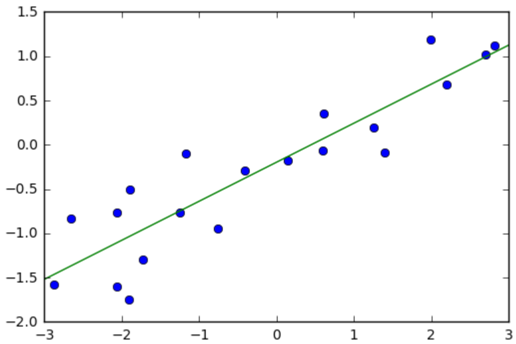
W4995 Applied Machine Learning

Linear models

02/01/17

Andreas Müller

Linear Models for Regression



$$\hat{y} = w^T \mathbf{x} + b = \sum_i w_i x_i + b$$

Linear Regression
Ordinary Least Squares

$$\hat{y} = w^T \mathbf{x} + b = \sum_i w_i x_i + b$$

$$min_{w \in \mathbb{R}^n} \sum_i ||w^T \mathbf{x}_i - y_i||^2$$

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

Ridge Regression

$$min_{w \in \mathbb{R}^n} \sum_i ||w^T x_i - y_i||^2 + \alpha||w||^2$$

Always has a unique solution.
Has tuning parameter alpha

(regularized) Empirical Risk Minimization

$$min_{f \in F} \sum_i \underbrace{L(f(\mathbf{x}_i), y_i)}_{\text{Data fitting}} + \underbrace{\alpha R(f)}_{\text{Regularization}}$$

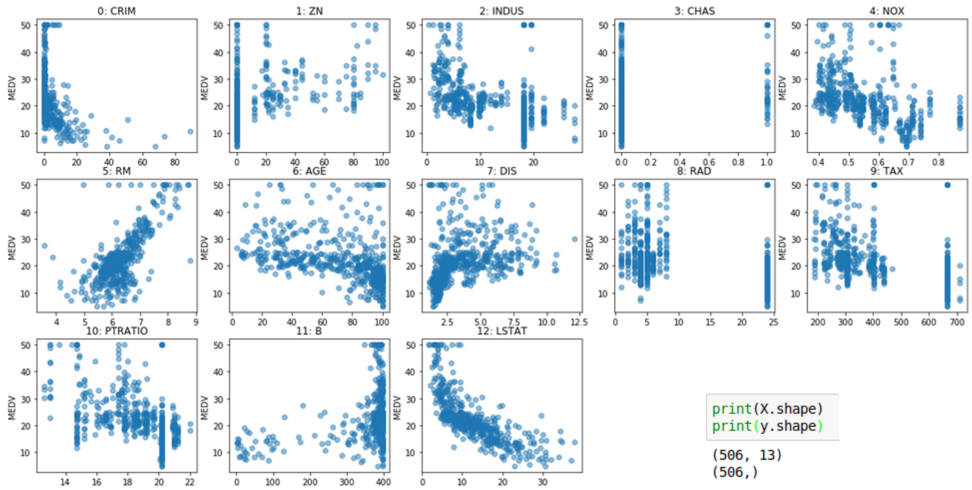
This leads us to one of the core principles of machine learning, which I will only sketch here, but which is the essence of all the machine learning theory.

If we can find a model that fits the training data well, and that is “simple” then we are guaranteed that it will work well on new data.

The machine learning problem can then be formalized as minimizing the error on the training set, say the squared error, while constraining the model to be simple.

This is called “regularized empirical risk minimization” and that’s basically what all machine learning algorithms do, with different choices of the domain of f, and different choices of the regularizer R.

