Model Stacking Performance Comparisons for Lifetime Estimation of CMOS ICs

by

Koen du Buf
student id: 4589602

University Supervisors:	Neil Yorke-Smith & Sebastijan Dumančić
NXP Supervisors:	René van den Berg & Quino Sandifort
Faculty:	Faculty of Electrical Engineering, Mathematics and Computer Science
Project Duration:	November 2022 - June 2023

Contents

1 Introduction

In the modern digital world integrated circuits (IC) are everywhere, you can find them in coffee machines and cameras, and they control pacemakers and vehicles. It is crucial for these devices to be reliable to prevent disasters from happening like losing control of a car. On top of that, ICs are expected to retain this reliability even after decades of continued use.

Nowadays IC manufacturers go through extensive testing to give estimations on the lifetime of their products. Current lifetime testing is done by measuring many variables on ICs during an extended period of operation in a high temperature environment. Such a test is called a High Temperature Operating Life (HTOL) test, and the duration of this test often spans months. Using such tests during the design of ICs is expensive and slows down the process, additionally even after such a simulation the exact lifetime of a circuit is unknown as it might vary due to intense usage, or usage in unexpected environments.

In this project we will aim to create a machine learning model to predict future HTOL measurements based on earlier measurements. This could not only reduce the HTOL test duration, but also ultimately lead to further development such as embedding a similar model on ICs for self diagnoses. We will also investigate various aspects of model stacking, a technique used to combine the results of various predictive models. Investigating this technique might help to improve the performance of our models, and has scientific value due to the limited amount of prior research done in this area. Additionally, we will look at the performance of a few machine learning models when using a lower bit precision, as this might be important for the implementation of these models on ICs by reducing the size and power needed when implementing these models in hardware.

This thesis was conducted at international semiconductor manufacturer NXP Semiconductors. NXP Semiconductors is a international semiconductor company based in the Netherlands. They specialize in developing high performance, low power, and secure ICs. Devices made at NXP are used in industrial, mobile and automotive markets, among others. With about 31.000 employees, they are present in more than 30 countries world wide.

It is in the interest of NXP to create reliable devices. To this end they are looking for ways to improve the reliability of their products. This thesis aims to predict the future state of ICs, such predictions could be used for multiple purposes. One such purpose is a reduction of the amount of HTOL tests that need to be ran during the solution design process. Another purpose is online predictions of a device's lifetime on board of the device itself. NXP is working with another student to look into the implementation of the models used in this thesis onto hardware, and a third student is working at NXP to analyse embedded sensors for features that could help in the prediction of ageing effects.

The goal of this thesis is two fold, the first objective is to predict future HTOL measurements from early measurements with the best possible performance given the available data. The second goal is to investigate the performance implications of using model stacking approaches in this use case, and in particular when working with such a small data set.

To guide this work we define 3 main research questions.

1. How can we improve our predictive performance for the future HTOL measurements problem using singular machine learning models?
2. What are the performance implications of using model stacking compared to singular machine learning models?
3. How do different variations of model stacking influence our predictive performance on this problem?
4. What is the impact on performance when using a lower bit precision for singular models and for stacked models?

In chapter 2 we will go over related work and background on IC ageing, model stacking and small data problems. Chapter 3 will contain a problem description, and explain methodology used in this project such as the measurements we will target, data pre-processing, the machine learning models we will use for our experiments and the experimental setup we will use. Following this, in chapter 4 we will list our experiments and their outcomes, including general performance, model stacking and few bit machine learning experiments. Finally, chapter 5 will contain the conclusion, discussion and some options for future work.

2 Background & Related work

2.1. Integrated Circuit Ageing

While ICs are essential in modern electronics, they are known to get less reliable over time. ICs are subject to ageing, this is a process that influences devices over time, and changes their behaviour. Ageing occurs due to various physical mechanisms such as thermal, electrical or mechanical stress. Ageing of ICs can lead to degraded performance and reliability, and eventually even to device failure. The speed and effects of ageing depend on factors like the circuit design, fabrication process and operating environment. To create extra reliability, ICs often employ tools like redundancy and error correcting codes.

Ageing under normal circumstances takes years to have a significant effect on ICs, depending on usage. During the design process engineers need to ensure that devices will stay reliable for an extended amount of time, but they cannot just wait years to see the ageing process. To speed up the ageing process, and thus test future reliability of ICs, manufacturers use accelerated lifetime testing. Such tests are done by subjecting devices to harsh environments that speed up the ageing process, like applying a constant higher voltage, or operation in increased humidity or extreme temperatures.

One common test to gain insight into the long term effects of ageing is a High Temperature Operating Life test (HTOL). In this test devices operate in extreme temperatures, greatly increasing ageing speed. HTOL tests used to test devices by NXP are ran at 150 degrees Celsius, and can run for as long as 2000 hours, which is nearly 3 months. With such a test setup devices will experience ageing effects equivalent to 20 years of regular usage.

HTOL data can provide valuable insights into the remaining lifetime and possible problems on ICs. Clearly executing a 1500 hour test is expensive and time-consuming, and thus it would be beneficial to estimate future measurements based on shorter simulations. Research has been done on prediction of failures [9], and the overall remaining lifetime of ICs [11] [23], but not on prediction of future HTOL measurements.

2.2. Lifetime Prediction

With the development of IC devices all over the world, more research is being done into their reliability and lifetime prediction. This includes machine learning research such as on chip monitoring to predict chip failures [9], and prediction of the remaining lifetime of devices both on [11] and off chip [23]. As far as literature study has shown, there have been no previous studies to predict specific future measurements on ICs using machine learning as performed in this study.

Other than machine learning research, exact models for device ageing also exist. When given the design of an IC, a simulation can be done to predict ageing effects over long periods of time. The issue with these simulations is the fact that they are ideal models where each IC is assumed perfect, that is they are assumed to be without impurities. For this reason, these models do not accurately predict the ageing effect on real life devices. As a machine learning model learns from real world data, it can learn to model the real effects of ageing, and thus potentially make more accurate predictions. This might come at the cost of consistency, as the performance of a machine learning model might deteriorate when presented with data outside its training distribution.

2.3. Small Data Problems

Machine learning is a way to learn patterns from data, and use these patterns to predict targets on unseen data. When a small data set is used for machine learning, the available data might not represent the real distribution of the problem domain, and thus finding the real patterns in such a case will be more challenging. Additionally, when only a small amount of data is available, the noise present in this data is more pronounced, thus making it even harder to find the correct patterns in the data, causing overfitting. Finally, outliers in the data are more significant, resulting in possible patterns based on these outliers and causing further overfitting. These factors affect the generalization performance of machine learning models, and make small data sets a particularly hard problem in machine learning.

To reduce the overfitting issues described above, we can reduce the degrees of freedom (DoF) available to machine learning models [22]. The DoF is a measure of complexity available to machine learning models to learn the patterns in the data, often similar to the amount of parameters available for a model to vary. The DoF can be controlled by using regularization methods such as the common L2 (Ridge) regularization. Regularization methods often include adding a penalty for using specific parameters or constraining parameters in the model. This way regularization effectively reduces the available parameter options that can be used, thus reducing the DoF. This reduction in the amount of parameter options available in the model reduces its ability to learn complex patterns. Due to this restriction, a model is forced to learn simpler patterns, resulting in reduced overfitting, as the model will be less likely to learn patterns based on outliers or noise in the data. When the DoF is reduced too far, a model will start underfitting as it is unable to learn the real patterns available in the data. For this reason using an appropriate DoF value is crucial for the performance of models on small data problems.

2.4. Ensamble Learning

Ensamble Learning refers to the idea of combining the predictions of multiple machine learning models to create a single prediction for a given problem. While many machine learning systems use a single model to create predictions, it has been shown that ensamble learning can improve performance in many cases[13], including the case of small data sets [14]. Ensamble systems have the best performance when the predictions of component models are dissimilar [3] [14]. Three of the most common ensamble learning techniques are bagging, boosting and stacking.

Bagging is the simplest of these methods. When using this method a set of homogeneous models, most commonly decision trees, is trained independently on different subsets of the training data. The final prediction is created by taking the average of the predictions of each of these models, or in case of classification the most common class is chosen. The most commonly known example of bagging is the random forest model. It has been shown that bagging is almost always more accurate than a single model [12].

