Interpretable Machine Learning on Synthetic Data

And Little Known Secrets About Linear Regression

Vincent Granville, Ph.D.
vincentg@MLTechniques.com
www.MLTechniques.com
Version 1.0, May 2022

Abstract

The technique discussed here handles a large class of problems. In this article, I focus on a simple one: linear regression. I solve it with an iterative algorithm (fixed point) that shares some resemblance to gradient boosting, using machine learning methods and explainable AI, as opposed to traditional statistics. In particular, the algorithm does not use matrix inversion. It is easy to implement in Excel (I provide my spreadsheet) or to automate as a black-box system. Also, it is numerically stable, can generalize to non-linear problems. Unlike the traditional statistical solution leading to meaningless regression coefficients, here the output coefficients are easier to understand, leading to better interpretation. I tested it on a rich collection of synthetic data sets: it performs just as well as the standard technique, even after adding noise to the data. I then show how to measure the impact of individual features, or groups of features (and feature interaction), on the solution. A model with m features has 2^m sub-models. I show how to draw more insights by analyzing the performance of each sub-model. Finally, I introduce a new metric called *score* to measure model performance. Based on comparison with the base model, it is more meaningful than R-squared or mean squared error.

Contents

1 Introduction

Here, X denotes the input. It is represented as a matrix with n rows and m columns; n is the number of observations, and m the number of features, also called dependent variables. The response (also called independent variable or output) is a column vector with n entries, and denoted as Y. The m regression coefficients (unknown, to be estimated) are stored in a column vector denoted as β. Thus we have

$$Y = X\beta + \epsilon, \tag{1}$$

where ϵ – also a column vector with n entries – is the error. The problem consists of finding a suitable β that in some way, minimizes the error. If there was no error term, equation (1) could be rewritten as $X^TY = X^TX\beta + \Lambda\beta - \Lambda\beta$, that is,

$$\beta = \Lambda^{-1}X^TY + (I - \Lambda^{-1}X^TX)\beta.$$

Here Λ is any non-singular (invertible) $m \times m$ matrix, and ^{T} denotes the matrix or vector transposition operator [Wiki]. This gives rise to the following iterative algorithm:

$$\beta_{k+1} = \Lambda^{-1}X^TY + (I - \Lambda^{-1}X^TX)\beta_k, \tag{2}$$

starting with some initial configuration β_0 for the parameter vector (the regression coefficients). I use a diagonal matrix for Λ, so the methodology does not involve complicated matrix inversions.

I also use the following notations: $M = X^TX$ and $S = I - \Lambda^{-1}M$. The convergence of this iterative algorithm, and how fast it converges, is entirely governed by how fast $S^k \to 0$ as $k \to \infty$. It requires a careful choice of Λ. I discuss later how to update Λ at each iteration k: in an adaptive version of this algorithm, Λ is replaced by Λ_k in (2), with the hope that it boosts convergence. The general term for this type of iteration, which also encompasses Newton optimization and gradient descent, is fixed point algorithm [Wiki].

The remaining of this discussion focuses on the choice of Λ and β_0, with convergence implications and computational complexity, tested on synthetic data [Wiki]. I show that with very few iterations, one generally gets a very good predictor, even though the estimated parameter vector is quite different from the target one used in the simulations. In short, the R-squared arising from rough approximations based on few iterations of the fixed point algorithm, is very similar to that obtained using the full standard statistical apparatus. Despite the non-statistical perspective and the absence of statistical model, I explain how to compute confidence intervals for the estimated regression coefficients and for the predicted values. The whole framework is designed to facilitate interpretation, and thus it falls in the category of explainable AI [Wiki].

Finally, I want to offer a simple version of this method, simple enough to easily be implemented in Excel. The choice $\beta_0 = 0$ and Λ minimizing the Frobenius norm of S [Wiki] (see also [2]), not only works well but it leads to simple formulas, and an interesting connection to eigenvalues [Wiki] (see also here). The last part of this article focuses on assessing the influence of each feature, and the impact of feature interaction. The synthetic training set data discussed in section 2 allows you to simulate and test a large number of varied situations. Eventually, model performance is measured on a validation set, not on a training set.

2 Synthetic data sets and the spreadsheet

Rather than testing the methodology on a few real-life data sets, I tested it on a large number of very different synthetic data sets, each with its unique correlation structure. These data sets are generated via simulations, as follows. First, generate m column vectors $Z_1, \ldots, Z_m$, with Z_i consisting of n deviates Z_{ij} $(1 \leq i \leq m, 1 \leq j \leq n)$. One of the simplest distributions to sample from is the generalized logistic. See section 2.1.1 in my book on stochastic simulations [5]. In this case I used

$$Z_{ij} = -\log \frac{U_{ij}^\gamma}{1 - U_{ij}^\gamma},$$

where $\gamma > 0$ is a parameter, and U_{ij}'s are independently and identically distributed uniform deviates on $[0, 1]$. Then, I generated m column vectors $X_1, \ldots, X_m$ as random linear combinations of the Z_i's:

$$X_i = \sum_{j=1}^{m} w_{ij} Z_i, \quad i = 1, \ldots, m. \tag{3}$$

The $m \times m$ matrix $W = (w_{ij})$ is called the weight matrix. Here again, the w_{ij} are deviates from the same family of generalized logistic distributions. Finally, the simulated response is

$$Y = \sum_{i=1}^{m} \alpha_i X_i + \tau\epsilon, \quad i = 1, \ldots, m,$$

where ϵ (a column vector with n independent entries) is an artificially generated white noise, and $\tau \geq 0$ controls the amount of noise. The α_i's can be pre-specified or randomly generated. In any case, the exact, pre-specified set of regression coefficients is the column vector $\alpha = (\alpha_1, \ldots, \alpha_m)^T$. The estimated coefficients, at the k-th iteration of the fixed point algorithm, using formula 2, is the column vector $\beta_k = (\beta_{k1}, \ldots, \beta_{km})^T$. Thus in this setting, we are able to measure how close the estimate β_k is to the exact value α. In real-life applications, the exact value is never known. If it was, there would be no need to perform statistical inference.

