W4995 Applied Machine Learning

Linear models for Regression

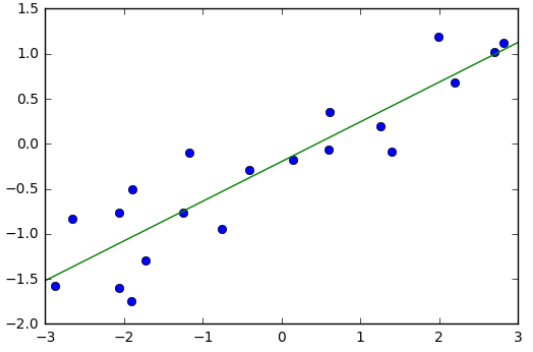
02/11/19

Andreas C. Müller

(Adapted and modified for CC 6021236 @ PCC/Ciencias/UCV by
Eugenio Scalise, September 2019)

Linear Models

Linear Models for Regression



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

Ordinary Least Squares

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

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

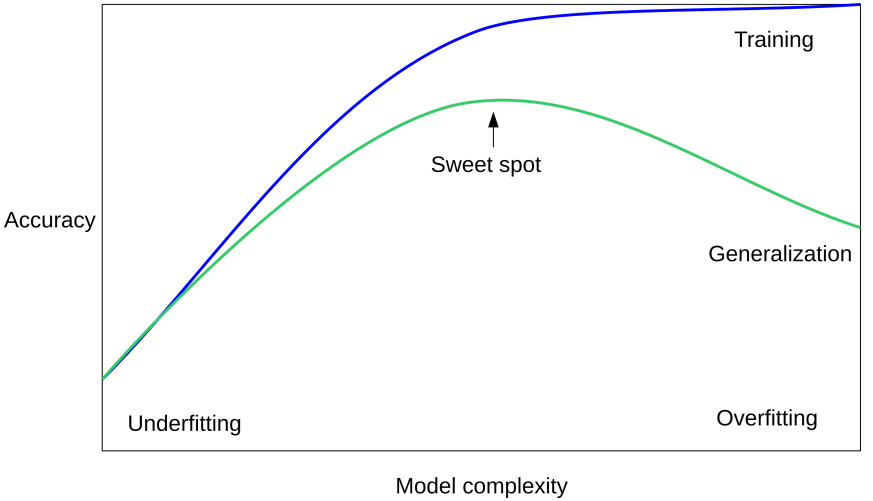
Unique solution if $\mathbf{X} = (\mathbf{x}_1, \ldots \mathbf{x}_n)^T$ has full column rank.

Ridge Regression

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

- Not only do we want to fit the training data well, we also want w to have a small squared l2 norm or squared euclidean norm.
- The idea here is that we're pushing the coefficients towards zero. This constrains the model to be more simple.
- Tuning parameter alpha.
- Two terms: (1) the objective function of the model and (2) the penalty or regularization term here that wants w to have small norm, and that doesn't depend on the data.

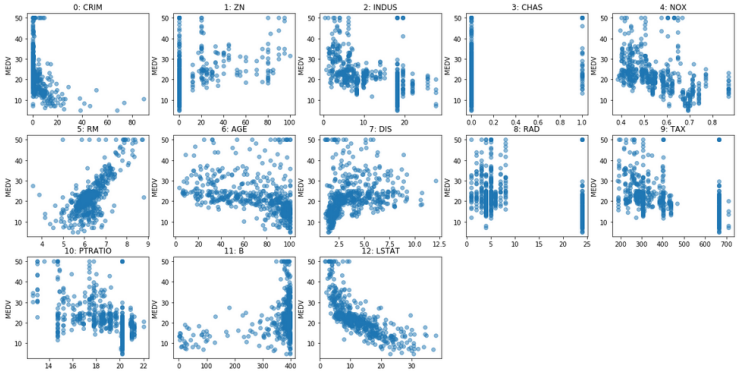
Reminder on model complexity



alpha high => we restrict the model (left side of the graph)

