Investigation of Optimal Polynomial Regression of Wine Quality Using Batch and Stochastic Gradient Descent

Priyanka Chakraborti
University of Nebraska-Lincoln
Lincoln, USA
bryan.moore@huskers.unl.edu

Bryan Moore
University of Nebraska-Lincoln
Lincoln, USA
priyanka.chakraborti@huskers.unl.edu

***Abstract*—We have utilized two types of gradient descent to investigate the optimal parameters in fitting a polynomial regression model for the UCI wine quality dataset. These are Batch Gradient Descent and Stochastic Gradient Descent. Both have been hand-built from scratch so as to allow us full granular control over the machinery of our model. The best results are reported, along with the corresponding optimal hyperparameters. Overall, 1st order polynomial regression via Batch Gradient Descent showed the best agreement with the test set. The corresponding hyper-parameters are learning rate of 0.1, regularization function l1, and regularization parameter 1.0.**

***Index Terms*—Batch Gradient Descent, Stochastic Gradient Descent**

I. Introduction

Linear regression, as the name suggests, requires the relationship between the dependent variable and the independent variable to be linear. However, if the distribution of the data is more complex, as is often the case with real-world data sets, one has to address the obvious problem of under-fitting which comes from applying a linear model to more complex dataset. Hence, polynomial regression is a very important addition to any machine-learning toolbox. There are various implementations of polynomial regression. One approach implements this via a closed form linear regression solution. This however requires inversion of our feature matrix, which can become intractable with large numbers of features.

In this article we implement two commonly used algorithms, Batch gradient descent and Stochastic gradient descent. We report the appropriateness of applying these models (as opposed to the above-mentioned closed-form solution) to the UCI wine quality data-set. For either form of gradient descent we implement hyper-parameter tuning with the primary goal being the establishment of the optimal parameters for the considered model. Since any iterative 'polynomial regression' approach is particularly sensitive to the units of the individual features we have implemented feature scaling as a means to achieve quicker, and more accurate, convergence to the optimal weights for each model. The appropriateness of this is discussed below.

II. DATA SUMMARY

This data consists of 12 features. These are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol, and quality. The quality of the wine has been given a rating from 0 to 10, with 10 representing the highest quality possible. The remaining features are all based on physio-chemical tests. The broad statistical description for each of these features is summarized in Table 1. For this investigation the quality of the wine is the target for which we wish to generate a prediction. We thus investigated whether the remaining eleven features correlate/covary significantly with any other features such that some can be considered redundant. The pairwise seaborn plot used for making these decisions has been included in the supplementary section of this paper.

From visual inspection of this plot we conclude that citric acid covaries significantly with fixed acidity (correlation=0.67) and volatile acidity (correlation=-0.55). Additionally, free sulfur dioxide shows strong covariance with total sulfur dioxide (correlation=0.67). Lastly, it appears that fixed acidity strongly covaries with density (correlation=0.67) and pH (-0.68). We thus decided to remove the features citric acid, free sulfur dioxide, and fixed acidity from our model. It should be clarified, as this is often a point of confusion, that correlation and covariance are not the same. It is possible for a relationship to have no correlation but still have a significant covariance. Both should thus be investigated when deciding whether to remove features from a model.

III. METHODS

For this investigation we split our data into testing and training sets, keeping 20% aside for testing. We then employed cross-fold validation to identify the optimal hyper-parameters (learning-rate, regularization parameter, and regularization function). Utilizing polynomial regression these parameters are then applied to the train and test sets. Learning curves are generated to assess (a)the optimal degree of polynomial to use for regression and (b)the optimal number of elements to

TABLE I
STATISTICAL DESCRIPTION OF FEATURES

	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Sulphates	Alcohol	Quality	Density	pH
Count	1599	1599	1599	1599	1599	1599	1599	16	1599	1599	1599	1599
Mean	8.322	0.53	0.27	2.54	0.087	15.87	46.47	0.66	10.42	5.64	1.00	3.31
Std	1.74	0.18	0.19	1.41	0.047	10.46	32.90	0.17	1.066	0.81	0.0019	0.15
Min	4.60	0.12	0.00	0.90	0.012	1.00	6.00	0.33	8.40	3.00	0.99	2.74
25%	7.10	0.39	0.09	1.90	0.070	7.00	22.00	0.55	9.50	5.00	1.00	3.21
50%	7.90	0.52	0.26	2.20	0.079	14.00	38.00	0.62	10.20	6.00	1.00	3.31
75%	9.20	0.64	0.42	2.60	0.090	21.00	62.00	0.73	11.10	6.00	1.00	3.40
Max	15.90	1.58	1.00	15.50	0.61	72.00	289.00	2.00	14.90	8.00	1.0037	4.01