Boosting is a more advanced variation of ensamble. Boosting, like bagging, also uses a set of homogeneous models, again most commonly decision trees. On the other hand, models used in boosting are trained sequentially. Each newly trained model is trained on a subset of the samples, prioritizing samples where previous models had the largest error. This way a new model will prioritize correctness of these samples. With each model the total training error will become increasingly lower. The final prediction is made by a weighted average of the predictions of each model, where the weight associated with each model is determined by their error. While boosting can give significantly better results than bagging, it is also more susceptible to overfitting [12].

Stacking differs from the aforementioned techniques by using a meta learner instead of using average techniques to combine predictions. This technique is the main focus of our ensamble research in this project and it is further described below.

2.5. Stacked Generalization

Stacked generalization [19], also called model stacking, uses a set of heterogeneous models. The most common use of stacking consists of a set of models, called level 1 models, followed by a single level 2 model, also called the meta learner. The predictions of level 1 models are used as features for the level 2 model, which then creates the final prediction. A diagram of this model stacking architecture can be seen in Figure 2.1. Training of the level 2 model is usually done by using out-of-fold predictions of level 1 models. This gives the meta learner predictions on data previously unseen by level 1 models and thus gives a better indication of their predictions on new instances. Stacking often increases the predictive performance of a model relative to its best component model, but it is not always the best ensamble [16].

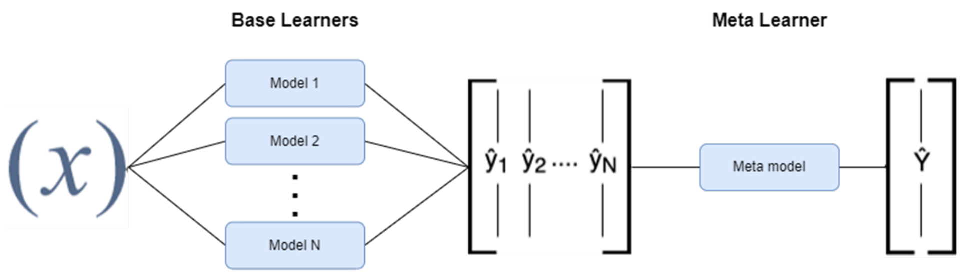

Figure 2.1: A visual overview of the general model stacking approach

Training a stacking ensamble generally involves fitting a lot of models, as it employs K-fold cross validation to generate training data from each of the base learners. This means for example that a stacking ensamble with M base learners and a meta learner requires first fitting the M base learners K times to generate training data for the meta learner, and then fitting all base learners and the meta learner on the full training set. This means in total we need to fit $(K+1)*M+1$ models.

Stacking can be used in many ways. We will list variations and important considerations below. These variations include changes in models, inputs, structure and usage of the general stacking approach.

2.5.1. Base Learners

When using model stacking the base learners are the first part of any stack. The base learners play a crucial part in the ensamble, as the performance of the base learners directly affects the performance of the ensamble. A general equation for the performance of the stack is as follows [8]:

$$E_{ensamble} = \langle E_{individuals} \rangle - \langle A \rangle \tag{2.1}$$

In this equation we use $\langle ... \rangle$ to denote the average of a vector, E stands for the prediction error and A is the ambiguity, or dissimilarity between individual models. From this formula we see that a trade off must be made between the performance of the individual base learners and the dissimilarity of said learners. In fact this means that it is not always better to use the best performing models as base learners [5, 3], as these models will likely have very similar results, and thus will not be of too much benefit to the ensamble.

Dissimilarity measures can be used to assess the choice of base learners, and can even be used to create a ensamble of a subset of base learners with similar performance to the original set [5]. There are various dissimilarity measures for regression models. We chose to use the *chi-square* measure as models selected based on this measure gave the best performance out of all tested dissimilarity measures in [5]. We will use this measure to steer the choice of our base learners.

2.5.2. Meta Learner

Another important part of a stacked ensamble is the meta learner. The meta learner is responsible for combining the outputs of base learners and creating the final prediction. This model can be selected just based on the performance of the overall ensamble. Due to our small data set the meta learner should pay extra attention to overfitting.

In classification problems a common meta learner is logistic regression, but the use of meta decision trees [21] has shown great results, and the use of weighted combination based meta learners was tested as the best performer in unbalanced classification [24]. As stacking is more commonly used for classification problems there are not yet any tried and true meta learners for regression problems.

2.5.3. Multiple Levels

When using model stacking, we generally use two levels: level 1 consisting of the base learners and level 2 which only has the meta learner. Stacking is however not limited to just two levels. We can introduce a third, fourth or any amount of levels, where each level takes input from the previous level and supplies the next level with its predictions. The amount of training time we need when adding another level scales only with the amount of learners in that level. Thus the overall training time of the ensamble scales linearly with the total amount of separate learners used.

Multi level stacking has previously been used with good results [15][17]. Multi level stacking can also be combined with multi view stacking to create a funnel looking structure of models that could theoretically use an arbitrary amount of input features without suffering from any of the downsides of highly dimensional inputs. This structure will be discussed later in section 2.5.6.

2.5.4. Blending

Blending is a technique that is very similar to stacking, but is more focused on training speeds. Blending was introduced by the winners of the Netflix 1 million dollar challenge [7] and has since been used frequently by the competitive machine learning communities like Kaggle competitions. As described in section 2.5, $(K + 1) * M + 1$ models need to be trained in order to train the whole regular stacking ensamble. Here K is the amount of folds use to generate training data for the meta learner, and M is the amount of base learners in the ensamble. Blending reduces this amount to $M + 1$ by getting rid of the cross validation part in the training.

Instead blending divides the original training data into a set of training data for the base learners and a set of remaining data. The remaining data is passed through the trained base learners to create predictions, and these predictions are then used to train the meta learner.

While blending greatly reduces the amount of training time, it does have its downsides. Blending has been shown to have overall worse performance than stacking when supplied with the same data [4]. On the other hand, blending is able to outperform stacking models, while using significantly less training time when supplied with more data [20].

2.5.5. Passthrough

Passthrough is another slight modification to the general stacking architecture. When using passthrough the original features that are given to the base learners are also given to the meta learner. This gives the meta learner more information, and can thus lead to better final predictions, but could also increase overfitting and increases the input dimensionality for the meta learner.

2.5.6. View Splitting

View splitting, also called 'Multi-view stacking' increases the capability for a model stacking ensamble to deal with more features. This is done by splitting the features into multiple sets, or 'views'. Each of these views is then used as the input features for one or a few of the base learners. This technique allows the ensamble to work with a larger amount of features, thus potentially extracting more information, while each base learner only uses a small set and thus does not suffer from large input dimensions. Additionally the base learners, now using different feature sets, will likely have more diverse predictions. This diversity allows the meta learner to better learn from the base learner predictions and thus achieve even better performance. When using multi-view stacking, a non-negative linear regression with lasso regularization is known to give promising results [10].

View splitting can be combined with multi-level model stacking to create what we will call "Multi level view splitting". This structure combines the division of features of view splitting with multiple levels by giving each level only a subset of features created by the previous level. This creates a funnel structure. For example in this structure the first level can have 30 learners, followed by 6 learners which each only use 5 predictions from the 30 generated by the first level, and finally a meta learner. Because of this, we can extend this structure to arbitrary length and use any amount of input features without creating dimensionality issues, as each separate learner will only use a small subset of the features. Theoretically this could lead to arbitrary performance.

2.6. Few Bit Machine Learning

While machine learning commonly is done on modern devices with 64 or 32 bit precision, it might be beneficial to use fewer bits for these algorithms. Research into using fewer bits is more common related to deep learning due to the large model size and huge amount of computations used by deep learning models. In that scenario, using fewer bits would reduce the storage size of the model and could reduce the otherwise large energy consumption of deep learning model computations [6]. Deep learning research has been moving towards using 16 bit value representations and 32 bit computations, and more recently there have been steps towards 8 bit representations and the usage of 16 bit computations [1][18].

In this project we explore the performance impact of using fewer bits in various different machine learning models for a different reason. Using fewer bits would be beneficial for the purpose of implementation of these models in hardware. Using fewer bits in a hardware implementation reduces IC size, power usage and cost. As the models studied in this work do not contain a huge amount of parameters, there is little benefit to reducing the bit precision of these models, and thus little previous research has been done on this topic.

