Linear Models for Regression

CPDSAI

Chantri Polprasert

Outline

- Linear Regression
- Regularized Linear Regression
  - Ridge Regression
  - Lasso Regression
  - Elastic Net
- Robust Regression

Introduction

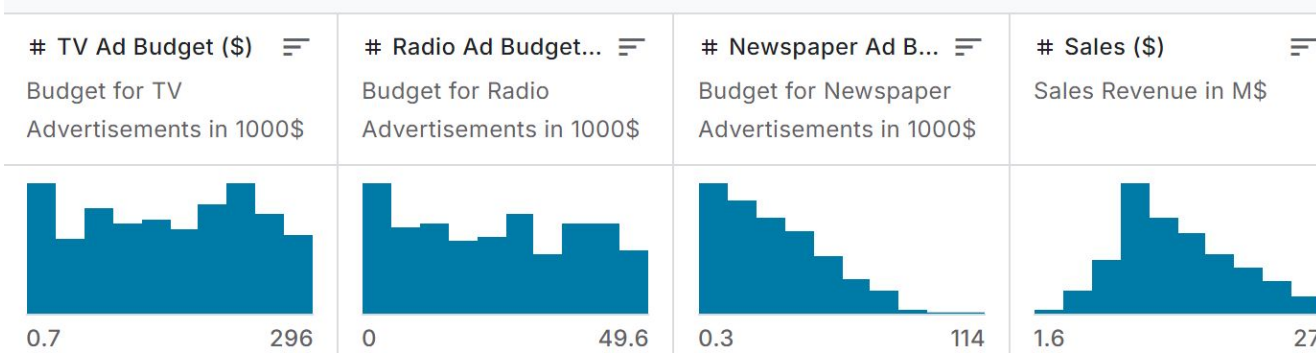
- Linear regression is a supervised machine learning algorithm used for modeling the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data.

- The objective of this model is to find the best-fitting line (or hyperplane) that represents the relationship between the dependent and independent variables.

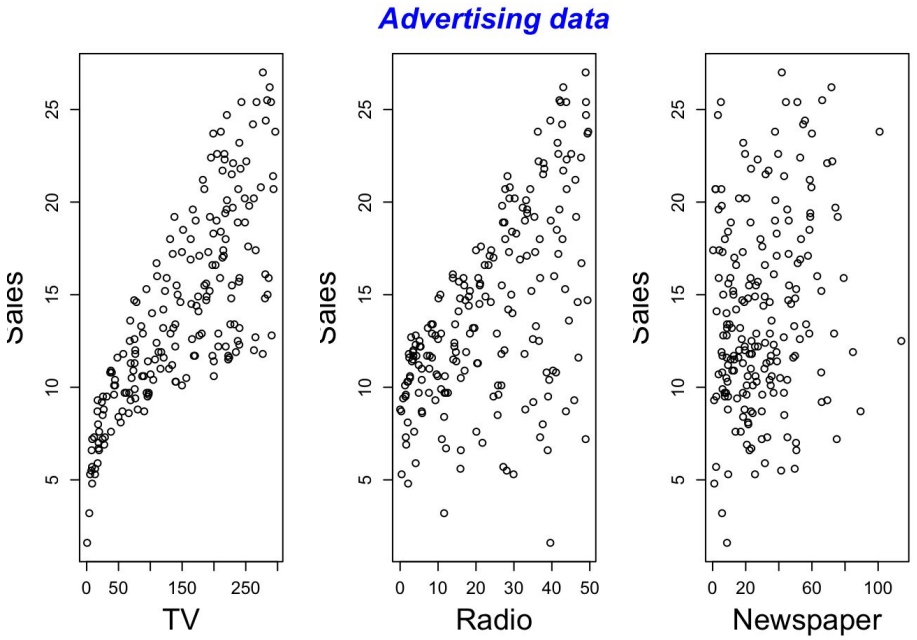
Why Linear Regression

- Linear regression is one of the simplest machine learning algorithms to understand and implement.
- Linear regression serves as a baseline or benchmark model for many machine learning tasks.
- Learning linear regression establishes a common foundation for understanding machine learning concepts. E.g.
  - Loss function
  - assumptions and diagnostics checks
  - regularization

Example: Advertising Sales Dataset



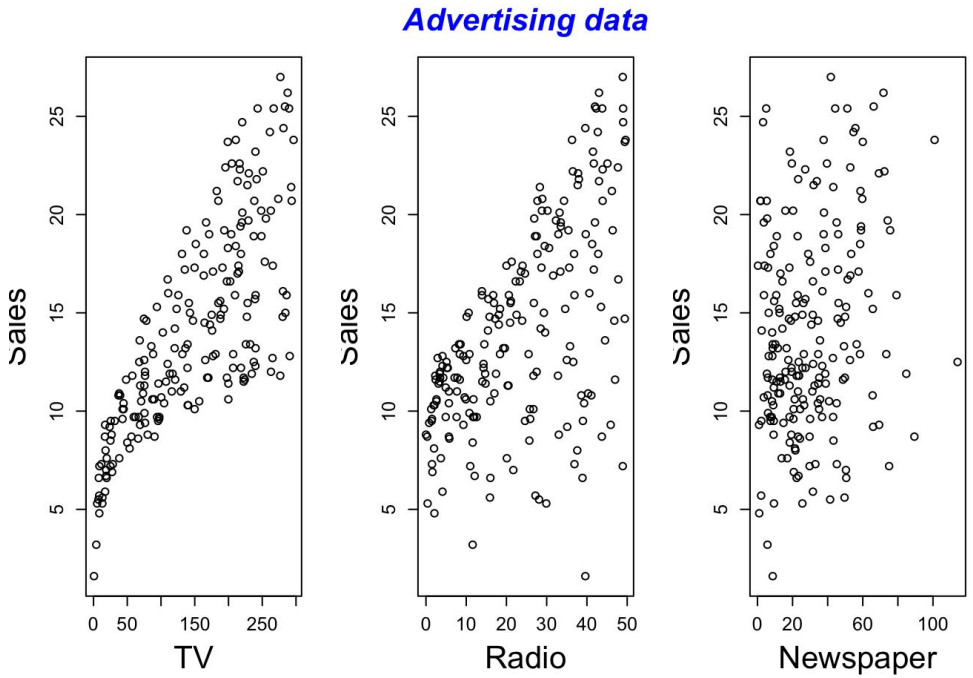
- The advertising dataset captures the sales revenue generated with respect to advertisement costs across multiple channels like radio, tv, and newspapers.
- It is required to understand the impact of ad budgets on the overall sales. an indirect increase in sales.
- Our goal is to develop an accurate model that can be used to predict sales on the basis of the three media budgets.



https://www.kaggle.com/datasets/yasserh/advertising-sales-dataset

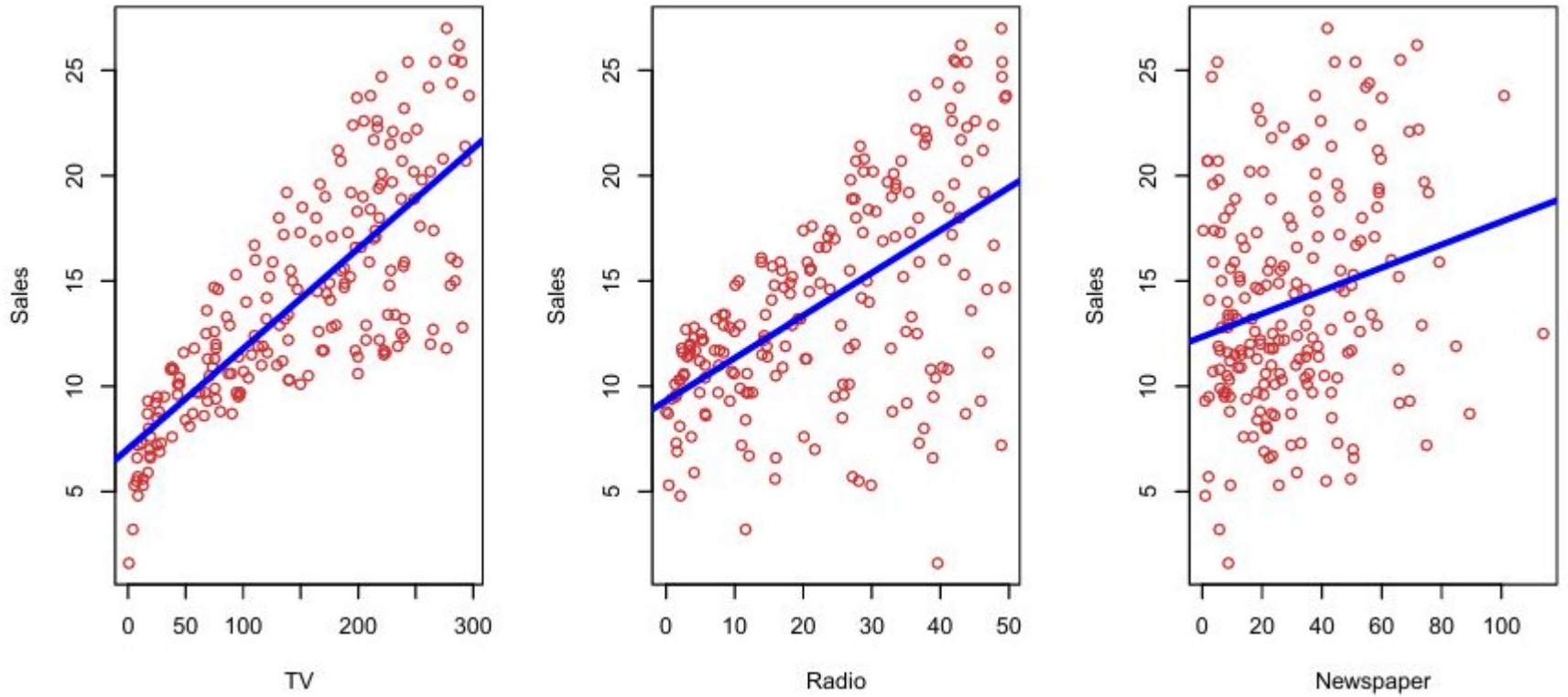
Sample Motivating Questions

1. Is there an association between advertising and sales ?
2. How accurately can we predict future sales?
3. Is the relationship linear?



Advertising vs Sales Plots

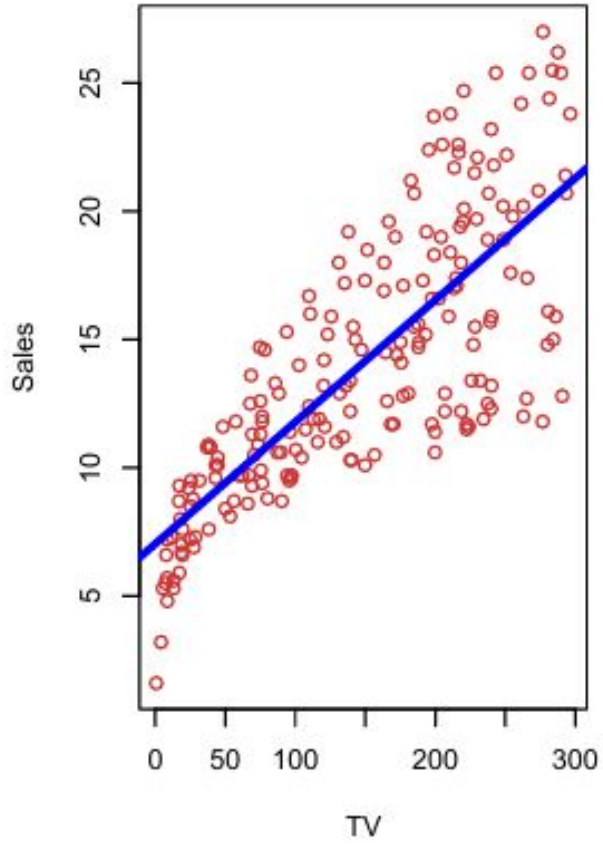
The blue line represents simple linear-regression model that characterizes relationships between each advertisement medium and sales



Simple Linear Regression

- Assume that we can write the following model to represent relationships between X and Y

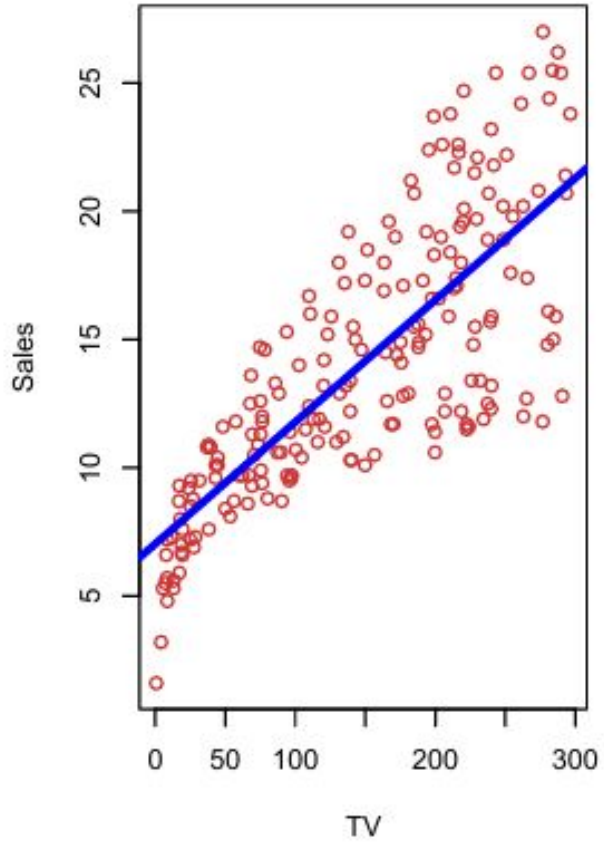
$$Y = \beta_0 + \beta_1 X + \epsilon$$



- Y is the quantitative response (dependent variable)
- X is the predictor (independent variable)
- β_0 is the intercept (unknown)
- β_1 is the slope (unknown)
- ϵ is the error
- For a simple linear regression (SLR), we have a single X

