# INF367A Project 1

Naphat Amundsen

April 3, 2020

## Introduction

This project is about linear regression using automatic relevance determination (ARD) when doing linear regression. The idea of ARD is to use a flexible prior to push weights of the irrelevant features to zero while not penalizing relevant features, as opposed to L2 regularization and similar methods which suppresses the weights to all features equally. ARD can be viewed as weighted regularization with respect to the features. The ARD weights are then parameters that needs to be optimized as well.

Assume that we have observed $n$ pairs $(x_i, y_i)$ where $x_i$ are the feature values and $y_i$ is the label. Let $w \in \mathbb{R}^d$ be the regression weights. The likelihood is

$$P(y|x, w, \beta) = \prod_{i=1}^{n} N(y_i|w^T x_i, \beta^{-1}) \tag{1}$$

The noise precision $\beta$, is modelled with a Gamma prior:

$$P(\beta) = \text{Gamma}(\beta|a_0, b_0) \tag{2}$$

where $a_0$ and $b_0$ are user defined hyperparameters.

The prior for the weights $w$ is a Gaussian distribution

$$P(w|\alpha_1, \dots, \alpha_d) = \prod_{j=1}^{d} N(w_j|0, \alpha_j^{-1}) \tag{3}$$

Comparing this to the standard way, the precision of the weights is not just a prior scalar, but an actual distribution. The prior for $\alpha_j$ is a Gamma distribution.

$$P(\alpha_j) = \text{Gamma}(\alpha_j|c_0, d_0) \quad \forall j = 1, \dots, d \tag{4}$$

where $c_0$ and $d_0$ are user defined hyperparameters.

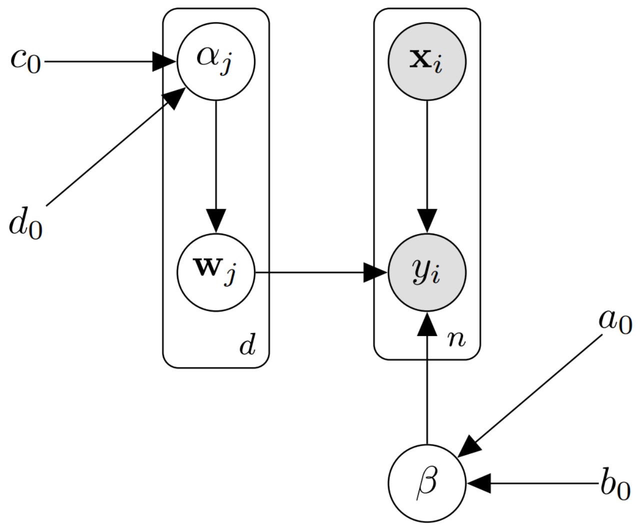


Figure 1: Plate diagram of the regression model with ARD prior. Variables $x_i$ , and $y_i$ are observed; We are interested in the posterior distribution of parameters $w$, $\alpha$ and $\beta$. The constants $a_0$ , $b_0$, $c_0$ , and $d_0$ are user-defined hyperparameters

The components so far can be represented as a Bayesian network, illustrated in Figure 1. The posterior can be conveniently deducted from the diagram to be

$$P(w, \alpha, \beta | y, x) \propto \prod_{i=1}^{n} P(y_i | w, x_i, \beta) \prod_{j=1}^{d} \left[ P(w_j | \alpha_j) P(\alpha_j) \right] P(\beta). \tag{5}$$

## 1 Gibbs sampling and conjugate priors

Gibbs sampling will be used to approximate the optimal ARD regression model parameters. Gibbs sampler requires the full conditional distributions for the model parameters; $P(w|y, x, \alpha, \beta)$, $P(\alpha|y, x, w, \beta)$, $P(\beta|y, x, w, \alpha)$, where $\alpha = (\alpha_1, \ldots, \alpha_d)$.

### 1.1 Deriving conjugate prior for $w$

$$P(w|y, x, \alpha, \beta) \propto \prod_{i=1}^{n} P(y_i | w, x_i, \beta) \prod_{j=1}^{d} P(w_j | \alpha_j)$$

$$= N(y_i | X^T w) \prod_{j=1}^{d} N(w_j | 0, (\sqrt{\alpha_j})^{-1})$$

$$\propto \exp\left( -\frac{1}{2} \beta (y - X^T w)^T (y - X^T w) \right) \prod_{j=1}^{d} \exp\left( -\frac{1}{2} w^2 (\sqrt{\alpha_j})^2 \right)$$

$$= \exp\left( -\frac{1}{2} \beta (y - X^T w)^T (y - X^T w) - \frac{1}{2} \sum_{j=1}^{d} w^2 \alpha \right)$$

$$= \exp\left(-\frac{1}{2}\beta(y - X^T w)^T (y - X^T w) - \frac{1}{2} w^T D w\right)$$

where $D$ is a $d \times d$ diagonal matrix with the values $\alpha_1, \alpha_2, \ldots, \alpha_d$

Completing the square seems reasonable at this point, now the goal is to complete the square, that is get the exponent in the form $\frac{1}{2} w^T A w + k^T w$. For convenience we will take the logarithm to remove the base exponential:

$$\begin{aligned}
\Rightarrow \log P(w|y, x, \alpha, \beta) &\propto -\frac{1}{2}\beta\left(y^T y - y^T X^T w - w^T X y + w^T X X^T w\right) - \frac{1}{2} w^T D w \\
&\propto -\frac{1}{2}\left[-2\beta y^T X^T w + \beta w^T X X^T w\right] - \frac{1}{2} w^T D w \\
&= \beta y^T X^T w - \frac{1}{2}\beta w^T X X^T w - \frac{1}{2} w^T D w \\
&= -\frac{1}{2}\left[w^T \beta X X^T w + w^T D w\right] + \beta y^T X^T w \\
&= -\frac{1}{2} w^T \underbrace{\left(\beta X X^T + D\right)}_{= A} w + \underbrace{\beta y^T X^T}_{= k^T} w
\end{aligned}$$

Then by completing the square we obtain the parameters $S$ (covariance) and $m$ (mean) for a Multivariate Gaussian that is proportional to $P(w|y, x, \alpha, \beta)$:

$$\begin{aligned}
S &= A^{-1} = \left(\beta X X^T + D\right)^{-1} \\
m &= A^{-1} k = S \beta X y \\
&\underline{\underline{\Rightarrow P(w|y, x, \alpha, \beta) \propto \mathcal{N}(w|m, S)}}
\end{aligned} \tag{6}$$