2.1 Correlation structure

Let Ω_X (respectively Ω_Z) be the $m \times m$ covariance matrix [Wiki] attached to $X_1, \ldots, X_m$ (respectively to $Z_1, \ldots, Z_m$). Likewise, define the correlation matrices as R_X, R_Z, with

$$R_X = [D(\Omega_X)]^{-1/2}\Omega_X[D(\Omega_X)]^{-1/2} \quad \text{and } R_Z = [D(\Omega_Z)]^{-1/2}\Omega_Z[D(\Omega_Z)]^{-1/2}.$$

Here $D[A]$ is the matrix consisting of the diagonal elements of the matrix A. We have

$$\Omega_X = W\Omega_Z W^T = (W\Omega_X^{1/2})(W\Omega_X^{1/2})^T, \quad \text{thus } W = \Omega_X^{1/2}\Omega_Z^{-1/2}.$$

These formulas allow you to easily compute R_X based on the weight matrix W and Ω_W. Though more difficult, it is possible to solve the inverse problem: pre-specify the correlation structure R_X of the data set, and then find W that yields the desired, target R_X. This is best accomplished using an iterative algorithm similar to the fixed point discussed in section 1, using the above formulas.

The formulas to solve the inverse problem involve the square root of positive semidefinite matrices [Wiki]. The solution is not unique. See how it is done, in my article "gentle introduction to linear algebra" [4]. Without loss of generality, a simplification consists of simulating standardized $Z_1, \ldots, Z_m$ (from a distribution with zero mean and unit variance) so that $R_Z = \Omega_Z$ is the identity matrix. If in addition, the observed $X_1, \ldots, X_m$ are also standardized, then $R_X = \Omega_X$, and thus, $W = R_X^{1/2}$. Multiple square roots exist, in the same way that 2 and -2 are two "square roots" of 4.

2.2 Standardized regression

Under stable conditions, the predicted values for Y are very close to those obtained via standard statistical regression, even though the estimated regression coefficients may be quite different. The accompanying spreadsheet and computations are now stable. However, in previous tests, with a different damping schedule (the matrix Λ), sometimes the β_k diverged as $k \to \infty$. Yet after normalizing β_k, the instability was essentially removed and again, the predicted Y was sound. I provide here the normalizing formula, to guarantee that the standard deviation of the response Y, denoted as σ_Y, is identical to that measured on the predicted Y. The new β_k, denoted as β_k^*, is computed as follows:

$$\beta_k^* = \frac{\sigma_Y}{\sqrt{\beta_k^T\Omega_X\beta_k}} \cdot \Omega_W\beta_k.$$

This may be useful if you modify Λ when doing some research, as a technique to stabilize the predictions. I also included the computation of β_k^* in the spreadsheet. However, it is best to avoid standardizing the regression coefficients when the algorithm is numerically stable. It results in more realistic variance in the predicted values as the non-corrected regression acts as a smoother, but it also comes with a price: a larger mean squared error.

To the contrary, shifting the predicted values so that their mean matches that of the observed values on the training set, is always useful. It is included in my computations (and in the spreadsheet) as a final, post-processing step. It does not impact the R-squared. Also, it allows you to ignore the intercept parameter in the regression model. Indeed, this is an easy workaround to using an actual intercept parameter.

2.3 Initial conditions

The neutral choice $\beta_0 = 0$ as the starting vector of regression coefficients, for the iterative fixed point algorithm, works well. Another option consists of choosing regression coefficients that preserve the correlation sign between the response Y, and each feature $X_1, \ldots, X_m$. Here, X_i is the i-th column of the matrix X. Let

$$c_i = \frac{\text{Cov}[Y, X_i]}{\text{Var}[X_i]}, \quad c = (c_1, \ldots, c_m)^T, \quad Q = \sum_{i=1}^{m} c_i X_i = Xc,$$

with ω a real number chosen to minimize the error $\epsilon^T\epsilon = (Y - \omega Q)^T(Y - \omega Q) = Y^TY - 2Q^TY\omega + Q^TQ\omega^2$. We have

$$w = \frac{Y^TQ}{Q^TQ} = \frac{Y^TXc}{c^TX^TXc} \quad \text{and } \text{Var}[c_iX_i] = \text{Var}[Y] \cdot \rho^2[X_i, Y] = \sigma_Y^2 \cdot \rho^2[X_i, Y],$$

where ρ denotes the correlation function. Now, $\beta_0 = \omega c$ is a starting point that makes sense, easy to interpret, and better then $\beta_0 = 0$. It significantly reduces the residual error, over the base model $\beta = 0$. In many cases, it yields a residual error almost comparable to that of the best predictors.

As an illustration, let's say that $X_2 = X_3$. You should avoid highly correlated features in your data set, but in some cases the inter-dependencies among several features are strong but much harder to detect, and results

in the same problem. In my example, assuming both X_2 and X_3 are positively correlated to the response Y, a model with $+4$ and -2 for the regression coefficients attached to X_2 and X_3, performs just as well as $-2, +4$ or $+1, +1$. The β_0 proposed here addresses this issue: it guarantees that the regression coefficients attached to X_2 and X_3 are both positive in this example, and identical if $X_2 = X_3$. It makes the regression coefficients much easier to interpret. In addition, this technique is numerically stable and more robust.

2.4 Simulations and Excel spreadsheet

Formulas and computations described in section 2 are implemented in my spreadsheet `Regression5.xlsx`, available here on my GitHub repository. This material covers a large chunk of the spreadsheet, with the remaining explained in the next sections.

The `Test` and `Results` tabs in the spreadsheet contain the following:

- The random deviates $Z_1, \ldots, Z_m$ are in the `Test` tab in columns `A:F`. The the zero-mean noise is in column `H`. The amount of noise in the response Y is controlled by the parameter `Noise` in cell `E:5` in the `Results` tab. As a general rule, cells highlighted in light yellow in the `Results` tab correspond to parameters or hyper-parameters that you can modify. The parameter γ just above in cell `E:4` is the core parameter of the generalized logistic distribution used to simulate the column vectors $Z_1, \ldots, Z_m$.
- The flag in cell `E:7` in the `Results` tab allows you to choose either $\beta_0 = 0$, or the special β_0 discussed in section 2.3.
- By default, the regression uses the traditional β_k. If you want to use the normalized β_k^* instead, they are iteratively computed in cells `AX2:ES7` in the `Test` tab. For instance, cells `BL2:BL7` represent β_{15}^*, that is, the 15th iterate ($k = 15$) in the fixed point algorithm, stored as a column vector. The standard β_k are computed in cells `AX20:ES25` in the same tab.
- Column `I` in the `Test` tab is the response Y. The features $X_1, \ldots, X_m$, also called independent variables, are in columns `J:O` in the same tab. They are generated as linear combinations of the random vectors $Z_1, \ldots, Z_m$. More precisely, $X = WZ$ as per equation (3), where W is the $m \times m$ weight matrix. The random weight matrix W is stored in cells `B15:G20` in the `Results` tab. The actual, true (random) regression parameters are just above in the same tab, in cells `B12:G12`.
- Intermediary computations are in the `Test` tab. For instance, the $m \times m$ matrix $X^T X$ is stored in cells `AP2:AU7`, the vector $X^T Y$ in cells `AN2:AN7`, the correlation matrix R_X in cells `AE3:AJ8`, and the covariance matrix Ω_X in cells `AE12:AJ17`.

