Project 4 Report : Compare Compare regression methods

Sajal Kumar

Implementation details on parameters

My implementation uses all 5 methods : Linear Regression, RANSAC Regression, Lasso Regression, Ridge Regression and Decision Tree Non-Linear Regression on their default setting except the following general parameters that could be changed:

- `random_state` : Random seed (set to 1 by default).
- `max_iter` : maximum number of iterations for methods using gradient descent (set to 50 by default)
- `n_jobs` : Number of parallel threads allowed (set to 4 by default).

My implementation allows changes to the following regressor specific parameters:

- RANSAC Regressor
  - `min_samples` denoted by `min_samples_ransac` (set to 0.5 by default).
- Lasso Regressor
  - `alpha` denoted by `lambda_l1` (set to 1 by default).
- Ridge Regressor
  - `alpha` denoted by `lambda_l2` (set to 1 by default).
- Decision Tree Non-Linear Regressor
  - `min_samples_split` denoted by `min_samples_split` (set to 25 by default).

Apart from the above mentioned parameter some other regressor parameters were changed but were not provided to the user:

- Linear Regressor, Lasso Regressor and Ridge Regressor

- `fit_intercept` was set to ‘False’ (since the data was properly scaled and centralized).

The above represents the ‘standard’ setting for all regression methods when no parameter is changed.

Linear Regression using normal equation solver

This is implemented as a separate class in the project. It requires no additional parameter. `-lin_solver` activates it. See the `ReadME` file for examples.

Performance on House Pricing Data-set

Info	**Lin. Reg.**	**RANSAC**	**Lasso**	**Ridge**	**DT Reg.**	**Lin. Reg. Eqn.**
runtime	0.0037	0.041	0.0006	0.001	0.002	0.0032
mse on test	0.23	0.24	1.09	0.23	0.14	0.23
r2 on test	0.78	0.78	-0.002	0.78	0.86	0.78
mse on train	0.28	0.33	0.96	0.28	0.1	0.28
r2 on train	0.71	0.65	-0.00045	0.71	0.89	0.71

Table 1: Result on ‘Housing price’ data-set with ‘standard’ configuration of 6 regressors.

Table 1 shows the runtime (in seconds), mse and r2 scores on testing and training data for ‘Housing price’ data-set using the ‘standard’ configuration on 6 regressors : Lin. Reg. (Linear Regression), RANSAC , Lasso, Ridge, DT Reg. (Decision Tree Non-Linear Regression) and Lin. Reg. Eqn. (Linear Regression using normal equations). The data-set was scaled using the ‘StandardScaler’ method from sklearn. 70% of the data was randomly partitioned for training and the rest 30% was used for testing. All methods do well on the this data-set except Lasso. I believe it is because Lasso works well when the number of independent variable is big. However, In this data set there are only 13 features and thus Lasso is under-fitting at `alpha = 1`. The non-linear Decision Tree regressor worked the best for this data-set.

Performance on California Energy Production Data-set

This data-set reported the amount of energy generated hourly by various renewable energy sources within the ISO grid in California. The only meaningful factor here is time and maybe some correlation between different energy sources, e.g : Hydro-electric energy generation can be affected by weather (precipitation to be exact) which will also affect both Solar and Wind energy. I removed timestamp from this data-set and introduced months. The rationale behind introducing the variable month was to introduce the context of seasons which can greatly affect Hydro, Wind and Solar energy. For Solar energy there were three features *'SOLAR'*, *'SOLAR PV'* and *'SOLAR THERMAL'* that contained missing values. Interestingly, *'SOLAR PV'* and *'SOLAR THERMAL'* had missing values for rows where *'SOLAR'* had valid entries and vice-versa. This could mean that the 3 features were related and thus I merged *'SOLAR PV'* and *'SOLAR THERMAL'* into *'SOLAR'* by using the following equation when *'SOLAR'* was missing value:

$$S[i] = \mu_S . \frac{SPV[i]}{\mu_{SPV}} + \mu_S . \frac{ST[i]}{\mu_{ST}} \tag{1}$$

where $S[i]$ represents the ith instance of *'SOLAR'* that was missing a value, $SPV[i]$ represents the ith instance of *'SOLAR PV'*, $ST[i]$ represents the ith instance of *'SOLAR THERMAL'*, μ_S represents mean of non-missing values in *'SOLAR'*, μ_{SPV} represents the mean of non-missing values in *'SOLAR PV'* and μ_{ST} represents the mean of non-missing values in *'SOLAR THERMAL'*.

I divided the data-set into two sub-datasets. One with *'SOLAR'* as response and *'MONTH'*, *'Hours'* and *'SMALL HYDRO'* as features; another with *'WIND TOTAL'* as response and *'MONTH'*, *'SOLAR'* and *'SMALL HYDRO'* as features.

