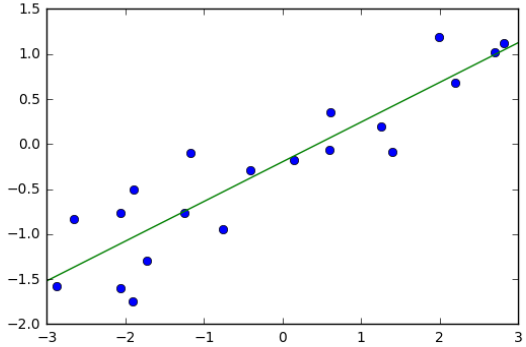
Applied Machine Learning

Linear models for Regression

Linear Models for Regression



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

Ordinary Least Squares

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

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

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

Ridge Regression

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

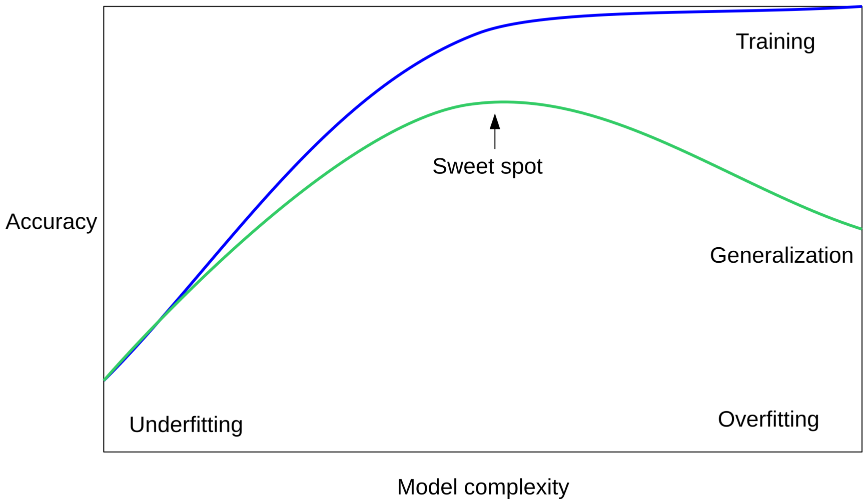
Always has a unique solution.

Tuning parameter: α.

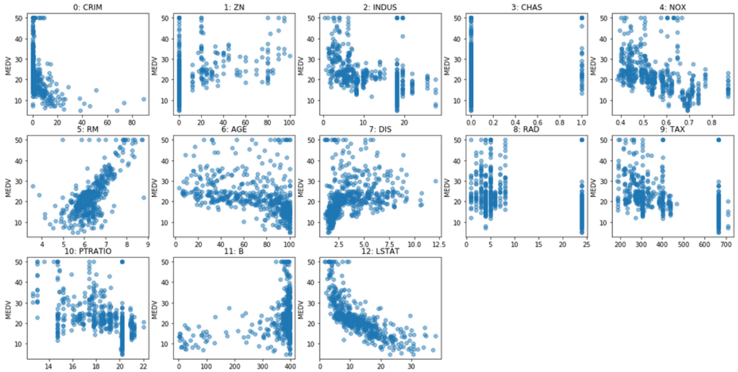
(regularized) Empirical Risk Minimization

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

Reminder on model complexity



Boston Housing Dataset

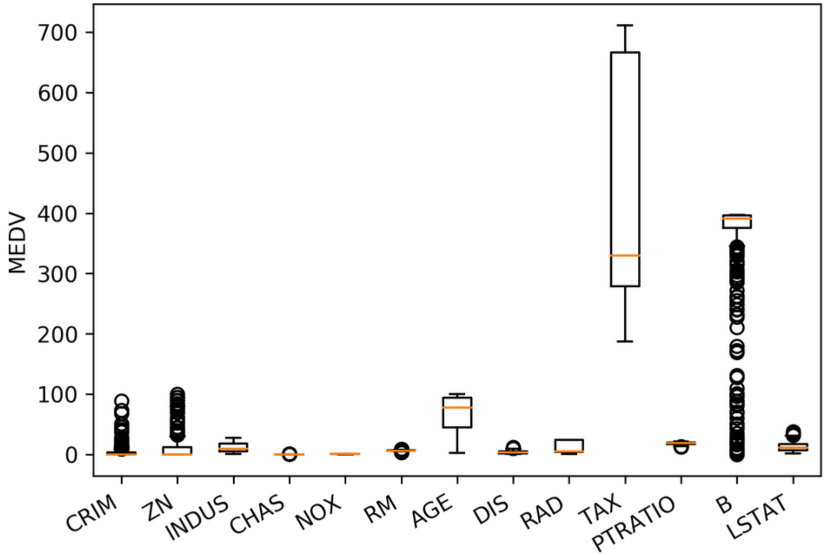



```
print(X.shape)
print(y.shape)
```

```
(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>
```



```python
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))
```

```
0.717
```

