Simulation-based Evaluation of Feature Selection Methods for Gaussian Process Models

Livia Fingerson

Northwestern University

IEMS399: Independent Study in Industrial Engineering

Advised by Professor Moses Chan

June 13, 2025

1 Introduction

In high-dimensional data analysis, reducing the number of input features can improve model performance by focusing computation on relevant information. Feature selection can reduce overfitting, boost interpretability, and decrease computational costs. As models grow more complex or are applied to increasingly large and noisy datasets, the ability to isolate the most informative features becomes especially valuable.

Gaussian process models are valuable tools for both regression and classification tasks because they can model uncertainty and capture complex, nonlinear relationships. This is achieved through the use of kernel functions, which define the similarity between data points based on their features. However, Gaussian process models can be computationally intensive, especially as the size of the dataset grows. In these applications, efforts to reduce computational cost are important, and feature selection creates an opportunity to minimize unnecessary computations. The cost of computing the full kernel matrix increases with both sample size and dimensionality. Specifically, calculating all pairwise similarities scales as $\mathcal{O}(n^2 d)$, where n represents the number of data points and d refers to the number of dimensions in the data.

This paper evaluates multiple methods of incorporating feature selection within Gaussian process models, using synthetic datasets with controlled sparsity and noise. By testing these methods across a wide range of conditions, this analysis explores how feature selection impacts model performance, with the goal of identifying strategies that maintain or improve predictive accuracy while reducing computational burden and model complexity.

2 Methodology

2.1 Synthetic Data Generation

The simulation output is defined $y_i = f(x_i) + \varepsilon(x_i) + \eta_i$, where $f(x) = x^\top \beta$ represents the linear mean function, $\varepsilon(x) \sim \mathcal{GP}(0, k(x, x'))$ represents structured noise from a Gaussian process, and $\eta \sim \mathcal{N}(0, \sigma^2)$ represents small independent Gaussian noise.

This simulation uses nine unique datasets to test the models on, defined by combinations of active feature proportion and noise. Each generated dataset contains 11,000 samples, large enough for multiple splits to create 5 datasets with the same ground truth. These replications account for variability and aid in assessing generalization. 1000 samples from each dataset are reserved as a validation set for all models. This allows for direct comparison between the results of the models.

Each dataset contains 30 input features (independent variables), with outputs generated via the simulation process. X is generated from a standard uniform distribution.

To construct the linear mean, an integer number of active dimensions is selected. In this simulation, active feature proportions of 0.1, 0.2, and 0.5 are tested. The active feature indices are randomly selected from the 30 features. The coefficients are drawn from a uniform distribution $\mathcal{U}(-5, 5)$. The linear mean $f(x)$ is the product of $X^T\beta$.

For the structured Gaussian process noise, a covariance function K is first created. Structured noise samples are drawn from a multivariate normal distribution $\varepsilon \sim \mathcal{N}(0, K)$. The covariance matrix K is constructed from the ARD kernel described below. To generate structured noise, three of the features used in the linear mean are randomly selected to also influence the Gaussian process covariance structure. To create K, I define the parameters for a radial basis function (RBF) kernel. However, to simulate realistic structured noise, each feature is allowed to have its own lengthscale in the kernel, reflecting the assumption that different features may influence the underlying process at different scales. This is modeled through an RBF kernel with separate lengthscales per active feature, referred to as an Automatic Relevance Determination (ARD) kernel. This takes the following form, where $\boldsymbol{\ell}$ corresponds to a vector of individual lengthscales.

$$K_{ij} = \sigma_{GP}^2 \exp\left(-\frac{1}{2}\left\|\frac{x_i - x_j}{\boldsymbol{\ell}}\right\|^2\right)$$

The kernel variance, σ_{GP}, is fixed at 0.05 to simulate low-variance, structured noise representing mild deviation from the mean function. The lengthscale values are randomly generated from a log-normal distribution with $\mu_\ell = 0$ and $\sigma_\ell = 0.5$. Each selected feature is scaled by its corresponding lengthscale. Then, pairwise Euclidean distances are computed between samples for each feature. This gives us the full covariance function

K, which defines the multivariate normal distribution used to generate the structured noise $\varepsilon \sim \mathcal{N}(0, K)$.