When using a lower bit precision the amount of bits required for computations is generally twice the amount used by the data representation. As it is harder to simulate computations using fewer bits on modern machines, our experiments will be limited to investigating the performance of models when using few bit representations for all values used by a model. This includes the input data, model predictions and any values used internally by the model e.g. the coefficients and intercept in the case of linear regression.

3 Methodology

To estimate the lifetime of ICs about 90 devices are monitored during a HTOL test. This test simulates extended use of an IC by using it at a high temperature. The data we will consider was gathered during a HTOL tests of 1500 or 1740 hours at 150 degrees Celsius. In such a test the device shows signs of ageing in 100 hours equivalent to roughly 1 year of regular device usage. During this test, each device was tested on about 6000 different values at 4 to 6 different time points. The setup of such a test is complicated, and it is difficult to get a lot of samples, thus one of the main challenges when creating this model is the small amount of available data. For this reason we will need to be careful when selecting or extracting features and pay extra attention to prevent overfitting. The gathered data includes but is not limited to pin voltages, leakage currents and hardware speeds. A device is considered unreliable when a parameter that is critical for the design, has a measurement value outside the range allowed by its specification.

In addition to the problem of creating the best predictions for future HTOL measurements, we will also investigate the performance of model stacking (see chapter 2.4) and its variations in this context. Finally, we will also look at the performance of machine learning models when using a low bit precision. Our hypotheses are as follows:

1. Model stacking will outperform singular models on the future HTOL problem
2. Model stacking variations, such as view splitting, will increase performance relative to basic model stacking
3. Using a lower bit precision will reduce the overall performance of our models

3.1. Product Data Sets

In this project, we will be working with 4 HTOL data sets. These data sets consist of HTOL measurement data from 4 past products created by NXP. For each of these data sets, different measurements were done, creating different features and targets. The products related to each of these data sets are made using similar technologies, and thus the measurements done for all data sets should be comparable in terms of variable relations. For our experiments we create models for each data set separately, but only display the result for one of the data sets. Results on all of the other data sets are similar unless specified otherwise.

Various attempts were made to combine the 4 product data sets into a single larger data set by matching the measurements between the data sets. The most promising of these attempts was matching on a combination of the Levenshtein edit distance and matching of similar distributions. Some matching names were found using the Levenshtein edit distance, and these measurements should have been very similar according to engineers. However, the distributions of these features were not similar, possibly due to different measurements procedures or due to still not matching the right features between data sets. In the end we did not succeed in creating a single larger data set.

3.2. Target Measurements

The targets we will predict are a subset of HTOL measurements, measured around 1500 hours into the test. We will not predict all measurements values for two reasons. Firstly creating separate models for thousands of targets is infeasible, and secondly it is not necessary to predict every measurement as there are many measurements that are either not important or are known to not move much over time. The target measurements that are chosen to be predicted are thus determined by these two factors. To find the specific targets that we will predict we use two metrics that quantify these factors.

The first metric we use is a Delta Sigma Analysis, defined here as:

$$DSA(X_{1500H}, X_{48H}) = \frac{median(X_{1500H}) - median(X_{48H})}{(Q_{95}(X_{1500H}) - Q_5(X_{1500H}))/3.29} \tag{3.1}$$

This is not a standard definition, but this is the definition as used commonly at NXP. Here Q_{95} and Q_5 represent the 95th and 5th percentile respectively. Intuitively this metric represents the movement of a measurement over time as a fraction of the variance of this measurement. A measurement is considered as a target only when the absolute value of this metric is below a predetermined value, in our case set to 1.5.

The second metric is a CPKn calculation. The formula for this metric is defined as:

$$\begin{aligned} Cpu(X_{1500H}) &= Spec_{upper} - median(X_{1500H}) \\ Cpl(X_{1500H}) &= median(X_{1500H}) - Spec_{lower} \\ CPKn(X_{1500H}) &= \frac{min(Cpl(X_{1500H}), Cpu(X_{1500H}))}{3 * (Q_{75}(X_{1500H}) - Q_{25}(X_{1500H}))/1.35} \end{aligned} \tag{3.2}$$

In this formula we again use Q as defined above, additionally $Spec_{upper}$ and $Spec_{lower}$ are the upper and lower specification limits for the given measurement. This metric considers the distance from the median measurement to the nearest specification limit, this gives a measure of how important a target is to keep the device operating within the specification limit. For our targets we only consider any measurements where the $CPKn$ value is below 4.

With these two criteria, we are left with 75 measurements out of the original 6000 in the case of our main project data set. These will be the targets that we will be predicting future values of. A visualization of this analysis is shown in Figure 3.1.

3.3. Feature Selection

Before starting on our experiments we will first need to do data pre-processing. Since the available data has about 6000 features, but only 90 samples, this means that there is a very high risk of overfitting. One of the ways we will use to try to mitigate this issue is changing the input features using feature selection, extraction and possible data augmentation. Of course any pre-processing that uses information from the data will only use information from the training set to prevent data leakage.

One relatively simple way to reduce the chance of overfitting is to only consider a small subset of the features. A common and powerful way to perform feature selection is the use of sequential feature selection (SFS). A downside to this technique is the run time of the selection, as SFS using 5-fold cross validation of 6000 features needs to fit about 30000 models to

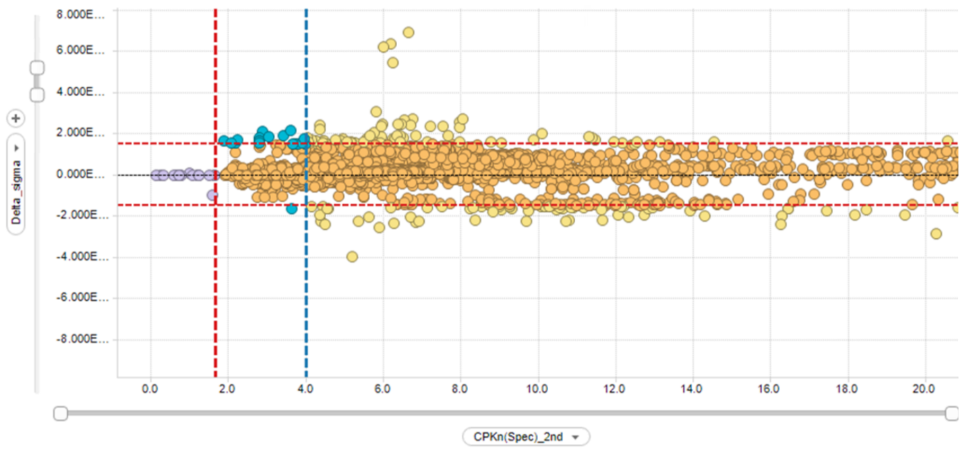

Figure 3.1: A visualization of the delta sigma and CPKn analysis for our target measurements. Each point represents a HTOL parameter with their Delta Sigma value and CPKn value. The horizontal and vertical lines are limits for the delta sigma and CPKn values. Blue points are outside these limits, and are the targets we will focus on predicting. All other points are measurements that do not fit our criteria.

select each feature. While this selection method might be feasible for infrequent use, it greatly increases the run time of each experiment.

To speed up the feature selection process we will use a three step approach. Firstly, we will hand select a single feature. Specifically, we select the feature corresponding to the earlier value of the same measurement that we are trying to predict, as we found that this feature will generally be useful for the model to consider. After this step we will select another M features as follows. We calculate the correlation of all features to the target and to each other. We then select $100 * M$ features that have a high correlation to the target but a correlation to each other lower than some threshold, currently set to 0.7. Finally, we run SFS on these $100 * M$ features to select the final M features. This approach greatly reduces the runtime of the feature selection, and it was found to give results as good as using SFS on all features.

