Linear Regression Models

Presented by Yasin Ceran

Table of Contents

1 Ordinary Least Squares

2 Ridge Regression

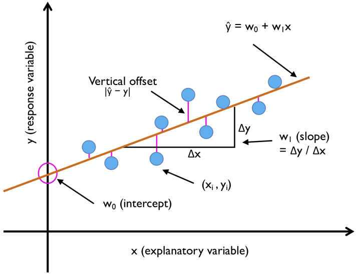
3 Lasso Regression

4 Understanding L1 and L2 Penalties

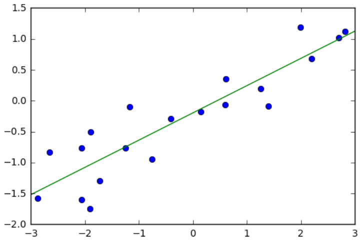
Linear Regression

$$\hat{y} = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{p} w_i x_i + b$$

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} (w^T\mathbf{x}_i + b - y_i)^2$$



Source: Python Machine Learning by S. Raschka



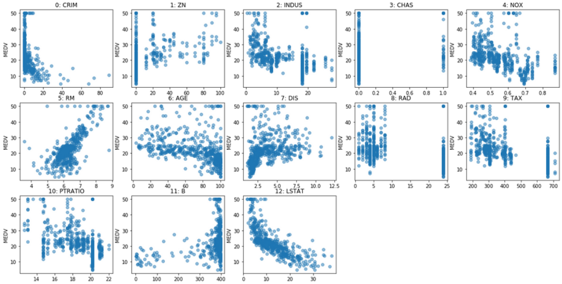
Source: Introduction to Machine Learning with Python by A. Muller

Training Linear Regression

- To train a Linear Regression model, we need to find the value of $\mathbf{w}$ that minimizes the RMSE.
- Normal Equation: $\hat{\mathbf{w}} = (\mathbf{x}^T\mathbf{x})^{-1}\mathbf{x}^T\mathbf{y}$ This approach may be inefficient when data size gets bigger and may not work if $(\mathbf{x}^T\mathbf{x})$ is not invertible.
- To estimate the model coefficients, you can use both Scikitlearn's 'LinearRegression' or 'SGDRegressor' classes.

```

from sklearn.linear_model import LinearRegression

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)
lr = LinerRegression().fit(X_train, y_train)
```

Exploring the Boston Housing Dataset



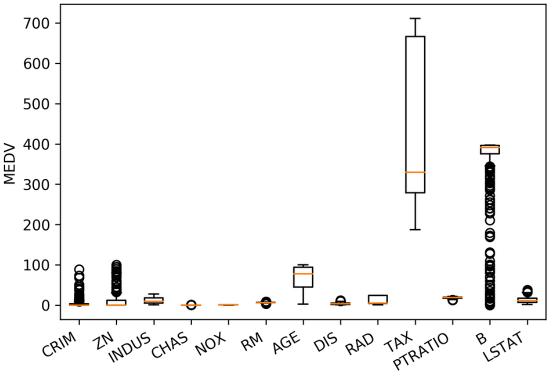

```
print(X.shape)
print(y.shape)
```

```
(506, 13)
(506,)
```

Boston Housing Dataset-Boxplot

```
plt.boxplot(X)
plt.xticks(np.arange(1, X.shape[1] + 1), boston.feature_names, rotation=30, ha="right")
plt.ylabel("MEDV")
```

```
<matplotlib.text.Text at 0x7f580303eac8>
```



Implementation of OLS with ScikitLearn

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split, cross_val_score
import numpy as np

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)
lr = LinerRegression()
np.mean(cross_val_score(lr, X_train, y_train, cv=10))