Lastly, small independent Gaussian noise is added, drawn independently from a zero-mean Gaussian distribution with variance σ^2_{noise}. σ^2_{noise} is determined by a proportion of the variance of the linear mean.

$$\sigma^2_{noise} = c \times \mathrm{Var}(f(x))$$

To calculate this, $Var(f(x))$, denoting the variance of the linear mean function, is estimated by $\frac{1}{n-1}\sum_{i=1}^{n}(f(x_i) - \bar{f})^2$, where $\bar{f}$ is defined to be $\frac{1}{n}\sum_{i=1}^{n} f(x_i)$. In this study, c assumes the values $[0.1, 0.5, 1]$ in order to test the effect of different levels of noise on the feature selection methods.

The final output y is a sum of the linear mean function, the structured Gaussian process noise, and the small independent Gaussian noise.

$$y = f(x) + \varepsilon(x) + \eta$$

Synthetic data allows for precise control over ground truth sparsity and noise, resulting in fair and reproducible comparison across the methods of feature selection.

Summary of Synthetic Dataset Characteristics

- Dimensionality: 30 features
- Lengthscale Values: Random, positive, and independently assigned per feature
- Proportion of active features: [0.10, 0.20, 0.50]
- Sample size: [50, 100, 500, 1000]
- i.i.d. Gaussian noise (as a percentage of variance, c): [0.1, 0.5, 1]

2.2 Gaussian Process Feature Selection Methods

To compare the performance of sparse feature selection methods in Gaussian process regression, I implemented the following five methods using a custom Python class and the GPflow Python library.

Each method is fit using GPflow's built-in Scipy optimizer, which computes the gradients of the loss function with respect to the trainable variables. This minimizes the negative log likelihood.

The number of observations used for training the Gaussian process, n, varies between 50, 100, 500, and 100, and the number of testing observations $n' = 1000$ across all variations. Each combination of sparsity and noise in the synthetic dataset

2.2.1 Standard Optimization

This method applies Gaussian process regression without any form of feature selection. All input features are included, and kernel parameters are shared across dimensions. The loss function used is the negative log marginal likelihood of the multivariate normal distribution.

$$\ell(\theta, \sigma^2, \beta; X, y) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|C| - \frac{1}{2}(y - X\beta)^T C^{-1}(y - X\beta)$$

Here, $C = K + \sigma^2 I$ is the total covariance matrix, composed of the kernel matrix K and the noise variance σ^2 multiplied by the identity matrix. During optimization, the loss is minimized with respect to the kernel hyperparameters θ, the noise variance hyperparameter σ^2, and the mean function parameters β.

2.2.2 ARD Kernel Optimization

This method does not directly select features; however, enabling Automatic Relevance Determination (ARD) in the kernel allows each input feature to be assigned its own lengthscale parameter. The kernel learns to reduce the influence of irrelevant features by increasing their corresponding lengthscales. As a lengthscale grows large, the feature contributes less to the covariance function, diminishing its impact on predictions.

As in the standard optimization, the loss function is the negative log marginal likelihood of the multivariate normal distribution. However, the key difference is that $C = K_{ARD} + \sigma^2 I$, where K_{ARD} is the covariance matrix produced using a radial basis function (RBF) kernel with distinct lengthscales for each input feature. The loss is minimized with respect to the kernel hyperparameters θ (one per input dimension), the noise variance σ^2, and the mean function parameters β.

2.2.3 Lasso Least Angle Regression Feature Selection Before Standard Optimization

This method fits a Lasso Least Angle Regression (LARS) to the data, with penalty strength parameter λ selected via 5-fold cross validation within the training data. This regularization technique penalizes the magnitude of the coefficients through a penalty term added to mean squared error.

$$\beta = \arg\min_{\beta} \|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

LARS Lasso is a more computationally efficient variation of the traditional Lasso algorithm. This algorithm gradually adds features to the model, continuing until the correlation between an additional variable and the original feature is equal. This operation pushes coefficients towards 0, but a tolerance of $1e^{-4}$ is used to treat small coefficients as effectively zero. The indices of the non-zero coefficients become the selected

features, and the GP mean function assumes the values of the non-zero coefficients and the intercept. The GP is then fit to the subset of selected features, with kernel parameters uniform across all features. The mean function coefficients and intercept are not trained after they have been fit using the Lasso Regression. The Gaussian process is then trained using the loss function defined in the standard optimization method.