Boston Housing Dataset




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

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

```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

```python
np.mean(cross_val_score(LinearRegression(), X_train, y_train, cv=5))
```

```
0.69844164175864798
```

```python
np.mean(cross_val_score(Ridge(), X_train, y_train, cv=5))
```

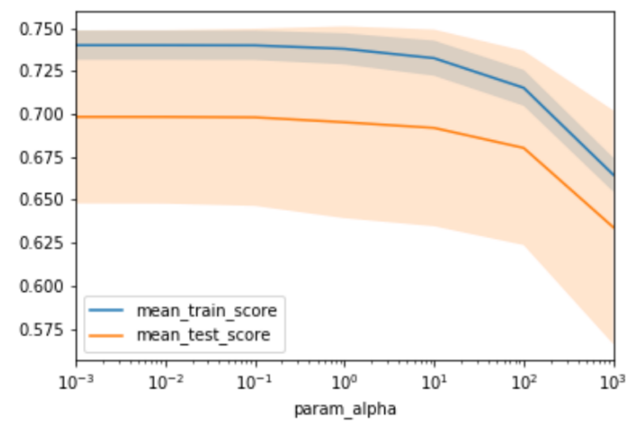
```
0.69530615273465046
```

Score is always R^2 for regression!

```
from sklearn.model_selection import GridSearchCV
param_grid = {'alpha': np.logspace(-3, 3, 7)}
print(param_grid)
```

```
{'alpha': array([  1.00000000e-03,   1.00000000e-02,   1.00000000e-01,
         1.00000000e+00,   1.00000000e+01,   1.00000000e+02,
         1.00000000e+03])}
```

```
grid = GridSearchCV(Ridge(), param_grid, cv=5)
grid.fit(X_train, y_train)
```



Adding features

```
from sklearn.preprocessing import PolynomialFeatures, scale
X_poly = PolynomialFeatures(include_bias=False).fit_transform(scale(X))
print(X_poly.shape)
X_train, X_test, y_train, y_test = train_test_split(X_poly, y, random_state=42)
```

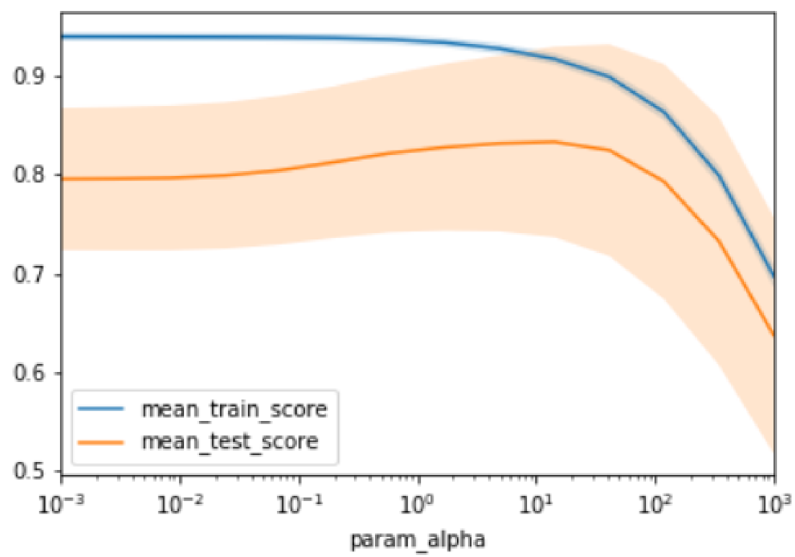
```
(506, 104)
```

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

```
0.79424725375805638
```

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