alpha small => we allow the model to fit the data more (right side of the

Boston Housing Dataset

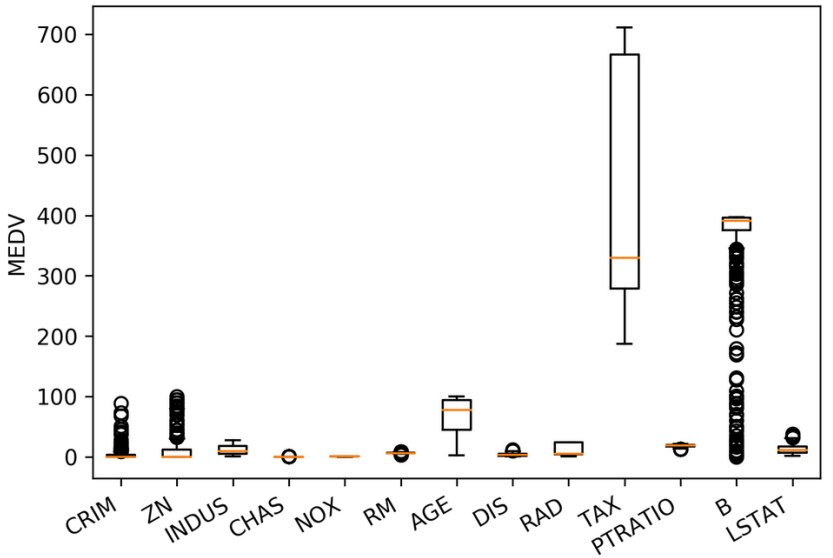



```
print(X.shape)
print(y.shape)
```

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

```
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>
```



```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)

np.mean(cross_val_score(LinearRegression(), X_train, y_train, cv=10))
# 10-fold cross validation
```

```
0.717
```

```
np.mean(cross_val_score(Ridge(), X_train, y_train, cv=10)) # alpha=1
```

```
0.715
```

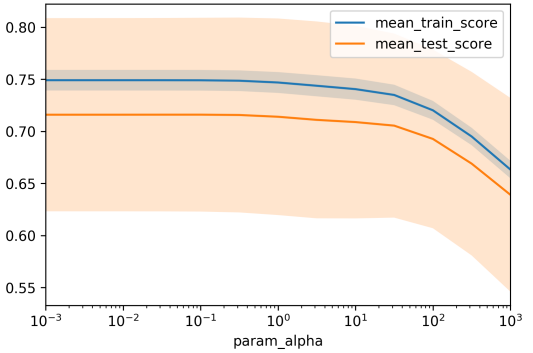
Scaling

```
from sklearn.linear_model import Ridge
from sklearn.preprocessing import StandardScaler
X, y = boston.data, boston.target
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=0)

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
ridge = Ridge().fit(X_train_scaled, y_train)

X_test_scaled = scaler.transform(X_test)
ridge.score(X_test_scaled, y_test)
```

```
from sklearn.model_selection import GridSearchCV
param_grid = {'alpha': np.logspace(-3, 3, 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(Ridge(), param_grid, cv=10)
grid.fit(X_train, y_train)
```