2.2.4 Lasso Least Angle Regression Before ARD Kernel Optimization

This method also begins by fitting a Lasso Least Angle Regression (LARS) to the training data, using 5-fold cross validation to select the penalty strength parameter λ. As in the previous method, this technique drives some coefficients to zero, and the indices of non-zero coefficients define the subset of selected features. A tolerance of $1e^{-4}$ is used to determine the non-zero values.

The mean function of the Gaussian process uses the coefficients and intercept from the Lasso fitting, and these parameters are not re-optimized.

The Gaussian process is then trained using the same negative log-likelihood loss function described in the previous method, but the covariance matrix K is computed using an ARD kernel, which allows each selected feature to have its own lengthscale parameter. This allows the magnitude of individual lengthscale parameters to control the influence of a feature on the output, with longer lengthscales smoothing a feature's contribution. This method provides a view into how feature selection (via Lasso) and feature relevance (via ARD) interact.

2.2.5 L1-Penalized Gaussian Process Optimization

This method is inspired by Lasso regularization, and it aims to penalize the size of the coefficients within the Gaussian process optimization directly. To avoid using λ as a trainable parameter, which introduces logical errors in the optimization process, λ is selected via 5-fold cross-validation.

A two-stage grid search is used to determine the optimal value of λ. First, a coarse grid of 12 values between $[10^{-1}, 10^{0.5}]$, or $[0.1, 3.16]$, is tested, and the value of λ_c^* that yields the lowest validation root mean squared error (RMSE) is then tested over a finer grid. The fine grid is constructed using 6 values between between $[10^{0.5\lambda_c^*}, 10^{2\lambda_c^*}]$. The value λ^* with the lowest overall cross-validation RMSE is selected and used as the penalty strength within the L1-penalized Gaussian process loss function, shown below.

$$\ell(\theta, \sigma^2, \beta; X, y, \lambda^*) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|C| - \frac{1}{2}(y - X\beta)^T(C)^{-1}(y - X\beta) - \lambda^*\|\beta\|_1$$

The final term encourages sparsity by penalizing large coefficients in the mean function, allowing for joint optimization of predictive accuracy and feature selection for Gaussian process models. Because the L1

penalty does not always drive coefficients exactly to zero, a dynamic threshold is applied after training: features with linear mean coefficients smaller than 5% of the largest absolute coefficient are deselected to provide a practical approximation of feature selection.

2.3 Evaluation Metrics

Through the simulation, the following metrics are recorded for each of the 5 methods for each sample.

- **Prediction error:**

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

  Root mean squared error (RMSE) is computed on both the training and testing datasets. Low RMSE on training data suggests that the model captures the patterns within the training set effectively. Low RMSE on testing data reflects good generalization to unseen data. Since each model size variation is evaluated on the same reserved test set, testing RMSE values can be directly compared across methods and sample sizes.

- **Selected features:**

  For methods that perform feature selection, the model returns a subset of the original 30 features. This allows for comparison of which features are selected across models trained on the same ground truth.

- **Feature precision:**

$$\text{Precision} = \frac{\#\text{ of active selected features}}{\#\text{ of total selected features}}$$

  Precision measures the proportion of selected features that are active in the data-generation process. High precision indicates that the model selects relevant features while avoiding false positives.

- **Feature recall:**

$$\text{Recall} = \frac{\#\text{ of active selected features}}{\#\text{ of total active features}}$$

  Recall measures the proportion of active features that are correctly identified by the model. High recall shows that the model is effective at recovering relevant features, regardless of how many irrelevant features are also selected.

