SANDIA REPORT
SAND2018-8018
Unlimited Release
Printed July 2018

Evaluation and Comparison of Machine Learning Techniques for Rapid QSTS Simulations

Logan Blakely, Matthew J. Reno, Robert J. Broderick

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.osti.gov/bridge

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online order:	http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2018-8018
Unlimited Release
Printed July 2018

Evaluation and Comparison of Machine Learning Techniques for Rapid QSTS Simulations

Logan Blakely, Matthew J. Reno, Robert J. Broderick
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1033

Abstract

Rapid and accurate quasi-static time series (QSTS) analysis is becoming increasingly important for distribution system analysis as the complexity of the distribution system intensifies with the addition of new types, and quantities, of distributed energy resources (DER). The expanding need for hosting capacity analysis, control systems analysis, photovoltaic (PV) and DER impact analysis, and maintenance cost estimations are just a few reasons that QSTS is necessary. Historically, QSTS analysis has been prohibitively slow due to the number of computations required for a full-year analysis. Therefore, new techniques are required that allow QSTS analysis to rapidly be performed for many different use cases. This research demonstrates a novel approach to doing rapid QSTS analysis for analyzing the number of voltage regulator tap changes in a distribution system with PV components. A representative portion of a yearlong dataset is selected and QSTS analysis is performed to determine the number of tap changes, and this is used as training data for a machine learning algorithm. The machine learning algorithm is then used to predict the number of tap changes in the remaining portion of the year not analyzed directly with QSTS. The predictions from the machine learning algorithms are combined with the results of the partial year simulation for a final prediction for the entire year, with the goal of maintaining an error <10% on the full-year prediction. Five different machine learning techniques were evaluated and compared with each other; a neural network ensemble, a random forest decision tree ensemble, a boosted decision tree ensemble, support vector machines, and a convolutional neural network deep learning technique. A combination of the neural network ensemble together with the random forest produced the best results. Using 20% of the year as training data, analyzed with QSTS, the average performance of the technique resulted in ~2.5% error in the yearly tap changes, while maintaining a <10% 99.9th percentile error bound on the results. This is a 5x speedup compared to a standard, full-length QSTS simulation. These results demonstrate the potential for applying machine learning techniques to facilitate modern distribution system analysis and further integration of distributed energy resources into the power grid.

CONTENTS

FIGURES

NOMENCLATURE

Boost	Boosted Ensemble
CNN	Convolutional Neural Network
DER	Distributed Energy Resource
DL	Deep Learning
DNN	Deep Neural Network
IEEE	Institute of Electrical and Electronic Engineers
ML	Machine Learning
NN	Neural Network (references the ensemble in this case)
OpenDSS	Open Distribution System Simulator™
PV	Photovoltaic
QSTS	Quasi-Static Time-Series
RF	Random Forest
SVM	Support Vector Machine

1. INTRODUCTION

The increasing presence of solar power and other distributed energy resources (DER) on electric power distribution systems present novel challenges in both analysis and implementation. Traditional methodologies for analyzing power systems are unable to cope with the dynamic nature of DER impacts, particularly the extreme variability in photovoltaic (PV) systems. Quasi-static time-series analysis presents a way to analyze the highly variable, time-dependent nature of DER in the distribution grid setting [1]. However, fully detailed QSTS simulations at one second resolution over an entire year are currently prohibitively computation intensive. New, rapid methods for conducting QSTS analyses are required to fully utilize the capabilities of QSTS in a practical setting. The goal of this research is to reduce the computational time of QSTS simulations by applying machine learning techniques to QSTS.

Historically, the analysis of distribution systems was conducted using a ‘snapshot’ methodology, choosing a handful of moments during the year that served as examples of the most extreme situations likely to be encountered by the distribution system [2]. This would include samples such as the hottest day of summer, coldest day of winter, etc. This type of analysis is sufficient for systems that change slowly over time and are highly predictable. The integration of DER technologies radically changes that paradigm with higher variability[3], energy storage capabilities [4], advanced inverters [5], and communication-based control latencies [6], [7]. While this research focuses on the impact of PV systems, much of the analysis described here is also applicable to other DER technologies. By nature, the power output of PV systems can be highly variable on a second-to-second basis due to moving cloud cover and variability in yearlong behavior due to seasonal trends. PV power output can also be smooth during certain periods (full cloud cover or completely clear days). There is no way to capture this type of dynamic behavior accurately using the snapshot methodology.

For this reason, many PV impact studies require QSTS analysis. There are many different types of impacts that QSTS analysis can inform for example, voltage regulator changes, steady-state voltage, voltage flicker, thermal loading, line losses, communication requirements, and/or control systems analysis. Each of those metrics has slightly different requirements in terms of the resolution and length of simulation that is required; for a detailed analysis please reference [2] [1]. This project focuses exclusively on the voltage regulator tap changes analysis. There are two reasons for this, first, this metric has been shown to be one of the more difficult metrics to analyze and predict, and second, it has been shown to be a significant driver of maintenance costs [8]. A high-penetration of PV on a distribution system can increase the number of tap changes that occur throughout the year which may lessen the lifetime of the voltage regulators [2], [8], [9]. Voltage regulators often have time-delays on the order of 30 seconds. To capture the interactions that cause the movement of the regulators, a resolution of <5 seconds is required [1], [2].

Solving a yearlong QSTS simulation at 1-second resolution is currently so computationally expensive that it prohibits widespread use in industry. The simulations can take anywhere from 10-120 hours depending on the hardware used and size/complexity of the feeder being analyzed [10]. For PV impact studies, at least two simulations must be run, a base case and a case with added PV installations. However, it is often desirable to know the impact of different types of PV installations or in different locations, which requires many more simulations and even longer simulation times. The overall goal of this research is to significantly decrease the time required for these types of studies using machine learning

This research conducted a survey of several supervised machine learning techniques in order to simulate only a portion of the year using QSTS and then allow the machine learning to predict the remainder of the year. Neural networks, two types of decision tree ensembles, random forest and boosted ensemble, support vector machines, and deep learning were chosen for this purpose [11, p. 10] [12]. There has been little investigation into applying machine learning techniques to QSTS and only slightly more to distribution systems analysis in general. This research details a methodology for applying machine learning to QSTS analysis, as well as detailing the advantages and disadvantages of machine learning and some of the specific techniques available.

The decision of which machine learning algorithms to investigate is not necessarily a straightforward one. There are a large number of choices, each with advantages and disadvantages, and at this point, there are no guidelines of which algorithm to choose that work consistently [11, p. 10], [12]. This was one motivation for choosing to investigate multiple methods. The chosen algorithms have a record of performing well on complex problems and strike a balance between older methods with more literature and newer techniques, while keeping the goal of speed in mind. All of the techniques investigated here are supervised machine learning techniques because the training labels are easily available by running the QSTS analysis on a portion of the year.

According to [12], the decision tree ensemble methodologies (random forest [13], [14] and boosted ensemble [15]–[17]) performed the best *overall* in their experiments. Neural network and support vector machines were the other top algorithms, and depending on the experiment/dataset sometimes performed better than the decision tree ensembles.

Deep learning is a newer technique than any of the others that were chosen. One of the main advantages is the ability of these networks to take the raw data as an input and calculate their own features in a feature extraction stage [18]. Convolutional neural networks were the type of deep learning chosen to be investigated here [19].

Each of these methods takes a different approach to machine learning. Neural networks learn the correlation between the inputs and the outputs as a function approximator. The decision tree ensembles leverage the power of weak learners (decision trees) in aggregate, by combining the predictive power of individual trees into an ensemble of hundreds of trees. Random forest uses an ensemble of general trees, and boosted ensemble uses an ensemble of specialized trees. Support vector machines utilize a transformation of the input into a higher dimensional space. Convolutional neural networks combine feature extraction from raw data with the function approximation power of standard neural networks.

It has been consistently shown that ensemble methods perform better than single instances in isolation [12]. An ensemble method combines individual instances of a technique (or multiple techniques) to form one prediction. For example, a single neural network could be used for the predictions, but an ensemble of neural networks each forming a prediction and then combined to form a single final prediction has higher accuracy in general than the single network by itself. Each of the methods investigated has been shown to perform better as part of an ensemble than in isolation, and the results shown here use the ensemble versions, with the exception of the deep learning. The reason ensemble learning was not employed in the deep learning case is further explained in Section 6. Deep learning The primary reason that ensembles tend to perform better is that each member of the ensemble will produce a slightly different result, and in aggregate, the results are much closer to the ground truth [12], [13], [15], [20]. The neural network results from this project's research further confirm that fact. Please see [21] for figures and further analysis.

2. METHODOLOGY

2.1 Overview

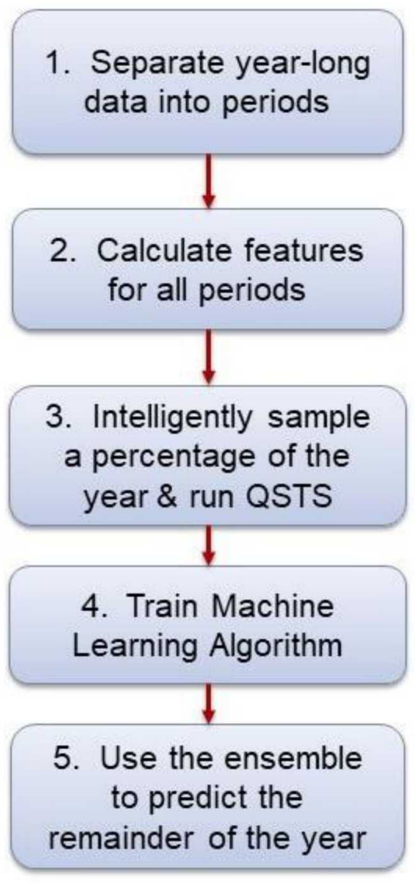

1. Separate year-long data into periods
2. Calculate features for all periods
3. Intelligently sample a percentage of the year & run QSTS
4. Train Machine Learning Algorithm
5. Use the ensemble to predict the remainder of the year

Figure 1 - Methodology overview

A graphical overview of the steps in the methodology are detailed in Figure 1. Steps 1 & 2 prepare the complete dataset to be used as input for the machine learning algorithms. Step 3 chooses a percentage of the year to be simulated using QSTS and then uses this set of data as an input into the machine learning algorithms. Step 4 trains an individual machine learning algorithm using the data from Step 3. Step 5 predicts the remainder of the year (the portion not chosen in Step 3). The sum of the QSTS results from Step 3 are then combined with the sum of the predictions from Step 5 for the full year prediction. Each method surveyed follows this same series of steps with one exception, for the deep learning method, the "Calculate Features" step is omitted and the load and PV time series are used directly as input data.

A complete, sequential QSTS was run on the complete dataset to produce the ground truth results, and the predicted tap changes are compared to that full-length simulation. In discussion with our industry partners for this project, they defined a 10% acceptable error threshold in the year-long predictions [1]. All of our results reference that threshold.

The remainder of the Methodology section is organized as follows, Section 2.2 QSTS Distribution System Test Circuit and Datasetdetails the dataset used in the research, Section 2.3 Feature Selection covers Steps 1 & 2, Section 2.4 Intelligent Samplingcovers Step 3, Section 2.5 Simulation Validationcovers simulation validation. The remainder of the paper covers the machine learning algorithms in Steps 4 & 5.

2.2 QSTS Distribution System Test Circuit and Dataset

The QSTS simulation is on a modified IEEE 13-bus test circuit that incorporates a centralized PV system at the end of the feeder, shown in Figure **2**. The circuit has three single-phase voltage regulators at the feeder head and one single-phase capacitor and a 3-phase capacitor bank out on the feeder. The voltage regulators are modified to provide ±5% regulation, and a voltage switching control is added to the 3-phase capacitor. The phase of some loads is changed to slightly balance the feeder, and all loads were increased by 20% to create some more extreme conditions. The load time-series is a 5-minute resolution normalized profile based on substation SCADA measurements from a feeder in California in 2013. A large 3-phase latitude-tilt 2MW PV system (~40% penetration of peak load) is added at the end of the feeder. The global horizontal irradiance (GHI) time-series measured at 1-second resolution in Oahu is converted to plane-of-array (POA) irradiance using the DIRINT decomposition model and the Hay/Davies transposition model. The 5-minute resolution load data was interpolated to match the 1-second resolution of the GHI time-series. The Sandia Array Performance model and Sandia Inverter models are used to convert the POA irradiance into PV power output time-series. The circuit is modeled in OpenDSS and the algorithm is coded in MATLAB using the GridPV toolbox to interact with OpenDSS. The simulation is a year at one-second resolution.

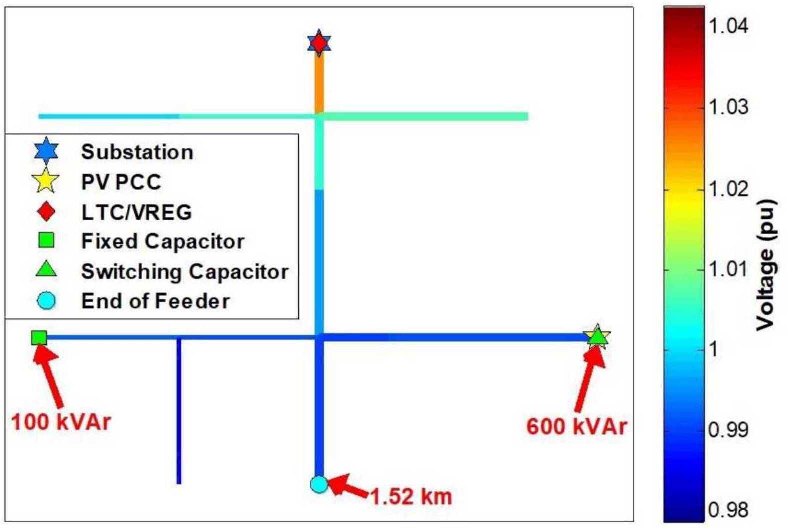


Figure 2 **- Diagram of the modified IEEE 13-bus feeder colored by voltage [21]**

2.3 Feature Selection

The first step in the methodology is to divide the year-long time-series dataset into periods. These periods are an arbitrary interval (minutes, hours, days, weeks, etc.). Regardless of the size of the period, the machine learning will learn the correlation between the data for that period and the number of tap changes. The reason for dividing the year into periods is that these machine learning algorithms require discrete units for training and testing data. For this project, 2-hour periods were chosen based on the work done in [21]. That work demonstrates results using a variety of values for the length of the time period, both smaller and larger, and two-hour periods provide the lowest average error in those experiments. A further reason for choosing 2-hour periods is discussed in Section 2.4.1 Intelligent Sampling Methodology and shown in Figure **5**. For the deep learning portion of the research 1-hour periods were chosen and the reasons for that choice are discussed in Section 6. Deep learning.

The next step is the feature selection itself which is the process for determining what to use as the actual input for the machine learning algorithm. In the deep learning case, the 1-second resolution load and PV time-series are used as inputs. One of the advantages of the deep learning approach is that the network is able to extract its own features from the raw data. In the case of the other ML algorithms that were chosen, the 1-second time-series needs to be converted into features that are selected by hand.

A total of 17 input features were generated for each period. Statistical features such as maximum, minimum, mean, etc. were calculated. It has been shown that solar variability is a significant driver in the number of tap changes [22], so a number of variability statistics were calculated as well. Day of the year, and hour of the day are highly correlated to solar power and so those were also included. Due to the time-dependent nature of QSTS [23] the standard deviation of the previous two weeks was added as a feature. See Table 1 for a complete list of the 17 features.

Table 1 - Full Feature List

Input Features	
Maximum (PV & Load)	Minimum (PV & Load)
Mean (PV & Load)	Range (PV & Load)
Length (PV & Load)	Standard Deviation (PV & Load)
Median (PV & Load)	Day of the Year
Standard Deviation of previous two weeks	Starting Hour

For neural networks choosing the final feature list is particularly important. The features need to be as relevant and non-redundant as possible. Some of the features in the full feature list in Table 1 are highly correlated. Within the neural network context, it is better to have a few features that are not correlated than many features that are correlated. Figure 3 plots the correlation coefficients between each of the features. For example, max, min, median, and mean are all highly correlated. Out of those four, only the most highly correlated, the mean, was included in the final feature list for the neural network ensemble. In the end, a down-selected list of 9 out of the 17 features were used as inputs to the neural network ensemble.

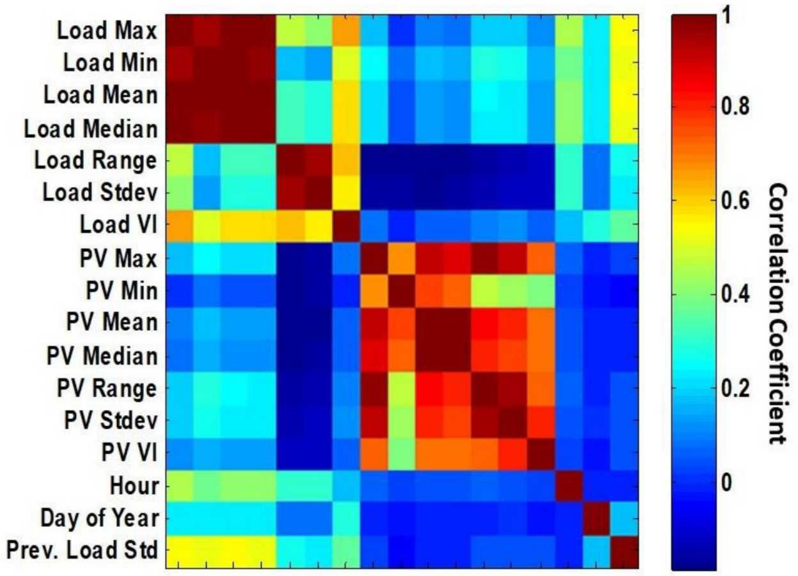


Figure 3 - Feature list correlations [21]

For the decision tree ensembles, the full 17-feature list was used as inputs. For both random forest and the boosted decision tree ensembles, part of the power of the ensemble comes from the diversity of trees included in the ensemble. Thus, keeping the larger feature list (despite some features being highly correlated) increases the diversity in the forest [13]. This was born out in our experiments as well, being that the 17-feature input version performed better than a down-selected feature list. The full input list was also used in the Support Vector Machine (SVM) case.

The search space of possible input features is extremely large, and it is impossible to do an exhaustive search. Therefore, the hand-picked subset of 17 features was chosen to be evaluated. This subset performs well in these experiments, however there is no guarantee that this subset of features is optimal. One reason for constraining the input feature list to 17 features, and down selecting to 9 for the neural network is that as the number of features increases, the number of training samples required also increases. This is sometimes termed the 'curse of dimensionality' [24]. Intuitively, the larger the number of features, the more training samples are required to accurately represent the combinations that those features could represent. Within the parameters of this research, the total amount of data is fixed, and so the number of features was kept small. In general, adding additional input features requires more parameters (weights in the neural network case), which increases model complexity as well. For further discussion of possible future work using other possible choices for features, please see 8.3.2 Future Avenues.