Solar energy prediction

Info	**Lin. Reg.**	**RANSAC**	**Lasso**	**Ridge**	**DT Reg.**
runtime	0.0058	0.10	0.0028	0.0028	0.094
mse on test	0.98	1.12	1.008	0.98	0.47
r2 on test	0.023	-0.113	-4.97E-07	0.023	0.53
mse on train	0.97	1.11	0.99	0.97	0.31
r2 on train	0.022	-0.11	-9.24E-08	0.022	0.68

Table 2: Result on ‘Solar Energy’ data-set with ‘standard’ configuration of 5 regressors.

Table 2 shows the runtime (in seconds), mse and r2 scores on testing and training data for ‘Housing price’ data-set using the ‘standard’ configuration on 5 regressors. The data-set was scaled using the ‘StandardScaler’ method from sklearn. 70% of the data was randomly partitioned for training and the rest 30% was used for testing. All linear methods do poorly on the this data-set, with Lasso doing exceptionally worse, (the same reasoning as 'House pricing' can be applied here). The non-linear Decision Tree regressor worked very well and was the only one that could capture the dynamics of the data-set.

Wind energy prediction

Table 3 shows the runtime (in seconds), mse and r2 scores on testing and training data for ‘Wind energy’ data-set using the ‘standard’ configuration on 5 regressors. The data-set was scaled using the ‘StandardScaler’ method from sklearn. 70% of the data was randomly

Info	**Lin. Reg.**	**RANSAC**	**Lasso**	**Ridge**	**DT Reg.**
runtime	0.008	0.4	0.004	0.002	0.128
mse on test	0.97	0.99	1	0.97	1.10
r2 on test	0.02	0.015	-4.01E-07	0.026	-0.102
mse on train	0.97	0.98	0.99	0.97	0.70
r2 on train	0.022	0.011	-7.40E-08	0.022	0.3

Table 3: Result on 'Wind Energy' data-set with 'standard' configuration of 5 regressors.

partitioned for training and the rest 30% was used for testing. All methods do poorly on the this data-set. The non-linear Decision Tree regressor is clearly overfitting having a much better performance on training data-set in contrast to that on the testing dataset. However, it makes sense because the default value of 25 on `min_samples_split` on a data-set with 67K instances is prone to overfitting.

A more comprehensive evaluation of regressors

In this section we would discuss and show changes in performance when certain parameters were tweaked. We are using 'House pricing', 'Solar energy' and 'Wind energy' data-set.

- Linear Regression : This method does not have any parameter that could have influenced the performance.
- RANSAC Regression :
  - `min_samples_ransac` : The following table shows the change in results at 2 different values of `min_samples_ransac` on 'House pricing' data-set The results in

values $\rightarrow$	**0.5**	**0.9**
runtime	0.041	0.048
mse on test	0.24	0.24
r2 on test	0.78	0.77
mse on train	0.33	0.29
r2 on train	0.65	0.69

Table 4: Change in RANSAC result with `min_samples_ransac`

Table 4 are expected, increasing `min_samples_ransac` does not effect performance on the testing data (very slightly) but improves performance on training data.

- Lasso
  - `lambda_l1` : The following table shows the change in results at 3 different values of `lambda_l1` on 'House pricing' data-set. The results in Table 5 support my

values →	**1**	**0.5**	**0.1**
runtime	0.0006	0.0008	0.002
mse on test	0.24	0.8	0.34
r2 on test	-0.002	0.26	0.68
mse on train	1.09	0.67	0.35
r2 on train	-0.00045	0.3	0.64

Table 5: Change in Lasso result with `lambda_l1`

earlier hypothesis that having lower number of features affects Lasso as decreasing `lambda_l1` greatly improves performance.

- Ridge : No significant changes were noticed by changing `lambda_l2`.
- Decision Tree Non Linear Regression
  - `min_samples_split` : The following table shows the change in results at 3 different values of `min_samples_split` on 'Solar energy' data-set. Changes on 'Wind energy' data-set has also been discussed. The results in Table 6 is expected. As

values →	**25**	**100**	**1000**
runtime	0.094	0.078	0.05
mse on test	0.47	0.41	0.4
r2 on test	0.53	0.59	0.61
mse on train	0.31	0.36	0.4
r2 on train	0.68	0.63	0.60

Table 6: Change in Decision Tree regression result with `min_samples_split`

the Decision Tree stops earlier, performance on test data improves, while that on training data degrades. Application on 'Wind Energy' showed similar behavior, however, the performance did not had any dramatic improvement.