This is a statistical description of all features included in this dataset. For ease of reading values have been rounded to two significant figures after the decimal.

retain in the training set during the fit. All methods utilized either Batch Gradient Descent or Stochastic Gradient Descent for calculation of the weights. The weights obtained from these optimal hyper-parameters are then used for producing predictions on our target feature (quality) in the test set. The option was introduced to allow the choice as to whether to scale the feature matrix before linear regression or not, but as is discussed in the Results section, scaling is necessary. More details about these procedures are included below.

A. Gradient Descent

We have chosen to implement gradient descent on this dataset instead of utilizing a closed-form solution. The main reason for this is that we would like to be able to perform regression for high-order polynomials. The time-complexity of the closed-form solution scales as O(d^3), where d is the number of features. Even for our model, where we have removed three of the 11 features, a fifth-order polynomial already has 1287 features after feature expansion. Thus, if we continue to use the closed-form solution this calculation quickly becomes intractable at large model complexities as these result in feature matrices with very large numbers of features.

For optimizing the fit we have implemented a gradient descent algorithm. This takes as input the feature matrix. It then goes through a maximum number of iterations, where with each iteration the weight vector is updated by following the condition corresponding to minimizing the error function, which corresponds to following the maximum negative gradient. At each iteration we evaluate the 'RMSE' between the generated prediction and the true target value. If desired a reasonable value for the tolerance function can be defined. This works such that if the change in error between subsequent iterations is greater than the tolerance, we proceed to the next iteration. However, if the condition is met the function quits and returns the weight vector which satisfies the tolerance condition.

If this condition is never met, or not desired, this downward hill climbing approach continues until we reach the maximum number of iterations. At that point we have hopefully converged to the global minima of our error function. If not then different hyper-parameters should be chosen, which is why we employ cross-fold validation to find the optimal combination of hyper-parameters. It must also be noted that the error function we have used is quadratic, which thus implies that the gradient is linear. Thus, for the approaches considered in this paper any minima found must, by definition, by the global minima. However, this should not be taken as a general result as the choice of our error function is not arbitrary, but a direct consequence of minimizing the KL divergence while assuming an underlying Gaussian distribution for the latent random features of our model space. The UCI wine dataset is far from being a classic 'sparse' dataset, and can reasonably be assumed to have a gaussian-like distribution for its features. We thus argue that our use of these error metrics is valid, although certainly it would be better if a test for Gaussian-behavior was included in our approach.

We have implemented the option to utilize Batch Gradient Descent or Stochastic Gradient descent. During Batch Gradient Descent the set of weights are updated by using all elements in the feature matrix. During Stochastic Gradient Descent a random element is chosen during each iteration. In general, Batch Gradient descent will converge to the optimal weights in less iterations. However, for each iteration Stochastic Gradient descent can make an update very quickly, as it only needs to use a single row of the feature matrix for its calculation. Thus, when compared with Stochastic Gradient Descent, Batch Gradient Descent can take a comparatively Long Time to converge as it must perform matrix operations on the entire feature matrix during each iteration. Thus, if given the same learning rate the Stochastic Gradient Descent will likely reach an optimal value first, in terms of time, but later in terms of number of iterations.

That said, although technically both approaches can reach an optimal set of weights, there are some significant differences to be aware of when using these. If considering an error metric with both local and global minima Stochastic Gradient Descent may be the optimal choice as the 'descent' will be more chaotic. Thus, with Stochastic Gradient Descent it is possible to for its descent to get past a local minima and proceed to find the true global minima. However, this chaotic behavior also has a downside. When Stochastic Gradient Descent does reach the 'area' containing the optimal weights, these predicted weights will likely continue to 'walk around' the optimal values. For our dataset implementing either algorithm leads to a similar set of optimal weights. Thus, for this data it appears that drawing a single sample at every iteration does not introduce chaos such that we are not able to settle on a near-optimal set of weights.