2.4 Intelligent Sampling

2.4.1 Intelligent Sampling Methodology

The Intelligent Sampling step in the methodology selects some percentage of the total number of periods in the year to be run using the QSTS simulation and subsequently used as inputs to the machine learning algorithm. The portion of the data used as the training set for the machine learning algorithm needs to be representative of the entire year, meaning that the distribution of the features inside the training set needs to reflect the distribution of the features in the dataset as a whole. Previous research has shown that simply using random sampling to select the training data requires sampling ~85% of the year in order to guarantee an error under the 10% threshold [1]. Intuitively, this high value means that randomly selected samples are not representative, for example the random sample may choose primarily clear days or perhaps primarily days in the summertime. Random sampling does not necessarily achieve a representative sampling until ~85% of the year is sampled. For this reason, the stratified sampling approach in [25], henceforth referred to as the "Intelligent Sampling" method, was chosen. Using a stratified sampling approach guarantees that a diverse selection of periods is chosen. At least one sample from each 'bin' is chosen and then weighted sampling based on the number of samples in each bin is used for the remaining samples. Figure 4 shows a plot of the stratified sampling bins, with Load Median plotted on the y-axis, Solar Variability Index (VI) on the x-axis, and the average number of tap changes in that bin in color. This methodology ensures that there is at least one sample from each bin included in the training set.

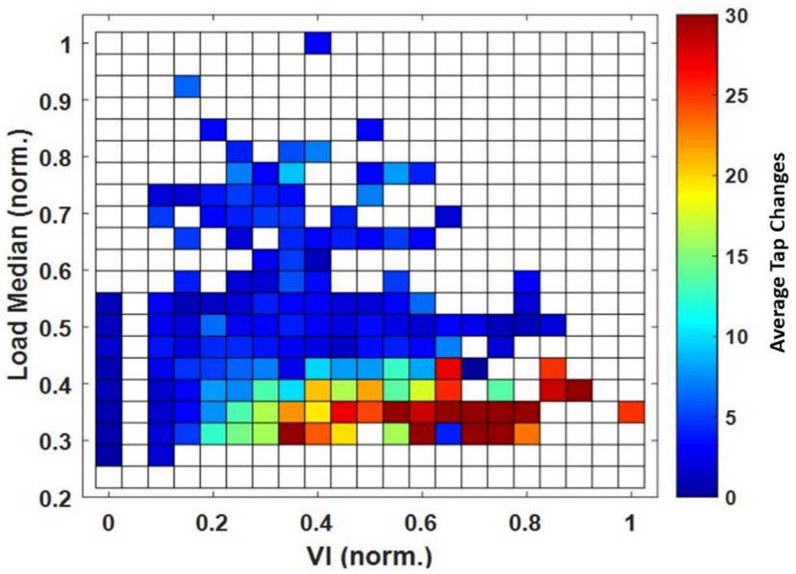


Figure 4 - Intelligent Sampling bins [25], white squares indicate there are no samples with those metrics

The percentage of the year being sampled determines the stratification level and grid size. So, when a smaller percentage of the year is being sampled, the bins themselves are larger to ensure a 'representative' sampling is obtained. When a larger percentage of the year is required the bins are smaller and therefore the resulting training set is a better representation of the full dataset. One of the important results of this research is determining what percentage of the year is required, i.e. discovering what percentage of sampling obtains an acceptable representation of the whole dataset.

[26], [27] demonstrate similar methodologies where the goal was to extrapolate the results of a hosting capacity analysis done on a representative subset of feeders to the remainder of the feeders in the group. The ability to select representative periods of the year is an interesting research question for a variety of reasons. [28] presents motivations for representative sample selection in a more general setting and describes a similar stratified sampling approach. Our results show it is possible to select a representative portion of the year and stay below an arbitrary error threshold of 10% for the QSTS analysis.

Once the training data set has been sampled using the methods described above, that data is simulated using QSTS. However, this training set is now a set of non-contiguous periods from throughout the whole year; it is no longer a regular time-series. What that means for the QSTS simulation step is that the periods are being run using QSTS individually, in isolation from the other periods. This introduces some error since the current state of the system is dependent on the previous state of the system [23]. Figure **5** shows the error introduced by running QSTS on the individual periods rather than sequentially. Overall, the error introduced is quite small, and these results also provide one reason that 2-hour periods (1-hour for deep learning) were chosen as the period size. This small error is included in the overall error results seen in the following sections. The choice of period size does not affect the total information contained in the dataset. The total number of tap changes in the yearlong dataset remains the same, regardless of the period size, and that limits the overall information available to the learning algorithms. The other considerations for choosing a period size are that using too small of a period size results in the statistical features being noisier; whereas, too large of a period will make the statistical features too general. Figure **5** introduces a lower bound on the period size choice in terms of introducing new error, and for periods more than two hours, the variation that drives the tap changes begins to be lost.

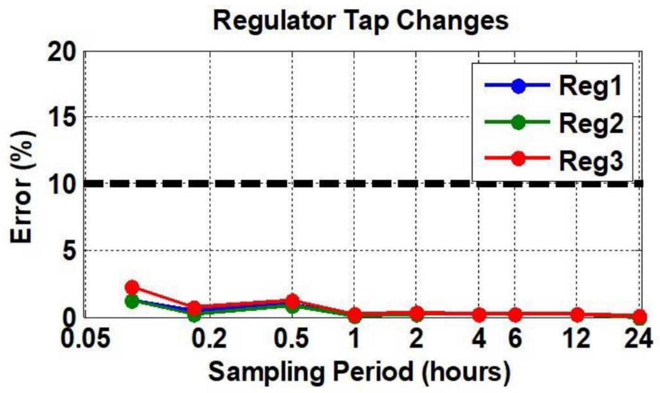


Figure 5 **- Error introduced by running QSTS on individual periods**

2.4.2 Speed versus Accuracy

It is worth noting that choosing a percentage of the year to simulate with QSTS and use as training data is a tradeoff between speed and accuracy. The more of the year that is simulated the more accurate the machine learning prediction will be and the lower the overall error. However, the higher the percentage of the year that is simulated the more time is spent on the QSTS computations. The time spent on the QSTS simulations for the training data is the driver of the overall time cost for these methodologies. The time spent on the machine learning portion is nearly negligible in comparison to the time necessary to run the QSTS simulations on the training data. Figure 6 illustrates this tradeoff. As the percentage of the year simulated increases the error decreases. The error would converge to zero when simulating 100% of the data, equivalent to the full-length QSTS simulation.

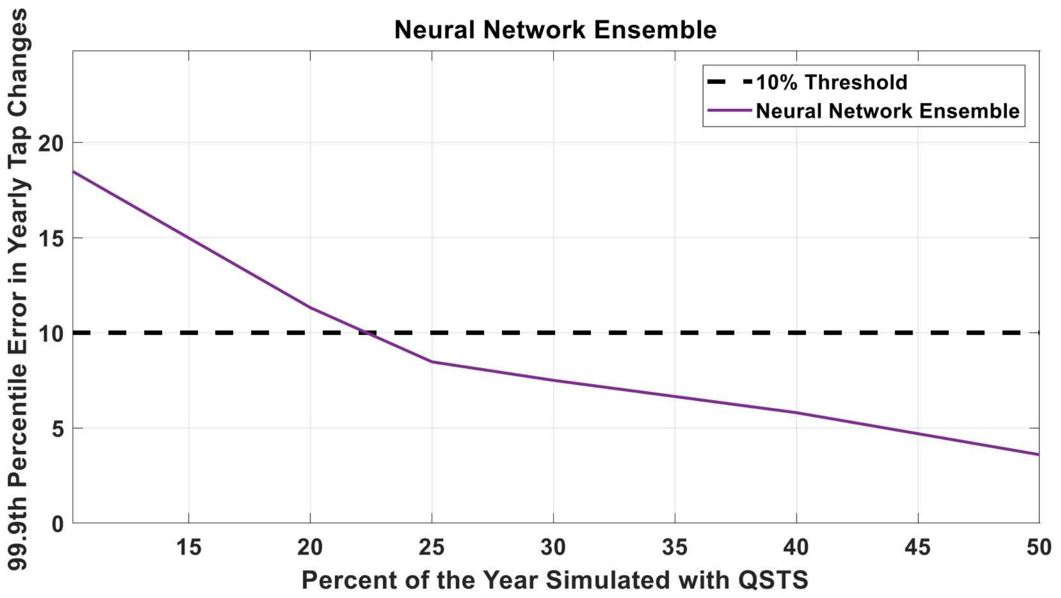


Figure 6 - Speed versus accuracy tradeoff

2.5 Simulation Validation

There are two steps in the methodology that introduce some elements of randomness into the results of the simulation. The Intelligent Sampling step introduces randomness in the choice of the samples used to train the machine learning algorithm, and the machine learning algorithms themselves introduce some randomness in their initial conditions and/or construction. In the average case, the Intelligent Sampling works well. However, because the sampling still contains a random component, it is possible that the Intelligent Sampling will sometimes return a non-optimal training set. The reason for this is that, while some bins potentially only have one sample and will always be selected, the majority of bins will have multiple samples, and the samples are randomly selected out of the bin. This technique was chosen to avoid overfitting and to allow the algorithm to generalize better to other feeders; a more deterministic algorithm may have produced better results in this particular instance but would be unlikely to perform well in general. In the same way, the initial setup for the neural network (weight initialization) or the construction of the decision trees (training subset choices and branching feature choices) can also return non-optimal results. The practical result of this randomness is that subsequent simulations are likely to produce slightly different results, and it is necessary to understand the extent of the possible error due to this randomness. To account for this and to bound the error for this methodology, a Monte Carlo approach was adopted. 10,000 Monte Carlo runs were conducted for each machine learning method, and the results were plotted as an error distribution over all 10,000 runs. All of the results figures that follow in this paper were created after 10,000 Monte Carlo runs for the given method. Figure 7 illustrates one reason for choosing 10,000 Monte Carlo runs. This figure was generated using the neural network ensemble methodology, and 10,000 runs is where the error begins to converge. We ran this experiment using only the neural network and used that as the basis for choosing 10,000 runs. Clearly, some tradeoff was made between the simulation time considerations and the number of Monte Carlo runs. The Sandia High-Performance Computing Cluster (HPC) was used to run these computationally intensive simulations, and in fact, running so many Monte Carlo simulations would require so much time that it would be unreasonable without the HPC resources. To be clear, the Monte Carlo simulations are only used to validate this methodology. In practice, only a single simulation would be run. The single simulations meet the speed goals set by the project.

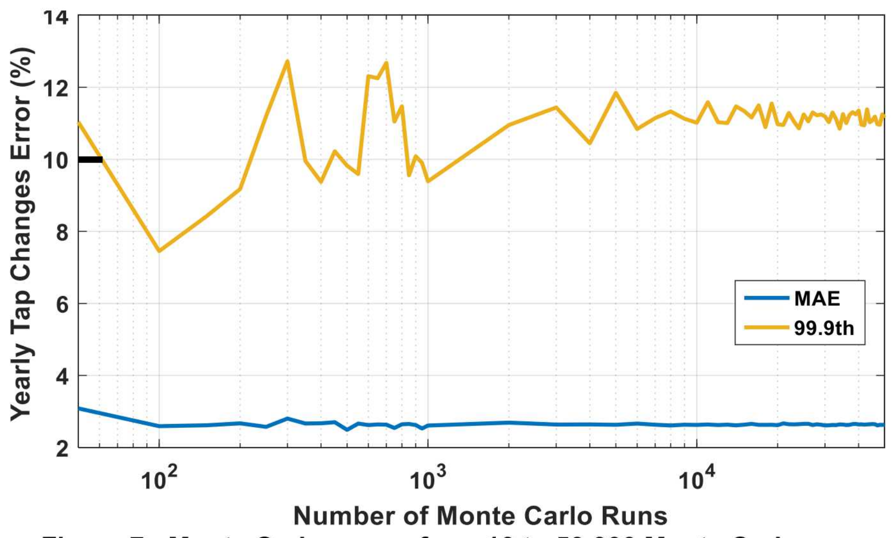


Figure 7 - Monte Carlo sweep from 10 to 50,000 Monte Carlo runs

Plotting a histogram of the error of each individual run over the 10,000 Monte Carlo runs, the distribution of errors is gaussian and therefore the maximum error goes to infinity as the number of simulations goes to infinity; see Figure 9 for an example. For these results, the 99.9th percentile error bound over 10,000 Monte Carlo simulations is shown, and is considered to be the error bound for this methodology. There is however a 1/1000 chance that a feeder would exceed this error bound.

3. NEURAL NETWORK ENSEMBLE

3.1 Overview

Neural networks are a supervised machine learning architecture designed to loosely model the workings of a human brain. A feed-forward, multi-layer perceptron neural network was chosen as the architecture for this portion of the project. Layers of 'neurons' are connected from one layer to the next via learned weights and the value of a neuron in the subsequent layer is calculated by taking the dot product of the previous layer's neuron value and the weights entering the neurons in the subsequent layer. The values for each neuron in the subsequent layer are calculated in this way and then a bias term is also added. Those values are then transformed into an 'activation' value using an activation function, in this case the sigmoid function. The values for the weights between layers are learned during the training of the network using the back-propagation methodology. For a more detailed description of how neural networks work and the training process see [29].

3.2 Architecture

The choice of the feed-forward neural network architecture was determined experimentally to be the best out of several alternatives that were investigated. Contrary to intuition, recurrent neural networks experimentally performed worse than the feed-forward versions. Given that the data is time-series in nature, recurrent networks would seem to be a logical choice. One hypothesis about this result is that after doing the Intelligent Sampling step the training data is no longer a regular time-series, being that a time-series is defined as samples collected at regular intervals [30]. There is supporting research showing similar results that recurrent networks may not perform as well as expected when the data is not a regular time series [31]. However, this hypothesis was not fully explored and it is possible that there are other factors contributing to the feed-forward network experimentally performing the best in these tests.

The final network architecture was an input layer of nine nodes (the number of input features used), a single hidden layer of 5 nodes, and a single output node as shown in Figure 8. The single output node outputs a numerical prediction for the number of tap changes in the given period. The number of hidden nodes was chosen using extensive experimental trial and error. There are some heuristics in literature for choosing the number of hidden nodes, but at this time there are no specific guidelines for doing so. The Levenberg-Marquardt method was used to train the network based on the work in [32].

An ensemble composed of 50 of the individual neural networks described above was used in the final ensemble architecture. For more details on the specifics of the architecture and tuning of the hyperparameters please see [21] which is the conference paper published based on the neural network research for this project.

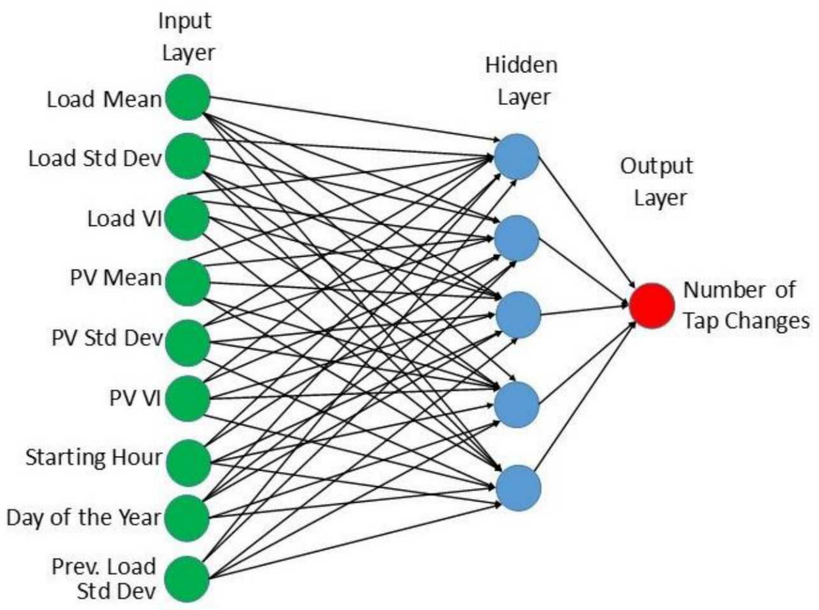


Figure 8 - Individual neural network architecture [21]

3.3 Results

Figure 9 shows the results of each of the Monte Carlo runs plotted in Histogram form. These results are using 20% of the year as training data. The mean absolute error for the neural network ensemble methodology is 2.6% error in the yearly tap changes and the 99.9th percentile error is 10.8%. You can see the error in general is quite good, however using 20% of the year does not bound the error below the 10% threshold set by our industry partners.

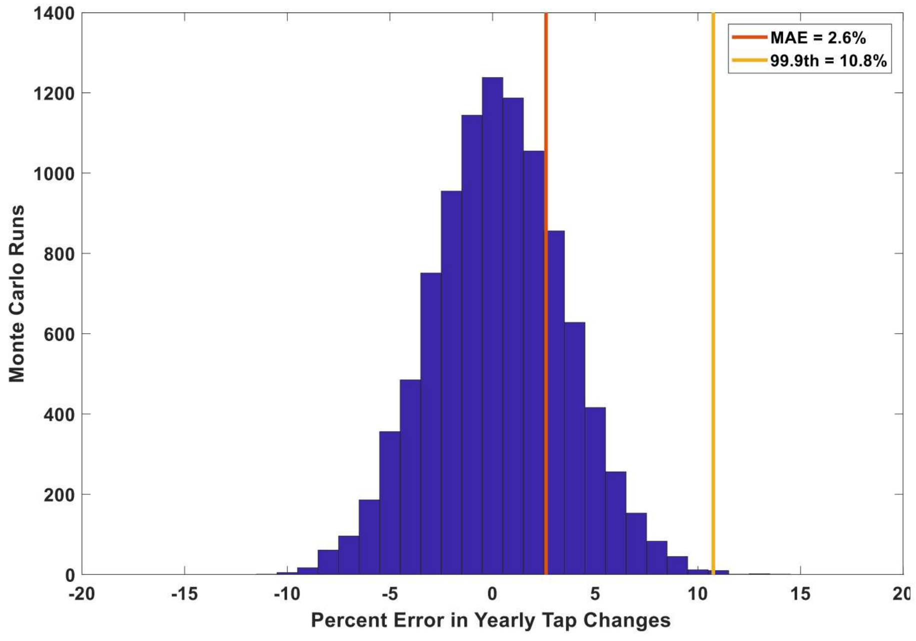


Figure 9 - Neural Network results histogram

Figure 10 shows the results for the Neural Network Ensemble using different percentages of the year as training data. These results were generated by running 10,000 Monte Carlo simulations for each of the percentages plotted below and then plotting the 99.9th percentile error for each percentage. You can see that between 20% and 25% of the year, the error drops below the 10% threshold. For comparison the results for Random Sampling and Intelligent Sampling are also shown. The Random Sampling line crosses the 10% threshold >85% of the year simulated. The Intelligent Sampling line are the results after doing linear interpolation on the intelligently sampled data. The neural network ensemble approach clearly outperforms Random Sampling, as well as simple linear interpolation on the intelligently sampled data.