The interactive spreadsheet extensively uses the `SumProduct` and `Transpose` Excel functions, to easily multiply a row vector by a column vector with just one simple operation. It also makes matrix multiplications easier.

3 Damping schedule and convergence acceleration

The $m \times m$ diagonal matrix Λ^{-1} in equation (2) is called the damping parameter, or preconditioner [Wiki] of the fixed point iteration. It governs the rate of convergence. The fixed point algorithm converges if $|\varphi| < 1$, where $\varphi = \varphi(S)$ is the largest eigenvalue of $S = I - \Lambda^{-1} X^T X$, in absolute value. The smaller $|\varphi|$, the faster the convergence. The convergence speed is eventually determined by how fast $S^k \to 0$ as $k \to \infty$, itself being a function of $|\varphi|$. Thus, it makes sense to choose Λ so that S is close to zero. One way to do it is to choose Λ that minimizes the Frobenius norm of S, or in other words, Λ that minimizes the sum of the square of the coefficients of S. This leads to

$$\lambda_i^{-1} = \frac{1}{m_{ii}} \sum_{j=1}^{m} m_{ij}^2, \quad i = 1, \ldots, m \tag{4}$$

where $\lambda_1, \ldots, \lambda_m$ are the diagonal elements of the diagonal matrix Λ, and m_{ij} is the j-th element in column i, of the matrix $M = X^T X$. In practice, with this choice of Λ, assuming M is not singular, the fixed point iteration always converges, and $m_{ii} > 0$. See discussion on this topic, here.

3.1 Spreadsheet implementation

The λ_i's are computed in cells `AL2:AL7` in the `Test` tab. Also, $|\varphi|$ is iteratively computed in cells AY59:ES59 in the same tab. The method used in the spreadsheet to compute $|\varphi|$ is known as power iteration [Wiki]. See also here.

Now we have everything in place to compute the regression coefficients. They are found in cells `K5:P5` (computation based on $k = 15$ iterations) and `K6:P6` (based on $k = 100$ iterations) in the `Results` tab. The final value of $|\varphi|$ is in cell `K21`, also in the `Results` tab. Note that values of $|\varphi|$ above 0.95 means that the

system is ill-conditioned [Wiki]. For instance, some features are almost linear combinations of other features. It typically results in poor performance. In this case, using few iterations ($k = 15$) together with the initial β_0 suggested in section 2.3, works best. It also avoids overfitting issues.

The predicted values of Y, obtained after $k = 15$ and $k = 100$ iterations, are in columns `P` and `Q` respectively, in the `Test` tab. The predicted values obtained with β_0 alone (as computed in section 2.3) are in column `S`, while those obtained with traditional regression are in column `U` and produced with the `Linest` Excel function. The filtered response, obtained by removing the artificial noise introduced in the observed (synthetic) data, is in column `R`. The computational complexity of the fixed point regression is the same as multiplying two $m \times m$ matrices, multiplied by the number of iterations in the fixed point algorithm.

3.2 Interpretable regression with no overfitting

The regression coefficient vector β_0 introduced in section 2.3 is intuitive, robust and preserves the correlation signs: by design, a feature positively correlated with the response Y gets assigned a positive correlation coefficient, as previously discussed. In addition, its performance is nearly as good as optimum regression coefficients, in many situations. It definitely performs well above the base model, unless the amount of noise is substantial. But in that case, all models perform badly, unable to improve over the base model. See section 4.1 for performance comparison. If your problem is ill-conditioned (for instance some features are nearly identical or linear combinations of other features), traditional techniques will lead to meaningless regression coefficients, or fail entirely. To the contrary, the β_0 in question handles this situation very well.

All of this makes the β_0 in question a good starting point for the fixed point iteration, if the goal is to obtain a robust solution, easy to interpret, and not prone to overfitting. It can be improved by using a few iterations of the fixed point algorithm. The more iterations, the closer you get to the standard "exact" solution, but you eventually lose interpretability. With fewer iterations, you may still get a good performance, yet avoid many of the aforementioned problems. One way to decide when to stop is to run many iterations, and compare your R-squared obtained after (say) 10 versus 100 iterations. If the difference is negligible, use the regression coefficients obtained at iteration $k = 10$. Instead of using R-squared, I suggest to use the metric s defined in section 4.

3.3 Adaptive damping

A possible way to boost convergence is to use a matrix Λ that depends on k, and denoted as Λ_k. The most simple example is when $\Lambda_k = \lambda(k) \cdot I$ where $\lambda(k)$ is a real number and I is the $m \times m$ identity matrix. Then, choose $\lambda(k)$ that minimizes $||(I - \lambda^{-1})\beta_k||_2$, that is

$$\begin{aligned}
\lambda(k) &= \arg\min_{\lambda} ||(I - \lambda^{-1})\beta_k||_2 \\
&= \arg\min_{\lambda} [(I - \lambda^{-1}M)\beta_k]^T (I - \lambda^{-1}M)\beta_k \\
&= \arg\min_{\lambda} \beta_k^T\beta_k - 2\lambda^{-1}\beta_k^T M\beta_k + \lambda^{-2}(M\beta_k)^T M\beta_k \\
&= \frac{(M\beta_k)^T M\beta_k}{\beta_k^T M\beta_k}.
\end{aligned}$$

Again, $M = X^T X$. Preliminary tests did not show large improvements over using a fixed Λ. Indeed, about 20% of the time, instead of converging, the successive iterates of the regression coefficients oscillate. Yet this issue is easy to address, and adaptive damping – where Λ depends on k – has potential to handle data sets with a larger m (the number of features). The worst case by far in my 100 tests or so, is pictured in Figure 1. The drawback, compared to using preconditioning only (a fixed Λ) is that it requires more computations. For a recent machine learning application of preconditioners in a similar context, see [1].

4 Performance assessment on synthetic data

I use three metrics to measure the goodness of fit, between the observed (synthetic) response Y and the predicted response, denoted here as P. These metrics are computed on a validation set, not on the training set: this is a standard cross-validation procedure to avoid overfitting and performance inflation. Thus, what I call R-squared is actually closer, but not identical, to "predictive R-squared" and PRESS statistics [Wiki]. The same applies to MSE. The synthetic control set is in the `Control` tab in the spreadsheet.