As mentioned above, Stochastic Gradient Descent is not as likely to find the optimal set of weights as accurately as Batch Gradient Descent can. However, one possible solution, which we have not implemented in this investigation, would be to decrease the learning rate for each iteration. This would thus minimize the 'walking around' which occurs near the optimal values for the weights. Another advantage to note for Stochastic Gradient Descent is that it can work just as well for data-sets where it is impossible to fit the entire set in memory. In this situations Batch gradient descent would fail, and Stochastic Gradient Descent would have to be used. With the dataset we are considering that is not the case, and all things considered the user appears to have a relatively equal choice between the two choices for gradient descent.

We have implemented three different options by which the weights can be updated on each iteration. These are Regression (unregularized descent), Lasso Regression (L1 regularized descent), or Ridge Regression (L2 regularized descent). For all three methods the user specifies the learning rate, and for Lasso Regression and Ridge Regression the user also specifies the regularization parameter. In practice these parameters are determined through cross-fold validation on the feature matrix being used in the model. For this cross-fold validation three lists, one for each of the above-mentioned values, are sent in to be assessed. The optimal combination is then identified. The user also sets a maximum number of iterations to go through and the tolerance level for exiting early.

B. Feature Space Expansion

In order to fit our data to higher order models, such as 3rd order polynomial regression, we must be able to calculate all terms in our feature space starting from first order up to the highest order of the specified model complexity. To accomplish this we have created a dynamic function which calculates all possible unique multiplicative terms for our features, up to the provided dimension. It then creates a new column for each possible multiplication. This feature matrix is scaled before it is returned for input into our gradient descent class.

C. Feature Space Expansion

Feature scaling has many benefits. One is that it results in a feature matrix which is computationally more tractable in terms of the magnitude of the values within the matrix. Additionally, it reduces problematic effects whereby certain features have a higher variance, and are thus more dominant than others. For a convex loss function this issue generally only affects the convergence speed, but for a non-convex loss function this can actually cause us to miss the optimal weights entirely. For the gradient descent algorithm one of the largest problems is that if the scale of each feature is significantly different than the rest, then for each step we take each feature will updated by a different amount. As such, certain features of high variance will be driven too quickly as compared to the others. As we only define a single overall Learning Rate this is a huge problem. In practice we even found that for higher order polynomials, if the feature matrix is not modified before gradient descent all outputs will be NaN. It is for these reasons that we always standardize our feature matrix before optimizing the weights via gradient descent.

D. Learning Curve for Training Set Size

Although cross-fold validation is used to optimize the choice of the regularization function, the regularization parameter, and the learning rate, there are still a few other parameters to optimize. Among these are the number of elements to send into the fit function from the training set and the order of the polynomial to fit to for polynomial regression. To assess these we have written functions which create learning curves to assist in visualizing both situations.

For deciding the optimal number of elements to utilize from the training set we define a bin size. We then send in elements in integer multiples of that number into the cross-validation program, one batch at a time. We record the RMSE for the training and validation sets for each training size sent in. This curve is then used to assess at what number of elements we no longer see a significant improvement in the RMSE for the validation set and generalization error between the training and validation sets.

E. Learning Curve for Model Complexity

To investigate the optimal model complexity (polynomial order) we implement the following approach: (a) Use our feature expansion program to create and send into the cross-fold validation function a feature matrix corresponding to the degree polynomial we want to assess. For example, for assessing 2nd order polynomial regression we would create the feature matrix corresponding to all 1st and 2nd order terms and send this into our cross-fold validation program. (b)Record the average RMSE for the testing and training sets after cross-fold validation is complete. From these results we manually make a judicious choice of learning rate, regularization parameter, and regularization function to send into the next part of this process. (c) With the optimal parameters from step (b) hard-coded in we run cross-validation on the training set (for these optimal parameters) and generate the RMSE values for the given order of polynomial regression.