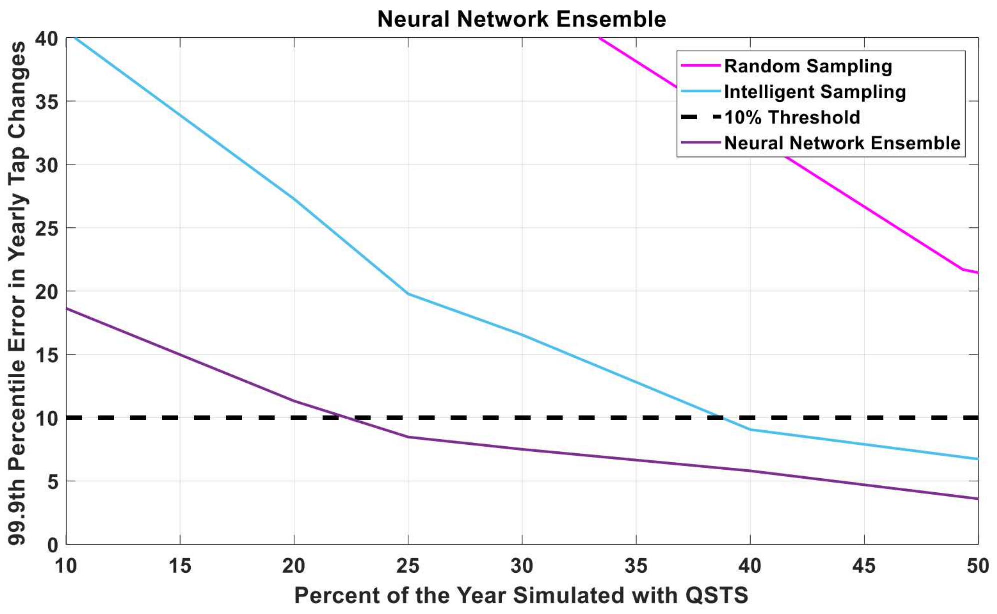


Figure 10 - Neural Network Ensemble results simulating different percentages of the year

4. DECISION TREE ENSEMBLES

4.1. Random Forest

4.1.1. Overview

Random forest is a supervised machine learning technique based on an ensemble of decision trees developed by Breiman [13] in 2001. Random forest leverages the predictive power of 'weak learners', in this case decision trees, in an ensemble 'forest' of trees to make a final prediction.

The individual trees are constructed using the CART (Classification and Regression Tree) process [33], with each tree making its own individual prediction for the number of tap changes during the given period. The CART methodology, at each branch point in the tree, randomly selects a subset of the available features and then chooses the feature from that subset that minimizes the error over all the samples. This process continues until the tree has reached its maximum size. At the construction of each tree, a subset of the training samples is chosen to construct that tree. The training samples are sampled with uniform probability with replacement. This is known as a 'bootstrapping' approach and when combined with a predictive ensemble is referred to as 'bootstrap aggregating' or 'bagging'. This sampling approach is one difference between this method and the boosting method described in the next section. The individual predictions are then averaged to form the final ensemble prediction. See Figure 11 for a visualization of the random forest process. For a more detailed description of random forest see [13] and [14].

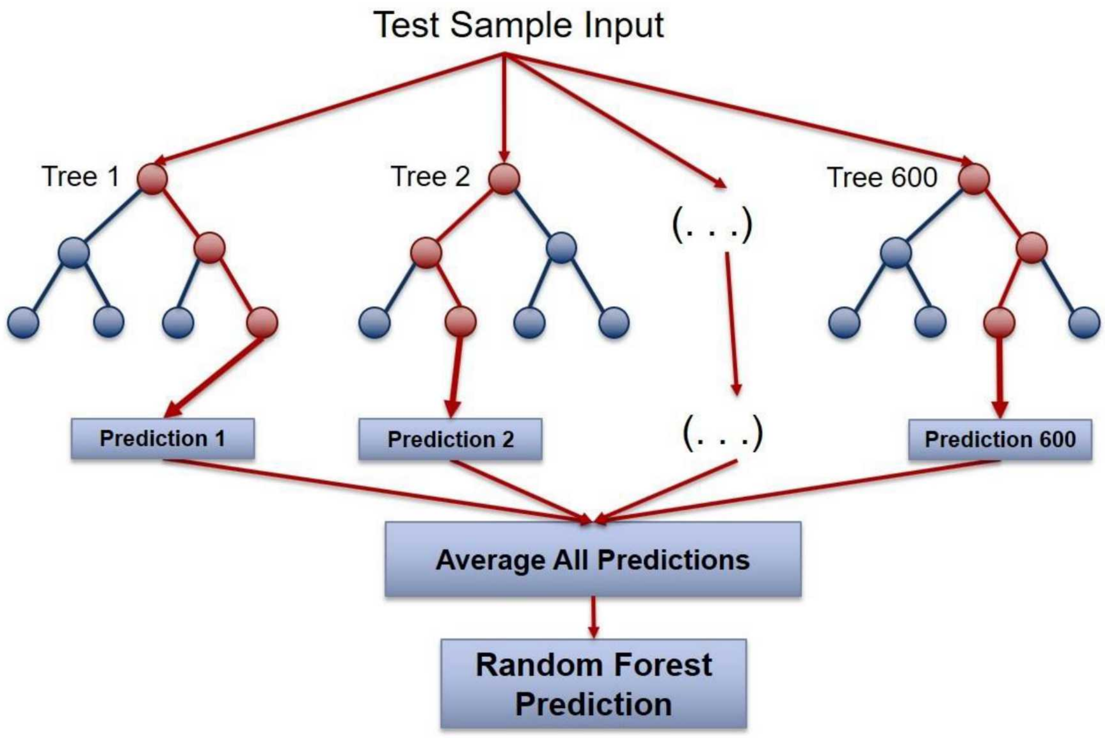


Figure 11 - Random Forest visualization

4.1.2. Architecture

As noted in Section 2.3 Feature Selection, the random forest uses all 17 of the available features from the Feature Selection section, and the algorithms randomly samples 5 to choose from at each branch during the construction of the tree. Five features constitute approximately one-third of the total features, which is one heuristic for choosing that hyperparameter; it was also verified using a parameter sweep over possible values. See Figure 12 for an example of the structure of one tree in the forest; all the trees in the random forest will have a similar architecture since they are all attempting to make the best *general* prediction possible. The triangles represent the 'if statement' branches on the selected feature. The red line represents a possible path that a test sample might take as it traverses the tree. Figure 13 shows a close-up of the green box from Figure 12. You can see some of the features chosen for those branches. Following the arrows, the sample branched left at the "Load Mean" node, meaning that the sample's Load Mean value was less than 1610.05. The 'leaves' of the tree represent the prediction of that tree for the given time period, 1.1667 tap changes for the two-hour period in this example. This random forest used 600 trees in the ensemble. For more details on the specific implementation for this research see the conference paper published on the results of the decision tree ensembles from this project [34].

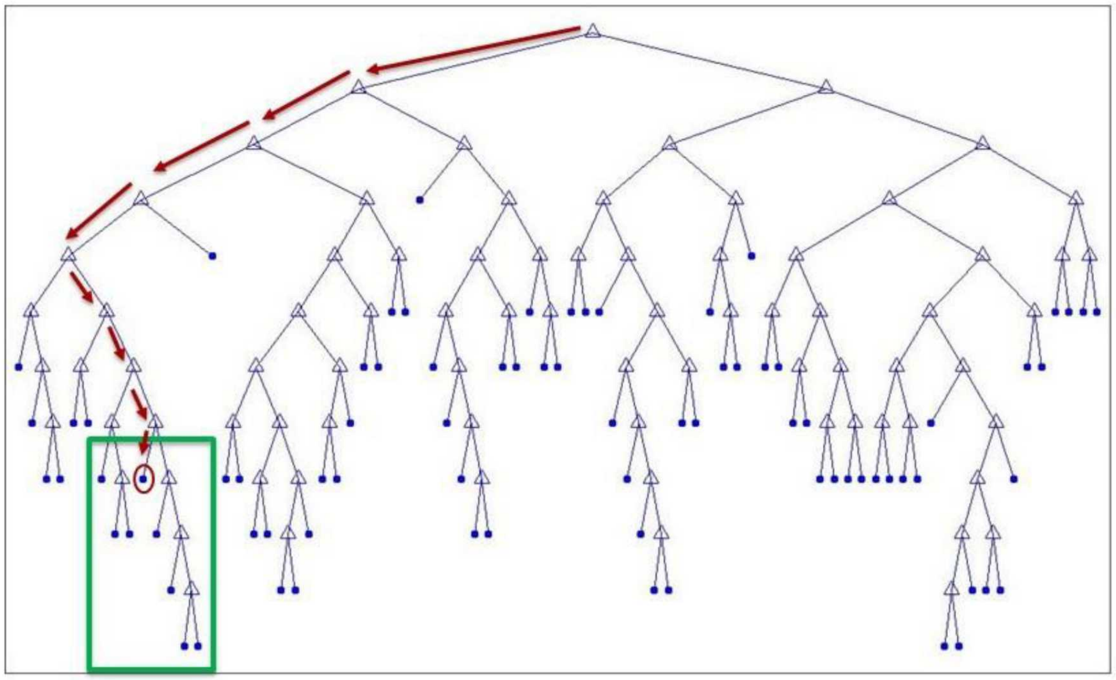


Figure 12 - Example random forest tree

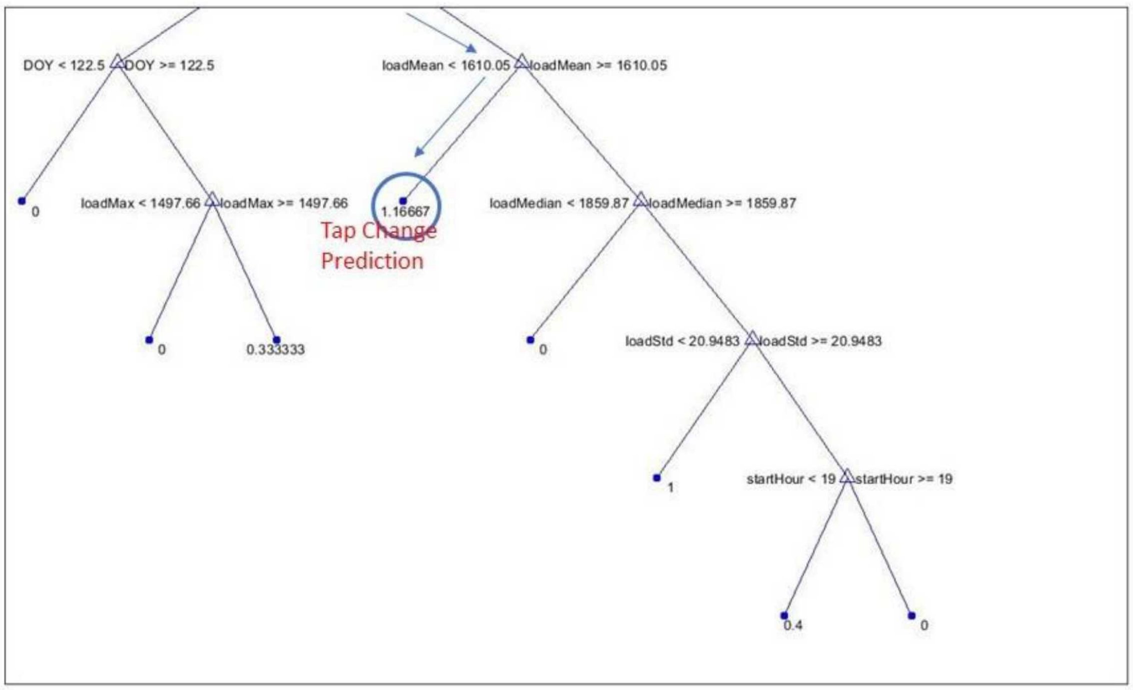


Figure 13 - Example close-up of a random forest tree

Looking at the individual trees in the forest, each tree is making the best possible *individual* prediction for the given time period. This is one of the key differences between the random forest method and the boosting method. Figure 14 shows the error distribution if each individual tree was forced to predict the entire year without the benefit of the ensemble. You can see the maximum error was 33.5%, and the 2.7% corresponds to the error of the random forest ensemble as a whole. If instead of predicting individually, this group of trees made an ensemble prediction, the 2.7% mean error would correspond to the error of that prediction.

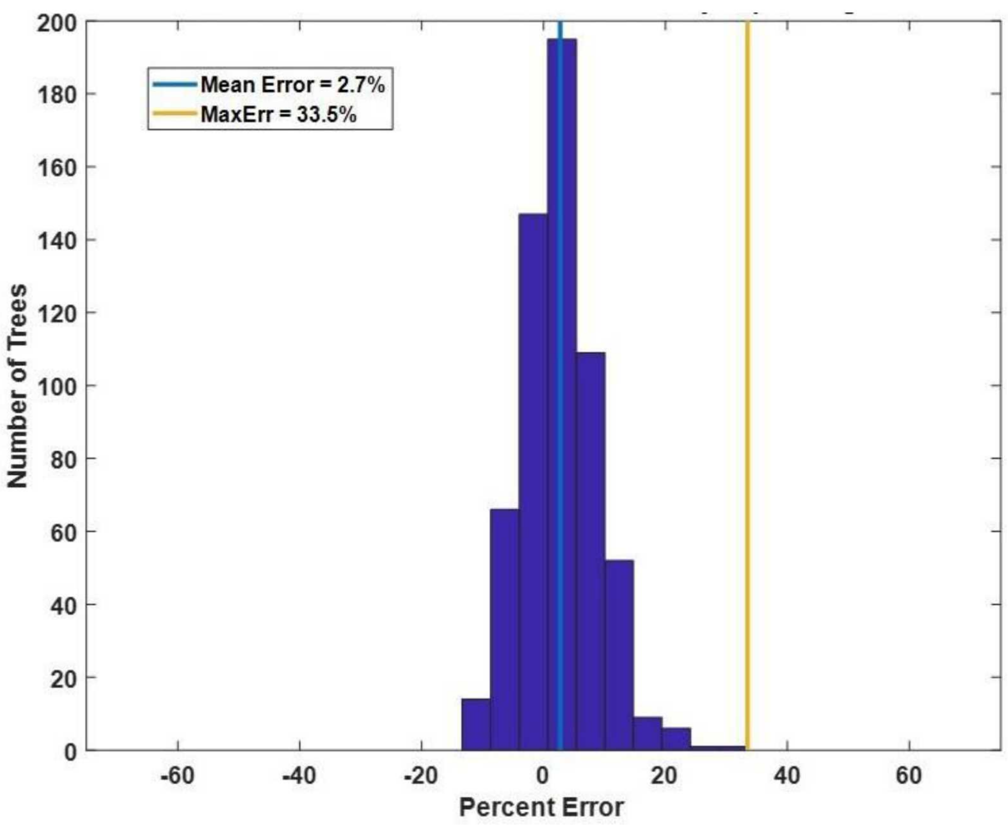


Figure 14 - Random Forest individual tree error

4.1.3. Results

Figure 15 shows the random forest results for the Monte Carlo runs using 20% of the year. You can see the Mean Absolute Error (MAE) of 2.6% and the 99.9th percentile error bound of 10.9%. The 99.9th percentile is quite similar to the neural network results from above and the overall error distributions look similar. Figure 16 shows the results simulating different percentages of the year using QSTS. At between 20% and 25% of the year, the method will result in errors below the 10% threshold.

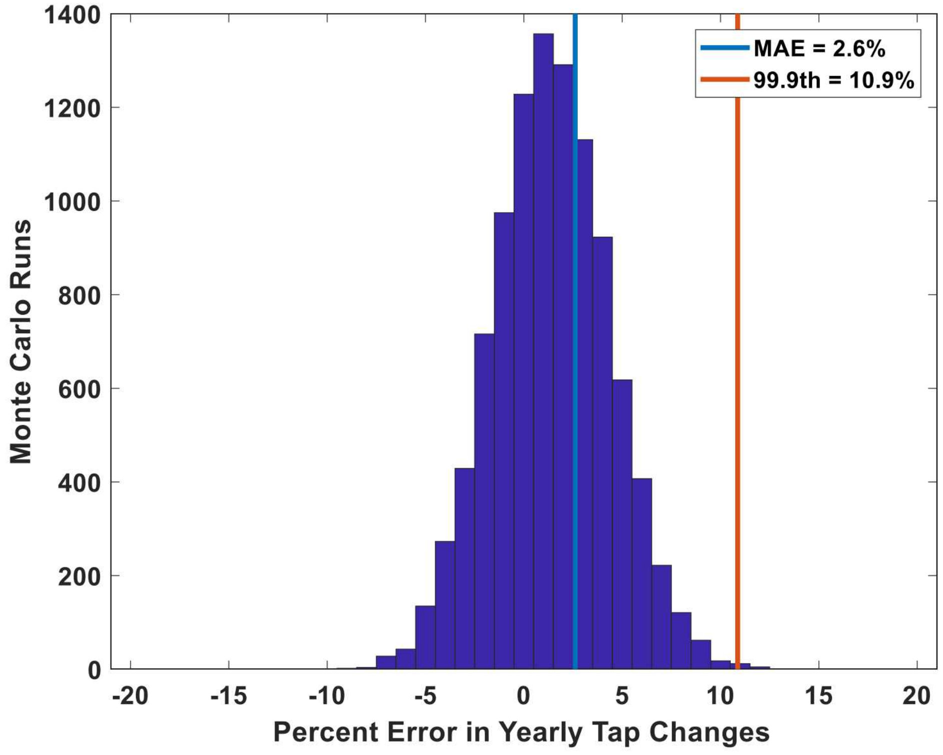


Figure 15 - Random Forest results histogram

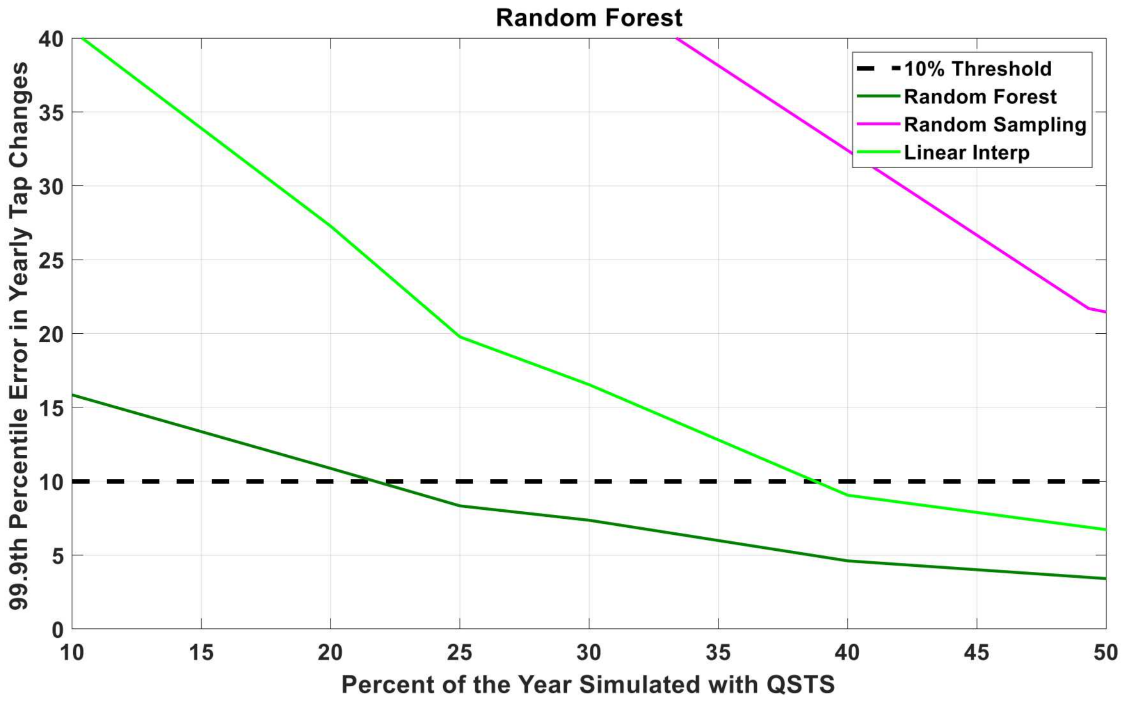


Figure 16 - Random Forest results simulating different percentages of the year

4.2. Boosted Decision Tree Ensemble

4.2.1. Overview

Boosting is a supervised machine learning technique proposed originally as AdaBoost [15] in 1997. The version used in this research is called LSBoost which is a regression variation of the AdaBoost algorithm. For a description of the specific workings of LSBoost see [16], [17].

