after the following combinations:

- Comments only
- Comments and Brown

Table 8: Corpa

Corpa	Description
Case Comments	The case comments from the technician support cases
Brown	The Brown University Standard Corpus of Present-Day American English
NPS Chat Corpus	A collection of 10,567 online chat and forum posts
Yahoo Answers	A collection of 60,000 questions and answers

- Comments and NPS
- Comments and Yahoo

Before being fed into the Word2Vec or Doc2Vec model, each sample sentence is preprocessed. First, all punctuation is removed. Next, the sentence is *tokenized*, where each word, number, or abbreviation is separated into its own element of a list. Each tokenized word is tagged with its part-of-speech (such as nouns, verbs, adverbs, etc.) using NLTK's tokenizer [50]. *Stop words* are then removed, which are common ″connecting″ words such as ″you″, ″they″, ″and″, and ″so″. For a full list of these, see Appendix B 2. A second filter is then used, removing any word that is not a noun, verb, adjective, or adverb. Finally, *lemmatization* is completed. This reduces each word to its base, singular, present-tense form. For example, ″ran″ becomes ″run″, and ″racing″ becomes ″race″. The lemmatizer used here is the one built-in to WordNet, an English lexical database published by Princeton [51]. It was accessed using using NLTK [50].

Now, with the Word2Vec and Doc2Vec models generated, we can extract numeric vector representations for words and sentences. In this case, we will use one vector that represents each case comment. For the Doc2Vec model, a vector representation of the case comment, or document, is calculated. For Word2Vec, a vector representation is generated for each word. These word vectors are then averaged per element to create a vector for the entire case comment. The vectors are then used to train support vector classifier (SVC) models using the kernels found in Table 9 and also a logistic regression model. An explanation of the principles behind SVCs and logistic regression

Table 9: Kernels used in SVC Models

Kernel	Definition
Linear	$K(\vec{x},\vec{y}) = \vec{x}^T\vec{y}$
Polynomial	$K(\vec{x},\vec{y}) = (\sigma\vec{x}^T\vec{y} + r)^d$
RBF	$K(\vec{x},\vec{y}) = \exp(-\frac{\lvert\lvert x-y\rvert\rvert^2}{2\sigma^2})$
Sigmoid	$K(\vec{x},\vec{y}) = \tanh(\sigma\vec{x}^T\vec{y} + r)$

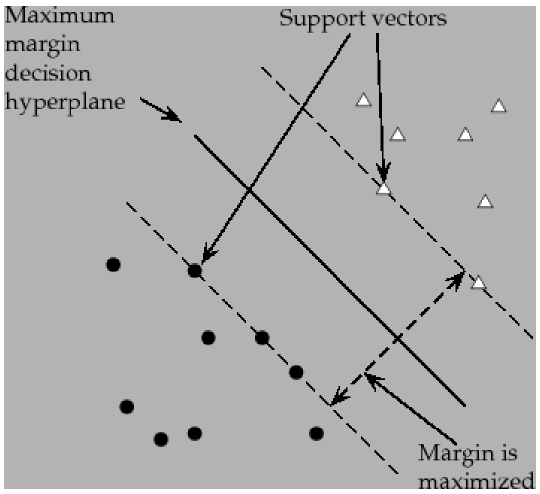


Figure 28: 2D SVC Example [36] (https://nlp.stanford.edu/IR-book/html/htmledition/support-vector-machines-the-linearly-separable-case-1.html)

follows.

Support Vector Classifiers (SVC) To demonstrate how a support vector classifier (SVC) works, we start with a simple, 2-dimensional case. Say we have two sets of points, such as the white triangles and black dots from Figure 28. We seek to create a line (or a hyperplane in higher dimensions) that splits the two sets, or categories. Such a line, or *decision boundary*, is indicated by the solid black line in Figure 28. Unlike simpler linear classification methods which simply try to find any decision boundary that works on the training data, SVCs find the boundary that maximizes the distance, or *margin* to the nearest data points [36]. To achieve this, SVCs work to identify the key data points that will be used to determine the margin while ignoring the rest, and then they use these selected data points to create *support vectors*.

In SVCs, the decision boundary is defined by its bias or intercept, b, and a vector that is normal to the hyperplane, $\vec{w}$. Now, let us assume that we have a series of training data, each with an input point, $\vec{x}_i$, and a corresponding output value, $\vec{y}_i$. The output value can belong to one of two classes, represented by -1 and 1.

For any point, we can determine the distance from the boundary, r, using Equation 15 [36]. We can then use the following requirements and constraints: $\frac{|\vec{w}|}{2}$ needs to be minimized, and $y_i\left(\vec{w}^T\vec{x}_i + b\right) \geq 1$. The first requirement is based on the idea that minimizing the magnitude of $\vec{w}$ necessarily makes the margin larger. For the second, any value > 0 could be used in place of 1. What is important is that all input-output pairs follow the same constraints. 1 is usually used because it is easy to remember and can make some calculations in the minimization simpler.

$$r = y\frac{\vec{w}^T\vec{x} + b}{|\vec{w}|} \tag{15}$$

With these constraints in place, we can use a quadratic minimization method to solve for the boundary. In this case, we use an implementation of the popular LIBSVM [52], a specialty quadratic minimization library designed specifically for SVMs and SVCs. Using the concept of Platt scaling [53], margin distance is also used to predict the probability of a class being correct. Additionally, the LIBSVM method can (and must for the tech support comment vectors) be extended to multi-class outputs by adding more boundary lines or planes. A simple example of this extension with 2-dimensional data is shown in Figure 29.

These simple examples work in the case of clearly-defined groups of points that are easily separated by linear boundaries, but in reality such situations are rare. The minimization method in these simple examples determined the closeness of points to each other through the Euclidean distance ($\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$). However, let's say that we have points illustrated in Figure 30. A human can easily see the two categories of points, but an SVC using Euclidean distance will not be

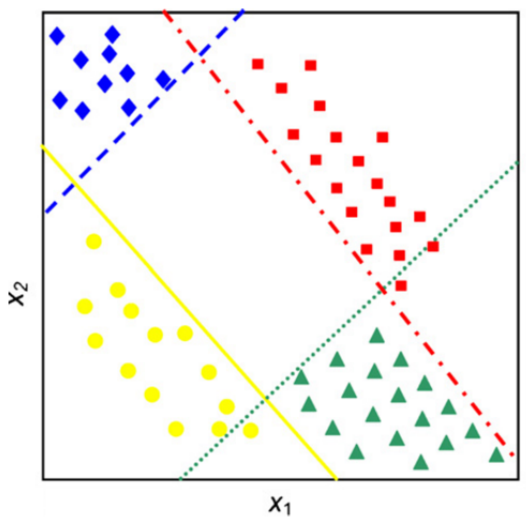


Figure 29: 2D Multi-Class SVC Example [54]

able to find a line that separates the two groups from each other.

To solve this problem, we use the *kernel method* (or *kernel trick*). These are, effectively, alternative distance measures calculated through a higher dimension. In the case of Figure 30, we can use a new kernel given by Equation 16. This means that distances between points are not being considered in $\mathbf{R}^2$, but now are in $\mathbf{R}^3$ as illustrated by Figure 31. Once the kernel method is applied, an SVM/SVC minimizer like LIBSVM is able to find a hyperplane that divides the two groups of points, as shown in Figure 32.

$$K(\vec{x},\vec{y}) = K([x_1,x_2],[y_1,y_2]) = \sqrt{(x_1-y_1)^2,(x_2-y_2),((x_1^2+x_2^2)-(y_1^2+y_2^2))} \tag{16}$$

Logistic Regression Logistic regression models are another technique that can be used to generate class-based probabilities of classes after being given a set of training inputs and classes. It is related to linear regression, where for a set of input points $\{\vec{x}_1,\vec{x}_2,\vec{x}_3,\cdots\}$ and associated output values $\{y_1,y_2,y_3,\cdots\}$, we attempt to find a weight vector, $\vec{w}$, and bias, b, such that $\frac{1}{N}\sum_{i=1}^{N}\hat{y}_i - y_i$ is minimized where $\hat{y}_i = \vec{w}\cdot\vec{x}_i + b$.

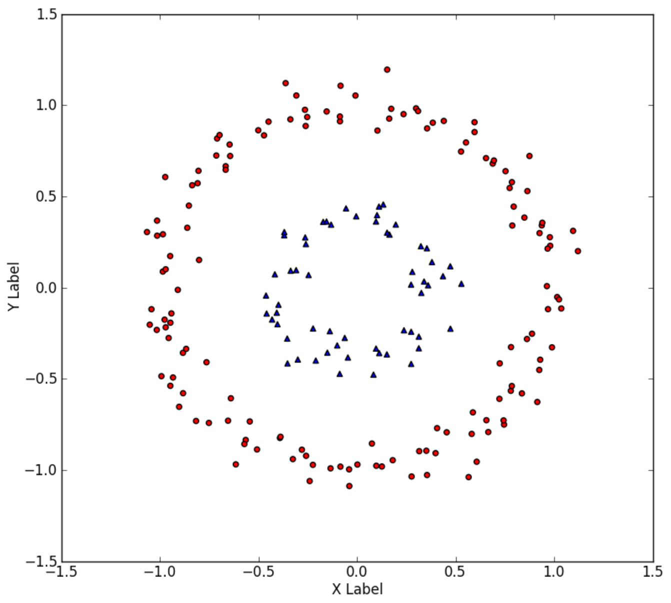


Figure 30: Example Points that cannot be Separated by a Boundary Line [55] (https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html)

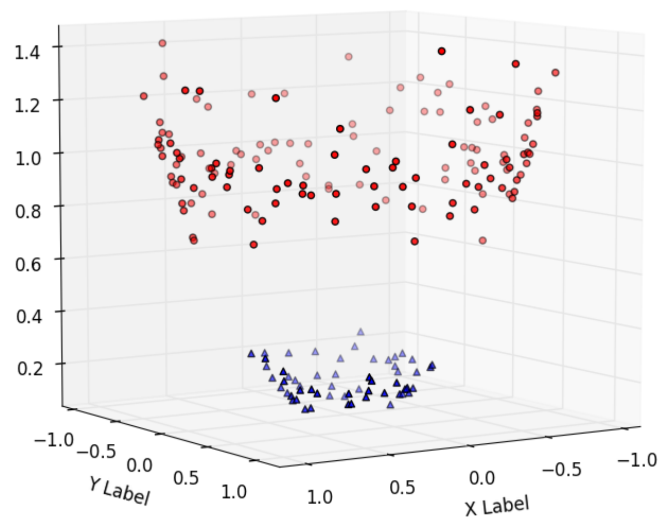


Figure 31: Example Points with Kernel Method Applied [55] (https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html)

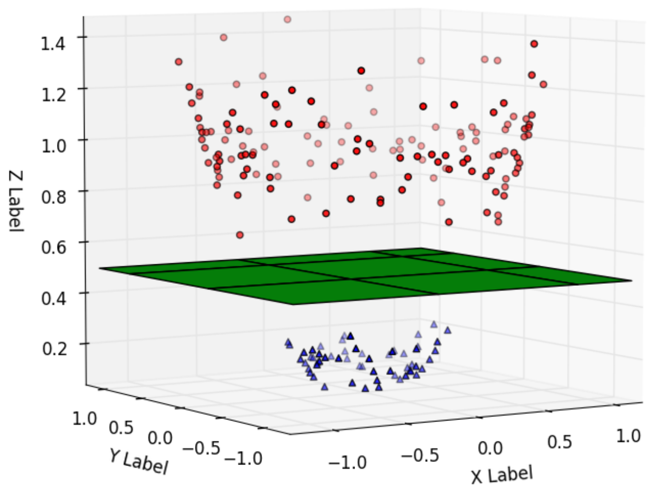


Figure 32: Kernel Method Example with Hyperplane [55] (https://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html)

In logistic regression, we keep most things the same, but $\hat{y}$ is calculated differently, and y now is a binary value indicating whether or not an event occurred. We now use the *sigmoid function* given by Equation 17. Once the weights, $\vec{\beta}$ are found, the logistic function in Equation 18 can be used to determine the probability of a new $\vec{x}$ resulting in the selected event. A final rearranging, shown in Equation 19, directly gives the probability.

$$\hat{y} = \frac{1}{1 + e^{-\vec{\beta}^T \vec{x}}} \tag{17}$$

$$\frac{p}{1-p} = \vec{\beta} \cdot \vec{x} \tag{18}$$

$$p = \frac{1}{1 + b^{-\vec{\beta} \cdot \vec{x}}} \tag{19}$$

For multi-class scenarios like with the types, groups, symptoms, and solutions of the case data, a few additional adjustments are needed. The bias, b, is now expanded into a vector, $\vec{b}$, as is $\hat{y}$, for each possible class. Also, the sigmoid function is replaced with the softmax function Equation 20, where M is the number of classes. Finally, instead of using the $\hat{y} - y$ error as a the value to minimize, the log-loss function of Equation 9 is used instead, where M is the total number of predicted classes j is a specific class, N is the number of training examples, i is a specific training example, $y_{i,j}$ is 1 when i is a member of class j, and $p_{i,j}$ is the model's calculated probability that i is a member of j. Similar to SVCs where multiple proven minimization methods are available, with the logistic regression analysis we use the popular L-BFGS [56]. It is a simplified version of its predecessor BFGS, and is useful in cases where there are a large number of possible classes, such as with the 92 possible symptom codes and 40 possible solution codes.

$$\hat{y} = \frac{e^{\vec{\beta}\cdot\vec{x}+\vec{b}}}{\sum_{i=1}^{M} e^{(\vec{\beta}\cdot\vec{x}+\vec{b})_i}} \tag{20}$$

With the SVC and logistic regression methods outlined above and the kernels in Table 9, we create several machine-learning models that output class probability. The implementation uses the Scikit-Learn Python library [57]. For training, these use 80% of the 64 and 150 element word vectors as inputs, and each word vector's associated type, group, symptom, and solution as outputs. After training, the remaining 20% of held-back word vectors are sent through the SVCs, which calculate probabilities of each of the types, groups, symptoms, and solutions being correct. These probabilities are scored as described in the Results section. Finally, case adaptation is handled very easily. When a new successful case is added, the weights of the Word2Vec neural network is updated to converge against the case base with the new case included. The SVC or logistic regression model is then retrained using the slightly-updated word vectors.