The polynomial degree plot is generated by repeating steps (a) and (b). Once that has been completed for all model complexities being considered we then cycle through (c) for polynomials up to order five. From the created Learning Curve we can then robustly assess whether the validation

TABLE II
HYPER-PARAMETER TUNING FOR POLYNOMIAL REGRESSION

	Order 1	Order 2	Order 3	Order 4	Order 5
MSE	0.8836	1.0013	10.1605	153.4894	832.3524
Learning Rate	0.1	0.1	0.01	0.0001	0.0001
Reg Fncn	l1	l1	l1	l1	l1
Reg Param	1.0	1.0	0.1	1.0	1.0

These are the results from hyper-parameter tuning for all orders of polynomials included in this investigation. MSE is the mean-squared-error calculated for the validation set during cross-fold validation, Reg Fncn is the regularization function used for updating the weights during gradient descent, and Reg Param is the corresponding regularization parameter.

RMSE is improving for different powers and also if there are clear signs of over/under fitting in our model at different complexities. Utilizing a separate set of hyper-parameters for each complexity being compared ensures that we are not biasing our investigation by using hyper-parameters which work well for certain orders and poorly for others.

IV. RESULTS

The results are summarized in Table II, Fig. I, Fig. II, and Fig. III. Overall, we find that 1st order polynomial regression via Batch Gradient Descent shows the best agreement with the test set. The corresponding hyper-parameters are learning rate of 0.1, regularization function l1, and regularization parameter 1.0. Below is a more detailed breakdown of the investigation, and resultant conclusions.

A. Hyper-parameter Tuning

Due to the overall smoother descent common to Batch Gradient Descent we begin by considering Batch Gradient Descent. After investigating this we will compare with the fit obtained from Stochastic Gradient Descent. Using Batch Gradient Descent we first employed cross-fold validation on the standard feature matrix, without performing any feature expansion. We identified that the optimal combination of hyper-parameters for our dataset is to use a l1 regularization function, a regularization parameter of 1.0, and a learning rate of 0.1. This was tested using a maximum iteration number of 100000 and a tolerance for identifying to significant improvement in error of 0.0001. This combination gives a MSE of 0.41 for the train set and a MSE of 0.43 for the test set.

As a final step, we augment our feature set to a space of polynomial of degree three and run cross-fold validation on it. The optimal hyper-parameters used for each model complexity are shown in Table II, and the resulting plot is shown in Fig. III.

B. Learning Curve for Training Size

We have investigated the Learning curve for training size for 1st order regression and also for 3rd order regression. These are both displayed in Fig. 1. For the 1st order fit we see that when a very low set of instances are used as a training set, the model learns the weights well and thus starts out with a very low 'RMS' error for the train set. However, it is enormous for the validation set as it is not possible to generalize the prediction from the few samples observed. A large 'RMS' error on both train and validation sets are typical of an 'under-fitting' model which matches the behaviour that we expect at low numbers of training samples. As the training size increases there is a sharp decrease in the generalization error as the error on the validation error quickly decreases. Essentially, the model is quickly able to learn the pattern much better once we include some more training samples. With increasing train sizes the 'RMSE' error does not show considerable improvement past about 500 samples, although it does still continue to drop slightly. At around 1000 there is both low RMSE on the validation set and small generalization error between the train and validation sets. We choose to use 1000 samples in our train set, although from the plot it is likely that anything greater than 800 would be adequate.