0.717
```

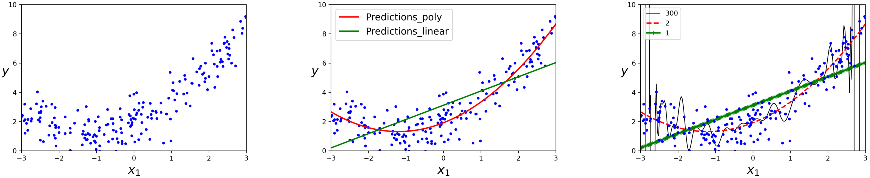
Coefficient of Determination R^2

- Our target variable values $y_1, y_2, \ldots, y_n$ (collectively vector $\mathbf{y} = [y_1, y_2, \ldots, y_n]^T$), each associated with a fitted (or modeled, or predicted) value $\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_n$ (known as a vector $\widehat{\mathbf{y}}$).
- Define the residuals as $e_i = y_i - \hat{y}_i$ (forming a vector e).
- $\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$ is the mean of the target variable.
- The total sum of squares (proportional to the variance of the data): $SS_{tot} = \sum_{i=0}^{n-1}(y_i - \bar{y})^2$
- The sum of squares of residuals, also called the residual sum of squares: $SS_{res} = \sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 = \sum_{i=0}^{n-1}(e_i)^2$
- The most general definition of the coefficient of determination is: $R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} = 1 - \frac{SS_{res}}{SS_{tot}}$

Can be negative for biased estimators - or the test set!

Polynomial Features

What happens when our data looks more complicated than a straight line, for example, looks like a nonlinear curve:



We use Scikit-Learn's '`PolynomialFeatures`' class to transform our training data, adding the polynomial of each feature in the training set as a new feature.

Adding Features

- `PolynomialFeatures()` adds polynomials and interactions.
- Transformer interface like scalers etc.
- Create polynomial algorithms with `make_pipeline`.

```
from sklearn.preprocessing import PolynomialFeatures, scale
poly = PolynomialFeatures(include_bias=False)
X_poly = poly.fit_transform(scale(X))
print(X_poly.shape)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y)

(506, 104)

np.mean(cross_val_score(LinearRegression(), X_train, y_train, cv=10))

0.74
```

Plotting Coefficient Values of LR

```
lr = LinearRegression().fit(X_train, y_train)
plt.scatter(range(X_poly.shape[1]),
            lr.coef_, c=np.sign(lr.coef_), cmap="bwr_r")
```

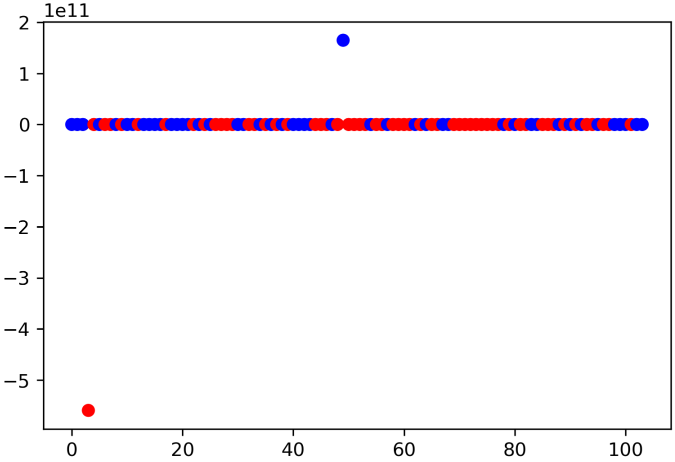


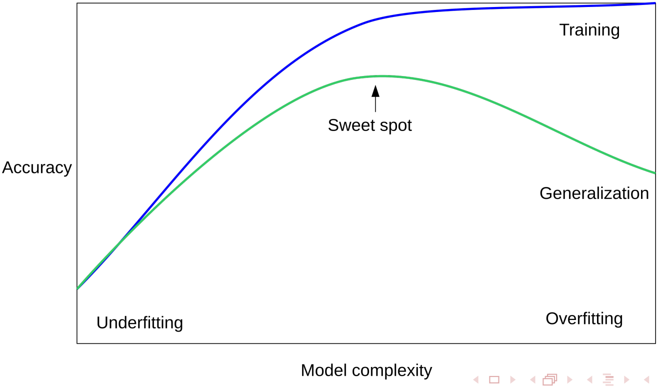
Table of Contents

1. Ordinary Least Squares
2. Ridge Regression
3. Lasso Regression
4. Understanding L1 and L2 Penalties

Ridge Regression

$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{p} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||^2$$

Tuning parameter: α.

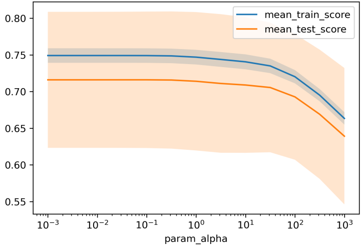


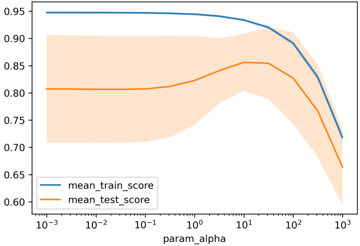
Ridge Trained on Boston Dataset

