IT496: Introduction to Data Mining

Lecture 15

Variants of Regression

[Regularization]

Arpit Rana

15th September 2023

Regularized Linear Models

You are building a predictor but its performance is not good enough. What should you do?

Some of the options include:

- gather more training examples;

may not help. . .

- remove noise in the training examples;
- add more features or remove features;

may in some cases worsen the performance. . .

- change model: move to a more complex model or maybe to a less complex model;
- stick with your existing model but add constraints to it to reduce its complexity or remove constraints to increase its complexity.

...it all depends on what is causing the poor performance (underfitting or overfitting).

Underfitting

If your model underfits:

- gathering more training examples will not help.

Your main options are:

- change model: move to a more complex model;
- collect data for additional features that you hope will be more predictive;
- create new features which you hope will be predictive (see examples of feature engineering in the LA-01 Lab colab file);
- stick with your existing model but remove constraints (if you can) to increase its complexity.

Overfitting

If your model overfits, your main options are:

- gather more training examples;
- remove noise in the training examples;
- change model: move to a less complex model;
- simplify by reducing the number of features;
- stick with your existing model but add constraints (if you can) to reduce its complexity.

Regularization

If your model underfits, we saw that one option is:

- stick with your existing model but remove constraints (if you can) to increase its complexity.

If your model overfits, we saw that one option is:

- stick with your existing model but add constraints (if you can) to reduce its complexity.

Constraining a model to make it less complex and reduce the risk of overfitting is called regularization.

Regularization is a general concept but we will explain it in the case of linear regression in the rest of this lecture.

Regularization for Linear Regression

Linear models are among the least complex models.

- Hence, we normally associate them with underfitting.

But, even linear regression (multivariate) might overfit the training data. If you are overfitting, you must reduce the degrees of freedom.

- One way is to discard some features.
    - Then you have fewer coefficients (β) that you can modify.
- Another way is to constrain the range of values that the coefficients can take:
    - E.g. force the learning algorithm to only choose small values (close to zero). This makes the distribution of the values of the coefficients more regular.

Regularization for Linear Regression

Recall that OLS linear regression finds coefficients that minimize

$$\begin{aligned} J(X, y, \beta) &= \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta\left(x^{(i)}\right) - y^{(i)} \right)^2 \\ &= \frac{1}{2} mean(X\beta - y)^2 \end{aligned}$$

Regularization imposes a penalty on the size of the coefficients.

This is how we regularize linear regression.

- In effect, it penalizes hypotheses that fit the data too well.

Lasso Regression: Using the L$_1$-norm

'Lasso' stands for 'least absolute shrinkage and selection operator' — but this doesn't matter!

- We penalize by the L$_1$-norm of β, which is simply the sum of their absolute values, i.e.

$$\sum_{j=1}^{n} |\beta_j|$$

Minor point: we don't penalize β_0, which is why j starts at 1.

So Lasso Regression finds the β that minimizes

$$J(X, y, \beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} |\beta_j|$$

λ is called the 'regularization parameter'.

Lasso Regression: Using the L_1-norm

Regularization Parameter (λ, in scikit-learn it is `alpha`)

It controls how much penalization we want and this determines the balance between the two parts of the modified loss function: *fitting the data* versus *shrinking the parameters*.

- As $\lambda \rightarrow 0$, Lasso Regression gets closer to being OLS Linear Regression.
- When $\lambda = 0$, Lasso Regression is the same as OLS Linear Regression.
- When $\lambda \rightarrow \infty$, penalties are so great that all the coefficients will tend to zero: the only way to minimize the loss function will be to make the coefficients as small as possible. It's likely that in this case we will underfit the data.

So, for regularization to work well, we must choose the value of λ carefully.

So what kind of thing is λ?