A boosted ensemble shares many similarities to the random forest ensemble. Both are composed of CART-type trees and both produce a single ensemble prediction for the given period of time. There are two key differences between the random forest 'bagged' ensemble and the 'boosted' ensemble. The first difference is that the subset of training samples used to grow each tree are not chosen randomly. Once the first tree has been grown subsequent trees will try to better predict the training samples that were poorly predicted by previous trees. That means that subsequent trees will use more of the incorrectly predicted samples from previous trees. So as the forest grows the trees will become specialized, designed to predict subsets of the data rather than each tree making the best general prediction, as in the random forest approach. The second difference follows from the first, instead of simply averaging the predictions of the individual trees, the boosted ensemble combines the individual trees using a weighted approach where the weight of each tree is determined by its training error. Figure 17 shows a visualization of this process.

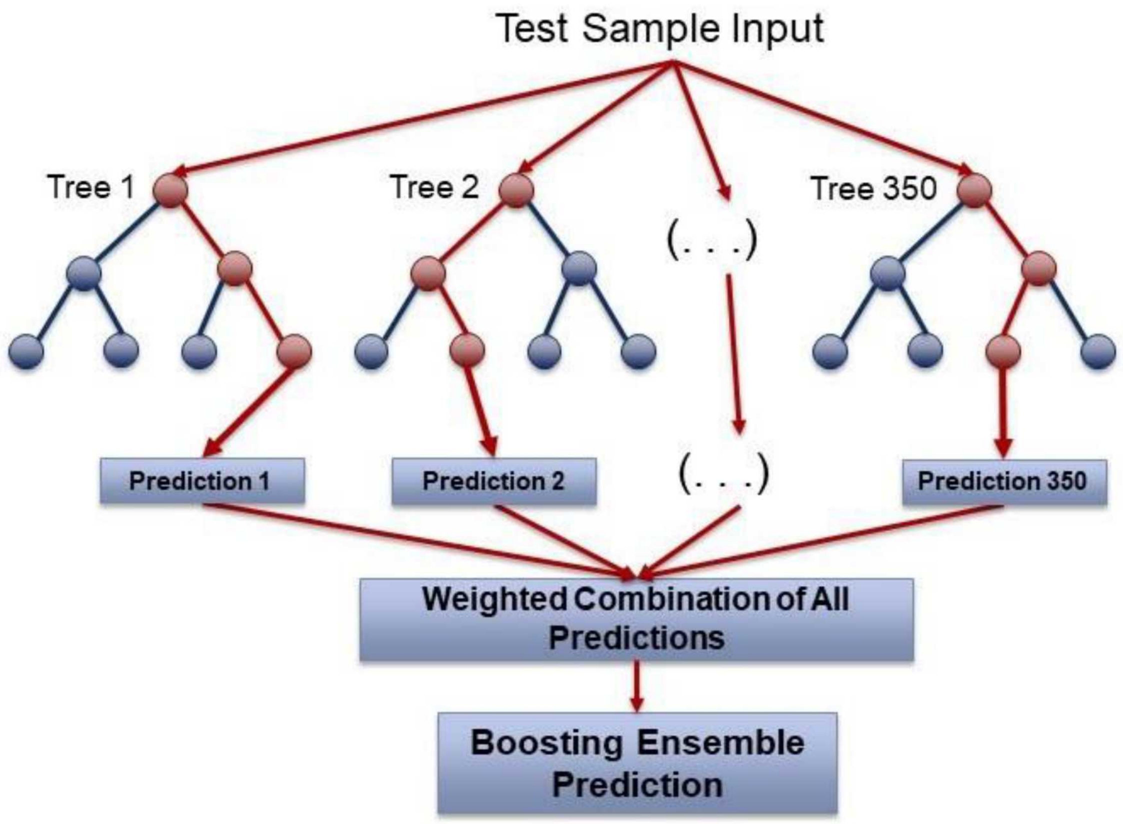


Figure 17 - Boosted ensemble process

4.2.2 Architecture

Figure 18 shows a progression of the trees grown in a boosted ensemble. Tree 1 looks very much like the example tree from the random forest in Figure 12. However, trees begin to change as they specialize in the samples that are difficult to predict. This boosted ensemble included 350 trees. That number was chosen using extensive experimental testing. As in the random forest case, the full 17 features were used to encourage a diversity of trees in the ensemble. For more details on the boosted implementation for this project please see the conference paper published on the results of the decision tree ensembles [34].

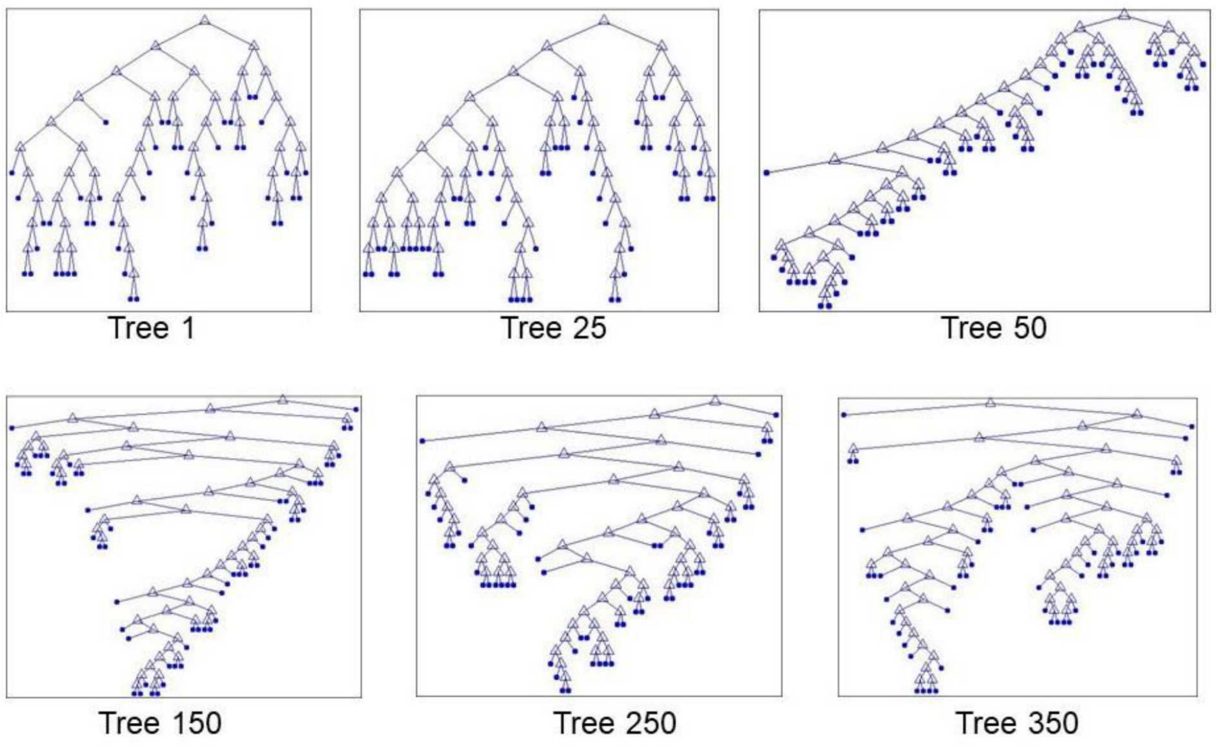


Figure 18 - Examples of trees in the boosted ensemble

4.2.3. Results

Figure 19 shows the results of the boosted ensemble in the Monte Carlo simulations. The Mean Absolute Error (MAE) is 2.6% and the 99.9th percentile error is 11.0%. Again, these error rates and distributions are quite similar to the random forest and neural network ensemble results. Figure 20 shows the results simulating different percentages of the year with the Random Sampling and Intelligent Sampling plotted for reference.

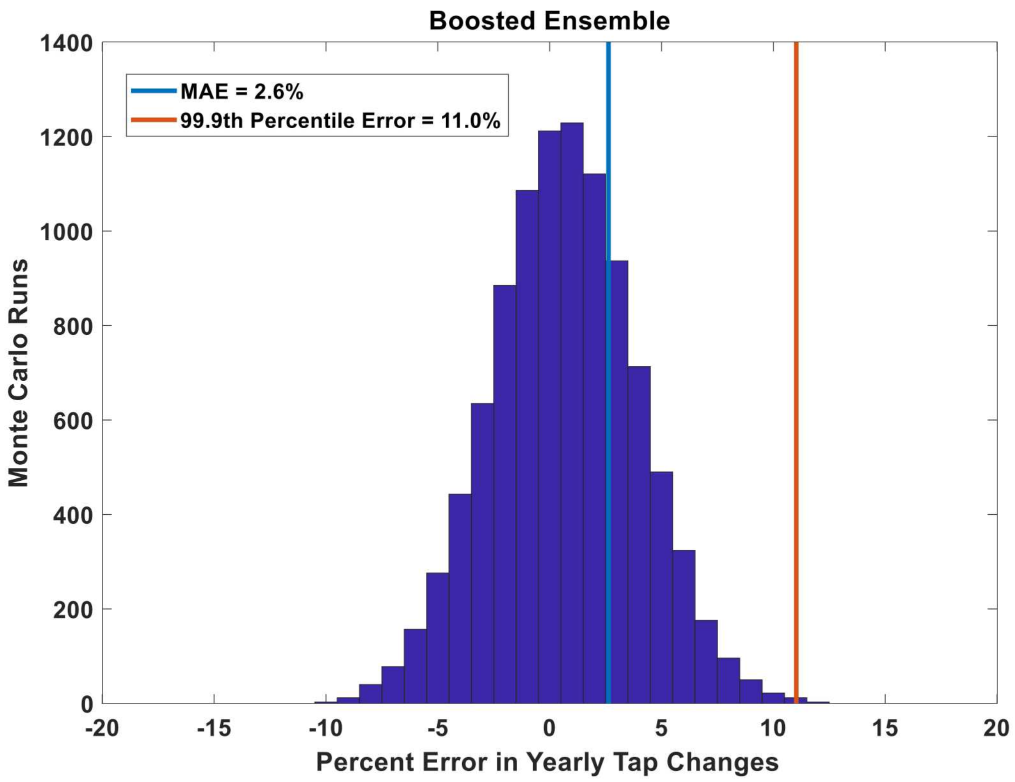


Figure 19 - Boosted ensemble results

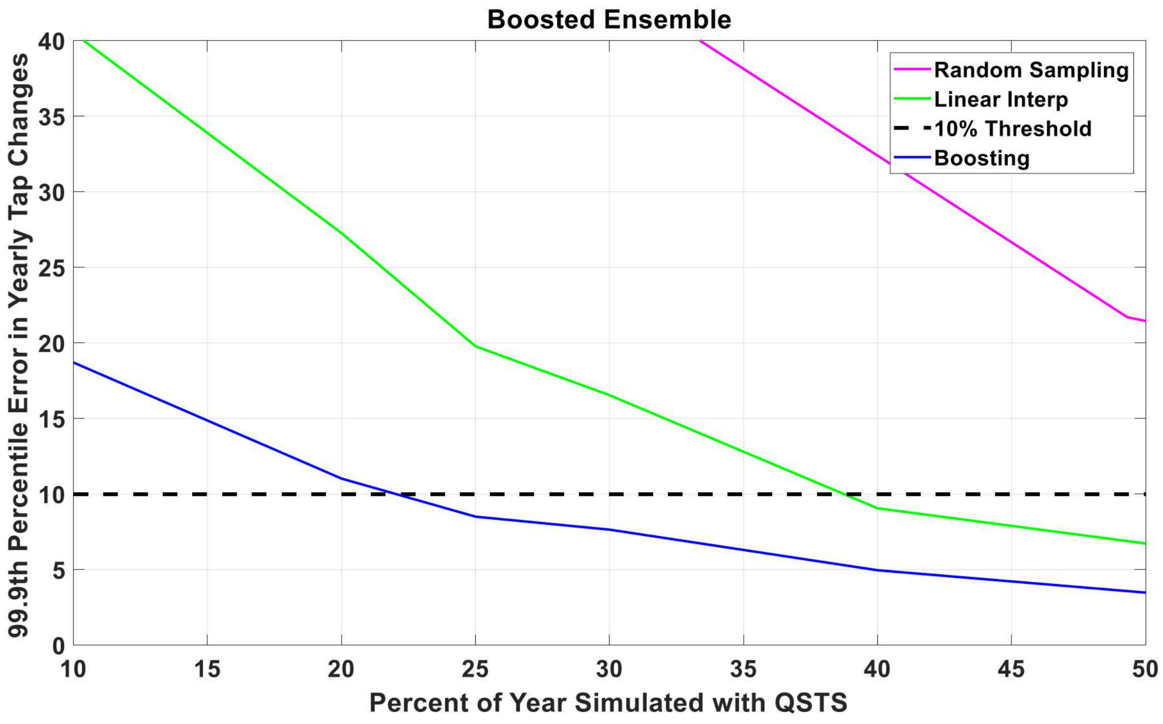
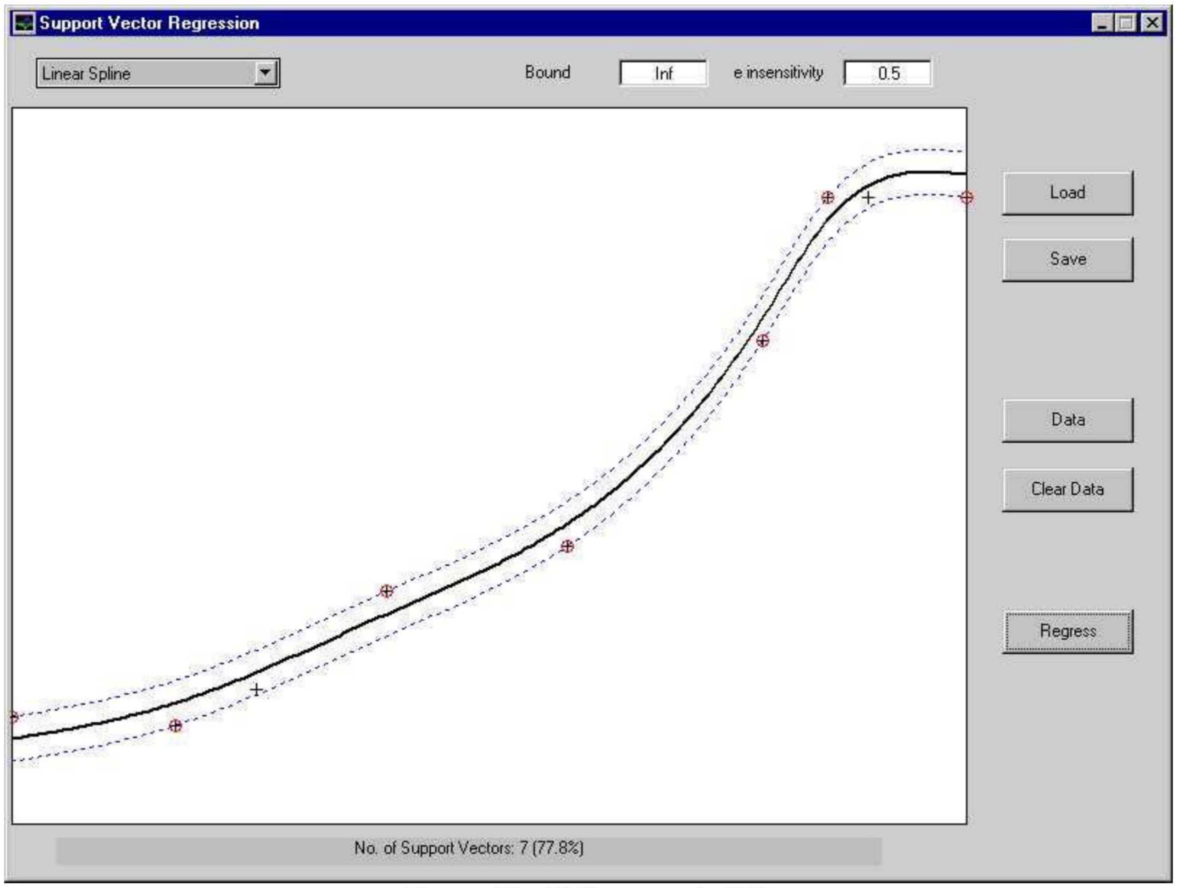


Figure 20 - Boosted ensemble results simulating different percentages of the year

5. SUPPORT VECTOR MACHINES

5.1. Overview

Support Vector Machines are a type of supervised machine learning developed by Vapnik [35]. In this paper we use the general term Support Vector Machine (SVM) to refer to regression version of the technique. SVM's use a (non-linear) kernel mapping to map the input into a higher dimensional feature space. Transforming the input in this way allows more flexibility in delineating between different types of samples. This allows the construction of a hyperplane (in the classification case) or a function approximation that may not have been possible in the original input space. In the regression case, the methodology attempts to create a function approximation that captures all the samples within some boundary around the approximation. Looking at Figure 21, the dotted blue line represents the boundary and the red circles represent the 'support vectors'. The support vectors, in the regression case, represent the samples that are on the boundary line (or sometimes outside); they are used to delineate the approximation, i.e. they define the boundary lines. Then a new sample can be compared to support vectors for the final SVM prediction, depending on where the new sample is compared to the support vectors a prediction can be obtained. For more detailed explanation of the SVM methodology see [35], [36], and [37].