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

```
0.715
```

Coefficient of determination R^2

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1} (y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1} (y_i - \bar{y})^2}$$

$$\bar{y} = \frac{1}{n} \sum_{i=0}^{n-1} y_i$$

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

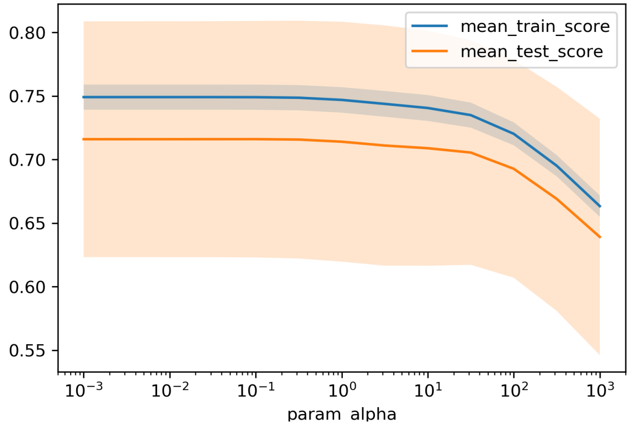
Scaling (if you want)

```
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)
```

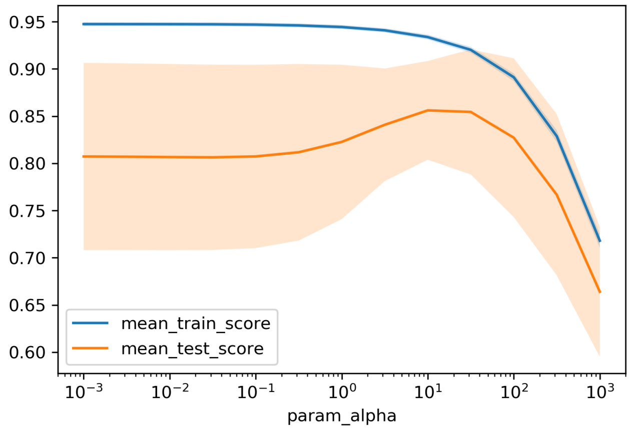
```
(506, 104)
```

```
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
```

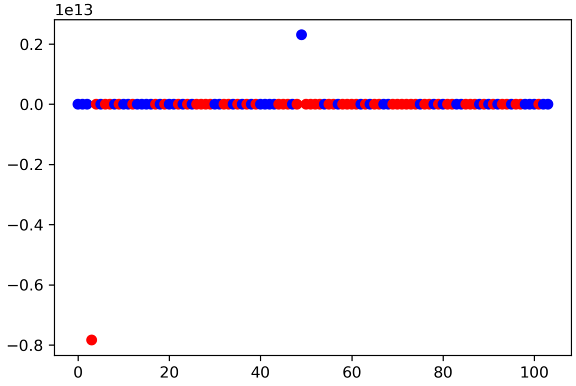



```
print(grid.best_params_)
print(grid.best_score_)
```

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

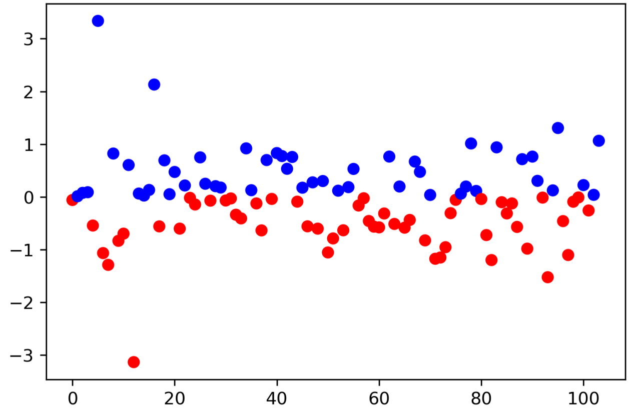
Plotting coefficient values (LR)