```
0.82530611400699261
```

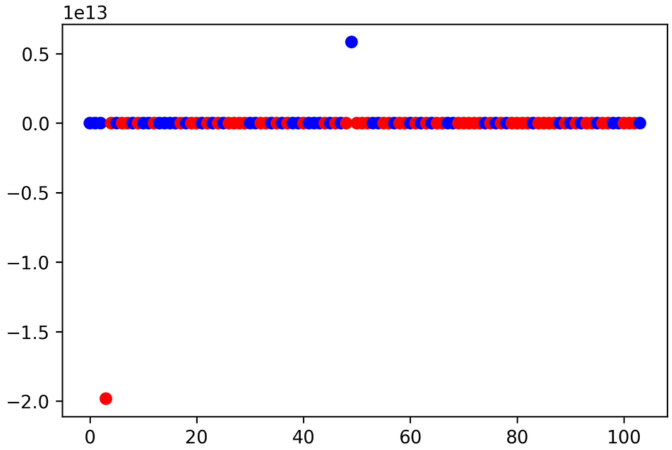



```python
print(grid.best_params_)
print(grid.best_score_)
```

```
{'alpha': 14.251026703029993}
0.833124993262
```

Plotting coefficient values (LR)

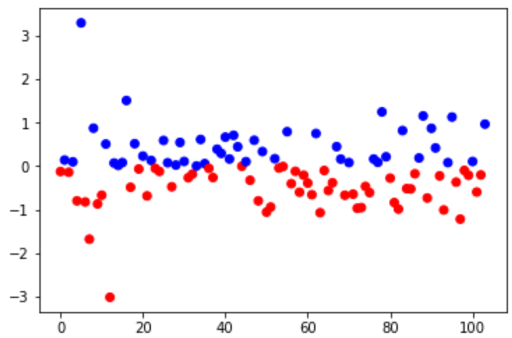
```
lr = LinearRegression().fit(X_train, y_train)
plt.scatter(range(X_poly.shape[1]), lr.coef_, c=np.sign(lr.coef_), cmap="bwr_r")
```



Ridge Coefficients

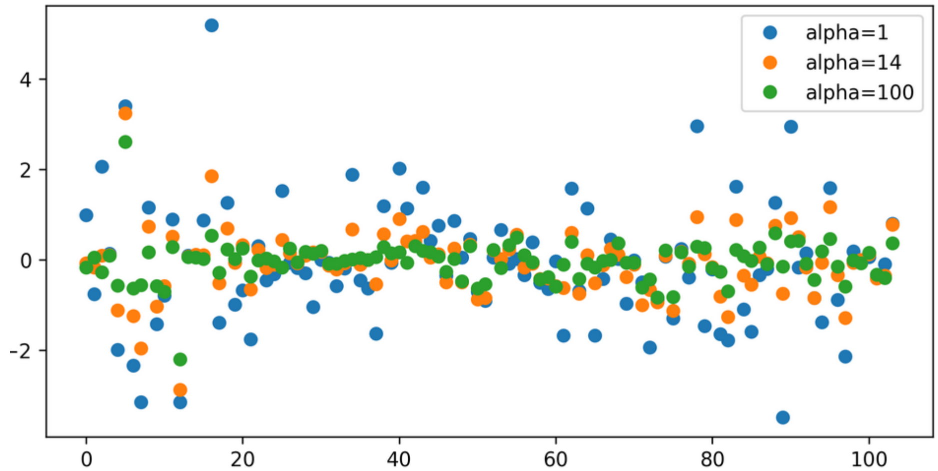
```
ridge = grid.best_estimator_
plt.scatter(range(X_poly.shape[1]), ridge.coef_, c=np.sign(ridge.coef_), cmap="bwr_r")
```

```
<matplotlib.collections.PathCollection at 0x7fda1025c780>
```

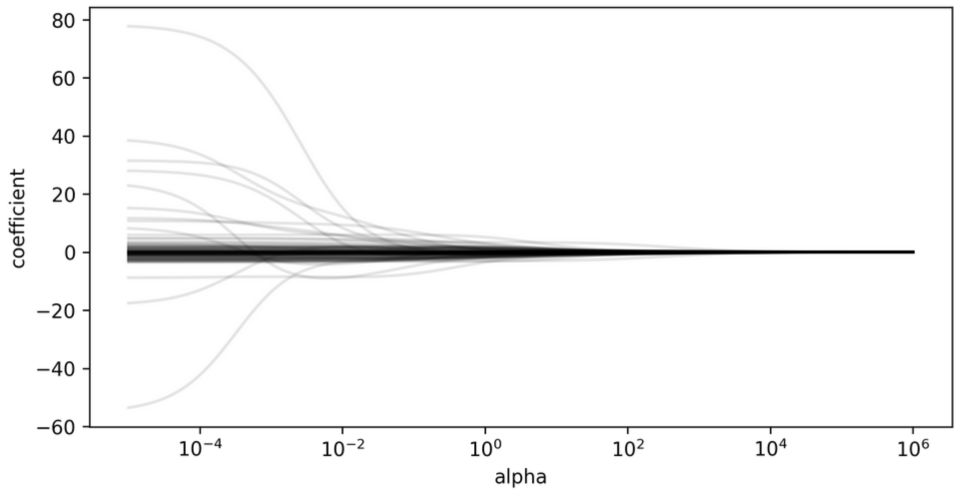