```
from sklearn.linear_model import Ridge
from sklearn.model_selection import cross_val_score, train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

np.mean(cross_val_score(LinearRegression(), X_train, y_train, cv=10))

0.716

np.mean(cross_val_score(Ridge(), X_train, y_train, cv=10))

0.714
```

Grid Search with Ridge

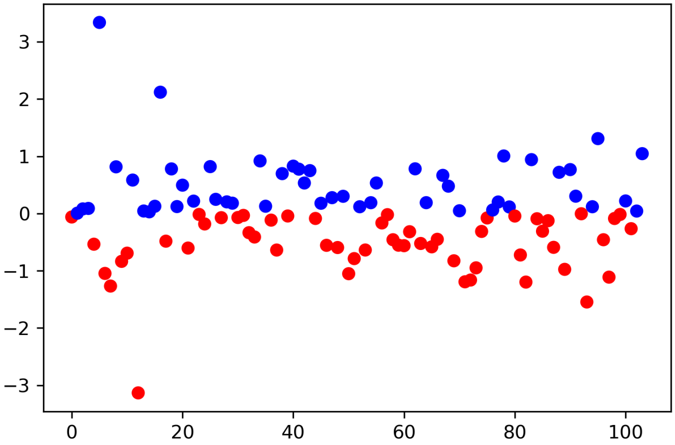
```python
from sklearn.model_selection import GridSearchCV
param_grid = {'alpha': np.logspace(-3, 3, 13)}

grid = GridSearchCV(Ridge(), param_grid, cv=10, return_train_score=True)
grid_no_poly=grid.fit(X_train, y_train)
grid_poly=grid.fit(X_poly_train, y_train)
```





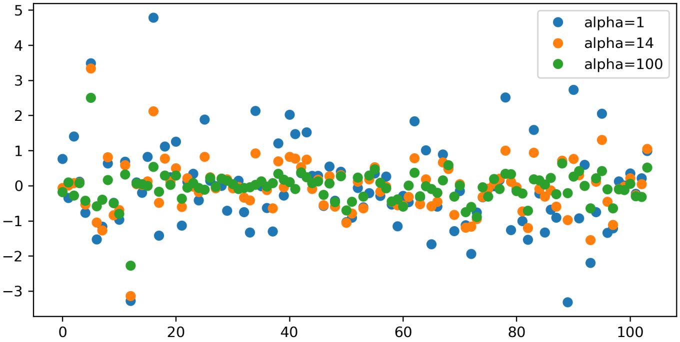
Plotting Coefficient Values of Ridge