2.3 Predictive System

Regardless of the type of CBR used, we need a way to represent both inputs and outputs of such a system. A case is made up of three main types of information: the parts that have already been investigated, the severity of the action used on the previously-mentioned parts, and a vector

Table 10: Parts considered in the CBR model

Air inlet	Display board	Gas pressure switch	Outlet sensor
Air pressure switch	Door	Gas valve	Outlet temperature differential
Aquastat	Drain switch	Heat exchanger	Pump
Baffle	Expansion tank	Igniter	Relief valve
Burner	Fan	Indirect tank	Ribbon cable
Communication board	Flame rod	Inlet sensor	Spark cable
Condensate drain	Flame sensor	Inlet temperature differential	Tank sensor
Condensate trap	Flow switch	Insulation	Transformer
Control board	Flue	Low voltage board	Wiring
Display	Flue sensor	Manifold	Wiring harness

Table 11: Actions considered in the CBR model

Action	Severity Multiplier	Stems
Check	1	Check, Connect, Inspect, Normal, Expected, Nominal, Fine
Cleaning	2	Clean
Unblock	3	Block
Adjustment	4	Adjust, Increas, Decreas
Service	5	Servic
Repair	6	Repair
Replacement	7	Replac

representation of the case comments. Table 10 contains a list of parts, and Table 11 contains a list of actions and their associated severity multipliers. We use these in the aim of creating three vectors to represent each case:

- Parts vector
- Action severity vector
- Comments vector

For each of the known cases, we need a way of extracting parts, actions, and pairing actions with their associated parts. For this, we use Algorithm 12

```
Input: comment
Input: parts list
Input: actions list
Output: $\vec{parts}$, a vector of parts identitified in the comment
Output: part action pairs, the list of parts with their associated actions
$\vec{parts} \leftarrow []$
$\vec{actions} \leftarrow []$
$part\ \vec{action}\ pairs \leftarrow []$
for each word and word index in comment do
    if word $\in$ parts list then
        $\vec{parts} \leftarrow$ (word, word index)
    end
    if word $\in$ actions list then
        $\vec{actions} \leftarrow$ (word, word index)
    end
end
for each (part, part index) in parts list do
    part action pair $\leftarrow$ (part, action with closest action index to part index)
    part action pairs $\leftarrow$ part action pair
end
```

Algorithm 12: Part and Action Pair Extraction Algorithm

After the parts and part-action pairs are extracted, we create two *mult-hot vectors* based off of each list. Each element of these 40-element vectors is 0 unless their corresponding part (sorted by alphabetical order) is observed. For example, a comment with an extracted parts list of [burner, fan, gas valve] would have its 4th, 14th, and 21st elements set to 1. The part-action multi-hot vector is created similarly, except the element associated with each seen part is assigned its multiplier from Table 11. Thus, a comment with extracted part-action pairs of [(burner, clean), (fan, check), (gas valve, adjust)] would have 2 in its 4th element, 1 in its 14th element, and 4 in its 21st element. A third vector is created using the previously-trained 64-element Word2Vec model. While the 150-element Word2Vec model shows higher accuracy, it is better to limit the length of the vector in this case to prevent overfitting of the CBR system to specific words and phrases used in comments. Ideally, the CBR system will only pick up on the general meaning of comments. Once each of these three vectors are generated, they are concatenated together into a 144-element *case input vector*.

We also need *case output vectors* that represent both solutions and provide a way to train the CBR system through feeding it correct answers. It is essential that the CBR system be able to tell a user what other steps should be taken given the level of action taken with previous parts, not simply to learn to pass certain elements of the input vector through. To do this, we create n input vectors with 144 elements and n output vectors with 40 elements (one for each part) from the original case input vector, where n is the number of parts in the case. For each new input vector, we mask one of the parts and its associated action. For each new output vector, we mask every element but the associated part element to reduce the multi-hot vector into a one-hot vector. This step is illustrated in Figure 33 for the previous burner, fan, and gas valve example.

With the case input vectors, output vectors, and the previously-mentioned CBR types in mind, we create three CBR systems. For all three, we generate a case-base (CB) that contains the case input vectors and case output vectors for each of the selected training comments. The first CBR system is a substitution system using nearest-neighbor search to find the case in the case-base that most matches the case to be solved. Let us say that c_{nearest} represents the nearest neighbor's case, s_{nearest} represents its solution, and case $c \in \text{CB}$ then c_{nearest} can be found through Equation 21 where the sim function is the Euclidean similarity from Equation 22. Once the nearest case is known, its solution can be looked up from the CB. Adaptation new cases is simple: after the revision stage, finished cases and their comments are sent through the same vectorizing, masking, and splitting process as before. The new input vector is added as a case and the new output vector is added as its associated solution.

$$c_{\text{nearest}} = \arg\max \text{sim}(c_{\text{new}}, c) \tag{21}$$

$$\text{euclidean similarity} = \frac{1}{\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}} \tag{22}$$

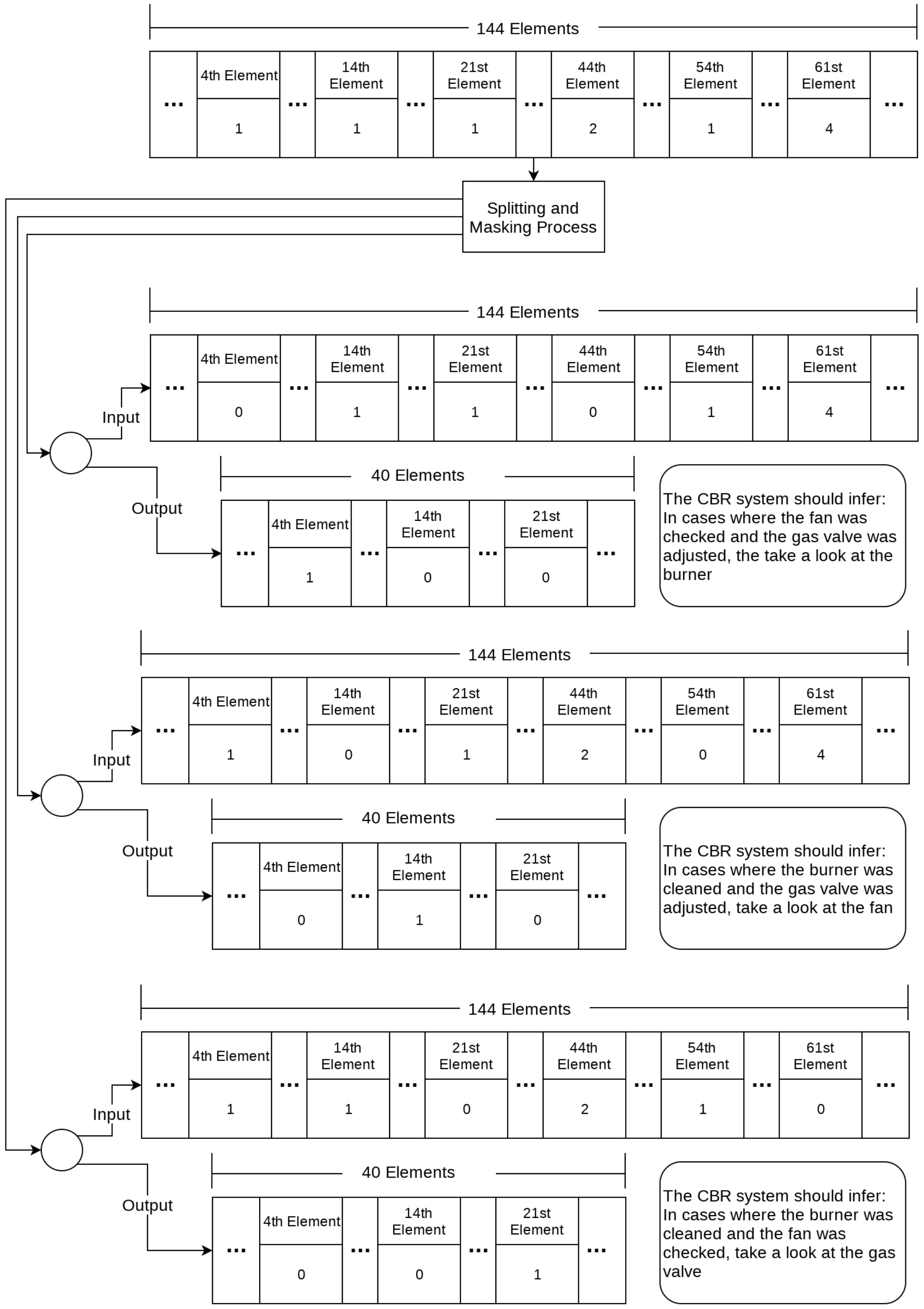


Figure 33: Input and Output Vector Splitting Example

The second is a substitution system using k-nearest neighbors search. Let us extract all of the solutions from the case base into a separate solution base (SB). In most cases, $|SB| < |CB|$, and it is guaranteed that $|SB| \leq |CB|$ as there is at most as many solutions as there are cases. In scenarios such as mechanical boiler repair, $|SB| << |CB|$, because many problems can be solved with the same solution. Let $s_i \in SB$ denote the ith possible solution seen in the solution base. If we have a function $I(s_j, i)$ that returns 1 where s_j is an example of the ith solution in the SB and otherwise returns 0, then we can find n_i, the count of the i solution seen, using Equation 23. The set of solutions, $\{s_1, s_2, \cdots s_k\}$ is made up of the solutions of the k closest cases found by using Equation 21. The nearest solution is then found with Equation 24. For these nearest-neighbor and k-nearest-neighbor methods, Figure 34 shows the full system diagram. Case adaptation is done in the same way as the nearest-neighbor CBR system.

$$n_1 = \sum_{j=1}^{k} I(s_j, i) \tag{23}$$

$$s_{\text{nearest}} = \arg\max n_i \tag{24}$$

The third CBR system in Figure 35 uses a feedforward artificial neural network (ANN or NN) as its retrieval method. Unlike the previous two substitution methods, it does not look up specific cases in its case base and return the most-matched solution. Instead, it makes inferences for a new cased based on cases that it has seen before and thus is a type of transformation CBR.

Like with the boiler prediction modeling, we must train the neural network model using an optimization algorithm. We again use the popular Adam optimization method [43], since it is known for its effectiveness when training neural networks to operate on sparse data like our combination of multi-hot and word vectors.

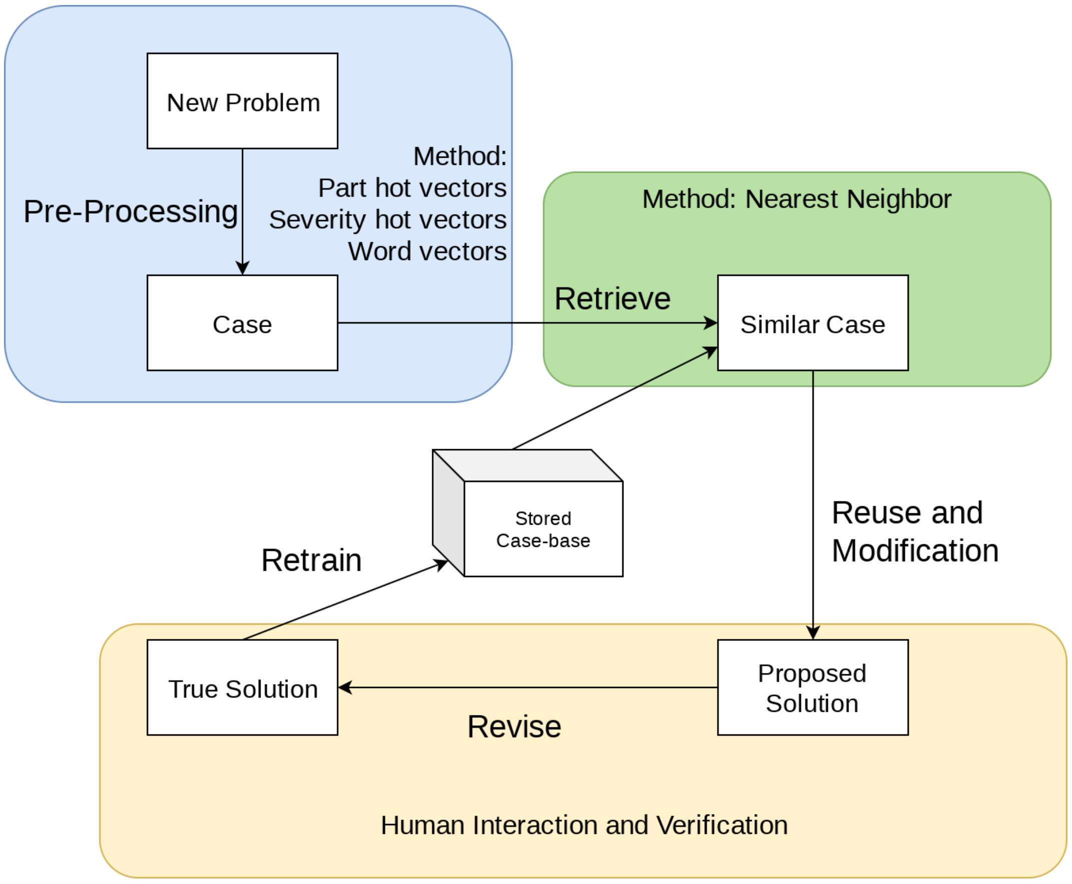


Figure 34: First and Second CBR Methods with Substitution

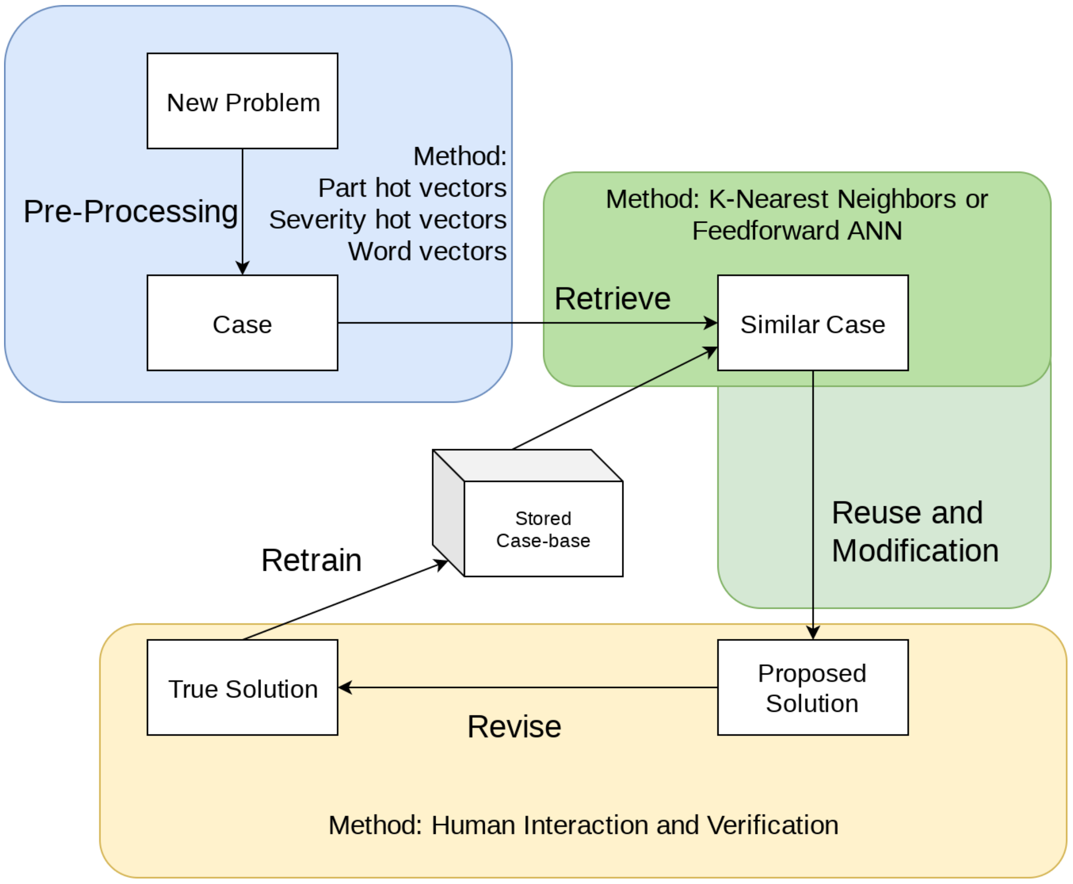


Figure 35: Third CBR Method with Transformation

Regardless of the specific retrieval and adaptation model used, the CBR system emulates multi-step solutions by treating each step as a ”new” case. For example, if the gas pressure switch is tripping and the entered case is that both the air pressure switch and gas pressure switches have been checked, it might suggest cleaning the gas valve. If this doesn’t fix the problem, a new run through the CBR system is performed, this time with checked air and gas pressure switches and a cleaned gas valve, and it may suggest adjusting the gas valve, which might also solve the problem. From the user’s perspective, the system stepped them through multiple steps in a generative approach, but in reality a generative approach wasn’t used because the CBR system is not able to determine which step will be the final one. It keeps suggesting its next-best solution based on previous cases and uses the other tiers as fall-backs if it cannot provide a solution.