## 1.2 Deriving conjugate prior for $\alpha$

$$\begin{aligned}
P(\alpha|y, x, w, \beta) &\propto \prod_{j=1}^{d} P(w_j|\alpha_j) P(\alpha_j) \\
&= \prod_{j=1}^{d} N(w_j|0, \alpha^{-1}) \,\mathrm{Gamma}(a_j|c_0, d_0) \\
&\propto \prod_{j=1}^{d} \frac{1}{\sqrt{\alpha_j^- 1}} \exp\left(-\frac{1}{2}\left[\frac{w_j}{\sqrt{\alpha}^{-1}}\right]^2\right) \alpha_j^{c_0 - 1} \exp\left(-d_0 \alpha_j\right) \\
&= \prod_{j=1}^{d} \exp\left(-\frac{w_j^2}{2}\alpha_j - d_0 \alpha_j\right) \alpha_j^{c_0 - 1 + 1/2} \\
&= \prod_{j=1}^{d} \exp\left(-\alpha_j\left(\frac{w_j^2}{2} + d_0\right)\right) \alpha_j^{c_0 - 1 + 1/2}
\end{aligned}$$

$$\propto \underline{\underline{\prod_{j=1}^{d} \text{Gamma}(\alpha_j | c_0 + \frac{1}{2}, \frac{w_j^2}{2} + d_0)}} \tag{7}$$

To sample an $\alpha_j$ from the distribution we simply sample from $\text{Gamma}(\alpha_j | c_0 - \frac{1}{2}, \frac{w_j^2}{2} + d_0)$.

### 1.3 Deriving conjugate prior for $\beta$

$$\begin{aligned}
P(\beta | y, x, w, \alpha) &\propto P(\beta) \prod_{i=1}^{n} P(y_i | w, x_i, \beta) \\
&= \text{Gamma}(\beta | a_0, b_0) \prod_{i=1}^{n} N(y_i | w^T x_i, \sqrt{\beta^{-1}}) \\
&= \beta^{a_0 - 1} \exp(-b_0 \beta) \prod_{i=1}^{n} \frac{1}{\sqrt{\beta^{-1}}} \exp\left( -\frac{1}{2} \left( \frac{y_i - w^T x_i}{\sqrt{\beta^{-1}}} \right)^2 \right) \\
&= \beta^{a_0 - 1} \exp(-b_0 \beta) \left( \beta^{1/2} \right)^n \exp\left( -\frac{1}{2} \sum_{i=1}^{n} \beta (y_i - w^T x_i)^2 \right) \\
&= \beta^{a_0 - 1} \beta^{n/2} \exp\left( -b_0 \beta - \frac{\beta}{2} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \right) \\
&= \beta^{a_0 - 1 + n/2} \exp\left( -\beta \left( b_0 + \frac{1}{2} \sum_{i=1}^{n} (y_i - w^T x_i)^2 \right) \right) \\
&\propto \underline{\underline{\text{Gamma}(\beta | a_0 + n/2, b_0 + \frac{1}{2} \sum_{i=1}^{n} (y_i - w^T x_i)^2)}}
\end{aligned} \tag{8}$$

### 1.4 Implementation

The core idea of the implementation is simple. To sample $w, \alpha, \beta$ we simply do:

1. Initialize $(w^{(1)}, \alpha^{(1)}, \beta^{(1)})$
2. For $t = 1, \ldots, T$
    - 2.1. sample $w^{(t+1)} \sim P(w | \alpha^{(t)}, \beta^{(t)})$
    - 2.2. sample $\alpha^{(t+1)} \sim P(\alpha | w^{(t+1)})$
    - 2.3. sample $\beta^{(t+1)} \sim P(\beta | w^{(t+1)})$

The code is implemented in an object oriented style, and pseudocode below represents the source code. For simplicity of this task, the only basis functions available are polynomial kernels as Scikit-Learn has a class called PolynomialFeatures [1] for them. The process of fitting a model to a given dataset $X$ and the labels $y$ primarily consists of two steps; the initialization phase, the sampling phase.

Let $X$ be a dataset with $n$ number of data points with features $f_1, f_2, \ldots, f_d$.

### 1.4.1 Initialization phase

This phase consists of initializing the first values for the parameters. The initial values for $w, \alpha, \beta$ are sampled from

**Algorithm 1** LinearRegressionARD.init_fit, method to initialize variables before fitting model using Gibbs sampling, Python-pseudocode

```
Input: self, $X$: datapoints, $y$: labels, degree: polynomial degree, n_gibbs: number of sampling iterations
 1: self.X ← preprocessed $X$ by using Scikit-Learn's PolynomialFeatures given polynomial degree
 2: self.d ← number of dimensions of the preprocessed $X$
 3:
 4: self.ws ← empty array with shape (n_gibbs, self.d)
 5: self.alphas ← empty array with shape (n_gibbs, self.d)
 6: self.betas ← empty array with shape (1, n_gibbs)
 7:
 8: // Initialize parameter values
 9: self.ws[0] ← sample self.d values from prior (3)
10: self.alphas[0] ← sample self.d values from prior (2)
11: self.betas[0] ← sample a value from prior (4)
```

### 1.4.2 Sampling phase

Directly following the initialization phase:

**Algorithm 2** LinearRegressionARD.fit, method to do Gibbs sampling, Python-pseudocode

```
Input: self, $X$: datapoints, $y$: labels, n_gibbs: number of sampling iterations
1: for t=1 to t=n_gibbs do
2:     self.ws[t] ← sample self.d values from $P(w|\alpha^{(t-1)}, \beta^{(t-1)})$
3:     self.alphas[t] ← sample self.d values from $P(\alpha|w^{(t)})$
4:     self.betas[t] ← sample a value from $P(\beta|w^{(t)})$
5: end for
```

### 1.4.3 Parameters for finalized model

Now, the goal is to do linear regression, i.e. we want a finalized model from our training that can be used for prediction. The sampling process will result with trace plots for the sampled parameters which we can visually inspect.