SLR Assumptions

1. **Linearity**: The relationship between the independent and dependent variables is assumed to be linear.
2. **Independence of Errors**: The errors (residuals) in the model should be independent of each other.
3. **Homoscedasticity** : the variance of the residuals is constant across all levels of the predictors



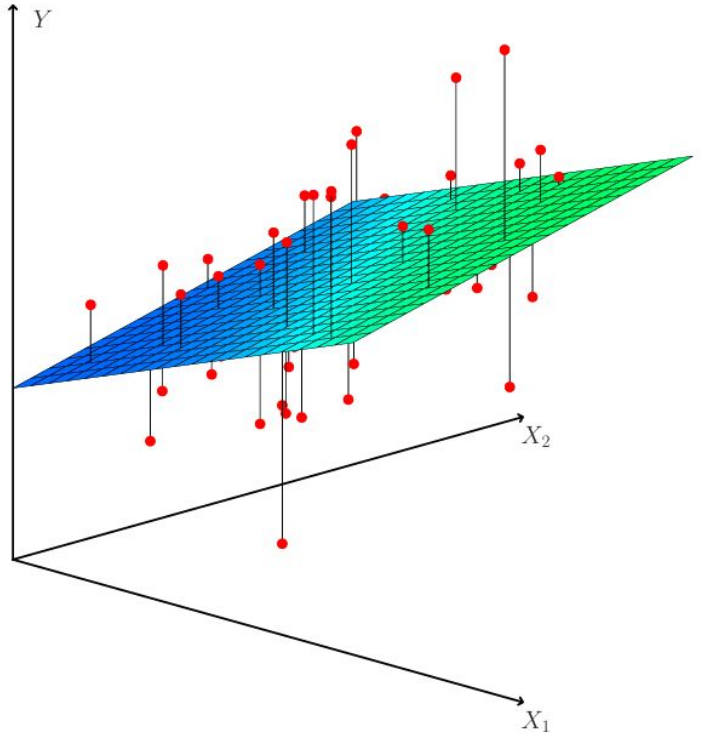
2&3 can be summarized as $\epsilon \sim N(0, \sigma)$

- Let X represent the TV advertising and Y represents the sales, the SLR can be written as

$$\texttt{sales} \approx \beta_0 + \beta_1 \texttt{TV}$$

Multiple Linear Regression

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$



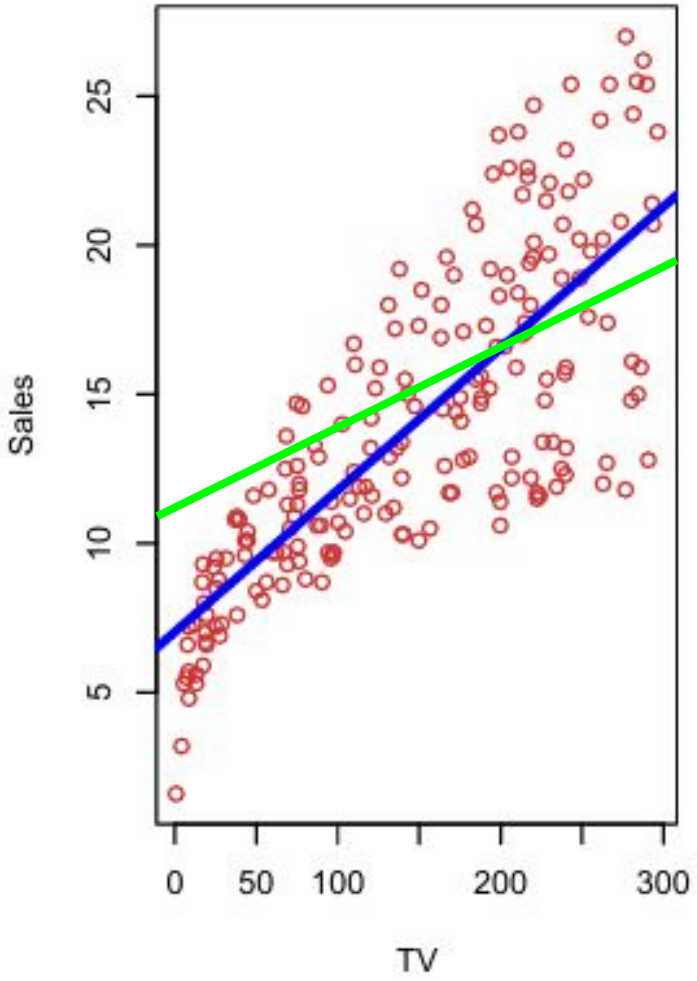
With 2 predictors (p=2) and one response, the regression model becomes a plane

- Y is the response variable
- Xj is the jth random predictor variable
- β_0 is the intercept
- β_j from j = 1,..p are the regression coefficients

- For example, X1 may represents TV advertising, X2 may represent Radio advertising and let Y represents sales. With this information, we can write the relationship between sales and advertising:

$$\texttt{sales} \approx \beta_0 + \beta_1 \texttt{TV} + \beta_2 \texttt{Radio}$$

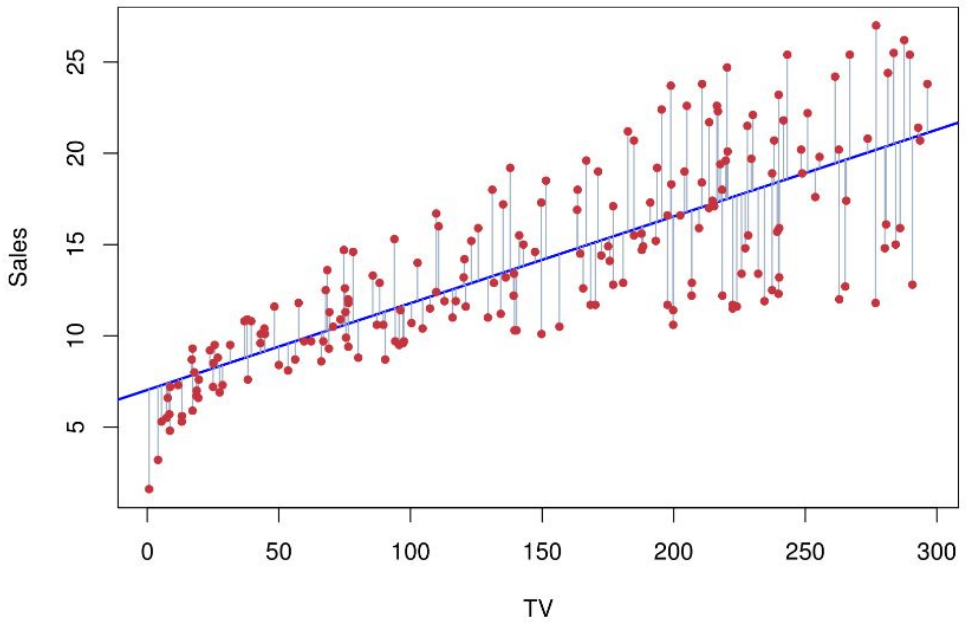
Which Model Fits the Best?



$$\hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- x_i is the ith predictor
- $\hat{y}_i$ is the estimate of the response
- $\hat{\beta}_0$ is the estimated intercept
- $\hat{\beta}_1$ is the estimated slope

Calculating the Residual Sum of Square (RSS)



- For any line, we can define the observed error (residual) as

$$e_i = y_i - \hat{y}_i$$

- Determine the line that minimizes the

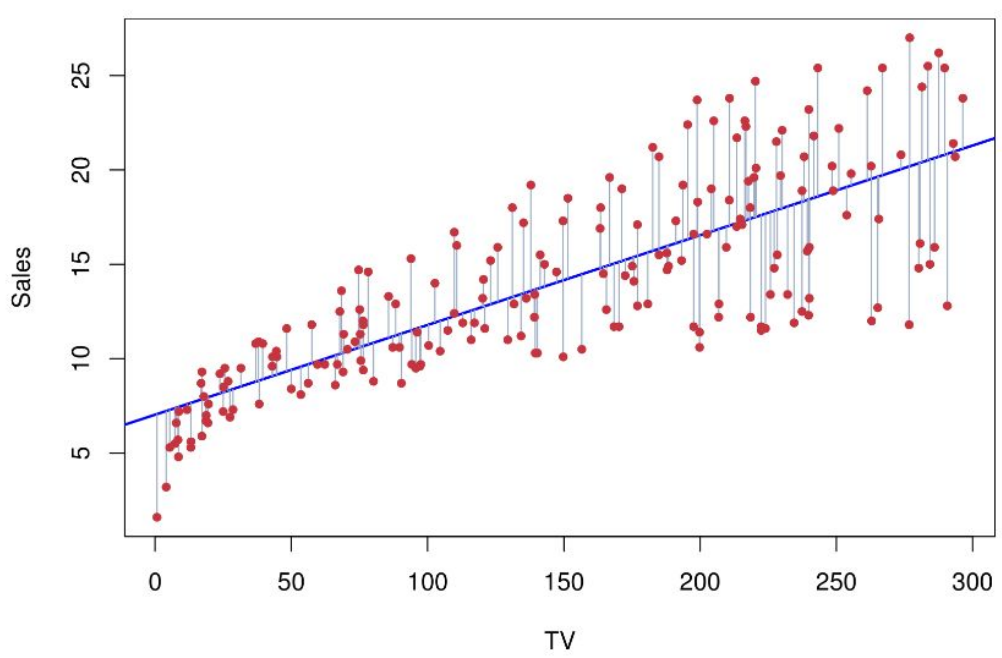
$$\text{RSS} = e_1^2 + e_2^2 + \ldots + e_n^2$$

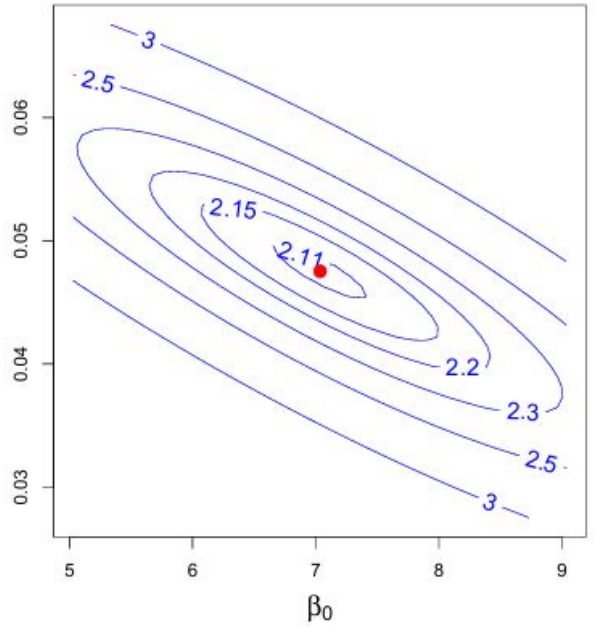
$$\hat{\beta}_1 = \frac{\sum(y_i - \bar{y})(x_i - \bar{x})}{\sum(x_i - \bar{x})^2}$$

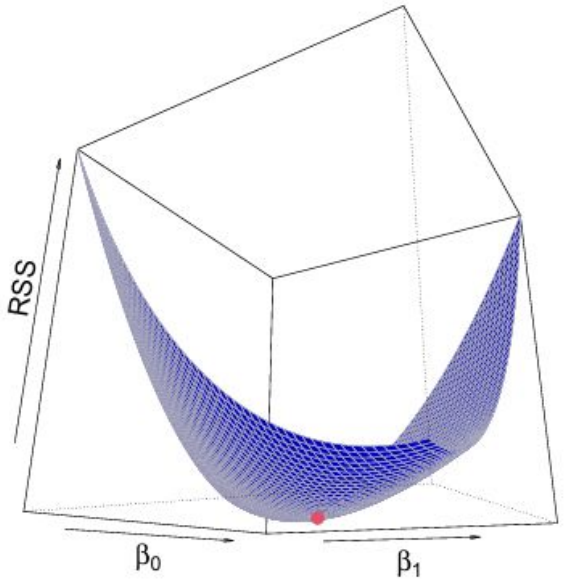
$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Derivation:
Deriving the least squares estimators of the slope and intercept (simple linear regression) (youtube.com)

Minimum RSS: Cost Function







$$J(\hat{\beta}_0, \hat{\beta}_1) = \sum_{i=1}^{n} \left(y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i) \right)^2$$

Measuring fit

1. RSS
2. The coefficient of determination R^2

$$R^2 = \frac{\text{TSS-RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})}$$

- R^2 is the proportion of the total variation that is explained by the regression line
- It measures how close the fitted line is to the observed data.
- It is easily to interpret as it lies between 0 and 1*
- $\text{TSS} = \sum(y_i - \bar{y})^2$ is the total sum of squares
- What is a good R^2?