Bound: Inf
e insensitivity: 0.5

No. of Support Vectors: 7 (77.8%)

Figure 21 - SVM example [36]

5.2. Architecture

The architecture of this implementation of support vector machines for regression was determined using a grid search of a subset of possible hyperparameter values using the Sandia High Performance Computing Cluster. A gaussian kernel was determined to be optimal. The parameter values were as follows Box Constraint: 215.44, Kernel Scale: 2.1544, and also normalizing the input features. This methodology used all 17 of the input features.

The SVM used an ensemble of 40 individual support vector machines. To construct the ensemble, a bagging approach was used, similar to the bagging used in the random forest methodology. A subset of the available training features was sampled with replacement for each of the members of the ensemble. The final ensemble prediction was generated by taking the average of each individual SVM prediction.

There are likely a variety of optimizations that could be made to this architecture with further study and experimentation. However, this was one of the final machine learning methodologies explored and research was discontinued for reasons discussed in the results section below.

5.3. Results

Figure 22 shows the results of the Monte Carlo simulations, using 20% of year for the QSTS simulations. The 99.9th percentile error is 16%, and the mean absolute error is 3.8%. The errors, although similar, are higher than the neural network, random forest, or boosted ensemble. Figure **23** shows simulations using different percentages of the year; with the current architecture and hyperparameters, ~30% of the year is necessary to simulate with QSTS to remain under the 10% threshold.

The decision was made to discontinue the tuning of the SVM model due to the fact that the results in the neural network case, Figure 9, random forest case, Figure 15, and boosted ensemble case, Figure 19, were so similar to each other. It is our hypothesis that, although the SVM errors are higher than those other three methods, with appropriate tuning of the model, the results would likely be quite similar to those results. Further discussion of why that might be the case can be found in Section 8.1. Comparison Between ML Methods Analyzed. Therefore, the research was stopped at this point without further tuning of the model.

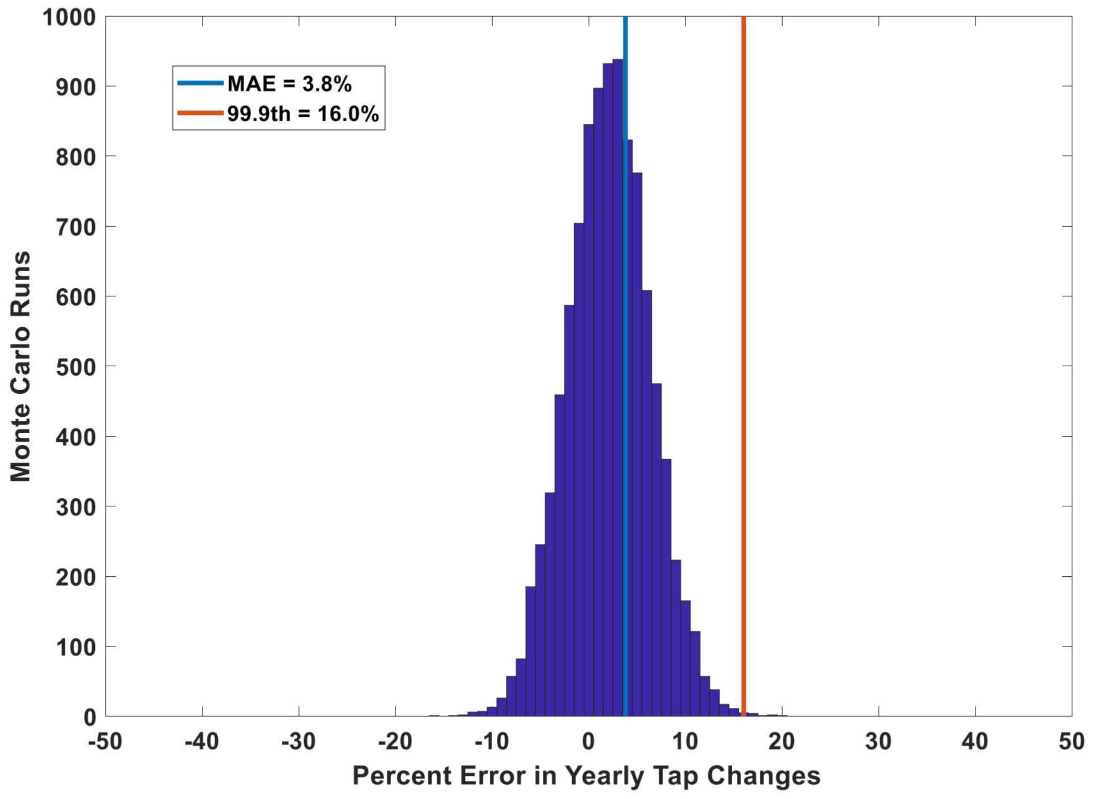


Figure 22 - SVM results histogram

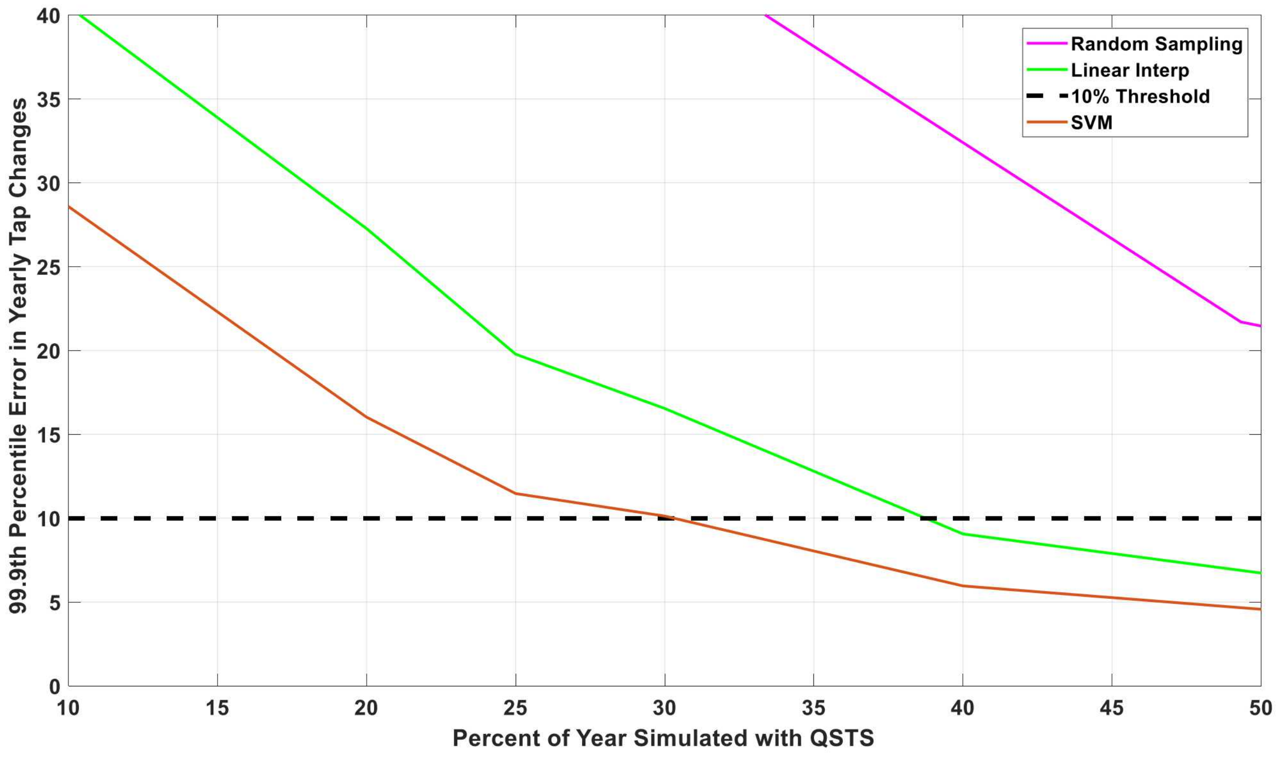


Figure 23 **- SVM results simulating different percentages of the year with QSTS**

6. DEEP LEARNING

6.1 Overview

"Deep Learning" covers a wide range of techniques and methodologies gaining increasing popularity in recent years, particularly in the field of image recognition. Deep learning is often described in terms of 'layering' different techniques. For example, in the following section the 'fully connected' layers are equivalent to the feed-forward neural network architecture discussed previously, and here they are components of the deep learning architecture. For this project convolutional neural networks, a supervised learning technique, were chosen for the deep learning architecture. The architecture used here is based on the AlexNet architecture from [19].

Unlike the other methodologies that used the statistical features discussed in the Section 2.3 Feature Selection, the convolutional neural network takes the raw data as input. One of the main advantages of the deep learning approach is the 'feature extraction' ability of convolutional neural networks. There is no guarantee that the features chosen in Section 2.3 are optimal or exhaustive. The purpose of the convolutional layers is to extract meaningful features from the raw data before the prediction step. So, the input to the network is the 1-second resolution load and PV profiles, and the intelligently sampled training set is further divided into an actual training set and a validation set, similar to the neural network ensemble.

6.2 Architecture

This network architecture, adapted from AlexNet [19], consists of two convolutional layers each followed by a ReLu activation function and Cross Channel Normalization, that is followed by a dropout normalization layer, then two fully-connected layers, and finally a regression layer. This architecture is a scaled back version of the original because the dataset is smaller than the one used for AlexNet, as well as for time considerations.

Deep learning in general requires significantly more data than other machine learning methods. One-hour periods were used to give the algorithm more distinct samples. While this does not increase the meaningful size of the dataset or the total information value, it does allow more distinct opportunities for the back-propagation algorithm to work. However, it is not clear that this was an effective technique, and it is likely that using the two-hour periods would have performed similarly. Two methods for data augmentation were attempted in addition to using the un-augmented dataset. Data augmentation is the technique of taking an existing dataset and altering the samples in some way to create new data samples. The transformation must be done in such a way as to preserve the essential structure of the sample. Some examples in the image recognition context are cropping the image differently, altering pixel intensities, applying 'warping' filters to the image, etc. In each of these cases, the pixel values are different, and so the sample is in effect a 'new' sample for the deep learning network, however the essential content of the image is unchanged. However, in instances such as this where the inputs are load time-series and PV time-series, ensuring that the essential structure and content is unchanged and that the resulting profiles reflect real world possibilities is much more difficult since they cannot be visually inspected.

The first method that that was attempted divided the one-hour periods into a first half and second half and randomly switched the halves among the training samples. The intuition behind this method being that it is already known how many tap changes occur in each of the halves and previous research has shown that only a small amount of error is introduced by running the QSTS period on isolated periods of the data [25].

The second method was based on the Synthetic Minority Over-sampling Technique (SMOTE) from [38]. This technique uses an auto-encoder to create a feature-space representation of the training data. Then to augment the training data the training samples are fed into the auto-encoder to obtain the feature-space representation and grouped by similarity (in this case by number of tap changes in the period). Two similar samples are subtracted from one another (still in the feature-space representation) and a random percentage of the difference between the two samples is added to the first sample to produce an augmented sample. The intuition behind this technique being that the new augmented sample will be 'between' the two samples that are similar and thus should have similar behavior.

The network was trained for a large number of epochs and the epoch with the lowest validation error was chosen as the final network. One epoch is defined as one pass through the training data which is broken up into batches, in this case of 128 samples per batch. A single epoch uses each of the training samples once for optimization and then the samples are randomized and used again. The chosen network was then used to predict the remaining periods (the testing set). Given the long training times required for this method and the overall poor results, the Monte Carlo simulations were not run for the deep learning architectures.

6.2.3 Results

Figure 24 shows a plot of the training root mean squared error versus the validation root mean squared error for one run of the convolutional network using the un-augmented dataset. Each iteration represents one training batch of 128 samples. This means that each of the 128 samples is used as input to the network, the errors values are calculated, then the suggested weight adjustments are calculated, but then update step uses the average of the weight adjustments for all 128 samples. The actual weight update occurs only once per batch. Looking at Figure 24, the minimum for the validation error is at iteration 5,800 as so that version of the network was used to predict the remaining samples. The final percent error in the yearly tap changes was ~16.5%.

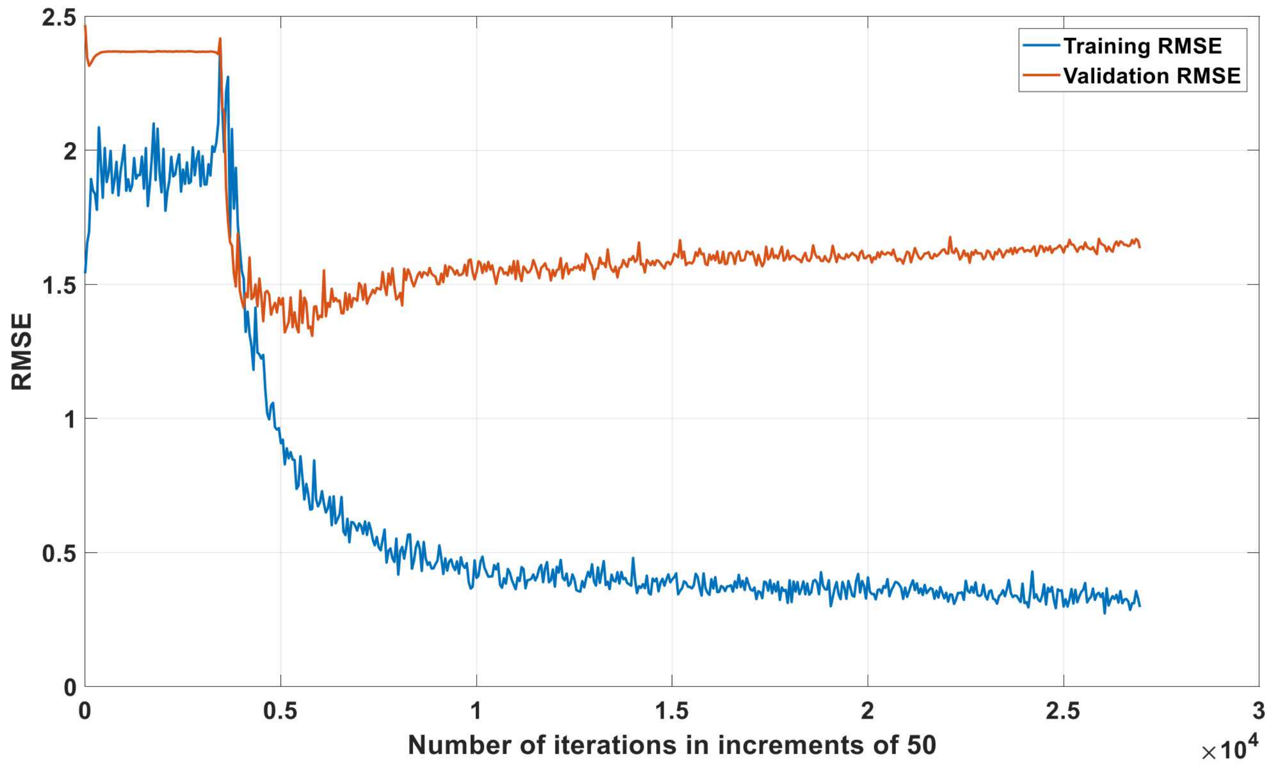


Figure 24 - Deep Learning comparison of training RMSE and validation RMSE

Figure 25 shows the training RMSE and validation RMSE for the network run using the augmented data – swapping method. The minimum for the validation error is at ~41000 iterations and using that network produced a yearly error in tap changes of ~15%.

Figure 26 shows the training RMSE and validation RMSE for the network using the augmented data – SMOTE method. The minimum validation error is at ~72000 iterations and the network from that iteration produced a yearly error of ~24.5%.