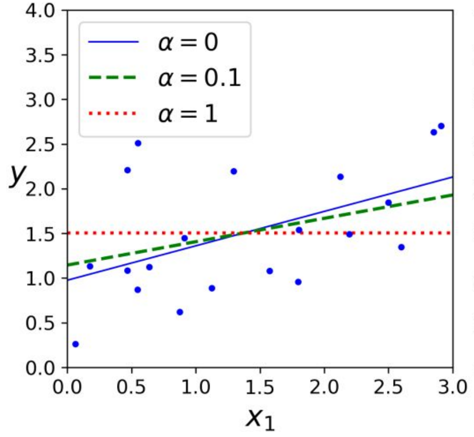
Lasso Regression: Using the L_1-norm

Regularization Parameter (λ)

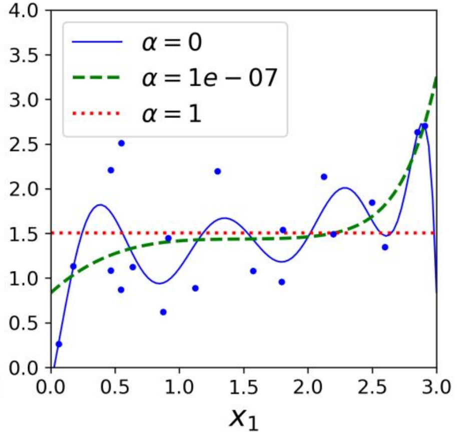
An important observation about Lasso Regression:

- As λ grows, some of the β will be driven to zero.
- This means that the model that it learns treats some features as irrelevant.
- Hence, it performs some feature selection too.

Linear Regression



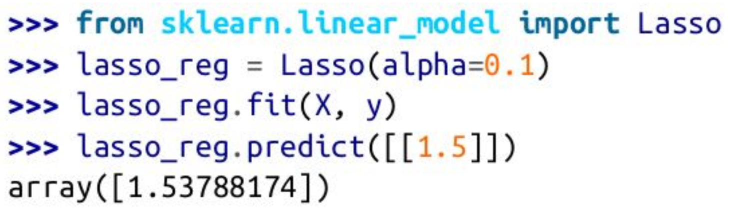
Polynomial Regression (degree=10)



Lasso Regression: Using the L_1-norm

Python Implementation

- scikit-learn has a class for this, called `Lasso`.
- Or you can use `SGDRegressor` with `penalty="l1"`.
- They both refer λ to as `alpha`!
- Scaling of feature values is usually advised.
- In case of polynomial regression, the data is first expanded using `PolynomialFeatures(degree=3)` , then it is scaled using `StandardScaler`, and finally the regularization is applied.



```
>>> from sklearn.linear_model import Lasso
>>> lasso_reg = Lasso(alpha=0.1)
>>> lasso_reg.fit(X, y)
>>> lasso_reg.predict([[1.5]])
array([1.53788174])
```

We could instead use `SGDRegressor(penalty=”l1”)`

Ridge Regression: Using the L$_2$-norm

We penalize by the L$_2$-norm of β, which is simply the sum of the squares of the coefficients, i.e.

$$\sum_{j=1}^{n} \beta_j^2$$

Minor point: we don't penalize β_0, which is why j starts at 1.

Note that the L$_2$-norm is the square root of the sum of squares.

So Ridge Regression finds the β that minimizes

$$J(X, y, \beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta\left(x^{(i)}\right) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^{n} \beta_j^2$$

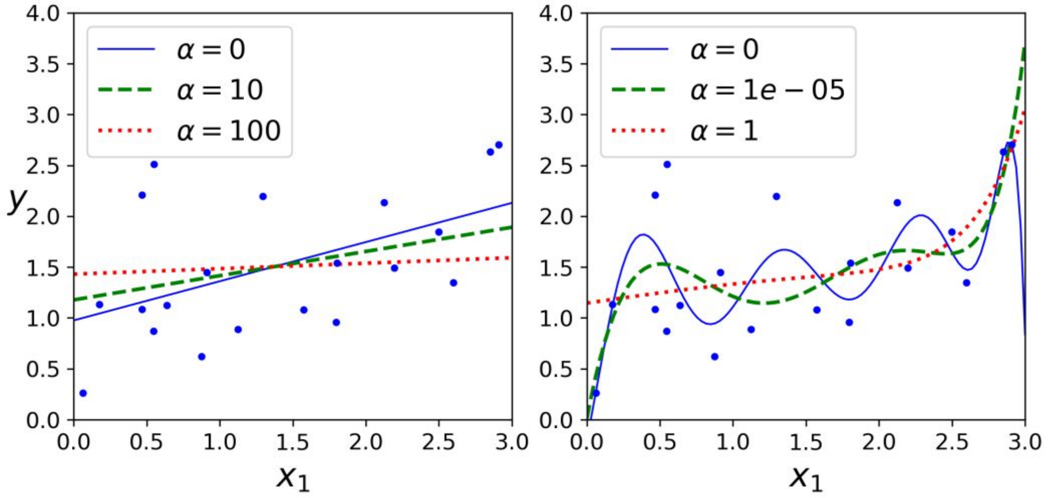
λ is called the 'regularization parameter'.