The three metrics are:

- The R-squared r, with $0 \leq r \leq 1$, is the square of the correlation between Y and P. It is also equal to $[\sigma_Y^2 - (Y - P)^T(Y - P)]/\sigma_Y^2$.

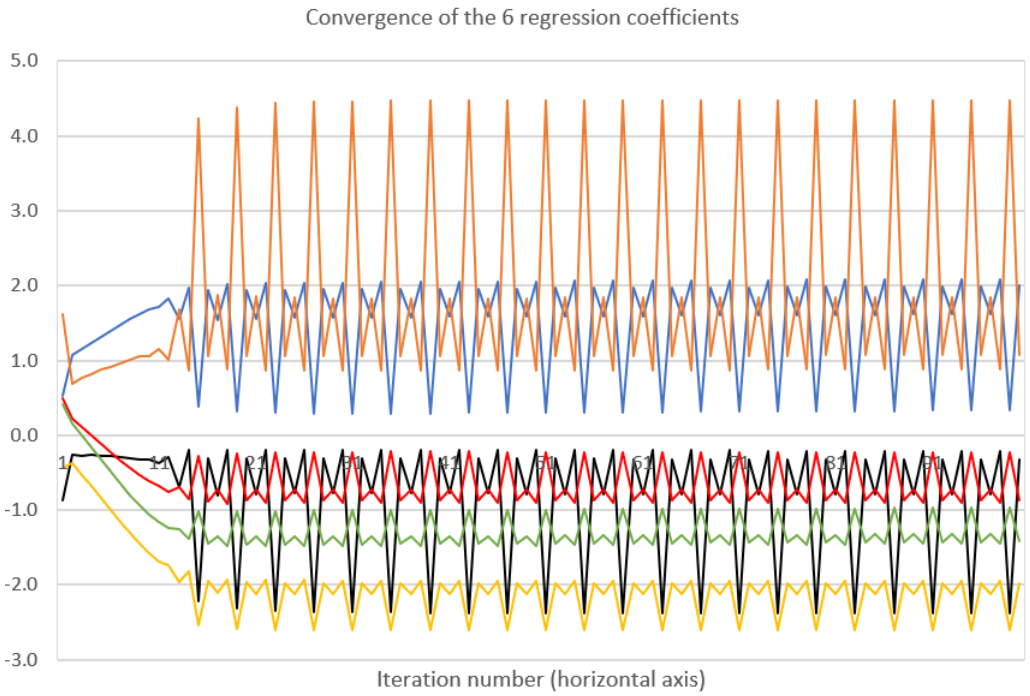

Figure 1: Regression coefficients oscillating when using adaptive damping

- The root mean squared error or RMSE [Wiki], defined as $e = \sqrt{(Y-P)^T(Y-P)}/n$, where n is the number of observations. It is a much better measure of the actual error, compared to the mean squared error (MSE). In particular, if Y is measured in (say) miles, then RMSE is measured in miles, while MSE is measured in square miles. Likewise, if Y is measured in years, RMSE is measured in years while MSE is measured in square years – a meaningless metric.
- The score s, with $0 \leq s \leq 1$, defined as $s = 1 - e(P)/e(P_0)$. Here, P_0 is the base model, corresponding to $\beta = 0$ (thus, the predicted value is constant after adjustment, equal to the mean value of Y computed on the training set). In particular, s measures the improvement over the base model, using RMSE ratios, while r does the same using MSE ratios instead. In my opinion, s is more intuitive and more realistic than r. It is related to r via the formula $r = 2s - s^2$, or equivalently, $s = 1 - \sqrt{1-r}$. We always have $r \geq s$, so r is an inflated measure of goodness of fit.

Since the model is trained on the test data set, but model performance is measured on the validation set, the above performance metrics are hybrid. In particular, as a result, the relationship $r = 2s - s^2$ is almost satisfied, but not exactly. Also, on rare occasions (with very noisy data), r can be slightly negative. The performance results are shown in the `Results` tab, in cells `J13:R19`. I also included the performance metrics for the simple model consisting in using the regression coefficient vector $\beta = \beta_0$ defined in section 2.3. In the spreadsheet, this model is referred to as "Predicted P_b".

In addition, the spreadsheet allows you to choose which features to include or not in the model, in the fixed point iteration. The feature selection flags are stored in cells `B4:B9` in the `Results` tab. The default value is 1, corresponding to inclusion. This is particularly useful to test how much model improvement is gained by using all features, versus a subset of them. On real data, some features are usually missing because the model developer was not aware of them, and they are not captured in the data set: in other words, they are unobserved. The feature selection flags allow you to compare the performance on observed data, versus the performance obtained with a dream data set that would include all the relevant features.

Finally, the use of synthetic data offers a big benefit: you can test and benchmark algorithms on millions of very different data sets, all at once. This assume that the synthetic data is properly generated. It is both an art and a science to design such data sets. It typically involves a decent amount of mathematics, sometimes quite advanced. It is my hope that this article will help the reader build good quality, rich synthetic data that covers a large spectrum of potential real life situations. Other synthetic data sets are found in my book on stochastic simulation [5], featuring synthetic clusters, and in my article "Computer Vision: Shape Classification via Explainable AI" [3], featuring synthetic shapes.

4.1 Results

I tested the methodology on a synthetic dataset with $n = 1000$ observations and $m = 6$ features. The validation set also has 1000 observations. Because the data is artificially generated, the exact regression coefficients are known. They are listed in the row labeled "Exact" in table 1. The observed data Y is a mixture of the exact data and artificial noise. The amount of noise is controlled by the parameter `Noise` in the spreadsheet. The exact data is the vector $X\beta$, with β being the true, pre-specified regression coefficients (also artificially generated). Note that X is also artificially generated, using the method described in section 2.4.