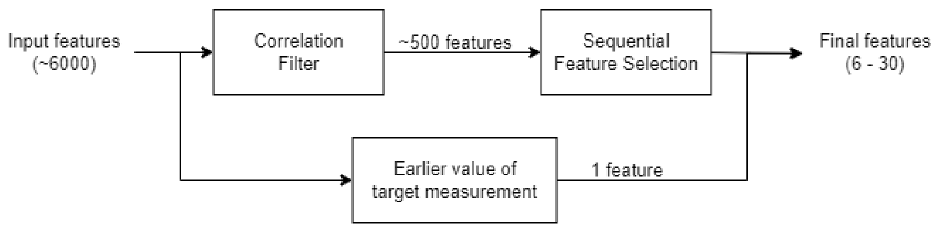

Figure 3.2: A high level overview of the basic feature selection process

Selecting features using the above mentioned method will only find any features that are by themselves responsible for creating reasonable predictions. We thus did not cover a case where a set of features only give useful information when combined. For this reason we explored the idea of creating nonlinear combinations of features by multiplying features together. It is not feasible to try this for all 6000 features, so experiments were done by randomly sampling 100 features at a time and considering all second degree polynomial combinations of all these features. After repeatedly running such tests we did not see any improvement in predictive performance of any model. We thus conclude that creating these combinations is not beneficial to performance in this case.

3.4. Data Augmentation and Oversampling

Other techniques that can help with the overfitting problem are data augmentation and oversampling. Data augmentation creates more training samples by creating synthetic data based on real samples. Common approaches for this are k-nearest-neighbor techniques or creating samples from real data with some added noise. Oversampling creates a more evenly distributed data set in case of unbalanced data by duplicating samples in underrepresented regions. While oversampling is very frequently used in unbalanced classification, it can also be used in regression problems using algorithms like SMOGN [2]. After running experiments with both of these techniques, we did not see any noteworthy improvements in performance, and thus these neither of these methods were used in the final experiments.

3.5. Additional Modifications

To further increase the performance of our models, we applied some additional modifications to the training data. The first modification is the selection of a few features by hand. The selection of these features was based on their frequent selection by the sequential feature selection procedure, and additional domain knowledge from engineers.

Another modification was the usage of multiple target times. The usual data set consists of the normal set of early measurements as input features and the 1500 hour measurements as targets. With this modification, we make more use of the data in between these early and late measurements. We do this by duplicating the input features 3 times, and setting the targets to be the measurement values at all measurement times, specifically these are 602, 1000 and 1500 hours. We also add a new input feature which is equal to the amount of hours in the time we are targeting in that sample. This approach has the benefit of creating more training samples to work with. The assumption is that targets at the other measurement times have a similar relation to the target at the test time of 1500 hours. Using this approach, the model could learn this relation from these extra samples and even use them to lessen the effects of measurement errors at 1500 hours. To keep scoring consistent, the test set when using this modification is the same as the test without it, consisting of only samples measured at 1500 hours.

3.6. Base Models

Now that the data is ready to use, we move on to the selection of our machine learning methods. These algorithms will be selected for two purposes, they are used to model the problem by themselves, and they are also used to build our stacking ensambles later on. These models are chosen based on multiple criteria. Firstly the selected models should be usable with a very limited degree of freedom. A small DoF is important given the small data set problem we are looking at to prevent overfitting [22]. Secondly, a model should not be too time consuming to train. While training time is not a problem when using the model on the problem by itself, it becomes a much larger issue when used in a stacking ensamble due to having to train the model many times as explained in section 2.5. Additionally, as we are creating hundreds of ensamble models to predict distinct measurements and using 5-fold cross validation for our tests, this would further amplify the training times. Because of these factors, any model for a single experiment will be trained $6 * 5 * 100 = 3000$ times. For perspective, this means a training time of 1 minute for a single base learner would lead to 50 hours of training time for just that base learner. We will go over the models that we chose to use, the reasoning behind this and what we use these models for.

Linear regression based models are generally very fast to train and also seem to be a intuitively good fit as the HTOL prediction problem could very well have a linear correlation. Combining this with regularisation this group of models ticks all the boxes to be considered for our selection. The models of this kind that we use in our ensambles are linear regression with L1 and L2 regularisation (Ridge & Lasso), Support Vector Regression, Kernel Ridge Regression and Polynomial Ridge Regression. The last two of these are not strictly linear as they introduce a non-linear space before using linear (Ridge) regression. This gives us some more freedom by not only considering linear functions, while keeping the DoF low and training times fast.

Decision trees based ensamble models are also selected as a good fit because they are known to perform well on small data set problems [12] and they can deal with a large amount of features in our problem. The training times of these models is not as fast as linear regression based models, but they are still quick to train. From this group the random forest model and xg boost model were considered. It is known that xg boost generally overfits faster than random forest [12], and experimentally we observed this as well. Because of this we will only use random forest in our problem and stacking experiments.

Other models we will be using for our problem and for our ensambles are as follows. *K Nearest Neighbors* is used, this model is not expected to have the best performance for this problem due to the small amount of available samples, but it has a fast training time and will bring added diversity to our ensamble models due to its workings differing from any other models used. Another model we will use is a *Gaussian Process*, because this type of model has fast training times and can give good performance even on small data sets. The last model we will use at least partially is Symbolic Regression. We will use Symbolic Regression because it has been shown to have good performance on small data set problems, but due to its long training times we will only be using this type of model as a singular model for the prediction problem, outside of the stacking ensamble. A model that was considered but ultimately not used in this project is the Multi Layer Perceptron. Multi Layer Perceptrons were not used because they generally need a large data set to perform well, and additionally the training times of these models are too long to use in our ensambles.

3.7. Experiment Setup

The goal of our experiments is to evaluate the performance of our models on the future HTOL problem. We do this by comparing various singular models to stacked models and various stacking variations, this will also give us insight into the performance of these various model stacking variations and thus answering our research questions. The questions we will be answering with these experiments are:

1. What predictive performance can be achieved on the future HTOL problem?
2. What performance can be gained from using various model stacking variations?
3. How is the performance impacted when using a lower bit precision for our models?

Because we have a limited amount of samples, the chance of overly optimistic or pessimistic test results is large when randomly choosing a test set. To mitigate this issue we use 5-fold cross validation to obtain our results. This is unrelated to the cross validation we use for hyper parameter tuning or the out-of-fold predictions of base learners. In addition to this, all experiments are repeated on 4 different data sets, corresponding to the HTOL tests of 4 different devices. In our experiments the results were found to be consistent across these data sets unless otherwise specified.

As our predictions could be used to evaluate the behaviour of measurements on ICs, outliers would be a bigger issue than a slightly higher but more consistent error. For this reason we will evaluate our models primarily on the consistency of their predictive performance, rather than their best achieved error.

3.8. Visualising Performance

The metric we will be using to indicate the performance of our models is the mean absolute error of our predictions compared to the true values, divided by the standard deviation of our target variable. We will multiply this value by 100 as this gives an easy intuitive understanding as a percentage error. We will call this metric the Percentage Standard Deviation Error (PSDE).

$$PSDE = abs(Y_{true} - Y_{pred})/std(Y_{true}) * 100 \tag{3.3}$$

This metric was chosen because it indicates performance of a model well, and unlike common metric like mean squared error, this metric can be compared between targets even when these targets are of a different scale. A similar more common metric called R^2 was also considered, but PSDE was chosen because it is more understandable than R^2 values. Unlike R^2, a smaller PSDE is better.

To interpret our experiment results we need a clear way to express the performance of our models. In the case of this project, we need to interpret the results of models on dozens of different targets, and thus a simple numeric comparison is not always representative of the real results. To visualize our results and make them easier to interpret we use empirical cumulative distributions. Intuitively at each position this curve shows the proportion of predictions on the Y axis with a smaller value, or in our case a smaller PSDE than shown on the X axis. For example if X = 40 at Y = 0.8, then 80% of targets have a PSDE smaller than 40%.

3.9. Stacking Defaults

As we will be comparing many model stacking variations, we need to set some default values for our stacking configurations. The two main things we need to keep consistent in our configuration are the base learners and the meta learner.

We will be using 6 base learners for each of our stacking experiments, this amount was chosen because it is enough to evaluate model stacking, and more models do not improve performance, but would increase training times. The specific base learners we will be using in our stacking experiments are random forest, ridge regression, kernel ridge regression, k nearest neighbors, support vector regression and polynomial regression. These base learners were chosen because they are a diverse set of models all with unique properties and reasonable solo performance. For the meta learner in our stacking experiments we will be using the random forest model, as this model came out as the best tested model in the meta learner experiments.

4 Results

In this chapter we will go over our experiments and compare the results. We will also talk about theories behind the results and possible further ways to improve them. In the final section we will go over the overall best performing models and summarize the results.