Ridge Regression: Using the L_2-norm

We penalize by the L_2-norm of β, which is simply the sum of the squares of the coefficients, i.e.

$$\sum_{j=1}^{n} \beta_j^2$$

Minor point: we don't penalize β_0, which is why j starts at 1.



Ridge Regression: Using the L$_2$-norm

Implementing Ridge Regression

There is an equivalent to the Normal Equation (solved, e.g., by Cholesky decomposition).

- Take the gradient, set it equal to zero, and solve for β (details unimportant)

$$\beta = \left(X^T X + \lambda I\right)^{-1} X^T y$$

- In the above equation, we chose not to penalize β_0, we want a zero in the top left, so I is not really the identity matrix.
- Also, you don't need to implement this with the pseudo-inverse. It's possible to prove that, provided $\lambda > 0$, then $(X^T X + \lambda I)$ will be invertible.

Ridge Regression: Using the L_2-norm

Implementing Ridge Regression

Alternatively, use Gradient Descent:

- The update rule for β_j for all *j* except *j*=0 becomes:

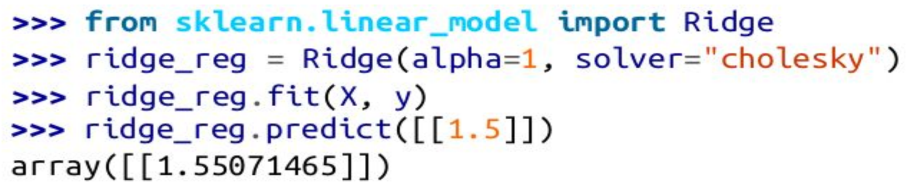
$$\begin{aligned}\beta_j &= \beta_j - \alpha\left(\frac{1}{m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right) \times x_j^{(i)} + \frac{\lambda}{m}\beta_j\right)\\ &= \beta_j\left(1 - \alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right) \times x_j^{(i)}\end{aligned}$$

This helps to show why this shrinks β_j.

Ridge Regression: Using the L_2-norm

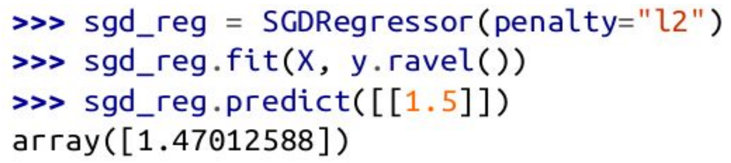
Python Implementation

- In scikit-learn, there is a special class for this, called `Ridge`.
- You can set its `solver` parameter to choose different methods, or leave it as default `auto`.
- Or you can use `SGDRegressor` with `penalty="l2"`.
- Scaling of feature values is usually advised.



```
>>> from sklearn.linear_model import Ridge
>>> ridge_reg = Ridge(alpha=1, solver="cholesky")
>>> ridge_reg.fit(X, y)
>>> ridge_reg.predict([[1.5]])
array([[1.55071465]])
```

Alternatively, using `SGDRegressor`



```
>>> sgd_reg = SGDRegressor(penalty="l2")
>>> sgd_reg.fit(X, y.ravel())
>>> sgd_reg.predict([[1.5]])
array([1.47012588])
```

Ridge Regression: Using the L_2-norm

Both Lasso and Ridge Regression shrink the values of the coefficients.