LinearRegression

```
class sklearn.linear_model.LinearRegression(*, fit_intercept=True, copy_X=True,
n_jobs=None, positive=False)                                              [source]
```

- Ordinary least squares Linear Regression.
- Fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

Syntax:

Import the class containing the regression method

```
from sklearn.linear_model import LinearRegression
```

Create an instance of the class

```
LR = LinearRegression()
```

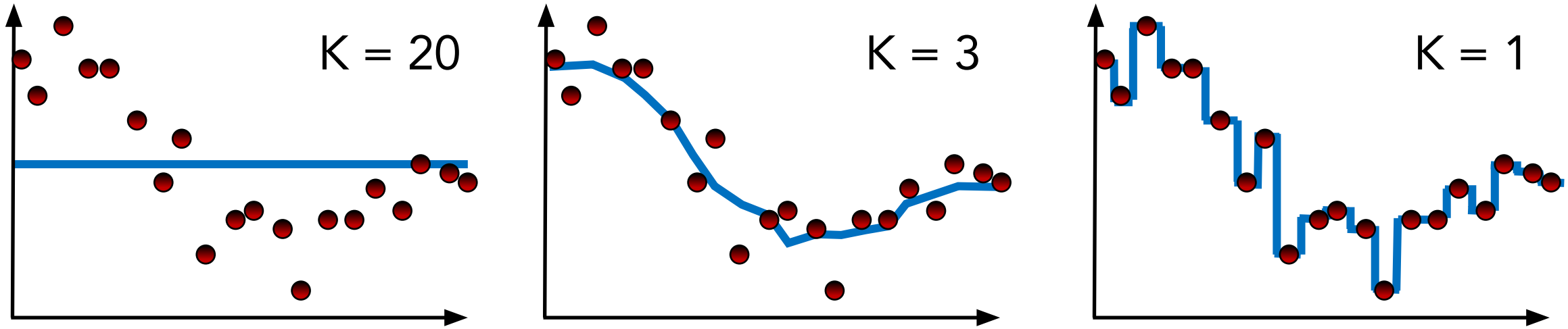
Fit the instance on the data and then predict the expected value

```
LR = LR.fit(X_train, y_train)
y_predict = LR.predict(X_test)
LR.score(X_test, y_test)
```

Attributes:

- coef_ : Estimated coefficients
- intercept_

Regression with KNN



The value predicted is the mean value of the neighbors.

- When k is the same value as the number of points, a single value is predicted that is the mean of all values
- When k is larger than 1, KNN regression acts as a smoothing function
- When k is 1, the prediction simply connects each point

Comparing Linear Regression and KNN

Linear Regression

- Parametric model
- Fitting involves minimizing cost function (*slow*)
- Model has few parameters (*memory efficient*)
- Prediction involves calculation (*fast*)

K Nearest Neighbors

- Non-parametric model
- Fitting involves storing training data (*fast*)
- Model has many parameters (*memory intensive*)
- Prediction involves finding closest neighbors (*slow*)

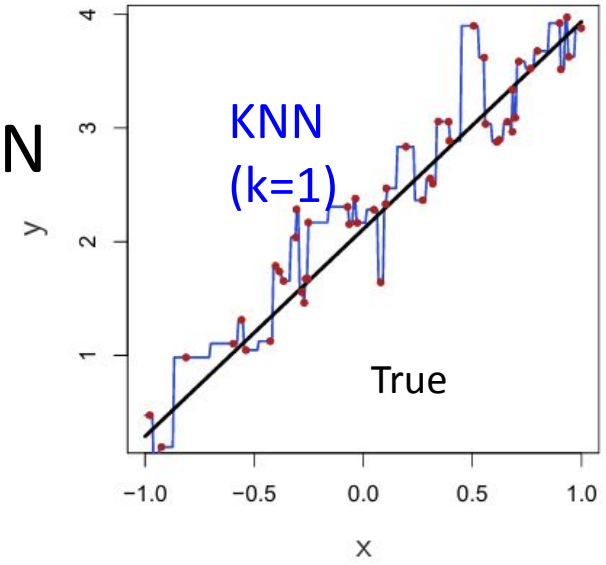
Which one is better?

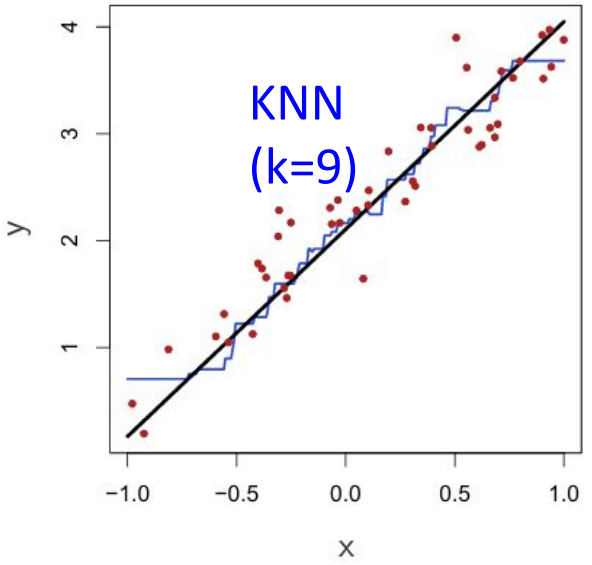
Comparing Linear Regression and KNN

- True relationship (linear with p=1)
- Regression model (SLR with p=1)

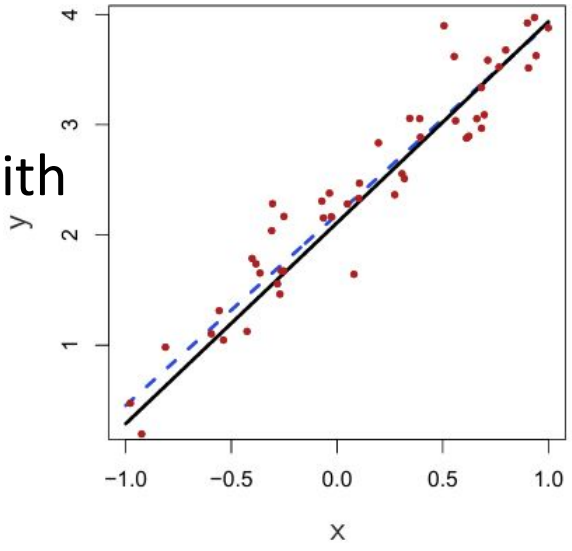
Which one is better?

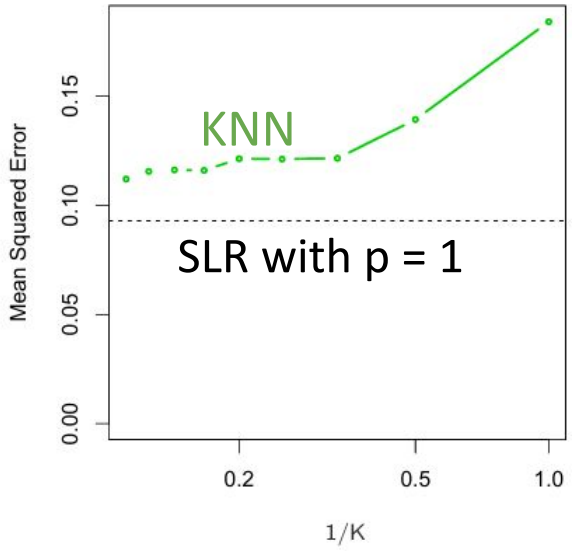
KNN





SLR with p = 1



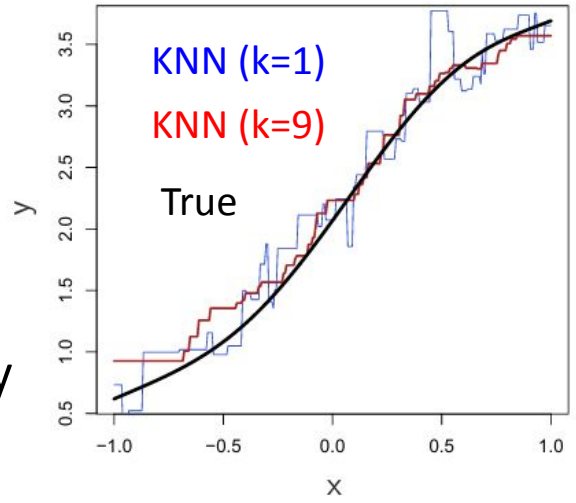


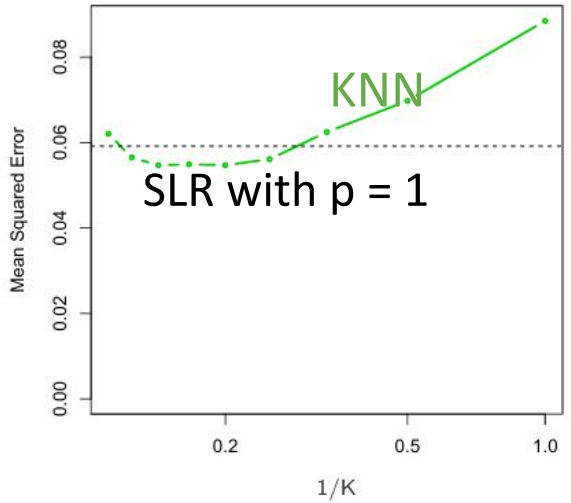
Comparing Linear Regression and KNN

- True relationship (non-linear)
- Regression model (SLR with p=1)

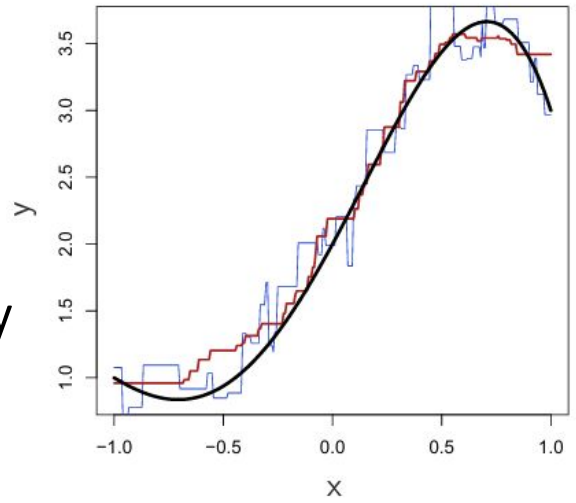
Which one is better?

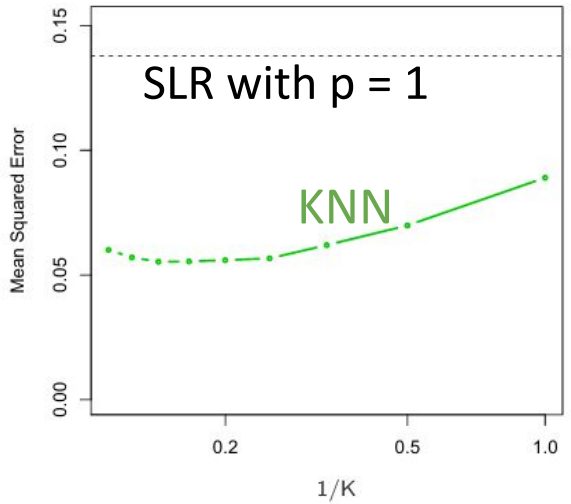
True relationship with small non-linearity





True relationship with large non-linearity





Linear Regression Vs Non-Linear Regression

$$y_\beta(x) = \beta_0 + \beta_1 x$$

$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$

$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

$$y_\beta(x) = \beta_0 + \beta_1 \log(x)$$

$$y_\beta(x) = \beta_0 + x^{\beta_1}$$

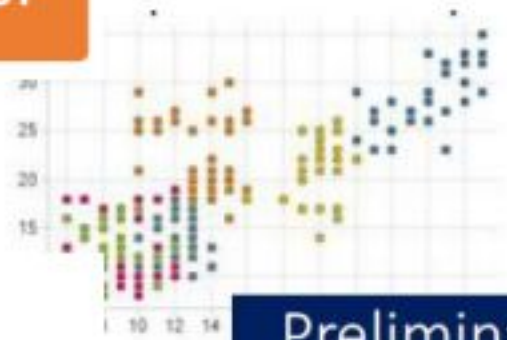
$$y_\beta(x) = \beta_0 + (\beta_1 - \beta_2) e^{(\beta_3 x^{\beta_4})}$$

Linear regression equation is **linear** in the **parameters**

Extension of the linear model

Exploratory Data Analysis (EDA)