```
ridge100 = Ridge(alpha=100).fit(X_train, y_train)
ridge1 = Ridge(alpha=1).fit(X_train, y_train)
plt.figure(figsize=(8, 4))

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()
```



Coefficient Paths



Learning Curve

- FIXME!!!

Lasso Regression

$$min_{w \in \mathbb{R}^n} \sum_i ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||_1$$

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

Grid-Search for Lasso

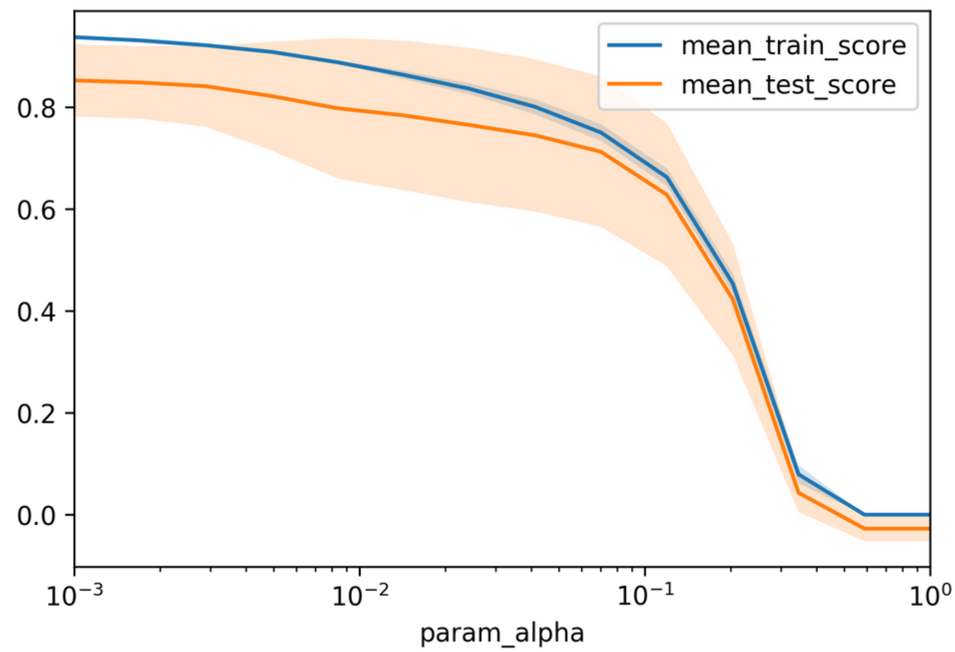
```
param_grid = {'alpha': np.logspace(-3, 0, 14)}
print(param_grid)
```

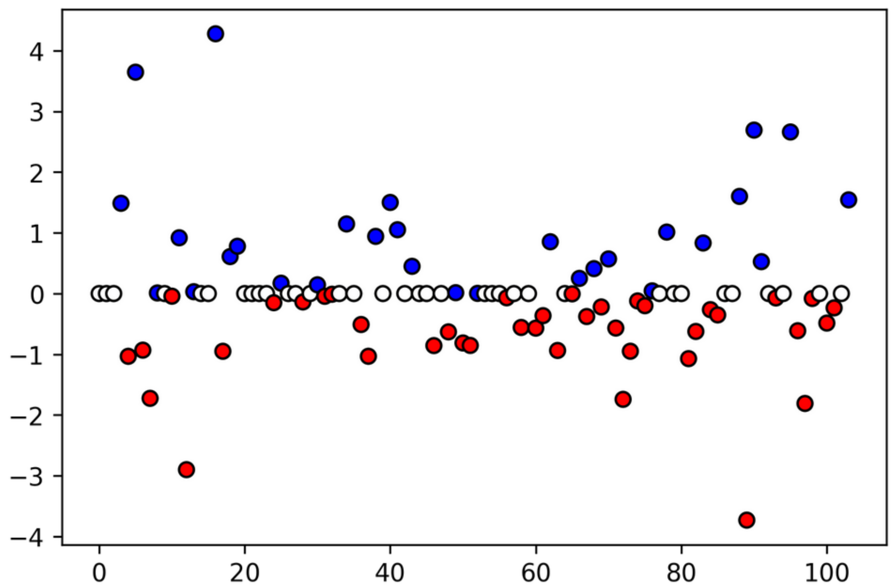
```
{'alpha': array([ 0.001     ,  0.00170125,  0.00289427,  0.00492388,  0.00837678,
        0.01425103,  0.02424462,  0.04124626,  0.07017038,  0.11937766,
        0.20309176,  0.34551073,  0.58780161,  1.        ])}
```