4.1. Feature Engineering

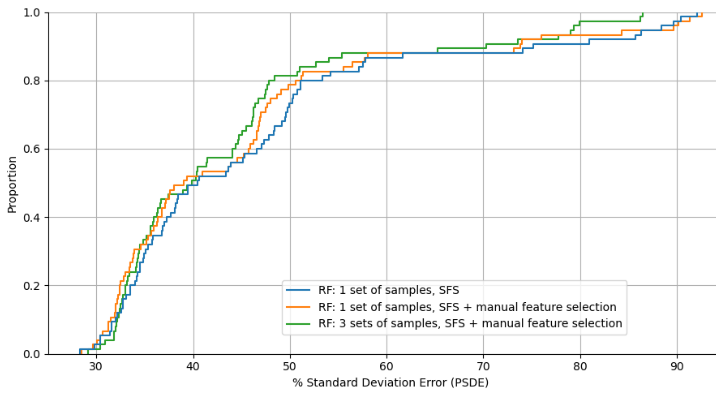

Figure 4.1: A comparison of scores using the random forest model using only samples from 1 input time, with and without manual feature selection and compared to using samples from multiple target times

We start our first experiment by investigating the performance implications of using different feature selection methods, as this can be a crucial first step to get the best overall performance. We compare instances of a random forest model where each instance is given different input features. The performance when using only a single set of target samples, and only features selected using sequential feature selection, is compared to models with manual feature selection and additional target samples. From Figure 4.1 we can see that these modifications improve the overall performance of our predictions, but only slightly. Because of the slight improvement we will use these modifications for our other experiments and for the final results.

4.2. Singular Models

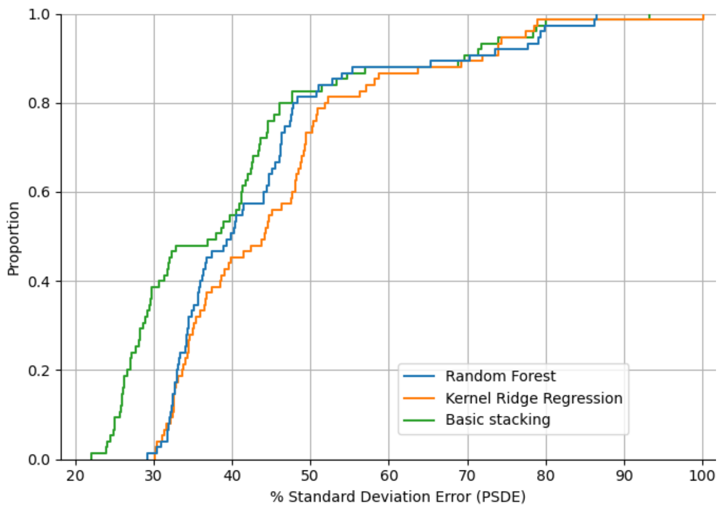

Figure 4.2: A comparison of scores over all targets between singular random forest and kernel ridge regression models, and a basic stacking ensamble

To get a baseline for the performance of our methods, we tested various singular machine learning models for this problem. From the tested models, the best results were obtained using random forest and kernel ridge regression as can be seen in Figure 4.2. When comparing this performance to a basic stacked ensamble we see that the singular models are clearly outperformed. Both of these singular models are themselves also present in the stack.

4.3. Base Learners

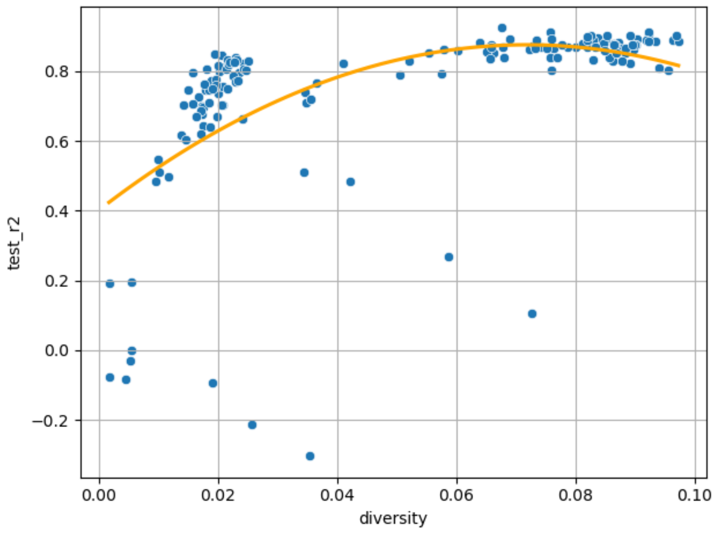

Figure 4.3: A measure of diversity of the base learners compared to the performance, and the general trend of this relation

To get the best performance from our stacked ensamble a good choice of base learners is essential, thus we experimented with various combinations of base learners. For the best results, the predictions created by the base learners should not be similar, as many similar predictions will not give useful information to the meta learner. From our experiments, we can see in Figure 4.3 that a higher diversity in the base learners generally leads to better predictive performance.

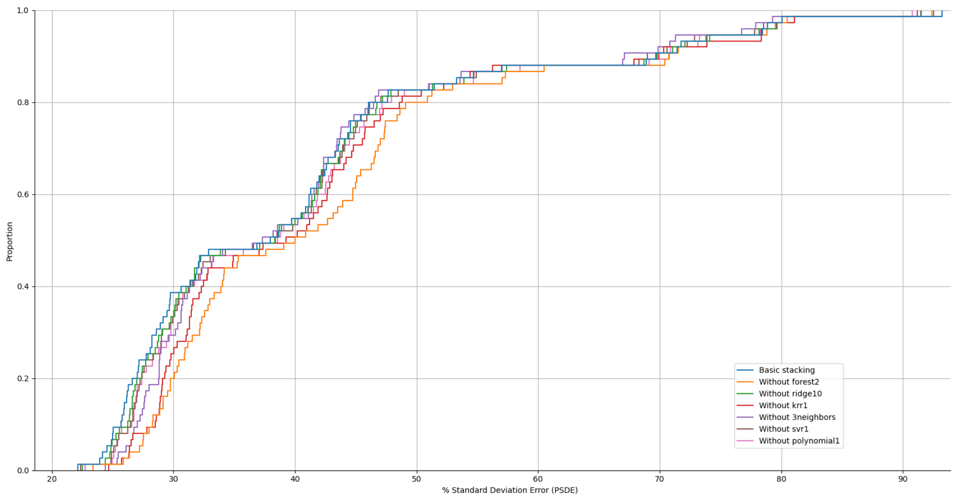

Figure 4.4: A comparison of scores of our stacked ensamble after removal of any of the base learners

After we chose an appropriate set of base learners, we when compared the performance of our ensamble model after removing any single base learner. From this we see that each of the base learners contributes to the performance of the stack as can be seen in Figure 4.4. The random forest and kernel ridge regression base learners are the most important of the six. The other four base learners do slightly improve the overall performance, but are not very important.

4.4. Meta Learner

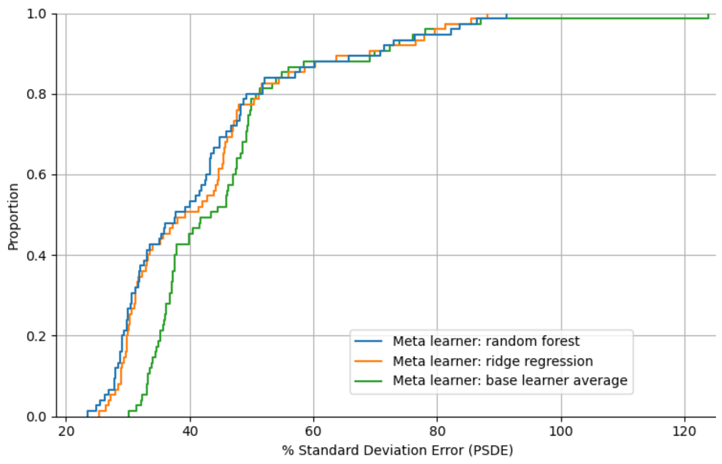

Figure 4.5: A comparison of scores when using model stacking with different meta learners

The next building block of the stack is the meta learner. When comparing various meta learners, the best performers were a random forest model and ridge regression. A comparison of these meta learners and a simplistic meta learner that takes the average of all base learner predictions can be seen in Figure 4.5. From this we can see that the performance is significantly better when using either of these two meta learners compared to the average of the base learners. Of the two meta learners the random forest model performs slightly better.