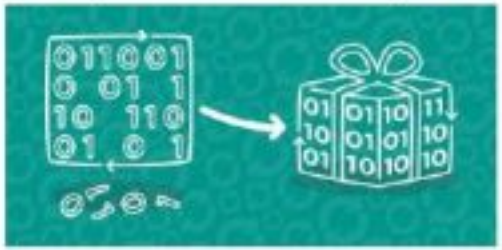
Step 2-A: Explore



Preliminary analysis

Understand nature of data

Step 2-B: Pre-process

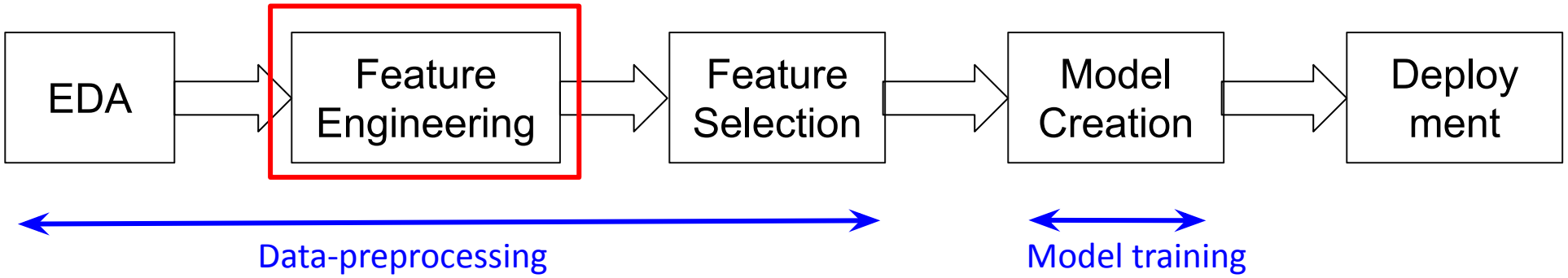


Clean

Integrate

Package

Feature Engineering

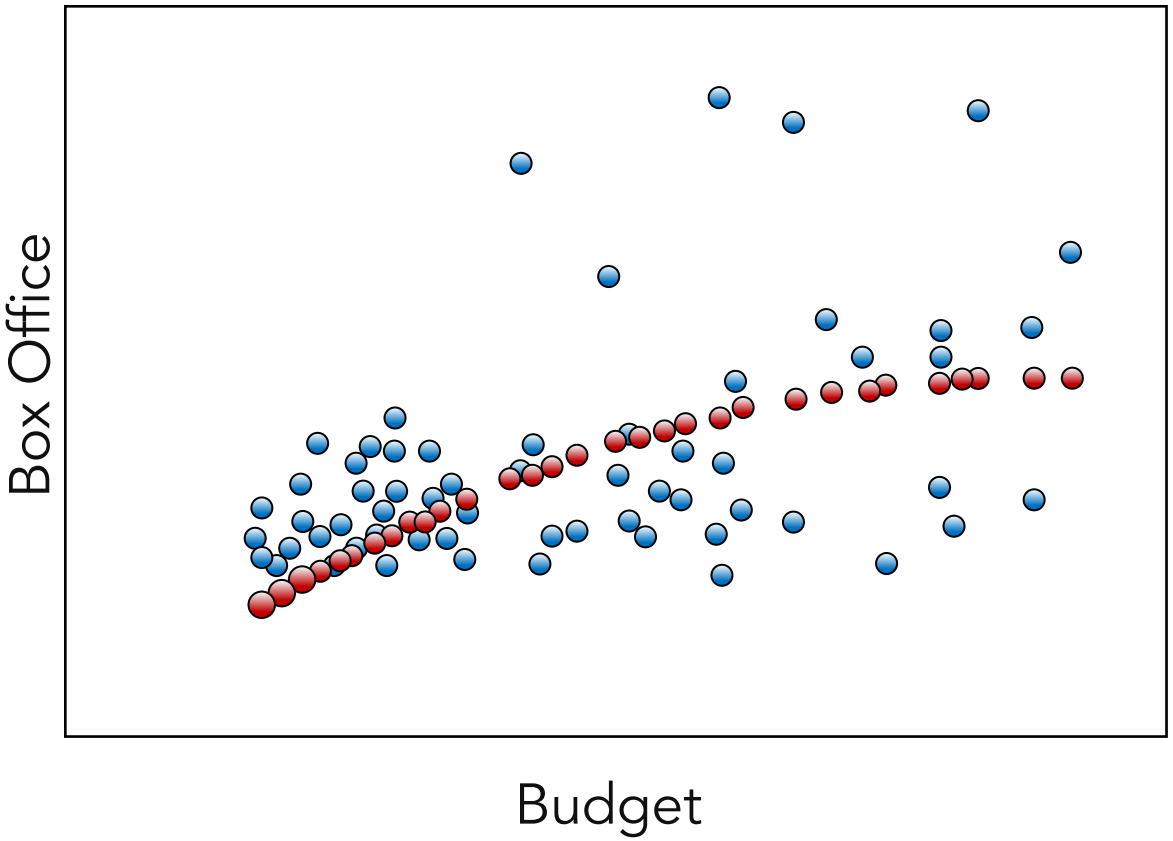


Addition of Polynomial Features

- Another pre-processing trick we can apply is to create new features out of the ones we already have.
- Capture higher order features of data by adding polynomial features

- "Linear regression" means linear combinations of features

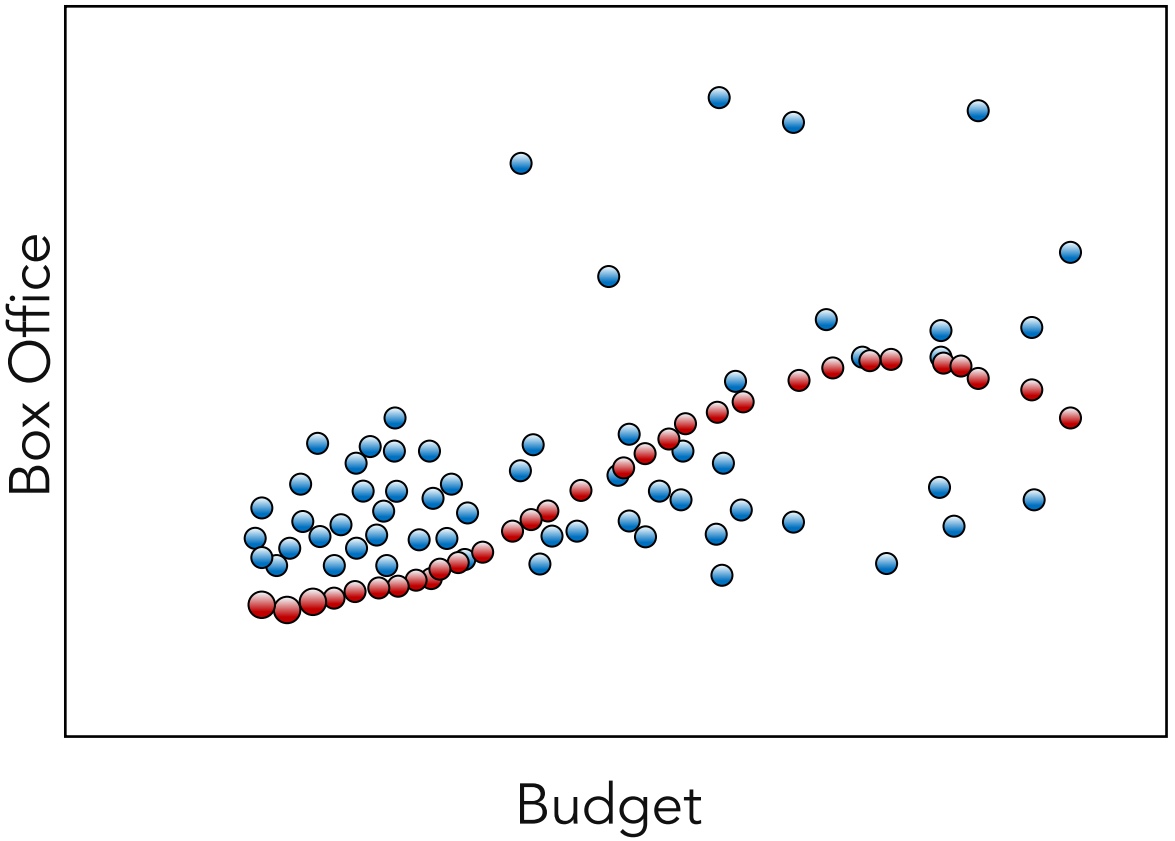
$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2$$



Addition of Polynomial Features

$$y_\beta(x) = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$$

- Another pre-processing trick we can apply is to create new features out of the ones we already have.
- Capture higher order features of data by adding polynomial features
- "Linear regression" means linear combinations of features

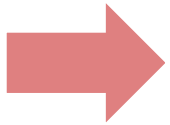


Addition of Polynomial Features

- Can also include variable interactions

$$y_\beta(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1 x_2$$

- How is the correct functional form chosen?



Check relationship of each variable or with outcome

PolynomialFeatures

sklearn.preprocessing.PolynomialFeatures

class sklearn.preprocessing.**PolynomialFeatures**(*degree=2, *, interaction_only=False, include_bias=True, order='C'*) [sourc

- Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree.
- For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, a^2, ab, b^2]

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

Polynomial Features: The Syntax

Import the class containing the transformation method

```
from sklearn.preprocessing import PolynomialFeatures
```

Create an instance of the class

```
polyFeat = PolynomialFeatures(degree=2)
```

Create the polynomial features and then transform the data

```
X_poly = polyFeat.fit_transform(X_data)
```

Stay tuned for the best practices in using the Transformer with make_pipeline later!

fit_transform: fit to data and transform it

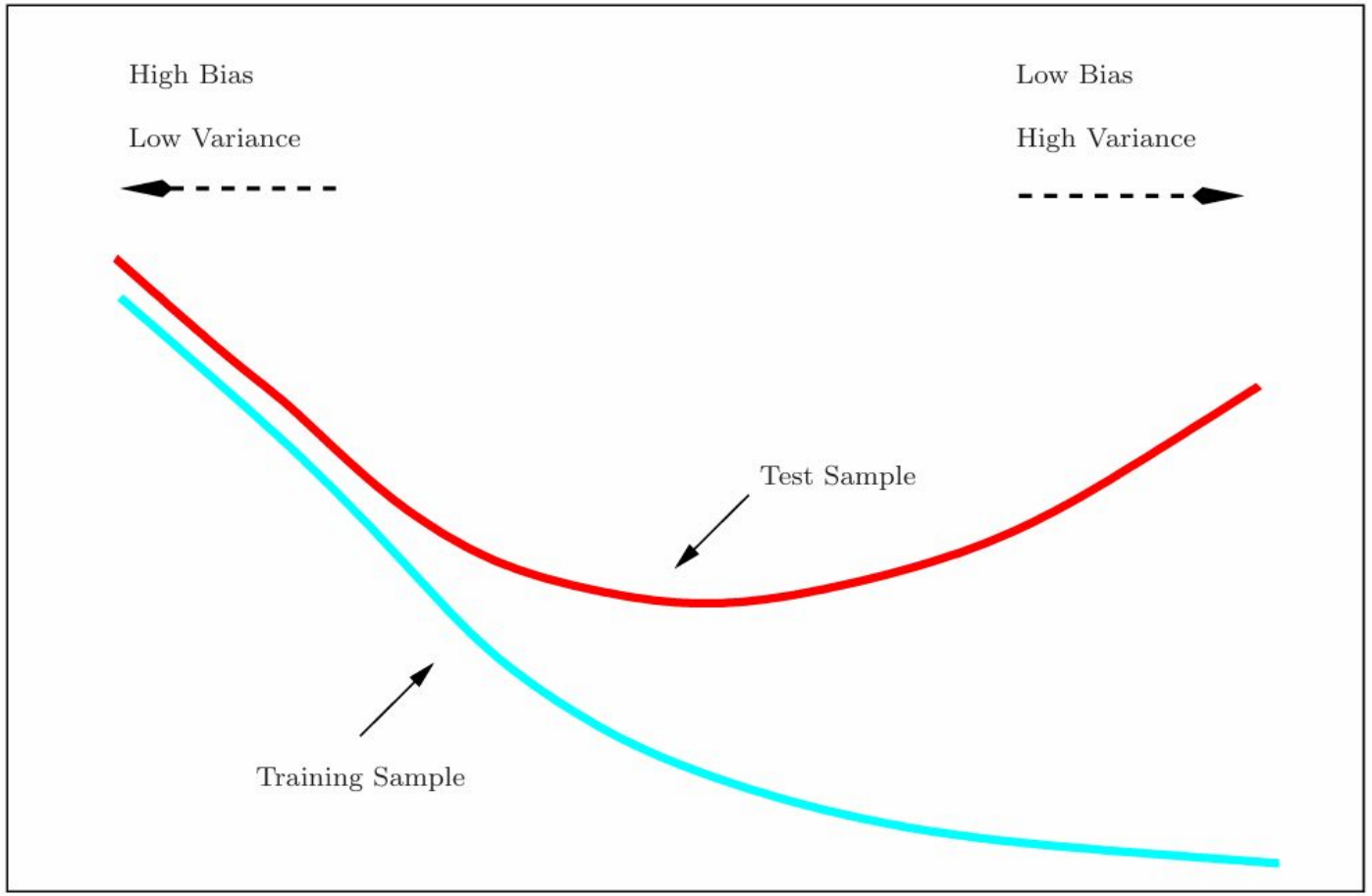
Method	Purpose	Syntax
fit_transform()	Learn the parameters and apply the transformation to new data	transformed_data = estimator.fit_transform(X)
fit()	Learn and estimate the parameters of the transformation	estimator.fit(X)
transform()	Apply the learned transformation to new data	transformed_data = estimator.transform(X)