- **Estimation error of coefficients:**

$$\varepsilon_\beta = \|\beta - \hat{\beta}\|_2$$

Estimation error provides a measure of the difference between the model's estimated linear mean function coefficients and the true coefficients used to generated the linear component of the data. It is computed using the L2 norm. Low estimation error indicates that the model more accurately captures the linear structure within the data.

- **Computation time:**

  To evaluate the efficiency of the methods, the following metrics are recorded.

  - Tuning time: seconds required for cross-validation to choose optimal penalty strength λ, if applicable
  - Fit time: seconds required to fit the Gaussian process to the training data.
  - Prediction time: seconds required for model to form predictions on the test set of 1000 samples. This can be compared across models trained on different sizes with the same ground truth.
  - Total runtime: seconds required for tuning, fitting, and prediction steps

3 Results

3.1 Sample Size

In this section, the assumption that increasing sample size improves predictive accuracy is tested. RMSE on the reserved test set of 1000 samples is compared across models trained on datasets of size [50, 100, 500, and 1000], with five replications for each size.

As expected, test RMSE decreases as sample size increases for all model types, confirming that larger datasets generally improve predictive accuracy across all methods tested. In Fig. 1, predictive error on the test set is compared across models for each sample size, using a dataset with noise level $c = 0.5$ and active feature proportion of 0.10. The standard Gaussian process (`std`) and Automatic Relevance Determination (`ard`) models show the largest gains in performance with increasing size, particularly between 50 and 500 samples. However, the confidence intervals displayed within the error bars indicate that variability is higher at smaller sample sizes as well.

In contrast, the Lasso-based models (`lasso_std` and `lasso_ard`) achieve consistently low test error across all sizes, demonstrating stronger regularization benefits in sparse, low-data settings. The direct optimization method, `l1_gp`, shows relatively flat error across dataset sizes, suggesting that its performance is less sensitive to the quantity of data and may underfit in this setting.

By sample size 1000, all models except `l1_gp` converge to a similar performance, but `lasso_ard` and `lasso_std` reach this level of error with substantially fewer samples, highlighting the efficiency of these methods for smaller datasets.

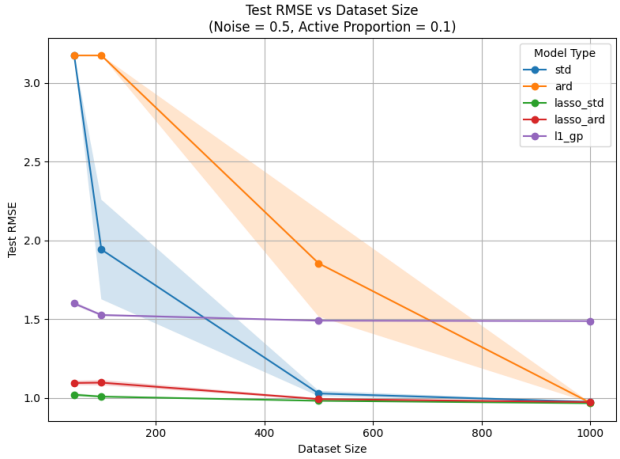

Figure 1: Test RMSE by Sample Size

3.2 Noise

The relationship between the scale of independent Gaussian noise and predictive accuracy is examined in this section. The hypothesis is that higher noise levels worsen performance, resulting in higher RMSE values on the reserved test set. To evaluate this, models were trained on datasets with noise proportions of 0.1, 0.5, and 1.0, with five replications per setting. Sample size and proportion of active features were held constant at 100 and 0.10, respectively.

Across all model types, test RMSE increases as the proportion of underlying noise increases, supporting the assumption that greater variability within the data interferes with predictive accuracy. The `std` and `ard` models display the largest increases in predictive error as noise increases, with `ard` showing a particularly sharp jump between 0.1 and 0.5. The direct optimization method `l1_gp` follows a similar pattern to the `std` model, although the increase in noise from 0.1 to 0.5 has less influence on RMSE. The `lasso_std` and `lasso_ard` models perform similarly to each other. Across the datasets with little noise, they achieve a much lower RMSE value, and display an almost linear increase in prediction error as noise increases. These results demonstrate that increasing noise impacts model performance, although each model's sensitivity varies. Methods incorporating feature-selection show a more stable performances as noise increases.