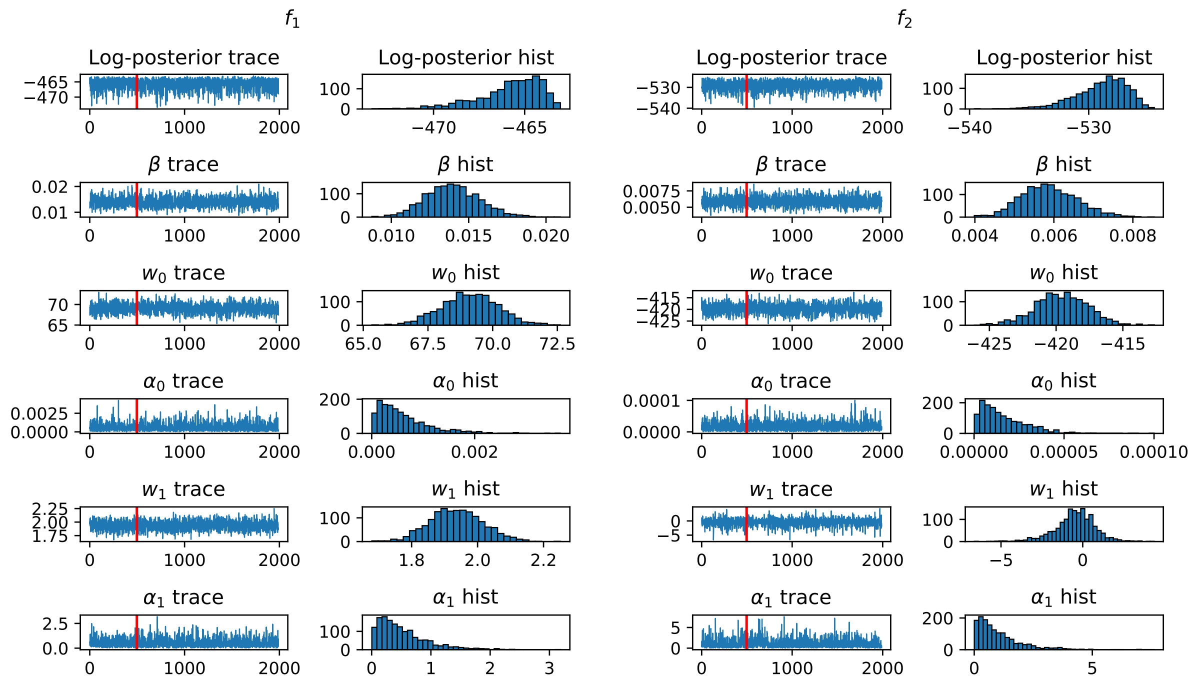


Figure 2: Here we see the trace plots using the implementation. Noisy data generated from the functions $f_1$ and $f_2$ are visualized in Figure 3 along with $f_1$ and $f_2$. The red line represents the fixed burnout cutoff of the first quarter of the samples. The histograms represents the distribution of samples after the cutoff. The sampling process in the plots consisted of 2000 iterations.

We are interested in the model weights $w_0, w_1, \ldots, w_d$. In order to determine what the final values for $w_0, w_1, \ldots, w_d$ should take we simply take the argmax of the histograms, i.e selecting the most frequent binned-value is sampled for each $w_i$. The motivation is that this is an discrete analogue to maximum likelihood. Hence we choose the most likely values, i.e. the most frequent ones. From visual inspection of the histograms, the argmax values are very reasonable and almost matches the true parameters for both $f_1$ and $f_2$ (see Figure 3).

## 2 Simulation study

To test the regression using ARD against the counterpart which does not use ARD we simulate some datasets to test on. The generated data will be generated using multivariate polynomials, as the implementations of the regression models are only capable of fitting polynomials.

### 2.1 The datasets

The simulated data will be generated from 8 polynomial functions:

$$f_1(x) = 2x + 69 \tag{9}$$

$$f_2(x) = 4x^3 + 2x^2 - 420 \tag{10}$$
$$f_3(x, y) = -4x + 20y^2x + 69 \tag{11}$$
$$f_4(x, y) = 69x + 69y - 420 \tag{12}$$
$$f_5(x, y) = x^2 - 2y^2 \tag{13}$$
$$f_6(x, y) = xy - x^2 + y^2 - 420 \tag{14}$$
$$f_7(x, y) = -x^3 + 42x^2 - 20x - y^3 + 42y^2 - 20y + 69 \tag{15}$$
$$f_8(x, y) = 2x^3 - y^3 - 3xy^2 + 3x^2y + x^3 - 3yx + 69 \tag{16}$$

The method of generating the data is simple. For each function $f_i$ do:

1. Randomly generate some data $X$ from the domain of $f_i$
2. Obtain response values $y$ by inputting $X$ into $f_i$
3. Offset each value of $y$ with gaussian noise (same noise distribution for all elements in $y$)

The domain sampling distribution and noise distribution with respect for each function are as follows:

Table 1: Note that the domain distributions are univariate even though the some of the functions are multivariate. Values from the domain distributions are simply sampled repeatedly for each function variable, in this case $x$ and $y$.

| function | domain distribution | noise distribution |
|---|---|---|
| $f_1(x)$ | Exp(rate = 10) | $N(0, 9)$ |
| $f_2(x)$ | $N$(mean = −1.69, std = 1.2) | $N(0, 12)$ |
| $f_3(x, y)$ | Exp(rate = 5) | $N(0, 22)$ |
| $f_4(x, y)$ | $N$(mean = 69, std = 4) | $N(0, 420)$ |
| $f_5(x, y)$ | Unif(min = −2, max = 20) | $N(0, 69)$ |
| $f_6(x, y)$ | $N$(mean = 0, std = 8) | $N(0, 22)$ |
| $f_7(x, y)$ | Unif(min = −2, max = 36) | $N(0, 420)$ |
| $f_8(x, y)$ | Unif(min = −6, max = 6) | $N(0, 69)$ |

Then generating 128 data samples for each function produces the following plots:

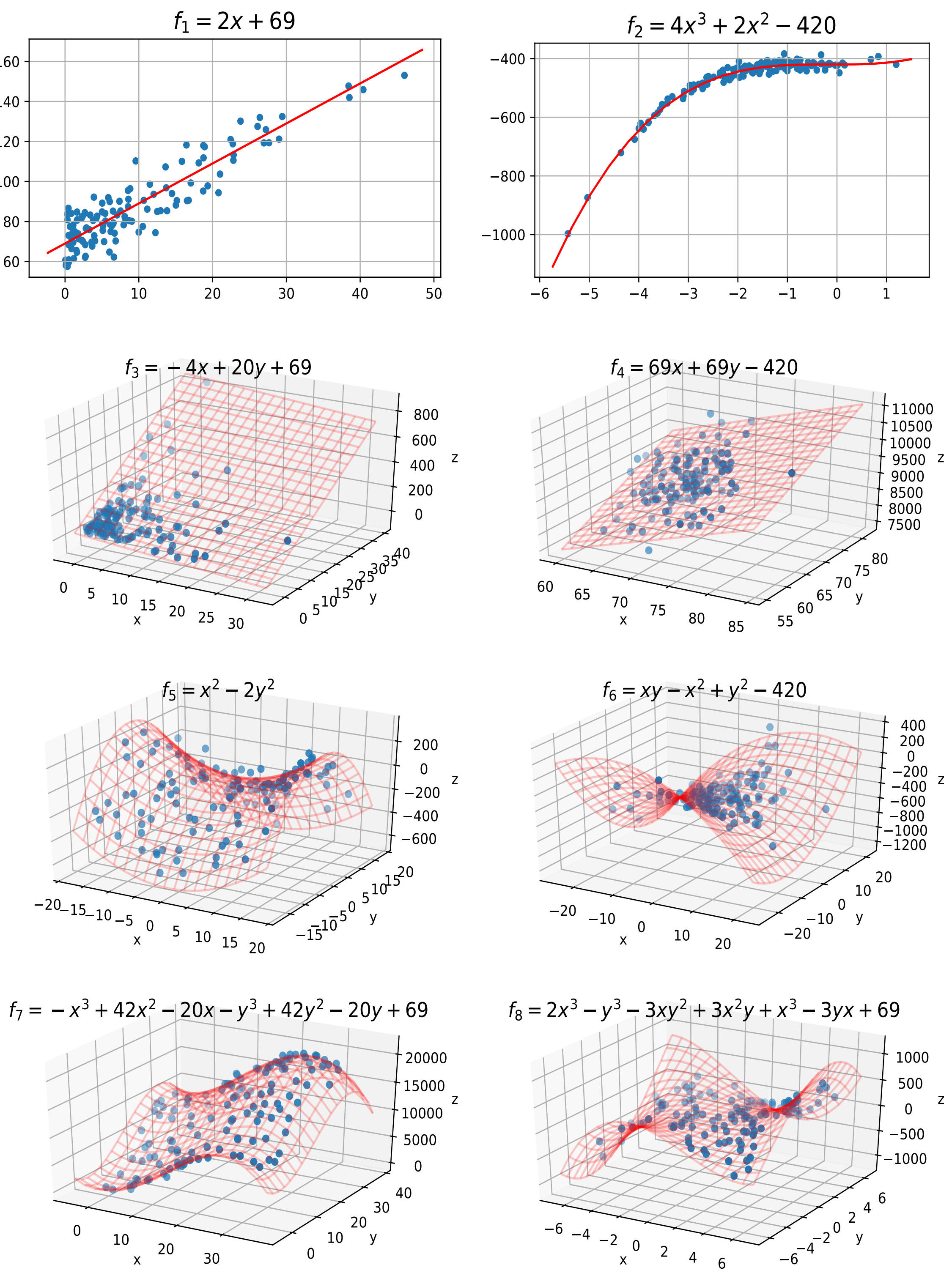

$f_1 = 2x + 69$
$f_2 = 4x^3 + 2x^2 - 420$
$f_3 = -4x + 20y + 69$
$f_4 = 69x + 69y - 420$
$f_5 = x^2 - 2y^2$
$f_6 = xy - x^2 + y^2 - 420$
$f_7 = -x^3 + 42x^2 - 20x - y^3 + 42y^2 - 20y + 69$
$f_8 = 2x^3 - y^3 - 3xy^2 + 3x^2y + x^3 - 3yx + 69$
x
y
z

Figure 3

## 2.2 Regression on the generated data

We fit regression models using ARD and without ARD to assess the differences in performance. We use the mean absolute error on the data as performance metric. To assess the models' performances under presence of irrelevant features, we simply a add few extra features to the data that does not affect the response variables. The distribution of the noise features are sampled from:

$$N\left(\text{Unif}\left(\text{min=-10, max=10}\right), \text{Unif}\left(\text{min=0.1, max=10}\right)\right)$$

The number of sampling iterations is chosen to be 2000 throughout this task. When training on the data, the models will be given the actual polynomial degree of the data. Any hyperparameters/user defined parameters are set to be 1 for the whole task for simplicity. The experiment should reveal how well the models manage to fit to training data given the actual polynomial degree. Formulated as a question; how well does the models manage to fit to data given the actual polynomial degree?

Table 2: "noise_features" column indicates number of added irrelevant features to the data. The degree column indicates which degree the polynomial regression was done in. Bolded elements represents the one with the lowest MAE.

| noise_features | function | degree | ARD_MAE | Regular_MAE | Regular_over_ARD |
|---|---|---|---|---|---|
| 0 | $f_1$ | 1 | **7.05** | 43.55 | 6.17 |
| | $f_2$ | 3 | **10.62** | 403.08 | 37.96 |
| | $f_3$ | 1 | **18.08** | 99.23 | 5.49 |
| | $f_4$ | 1 | **342.17** | 8899.03 | 26.01 |
| | $f_5$ | 2 | 55.64 | **55.43** | 1.00 |
| | $f_6$ | 2 | **51.73** | 295.41 | 5.71 |
| | $f_7$ | 3 | **332.81** | 4631.12 | 13.92 |
| | $f_8$ | 3 | **58.82** | 71.39 | 1.21 |
| 1 | $f_1$ | 1 | **7.06** | 40.56 | 5.75 |
| | $f_2$ | 3 | **10.32** | 266.15 | 25.78 |
| | $f_3$ | 1 | **18.15** | 96.93 | 5.34 |
| | $f_4$ | 1 | **347.15** | 8910.28 | 25.67 |
| | $f_5$ | 2 | **55.97** | 56.47 | 1.01 |
| | $f_6$ | 2 | **50.48** | 236.38 | 4.68 |
| | $f_7$ | 3 | **324.87** | 4499.19 | 13.85 |
| | $f_8$ | 3 | **56.19** | 71.78 | 1.28 |
| 2 | $f_1$ | 1 | **7.99** | 8.41 | 1.05 |
| | $f_2$ | 3 | **11.51** | 274.97 | 23.90 |
| | $f_3$ | 1 | **18.22** | 77.85 | 4.27 |
| | $f_4$ | 1 | **355.07** | 8919.88 | 25.12 |
| | $f_5$ | 2 | 63.10 | **62.49** | 0.99 |
| | $f_6$ | 2 | **49.37** | 219.12 | 4.44 |
| | $f_7$ | 3 | **395.03** | 3565.01 | 9.02 |
| | $f_8$ | 3 | **68.88** | 64.46 | 0.94 |
| 3 | $f_1$ | 1 | **7.69** | 8.40 | 1.09 |
| | $f_2$ | 3 | **12.72** | 120.63 | 9.48 |
| | $f_3$ | 1 | **18.25** | 99.53 | 5.45 |
| | $f_4$ | 1 | **352.42** | 8914.88 | 25.30 |
| | $f_5$ | 2 | 53.55 | **53.17** | 0.99 |
| | $f_6$ | 2 | **81.40** | 99.97 | 1.23 |
| | $f_7$ | 3 | **679.88** | 4405.54 | 6.48 |
| | $f_8$ | 3 | 75.26 | **58.70** | 0.78 |

Plots corresponding to the regression with no noise features and three noise features can be found in the appendix (Figure 8 and 9). After playing around with the functions and inspecting the results from the table, it seems to be that that regression without ARD struggles to estimate the intercept term, thus ruining its scores. Regression with ARD generally performed worse under the presence of noise features, but for some reason the version without ARD managed to get better with more noise features. Although it should be noted that we did not try to put a large number of noise features here, and it could simply be due to randomness. Generally, regression using ARD seems to be the best bet in general as it generally manages to get several times lower MAE values than without ARD.

Implementation with ARD seems to fit well to the data, but without ARD seems to struggle with intercept term.