Adding features

```
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)
```

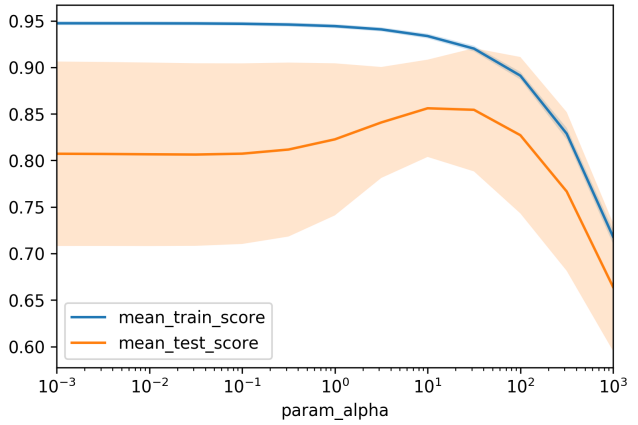
```
(506, 104)
```

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

```
0.74
```

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

```
0.76
```

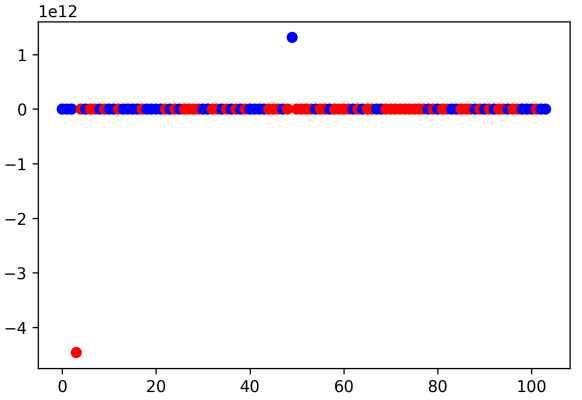



```
print(grid.best_params_)
print(grid.best_score_)
```

```
{'alpha': 31.6}
0.83
```

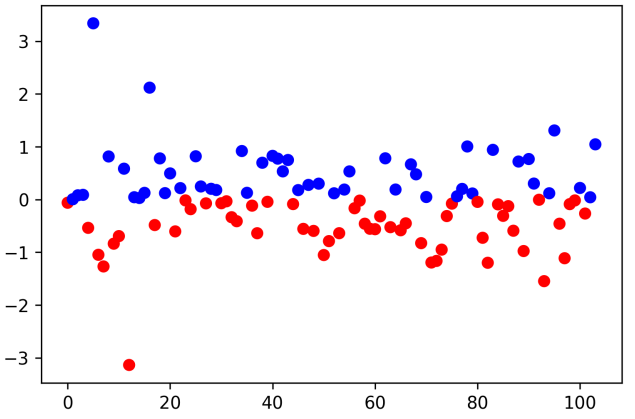
Plotting coefficient values (LR)

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

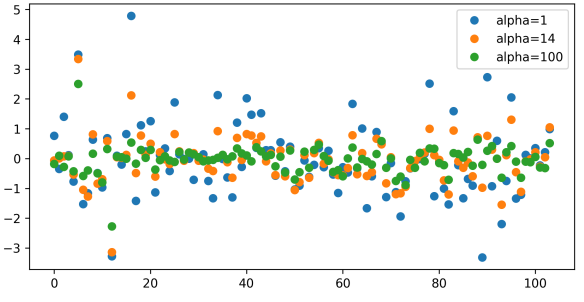


Ridge Coefficients

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



```
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()
# alpha (on average) pushes all the coefficients toward zero.
```



Lasso Regression

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

- Shrinks w towards zero like Ridge
- The L2 norm (Ridge) penalizes very large coefficients more, the L1 (Lasso) norm penalizes all coefficients equally.
- Sets some w exactly to zero - automatic feature selection!
- The coefficient of zero means it doesn't influence the prediction and so you can just drop it out of the model.
- This model does features selection together with prediction. Ideally what you would want is, let's say you want a model that does features selections.