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

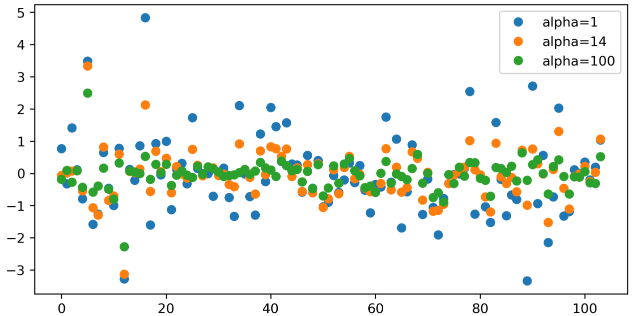


Ridge Coefficients

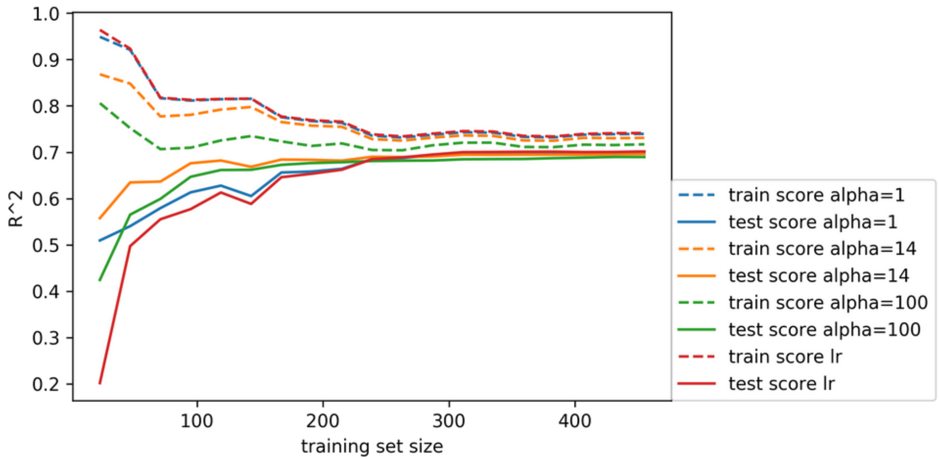
```
ridge = grid.best_estimator_
plt.scatter(range(X_poly.shape[1]), ridge.coef_,
            c=np.sign(ridge.coef_), cmap="bwr_r")
```



```
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()
```



Learning Curves



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!

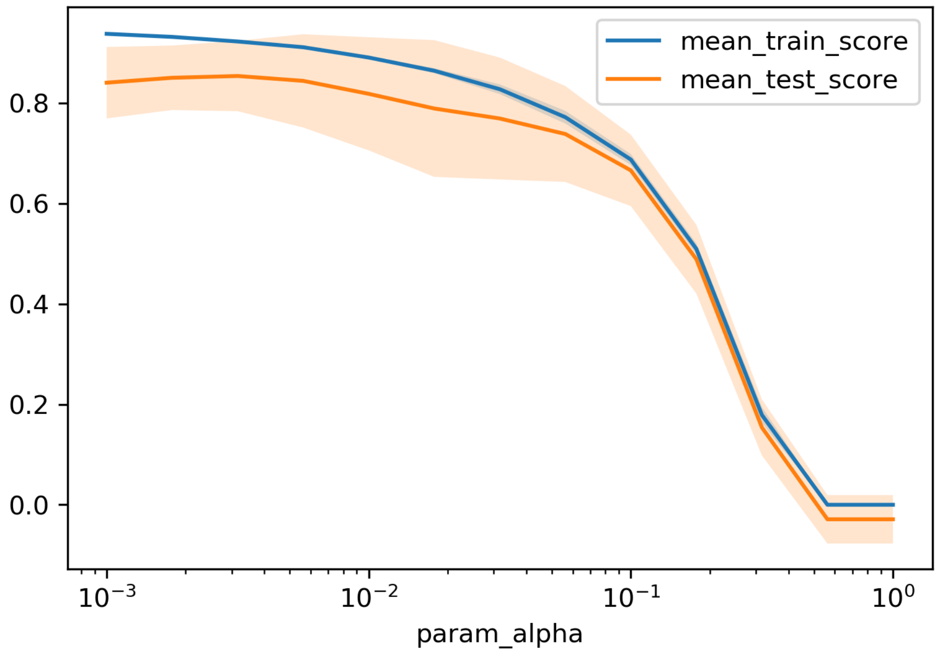
Grid-Search for Lasso

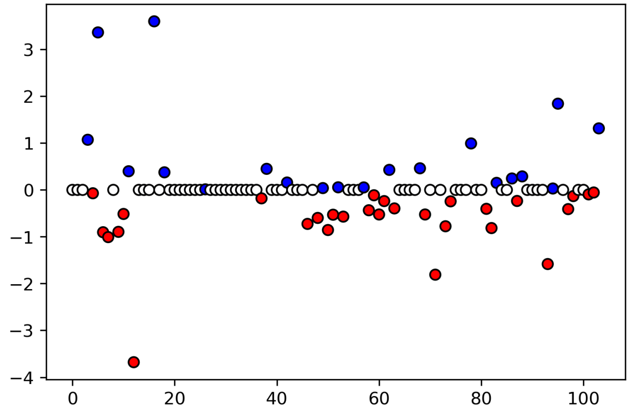
```
param_grid = {'alpha': np.logspace(-3, 0, 13)}
print(param_grid)
```

```
{'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
```


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.

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

Parametrization in scikit-learn

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

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

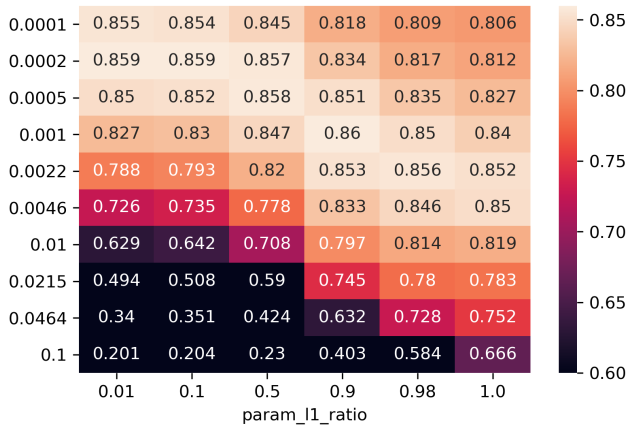
Grid-searching ElasticNet

```
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')
```



Questions ?