IV RESULTS AND DISCUSSION

1 Clustering

In order to put the clustering and prediction results in context, let's first start with a very simple prediction model: for each desired output, the predicted output for the next day for every boiler is the average of that value from the current day. It is not expected to be accurate, but it can be used as a reference to show the improvements due to clustering. First, Figure 36 shows the MAE of the neural network model (without training any submodels) and the average repeat method referred to as "Base Repeat." The RMSE, and all of the RMSE plots, are in APPENDIX C. Like all of the accuracy statistics, these are calculated over the period from April 1st, 2018 to June 30th, 2018.

In general, the NN model performs better than the naive repeating model. However, it does perform worse in the areas of fan speeds, flue temperatures, inlet temperatures, and outlet temperatures. This indicates that the average fan speeds and temperatures outside of the hot water generator temperature are stable day-to-day. Let's keep the accuracy of this naive model in mind with the next comparison with clusters added.

Next, Figure 37 and Figure 54 show the MAE and RMSE respectively of the repeat model when using the various clustering methods against the base, unclustered NN model. We see similar trends from before: the NN model is better at predicting outputs except for fan speeds, flue temperatures, inlet temperatures, and outlet temperatures. Most clustering methods also show a small improvement in their MAE and RMSE for most of the clustering methods. Hot water generator temperature improves noticeably across all clustering methods. Pump output also improved considerably. In most cases, the system pump information (Pump Input, Pump Output, and Pump Status) are more accurate with a lower-number of clusters and sequential updating than methods that create a higher-number of clusters, but all clustering methods were more accurate

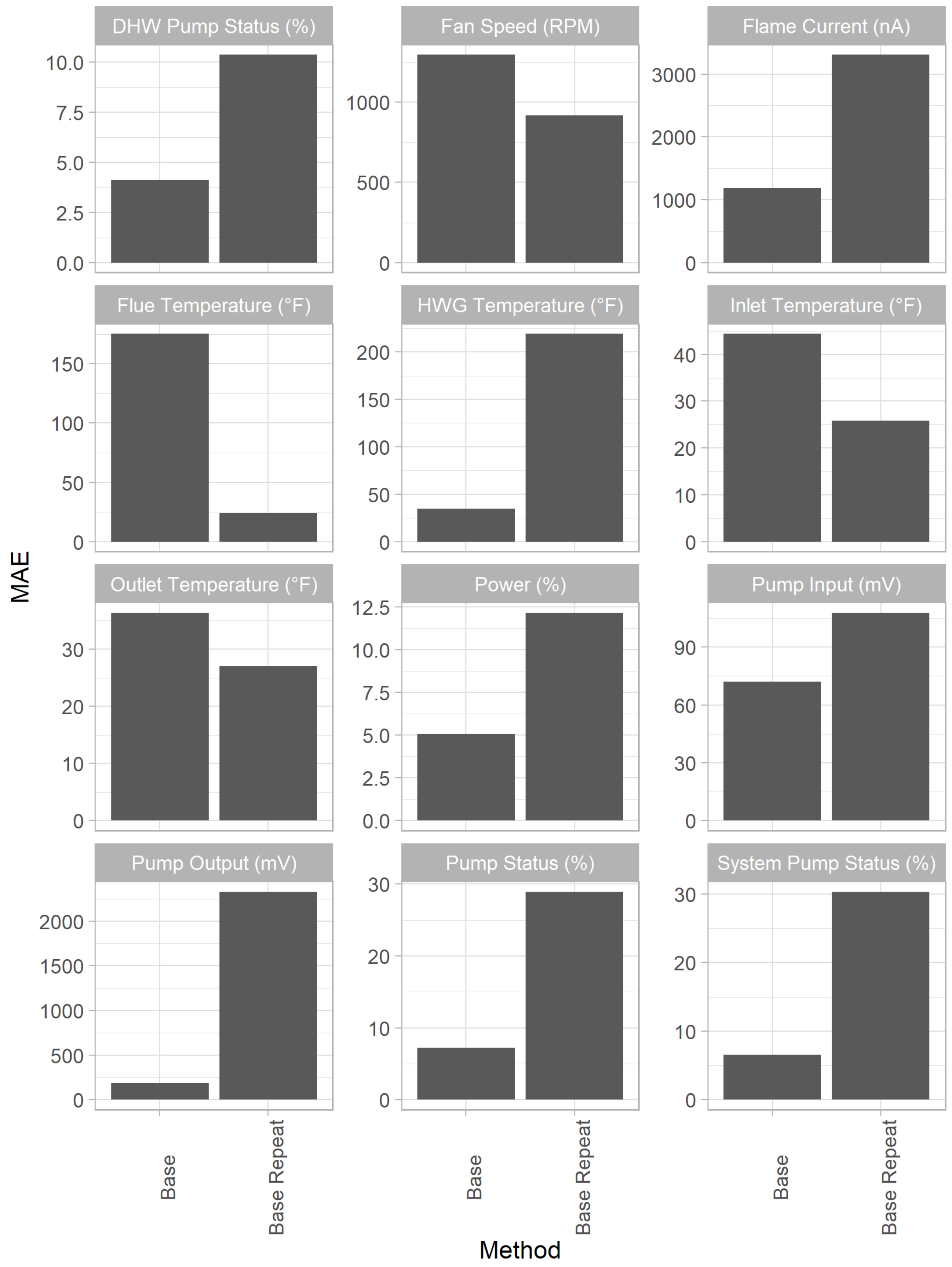


Figure 36: MAE of Base NN and Base Repeat Prediction Models

than the non-clustering repeat method. This shows that, at least for the naive method, moving of the boilers with the sequential updating method is oftentimes being of pump information, but this only helps up to a certain point. Also, the temperature predictions are slightly more accurate with the cohesion method than the sequential method, whereas sequential updating works better for the pump information. This shows that for the naive repeating method, stable clusters show better temperature prediction.

Percent improvements of the clustered repeat model is shown in Figure 38. While based on the same data, this view helps to highlight differences. For no parameters did clustering decrease accuracy. This indicates that all of the methods create clusters in reasonable ways. It is clearly seen that ROC and DBSCAN do not perform as well as K-means and Ward across the board. Outside of some examples in temperature prediction, the improvements when the sequential method are used are greater than when the cohesion method is used. This demonstrates that, at least for the naive method, putting a boiler with its closest cluster center is better for grouping boilers.

Let's now look at the MAE of the clustered models in Figure 39. With clustering and the neural network submodel method, all methods improve over the general neural network model. Seven of the parameters - fan speed, flue temperature, hot water generator temperature, inlet temperature, outlet temperature, pump input voltage, and pump output voltage - all show differing MAEs depending on the model. Interestingly, DYNC appears to help the predictive ability of most clustering models when the NN model is used. This is the opposite of the simple repeat prediction, where sequential updating was usually better. This suggests that there is value to the neural network model for boilers to remain in their clusters unless the cluster itself is removed. This would be expected if the submodels were saved between each simulated day since each NN would become increasingly familiar with its cluster over time (if catastrophic forgetting could also be prevented). They are not, though, and each submodel is recreated from the base NN for each day.

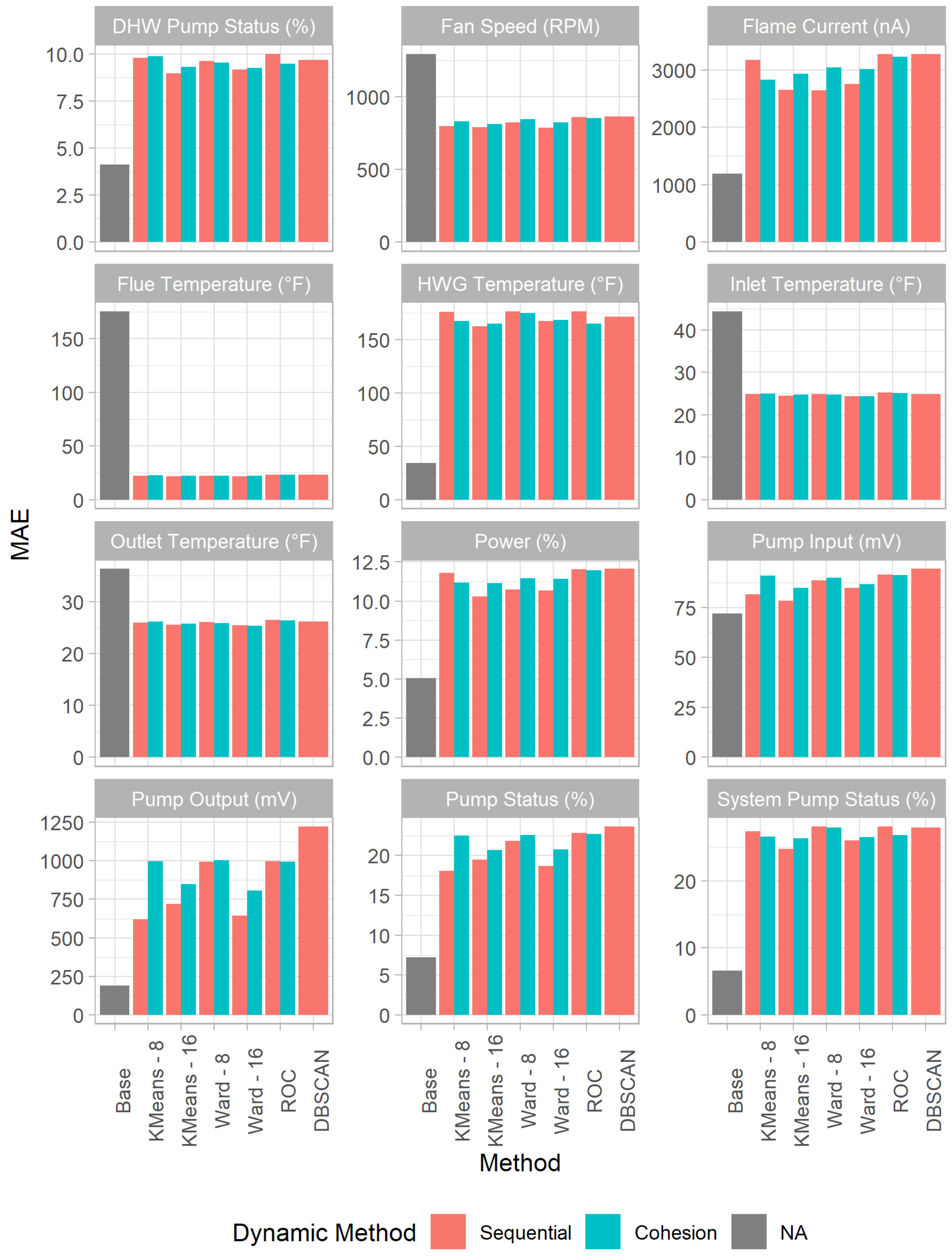


Figure 37: MAE of Base NN and Clustered Repeat Prediction Models

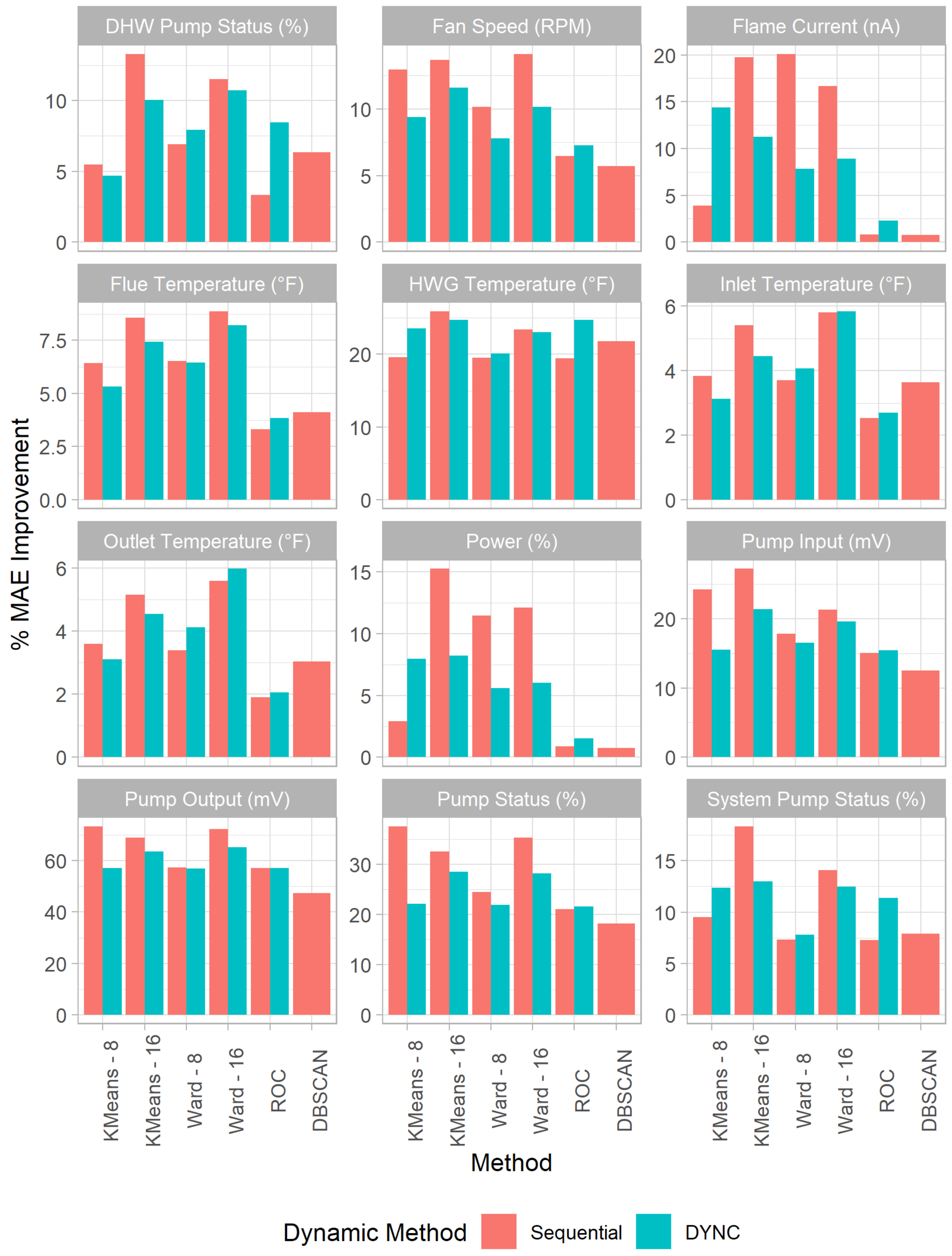


Figure 38: MAE Improvement of Clustered Repeat Prediction Models over Base Repeat Model

Also, the differences in MAE between clustering methods is larger than the differences in MAE between update methods using the same clustering method. There is room for further exploration here, but it is possible that the differences seen are mostly due to chance.

Unlike the base repeat clustered models, where lower-numbers of clusters with K-means and Ward's method performed well, Ward with 8 clusters and ROC (outside of pump output) have the best performance with the NN models. We already showed that Ward's method produces a well-distributed hierarchy of boilers in Figure 18, and that ROC, through its RBF kernel, is able to pick up on closeness of boilers in a higher dimension ($\mathbf{R}^{247}$) than the other methods in $\mathbf{R}^{246}$[1]. DBSCAN is used to identify non-hyperspherical clusters, so both of the update methods that use Euclidean distance measures are not able to effectively keep the clusters updated in a meaningful way.