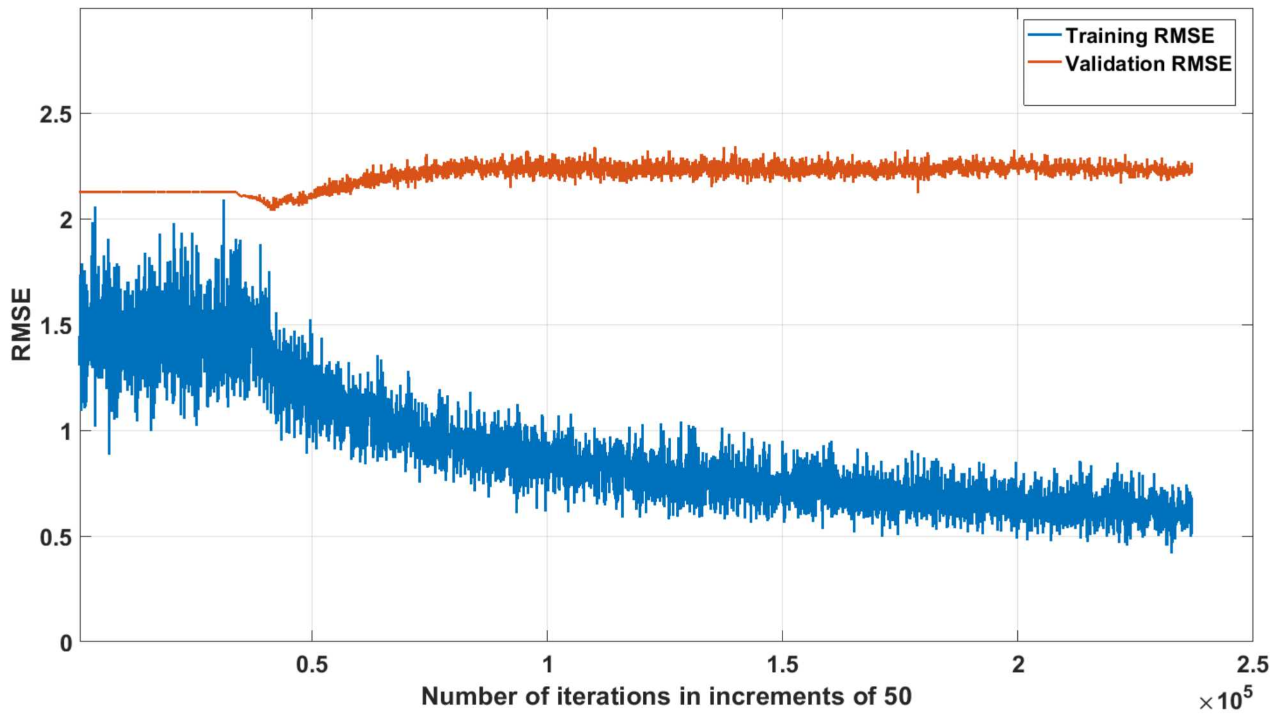


Figure 25 - Deep Learning with augmented data using the swapping methodology

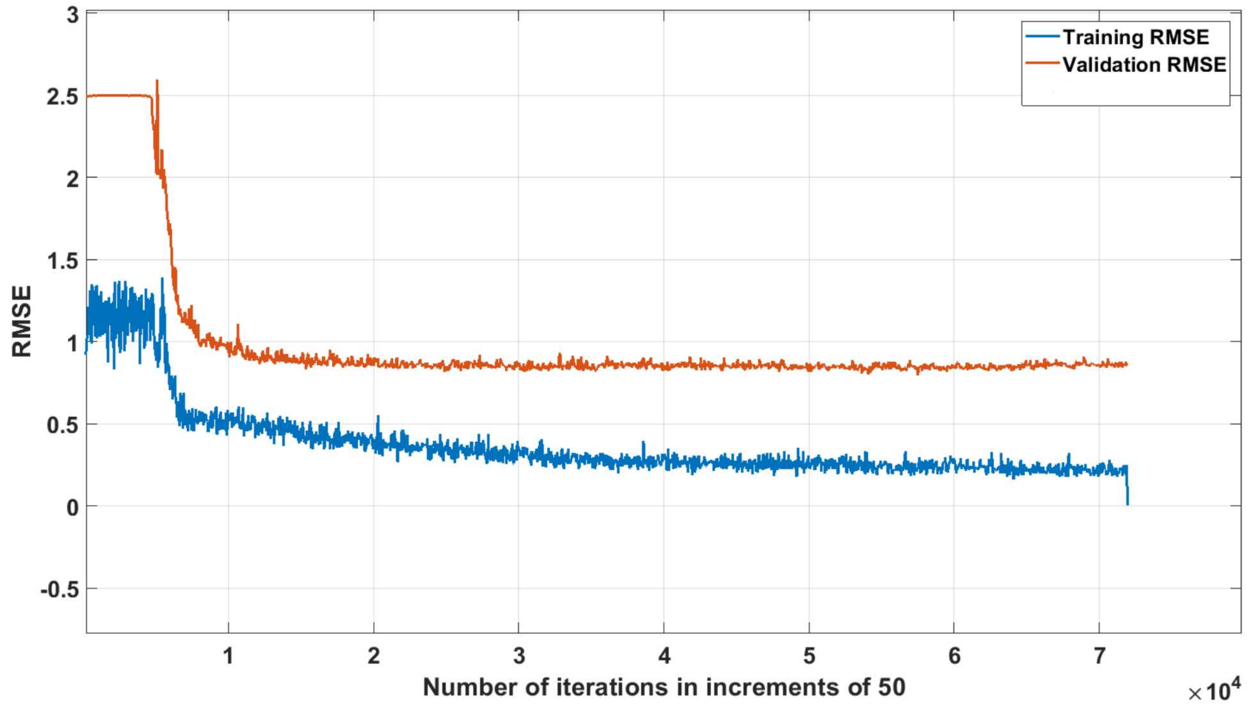


Figure 26 - Deep Learning with augmented data using the SMOTE methodology

These results are not favorable when compared to the results from the other machine learning techniques, ~10.8% 99.9th percentile error and ~2.6% mean absolute error for a neural network ensemble Figure 9 and ~11% 99.9th percentile error and ~2.6% mean error for decision tree ensembles Figure 15 and Figure 19. These deep learning results should be considered a rough, preliminary investigation of deep learning, convolutional neural networks, and data augmentation within this context. There are several places where certainly optimizations could be made to perhaps improve the performance. The network architecture is unlikely to be optimal and could almost certainly be improved given time and effort. There are many adjustments that could be made within the AlexNet structure and there are many other options altogether for deep learning architectures (Long-Short Term Memory modules may be an appropriate choice). Neither data augmentation method produced satisfactory results. However, the augmentation using the SMOTE methodology has room for tuning that would likely produce better results.

The decision was made to discontinue research in deep learning avenues at this early stage for two key reasons. The focus of this QSTS project is to improve the speed of a year-long QSTS simulation, and deep learning is clearly unsuited for that objective. Training of this architecture was conducted on the High-Performance Computing Cluster and still took hours to complete. Changes to the network architecture that would improve performance would likely increase, rather than decrease the time required. This was another reason to start with a scaled back version of AlexNet. The second reason is that deep learning requires more data than other options, and the nature of the QSTS dataset is that there is only a limited amount of data. Pursuing the data augmentation might be worthwhile in other contexts but in this case, there is no clear way to use deep learning due to the time constraints.

7. ENSEMBLE OF ENSEMBLES

There is a variety of recent research investigating the efficacy of ‘ensembles of ensembles’ as a hybrid machine learning technique [20], [39]. An ensemble of ensembles is simply more than one machine learning technique combined to form a single prediction. For example, a neural network ensemble, a random forest, and an SVM ensemble could be combined to form a single ensemble. In this case, an ensemble consisting of the neural network ensemble and the random forest together did produce an improvement in the overall prediction. Figure 27 shows the histogram of the Monte Carlo runs, with a 99.9th percentile error of 9.9%, an improvement over the best so far of 10.8% from the neural network ensemble in Figure 9. Figure 28 shows the ensemble of ensemble results simulating different percentages of the year; this method actually remains below the 10% threshold using 20% of the year. This is significant because using 20% of the year is a 5x overall speedup compared to using 25% of the year which is a 4x overall speedup.

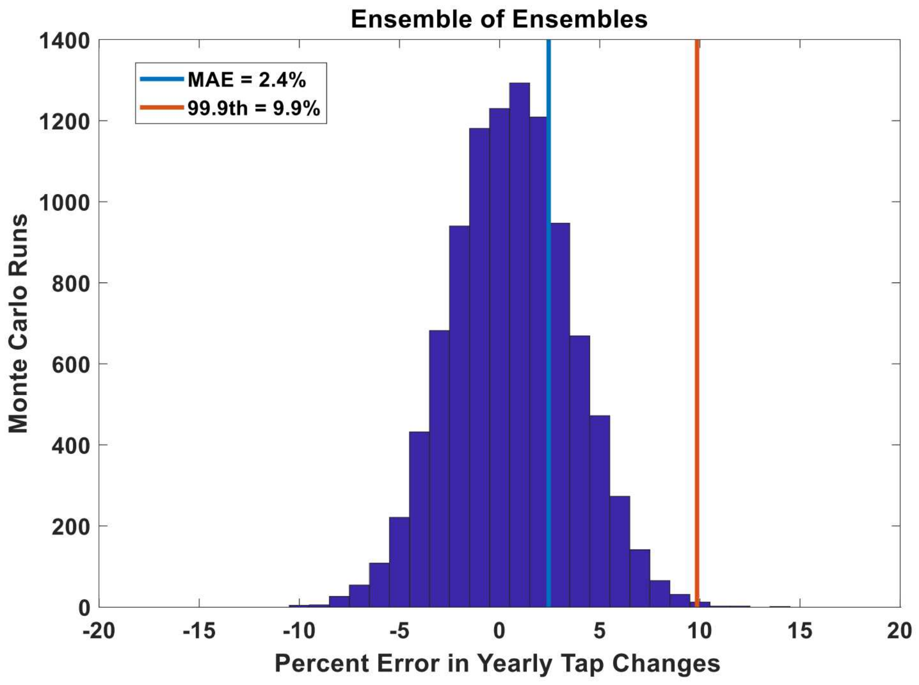


Figure 27 - Ensemble of Ensembles histogram

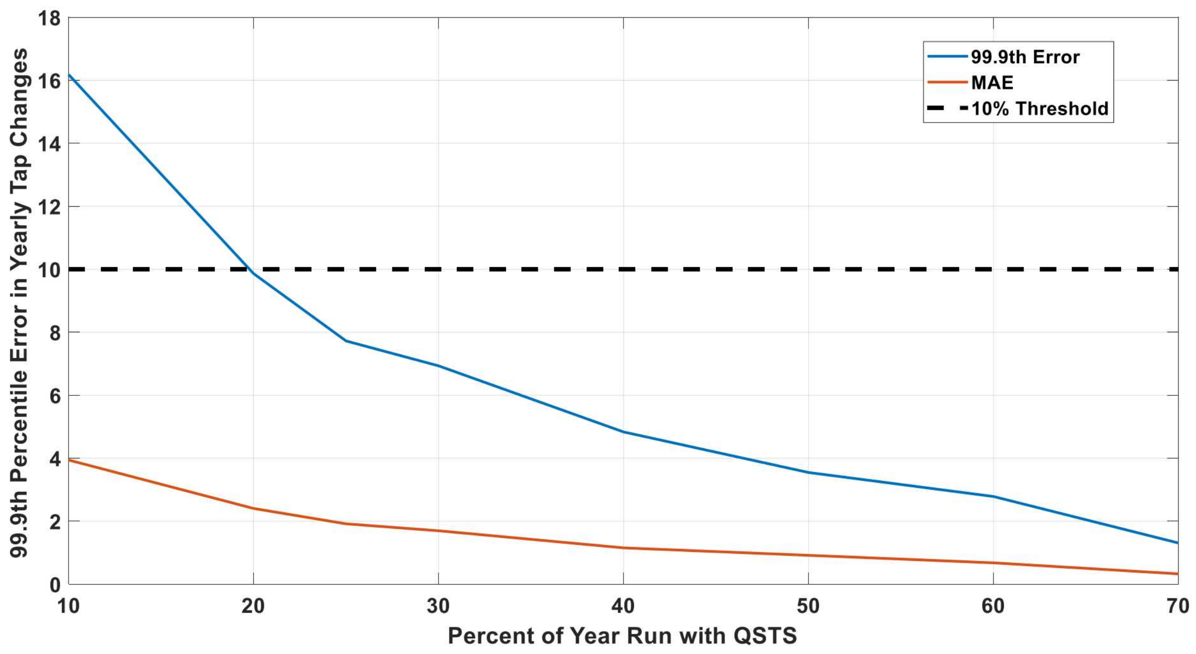


Figure 28 - Ensemble of Ensemble simulating different percentages of the year

The best results were obtained using only the neural network ensemble with the random forest. The two separate predictions were averaged to obtain the final ensemble of ensemble prediction. Other combinations were also experimentally tested. Adding in the boosted ensemble made no difference to the results, and the neural network ensemble with the boosted ensemble produced the same results as the neural network ensemble with the random forest. It seems to make sense that the two decision tree ensemble methods contain similar information and the neural network contains slightly different information, so adding a second decision tree ensemble does not aid in the overall prediction. It does imply however, that the neural network and the decision tree ensembles were solving the prediction problem in different ways and the combination is beneficial to the overall prediction which is an interesting conclusion. Since the SVM model was not performing as well as the other methods when research was stopped, it was not included in the ensemble of ensemble analysis. That would be an interesting extension of this research to discover if a well-performing SVM would add to the ensemble of ensembles accuracy. There are many variations on the ensemble of ensembles technique that were not explored here. For example, often the ensembles are combined using a weighted average of the members of the ensemble. These weights can be determined in a variety ways. For example, different weights can be given to different members based on some classification of the input before running the machine learning algorithm [20]. For instance, perhaps some members of the ensemble perform better during certain weather conditions than others and should be weighted according to the weather for the specific input sample.

8. COMPARISON AND DISCUSSION

8.1. Comparison Between ML Methods Analyzed

Each of the machine learning algorithms that were investigated over the course of this phase of the project is considered to be a supervised learning technique, meaning that they are trained using labeled data. For a discussion of a potential unsupervised learning option see the discussion of clustering techniques in Section 8.3. Discussion of Results and Other Possible Machine Learning Methods. All machine learning techniques require ‘significant’ amounts of training data, deep learning in particular has a huge data requirement, but precisely what the data requirement is remains poorly defined and is highly dependent on the problem context. Here the amount of data available is constrained by the speed vs accuracy tradeoff described in Section 2.4.2 Speed versus Accuracy.

Neural networks, random forest, and the boosted ensemble all produced extremely similar results (and we hypothesize that a fully-tuned SVM model would also converge to comparable results), and this was not an expected result. For a side by side comparison, see Figure 29, Figure 30, Figure 31, and Figure 32. In general, these techniques tend to perform differently depending on the problem, and one would not expect the results to be so similar [12]. The similarity suggests that there is a limiting factor at work that is unrelated to the choice of algorithm. We hypothesize that this similarity is due to the data constraints. The speed versus accuracy tradeoff that is present in this work dramatically constrains the amount of data available. There are 4368 two-hour periods in the yearlong dataset, and taking 25% of those gives a training set of 1092 samples. This is a small sample set compared to many common machine learning datasets; MNIST [40] (basic ML benchmark dataset) contains a training set of 60,000 handwritten digit samples, and the ImageNet dataset (deep learning benchmark dataset) contains millions of training samples [41]. With more training data, it is reasonable to expect that there would be more significant differences between the algorithms’ results. This was the primary reason for stopping the research into SVM earlier than the other methods, and also for not investigating other possible algorithm choices. Looking at Figure 29 and Figure 30, you can see the comparison of algorithms’ error as more of the year is simulated with QSTS.

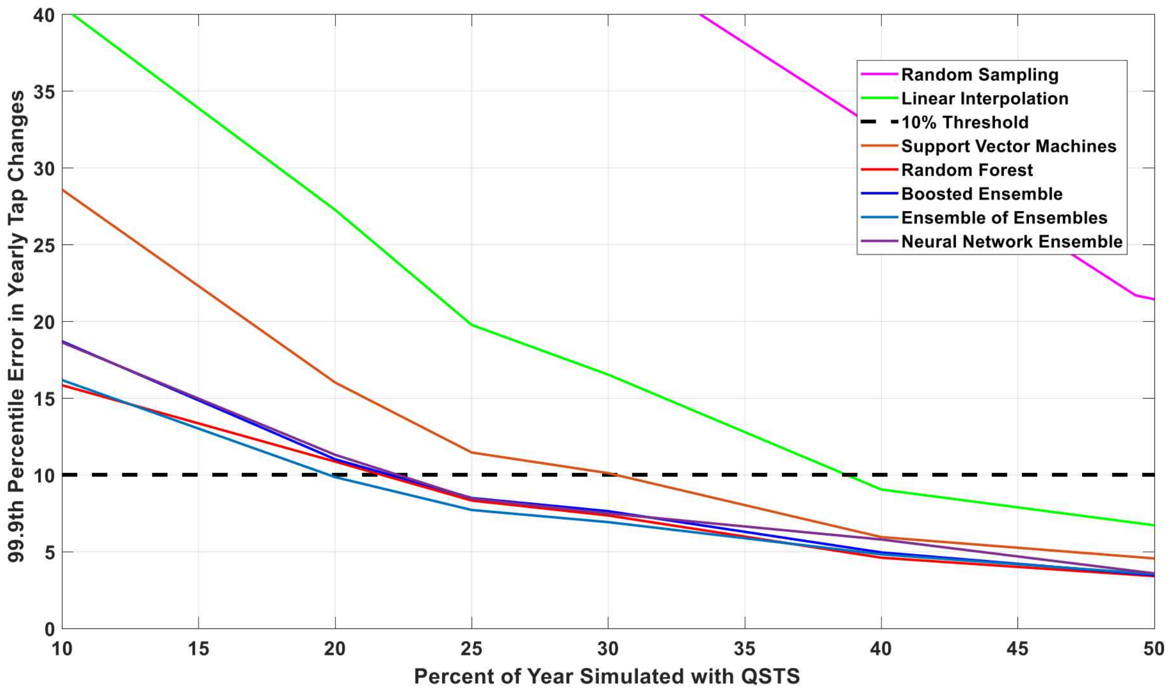


Figure 29 - Comparison of algorithms simulating different percentages of the year with QSTS

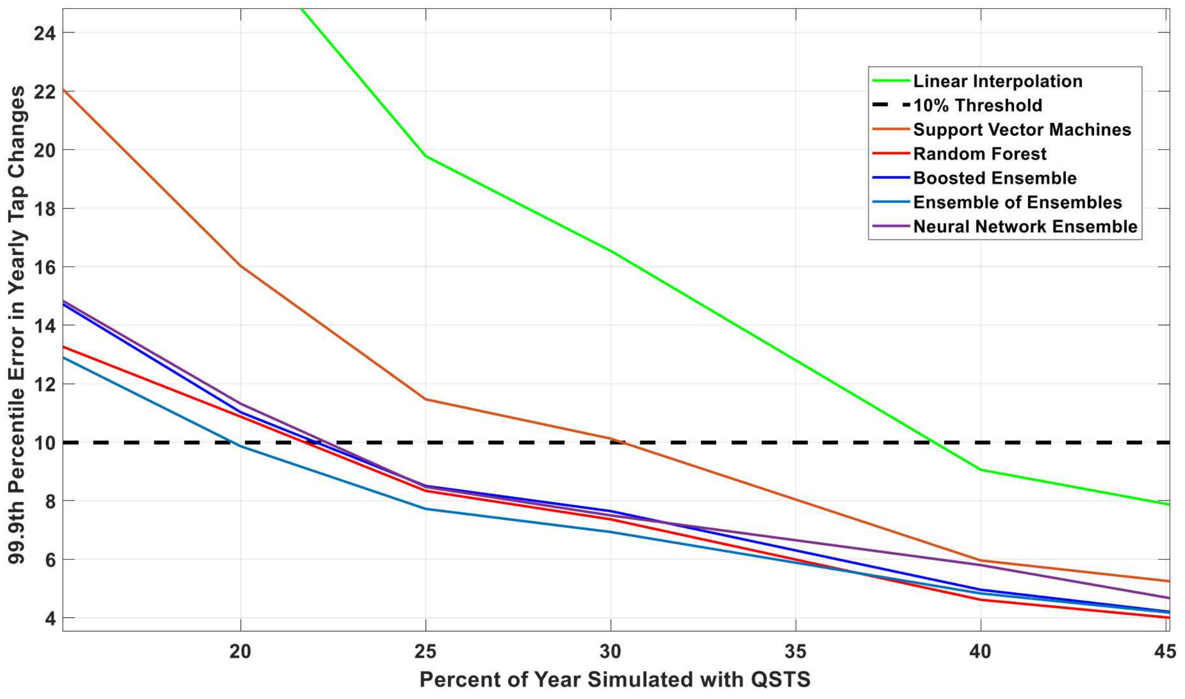


Figure 30 - Comparison of algorithms (zoomed in version)