Regularized Linear Regression

Outline

- Regularized Linear Regression
  - Ridge Regression
  - Lasso Regression
  - Elastic Net
- Robust Regression

Reminder on model complexity

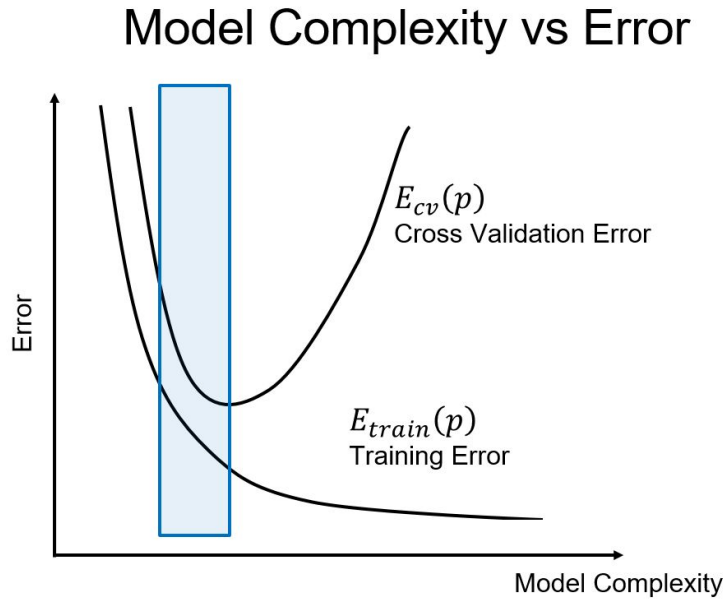


- Models that are too complex for the amount of training data available are said to overfit and are not likely to generalize well to new examples.
- Models that are too simple, that don't even do well on the training data, are said to underfit and also not likely to generalize well.

- Adding regularization move us from right to left

Regularized Linear Regression

- Intuition Idea
  - When building a model, we want training and test (cross-validation) errors to be small. We discussed building a simpler model if training error was small and test (cross-validation) error was large. There is another way to achieve this! **Regularization**
  - Key: Add a regularization term to the loss function to prevent overfitting



Ridge Regression

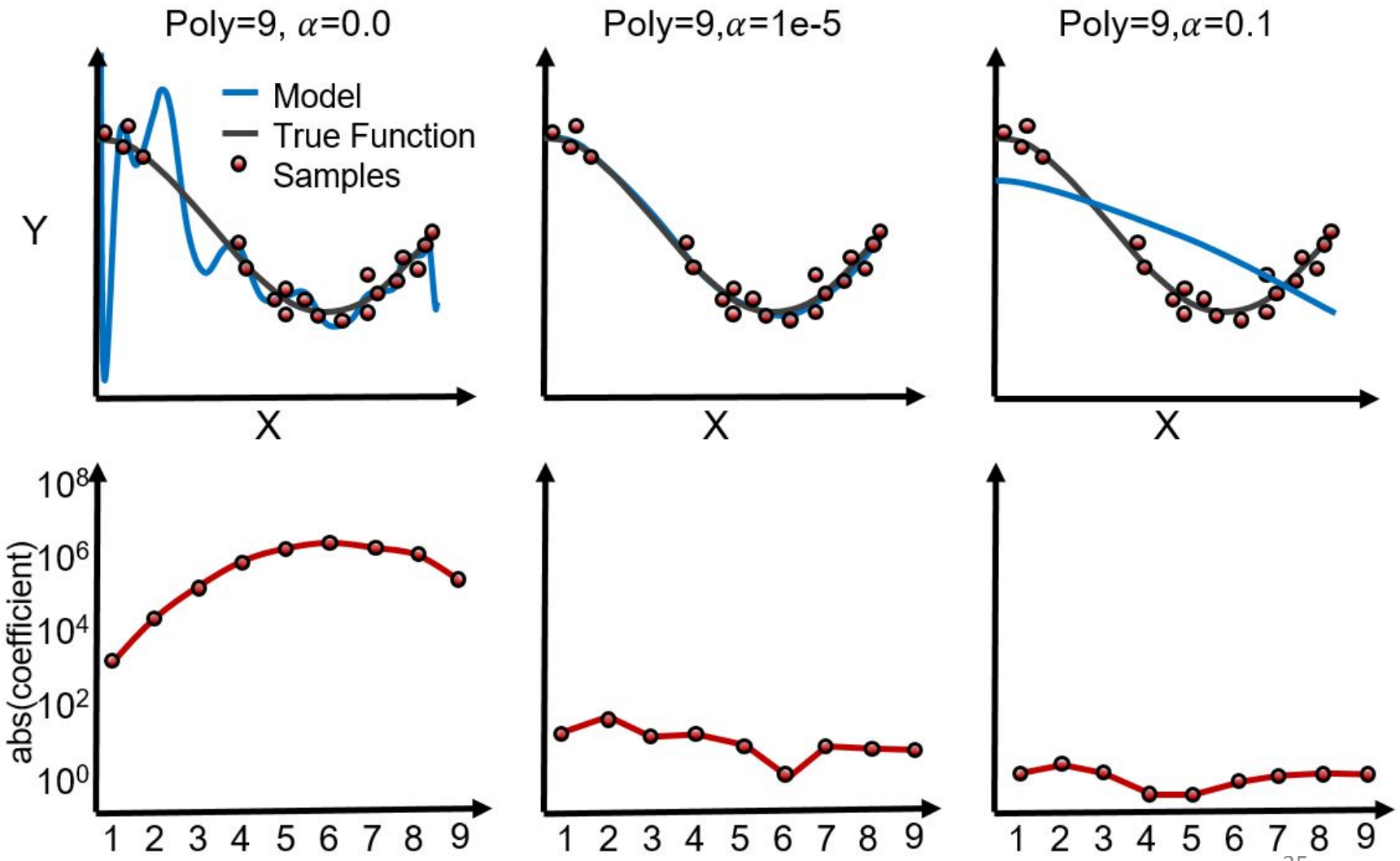
$$\min_{w \in \mathbb{R}^p} \sum_{i=1}^{n} ||w^T \mathbf{x}_i - y_i||^2 + \alpha ||w||^2$$

- Once the parameters are learned, the ridge regression prediction formula is the same as ordinary least-squares.
- The addition of a parameter penalty in ***w*** parameters is called regularization. Regularization prevents overfitting by restricting the model, typically to reduce its complexity.
- Ridge regression uses L2 regularization: minimize sum of squares of ***w*** entries (squared euclidean norm)
  - Not only do we want to fit the training data well, we also want ***w*** to have a small squared L2 norm.
- The influence of the regularization term is controlled by the α parameter.
  - Higher alpha means more regularization and simpler models.

$$\boldsymbol{w}^*, b^* = \underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} (f(x_i; \boldsymbol{w}, b) - y_i)^2 + \alpha \sum_{k=1}^{p} (w_k)^2$$

Effects of Regularization

- These are the regularization costs for the polynomials.
- It is very high for the first blue curve, which is a very overfitting model.
- For the second model, the regularization cost is not as high.
- It is the lowest for the 3rd curve.



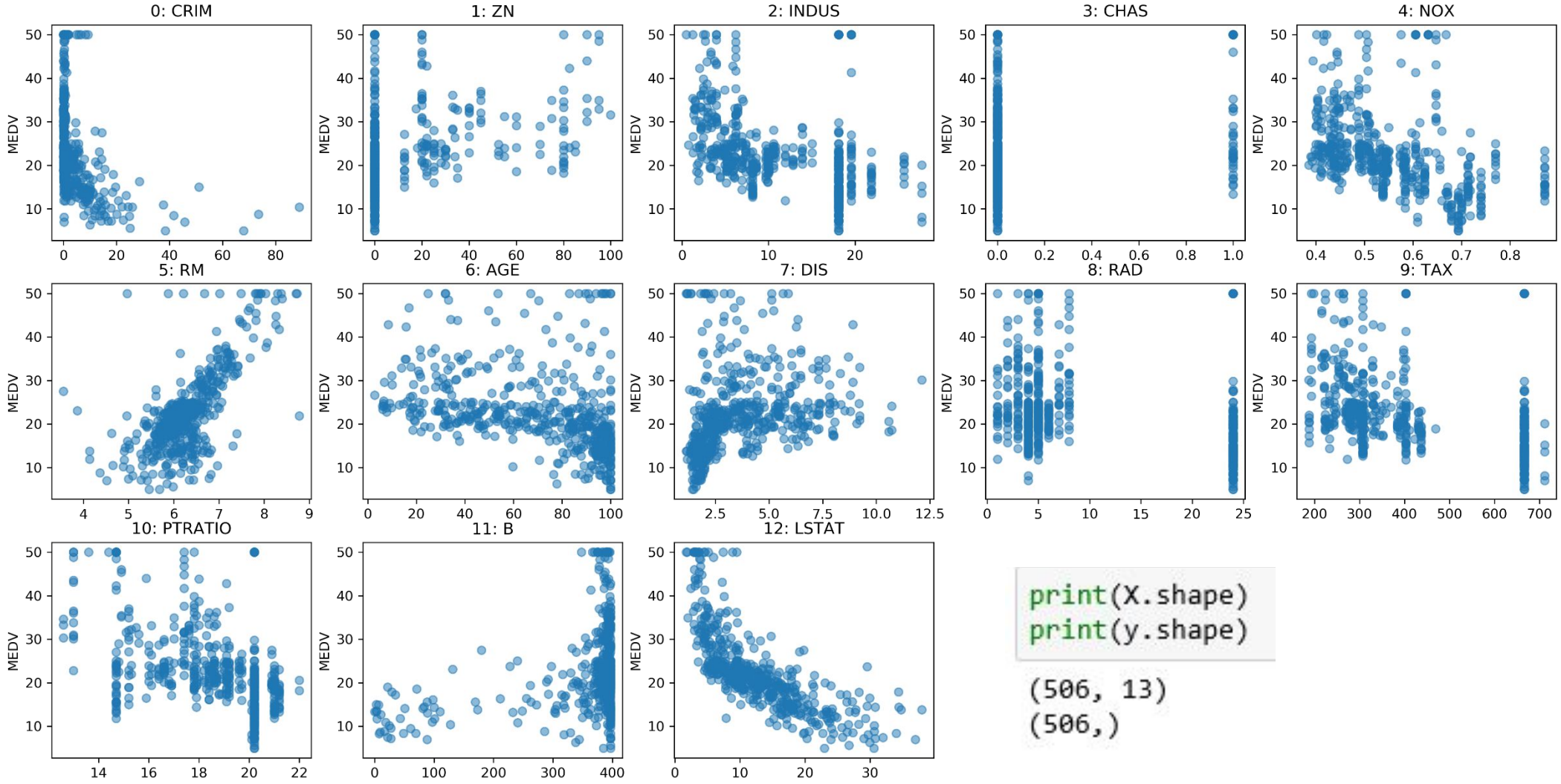
(regularized) Empirical Risk Minimization

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

- We formulate the machine learning problem as an optimization problem over a family of functions.
- The minimization problem consists of two parts, the data fitting part and the model complexity part.
- Most machine learning algorithms can be cast into this, with a particular choice of family of functions f, loss function L and regularizer R.

Boston Housing Dataset

- To predict prices of property in the Boston area in different neighbourhoods.