The flatness of MAE of the other five parameters - DHW pump status, flame current, power, pump status, and system pump status is apparent. However, clustering does still help these compared to the base NN accuracy. This flatness does not exist in the simple repeat method, although the clustering NN model has a lower MAE in all of these categories than the repeat method as shown in Figure 37. Because we're using the same clusters, just different predictive models, this suggests that we have hit some floor in the accuracy of the NN system as it is currently designed. If this floor were removed by using a different type or shape of NN, or a different predictive model entirely, we would expect to see more than slight differences between the clustering and update methods.

Figure 41 shows a similarity between each combination of clustering method. If $(\mathbf{p}_i, \mathbf{p}_j)$ are points that are in the same cluster as each other in clustering method A and they are also in the same cluster as each other in clustering method B, then they are considered to be a similar pair.

[1] 239 parameters + 2 weather parameters + 5 cluster identifiers = 246 parameters

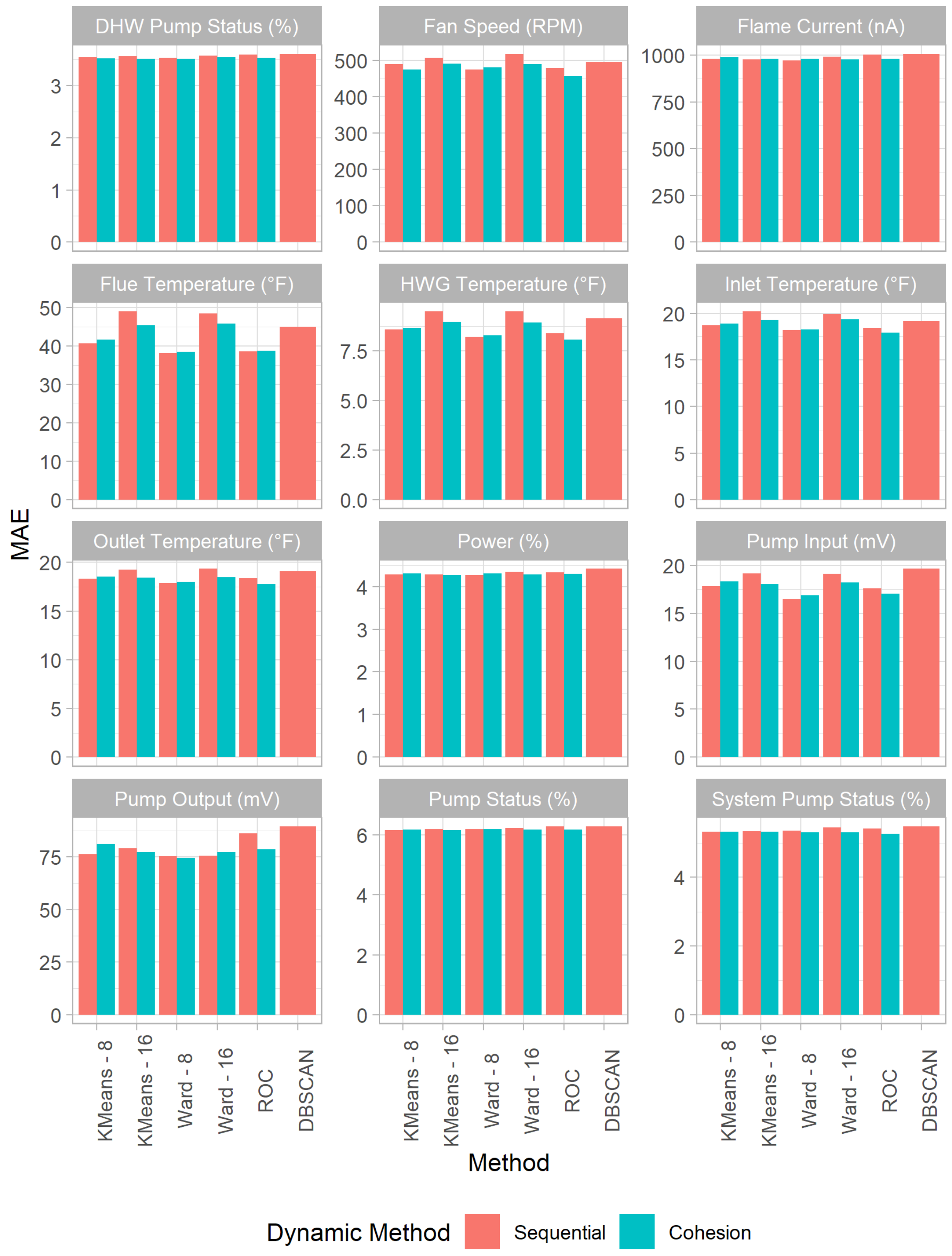


Figure 39: MAE of Clustered NN Models

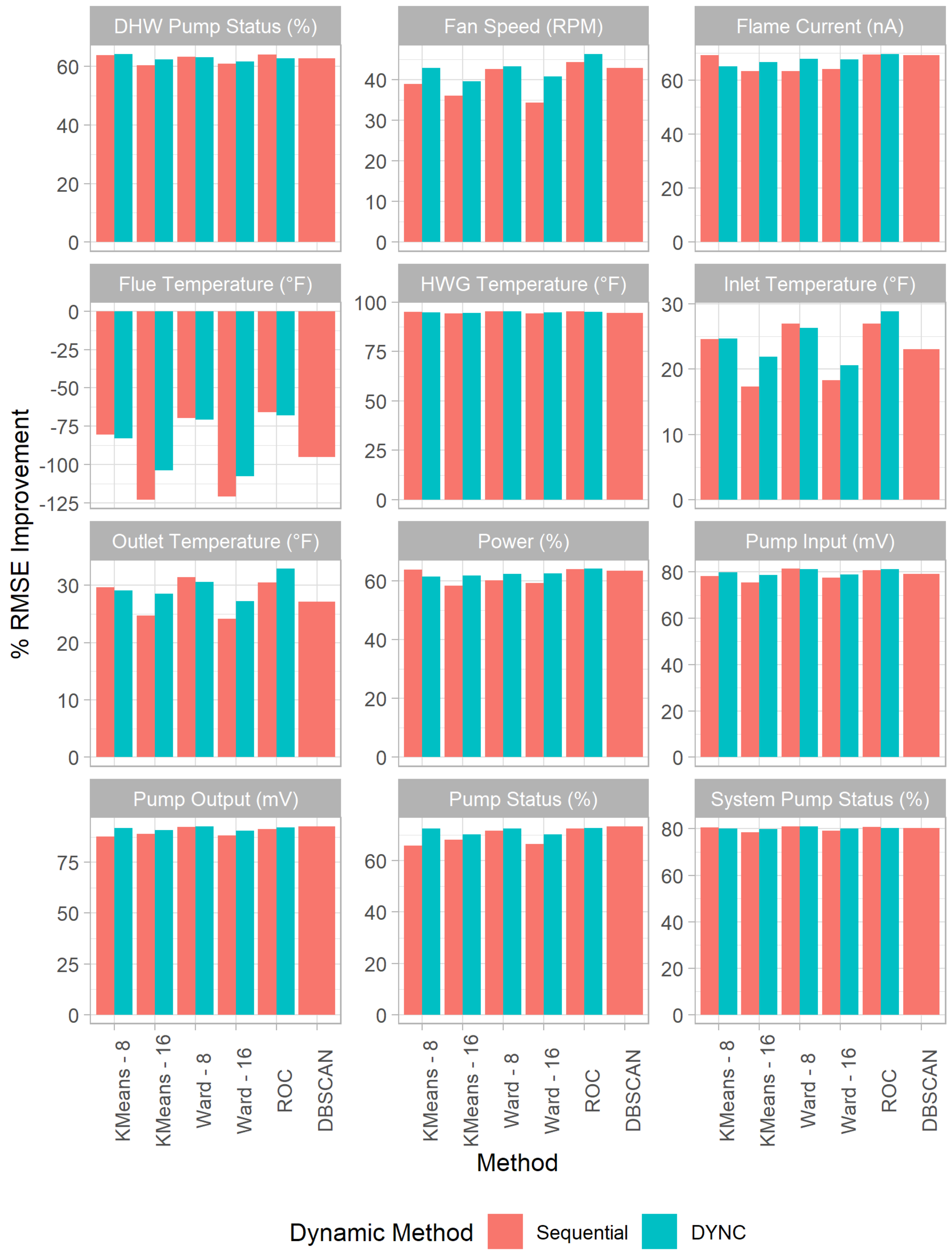
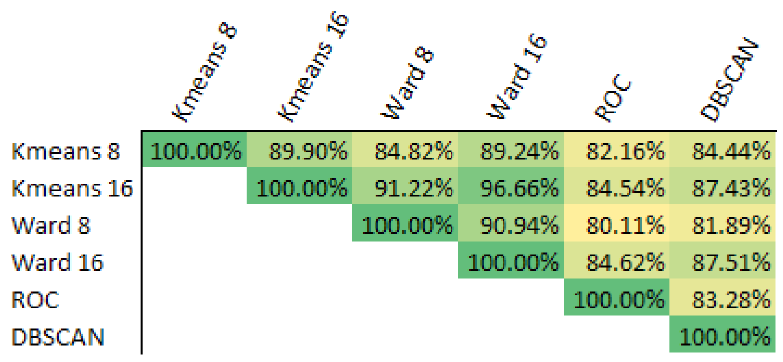


Figure 40: MAE Improvement of Clustered NN Models over Clustered Repeat Models



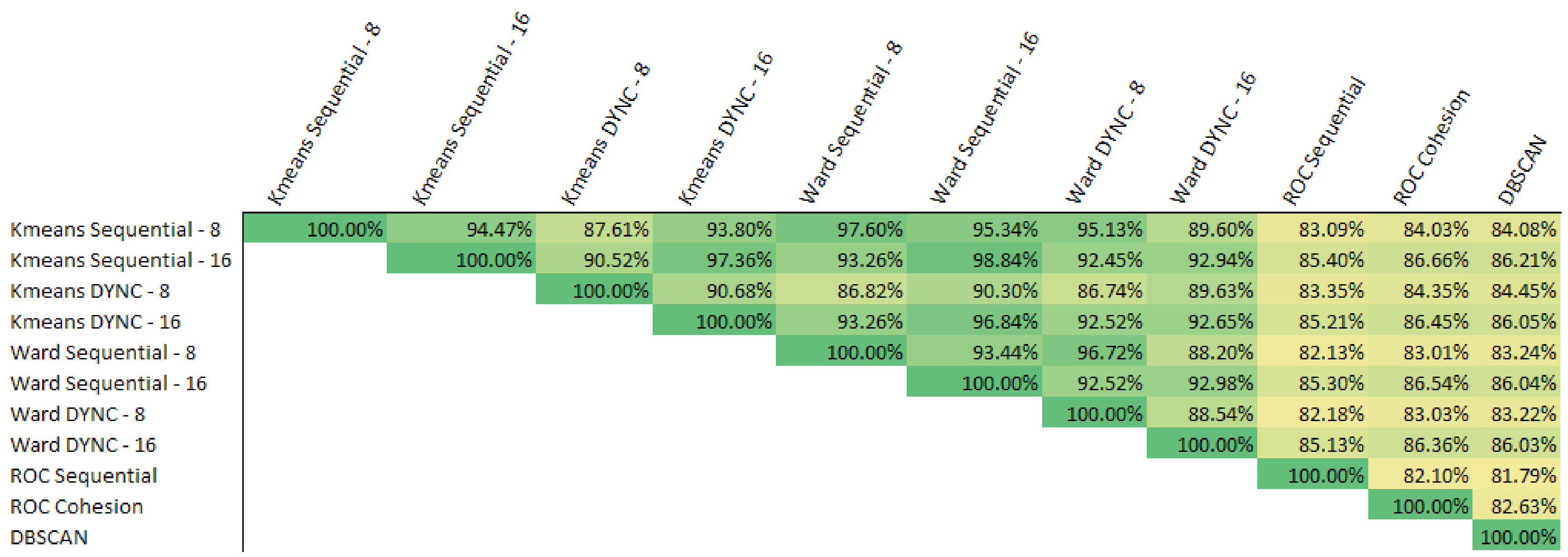
	Kmeans 8	Kmeans 16	Ward 8	Ward 16	ROC	DBSCAN
Kmeans 8	100.00%	89.90%	84.82%	89.24%	82.16%	84.44%
Kmeans 16		100.00%	91.22%	96.66%	84.54%	87.43%
Ward 8			100.00%	90.94%	80.11%	81.89%
Ward 16				100.00%	84.62%	87.51%
ROC					100.00%	83.28%
DBSCAN						100.00%

Figure 41: Similarity of Clustering Methods on April 01, 2018

The similarity is the rate of similar pairs to all possible pairs. At the start, there are no differences between the clustering methods because they have not been used yet. The cluster differences in Figure 42 from the end of the testing period does distinguish between the updating methods. The high similarity between clustering methods provides an alternative explanation for the flatness of some of the predictions. The clusters are, for the most part, similar between each cluster, and the differences could be significantly determined by the parameters that show high variability in prediction MAE. A detractor from this possibility is that such an explanation should also hold for the MAE of the clustered repeat method.

2 Case-Based Reasoning

For the probability and comment-based model, we compare accuracies of specific implementations using two metrics. The first is the log-loss metric introduced in Equation 9. Outside of its usefulness as an objective function in neural network training, it is also a useful metric for scoring probabilistic models, with a lower score being better. In many cases where log-loss is used, the exact order, or rank, of the results is not crucial. However, in a system that recommends next steps to take to solve a problem, the rank does matter. To score a model's ability to suggest the correct action as the first suggestion (or at least near the top of the suggestions list), a metric called *mean*



	Kmeans Sequential - 8	Kmeans Sequential - 16	Kmeans DYNC - 8	Kmeans DYNC - 16	Ward Sequential - 8	Ward Sequential - 16	Ward DYNC - 8	Ward DYNC - 16	ROC Sequential	ROC Cohesion	DBSCAN
Kmeans Sequential - 8	100.00%	94.47%	87.61%	93.80%	97.60%	95.34%	95.13%	89.60%	83.09%	84.03%	84.08%
Kmeans Sequential - 16		100.00%	90.52%	97.36%	93.26%	98.84%	92.45%	92.94%	85.40%	86.66%	86.21%
Kmeans DYNC - 8			100.00%	90.68%	86.82%	90.30%	86.74%	89.63%	83.35%	84.35%	84.45%
Kmeans DYNC - 16				100.00%	93.26%	96.84%	92.52%	92.65%	85.21%	86.45%	86.05%
Ward Sequential - 8					100.00%	93.44%	96.72%	88.20%	82.13%	83.01%	83.24%
Ward Sequential - 16						100.00%	92.52%	92.98%	85.30%	86.54%	86.04%
Ward DYNC - 8							100.00%	88.54%	82.18%	83.03%	83.22%
Ward DYNC - 16								100.00%	85.13%	86.36%	86.03%
ROC Sequential									100.00%	82.10%	81.79%
ROC Cohesion										100.00%	82.63%
DBSCAN											100.00%

Figure 42: Similarity of Clustering Methods on June 30th, 2018

squared ranking error (MSRE) is used. It is defined in Equation 25, where N is the number of cases, M is the number of possible classes, and r_c is the 0-indexed guessed rank of case c.

$$\frac{1}{N}\sum_{c}^{N}\left(1-\frac{r_c}{M}\right)^2 \tag{25}$$

As a single-case example, let us say that we know that the correct answer to case c is solution s out of 10 possible solutions. If a model, upon taking c as an input, predicts that s is the most likely solution and gives it rank 0, its scored MSRE will be 1. If another model, given the same c, predicts that s is the third most likely solution and gives it a rank of 2, its MSRE is $\left(1-\frac{2}{10}\right)^2 = 0.64$. You might notice that this disproportionately punishes errors in ranking near the top because it has a high slope in the area of the top ranks. This is intentional, because such a metric is necessary to separate good models from great ones.