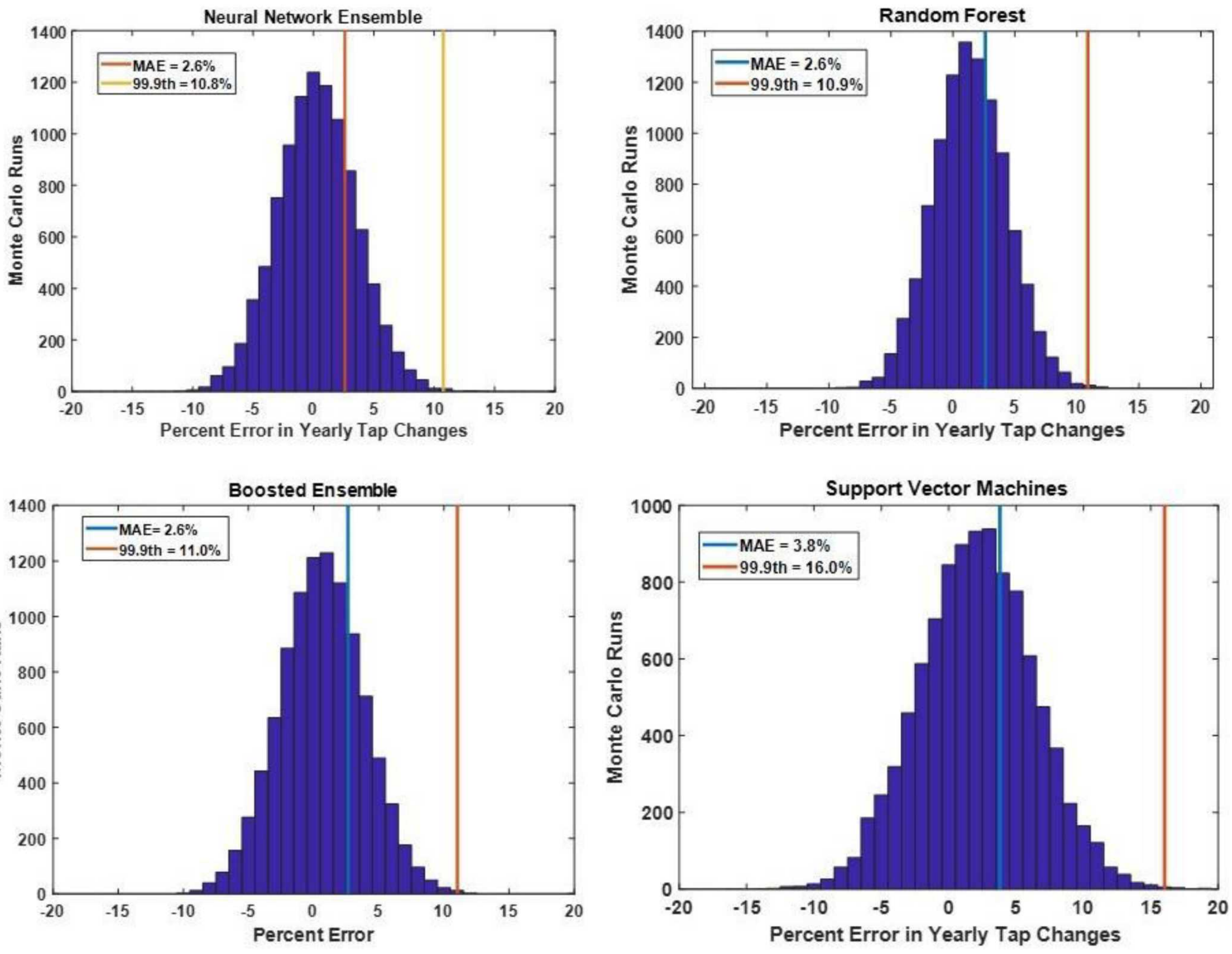


Figure 31 - Comparison of algorithms histogram

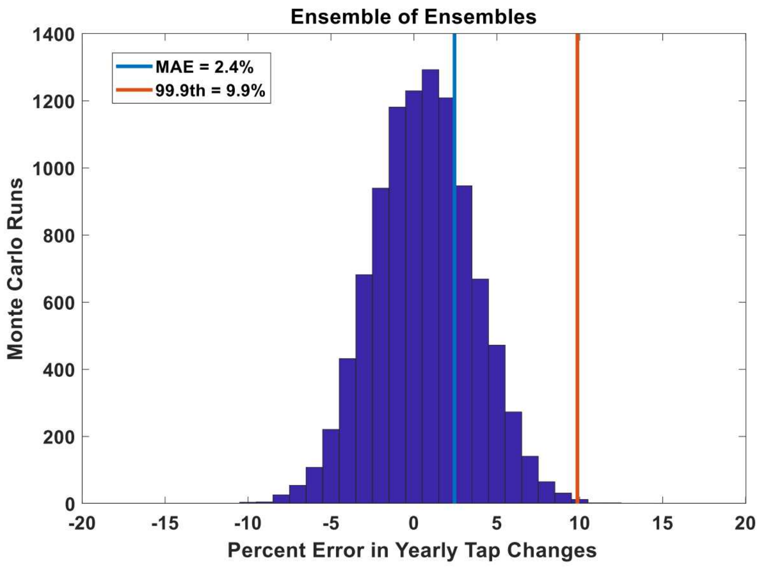


Figure 32 - Ensemble of Ensembles histogram

All of the timing information that follows is based on a laptop computer with the following hardware: Intel® Core™ i7-3687U CPU @ 2.10GHz 2.60 GHZ with 8.00 GB RAM. Table 2 provides a summary of the timing results. Note that the brute force time for this dataset is 30 minutes. The boosted ensemble provides the best single-method time of ~7.5 minutes (75% reduction) and the ensemble of ensembles produces the best overall time of ~7 minutes (80% reduction). The percentage of the year simulated with QSTS in Table 2 reflects the percentage required to stay under the 10% error threshold. Also note, that due to the simulation time required, a full sweep of the percent of the year simulated with QSTS was not conducted for the deep learning and so the ~25 is simply an estimation.

Figure 33 shows a comparison of the machine learning portion of the timing (without the QSTS simulation portion). Although random forest and the boosted ensemble have nearly identical prediction results, the boosted ensemble is faster at 4 seconds, compared to 12 seconds for the random forest. The neural network takes 25 seconds and the support vector machine takes 21 seconds. The deep learning on the far right of the graph extends up to 75,633 seconds, or 21 hours for the machine learning portion.

Figure 34 provides the full picture for each methodology timing. The machine learning portion is still in red and the purple represents the time required for the QSTS simulation. Clearly the QSTS simulation time dominates, regardless of the machine learning algorithm chosen (with the exception of the deep learning method). Note that due to the small machine learning time for the boosted ensemble (4 seconds), the red portion does not show well on the figure, but there is 4 seconds of machine learning on top of the QSTS time. For clarity, the abbreviations in the Figure 33 and Figure 34 are 'Lin Interp': linear interpolation from the intelligent sampling, 'NN': neural network ensemble, 'SVM': support vector machine, 'EE': ensemble of ensembles, 'RF': random forest, 'Boost': boosted ensemble, 'DL': deep learning, and 'Brute Force': standard, full-length QSTS simulation.

Table 2 - Algorithm times summary

Algorithm	Percentage of the year simulated with QSTS	ML Time (seconds)	Total Simulation Time	Times Faster
Brute force	100%	-	30 min	-
Non-ML Sampling	40%	-	12 min	2.5
Deep Learning	~25%	75,633 (21 hours)	21 hours	-
Support Vector Machine	30%	21	~8 min	3.3
Neural Network	25%	25	~8 min	4
Random Forest	25%	12	~ 8 min	4
Boosted Ensemble	25%	4	~ 7.5 min	4
Ensemble of Ensembles	20%	37	~7 min	5

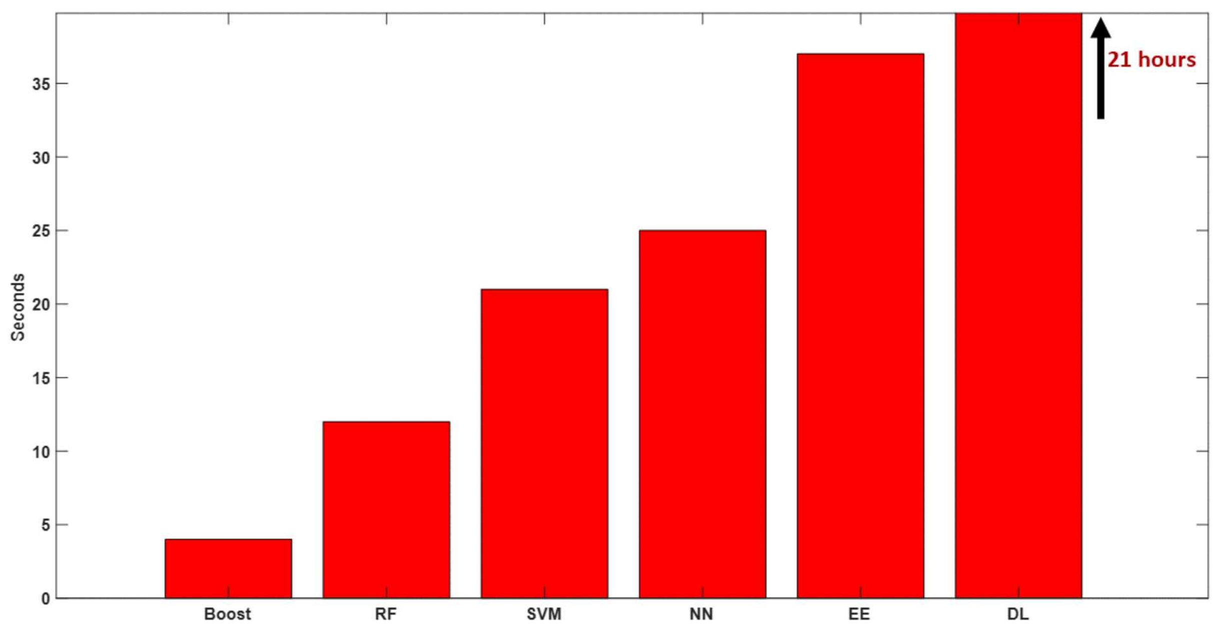


Figure 33 - Machine Learning timings

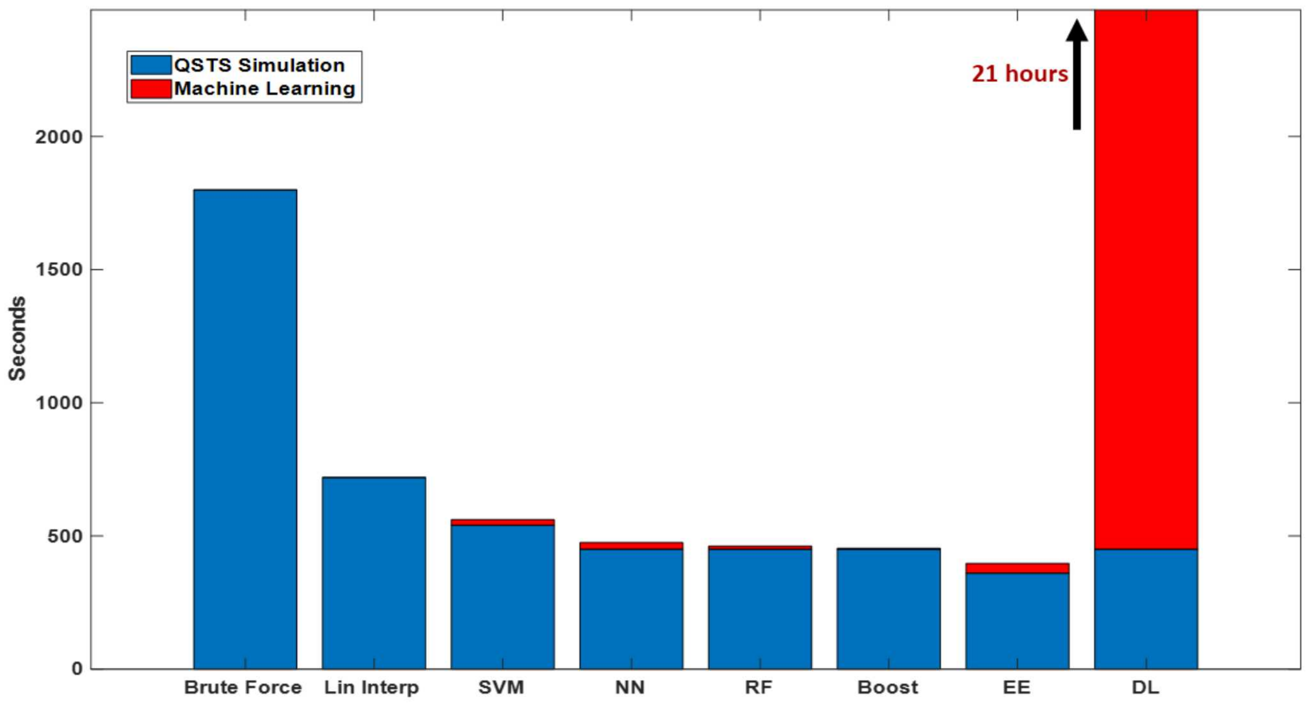


Figure 34 – Algorithm computational timing comparison

8.2. Comparison to Other Ongoing QSTS Research

While this report focused on using different types of machine learning to speed up distribution system QSTS simulation, there has been significant research in this area recently for other ways to make the QSTS simulation faster [42]. For example, a reduced order model of the distribution system can be used to simplify the computations while maintaining an equivalent circuit model in time-series [43]. More computational power can be brought to solving QSTS simulations using parallelization [44], either separating the system spatially into subcircuits [45], [46] or temporal separations with parts of the year going to each processor [47], [48]. QSTS algorithms traditionally step forward in time using a fixed time-step, but new variable time step algorithms can focus the computational effort on specific variable times and even backtrack for large system events for additional speed [10], [49]–[51]. In a similar fashion, discrete event-based simulation techniques can be used for QSTS simulations by applying linear regression models and voltage sensitivities to determine the next event and fast forward in the time-series [52], [53]. Novel QSTS vector quantization algorithms have shown massive speed improvements by clustering the state space of power flow solutions and using a look-up tables for previously solved power flows [54]–[56]. Current research is working on combining various of the methods, such as variable time-step with vector quantization [57] or temporal parallelization with circuit reduction.

It can be quite challenging to do a straight forward comparison of the speed improvements and errors of each of the rapid QSTS methods referenced. For example, the results are specific to the distribution system being analyzed for characteristics such as topology, PV penetration, voltage regulator settings, number of loads with unique timeseries profiles, and many other things. Also, in a quickly moving research field, each of these fast QSTS algorithms is constantly changing, improving, and becoming faster. For certain algorithms, such as parallelization, the speed improvements are also directly related to the computational hardware and the number of cores available to the user. Even with these caveats, a rough comparison of the speeds and errors is shown in Figure 35. Many of the new QSTS algorithms are performing more than 20 times faster than traditional brute-force QSTS simulations, and vector quantization and event-based simulations are around 200 times faster. In comparison, as summarized in Section 8.1, the machine learning methods presented in this report are roughly 4-5 times faster. The machine learning timings shown in Figure 35 are from the ensemble of ensembles results in Figure 30.

The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

Figure 35 - Comparison of rapid QSTS methods

8.3. Discussion of Results and Other Possible Machine Learning Methods

There is an extensive list of other possible machine learning algorithms that could be investigated for this purpose, and there are a variety of ways this research could be continued, or perhaps more appropriately, expanded in a different direction. This section will discuss a selection of other algorithms that were not explored in this research, a brief discussion of areas into which this research could be expanded, and finally a discussion of why this portion of the project was concluded.

8.3.1 Other Machine Learning Methods

One type of algorithm which was not explored during this research is unsupervised learning (learning without labeled data). All of the training data for the preceding algorithms was labeled with the correct number of tap changes for the period. An alternative approach would be some type of clustering, K-means, hierarchical clustering, etc. For an overview of clustering techniques, see [58]. This could be framed in terms of clustering the intelligently sampled data and then assigning the remaining 'test' periods to a cluster. Although, clustering is generally considered unsupervised learning, strictly speaking this would not be fully unsupervised learning in the sense that after clustering the training data (without using the labels) each cluster would then be assigned a 'cluster label' based on the tap changes for the samples within each cluster. Radial basis networks are another possible choice. They are a supervised learning technique within the neural network family [59]. There are multitude of techniques within the 'deep learning' family, however none of them show promise in overcoming the fundamental difficulties encountered for this project, explained in Section 6. Deep learning. There is not necessarily any clear reason to believe that any of these other techniques would perform better than the ones investigated over the course of this research given this project's set of constraints.

8.3.2 Future Avenues

Perhaps more worthwhile are a few considerations for possible different directions to take this research. Returning to the idea of deep learning, a key advantage of deep learning is the ability to extract features from raw data. It might be possible to use deep learning techniques as a feature extractor with a different end goal in mind than for this project. Potentially, by taking a large enough number of PV and load profiles, better features could be extracted using deep learning than simply using human-constructed statistics. These load and PV profile features could then be used for any number of other techniques or goals.

There are possibilities for the data augmentation as well. As the demand for data grows and more and more analysis is required in distribution system research, data augmentation might be a solution. Critics of data augmentation argue that because the data is generated rather than observed that is not worthwhile. However, many of the advantages of data augmentation come in 'completing' a real-world dataset; a real-world dataset will never cover all possibilities or edge-cases and data augmentation can go a long way in making a dataset more complete, particularly if there is a way to verify that the augmented data is reasonable in the real-world. With these types of QSTS analyses it is possible to run the true simulations to verify if the data augmentation is working correctly. Autoencoders [38], [60], variational autoencoders [61], and generative adversarial networks [62] are all examples of methods used for data augmentation.

As discussed in 2.3 Feature Selection, the subset of features that are used as input to the machine learning algorithms is both critical and non-trivial to determine. There is an extremely large range of other possible features that could be explored for this problem. This research chose to use the standard deviation of the previous two weeks of data, one feature for the PV data and one feature for the load data. It is possible to tune the timeframe for adding information about what occurred previously, or include more than one data point for different timeframes, for example perhaps adding information both about the previous period as well as the previous two weeks would be beneficial. Another approach would be to add information specific to the feeder, for example the location of the PV installations, the delay on the voltage regulators, etc. There are also possibilities for 'automatic' feature extraction, for example using an autoencoder to extract the features and use those features as input to the machine learning algorithms in the place of the statistical features. These are just a few of the possible avenues, and we leave the further exploration of this issue to upcoming research.