4.5. Blending

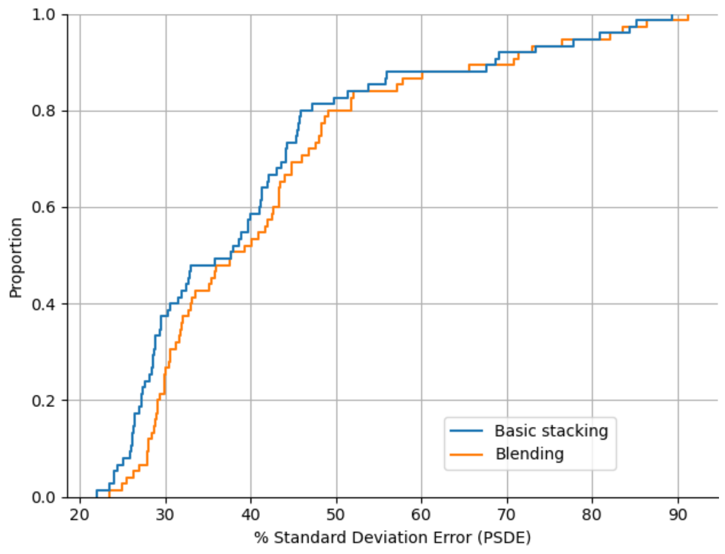

Figure 4.6: A comparison of stacked ensamble scores with and without blending

A stacked ensamble takes a long time to train, as discussed in chapter 2.4. Blending is investigated due to the significant reduction in training time it offers, at the trade off of a reduced performance. When blending is used, we can see a overall slightly worse performance as can be seen in Figure 4.6. As the performance of blending is not much worse, it is still a viable option for testing and prototyping, as the training time is significantly reduced. Using our experiment setup, the training times with and without blending for all 75 targets are roughly 1 and 6 hours respectively.

4.6. Passthrough

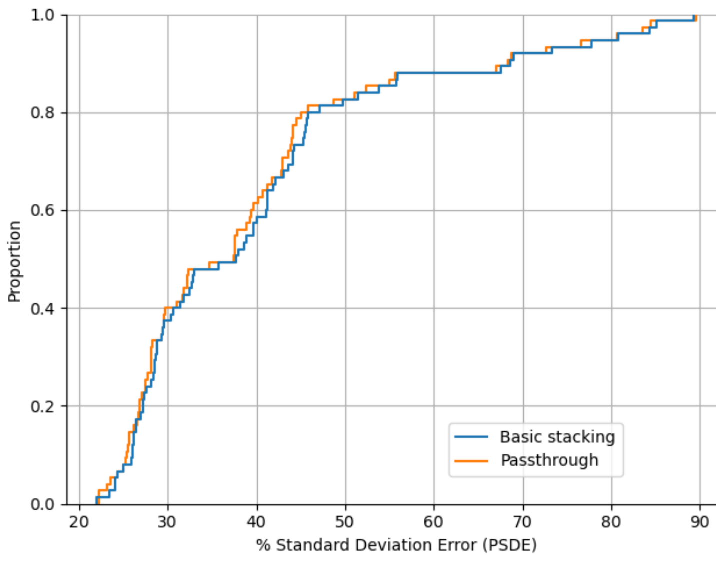

Figure 4.7: A comparison of stacked ensamble scores with and without passthrough

Passthrough is a technique that is not frequently used, but might still be worth exploring. The use of passthrough did occasionally have a slightly better performance, as was the case for multiple tested meta learners. Overall the use of passthrough in our stacking ensambles did not have a notable influence of the predictive performance. A comparison of the stacking ensamble with and without passthrough can be seen in Figure 4.7.

4.7. Multi Level Stacking

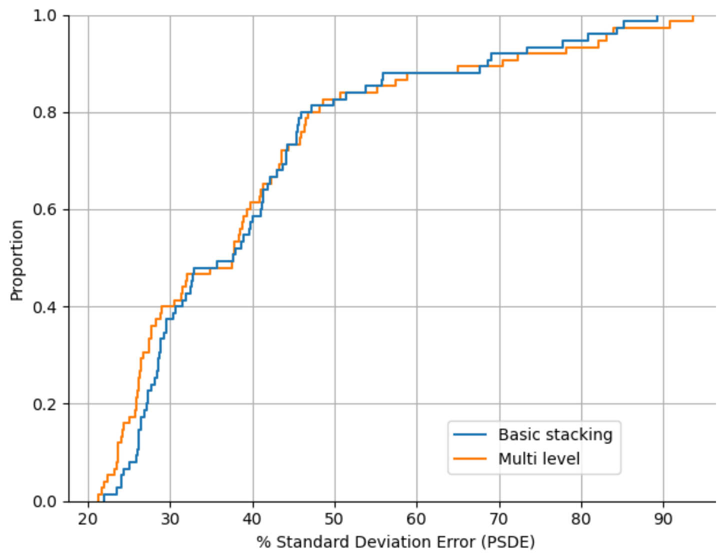

Figure 4.8: A comparison of scores with a default 2 level stack and a 3 level stack

Multi level stacking has not been well researched, but might be worth exploring for its performance and for a promising combination with view splitting as discussed later. In our experiments increasing the amount of levels in a stack did not significantly improve performance from stacking with just 2 levels. The results of this experiment can be seen in Figure 4.8 with the comparison between a 2 level and a 3 level stack. An even larger amount of levels in the stack did not improve performance. While multi-level stacking on its own did not improve the results, usage of this technique in combination with techniques like view splitting creates a significantly better model with theoretically arbitrary predictive performance.

4.8. View Splitting

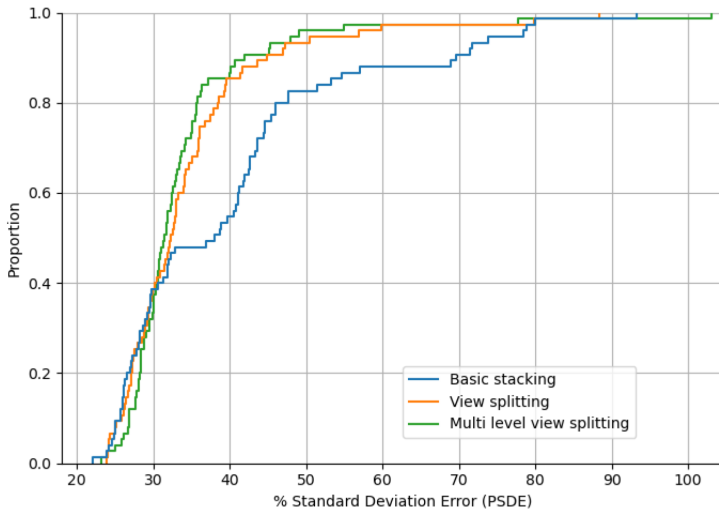

Figure 4.9: A comparison of scores of a normal stacked model and a stacked model using view splitting

View splitting is the most promising technique out of all tested model stacking variations. We particularly focused on this technique because it promising in its ability to use many features, and to increase diversity between base learners. The performance of a stack using view splitting is significantly better than without it, as can be seen in Figure 4.9. This is likely due to the fact that the view splitting model can extract information from more features without negative high dimensionality effects. While other models like random forest are also adapt at dealing with a large number of input features, the view splitting model is still significantly better when given the same inputs.

4.9. Multi Level View Splitting

The combination of multi-level model stacking and view splitting creates a model architecture that can theoretically be extended to achieve arbitrary precision. To test this we created a model with 18 base learners, 6 learners in the second level and a third level meta learner. The performance of a smaller configuration of this model was better than all other tested models, as can be seen in Figure 4.9.