A useful model will have a low log-loss score and a high MSRE score. A model with poor log-loss and good squared ranking error might be good for the most common of scenarios, but if none of the top suggestions lead anywhere then the rest of the probabilities won't be useful for determining the next most likely solution. A model with good log-loss and poor squared ranking error is good at generally guessing the likelihood of a symptom, solution, but is bad when it comes to exactly ranking them. A comparison across the various corpa, vector sizes, vector extraction methods, and each prediction category is shown in Figure 43 through Figure 50.

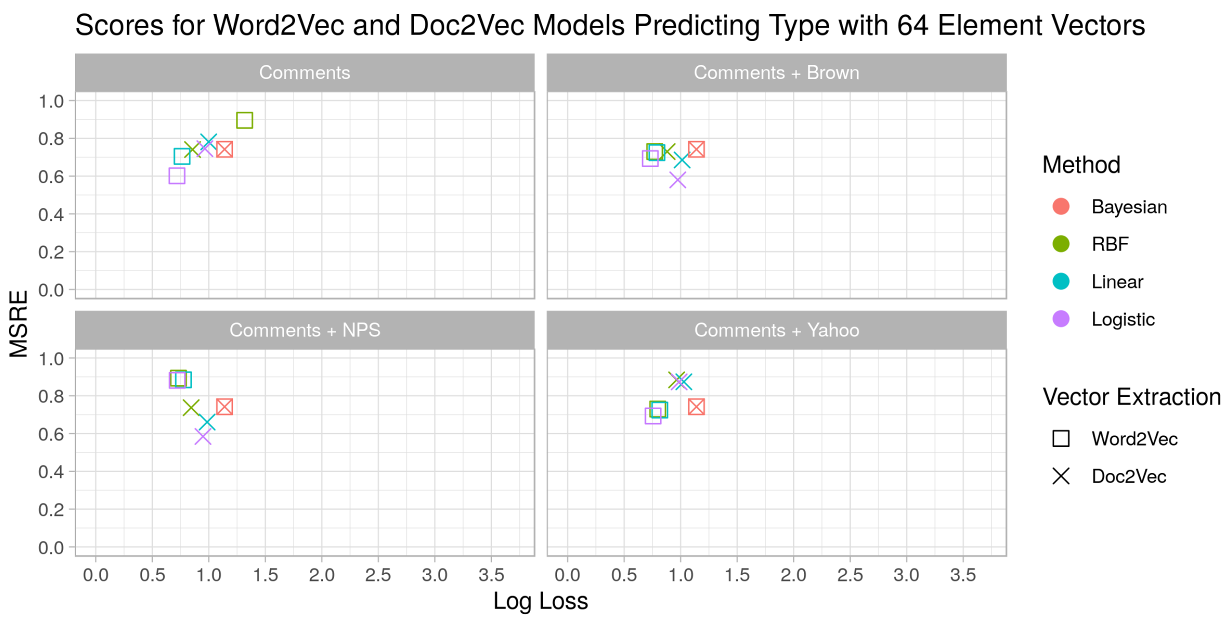


Figure 43: Model scores when predicting symptom types with 64 element vectors

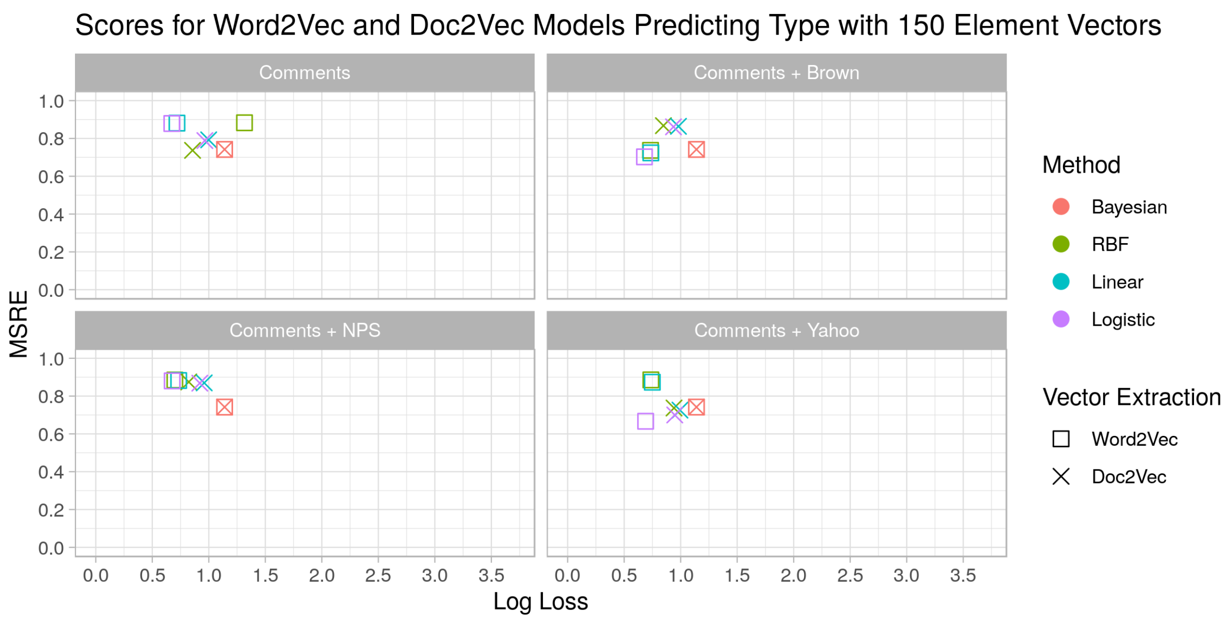


Figure 44: Model scores when predicting symptom types with 150 element vectors

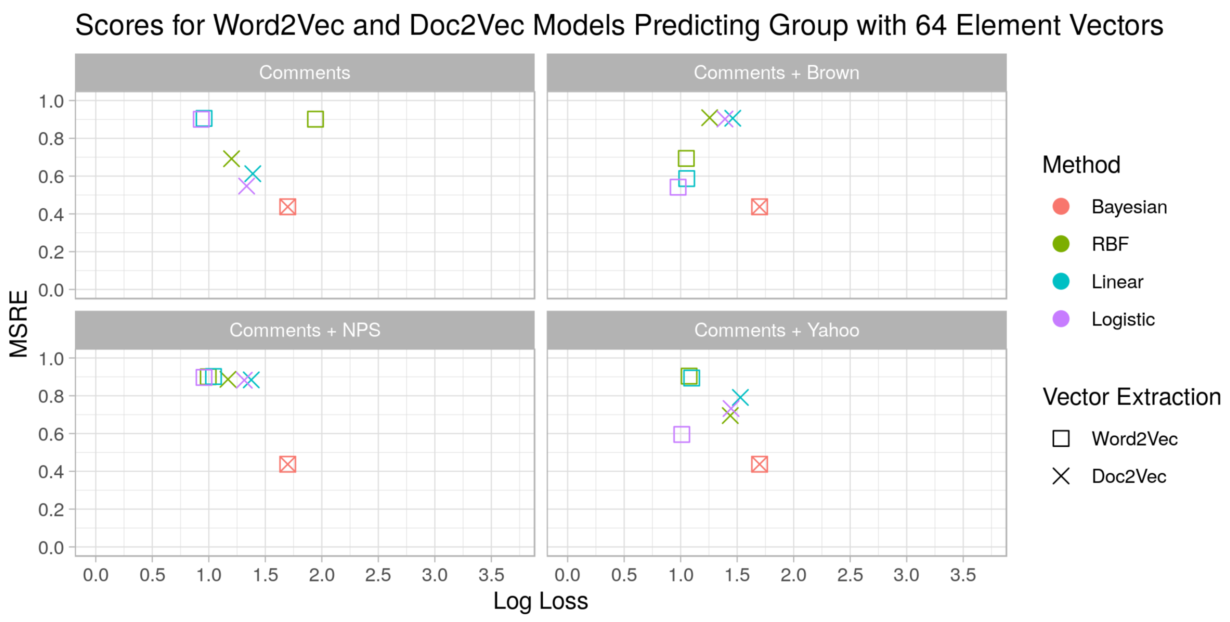


Figure 45: Model scores when predicting symptom groups with 64 element vectors

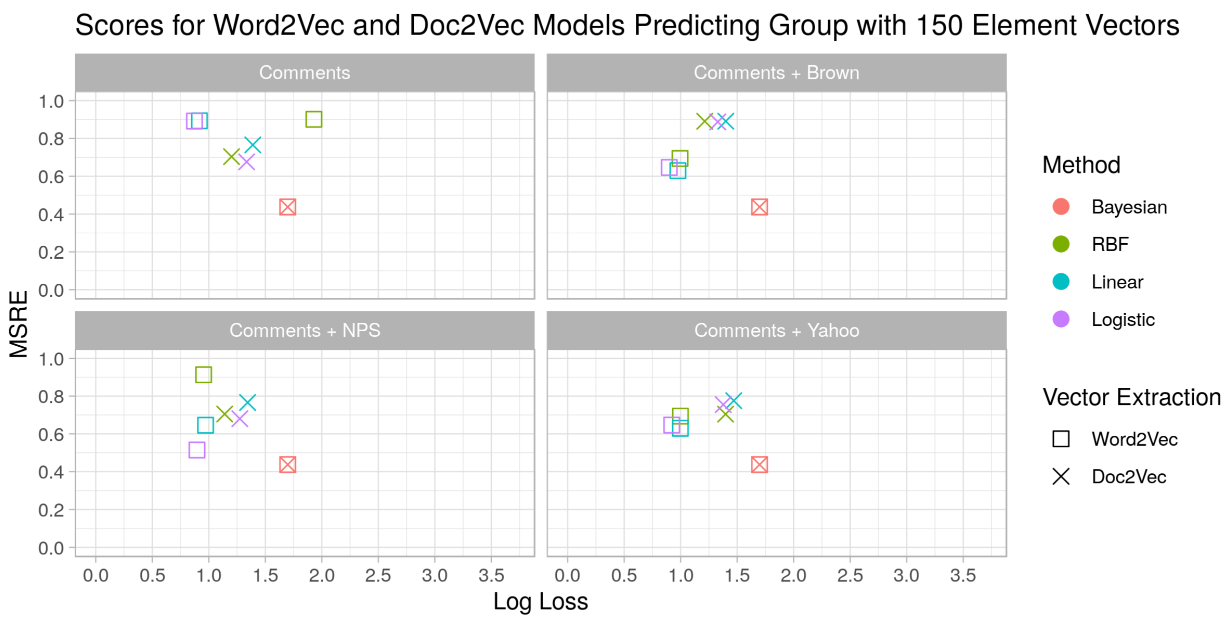


Figure 46: Model scores when predicting symptom groups with 150 element vectors

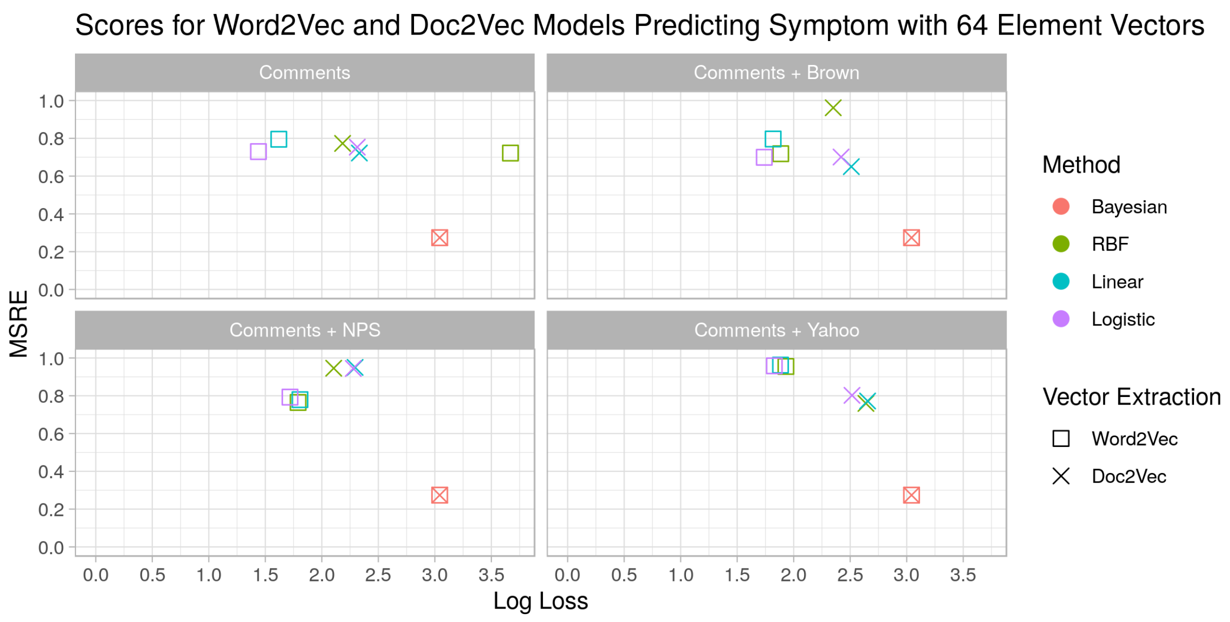


Figure 47: Model scores when predicting symptoms with 64 element vectors

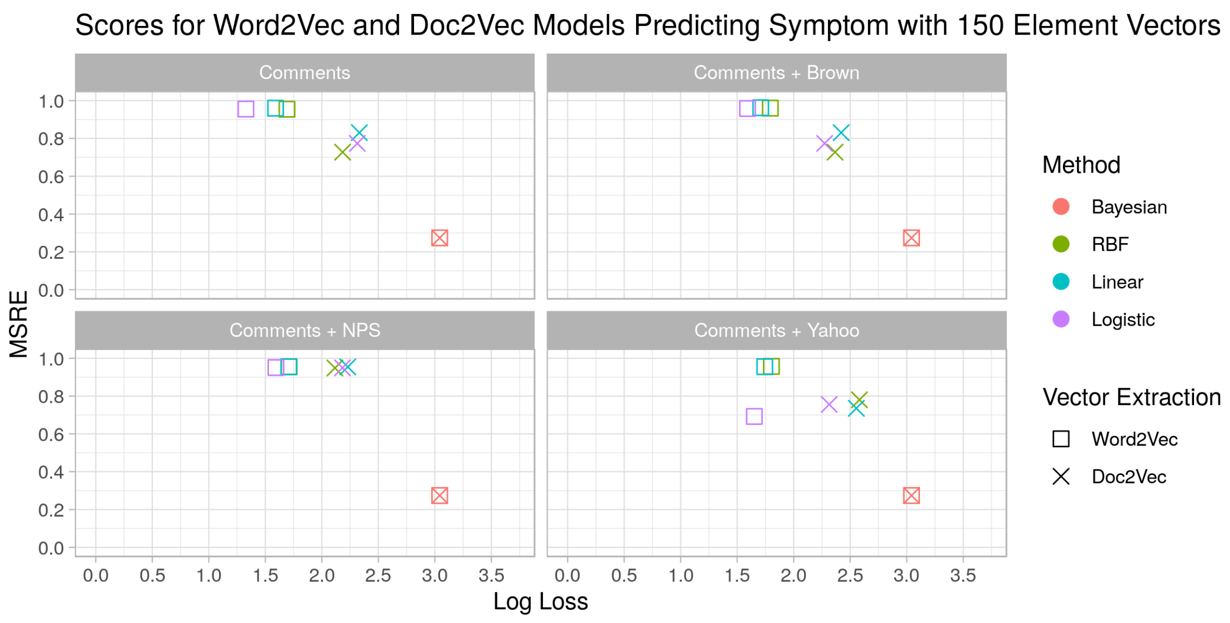


Figure 48: Model scores when predicting symptoms with 150 element vectors

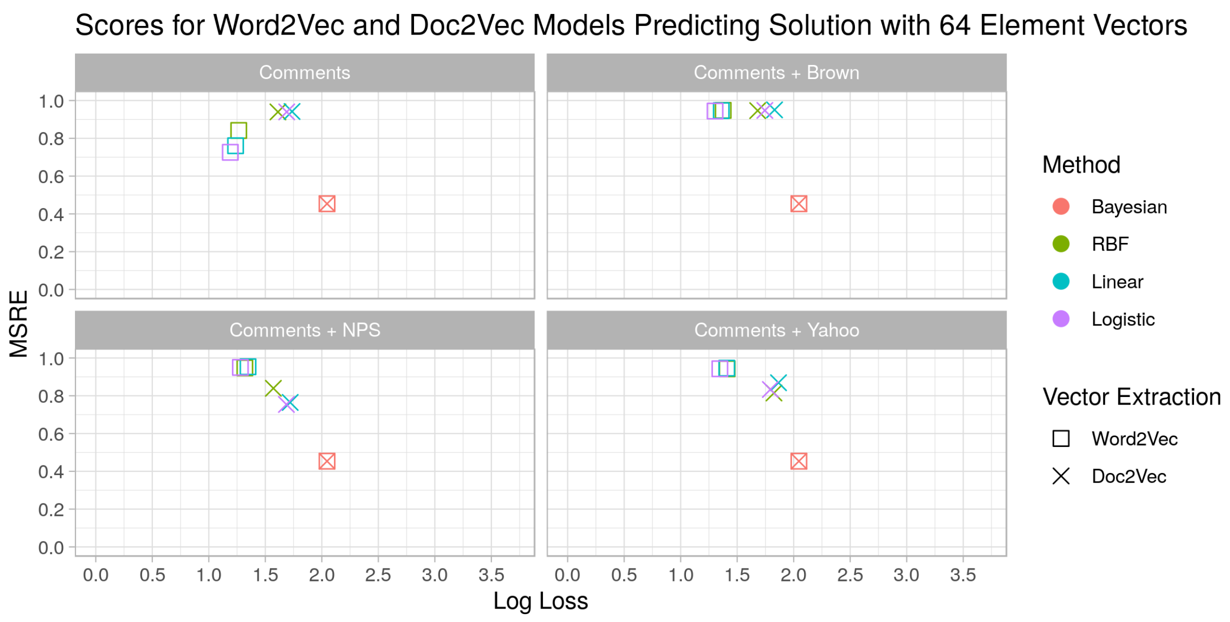


Figure 49: Model scores when predicting solutions with 64 element vectors

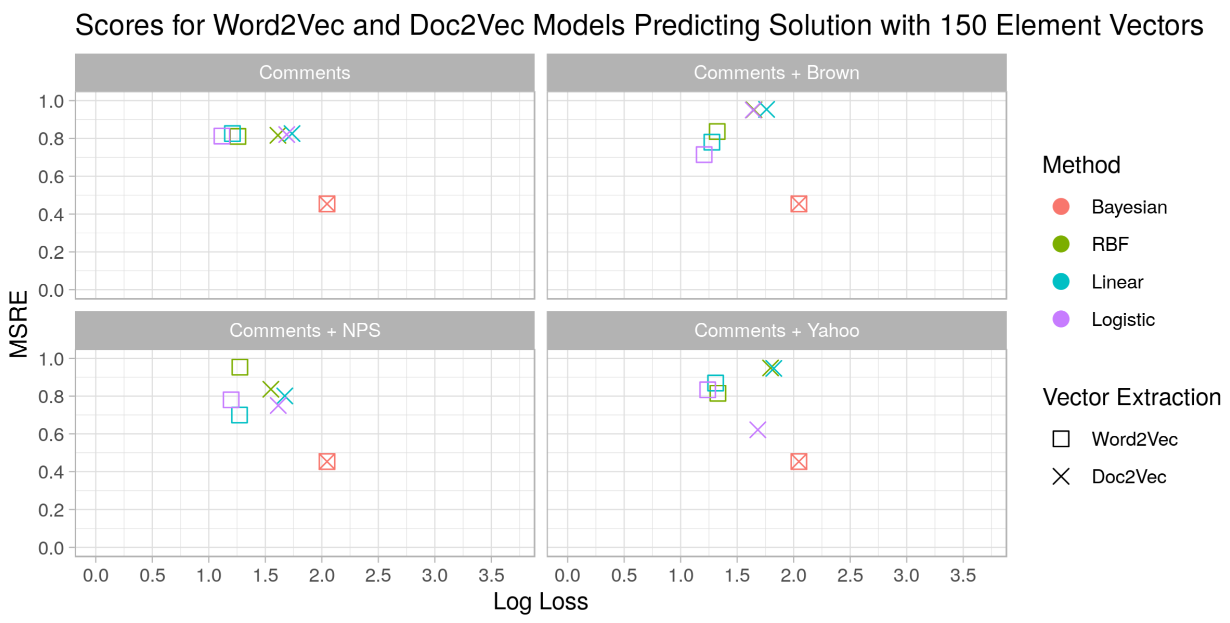


Figure 50: Model scores when predicting solutions with 150 element vectors

Scoring of the three CBR models is done a little differently. Instead of relying on the pre-existing type, group, symptom, and solution categories, it generates part-based suggestions. To get a sense of its accuracy, we do a 5-fold cross validation. We split each of the cases into one of 5 evenly-split groups, labeled 1 through 5. We take the first 4 groups, hold back the last, and pre-process the cases into their case and solution vectors. For the nearest-neighbor and k-nearest-neighbor systems, we then add these vectors to the case base. For the ANN model, we train the model using the case and solution vectors. Next, we pre-process the held-out group. The case vector (no solution vector) is passed through the CBR system. The output vectors from the model are compared against their associated known solution vectors. We grade the accuracy by calculating the true positive rate, true negative rate, false positive rate (Type 1 errors) and false negative rate (Type 2 errors). The entire process is repeated 4 more times so that each of the 5 *folds* are held out once. The positive, negative, and error rates are then averaged across the folds to calculate their overall scores. Figure 51 show the true positive and true negative error rates of each tested system, while Figure 52 shows the Type I and Type II error rates.

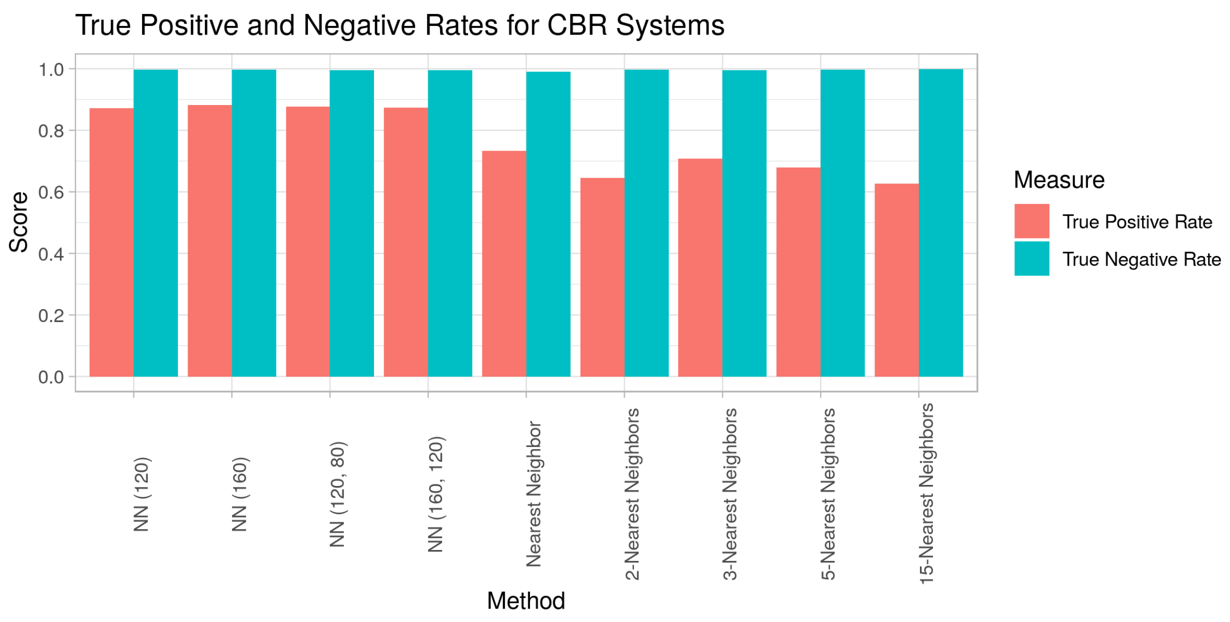


Figure 51: Model scores when predicting solutions with 64 element vectors

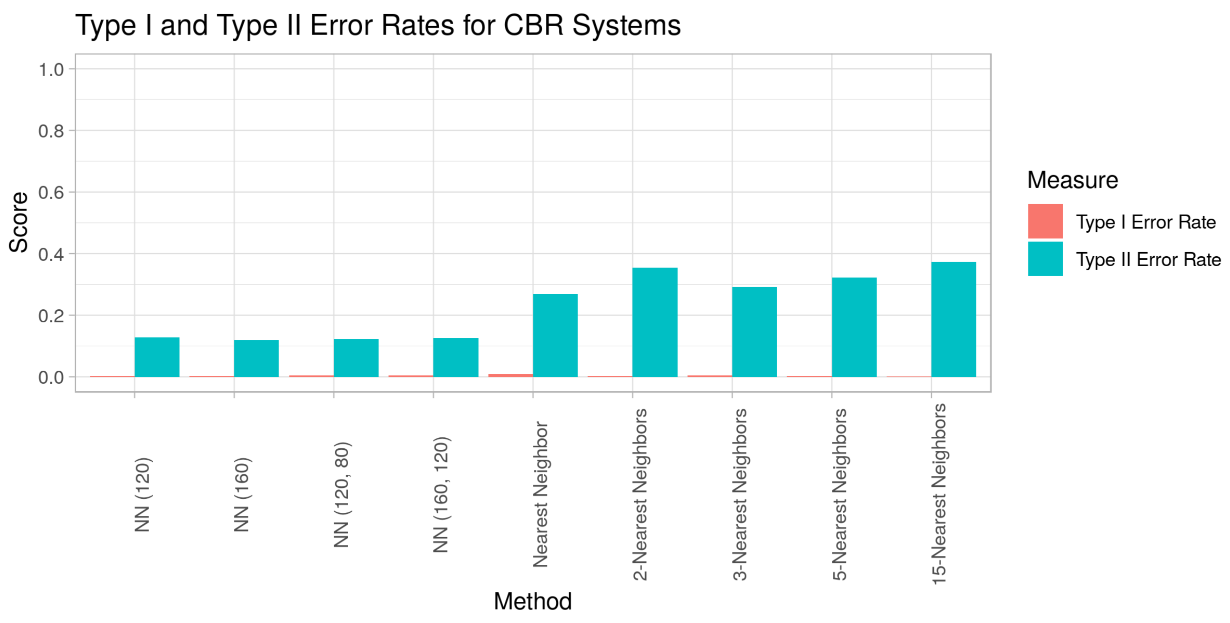


Figure 52: Model scores when predicting solutions with 150 element vectors

V CONCLUSION

1 Clustering

We have created a system for handling many boiler parameters in unevenly-spaced time series and processing them into a way that is useful for clustering with a reasonable amount of computational effort. In addition to the direct boiler parameters, we include additional information about the model family and weather data pulled from the NCDC to add additional clustering features. A framework for comparing K-means, agglomerative clustering methods, ROC, and DBSCAN was developed. This included the ability to test multiple cluster update methods as well.

We checked the validity and usefulness of the clusters by predicting next-day sensor readings through two predictive models: a naive method that simply repeats the average of the previously-seen parameters in each cluster, and a neural-network based system that is trained on the full dataset and further enhanced by the specific clusters is used.