```python
ridge_poly = grid_poly.best_estimator_
plt.scatter(range(X_poly.shape[1]), ridge_poly.coef_, c=np.sign(ridge_poly.coef_), cmap="bwr_r")
```



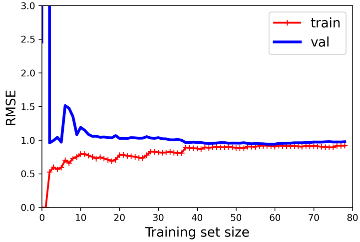
Impact of α Parameter Value on Ridge Coefficients

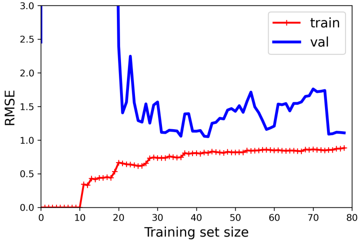
```
ridge100 = Ridge(alpha=100).fit(X_train, y_train)
ridge1 = Ridge(alpha=1).fit(X_train, y_train)
plt.figure(figsize=(8, 4))
plt.plot(ridge1.coef_, 'o', label="alpha=1")
plt.plot(ridge.coef_, 'o', label="alpha=14")
plt.plot(ridge100.coef_, 'o', label="alpha=100")
plt.legend()
```



Learning Curves

- How do you decide how complex your model should be? And how can you tell if your model is overfitting or underfitting the data?
- You may use crossvalidation
- You may also use learning curves: plots of model performance as a function of training data size.





Regularization and Learning Curves

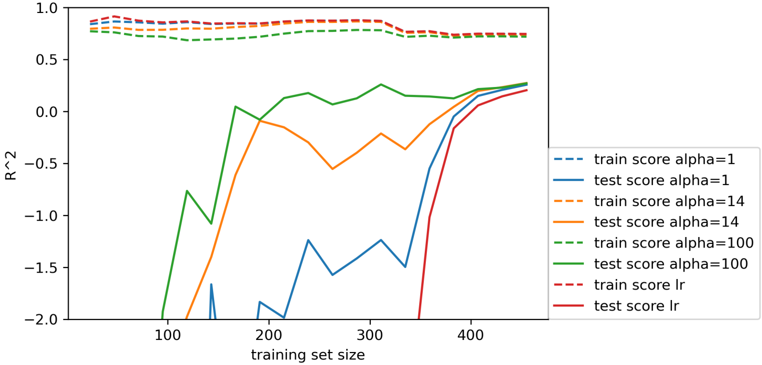


Table of Contents

1. Ordinary Least Squares
2. Ridge Regression
3. Lasso Regression
4. Understanding L1 and L2 Penalties

Lasso Regression

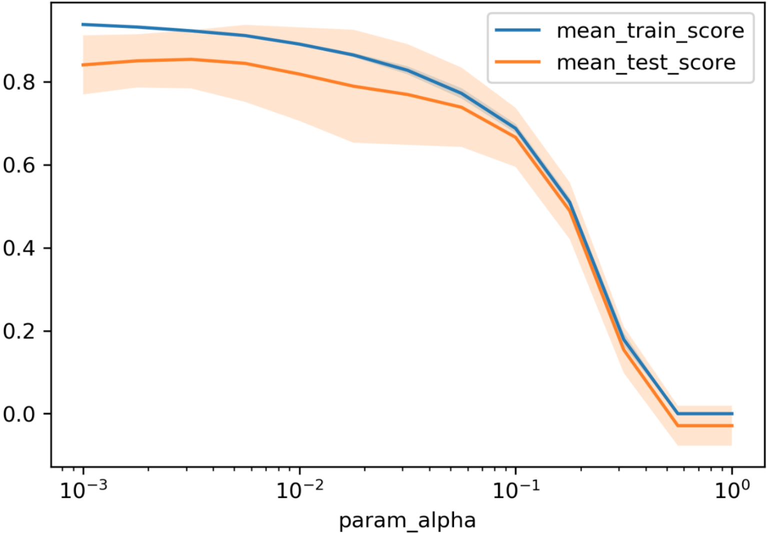
$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||_1$$

- Shrinks w towards zero like Ridge
- Sets some w exactly to zero - automatic feature selection!

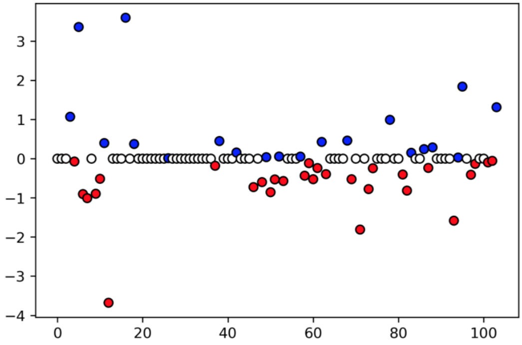
Grid Search for Lasso

```
param_grid = {'alpha': np.logspace(-3, 0, 13)}
print(param_grid)

{'alpha': array([ 0.001,  0.003, 0.01, 0.032, 0.1, 0.316, 1., 3.162,
                 10., 31.623, 100., 316.228, 1000.])}


grid = GridSearchCV(Lasso(normalize=True), param_grid, cv=10)
grid.fit(X_train, y_train)
print(grid.best_params_)
print(grid.best_score_)


{'alpha': 0.001}
0.837
```

Grid Search Plot for Lasso



Plotting Coefficient Values of Lasso