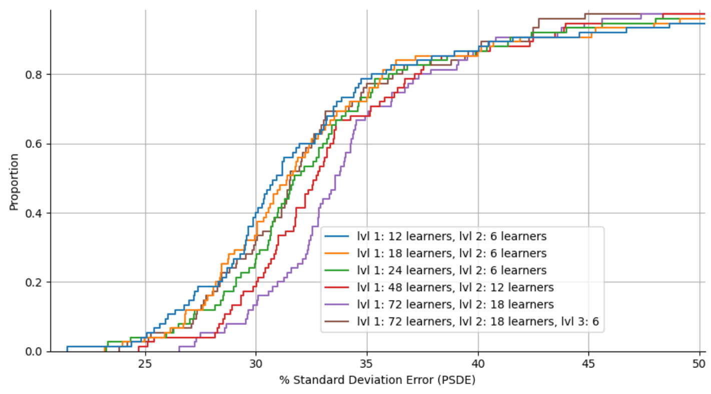

Figure 4.10: The performance of models using multi-level view splitting with a varied amount of learners

Although this architecture should theoretically be able to gain arbitrary performance by adding more learners, from our experiments we see that adding additional learners does not improve the performance over that of a smaller multi-level view splitting stack. One theory to explain this observation would be related to the small amount of diversity present in the predictions of models of higher levels in the stack. This lack of diversity could mean additional predictions might not add any information for the next layer models, and only create a higher feature dimensionality, thus slightly worsening the performance.

4.10. Final Performance

To give a quick overview of the performance of the tested models and model stacking variations we will summarize them in a table below. In this table we will compare the performance of each of the models over all targets using the PSDE obtained on at least 80% of all targets.

Model	80th percentile PSDE
Kernel Ridge Regression	52.0%
Random Forest	48.0%
Blended Stacked Ensamble	48.5%
Basic Stacked Ensamble	46.4%
Stacking + Passthrough	45.5%
Stacking + Multi level	47.4%
Stacking + View splitting	38.5%
Stacking + Multi level view splitting	35.7%

Finally, we visualize the performance of the best singular model compared to basic stacking and the best stacked ensamble variation in Figure 4.11. From this we see the performance benefit that can be realized when using these variations of model stacking.

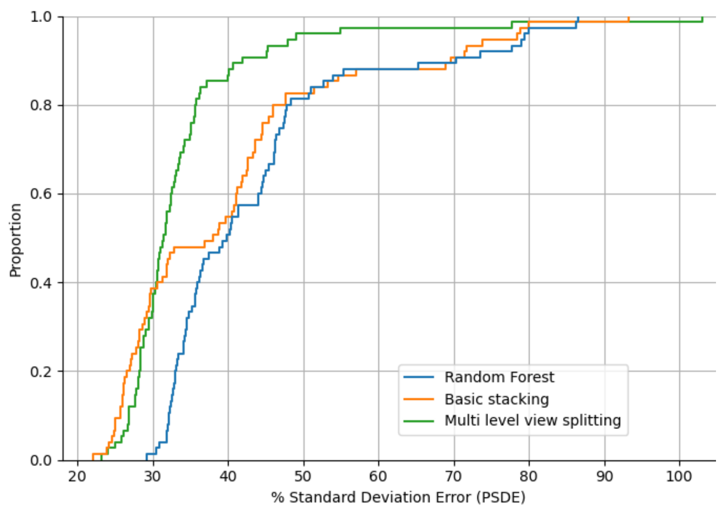

Figure 4.11: A performance comparison of the best singular model, basic model stacking and model stacking with multi-level view splitting

With these results, we see that singular models are indeed outperformed by stacked ensambles. Furthermore, the performance of stacked models is improved by variations like view splitting. With these two observations we can confirm our first two hypotheses.

4.11. Few Bit Precision Modeling

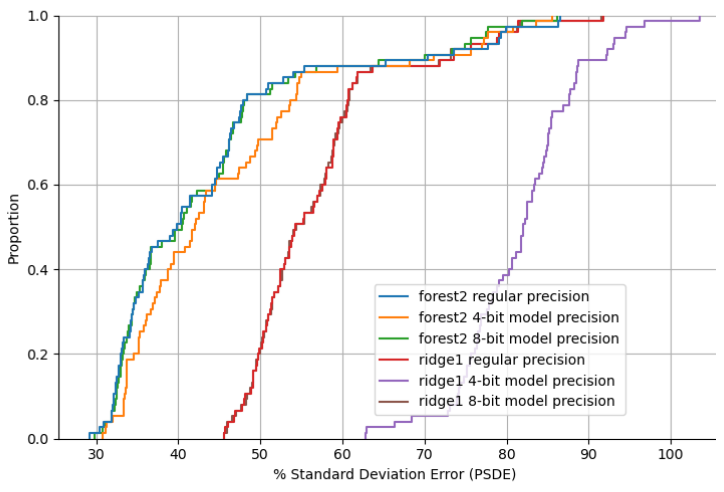

Figure 4.12: Performance of Random Forest and Ridge Regression models with default precision compared to their performance when using 8 and 4 bit value precision

To answer the question of model performance when all values used by the model are restricted to few bits we ran experiments with multiple bit precision settings. From these experiments we see that the performance of models using few bits varies greatly based on the model architecture. While some models like the random forest model only notably drop in performance when reduced to 4 bits, the performance of other models like ridge regression degrade sooner. This can be explained by the fact that models like ridge regression will use multiplications where precision matters greatly for the final result while random forest uses comparisons and additions, both of which are much less susceptible to inaccuracies due to the usage of lower precision values.

An even more interesting finding is the performance of a stacked ensamble when the values of all component models and data is limited. The performance of the ensamble was consistent when using regular, 8 bit and 4 bit precision. This is likely due to the fact that some of the base learners will still have a good performance at low precision combined with a possibly higher diversity between base learners due to the rounding effect of the lower bit precision. It should be noted that the few bit performance is likely strongly related to precision and size of the data set. Our current small data set could be reasonably represented in few bits, but for a larger or more precise data set the amount of bits should be increased to at least reasonably represent model inputs and output. This result was unexpected, and partially goes against our last hypothesis, as it was expected that the performance of stacked models decreased in performance similar to singular model when using a lower bit precision.

5 Conclusion

With the current amount of available data, the HTOL predictions have no real world value, as there is not enough data to train a model that would be able to generalize to other products. On the other hand, the findings regarding the possibility of HTOL prediction, the performance of model stacking variations and the performance of few bit models could have useful applications. As shown by the experiments, the predictive performance of stacked ensambles outperforms the usage of singular machine learning models. We have also shown that multiple variations of stacking can be used with various trade-offs. The most promising of these variations revolves around view splitting, as this technique achieved a 20% relative performance increase compared to basic model stacking. The use of stacking comes at the cost of increased evaluation times, and heavily increased training times. Using the view splitting technique a mean error of 35.7% of the target standard deviation was obtained. Regarding the usage of a lower bit precision on these models we observed a decrease in performance of various singular models when using few bit precision. The tested stacked ensambles showed less change in performance when using few bit precision, likely due to an increased diversity in base learner predictions.

In regards to the data used in this project, multiple issues can be seen. First of all, each given data set is limited to one single product, and as we did not find a way to combine these data sets any trained models cannot generalize to other products. Secondly, if we would be able to generalize the data sets and thus the models created in this work, device failures can still not be predicted as there is no data available on device failures at this time.

Model stacking variations are plenty and combining them yields an even larger set of possibilities. In this project we investigated the variations that we found to be promising and their combinations. This is not exhaustive and other experiments can still be done into different model stacking architectures and combinations of variations.

Additional work that can be done on these topics include the generalization of the models to a larger set of products when more data is available. The findings of this work can also be tested on other data sets with different distributions and larger sizes. Regarding lifetime prediction, research can be done into predicting the remaining lifetime of devices based on device measurements. Such a similar application would likely also benefit from performance increases when using model stacking as seen in this work.

References

[1] Aishwarya Bhandare et al. “Efficient 8-Bit Quantization of Transformer Neural Machine Language Translation Model”. In: *CoRR* abs/1906.00532 (2019). arXiv: `1906.00532`. URL: `http://arxiv.org/abs/1906.00532`.

[2] Paula Branco, Luís Torgo, and Rita Ribeiro. “SMOGN: a Pre-processing Approach for Imbalanced Regression”. In: Sept. 2017.

[3] A.M.P. Canuto et al. “Performance and diversity evaluation in hybrid and non-hybrid structures of ensembles”. In: *Fifth International Conference on Hybrid Intelligent Systems (HIS’05)*. 2005, 6 pp.-. DOI: `10.1109/ICHIS.2005.87`. URL: `https://ieeexplore.ieee.org/abstract/document/1587762`.