In both the naive and NN-based models, each of the clustering methods used creates predictive models that are more accurate than comparable non-clustered methods. K-means and Ward's method generally show the best performance improvements for the naive models, while Ward's method and ROC generally show the best improvements when the NN models are used. In all parameters except flue temperature, the clustered NN model outperforms the clustered naive model.

When the naive prediction model is used, clusters updated with a modified sequential K-means algorithm noticeably outperforms the cohesion-based method, DYNC. However, when the NN model is used, DYNC slightly exceeds the sequential method performance for most parameters.

Further work can be done to understand the accuracy floor that the NN model hit. There is an opportunity to further revise the DYNC algorithm - in its current implementation, it considers points in Euclidean space, even if the original clusters were not generated in it. A kernel-based cohesion calculation could potentially be used in place of the current Euclidean distance calculation. Additionally, it currently has no way of adding new clusters once they have been removed. The timescale that the tests in this thesis were run on were not long enough to significantly reduce the number of clusters, but over long enough time scales it could cause problems. Although it was not seen in 2D tests, there is also a potential for runaway cluster removal when it is applied to the boiler data. This was alleviated with a slight modification by changing the cohesion threshold, but in a future revision this cohesion threshold could be adapted automatically.

2 Case-Based Reasoning

A multi-tiered case-based reasoning system was developed to assist with boiler repair and issues. At the primary level, users enter comments and, if available, tell the system which parts have already been looked at and what was action was done with them. The model then tells the user what it believes is the next best step if there is still a problem with the boiler. If it predicts that no change is necessary based on what was seen in previous cases but there is still a problem, the secondary system uses the entered case comments to suggest possible symptoms to look for and which solutions are the most likely. The third tier is a fall-back probability model that simply states the probability of various solutions being correct given a symptom type, symptom group, symptom, or a combination of all three.

For the first tier, a one-hidden-layer neural network with 120 neurons in its hidden layer proved to be the most accurate at case retrieval and adaptation. Somewhat counter-intuitively, outside of the NN models, nearest-neighbor retrieval was the second best method, with K-nearest neighbor performing worse than simply taking the most similar case that was previously seen. In the second tier, we use log-loss and a metric called mean squared ranking error (MSRE) to evaluate accuracy.

MSRE is introduced because it is not just important that the approximate probability of a symptom or solution is correct, but it is also important that the predicted ranks of the classes in the categories is correct. For the repair scenario, it is more useful that the correct solution is predicted at 30% probability and be the highest-ranked solution than it be predicted at 35% but have a wrong solution predicted at 40%. Log-loss would score both scenarios similarly, but MSRE punishes the second scenario. We show that a logistic regression model based on 150-element Word2Vec vectors generally produces the most accurate symptom type, symptom group, symptom, and solution predictions for both metrics.

While the individual components of the multi-tiered CBR system have been evaluated for accuracy using cross-validation, the system has not been evaluated as a whole set against real problems. Future work on this topic would involve creating a GUI front-end for accessing the CBR system and allowing real technicians to use it.

REFERENCES

[1] The American Society of Mechanical Engineers. 2010 ASME Boiler and Pressure Vessel Code: Recommended Guidelines for the Care of Power Boilers. Technical report, The American Society of Mechanical Engineers, New York, New York, 2011.