```
print(X_poly.shape)
np.sum(lasso.coef_ != 0)
```

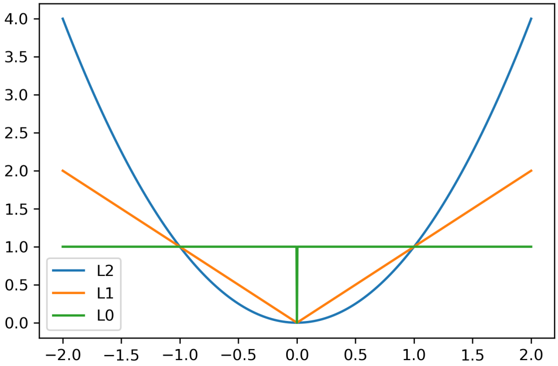
```
(506, 104)
64
```

Table of Contents

1. Ordinary Least Squares
2. Ridge Regression
3. Lasso Regression
4. Understanding L1 and L2 Penalties

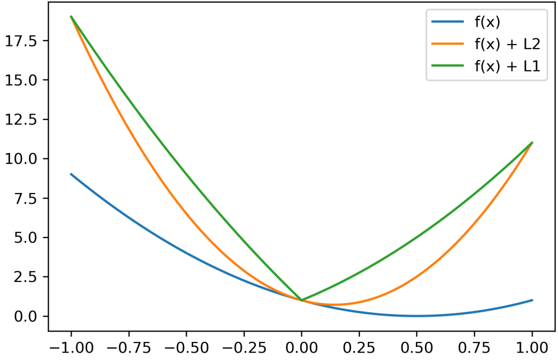
Understanding L1 and L2 penalties

- $\ell_2(w) = \sqrt{\sum_i w_i^2}$
- $\ell_1(w) = \sum_i |w_i|$
- $\ell_0(w) = \sum_i 1_{w_i != 0}$

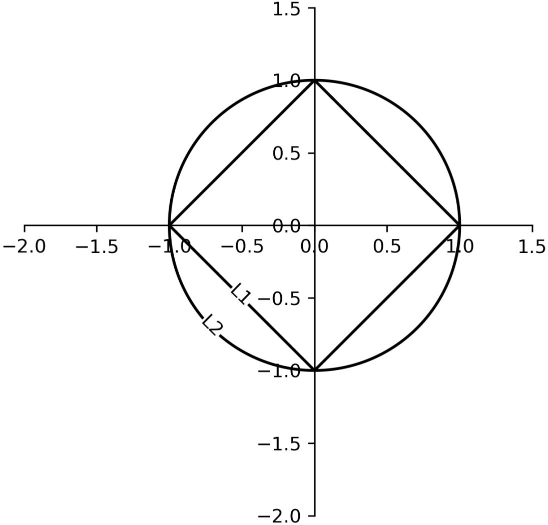


Understanding L1 and L2 penalties

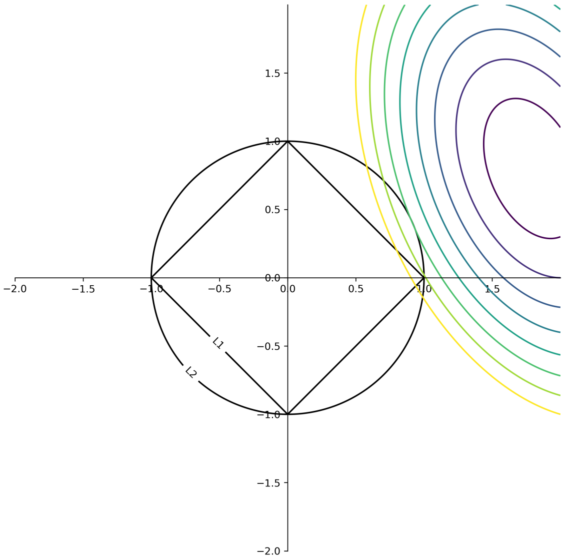
- $f(x) = (2x - 1)^2$
- $f(x) + L2 = (2x - 1)^2 + \alpha x^2$
- $f(x) + L1 = (2x - 1)^2 + \alpha |x|$



Understanding L1 and L2 penalties



Understanding L1 and L2 penalties



Summary

- Linear Regression
- Regularization:

$$min_{f \in F} \sum_{i=1}^{n} L(f(\mathbf{x}_i), y_i) + \alpha R(f)$$

- Ridge Regression uses l_2 norm
- Lasso Regression uses l_1 norm