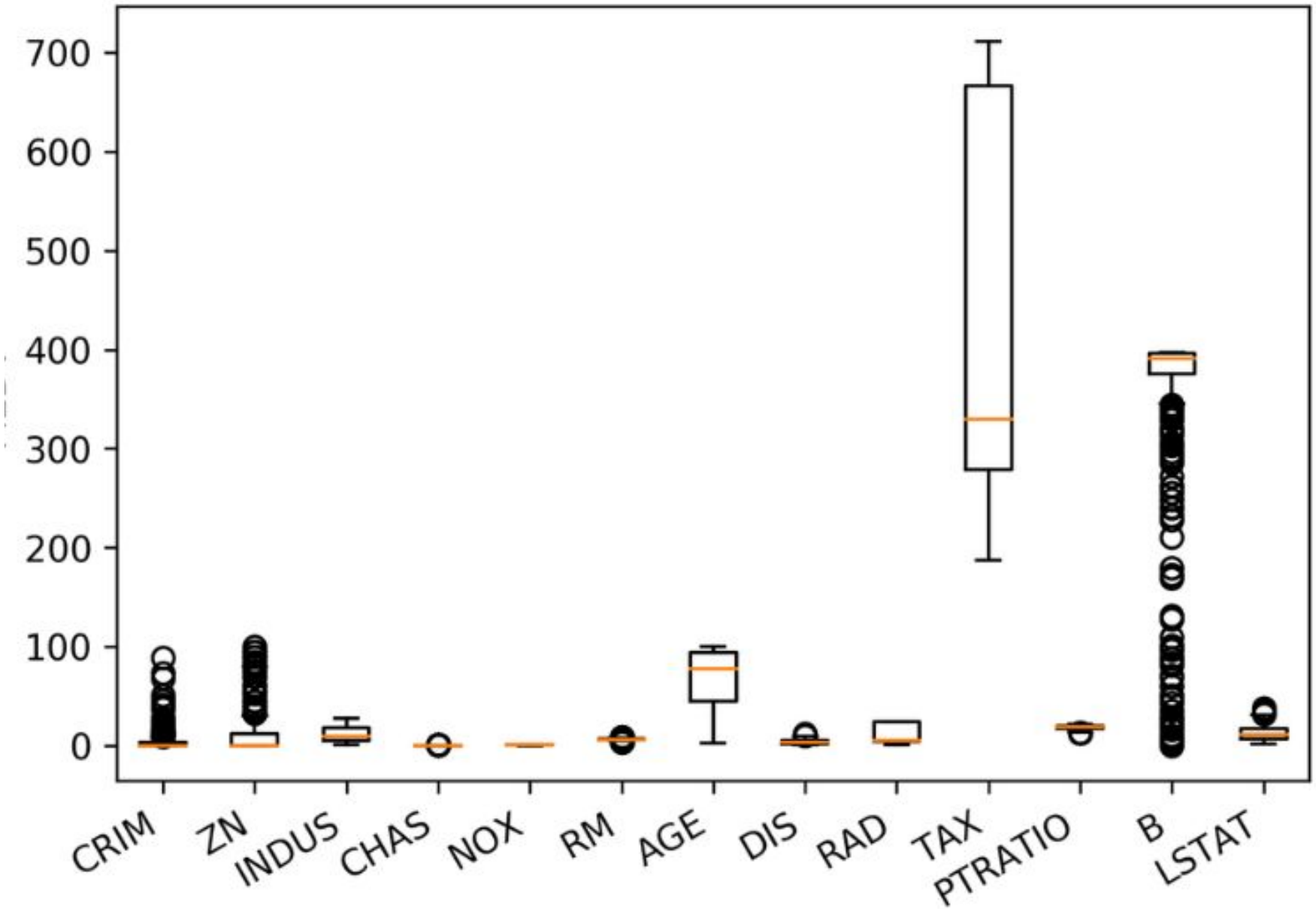



```
print(X.shape)
print(y.shape)
```

```
(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>
```



Scikit-learn fit the Linear/Ridge Regression

```
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
```

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

```
0.715
```

$$R^2 = 1 - \frac{SSE}{TSS}$$

Score is Coefficient of determination: R^2

- A measure of goodness of a prediction for a regression model
- Generally, a score has the value between 0 and 1, can be negative.
- A value of 1 corresponds to a perfect prediction, and a value of 0 corresponds to a constant model that just predicts the mean of the training set responses

Scaling

Ridge regression assumes the predictors are standardized and the response is centered!

```
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)
```

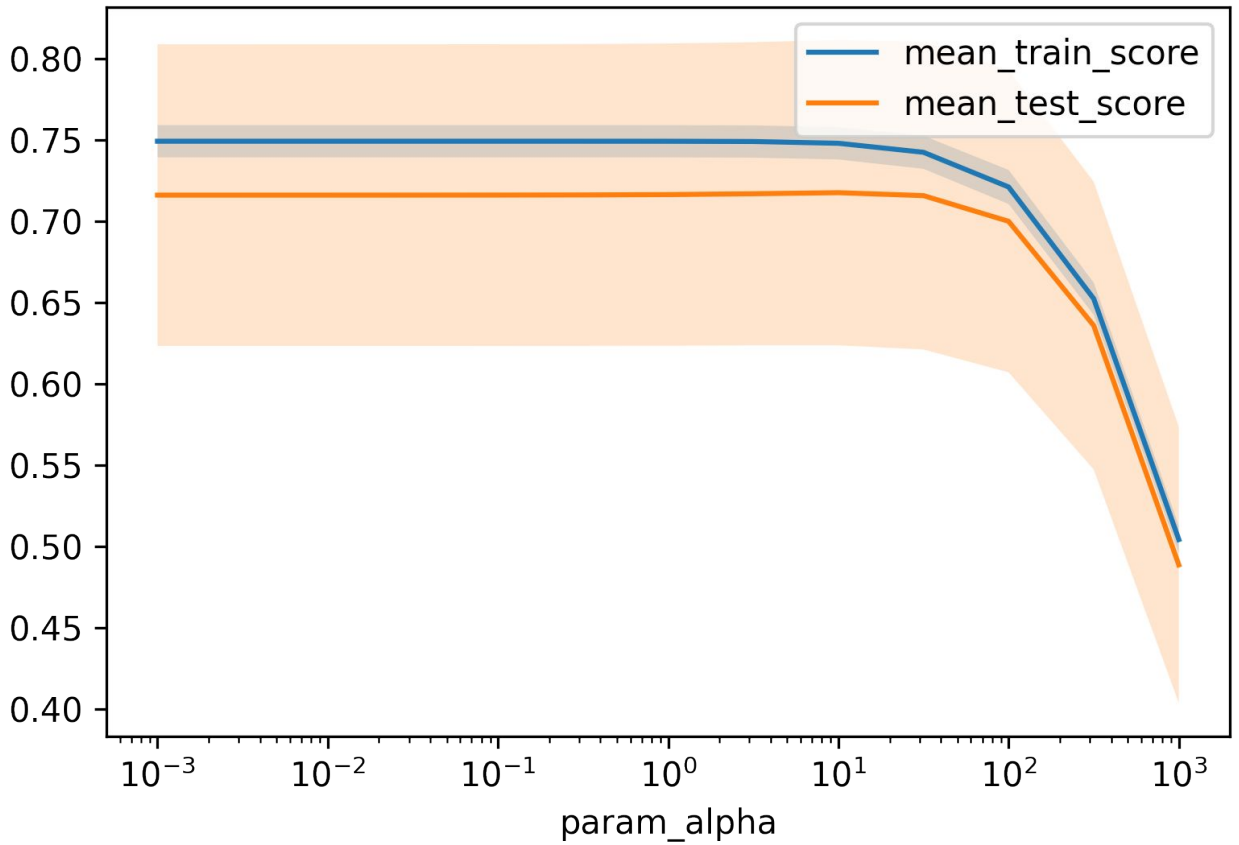
```
0.68
```

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

```
1 grid = GridSearchCV(Ridge(), param_grid, cv=10, return_train_score=True)
2 grid.fit(X_train_scaled, y_train)
```

Grid-Search on Alpha for Ridge



Adding Polynomial Features

```
1 from sklearn.preprocessing import PolynomialFeatures, scale
2 X, y = boston.data, boston.target
3 X_poly = PolynomialFeatures(include_bias=False).fit_transform(scale(X))
4 print(X_poly.shape)
5 X_train, X_test, y_train, y_test = train_test_split(X_poly, y, random_state=42)
```

```
(506, 104)
```

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

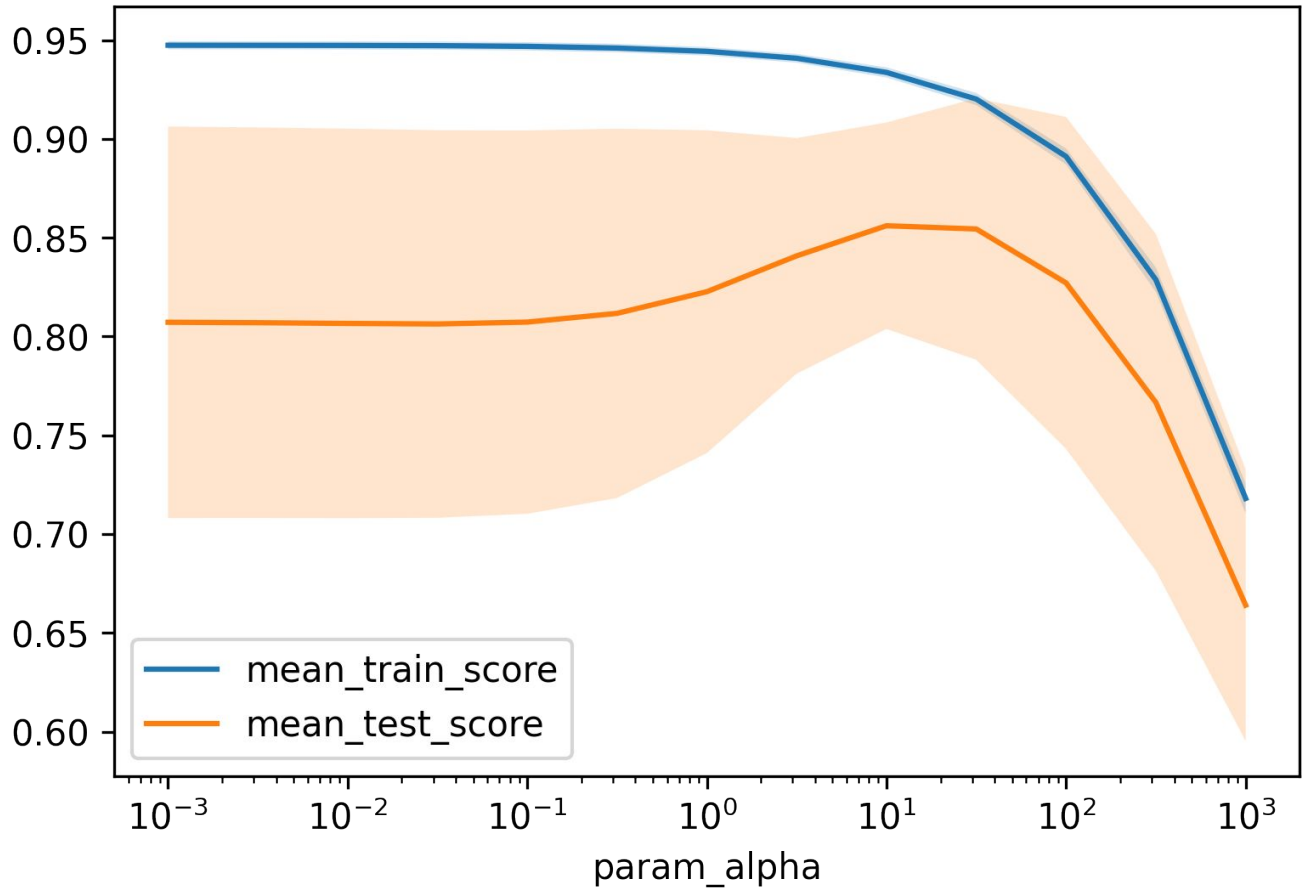
```
0.8065470566933797
```

```
1 np.mean(cross_val_score(Ridge(alpha=1.0), X_train, y_train, cv=10))
```

```
0.8227099613418707
```

Note: Fit_transform the scale of X before train_test_split is actually bad data leakage (more on this later).

Grid-Search on Alpha for Ridge

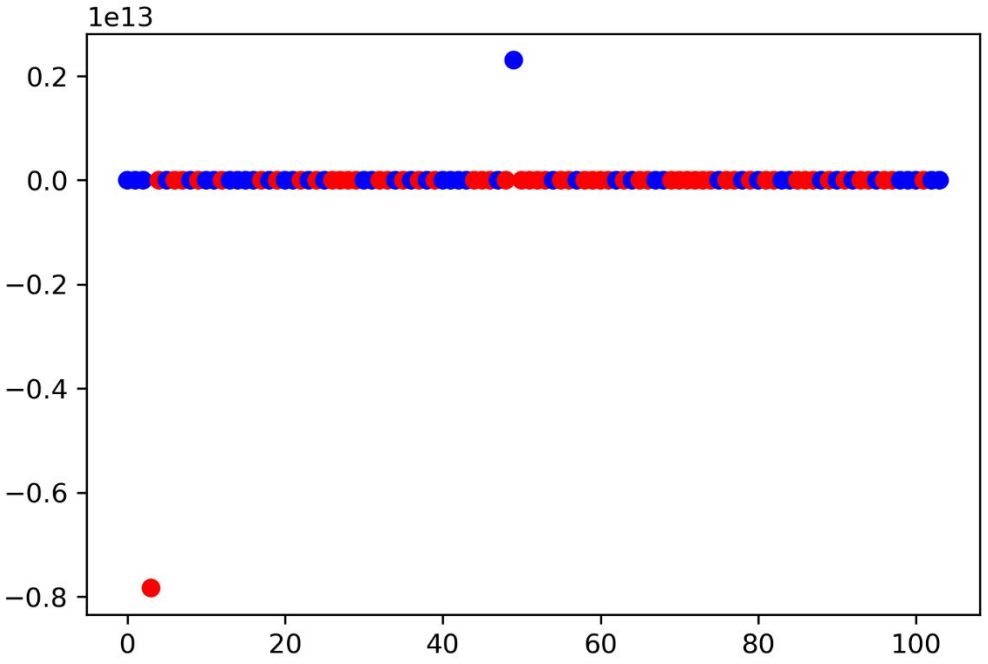



```
1 print(grid.best_params_)
2 print(grid.best_score_)
```