[2] Pavel Berkhin. A Survey of Clustering Data Mining Techniques. In *Grouping Multidimensional Data*, pages 25–71. Springer, Berlin, 2006.

[3] A.M. Fahim, G. Saake, A.M. Salem, F.A. Torkey, and M.A Ramadan. K-Means for Spherical Clusters with Large Variance in Sizes. *International Journal of Computer and Information Engineering*, 2(9):2923–2928, 2008.

[4] J.A. Hartigan and M.A Wong. Algorithm AS 136: A K-Means Clustering Algorithm. *Journal of the Royal Statistical Society, Series C (Applied Statistics)*, 28(1):100–108, 1979.

[5] Pratik Worah and Sandeep Sen. A linear time deterministic algorithm to find a small subset that approximates the centroid. *Information Processing Letters*, 105(1):17–19, 2007.

[6] Nuwan Ganganath, Chi Tsun Cheng, and Chi K. Tse. Data clustering with cluster size constraints using a modified k-means algorithm. *Proceedings - 2014 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2014*, (1):158–161, 2014.

[7] Yasunori Endo and H. Iwata. On Dynamic Clustering and Two Options. *The 14th IEEE International Conference on Fuzzy Systems, 2005. FUZZ '05.*, pages 996–1001, 2005.

[8] Joe H. Ward. Hierarchical Grouping to Optimize and Objective Function. *Journal of the American Statistical Association*, (58):236–244, 1963.

[9] Wang Qiu-Dong. The global solution of the N-body problem. *Celestial Mechanics and Dynamical Astronomy*, 50(1):73–88, mar 1990.

[10] Martin Ester, Hans-Peter Kriegel, Sander Jorg, and Xiaowei Xu. A Density-Based Clustering Algorithms for Discovering Clusters. *KDD-96 Proceedings*, 96(34):226–231, 1996.

[11] Erich Schubert, Jörg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. *ACM Transactions on Database Systems*, 42(3), 2017.

[12] Junhao Gan and Yufei Tao. Dynamic Density Based Clustering. In *SIGMOD '17 Proceedings of the 2017 ACM International Conference on Management of Data*, pages 1493–1507, 2017.

[13] Janet Kolodner. An Introduction to Case-Based Reasoning. *Artificial Intelligence Review*, (6):3–34, 1992.

[14] Agnar Aamodt and Enric Plaza. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, 7(1):39–59, 1994.

[15] I. Watson. Case-based reasoning is a methodology not a technology. *Knowledge-Based Systems*, 12(5-6):303–308, 1999.

[16] Kerstin Bach, Odd Erik Gundersen, Christian Knappskog, and Pinar Ozturk. Automatic Case Capturing for Problematic Drilling Situations. In *22nd International Conference on Case-Based Reasoning Research and Development*, pages 48–62, Cork, Ireland, 2014. Springer.

[17] Kellen Gillespie, Kalyan Moy Gupta, and Michael Drinkwater. Case-Based Object Placement Planning. In *22nd International Conference on Case-Based Reasoning Research and Development*, pages 170–184, Cork, Ireland, 2014. Springer.

[18] Stewart Massie, Glenn Forbes, Susan Craw, Lucy Fraser, and Graeme Hamilton. FITsense: Employing Multi-modal Sensors in Smart Homes to Predict Falls. In *26th International Conference on Case-Based Reasoning Research and Development*, pages 249–263, Stockholm, Sweden, 2018. Springer.

[19] Yoke Yie Chen, Xavier Ferrer, Nirmalie Wiratunga, and Enric Plaza. Sentiment and Preference Guided Social Recommendation. In *22nd International Conference on Case-Based Reasoning Research and Development*, pages 79–94, Cork, Ireland, 2014. Springer.

[20] Yoke Yie Chen, Xavier Ferrer, Nirmalie Wiratunga, and Enric Plaza. Aspect Selection for Social Recommender Systems. In *23rd International Conference on Case-Based Reasoning Research and Development*, pages 60–72, Frankfurt, Germany, 2015. Springer.

[21] Flávio Ceci, Rosina O. Weber, Alexandre L. Gonçalves, and Roberto C. S. Pacheco. Adapting Sentiments with Context. In *23rd International Conference on Case-Based Reasoning Research and Development*, pages 44–59, Frankfurt, Germany, 2015. Springer.

[22] Sebastian Dieterle and Ralph Bergmann. A Hybrid CBR-ANN Approach to the Appraisal of Internet Domain Names. In *22nd International Conference on Case-Based Reasoning Research and Development*, page 95, Cork, Ireland, 2014. Springer.

[23] Santiago Ontañón, Enric Plaza, and Jichen Zhu. Argument-Based Case Revision in CBR for Story Generation. In *23rd International Conference on Case-Based Reasoning Research and Development*, pages 290–305, Frankfurt, Germany, 2015. Springer.

[24] Antonio Sanchez-Ruiz and Santiago Ontanon. Least Common Subsumer Trees for Plan Retrieval. In *22nd International Conference on Case-Based Reasoning Research and Development*, pages 405–419, Cork, Ireland, 2014. Springer.

[25] Ramon Lopez de Mantaras, David McSherry, Derek Bridge, David Leak, Barry Smyth, Susan Craw, Boi FAltings, May Lou Maher, Michael T Cox, Kenneth Forbus, Mark Keane, Agnar Aamodt, and Ian Watson. *Retrieval, reuse, revision, and retention*, volume 00. 2005.

[26] Zhiqiang Yu, Chun Hua Zhao, Dalin Zhu, and Mingsong Zhang. Research on case representation in printing machine fault diagnosis expert system based on case-based reasoning. *Proceedings of International Conference on Computer Science and Software Engineering, CSSE 2008*, 1:233–236, 2008.

[27] Ziyan Wen, Jacob Crossman, John Cardillo, and Yi L. Murphey. Case Base Reasoning in Vehicle Fault Diagnostics. *Proceedings of the International Joint Conference on Neural Networks*, 4:2679–2684, 2003.

[28] Anudeep Boilers. Anudeep Boilers: Hot Water Generator, 2019.

[29] Lochinvar. Hot Water Generator Systems, 2013.

[30] Ipstack. ipstack. https://ipstack.com/, 2019.

[31] NCDC. NCDC Climate Date Online. https://www.ncdc.noaa.gov/cdo-web/, 2019.

[32] Andreas Eckner. A framework for the analysis of unevenly spaced time series data. 2014.

[33] William Mendenhall and Terry Sincich. *A Second Course in Statistics Regression Analysis*. Pearson Education, Inc., Upper Saddle River, New Jersey, 6th edition, 2003.

[34] I.D. Guedalia, M. Londom, and M. Werman. An on-line agglomerative clustering method for non-stationary data 1 Introduction. *Neural Computation*, 11(2):521–540, 1998.

[35] Daoqiang Zhang, Songcan Chen, and Keren Tan. Improving the robustness of 'online agglomerative clustering method' based on kernel-induce distance measures. *Neural Processing Letters*, 21(1):45–51, 2005.

[36] Christopher D. Manning, Prabhakar Raghavan, and Henrich Schutze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[37] Fionn Murtagh and Pierre Legendre. Ward's Hierarchical Clustering Method: Clustering Criterion and Agglomerative Algorithm. (June):1–20, 2011.

[38] Mohammad Rezaei and Pasi Franti. Set matching measures for external cluster validity. *IEEE Transactions on Knowledge and Data Engineering*, 28(8):2173–2186, 2016.

[39] Richard O. Duda. Sequential k-Means Clustering. https://www.cs.princeton.edu/courses/archive/fall08/cos436/Duda/C/sk_means.htm, 2007.

[40] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J van der Walt, Matthew Brett, Joshua Wilson, K Jarrod Millman, Nikolay Mayorov, Andrew R.~J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, \.Ilhan Polat, Yu Feng, Eric W Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E.~A. Quintero, Charles R Harris, Anne M Archibald, Antônio H Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0–Fundamental Algorithms for Scientific Computing in Python. *arXiv e-prints*, page arXiv:1907.10121, jul 2019.

[41] Akshay L. Chandra. Perceptron: The Artificial Neuron (An Essential Upgrade To The McCulloch-Pitts Neuron), 2018.

[42] Michael A. Neilsen. *Neural Networks and Deep Learning*. Determination Press, San Francisco, CA, 2015.

[43] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference on Learning Representations*, pages 1–15, San Diego, CA, 2015. ICLR.

[44] Peter F. Brown, Peter V. DeSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18(4):467–479, 1992.

[45] Daniel Jurafsky and James H. Martin. *Speech and Language Processing*. Prentice Hall, Upper Saddle River, New Jersey, 2nd edition, 2008.

[46] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and their Compositionality. *NIPS '13 Proceedings of the 26th International Conference on Neural Information Processing Systems*, 2:111–3119, 2013.

[47] Quoc Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. 2014.

[48] Xin Rong. word2vec Parameter Learning Explained. 2014.

[49] Radim Hurek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, 2010. ELRA.

[50] NLTK Project. Natural Language Toolkit. https://www.nltk.org/, 2019.

[51] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[52] Chih Chung Chang and Chih Jen Lin. LIBSVM: A Library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–39, 2011.

[53] John C. Platt. Probabilistic Outputs for SVMs Comaprison to Regularized Likelihood Methods. 1999.

[54] Metin Turkay, Ozlem Yilmaz, and Fadime Uney Yuksektepe. Prediction of secondary structures of proteins using a two-stage method. *Computer Aided Chemical Engineering*, 21(C):1679–1685, 2006.

[55] Eric Kim. Everything You Wanted to Know about the Kernel Trick. https://www.eric-kim.net/eric-kim-net/ posts/1/kernel_trick.html, 2017.

[56] Galen Andrew and Jianfeng Gao. Scalable training of L1-regularized log-linear models. *ACM International Conference Proceeding Series*, 227:33–40, 2007.

[57] F Pedregosa, G Varoquaux, A Gramfort, V Michel, B Thirion, O Grisel, M Blondel, P Prettenhofer, R Weiss, V Dubourg, J Vanderplas, A Passos, D Cournapeau, M Brucher, M Perrot, and E Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

APPENDIX A - DATA FORMATS

Table 12: Full format of raw data from boilers

Item	Description	Used
`oem_id`	The OEM ID – all are 132d4043	No
`oem_model`	The board model – either Herald or Page	Yes
`dsn`	The boiler serial number	Yes
`property_name`	The name of the property	Yes
`display_name`	The property display name	No
`base_type`	The type of the property	Yes
`time_uuid`	A UUID for the property update time	No
`created_at_from_device`	Blank	No
`updated_at`	Time that the property is updated in the database	No
`created_at`	The time that the property is uploaded by the boiler	Yes
`user_uuid`	A UUID for the boiler	No
`echo`	False for roHistoryAverage, True for all else	No
`closed`	If the property is discontinued	No
`discarded`	If a property record is discarded	Yes
`scope`	User is currently the only choice	No
`val_int`	An integer value	Yes
`val_decimal`	A decimal value	Yes
`val_float`	A floating point value	Yes
`val_boolean`	A boolean value	Yes
`val_string`	A string value	Yes
`metadata`	Always e30=	No
`direction`	Whether a value is considered input or output	No