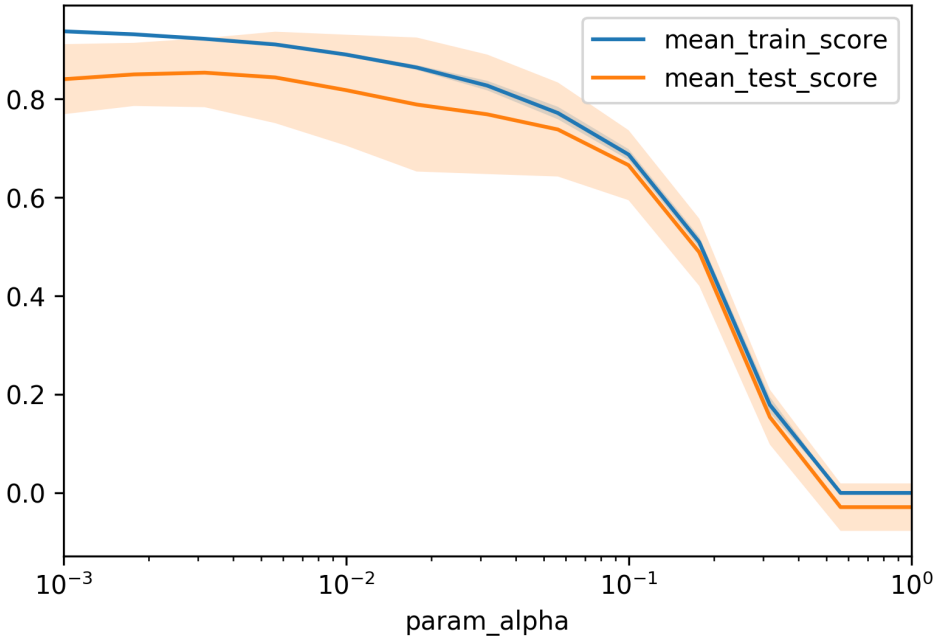
Grid-Search for Lasso

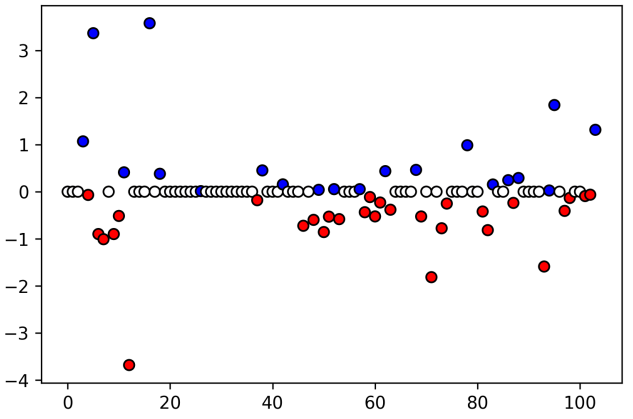
```
param_grid = {'alpha': np.logspace(-3, 3, 13)}
print(param_grid)
```

```
{'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
```


mean_train_score
mean_test_score
0.8
0.6
0.4
0.2
0.0
10^{-3}
10^{-2}
10^{-1}
10^{0}
param_alpha




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

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

Elastic Net

- Combines benefits of Ridge and Lasso
- two parameters to tune.

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} ||w^T \mathbf{x}_i + b - y_i||^2 + \alpha_1 ||w||_1 + \alpha_2 ||w||_2^2$$

- Generally, ridge helps generalization. So it's a good idea to have the ridge penalty in there, but also maybe if there are some features that are really not useful, the L1 penalty helps makes the same exactly zero.

Parametrization in scikit-learn

$$\min_{w \in \mathbb{R}^p, b \in \mathbb{R}} \sum_{i=1}^{n} ||w^T \mathbf{x}_i + b - y_i||^2 + \alpha\eta||w||_1 + \alpha(1-\eta)||w||_2^2$$

Where η is the relative amount of l1 penalty (`l1_ratio` in the code).

- In scikit-learn, you have a parameter alpha, which is the amount of regularization and then there's a parameter called l1_ratio, that says how much of the penalty should be L1 and L2. If you make this one, you have Lasso, if you make it zero, you have a Ridge.
- This actually works pretty well often.

Grid-searching ElasticNet

```
from sklearn.linear_model import ElasticNet
param_grid = {'alpha': np.logspace(-4, -1, 10),
              'l1_ratio': [0.01, .1, .5, .9, .98, 1]}

grid = GridSearchCV(ElasticNet(), param_grid, cv=10)
grid.fit(X_train, y_train)

print(grid.best_params_)
print(grid.best_score_)
```

```
{'alpha': 0.0001, 'l1_ratio': 0.01}
0.718
```

Analyzing grid-search results

```
import pandas as pd
res = pd.pivot_table(pd.DataFrame(grid.cv_results_),
    values='mean_test_score', index='param_alpha', columns='param_l1_ratio')
```