- But, as we mentioned, Lasso Regression may additionally result in coefficients being set to zero.
- This does not happen with Ridge Regression.

Roughly speaking, Lasso Regression shrinks the coefficients by approximately the same constant amount (unless they are so small that they get shrunk to zero),

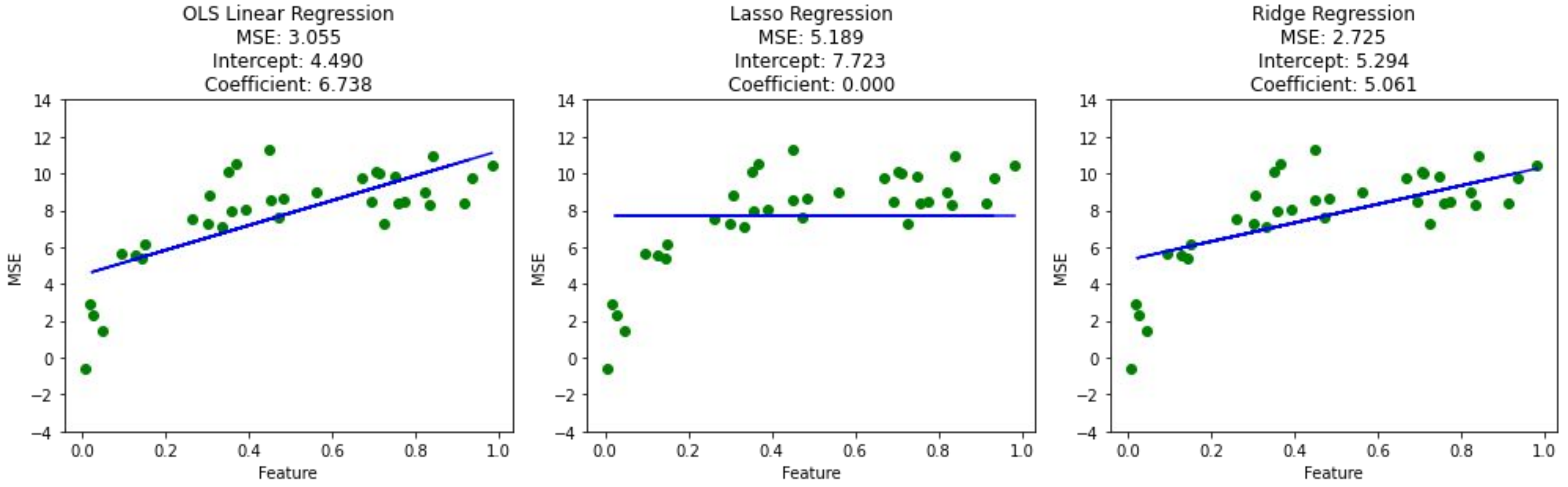
whereas,

Ridge Regression shrinks the coefficients by approximately the same proportion.

Lasso and Ridge Regression

Illustrating the Effects of Lasso and Ridge Regression

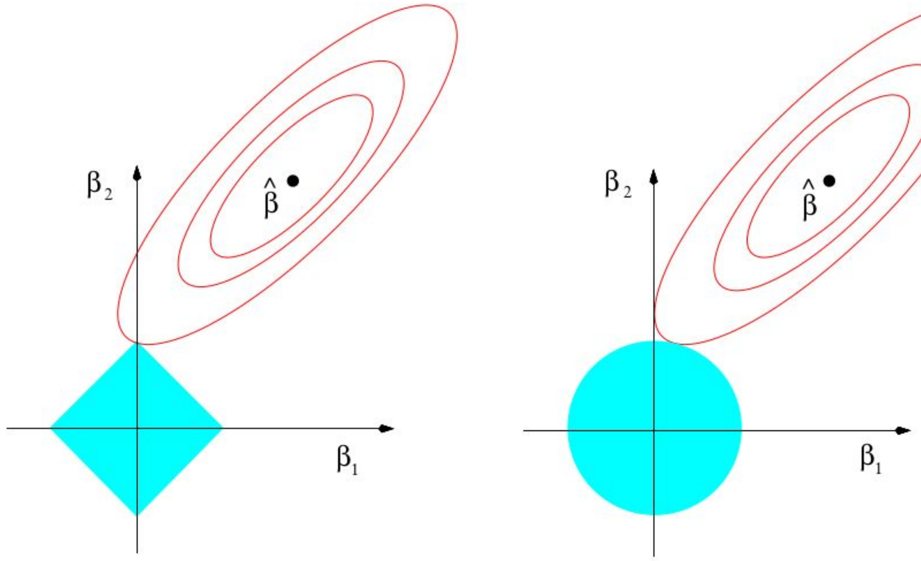
- We generate a random, non-linear dataset.
- Then we fit an unregularized linear model and two regularized models (Lasso and Ridge).