APPENDIX B - CLUSTERING

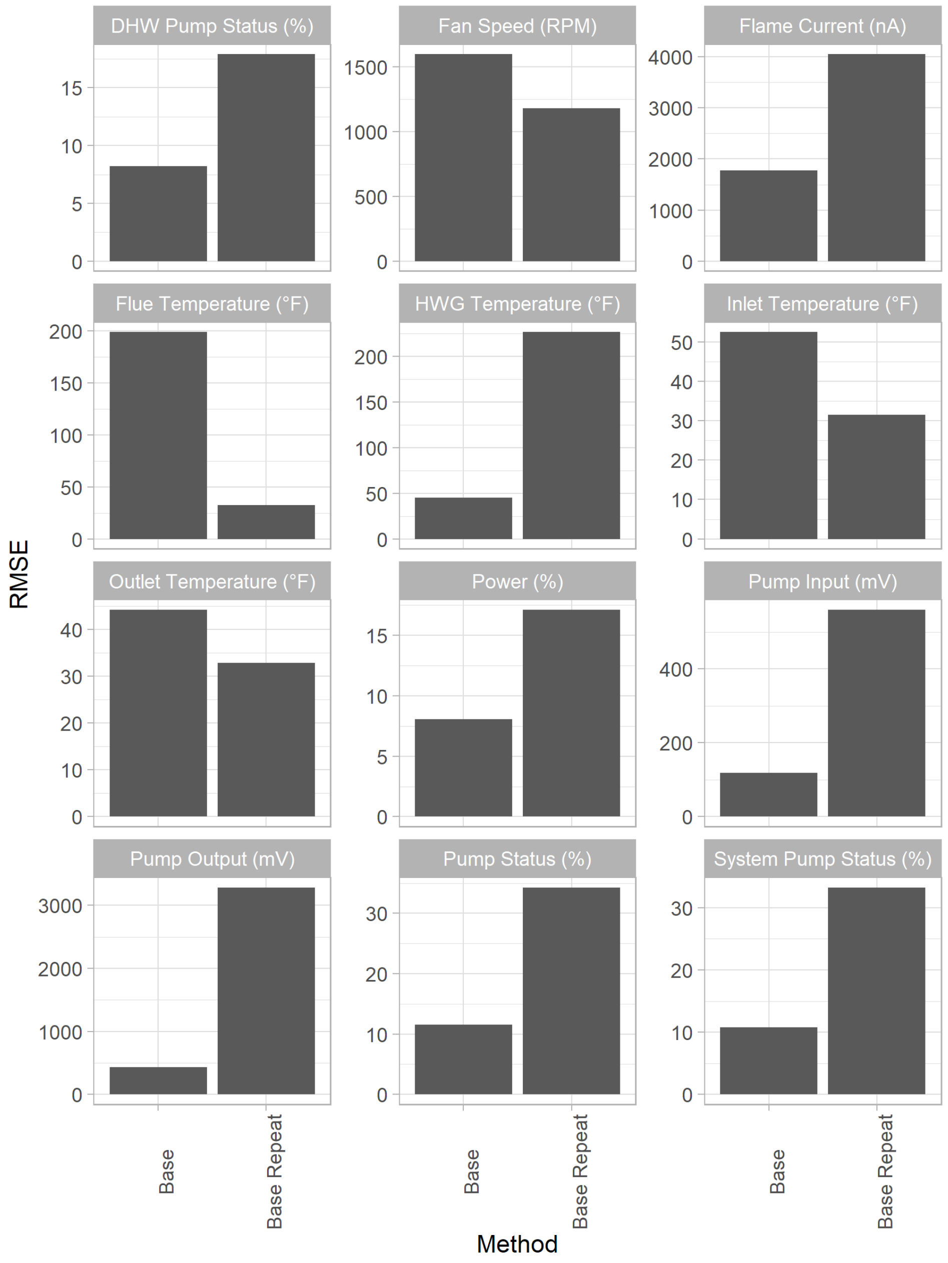


Figure 53: RMSE of Base NN and Base Repeat Prediction Models

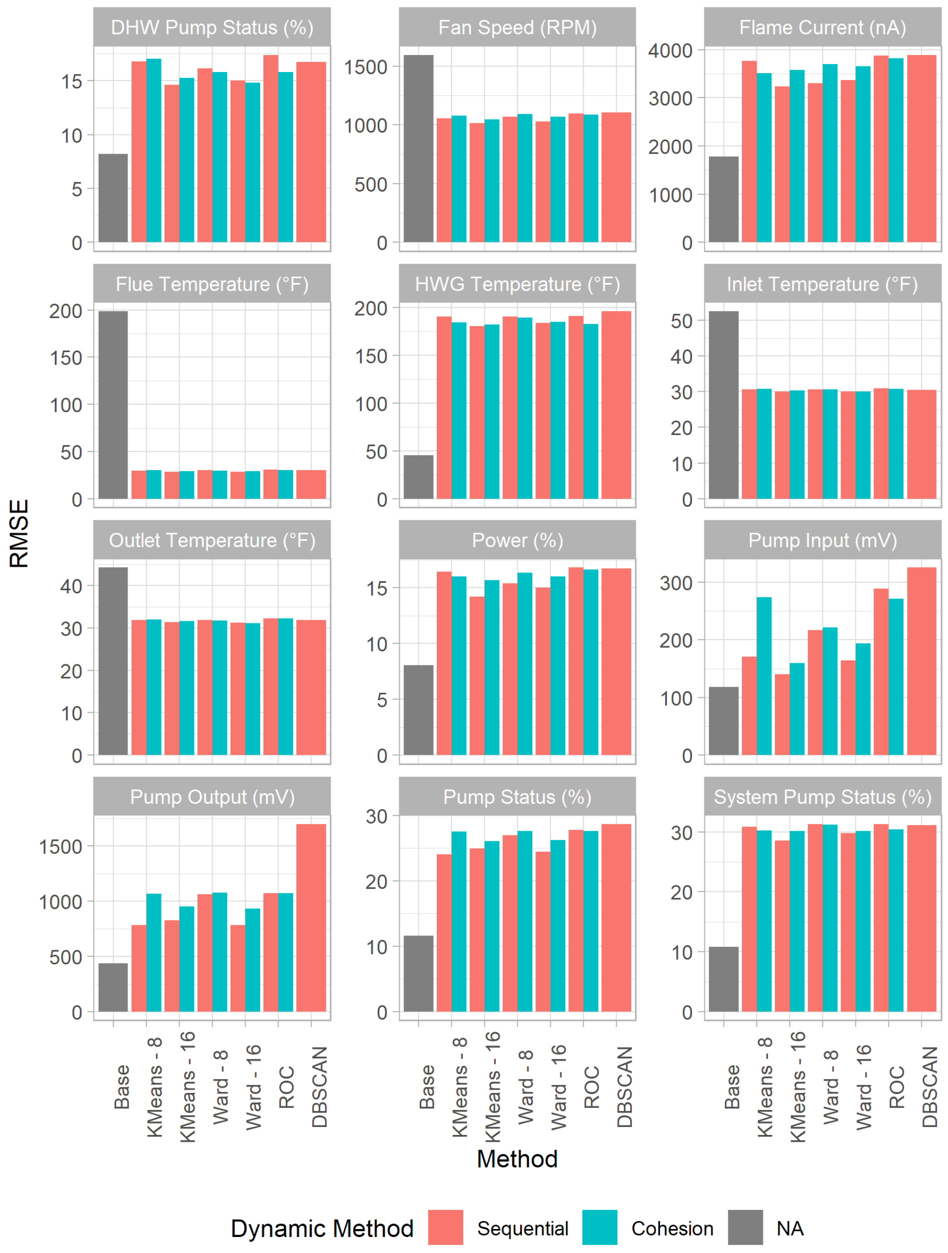


Figure 54: RMSE of Base NN and Clustered Repeat Prediction Models

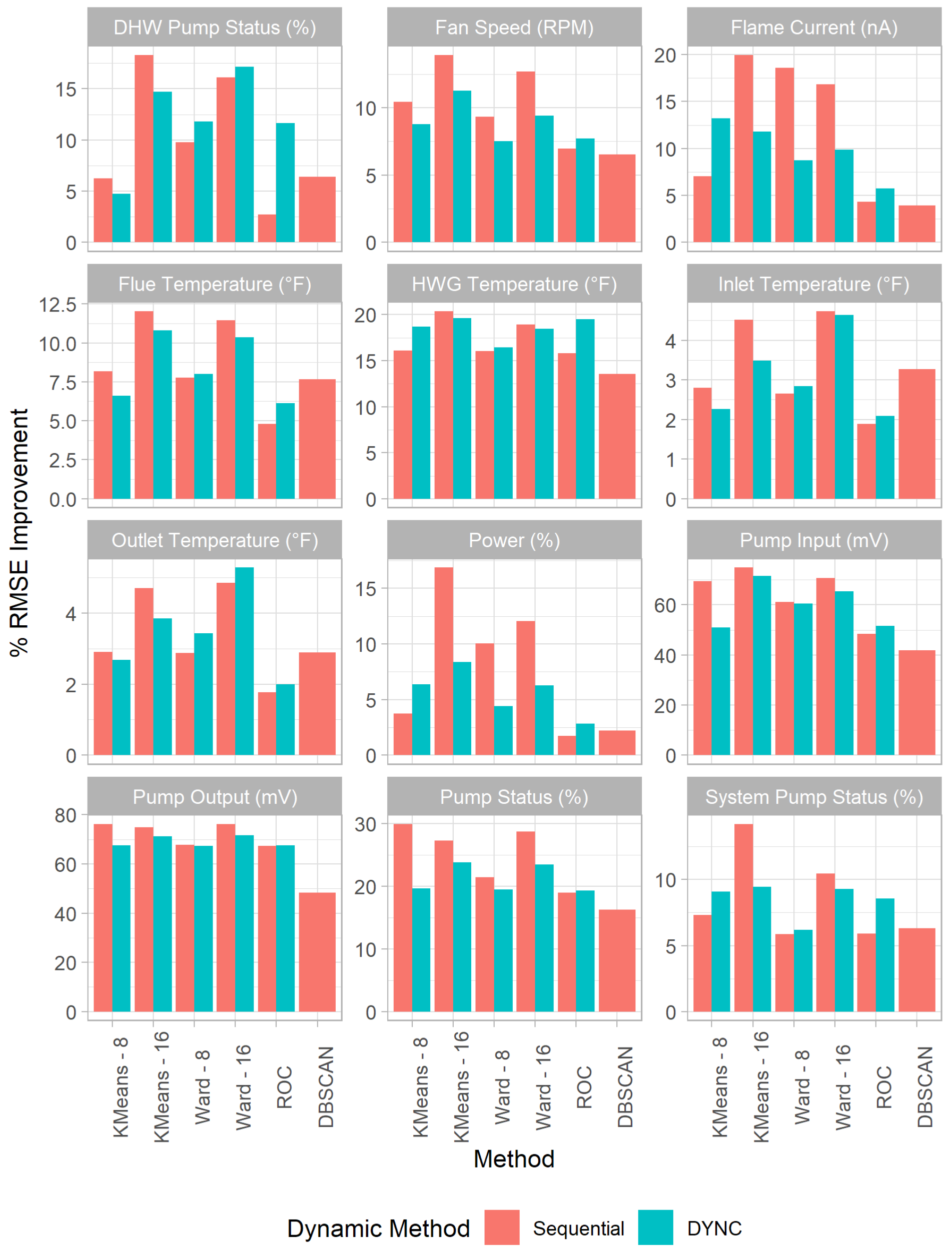


Figure 55: RMSE Improvement of Clustered Repeat Prediction Models over Base Repeat Model

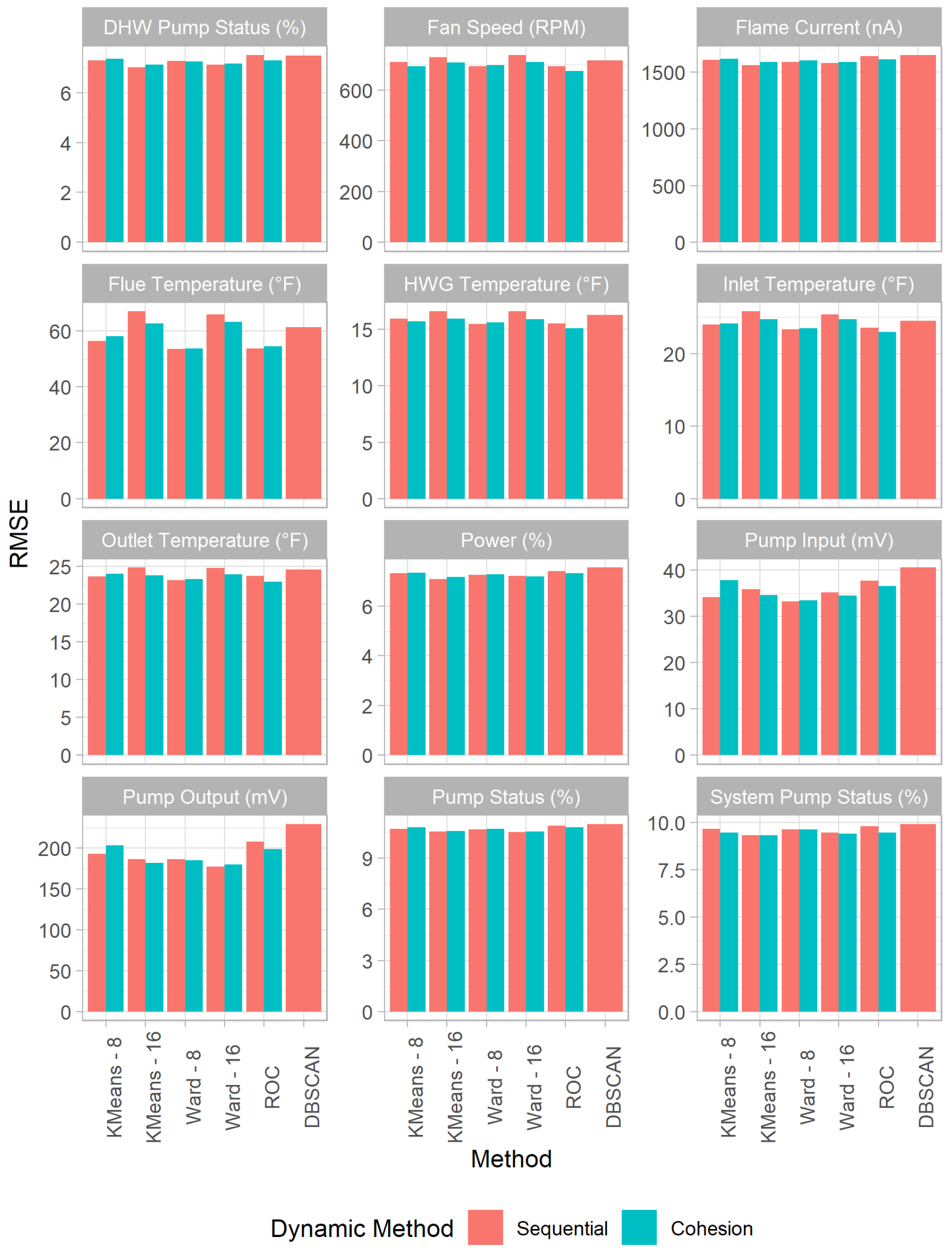


Figure 56: RMSE of Clustered NN Models

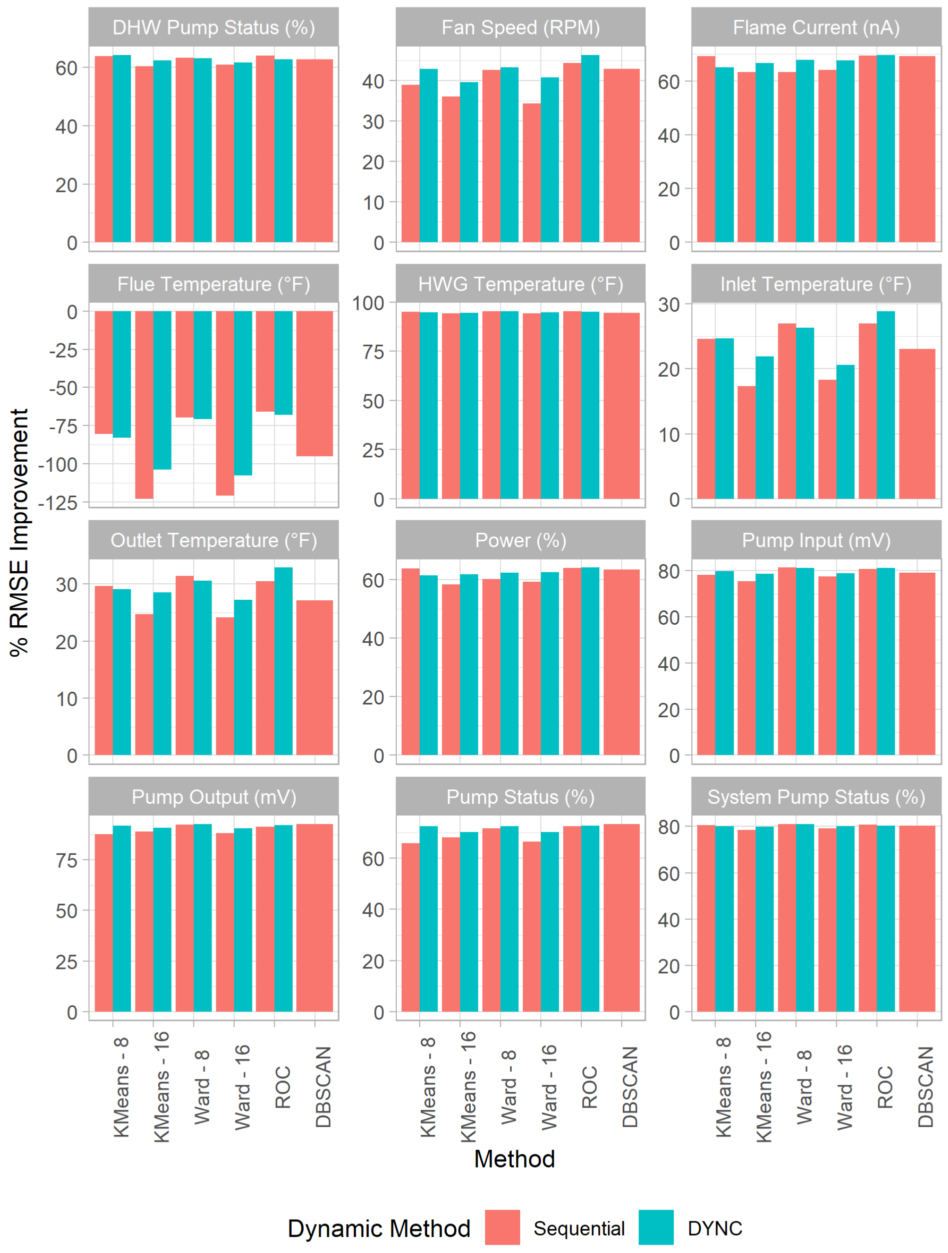


Figure 57: RMSE Improvement of Clustered NN Models over Clustered Repeat Models

APPENDIX C - CASE-BASED-REASONING

List of Stop Words

up only she's after hers himself why an that'll was weren't each didn themselves too its wouldn't with when most a aren't ve you these are under m them re weren doesn't those but yours if won or doesn mustn't yourselves doing my against am between hasn't who needn into off than such haven how any d being very your hasn some once no of by here there where they he be don theirs further not have haven't o myself now to you're ma what ours until because their mustn shan't hadn having yourself needn't above again wasn't while our his ourselves both it's all just were can been through do same ain shouldn shouldn't about ll won't has s you'll few itself didn't that nor and other shan mightn don't isn't below as y she him for i from down we over you'd which in then out couldn't mightn't whom t you've hadn't will wasn had wouldn aren more does her at before the own should've did should it this on herself is couldn during me so isn