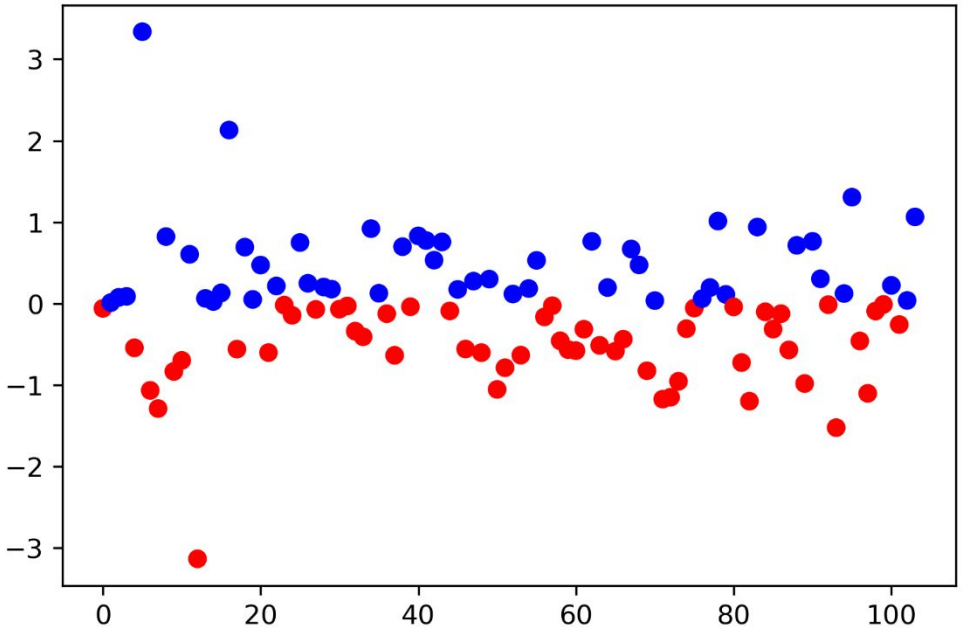
```
{'alpha': 10.0}
0.8560976200047948
```

Plotting coefficient values on polynomial features

Linear Regression

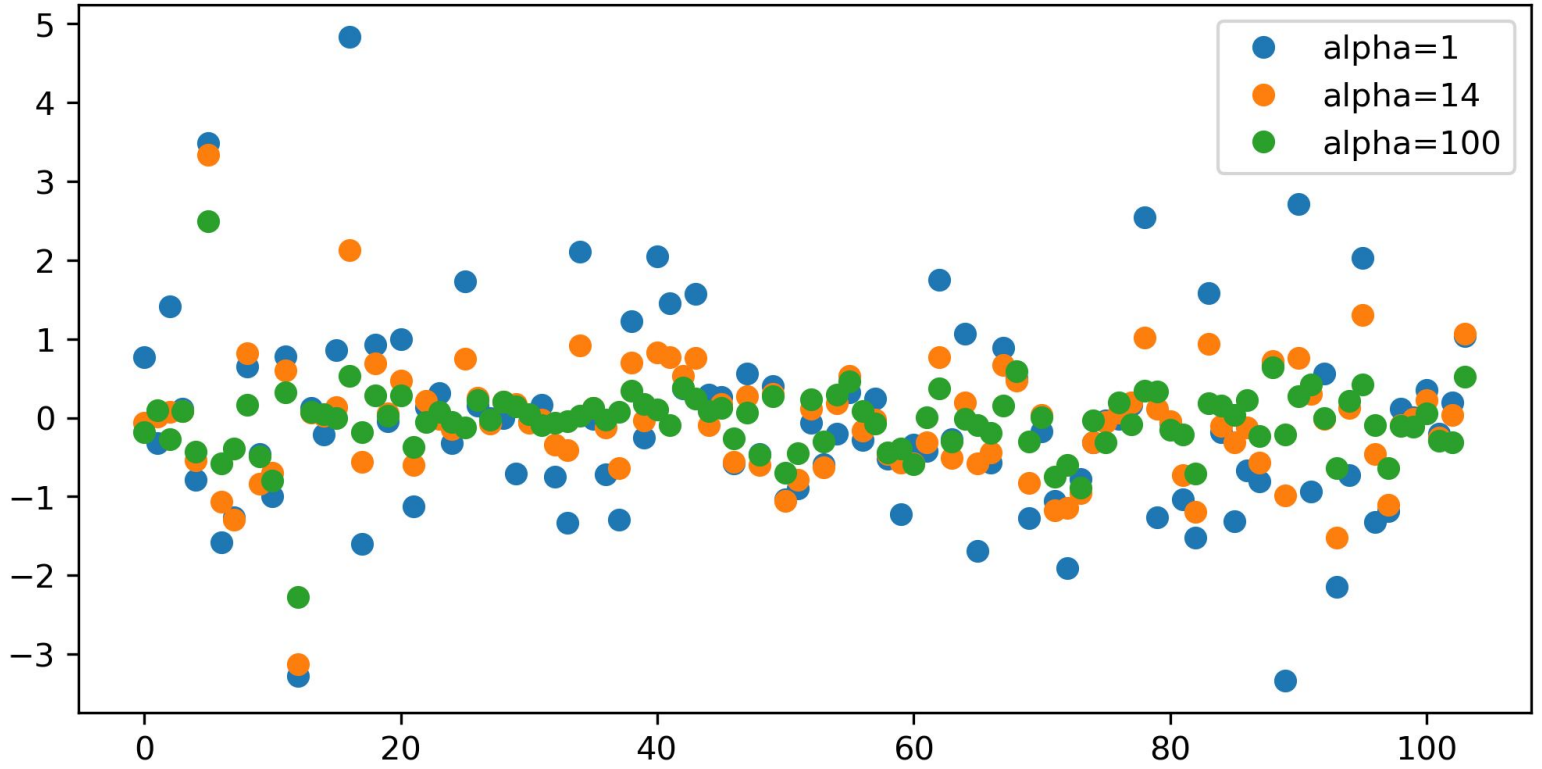


Ridge Regression



Ridge coefficients at different alpha

- As alpha large, it pushes the coefficients toward zero, but pushes at different speed of alpha magnitude, even flips the sign. So, harder to interpret with regularization.



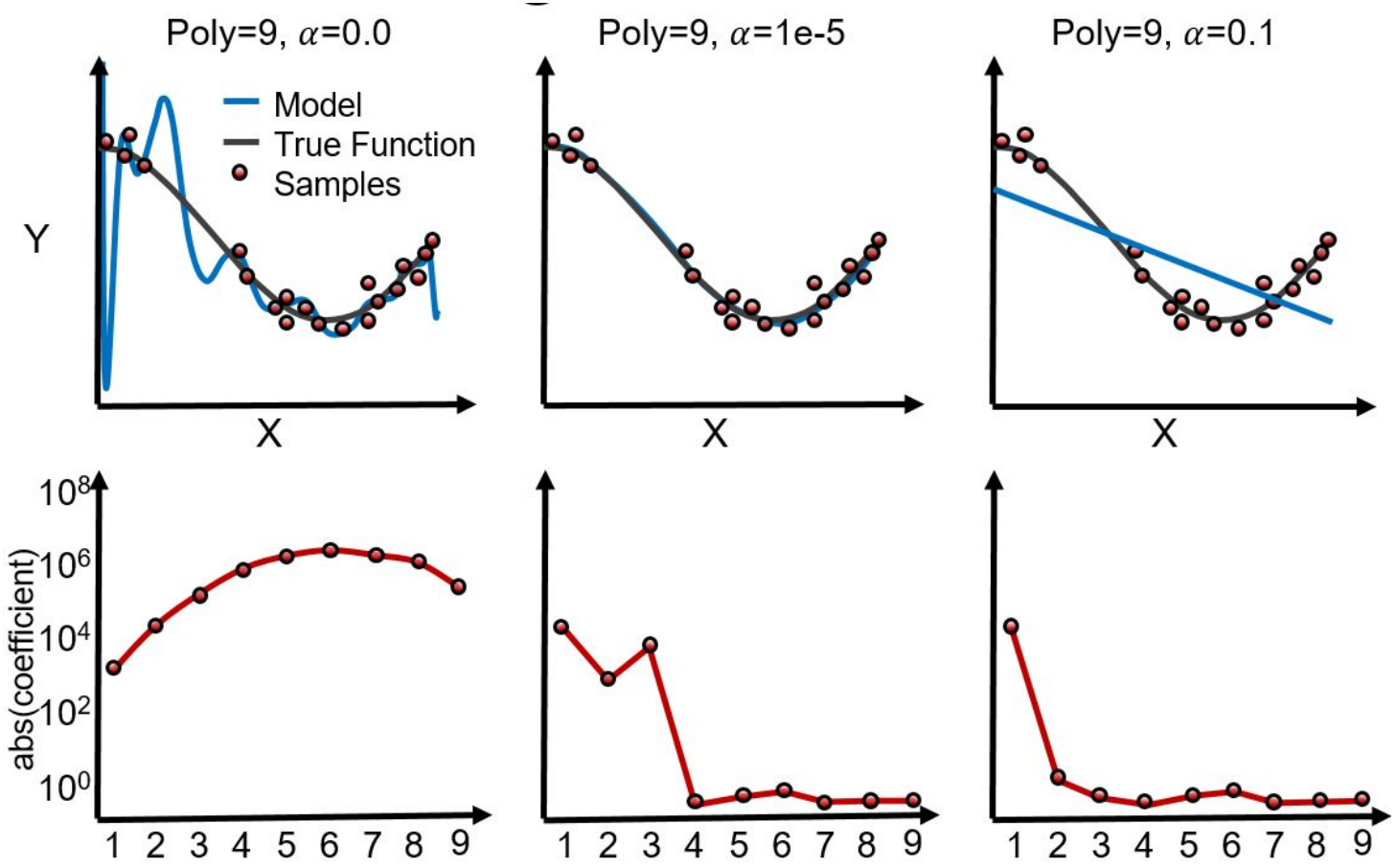
Lasso Regression

(Least Absolute Shrinkage and Selection Operator)

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

- Lasso regression is another form of regularized linear regression that uses an L1 regularization penalty for training (instead of ridge's L2 penalty)
- L1 penalty: Minimize the sum of the absolute values of the coefficients
- This has the effect of setting parameter weights in ***w*** to zero for the least influential variables. This is called a sparse solution: a kind of feature selection
- The parameter α controls amount of L1 regularization (default = 1.0).
- The prediction formula is the same as ordinary least-squares.
- When to use ridge vs lasso regression:
  - Many small/medium sized effects: use ridge.
  - Only a few variables with medium/large effect: use lasso.

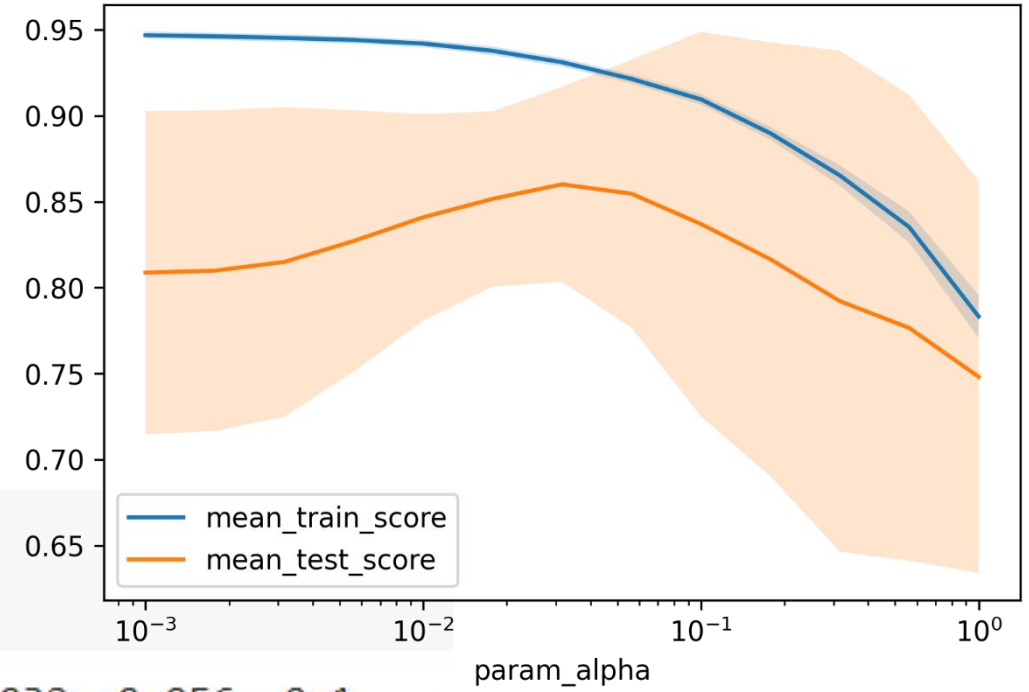
Effect of Lasso Regression on Parameters



$$\boldsymbol{w}^*, b^* = \underset{w,b}{\operatorname{argmin}} \sum_{i=1}^{n} (f(x_i; \boldsymbol{w}, b) - y_i)^2 + \alpha \sum_{\boldsymbol{k=1}}^{\boldsymbol{p}} |w_k|$$

- Here, some coefficients can stay high, and others are close to zero, or zero.