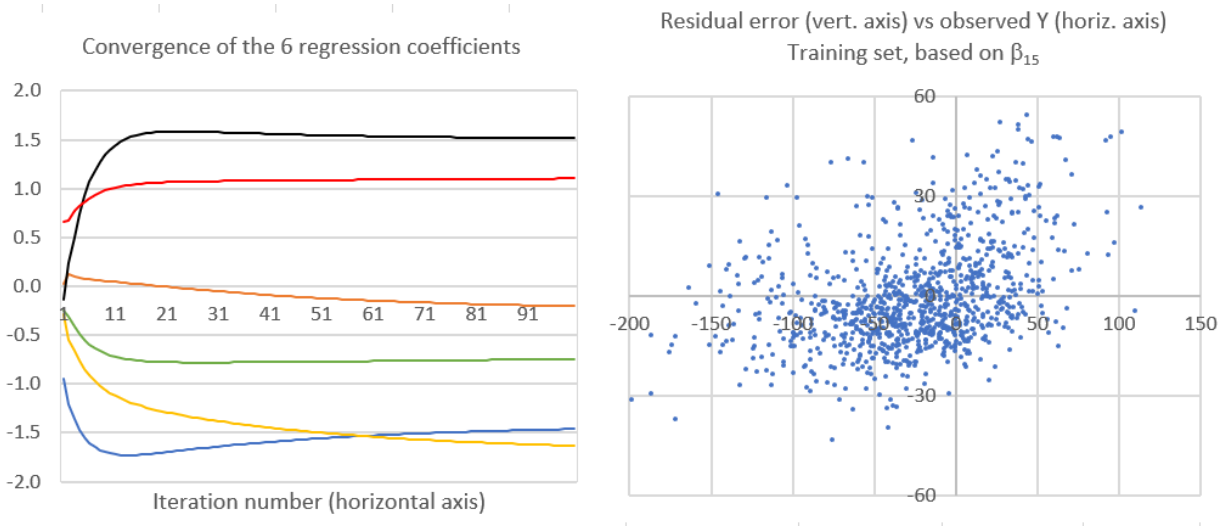

Figure 2: Convergence of regression coefficients (left) and distribution of residual error (right)

The metric r in the `Exact` row in table 1 measures the R-squared between the observed data Y, and the exact data. In practice, the exact data is not known. But one of the benefits of using synthetic data, is that we can simulate both the unobserved, exact data, and the observed, noisy, unfiltered version, denoted as Y.

The metric r in table 1 is the R-squared, but measured on the validation set, not on the training set. By contrast, in figure 3, the R-squared on the right plot, attached to the blue dots and blue regression line, is computed on the training set. On the left plot, it is computed on the validation set, and it matches the r value attached to β_{15} in table 1. The blue dots in figure 3 is the scatterplot of Y versus the predicted values obtained with β_{15}. The orange dots is the scatterplot of Y versus the exact (unobserved) data.

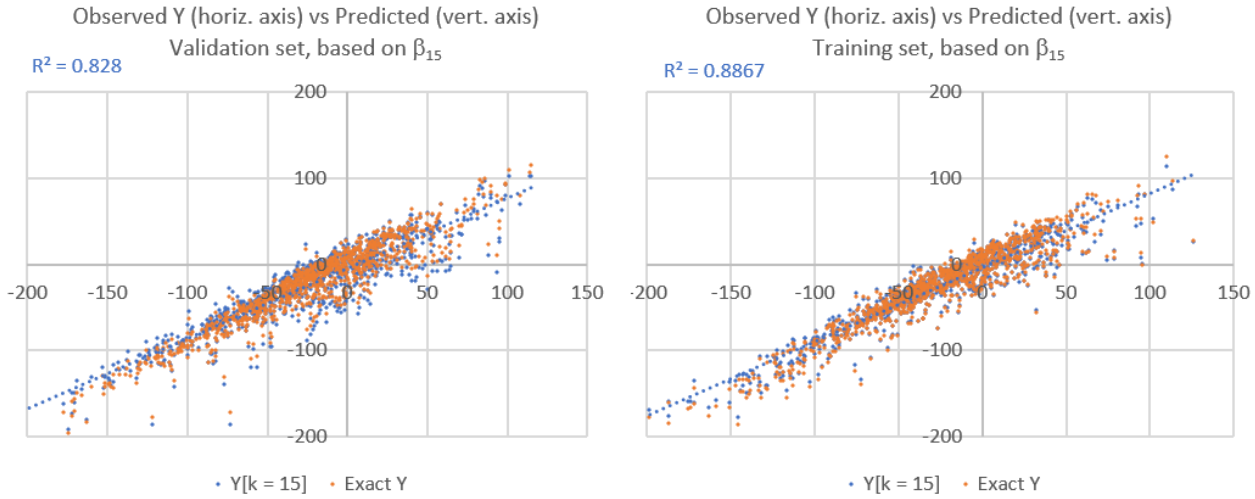

Figure 3: Goodness of fit: training set (right) versus validation set (left)

Figure 2 (left plot) shows how the six regression coefficients converge in the fixed point iterations, starting with the vector $\beta_0 = 0$. The same matrix Λ is used at all times, unlike in figure 1. The oscillating behavior seen in figure 1 never occurs with the fixed Λ defined by equation 4. The right plot in figure 2 is a scatterplot of Y versus the residual error $\epsilon = Y - X\beta$, using $\beta = \beta_{15}$ measured on the training set. In idealized statistical

models, ϵ is assumed to be a white noise. Clearly, this is not the case here, with larger observed values (on the horizontal axis) having a tendency to have a larger error. However, this data is very useful to simulate ϵ, to use in model-free confidence and prediction intervals, as discussed in section 4.2.

Method	r	s	Regression coefficients					
Exact	0.891	67.0%	−1.513	−0.202	1.514	−1.647	1.079	−0.799
β_{15}	0.828	58.5%	−1.726	0.025	1.544	−1.208	1.045	−0.758
β_{100}	0.830	58.8%	−1.460	−0.204	1.517	−1.631	1.103	−0.746
β_0	0.596	36.2%	−1.630	0.116	0.072	−0.947	0.627	−0.689
Excel	0.823	58.7%	−1.426	−0.236	1.542	−1.666	1.153	−0.669

Table 1: Regression coefficients and performance metrics r, s based on methodology

The synthetic data set used in tables 1 and 2 is different from that used in section 4.2. In particular, I increased the amount of noise in the validation set, compared to the training set. I did it to better emulate real data, where performance is usually lower outside the training set. It explains why the R-squared is sensibly better when measured on the training set, as opposed to the validation set.

The performance metric s in table 1 is discussed in section 4. It is much more meaningful than the R-squared r. If measured on the training set, we would have $s = 1-\sqrt{1-r}$. Here, it is measured on the validation set. Yet the equality is still almost satisfied. Also in table 1, the row labeled β_0 corresponds to the special, basic regression method discussed in section 2.3. The `Excel` row corresponds to the standard regression coefficients computed by any statistical package, in this case with the Excel `Linest` function. Note that β_{15}, β_{100} and the Excel regression yield different sets of regression coefficients. Yet the three of them have the same performance.