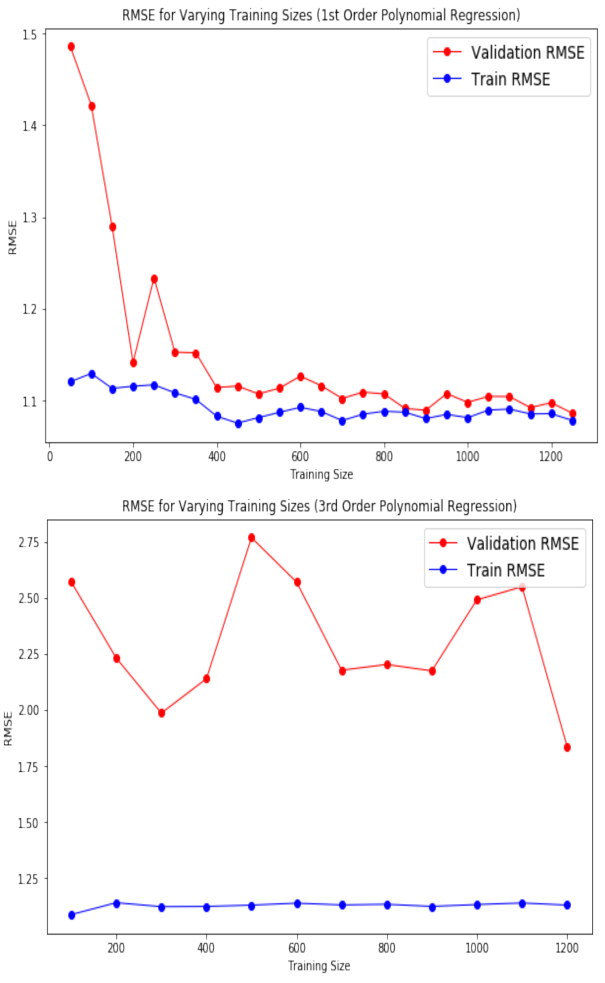

Fig. 1. These show the training curves for 1st and 3rd order regression for different training sizes. We can easily see that the 3rd order regression does not have any training size at which the generalization or validation error is low. It appears to have severe over-fitting problems for 3rd order regression..

When looking at the learning curve for a polynomial of degree three at first glance it appears to show considerable reduction in the 'RMS' error for both train and validation sets by about 600 samples. There is however, a distinctive 'gap' between where the errors plateau for the two sets. This is symptomatic of an 'overfitting' model. Table-II clearly shows that for third-degree polynomial regression the optimal RMSE for the validation set is already a magnitude higher than that for the 1st or 2nd order regression models. This is clearly seen in Fig-1 as well. Fig. 2 shows this clearly as well. The RMSE increases drastically at 3rd order polynomial regression. There is clearly no training set size at which a 3rd order polynomial regression approach on this data provides a reasonable RMSE for the validation set of a reasonable generalization error.

For a more in-depth investigation of the relationship between model complexity, RMSE, and generalization error we have created a Learning Curve to assess the impact for all possibilities model complexities, from 1st order linear regression to 5th order polynomial regression. The results are displayed in Fig. 3. To ensure that this is a fair comparison we have hard-coded in the optimal parameters for each model complexity (given in Table II). This increases the probability that the RMSE we see for each order number corresponds to the minimum RMSE possible.

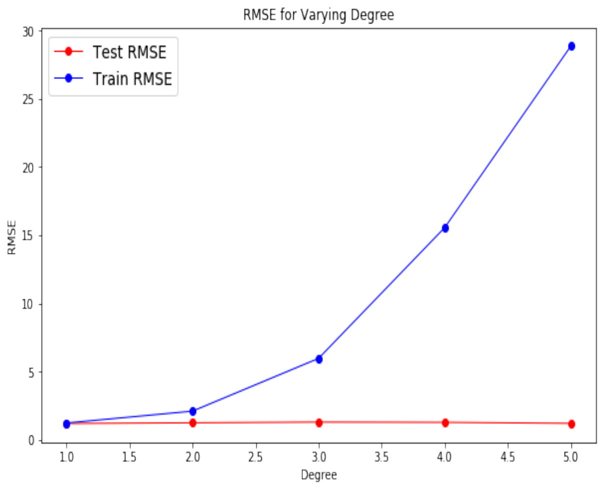

Fig. 2. Assessment of optimal model complexity. From this plot we can clearly see that while 1st and 2nd order polynomial regression do give reasonable validation and generalization errors, that any orders past that are much worse. As we see that the RMSE for 1st order regression (0.8836) is slightly lower than the RMSE for 2nd order regression (1.0013) and that the generalization error is visually slightly larger for 2nd order regression, we will choose to use 1st order regression for our investigation.

C. *Optimal Model*

So far we have been investigating Batch Gradient Descent. For this we find that the optimal model utilizes 1st order linear regression, a learning rate of 0.1, the regularization function l1, and a regularization parameter of 1.0. Running these parameters on the test set gives a MSE of 0.44. This is a fairly good fit.

However, we have not yet tested Stochastic Gradient Descent. As the underlying behavior of the data is not different between the two, we choose to compare the previous results to Stochastic Gradient Descent for 1st order regression. Performing cross-validation on this provided the following optimal hyper-parameters. The error is minimized for a learning rate of 3.0, the regularization function l2, and a regularization parameter of 0.01. Running these parameters on the test set gives a MSE of 0.49. This is nearly as good as we were able to get using Batch Gradient Descent, but a little bit higher. Thus, we will choose the use Batch Gradient Descent as it is able to identify the optimal weights a little bit more accurately than Stochastic Gradient Descent for this dataset.