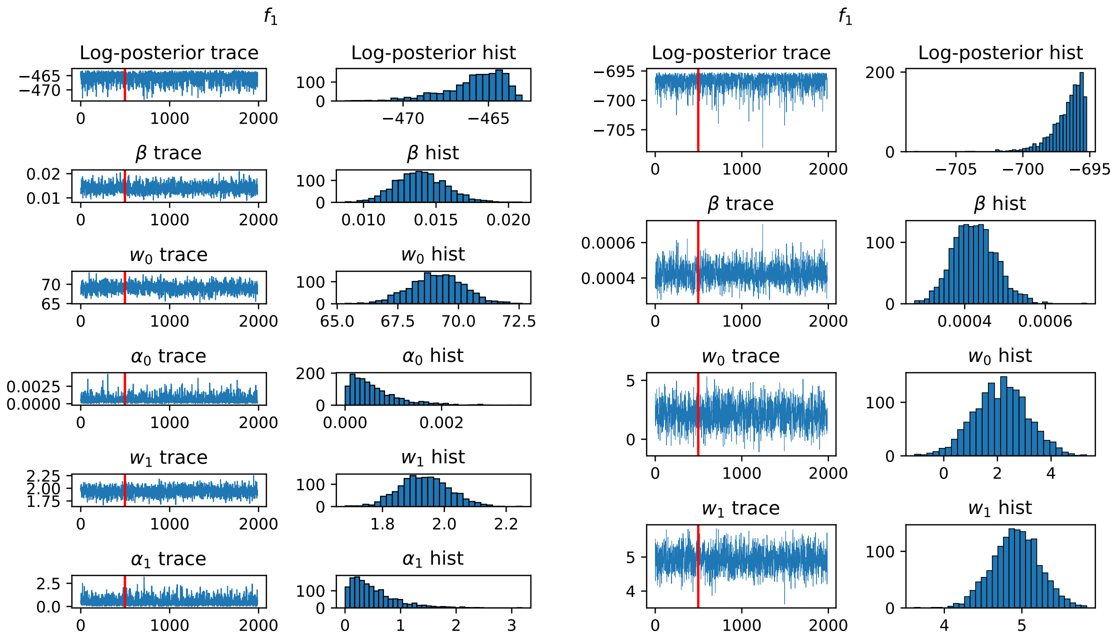


Figure 4: Left plot is ARD, right is without ARD. The true intercept for $f_1$ is 69. We can see that the intercept $(w_0)$ is mixing well for both, but the right plot (without ARD) does not converge to the correct value at all.

## 3 Testing on real data

A real dataset is given to test the regression algorithms on. The dataset consists of 100 features. The selected polynomial degree to use is 1, as any higher degree will require infeasible computational times on my laptop. In order to assess which features are the most important we find the term with the highest polynomial absolute coefficient (ignoring the intercept term as it does not depend of any of the features). We assess this for both the model with and without ARD.

Table 3: Feature importance in descending order of top five features.

| | Feature | coefficient |
|---|---|---|
| with ARD | delta | -2.60 |
| | hsa.miR.29c. | 1.30 |
| | hsa.miR.933 | -1.19 |
| | hsa.miR.328 | 1.09 |
| | hsa.miR.24 | 0.88 |
| without ARD | delta | -2.60 |
| | hsa.miR.29c. | 1.30 |
| | hsa.miR.328 | 1.09 |
| | hsa.miR.770.5p | 0.44 |
| | hsa.miR.933 | -1.19 |

Both algorithms are agreeing on the two first features. Even the coefficient values are the same for the top two features. The features are delta and hsa.miR.29c. I would personally expect Age to play a bigger role, but it didn't apparently play that big of a role as we can see in the trace plots (Age coefficient is $w_2$). Visualizing the first 15 weights and variances shows that at least the first 15 the parameters were mixing fine during the sampling process. As a reminder, the red vertical lines shows the cutoff value which is set to be at the first quarter of all the samples. Only the samples after the cut are used for determining the finalized values. It should be reasonable to assume that the rest of the parameters also mix well.

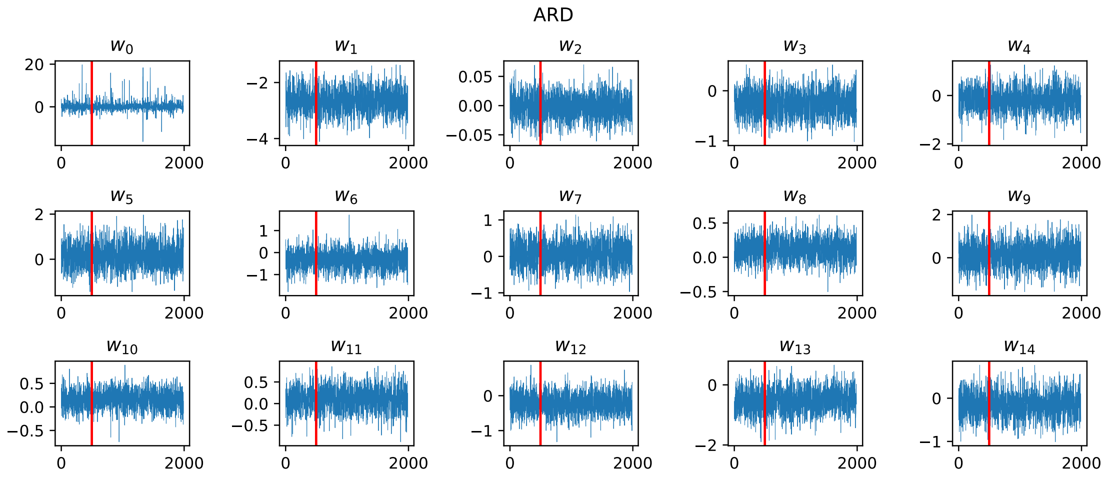


Figure 5: Trace plots of first 15 regression weights from ARD model.

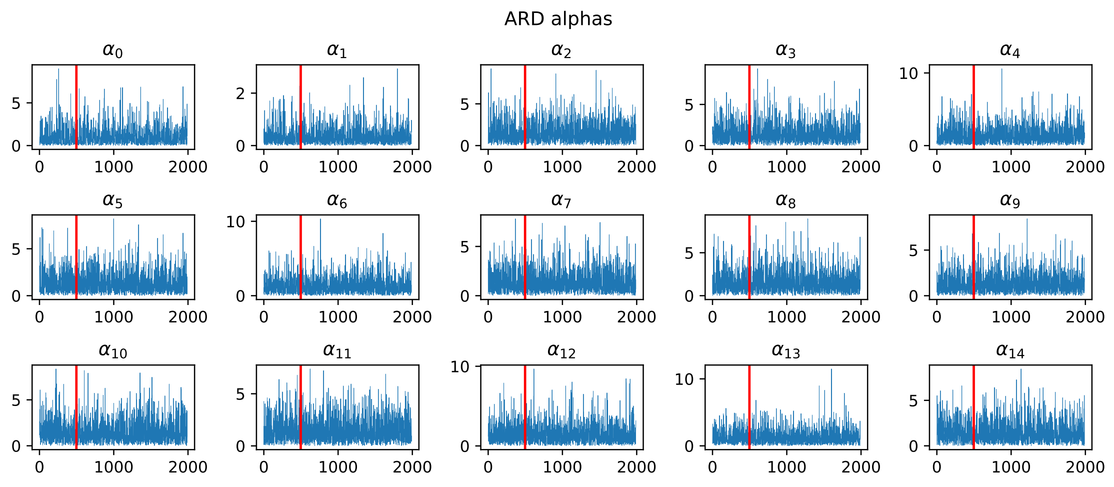


Figure 6: Trace plots of first 15 alpha values from ARD model.