The easiest to interpret is β_0, followed by β_{15}: in particular, they yield regression coefficients with the same sign as the correlations (the row labeled Y in table 2). If you run the fixed point iteration long enough, eventually β_k will tend to the Excel solution, as $k \to \infty$. Because of the noise in the dataset, even with infinitely many iterations, it is impossible to retrieve the exact, pre-specified regression coefficients featured in the `Exact` row. However, β_{100} and `Excel` provide good approximations.

	Y	X_1	X_2	X_3	X_4	X_5	X_6
Y	1.000	−0.642	0.077	0.033	−0.481	0.404	−0.271
X_1	−0.642	1.000	−0.216	0.458	0.559	−0.266	−0.162
X_2	0.077	−0.216	1.000	−0.061	−0.840	−0.447	0.762
X_3	0.033	0.458	−0.061	1.000	0.048	−0.543	−0.088
X_4	−0.481	0.559	−0.840	0.048	1.000	0.281	−0.620
X_5	0.404	−0.266	−0.447	−0.543	0.281	1.000	−0.700
X_6	−0.271	−0.162	0.762	−0.088	−0.620	−0.700	1.000

Table 2: Correlation matrix

4.2 Distribution-free confidence intervals

This will be the topic of a future article. Here I provide a quick overview, using resampling [Wiki] and related data-driven techniques. Let $Y = P + \epsilon, P = X\beta$ be the regression model, with Y the observed (synthetic) response, P the predicted values, X the $n \times m$ matrix representing the features (with values synthetically generated; n is the sample size), β the regression coefficient vector, and ϵ the residual error vector. In other words, β is the vector minimizing $(Y - X\beta)^T(Y - X\beta)$, or its approximation using the fixed point algorithm.

The starting point is to randomly shuffle the n entries of ϵ, to create a new error vector ϵ'. Let $Y' = Y - \epsilon + \epsilon'$, that is $Y' = X\beta + \epsilon'$. Now find β' that minimizes $(Y' - X\beta')^T(Y' - X\beta')$. Now you get a new set of regression coefficients, β'. Repeat this operation 100 times, and you get 100 sets of regression coefficients: β, β', β'' and so on.

This model-free procedure immediately provides confidence intervals for the regression coefficients, by looking at the empirical distribution [Wiki] of each regression coefficient across the 100 data sets. Now for a

particular feature vector $(x_1, \ldots, x_m)$ – whether part of the training set or not – the predicted value can be obtained in 100 different ways: $p = x\beta, p' = x\beta', p'' = x\beta''$ and so on. The empirical distribution of p, p', p'' and so on, provides a prediction interval for the predicted value of the response, at the arbitrary location x in the feature space. Another way to do it to resample ϵ with replacements (rather than by random permutations). This is then a bootstrapping technique [Wiki].

This methodology assumes that the individual residual errors are independently and identically distributed. I discuss how to address this issue in section 4.2.1. Also it assumes that the error is additive, not multiplicative. In the latter case, one might want to work with a transformed version of Y rather than Y itself. For an Excel implementation, see the `Regression5_Static` spreadsheet on my GitHub repository, here. In the `Test` tab, column `AD` corresponds to $\epsilon' = \epsilon$, and column `AE` to ϵ''. Sort columns `AE:AF` by `AF` (uniform deviates) to reshuffle the residual error. Then Y is automatically replaced by Y'' in column `I`, and the new regression coefficient vector β'' is in cells `K6:P6` in the `Results` tab.

4.2.1 Parametric bootstrap

Another option is to use Gaussian deviates for ϵ', ϵ'' and so on. They need to have the same variance as the one computed on the observed residual error ϵ. This approach is known as parametric bootstrap.

If ϵ is auto-correlated, for example if you are dealing with time series and one of the features is the time, it is possible to use an autoregressive process that has the samee auto-correlation structure as ϵ, to simulate ϵ', ϵ'' and so on. Likewise, if ϵ is correlated to Y, this can be handled with some appropriate parametric model. Parametric bootstrap for linear regression is discussed in [6]. Distribution-free predictive inference for regression is discussed in [7].

5 Feature selection

In section 5.1, I compare the performance of the regression coefficients obtained for each of the potential feature combinations. The performance metric is a significantly improved version of the R-squared; also, it is applied to the validation set, not to the training set. Then, in section 5.2, I discuss stepwise feature selection techniques (forward, backward), adding or removing one or two features at a time, based on the feature table built in section 5.1. The comprehensive feature summary table allows you to quickly perform stepwise regression (which is more interpretable than a full fledged regression), and to assess whether or not this technique fails to catch good enough configurations, at least for the data set investigated here.

5.1 Combinatorial approach

Here I look at all the $2^m - 1$ possible configurations of features $X_1, \ldots, X_m$, to asses the importance of individual features and feature interaction, in a way that is more insightful than looking at cross-correlations between features. For each configuration, I computed the regression coefficient vector using three methods: fixed point with 15 iterations, fixed point with 100 iterations (starting with $\beta_0 = 0$ in both cases), and then the special β_0 alone (no iteration) defined in section 2.3. All rankings, unless otherwise specified, are based on 100 iterations of the fixed point algorithm.

I use the score metric s defined in section 4 – a meaningful, monotonic function of the R-squared – to measure performance. If you use the R-squared instead, the rankings would still be the same. The full model has $m = 6$ features. All other models are sub-models, referred to as configurations: they miss one or more features. The metric s is computed on the validation set, not the training set.

The summary table in in the `Regression5_Static.xlsx` spreadsheet, in the `Results` tab: see columns `U:AS`. The spreadsheet is available here. It has the same structure as the spreadsheet described in sections 2, 2.4, and 3.1.The difference is that the data set is static in this case, so you can't generate different data sets. The methodology is inspired by the book "Interpretable Machine Learning" [8], particularly chapter 7 focusing on permutation feature importance and feature interactions. The higher s, the better the performance. Note that if the performance was computed on the training set rather than the validation set, then we would have $0 \leq s \leq 1$.

m	$s(\beta_{15})$	$s(\beta_{100})$	$s(\beta_0)$
1	0.2892	0.2892	0.2892
2	0.4080	0.4080	0.4075
3	0.4593	0.5019	0.4357
4	0.5175	0.5541	0.4382
5	0.5332	0.5865	0.4274
6	0.5243	0.5889	0.3597

Table 3: Best performance given m (number of features)

Table 3 shows the top achievable performance given m (the number of features used in the computation) on a same (synthetic) data set. The three performance columns correspond respectively to 15 iterations, 100 iterations, and the special β_0. The table is also in the spreadsheet in cells `AP12:AS19`, where you can find the detailed computations. For the performance of each individual configuration, see tables 4 and 5: a configuration denoted as $3, 4, 6$ means that only X_3, X_4 and X_6 are used for the computation of the regression coefficients.