V. SUPPLEMENTAL MATERIAL

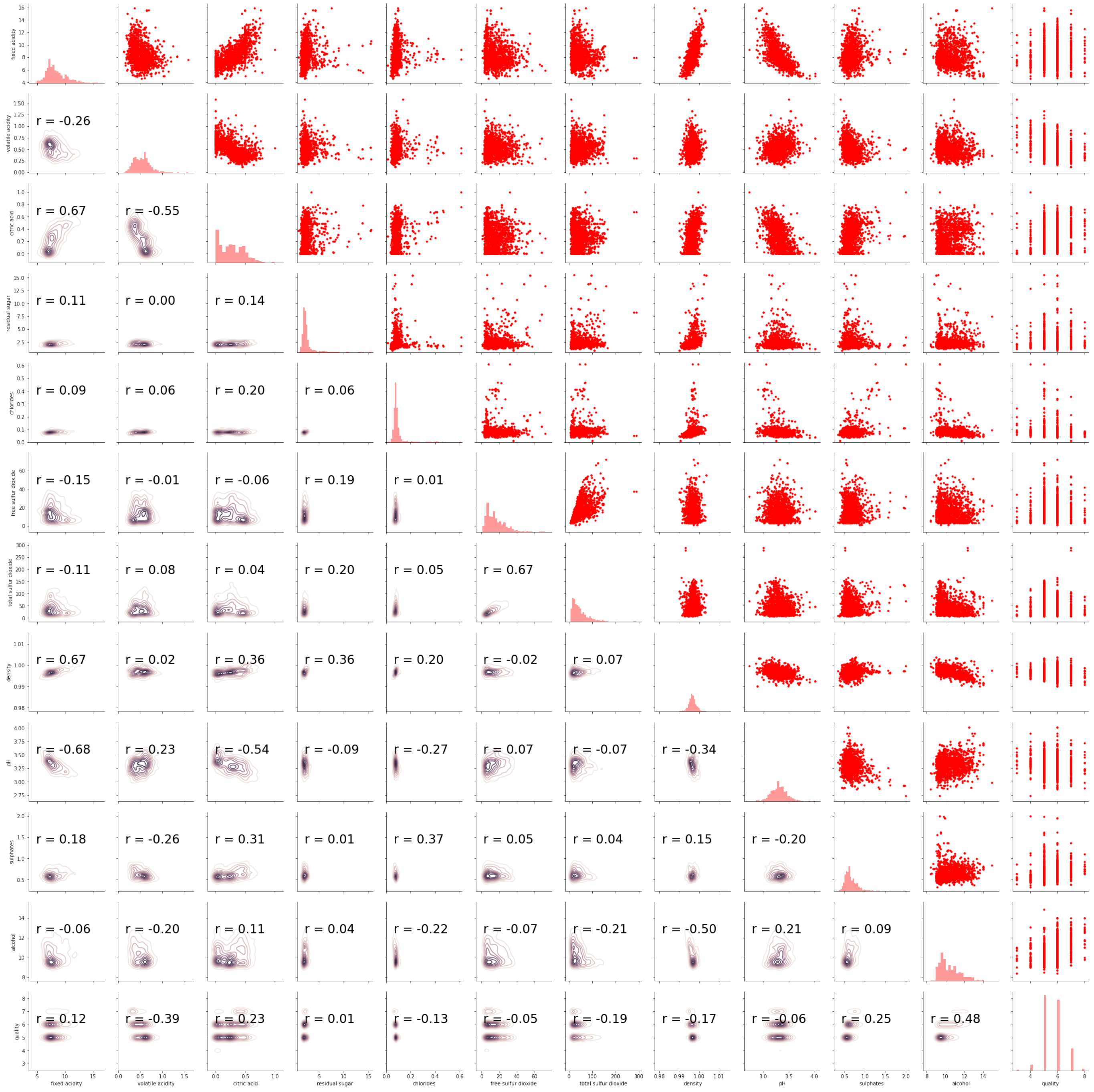

Fig. 3. These are pairwise plots which are used to identify redundant features. The number displayed below the diagonal corresponds to the correlation between features.