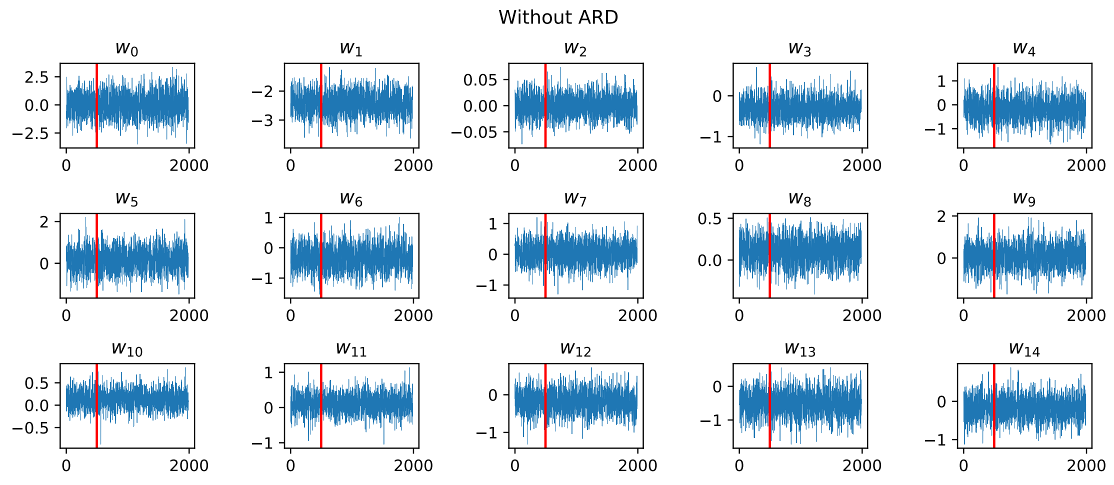


Figure 7: Trace plots of first 15 regression weights from model without ARD.

The mean absolute errors for ARD and without were 33.71 and 27.31 respectively. The ARD model got beaten actually. Whether the difference is large or not dependents on the real noise distribution of the data.

## 4 Discussion

### 4.1 Alternative to MAE on data as scoring metric

Using mean absolute error on the data as a scoring metric emphasizes the models' ability to fit to training data. An alternative metric would be to check how close the estimated model parameters were to the real parameters, i.e use MAE with respect to model parameters instead.

### 4.2 Approach to check for overfitting tendencies

The simulation part checked how the models manage to fit to the training data. Generally, polynomials model will tend to overfit if one assumes a polynomial degree higher than the actual function that the data was generated from. To check the tendency overfitting during such cases, we could try to fit high degree polynomial models with and without ARD, and then check the parameter loss between the estimated and actual polynomials.

### 4.3 Reason for struggling with intercept term

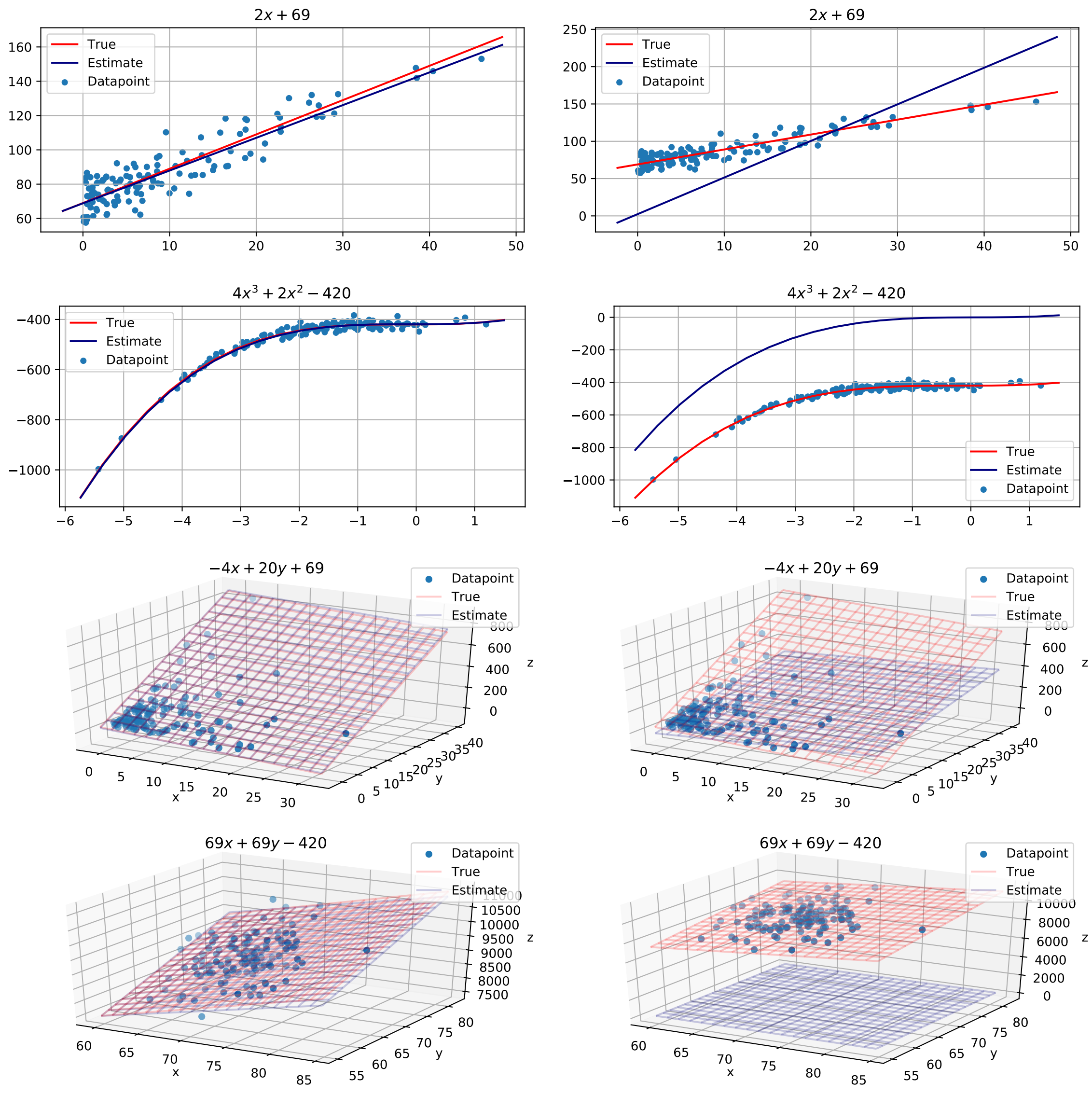
The implementation without ARD really struggled with estimating the intercept term. It should be noted that the intercept terms utilized in the simulation task were quite a bit larger than the other coefficients, and that is maybe why it struggled so much, but it still kind of odd that it would struggle this much. Further examination is needed.