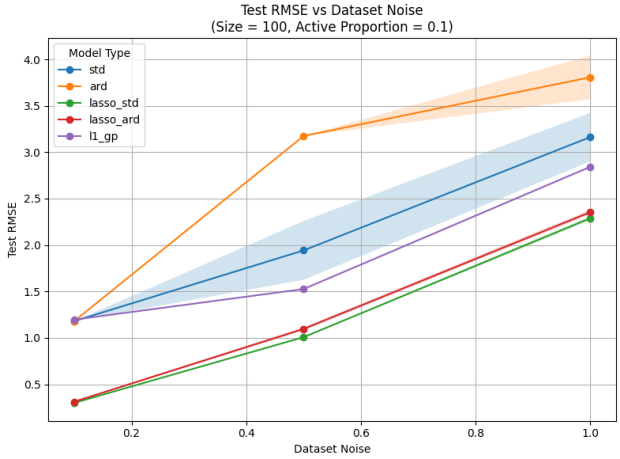

Figure 2: Test RMSE by Dataset Noise

3.3 Feature Sparsity

This section assesses how the proportion of active features in the data influences predictive accuracy. The baseline assumption that is evaluated is that models perform better when the dataset is sparse, meaning that the signal is concentrated among fewer features. To evaluate this, RMSE on the test set is compared across models trained on datasets with active feature proportions ranging from 0.10 to 0.50. Fig. 3 shows results with sample size fixed at 50 and noise level set to 1.0.

In this plot, test RMSE increases with the number of active features across all model types. The `lasso_std` and `lasso_ard` models achieve the lowest RMSE overall, particularly at the lowest active proportions. This suggests that they are more effective at identifying active signals in sparse conditions. The `ard` and `std` models show higher test error and a steep increase in RMSE as proportion of active features increases from 0.10 to 0.20. The `l1_gp` method performs similarly to the `ard` and `std` models, although it has a lower initial error.

These results show that as the number of active features increases, predictive performance declines across all models, but those incorporating regularization and feature selection remain more robust to lower sparsity.

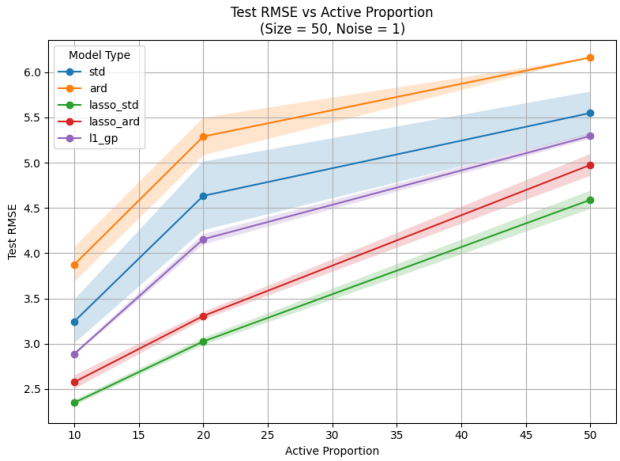

Figure 3: Test RMSE by Active Proportion

3.4 Feature Recovery Performance

This section evaluates how accurately each method identifies the true active features under varying noise levels. Performance is measured using two metrics: precision (the proportion of features that the model selects that are actually active) and recall (the proportion of all actually active features that are successfully identified by the model). High precision indicates that a method avoids false positives, while high recall reflects its ability to recover all relevant features, even if some irrelevant ones are also included. The ideal feature selection method achieves both high precision and high recall. However, in this setting, recall may be the more important metric, as it reflects a model's ability to avoid missing relevant signals.

Figs. 4 and 5 show precision and recall results across noise levels of 0.1, 0.5, and 1.0, with the training size fixed at 500 and the active proportion fixed at 0.10. These settings represent sparse conditions where feature selection is important but performance may vary due to noise.

Notably, the `l1_gp` model reports precision and recall values of zero across all noise levels. This may indicate that the model is not identifying any active features or that selected features are not being properly recorded. Additional debugging is necessary to determine whether this is due to error in implementation or a limitation of the model's structure. The `l1_gp` results will be ignored in this section due to this error.