The amount of data present for this research and the speed versus accuracy tradeoff was found to be a limiting factor for the machine learning algorithms investigated here. However, this research was bounded by looking at a single year and a single distribution system feeder at a time. There might be better opportunities for using machine learning algorithms in a situation where there is more data available. The other techniques mentioned in the section above certainly outperform the machine learning in terms of timing on the single feeder scale, however another interesting question might be, "How do these techniques scale to thousands of feeders?". Machine learning techniques might be a technique better utilized from a big picture perspective, thousands of feeders, entire distribution systems, all customers in a region, etc. Those are the types of questions that might produce interesting results by using machine learning algorithms.

8.3.3 Discussion of Results

There are three primary reasons that this phase of research for rapid QSTS has been concluded. Under the specific constraints of this project, although achieving ~4-5x speedup when compared to a brute force solution of the QSTS simulation, it seems clear that some of the other rapid QSTS techniques are performing better than the machine learning algorithms investigated here. Vector quantization, for example, from Figure 35 achieves ~200x speedup compared to brute force. From Section 8.1. Comparison Between ML Methods Analyzed, it is clear that the amount of data available is a limiting factor for the machine learning algorithms. Also, this research has limited itself to predicting the number of tap changes in a period, and that is just one possible desired metric from the QSTS analysis. Under this methodology, separate networks/ensembles would need to be trained to predict other metrics. Some of the other rapid QSTS techniques described above are able to obtain results during the simulation for any desired metrics, such as capacitor switches, bus voltages, time that components exceed their thermal rating, and many others. For all of these reasons, this portion of the rapid QSTS project has been concluded.

9. CONCLUSION

This project investigated several machine learning algorithms for the purpose of decreasing the time necessarily for a yearlong QSTS analysis to determine the number of voltage regulator tap changes in year given specific PV and load profiles. A representative portion of the year was selected, a machine learning algorithm trained on that data, and the remainder of the year predicted, while remaining under the 10% yearly error threshold. The ensemble of ensembles methodology achieved a 5x speedup compared to the brute force approach, and the best single-method, boosted decision tree ensemble, achieved a 4x speedup compared to the brute force approach. The speedup achieved depends on the amount of the year that is simulated using QSTS and used as training; there is a tradeoff between the overall speed and the overall accuracy of the results. The methods that were tested produced similar results which we hypothesize is due to the speed versus accuracy tradeoff resulting in the amount of training data being the limiting factor rather than the machine learning techniques themselves. The deep learning techniques proved too time-intensive for use in this situation, regardless of the amount of data used. Further increases in speed for these techniques are limited due to the data limitations, and other techniques for fast QSTS show more promising results. Figure 36 (same figure as in Section 8.1) visually summarizes the methods investigated and the time that was achieved for each method, and Table 3 (same figure as in Section 8.1) provides a textual summary. Due to the randomness involved in selecting the training data and in the machine learning techniques, the 99.9th percentile error was reported over 10,000 Monte Carlo simulations as an error bound on these techniques. In the best overall case, ~20% of the year was required to achieve a <10% 99.9th percentile error, and in the best individual case, ~25% of the year was required. In fact, the average case is much better than those bounds, ~2.5% error, or ¼ of the 99.9th percentile error, in the yearly tap changes in most cases. Moving forward there are other opportunities to use machine learning for distribution system analysis; particular attention should be paid to situations where the amount of data involved is prohibitively large for traditional techniques and machine learning could be used to aid in that type of large-scale analysis.

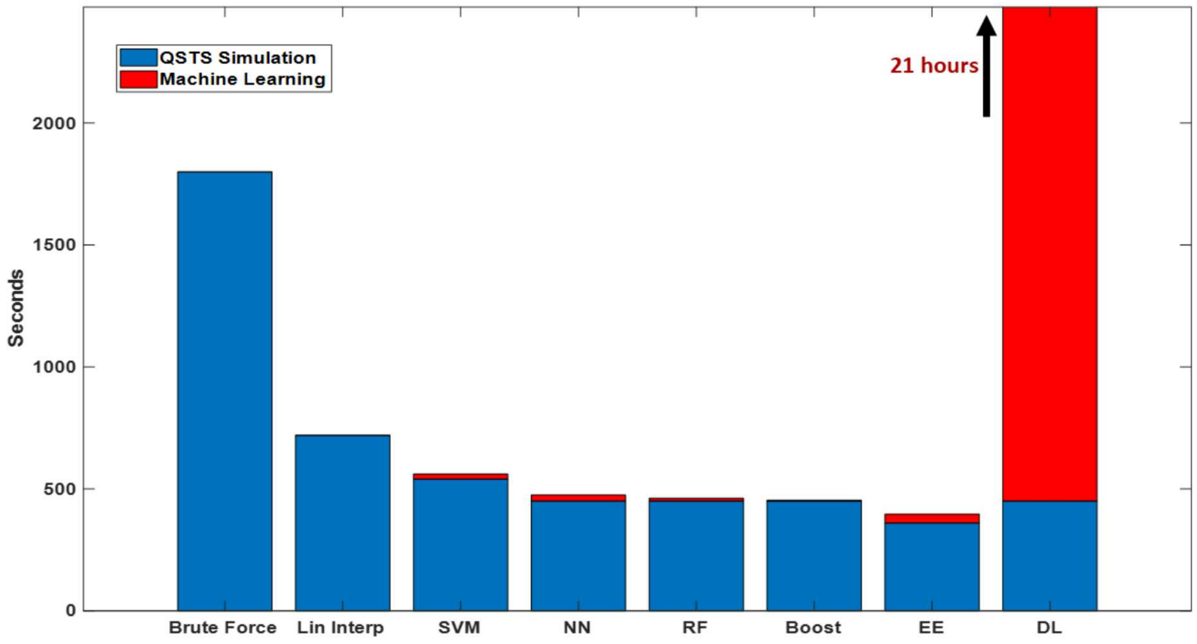


Figure 36 - Final summary of computational times for each algorithm

Table 3 - Algorithm times summary

Algorithm	Percentage of the year simulated with QSTS	ML Time (seconds)	Total Simulation Time	Times Faster
Brute force	100%	-	30 min	-
Non-ML Sampling	40%	-	12 min	2.5
Deep Learning	~25%	75,633 (21 hours)	21 hours	-
Support Vector Machine	30%	21	~8 min	3.3
Neural Network	25%	25	~8 min	4
Random Forest	25%	12	~ 8 min	4
Boosted Ensemble	25%	4	~ 7.5 min	4
Ensemble of Ensembles	20%	37	~7 min	5

10. REFERENCES

[1] M. Reno, J. Deboever, and B. Mather, "Motivation and Requirements for Quasi-Static Time Series (QSTS) for Distribution System Analysis," *IEEE PES Gen. Meet.*, 2017.

[2] R. J. Broderick, J. E. Quiroz, and M. J. Reno, "Time Series Power Flow Analysis for Distribution Connected PV Generation," *SAND2013-0537*, 2013.

[3] M. Lave, J. E. Quiroz, M. J. Reno, and R. J. Broderick, "High Temporal Resolution Load Variability Compared to PV Variability," *IEEE Photovolt. Spec. Conf.*, 2016.

[4] M. J. Reno, M. Lave, J. E. Quiroz, and R. J. Broderick, "PV Ramp Rate Smoothing Using Energy Storage to Mitigate Increased Voltage Regulator Tapping," *IEEE Photovolt. Spec. Conf.*, 2016.

[5] J. Seuss, M. J. Reno, R. J. Broderick, and S. Grijalva, "Analysis of PV Advanced Inverter Functions and Setpoints under Time Series Simulation," *Sandia Natl. Lab. SAND2016-4856*, 2016.

[6] M. J. Reno, J. E. Quiroz, O. Lavrova, and R. H. Byrne, "Evaluation of Communication Requirements for Voltage Regulation Control with Advanced Inverters," *North Am. Power Symp.*, 2016.

[7] J. E. Quiroz, M. J. Reno, O. Lavrova, and R. H. Byrne, "Communication Requirements for Hierarchical Control of Volt-VAr Function for Steady-State Voltage," *IEEE Innov. Smart Grid Technol. ISGT*, 2017.

[8] M. Coddington and et al., "Updating Interconnection Screens for PV System Integration," *Contract*, vol. 303, pp. 275–300, 2012.

[9] B. Palmintier *et al.*, "On the Path to SunShot: Emerging Issues and Challenges in Integrating Solar with the Distribution System," *Natl. Renew. Energy Lab.*, vol. NREL/TP-5D00-65331, 2016.

[10] M. J. Reno and R. J. Broderick, "Predetermined time-step solver for rapid quasi-static time series (QSTS) of distribution systems," in *2017 IEEE Power Energy Society Innovative Smart Grid Technologies Conference (ISGT)*, 2017, pp. 1–5.

[11] X. Wu *et al.*, "Top 10 algorithms in data mining," *Knowl. Inf. Syst.*, vol. 14, no. 1, pp. 1–37, Jan. 2008.

[12] R. Caruana and A. Niculescu-Mizil, "An Empirical Comparison of Supervised Learning Algorithms," *Proc. 23rd Int. Conf. Mach. Learn.*, pp. 161–168, 2006.

[13] L. Breiman, "Random Forest," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[14] G. Biau, "Analysis of a Random Forests Model," *J. Mach. Learn. Res.*, no. 13, pp. 1063–1095, 2012.

[15] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *J. Comput. Syst. Sci.*, vol. 55, no. 1, pp. 119–139, Aug. 1997.

[16] J. H. Friedman, "Greedy Function Approximation: A Gradient Boosting Machine," *Ann. Stat.*, pp. 1189–1232, 2001.

[17] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," *Front. Neurorobotics*, vol. 7, Dec. 2013.

[18] Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learing," *Nature*, vol. 521, no. 436, 2015.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Neural Inf. Process. Syst. NIPS*, 2012.

[20] A. Mohammed, W. Yaqub, and Z. Aung, *Probabilistic Forecasting of Solar Power: An Ensemble Learning Approach*. 2015.
[21] M. Reno, R. Broderick, and L. Blakely, *Machine Learning for Rapid QSTS Simulations using Neural Networks*. 2017.
[22] M. Lave, M. J. Reno, and R. J. Broderick, "Characterizing Local High-Frequency Solar Variability and the Impact to Distribution Studies," *Sol. Energy*, 2015.
[23] J. Deboever, X. Zhang, M. Reno, R. Broderick, S. Grijalva, and F. Therrien, *Challenges in reducing the computational time of QSTS simulations for distribution system analysis*. 2017.
[24] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *Computational Intelligence and Bioinspired Systems, Lecture Notes in Computer Science 3512*, 2005, pp. 758–770.
[25] J. Galtieri and M. J. Reno, "Intelligent Sampling of Periods for Reduced Computational Time of Time Series Analysis of PV Impacts on the Distribution System," *IEEE Photovolt. Spec. Conf.*, 2017.
[26] R. J. Broderick, K. Munoz-Ramos, and M. J. Reno, "Accuracy of Clustering as a Method to Group Distribution Feeders by PV Hosting Capacity," *IEEE PES Transm. Distrib. Conf. Expo.*, 2016.
[27] M. J. Reno, K. Coogan, S. Grijalva, R. J. Broderick, and J. E. Quiroz, "PV Interconnection Risk Analysis through Distribution System Impact Signatures and Feeder Zones," *IEEE PES Gen. Meet.*, 2014.
[28] B. Palmintier, B. Bugbee, and P. Gotseff, "Representative Day Selection Using Statistical Booststrapping for Accelerating Annual Distribution Simulations," *IEEE Innov. Smart Grid Technol. ISGT*, 2017.
[29] M. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.
[30] C. M. Lee and C. N. Ko, "Time Series Prediction Using RBF Neural Networks with a Nonlinear Time-varying Evolution PSO Algorithm," *Neurocomputing*, vol. 73, 2009.
[31] J. L. Carmo and A. J. Rodrigues, "Adaptive Forecasting of Irregular Demand Processes," *Eng. Appl. Artif. Intell.*, vol. 17, 2004.
[32] D. Brezak, T. Bacek, D. Majetic, J. Kasac, and B. Novakovic, "A Comparison of Feed-forward and Recurrent Neural Networks in Time Series Forecasting," *IEEE Conf. Comput. Intell. Financ. Eng. Econ. CIFEr*, 2012.
[33] L. Breiman, *Classification and Regression Trees*. New York: Routledge, 1984.
[34] L. Blakely, M. J. Reno, and R. J. Broderick, "Decision Tree Ensemble Machine Learning for Rapid QSTS Simulations," *IEEE Innov. Smart Grid Technol. ISGT*, 2018.
[35] V. Vapnik, *The Nature of Statistical Learning Theory*. Springer Verlag N.Y., 1995.
[36] S. R. Gunn, "Support Vector Machines for Classification and Regression," *Tech. Rep. Sch. Electron. Comput. Sci. Univ. Southampt.*, 1998.
[37] M. . Hearst, B. Scholkopf, S. Dumais, E. Osuna, and J. Platt, "Trends and Controversies - Support Vector Machines," *IEEE Intelligent Systems*, vol. 4, no. 13, pp. 18–28, 1998.
[38] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *J. Artif. Intell. Res.*, no. 16, pp. 321–357, 2002.
[39] B. Lange *et al.*, "Wind Power Prediction in Germany - Recent Advances and Future Challenges," 2014.
[40] "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges." [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 09-Apr-2018].
[41] "ImageNet." [Online]. Available: http://www.image-net.org/. [Accessed: 09-Apr-2018].

[42] F. Therrien, M. Belletête, J. Lacroix, and M. J. Reno, "Algorithmic Aspects of a Commercial-Grade Distribution System Load Flow Engine," *IEEE Photovolt. Spec. Conf.*, 2017.

[43] Z. K. Pecenak, V. R. Disfani, M. J. Reno, and J. Kleissl, "Multiphase Distribution Feeder Reduction," *IEEE Trans. Power Syst.*, vol. 33, no. 2, pp. 1320–1328, Mar. 2018.

[44] D. Montenegro, R. C. Dugan, and M. J. Reno, "Open Source Tools for High Performance Quasi-Static-Time-Series Simulation Using Parallel Processing," *IEEE Photovolt. Spec. Conf.*, 2017.

[45] D. Montenegro, G. A. Ramos, and S. Bacha, "A-Diakoptics for the Multicore Sequential-Time Simulation of Microgrids Within Large Distribution Systems," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1211–1219, May 2017.

[46] D. Montenegro, G. A. Ramos, and S. Bacha, "An Iterative Method for Detecting and Localizing Islands Within Sparse Matrixes Using DSSim-RT," *IEEE Trans. Ind. Appl.*, vol. 54, 2018.

[47] R. Hunsberger and B. Mather, "Temporal decompostion of a distribution system Quasi-Static time-series simulation," in *2017 IEEE Power Energy Society General Meeting*, 2017, pp. 1–5.

[48] R. Hunsberger and B. Mather, "Monte Carlo Based Method for Parallelizing Quasi-Static-Time-Series Simulations," *IEEE Int Conf Prob Methods Appl Pwr Sys*, 2018.

[49] D. Montenegro, J. Gonzalez, and R. Dugan, "Multi-rate control mode for maintaining fidelity in Quasi-Static-Time-Simulations," in *2017 IEEE Workshop on Power Electronics and Power Quality Applications (PEPQA)*, 2017, pp. 1–5.

[50] B. Mather, "Fast Determination of Distribution-Connected PV Impacts Using a Variable Time-Step Quasi-Static Time-Series Approach," *IEEE Photovolt. Spec. Conf.*, 2017.

[51] M. J. Reno and B. Mather, "Variable Time-Step Implementation for Rapid Quasi-Static Time-Series (QSTS) Simulations of Distributed PV," *IEEE Photovolt. Spec. Conf.*, 2018.

[52] M. U. Qureshi, S. Grijalva, and M. J. Reno, "A Fast Quasi-Static Time Series Simulation Method for PV Smart Inverters with Var Control Using Linear Sensitivity Model," *IEEE Photovolt. Spec. Conf.*, 2018.

[53] X. Zhang, S. Grijalva, M. J. Reno, J. Deboever, and R. J. Broderick, "A Fast Quasi-Static Time Series (QSTS) Simulation Method for PV Impact Studies Using Voltage Sensitivities of Controllable Elements," *IEEE Photovolt. Spec. Conf.*, 2017.

[54] J. Deboever, S. Grijalva, M. J. Reno, and R. J. Broderick, "Fast Quasi-Static Time-Series (QSTS) for Yearlong PV Impact Studies Using Vector Quantization," *Sol. Energy*, 2018.

[55] J. Deboever, S. Grijalva, M. J. Reno, X. Zhang, and R. J. Broderick, "Scalability of the Vector Quantization Approach for Fast QSTS Simulation for PV Impact Studies," *IEEE Photovolt. Spec. Conf.*, 2017.

[56] J. Deboever, S. Grijalva, M. J. Reno, and R. J. Broderick, "Algorithms to Effectively Quantize Scenarios for PV Impact Analysis using QSTS Simulation," *IEEE Photovolt. Spec. Conf.*, 2018.

[57] B. Li, B. Mather, J. Deboever, and M. J. Reno, "Fast QSTS for Disributed PV Impact Studies using Vector Quantization and Variable Time-Steps," *IEEE Innov. Smart Grid Technol. ISGT*, 2018.

[58] T. S. Madhulatha, "An Overview on Clustering Methods," *IOSR J. Eng.*, vol. 2, no. 4, pp. 719–725, 2012.

[59] H. Yu, T. Xie, S. Paszczynski, and B. M. Wilamowski, “Advantages of Radial Basis Function Networks for Dynamic System Design,” *IEEE Trans. Ind. Electron.*, vol. 58, no. 12, pp. 5438–5450, 2011.

[60] P. Baldi, “Autoencoders, Unsupervised Learning, and Deep Architectures,” *JMLR Workshop Unsupervised Transf. Learn.*, 2012.

[61] C. Doersch, “Tutorial on Variational Autoencoders,” *ArXiv160605908 Cs Stat*, Jun. 2016.

[62] I. J. Goodfellow and et al., “Generative Adversarial Networks,” *Adv. Neural Inf. Process. Syst.*, 2014.

[63] M. J. Reno, J. A. Azzolini, and B. Mather, “Variable Time-Step Implementation for Rapid Quasi-Static Time-Series (QSTS) Simulations of Distributed PV,” IEEE Photovoltaic Specialists Conference (PVSC), 2018.

DISTRIBUTION

1	MS1033	Robert J. Broderick	8812
1	MS1033	Abraham Ellis	8812
1	MS1140	Logan Blakely	8813
1	MS1140	Matthew J. Reno	8813
1	MS0899	Technical Library	9536 (electronic copy)

Sandia National Laboratories