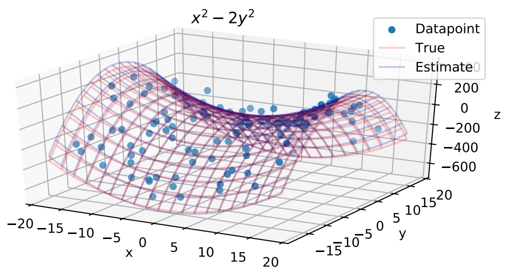
## 5 Conclusion

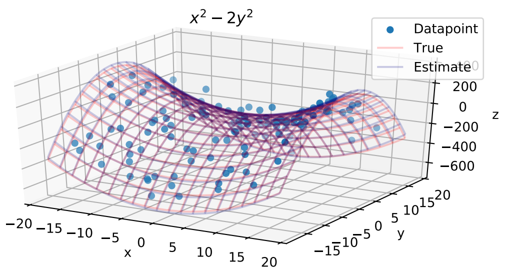
Fitting polynomial regression models using Gibbs sampling with ARD generally performs better than without ARD when specifying the actual polynomial degree of the polynomials the data was generated from. The method using Gibbs sampling to fit with polynomial kernels works fine for low dimensional data, but becomes very computationally heavy under the presence of many dimensions as the polynomial kernel increases in size polynomially. Linear regression without ARD seems to significantly struggle to estimate the intercept term. Both the algorithms works for the lung cancer data (when assuming polynomial of order one). The trace plots shows good mixing, and the it seems to be that the most important feature to predict the y-value is the delta feature.

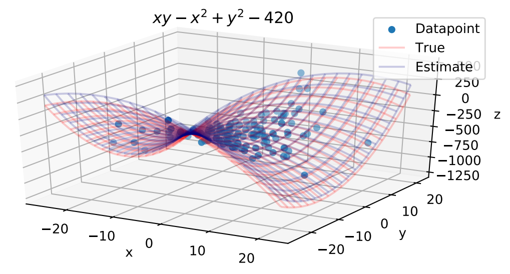
## A Estimates without noise features

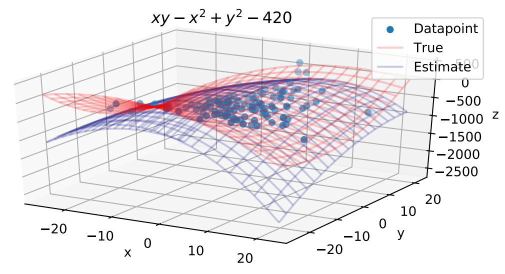
Figure 8: Left column is ARD, right column is without ARD