Below are some highlights, after ranking the feature configurations according to performance.

- The top 8 configurations include the simultaneous presence of features X_1, X_2 and X_6. While X_1 is strongly correlated to Y, the features X_2 and X_6 are not. Also, X_2 is negatively correlated to X_1.
- The worst configurations are X_2 alone, X_6 alone, and X_2, X_6 combined together. This is surprising, since X_2 and X_6 are both required in all top configurations. It shows that this type of feature interaction analysis is more powerful than looking at the feature cross-correlation structure.
- The worst configuration out of 63, consisting of X_2 alone, is so bad that it is the only one with a negative s when computed on the validation set. It is worse than using no feature at all (that is, using the mean value of Y for your predictions).
- The 4th configuration has only $m = 4$ features and does quite well. It is also the best possible configuration, among all configurations based on β_0 alone (defined in section 2.3). It is thus the best configuration if you do not use a single iteration of the fixed point algorithm. With the full fixed point algorithm, only one configuration with fewer than 4 features, beats that performance (it needs 3 features only).
- The top configuration performs just as well as the classic statistical solution with $m = 6$, but it also requires all 6 features.
- The biggest improvement is from using two features, over one feature. Beyond two features, gains are smaller.
- The regression coefficient attached to X_5 is positive in the top configuration, but absent (zero) or negative in all the other top 8 configurations except the very top one. A negative value makes more sense, since the correlation between Y and X_5 is strongly negative.
- If for whatever reason, the feature X_6 is not in your data set, you miss all the top 14 configurations, out of 63. This is really the feature that you can't afford not to have. Surprisingly though, it is not highly correlated to Y, much less than X_1, X_3 or X_5. It shows its power not when left alone, but when combined with other features. A bit like charcoal that sounds inoffensive, but when combined with sulfur and saltpeter, makes gun powder.

Of course identifying the ideal configuration is like cherry-picking. However, the goal is to minimize over-fitting and favor simplicity and interpretability. In that regard, the 4th configuration is my favorite, as it uses only $m = 4$ features out of 6, and it is also the winner if you use the special β_0 alone with that same feature configuration. My second pick is the 15th configuration if you use the special β_0 alone. It is the second best configuration if using β_0 alone, it uses only 3 features, and it performs just as well (at that level) as using 100 iterations of the fixed point algorithm. It is also the best configuration without X_6.

5.2 Stepwise approach

Based on the tables 4 and 5, it is easy to reconstruct how stepwise regression progresses [Wiki]. This method is a stepwise feature selection procedure. Here (say) $\{2, 3\}$ denotes the configuration consisting of the two features X_2, X_3.

- Forward regression, adding one feature at a time:

$$1 \to \{1, 3\} \to \{1, 3, 6\} \to \{1, 2, 3, 6\} \to \{1, 2, 3, 4, 6\} \to \text{Full}.$$

The scores $s(\beta_{100})$ are respectively $0.289, 0.408, 0.437, 0.554, 0.587, 0.589$ and the ranks are respectively $40, 23, 15, 4, 2, 1$.

- Backward regression, removing one feature at a time:

$$\text{Full} \to \{1,2,3,4,6\} \to \{1,2,3,6\} \to \{1,2,6\} \to \{1,2\} \to 1.$$

The scores $s(\beta_{100})$ are respectively $0.589, 0.587, 0.554, 0.502, 0.318, 0.289$, and the ranks are respectively $1, 2, 4, 8, 33, 40$.

- Pairwise forward regression, adding two features at a time: $\{1,3\} \to \{1,2,3,6\} \to$ Full.
- Pairwise backward regression, removing two features at a time: Full $\to \{1,2,3,6\} \to \{1,3\}$.

Note that the best configurations respectively with $m = 5, 4, 3, 2, 1$ features, are $\{1,2,3,4,6\}, \{1,2,3,6\}, \{1,2,6\}, \{1,3\}, \{1\}$ scored respectively $0.587, 0.554, 0.502, 0.408, 0.289$, and ranked respectively $2, 4, 8, 23, 40$. So the stepwise procedures, while not fully optimum when involving only one configuration at a time, are nevertheless doing a rather decent job on this data set. The forward regression is easily interpretable if you stop at 4 features.

6 Conclusion

Using linear regression as an example, I illustrate how to turn the obscure output of a machine learning technique, into an interpretable solution. The method described here also shows the power of synthetic data, when properly generated. The use of synthetic data offers a big benefit: you can test and benchmark algorithms on millions of very different data sets, all at once.

I also introduce a new model performance metric, superior to R-squared in many respects, and based on cross-validation. The methodology leads to a very good approximation, almost as good as the exact solution on noisy data, with few iterations, natural regression coefficients easy to interpret, while avoiding over-fitting. In fact, given a specific data set, many very different sets of regression coefficients lead to almost identical predictions. It makes sense to choose the ones that offer the best compromise between exactness and interpretability.

My solution, which does not require matrix inversion, is also simple, compared to traditional methods. Indeed, it can easily be implemented in Excel, without requiring any coding. Despite the absence of statistical model, I also show how to compute confidence intervals, using parametric and non-parametric bootstrap techniques.

References

[1] Jan Ackmann et al. Machine-learned preconditioners for linear solvers in geophysical fluid flows. *Preprint*, pages 1–19, 2020. arXiv:2010.02866 [Link]. 5

[2] Oliver Bröker and Marcus J. Groteb. Sparse approximate inverse smoothers for geometric and algebraic multigrid. *Applied Numerical Mathematics*, 41(1):61–80, 2002. 2

[3] Vincent Granville. Computer vision: Shape classification via explainable AI. *Preprint*, pages 1–7, 2022. MLTechniques.com [Link]. 6

[4] Vincent Granville. Gentle introduction to linear algebra, with spectacular applications. *Preprint*, pages 1–9, 2022. MLTechniques.com [Link]. 3

[5] Vincent Granville. *Stochastic Processes and Simulations: A Machine Learning Perspective*. MLTechniques.com, 2022. [Link]. 2, 6

[6] Chigozie Kelechi. Towards efficiency in the residual and parametric bootstrap techniques. *American Journal of Theoretical and Applied Statistics*, 5(5), 2016. [Link]. 9

[7] Jing Lei et al. Distribution-free predictive inference for regression. *Journal of the American Statistical Association*, 113:1094–1111, 2018. [Link]. 9