[4] Angelos Chatzimparmpas et al. “Empirical Study: Visual Analytics for Comparing Stacking to Blending Ensemble Learning”. In: *2021 23rd International Conference on Control Systems and Computer Science (CSCS)*. 2021, pp. 1–8. DOI: `10.1109/CSCS52396.2021.00008`.

[5] Haimonti Dutta. “Measuring Diversity in Regression Ensembles.” In: Jan. 2009, pp. 2220–2236. URL: `https://www.researchgate.net/publication/220888207_Measuring_Diversity_in_Regression_Ensembles`.

[6] Suyog Gupta et al. “Deep Learning with Limited Numerical Precision”. In: *Proceedings of the 32nd International Conference on Machine Learning*. Ed. by Francis Bach and David Blei. Vol. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, July 2015, pp. 1737–1746. URL: `https://proceedings.mlr.press/v37/gupta15.html`.

[7] Yehuda Koren. “The bellkor solution to the netflix grand prize”. In: *Netflix prize documentation* 81.2009 (2009), pp. 1–10.

[8] Anders Krogh and Jesper Vedelsby. “Neural Network Ensembles, Cross Validation and Active Learning”. In: *Proceedings of the 7th International Conference on Neural Information Processing Systems*. NIPS’94. Denver, Colorado: MIT Press, 1994, pp. 231–238. URL: `https://dl.acm.org/doi/10.5555/2998687.2998716`.

[9] Evelyn Landman, Noam Brousard, and Tamar Naishlos. “A novel approach to in-field, in-mission reliability monitoring based on Deep Data”. In: *2020 IEEE International Reliability Physics Symposium (IRPS)*. 2020, pp. 1–8. DOI: `10.1109/IRPS45951.2020.9128846`. URL: `https://ieeexplore.ieee.org/document/9128846`.

[10] Wouter Loon et al. “View selection in multi-view stacking: Choosing the meta-learner”. In: (Oct. 2020). URL: `https://arxiv.org/pdf/2010.16271.pdf`.

[11] Antonio Leonel Hernández Martínez et al. “Online Remaining Useful Lifetime Prediction Using Support Vector Regression”. In: *IEEE Transactions on Emerging Topics in Computing* 10.3 (2022), pp. 1546–1557. DOI: `10.1109/TETC.2021.3106252`. URL: `https://ieeexplore-ieee-org.tudelft.idm.oclc.org/document/9522047`.

[12] D. Opitz and R. Maclin. “Popular Ensemble Methods: An Empirical Study”. In: (Aug. 1999). DOI: `10.1613/jair.614`. URL: `https://www.jair.org/index.php/jair/article/view/10239`.

[13] Bohdan Pavlyshenko. “Using Stacking Approaches for Machine Learning Models”. In: *2018 IEEE Second International Conference on Data Stream Mining & Processing (DSMP)*. 2018, pp. 255–258. DOI: 10.1109/DSMP.2018.8478522. URL: https://sci-hub.se/10.1109/dsmp.2018.8478522.

[14] R. Polikar. “Ensemble based systems in decision making”. In: *IEEE Circuits and Systems Magazine* 6.3 (2006), pp. 21–45. DOI: 10.1109/MCAS.2006.1688199. URL: https://ieeexplore.ieee.org/document/1688199.

[15] Lakshmana Rao Kalabarige et al. “Multilayer Stacked Ensemble Learning Model to Detect Phishing Websites”. In: (2022). DOI: 10.1109/ACCESS.2022.3194672. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9843994.

[16] Alexander K. Seewald. “Meta-Learning for Stacked Classification”. In: *Austrian Research Institute for Artificial Intelligence* (2002). URL: https://ofai.at/papers/oefai-tr-2002-05.pdf.

[17] Saeed Shafieian and Mohammad Zulkernine. “Multi-layer stacking ensemble learners for low footprint network intrusion detection”. In: *Complex & Intelligent Systems* (2022). DOI: 10.1007/s40747-022-00809-3. URL: https://doi.org/10.1007/s40747-022-00809-3.

[18] Naigang Wang et al. “Training Deep Neural Networks with 8-bit Floating Point Numbers”. In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper_files/paper/2018/file/335d3d1cd7ef05ec77714a215134914c-Paper.pdf.

[19] David Wolpert. “Stacked Generalization”. In: *Neural Networks* 5 (Dec. 1992), pp. 241–259. DOI: 10.1016/S0893-6080(05)80023-1. URL: https://www.researchgate.net/publication/222467943_Stacked_Generalization.

[20] Tianao Wu et al. “Evaluation of stacking and blending ensemble learning methods for estimating daily reference evapotranspiration”. In: *Computers and Electronics in Agriculture* 184 (2021), p. 106039. ISSN: 0168-1699. DOI: https://doi.org/10.1016/j.compag.2021.106039. URL: https://www.sciencedirect.com/science/article/pii/S0168169921000570.

[21] Bernard Ženko, Ljupco Todorovski, and Sašo Džeroski. “A Comparison of Stacking with Meta Decision Trees to Bagging, Boosting, and Stacking with Other Methods”. In: Feb. 2001, pp. 669–670. ISBN: 0-7695-1119-8. DOI: 10.1109/ICDM.2001.989601. URL: https://www.researchgate.net/publication/3940303_A_Comparison_of_Stacking_with_Meta_Decision_Trees_to_Bagging_Boosting_and_Stacking_with_Other_Methods.

[22] Ying Zhang and Chen Ling. “A strategy to apply machine learning to small datasets in materials science”. In: *npj Computational Materials* 4 (May 2018). DOI: 10.1038/s41524-018-0081-z. URL: https://www.nature.com/articles/s41524-018-0081-z.

[23] Yong Zhao and Hans G. Kerkhoff. “A genetic algorithm based remaining lifetime prediction for a VLIW processor employing path delay and IDDX testing”. In: *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*. 2016, pp. 1–4. DOI: 10.1109/DTIS.2016.7483805. URL: https://ieeexplore.ieee.org/document/7483805.

[24] Seng Zian, Sameem Abdul Kareem, and Kasturi Dewi Varathan. “An Empirical Evaluation of Stacked Ensembles With Different Meta-Learners in Imbalanced Classification”. In: *IEEE Access* 9 (2021), pp. 87434–87452. DOI: 10.1109/ACCESS.2021.3088414. URL: https://ieeexplore.ieee.org/document/9452051.

A
The Ratios Problem

Before investigating the problem of predicting the wear of ICs, another problem was proposed. I've spent the first few weeks of this project looking into this problem. The problem consisted of 3 sets of "ringo" values, "HVT", "SVT" and "LVT", and a final value for the total leakage current drawn cumulatively by these groups. I was asked to find the contribution of each group to the total leakage current given that the current for each group depended on their ringo values.

While I did try, I was not able to find a solution to this problem with the data I had available. I did however try many things and realized some things that might be interesting. One of the most interesting finds was the fact that all the data I had was very highly correlated. For example it was possible to predict the ringo values of any group given any other with high accuracy, and the total leakage current could also easily be determined from any single group with a similarly high accuracy.

While this problem is clearly far from a normal supervised machine learning problem as there is no labeled data to train on, I did try various approaches in an attempt to solve this problem. Below I will list some of the most promising approaches I tried and their conclusions.

- The first thing I tried was using linear regression with intercepts on all 3 sets to predict the total current, and then looking at the fitted intercepts for possible contribution ratios for each set. The reasoning behind this is that the intercept of some set would contain the contribution towards the total current of the two left out sets.
- Another approach was to reduce each set of ringo values to 1 feature per set, and then do a linear regression using the 3 calculated features to predict the total current. Using this approach the coefficients for the linear regression belonging to the feature of a set could show a contribution of that set to the total current. The correlation of all ringo values made it easy to create a single feature for each set, and for this I tried multiple ways as well like picking a feature by hand, taking averages and dimensionality reduction methods like principal component analysis. This method also did not give any good results.
- The last attempt worth mentioning was to find groups of samples where 2 out of the 3 groups all had the same or very similar ringo values. Then a linear regression of the final free group could give some indication of how it scales with the total current, and thus doing this for all 3 groups could give ratios. This approach again did not give any good results, in this case I reasoned that the high correlation between the sets might have caused the final free group to also have very similar values and thus any noise would be more significant, resulting in inaccurate linear coefficients.