```
grid = GridSearchCV(Lasso(normalize=True, max_iter=1e6), param_grid, cv=10)
grid.fit(X_train, y_train)
```

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

```
{'alpha': 0.001}
0.852388809881
```


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



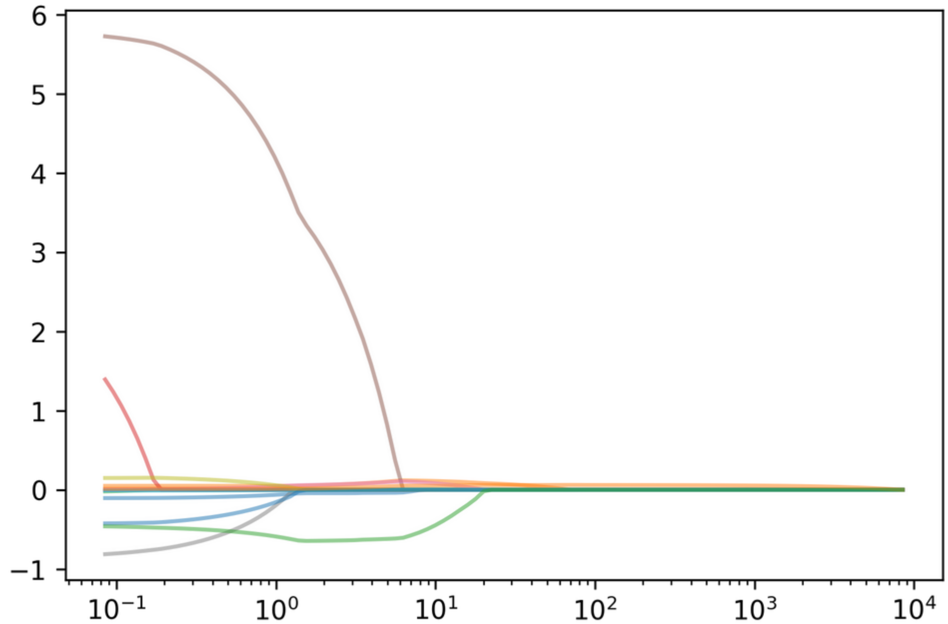
In [31]:
```
print(X_poly.shape)
np.sum(lasso.coef_ != 0)
```

```
(506, 104)
```

Out[31]:
```
69
```

```python
from sklearn.linear_model import lasso_path
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
alphas, coefs, _ = lasso_path(X_train, y_train, eps=0.00001)
```