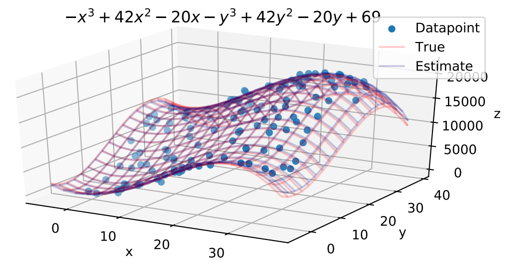


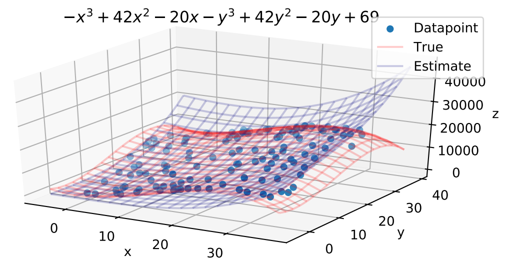

$x^2 - 2y^2$
Datapoint
True
Estimate
x
y
z

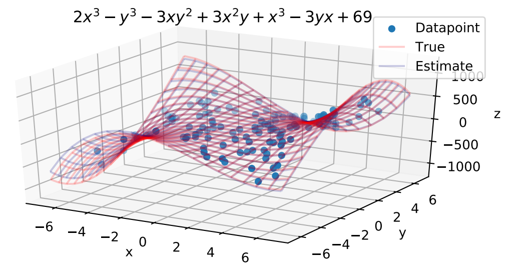

$x^2 - 2y^2$
Datapoint
True
Estimate
x
y
z

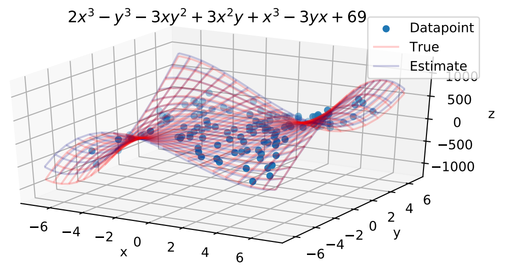

$xy - x^2 + y^2 - 420$
Datapoint
True
Estimate
x
y
z


$xy - x^2 + y^2 - 420$
Datapoint
True
Estimate
x
y
z


$-x^3 + 42x^2 - 20x - y^3 + 42y^2 - 20y + 69$
Datapoint
True
Estimate
x
y
z


$-x^3 + 42x^2 - 20x - y^3 + 42y^2 - 20y + 69$
Datapoint
True
Estimate
x
y
z


$2x^3 - y^3 - 3xy^2 + 3x^2y + x^3 - 3yx + 69$
Datapoint
True
Estimate
x
y
z


$2x^3 - y^3 - 3xy^2 + 3x^2y + x^3 - 3yx + 69$
Datapoint
True
Estimate
x
y
z

## B Estimates with three noise features

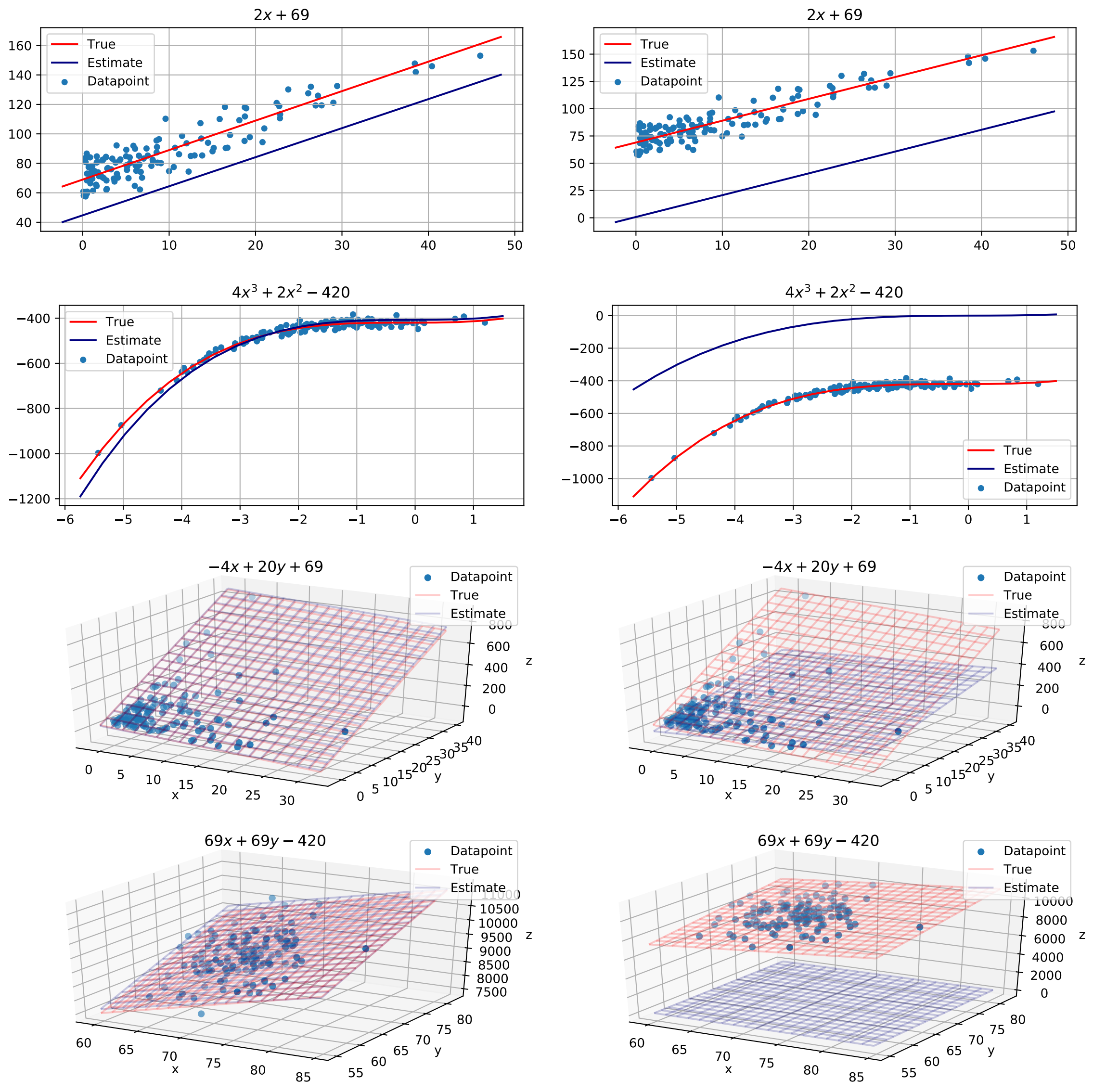
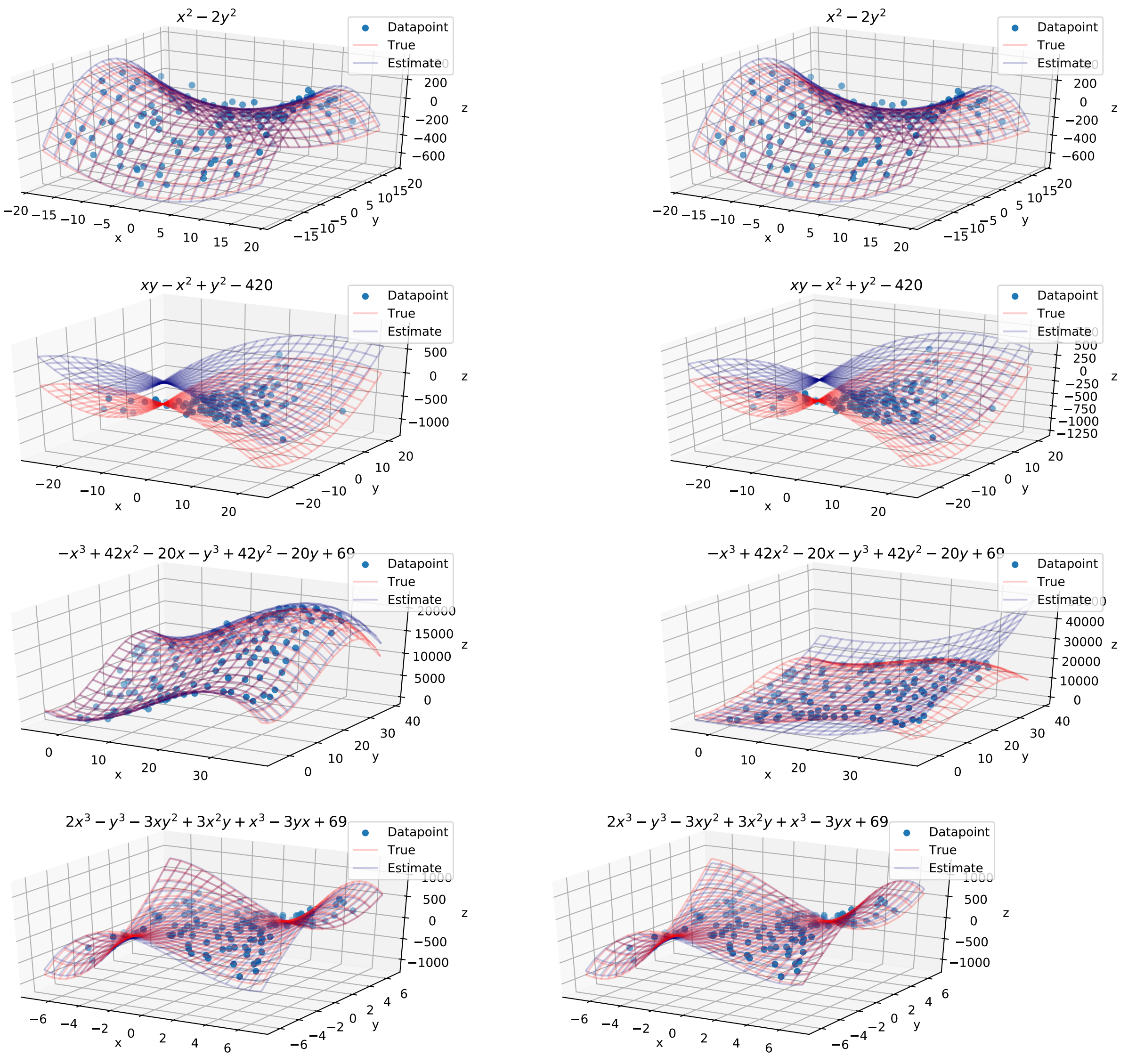


Figure 9: Left column is ARD, right column is without ARD



## References

[1] Scikit-learn, "Polynomialfeatures sklearn details," 2020, [accessed 31-March-2020]. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html