[8] Christoph Molnar. *Interpretable Machine Learning*. ChristophMolnar.com, 2022. [Link]. 9

Rank	Configuration	m	$s(\beta_{15})$	$s(\beta_{100})$	$s(\beta_0)$	Regression coefficients attached to β_{100}					
1	$1,2,3,4,5,6$	6	0.524	0.589	0.360	1.421	2.323	1.158	−1.296	0.211	2.221
2	$1,2,3,4,6$	5	0.533	0.587	0.427	1.419	2.411	1.013	−1.119		2.230
3	$1,2,3,5,6$	5	0.515	0.564	0.381	1.703	2.366	0.698		−0.405	1.873
4	$1,2,3,6$	4	0.517	0.554	0.438	1.838	2.163	0.957			1.742
5	$1,2,5,6$	4	0.488	0.549	0.323	1.672	2.909			−0.914	2.246
6	$1,2,4,5,6$	5	0.489	0.548	0.287	1.718	2.849		0.205	−0.961	2.141
7	$1,2,4,6$	4	0.463	0.516	0.285	1.853	2.962		−0.872		2.566
8	$1,2,6$	3	0.459	0.502	0.308	2.165	2.747				2.170
9	$2,3,4,5,6$	5	0.407	0.501	0.273		1.751	2.177	−4.243	1.080	2.995
10	$1,3,4,5,6$	5	0.444	0.474	0.359	1.077		2.052	−1.527	1.011	0.898
11	$2,3,4,6$	4	0.439	0.472	0.357		2.028	1.453	−3.138		2.886
12	$1,3,4,6$	4	0.443	0.442	0.426	1.128		1.391	−0.306		0.565
13	$1,3,5,6$	4	0.436	0.439	0.380	1.405		1.533		0.309	0.459
14	$3,4,5,6$	4	0.366	0.437	0.272			2.784	−4.031	1.688	1.826
15	$1,3,6$	3	0.437	0.437	0.436	1.261		1.363			0.471
16	$1,2,3,4,5$	5	0.413	0.417	0.315	1.884	0.641	1.098	0.877	−0.246	
17	$1,2,3,4$	4	0.414	0.416	0.367	1.879	0.550	1.268	0.646		
18	$1,2,3,5$	4	0.412	0.415	0.345	1.692	0.360	1.512		0.254	
19	$1,2,3$	3	0.413	0.414	0.408	1.599	0.411	1.367			
20	$1,3,5$	3	0.408	0.411	0.344	1.570		1.607		0.347	
21	$1,3,4,5$	4	0.408	0.411	0.314	1.582		1.549	0.135	0.274	
22	$1,3,4$	3	0.409	0.409	0.367	1.538		1.369	0.376		
23	$1,3$	2	0.408	0.408	0.408	1.412		1.417			
24	$1,2,4,5$	4	0.379	0.397	0.245	2.168	1.226		2.320	−1.384	
25	$3,4,6$	3	0.374	0.374	0.356			1.704	−2.066		1.319
26	$2,4,5,6$	4	0.319	0.372	0.200		2.604		−2.077	−1.247	3.095
27	$1,4,5,6$	4	0.337	0.343	0.286	1.516			1.448	−1.071	0.128
28	$1,2,4$	3	0.337	0.341	0.235	2.532	0.901		1.312		
29	$1,4,5$	3	0.336	0.341	0.244	1.593			1.649	−1.123	
30	$1,2,5$	3	0.332	0.335	0.292	1.597	0.731			−0.788	
31	$1,5,6$	3	0.330	0.330	0.321	1.122				−0.714	0.593
32	$2,4,6$	3	0.291	0.328	0.175		2.727		−3.738		3.756

Table 4: Feature comparison table (top 32 feature combinations)

Rank	Configuration	m	$s(\beta_{15})$	$s(\beta_{100})$	$s(\beta_0)$	Regression coefficients attached to β_{100}					
33	$1, 2$	2	0.318	0.318	0.291	2.025	0.653				
34	$1, 4, 6$	3	0.309	0.311	0.283	1.681			0.350		0.494
35	$1, 6$	2	0.310	0.310	0.304	1.536					0.605
36	$1, 5$	2	0.304	0.305	0.291	1.320				−0.728	
37	$1, 4$	2	0.300	0.300	0.234	2.033			0.938		
38	$2, 3, 5, 6$	4	0.276	0.298	0.242		1.454	0.501		−1.711	1.878
39	$2, 5, 6$	3	0.276	0.293	0.205		1.845			−2.057	2.138
40	1	1	0.289	0.289	0.289	1.746					
41	$3, 5, 6$	3	0.252	0.252	0.242			1.076		−1.089	0.945
42	$2, 3, 4, 5$	4	0.227	0.249	0.200		−0.989	2.350	−1.997	0.477	
43	$2, 3, 4$	3	0.241	0.241	0.214		−0.820	2.022	−1.535		
44	$4, 5, 6$	3	0.216	0.216	0.199				−0.638	−1.340	1.119
45	$3, 4$	2	0.215	0.215	0.216			2.120	−1.879		
46	$3, 4, 5$	3	0.204	0.209	0.200			1.592	−1.070	−0.716	
47	$5, 6$	2	0.203	0.203	0.203					−1.642	0.972
48	$2, 3, 5$	3	0.188	0.188	0.178		−0.564	1.329		−1.029	
49	$3, 6$	2	0.180	0.180	0.179			1.891			1.198
50	$3, 5$	2	0.180	0.180	0.179			1.123		−1.384	
51	$2, 3, 6$	3	0.177	0.178	0.179		−0.242	1.910			1.019
52	$4, 6$	2	0.169	0.169	0.171				−2.353		1.730
53	5	1	0.139	0.139	0.139					−1.970	
54	$2, 3$	2	0.138	0.138	0.094		−1.132	2.086			
55	$2, 5$	2	0.137	0.137	0.139		−0.190			−1.887	
56	$4, 5$	2	0.135	0.134	0.126				0.487	−2.163	
57	$2, 4, 5$	3	0.135	0.133	0.126		−0.158		0.454	−2.081	
58	3	1	0.096	0.096	0.096			2.257			
59	4	1	0.071	0.071	0.071				−2.187		
60	$2, 4$	2	0.033	0.033	0.072		−1.073		−1.719		
61	$2, 6$	2	0.021	0.025	0.022		0.127				1.735
62	6	1	0.020	0.020	0.020						1.643
63	2	1	−0.057	−0.057	−0.057		−1.433				

Table 5: Feature comparison table (bottom 31 feature combinations)