```python
plt.plot(alphas, np.array(coefs).T, alpha=.5)
plt.xscale("log")
```

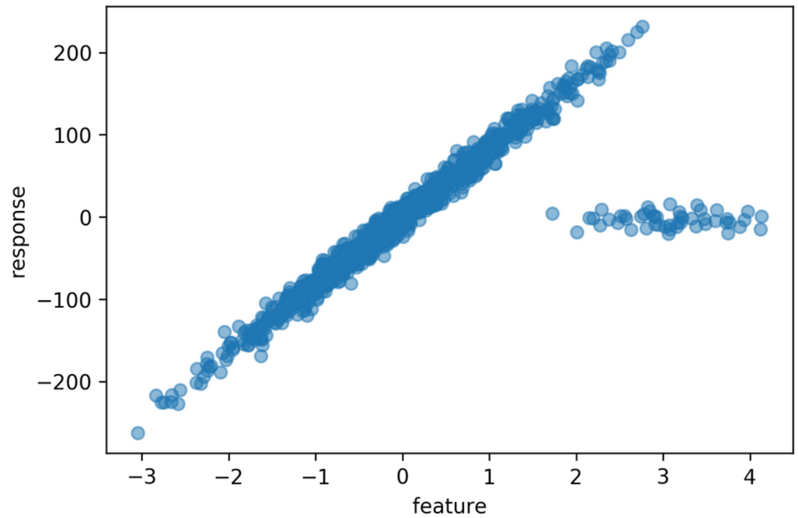


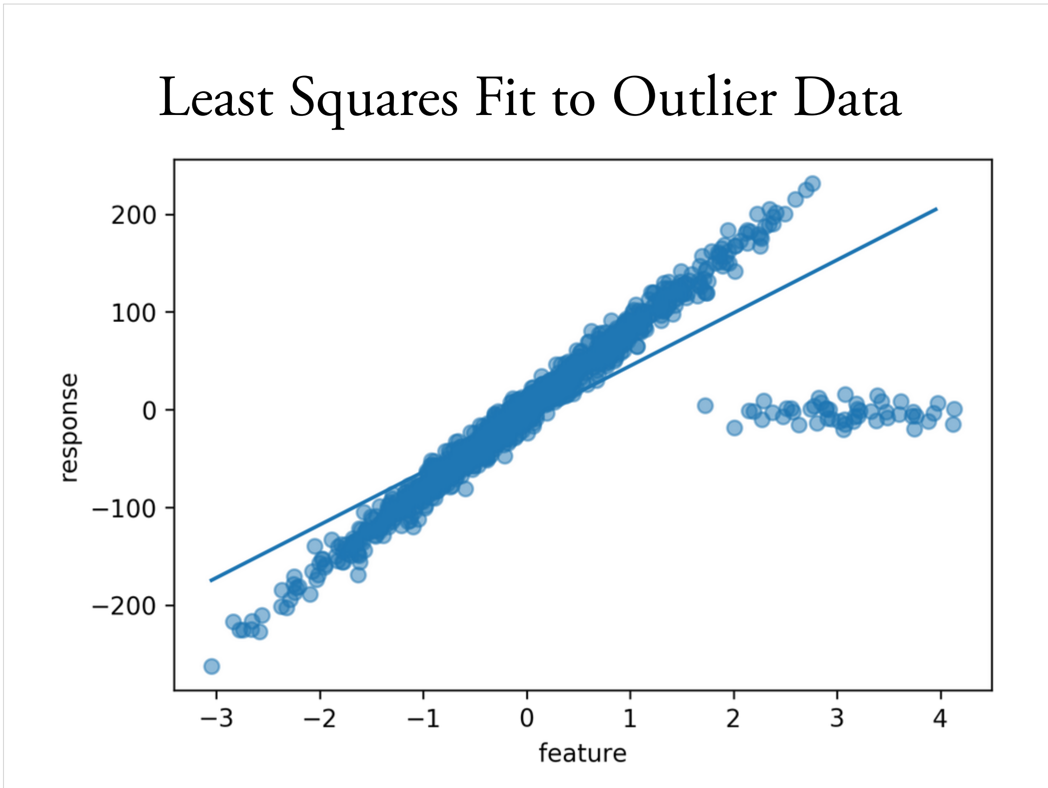
Elastic Net

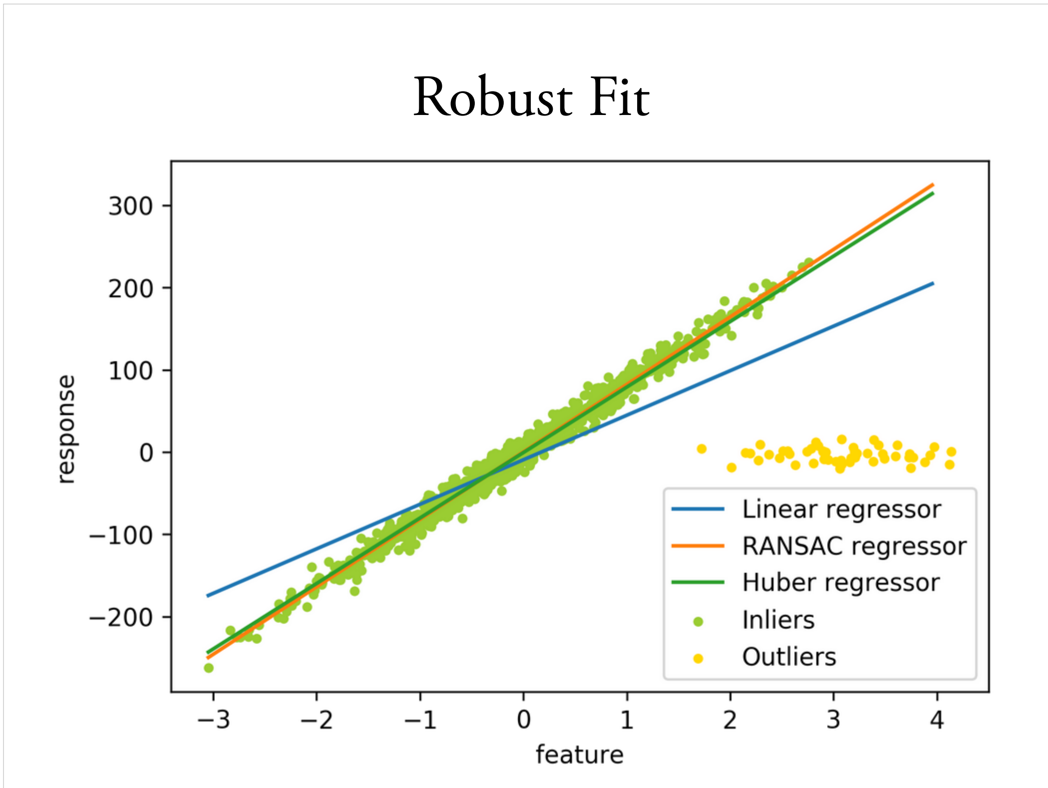
- Combines benefits of Ridge and Lasso
- two parameters to tune.

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

Robust Regression
RANSAC and Huber