Lasso and Ridge Regression

Illustrating the Effects of Lasso and Ridge Regression

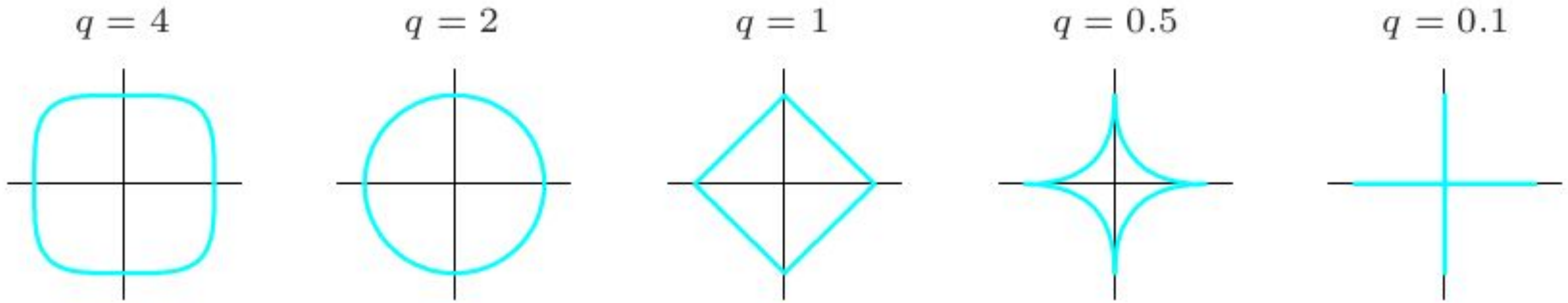
Shown are contours of the error and constraint functions. The solid areas are the constraint regions $|\beta_1| + |\beta_2| \leq t$ and $\beta_1^2 + \beta_2^2 \leq t^2$, respectively, while the red ellipses are the contours of the least squares error function.



Lasso and Ridge Regression

Generalizing Regularization

$$J(X, y, \beta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\beta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} |\beta_j|^q$$



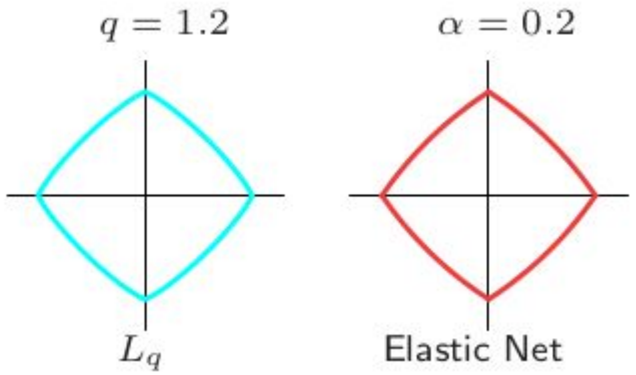
Elastic Net Regression

Combining Lasso and Ridge Regression

For completeness, we mention Elastic Net, which combines Lasso and Ridge regularization, with yet another hyperparameter to control the balance between the two.

$$\lambda \sum_{j=1}^{n} \left(\alpha \beta_j^2 + (1-\alpha)|\beta_j| \right)$$

```
>>> from sklearn.linear_model import ElasticNet
>>> elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
>>> elastic_net.fit(X, y)
>>> elastic_net.predict([[1.5]])
array([1.54333232])
```



Next lecture

Logistic Regression

19th September 2023