For recall (Fig. 4), the `std` and `ard` models consistently include all relevant features, because they perform no feature selection. The `lasso_ard` and `lasso_std` methods have perfect recall at low noise levels (0.1) and at high noise levels (1.0), although they struggle at moderate levels of noise. This pattern suggests instability in the feature recovery process when noise is neither negligible nor overpowering.

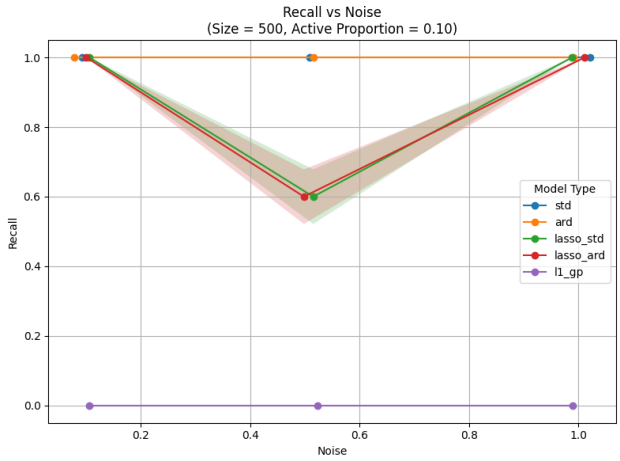

Figure 4: Recall by Noise

Looking at precision (Fig. 5), the `std` and `ard` models are consistent with precision of 0.1. Again, this is because they do not perform feature selection, so they always include the inactive features, resulting in low precision. The `lasso_ard` and `lasso_std` models exhibit the highest precision at low noise, but their performance drops under more variable conditions.

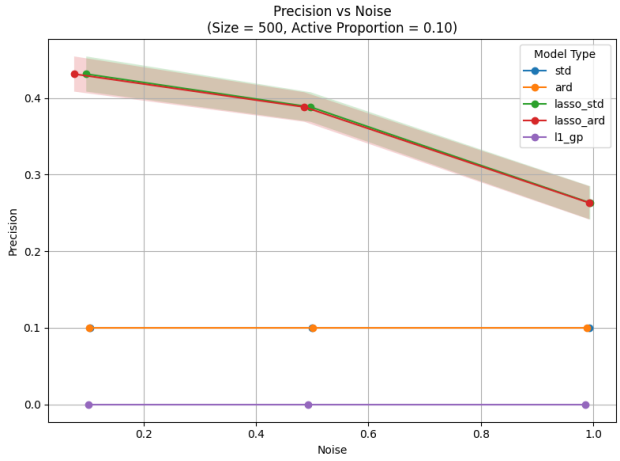

Figure 5: Precision by Noise

Overall, these results highlight a tradeoff in sparse settings, where models that do not perform feature selection consistently include all active and inactive features, while feature-selecting models are more sensitive to noise and may fail to recover active features under certain conditions.

3.5 Computational Efficiency

This section examines the differences in computation time required to apply each method across the fitting and prediction steps.

Fig. 6 and Fig. 7 report prediction and fit times, respectively, for each model, grouped by the proportion of active features in the training data. Prediction time refers to the time required to produce predictions on the test set of size 1000, while fit time reflects the time required to train the model on the corresponding training set.

In terms of prediction time (Fig. 6), all models are relatively fast, with average times below one second. The `std` and `ard` models are the most consistent, with minimal variability regardless of feature sparsity. In contrast, the `lasso_std`, `lasso_ard`, and `l1_gp` models display greater variation, particularly at an active proportion of 0.20.