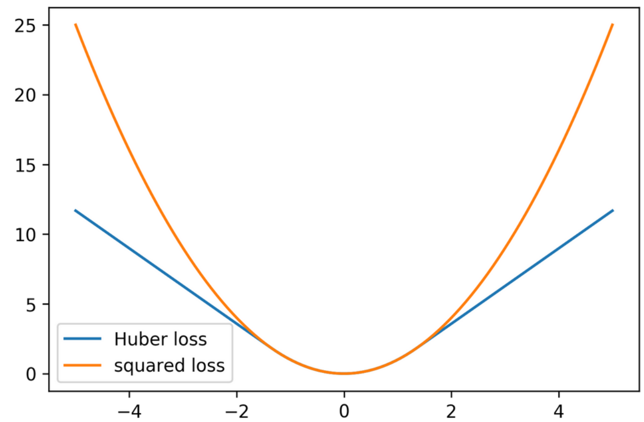

Least Squares Fit to Outlier Data
response
feature
200
100
0
−100
−200
−3
−2
−1
0
1
2
3
4


Robust Fit
response
feature
300
200
100
0
−100
−200
−3
−2
−1
0
1
2
3
4
Linear regressor
RANSAC regressor
Huber regressor
Inliers
Outliers

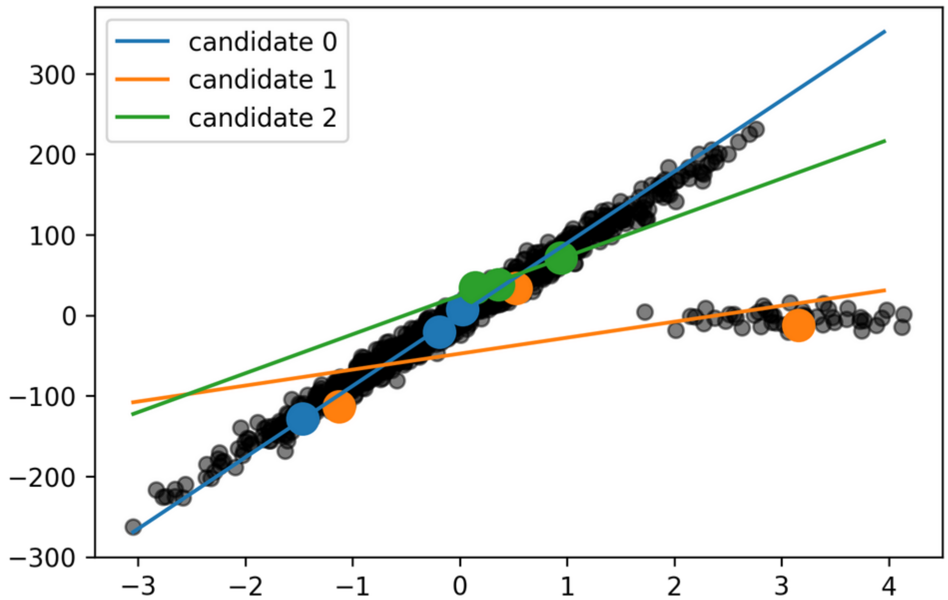
$$\min_{w,\sigma} \sum_{i=1}^{n} \left(\sigma + H_m\left(\frac{X_i w - y_i}{\sigma}\right) \sigma \right) + \alpha ||w||_2{}^2$$

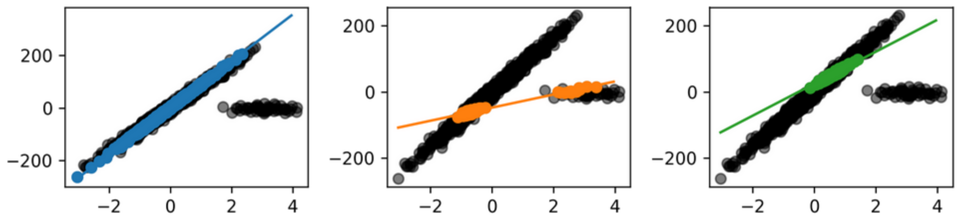
Huber Loss

$$H_m(z) = \begin{cases} z^2, & \text{if } |z| < \epsilon, \\ 2\epsilon|z| - \epsilon^2, & \text{otherwise} \end{cases}$$



RANSAC


candidate 0
candidate 1
candidate 2
300
200
100
0
−100
−200
−300
−3
−2
−1
0
1
2
3
4



200
0
−200
−2
0
2
4
200
0
−200
−2
0
2
4
200
0
−200
−2
0
2
4