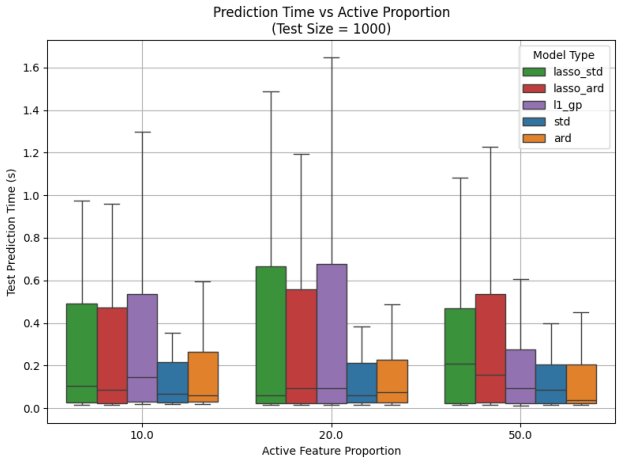

Figure 6: Prediction Time by Active Proportion

However, a different trend can be observed when looking at fit time (Fig. 7). The `l1_gp` model is the most computationally expensive to train overall, with both mean and variability increasing as the number of active features rises. The `ard` model also becomes more costly to train as sparsity decreases. Meanwhile, the `lasso_std`, `lasso_ard`, and `std` models show relatively low and stable fit times across conditions.

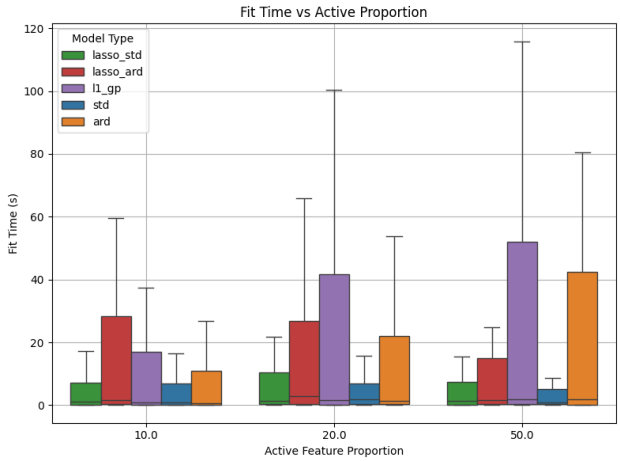

Figure 7: Fit Time by Active Proportion

Together, these results illustrate the tradeoff between model complexity and computational burden. While methods like `l1_gp` and `ard` offer more complex structure and individualized lengthscales, they require a higher computational cost. The `lasso_ard` model provides a balance, in using feature selection to reduce the number of individual kernel parameters that must be optimized. Among the models compared, `std` achieves the most consistently low prediction and training time, although it offers limited flexibility in modeling complex data.

4 Discussion

Within this study, Gaussian process models with different approaches for handling sparse data were examined across simulated datasets with variations in sample size, noise, and sparsity. Results show that incorporating Lasso-based feature selection in the fitting of a Gaussian process model improves predictive accuracy, especially in sparse and low-data settings. The ability to filter out irrelevant features before model fitting allows for more stable and interpretable results.

However, this analysis revealed limitations especially in the direct optimization (`l1_gp`) approach. This method reported precision and recall values of zero across all settings, indicating either that the model is not identifying any active features or that selected features are not being properly recorded. Additional debugging is necessary to determine whether this is due to error in implementation or a structural issue with the method itself.

In addition, within the direct optimization approach, the current process of finding an optimal lambda value is extremely computationally intensive and offers variable results. Future work in integrating L1-

regularization within Gaussian process optimization could explore more efficient alternatives, such as estimating λ using Bayesian methods like Markov Chain Monte Carlo (MCMC).

5 Conclusion

This study compared several Gaussian process models and feature selection methods under varying conditions of sample size, noise, and feature sparsity. Across simulations, Lasso-based models (`lasso_std` and `lasso_ard`) demonstrated strong predictive performance and efficient feature recovery in sparse, low-data settings. Standard Gaussian processes (`std`) demonstrated comparable results on larger datasets, but struggled when sample size was small. The ARD approach (`ard`) offers more flexibility than the standard Gaussian process, but at an increased computation cost, especially as the proportion of active features increases. Lastly, the direct optimization approach (`l1_gp`) produced promising results but encountered implementation issues that limited its interpretability.

Overall, Gaussian process models that incorporate feature-selection into the fitting and optimization process offer more robust performance in sparse and low-sample size conditions, and enhance model interpretability.