Grid-Search for Lasso




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

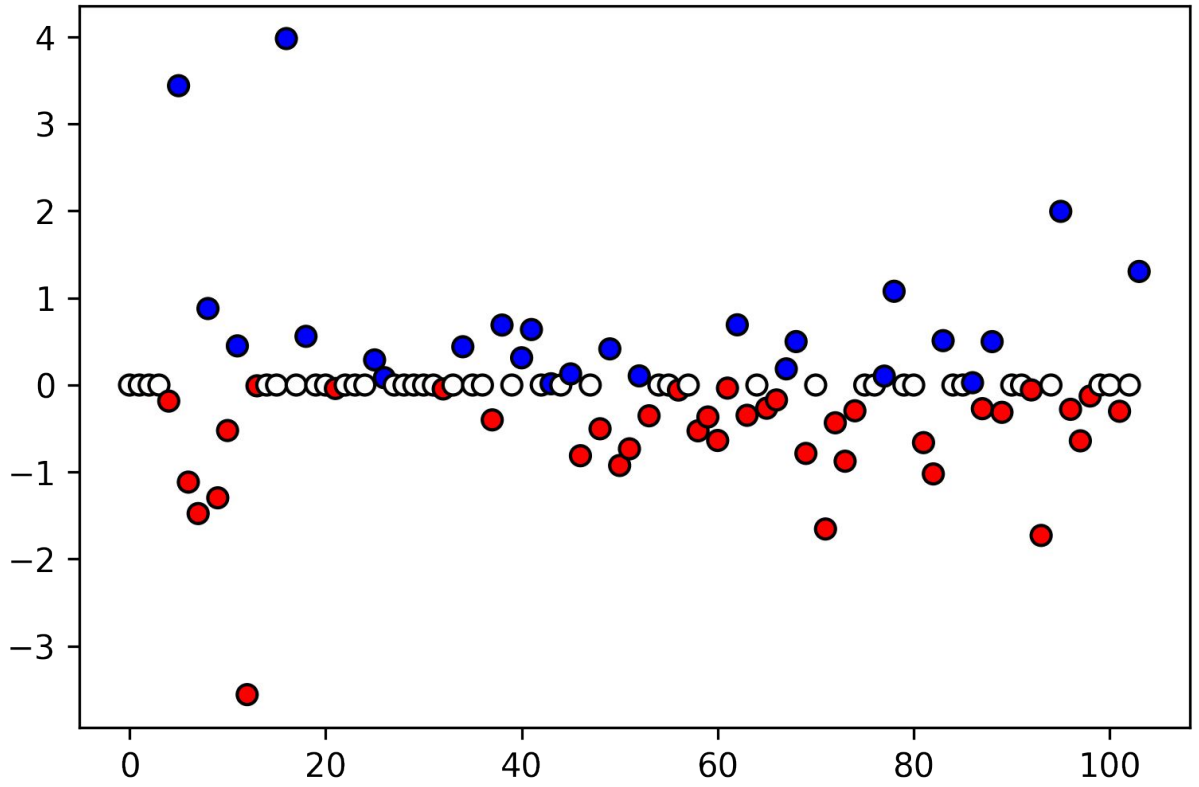
```
{'alpha': array([0.001, 0.002, 0.003, 0.006, 0.01 , 0.018, 0.032, 0.056, 0.1  ,
       0.178, 0.316, 0.562, 1.   ])}
```

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

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

```
{'alpha': 0.03162277660168379}
0.8600886917250629
```

Lasso coefficients of polynomial features




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

```
(506, 104)
63
```

Elastic Net

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

- Elastic-Net is a mixture of Ridge and Lasso, i.e., (compromise of both Ridge and Lasso regression)
  - The idea is to get the best of both worlds.
  - The alpha (α) controls how strongly we want to regularize.
- Requires tuning of additional parameter that distributes regularization penalty between L1 and L2
- More parameters to tune, but more flexible

Elastic Net 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 *(eta)* is the relative amount of L1 penalty (l1_ratio in the code).

- l1_ratio = 0 → Ridge
- l1_ratio = 1 → Lasso

So, don't set alpha to zero, it will be LR.

Grid-searching ElasticNet

```
1 from sklearn.preprocessing import PolynomialFeatures, scale
2 X, y = boston.data, boston.target
3 X_poly = PolynomialFeatures(include_bias=False).fit_transform(scale(X))
4 print(X_poly.shape)
5 X_train, X_test, y_train, y_test = train_test_split(X_poly, y, random_state=42)
6
7 param_grid = {'alpha': np.logspace(-4, -1, 10), 'l1_ratio': [0.01, .1, .5, .9, .98, 1]}
8 print(param_grid)
```

```
(506, 104)
{'alpha': array([0.    , 0.    , 0.    , 0.001, 0.002, 0.005, 0.01 , 0.022, 0.046,
       0.1  ]), 'l1_ratio': [0.01, 0.1, 0.5, 0.9, 0.98, 1]}
```

```
1 from sklearn.linear_model import ElasticNet
2 grid = GridSearchCV(ElasticNet(max_iter=1e6), param_grid, cv=10, return_train_score=True)
3 grid.fit(X_train, y_train)
```

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

```
{'alpha': 0.046415888336127774, 'l1_ratio': 0.1}
0.8587392730776561
```

```
1 grid.score(X_test, y_test)
```

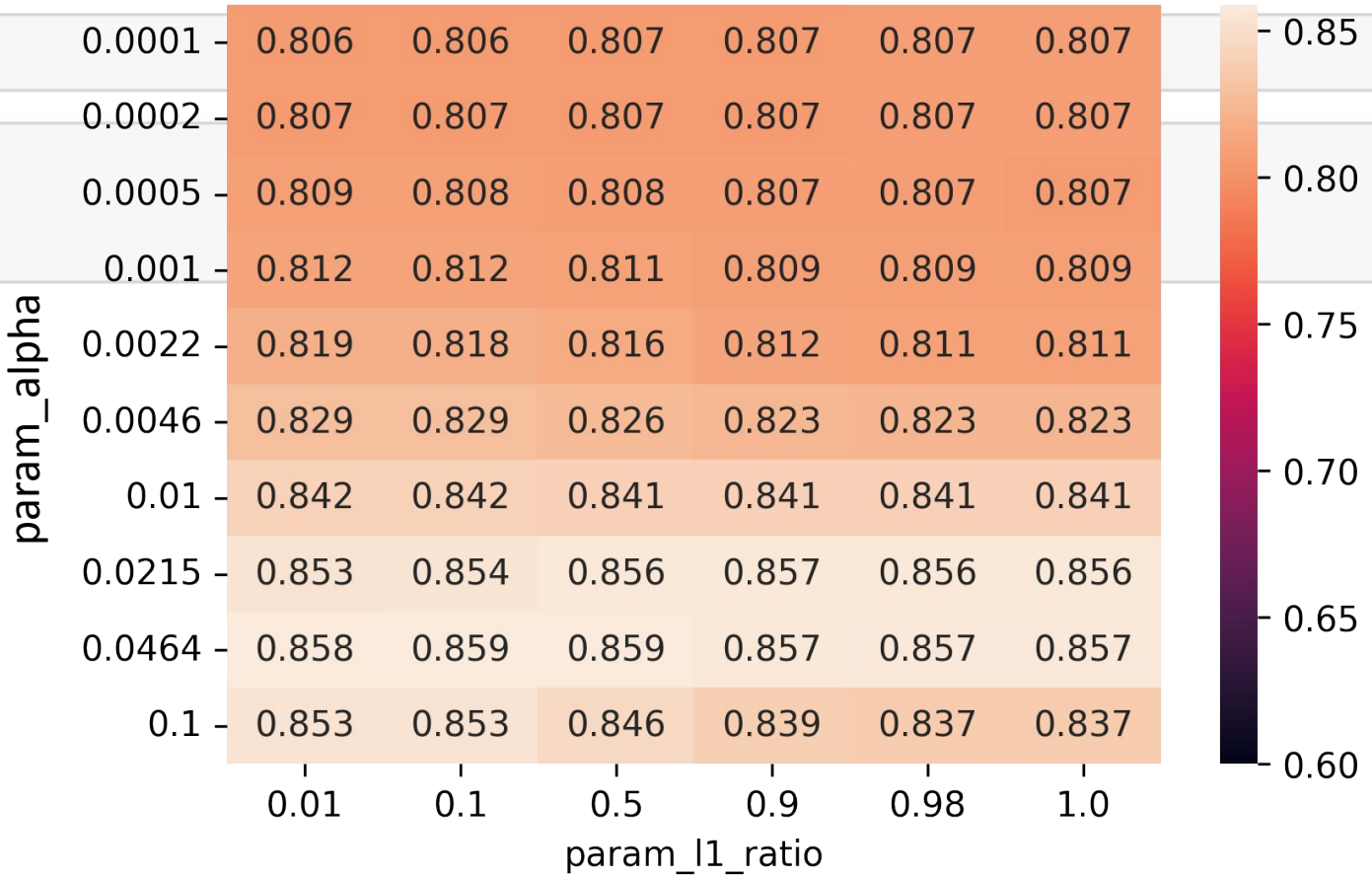
```
0.8042166021519406
```

Analyzing Grid-search results of CV Score (R^2)

```
import pandas as pd
res = pd.pivot_table(pd.DataFrame(grid.cv_results_), values='mean_test_score', index='param_alpha', columns='param_l1_ratio')
pd.set_option("display.precision",3)
res = res.set_index(res.index.values.round(4))
```

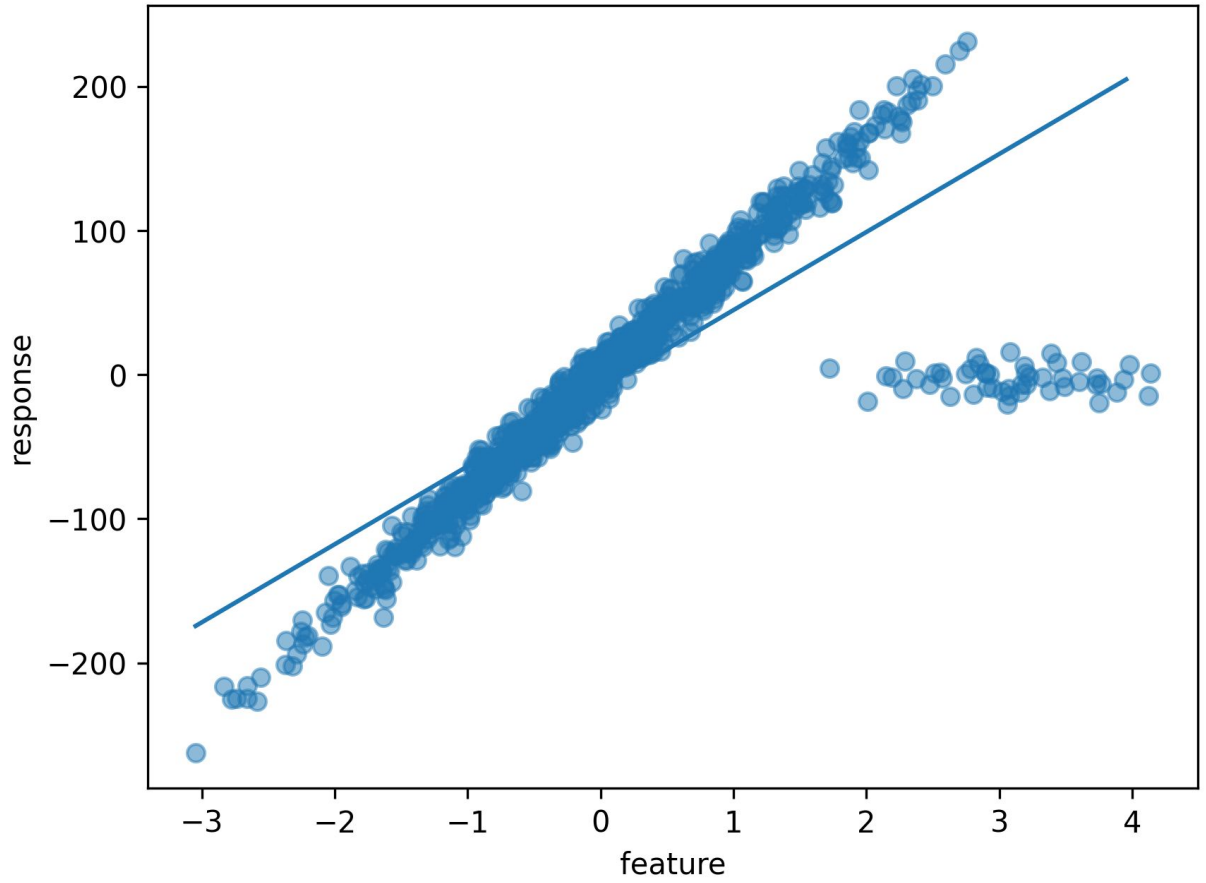
```
res
```

```
import seaborn as sns
sns.heatmap(res, annot=True, fmt=".3g", vmin=0.6)
#plt.savefig("elasticnet_search.png")
```



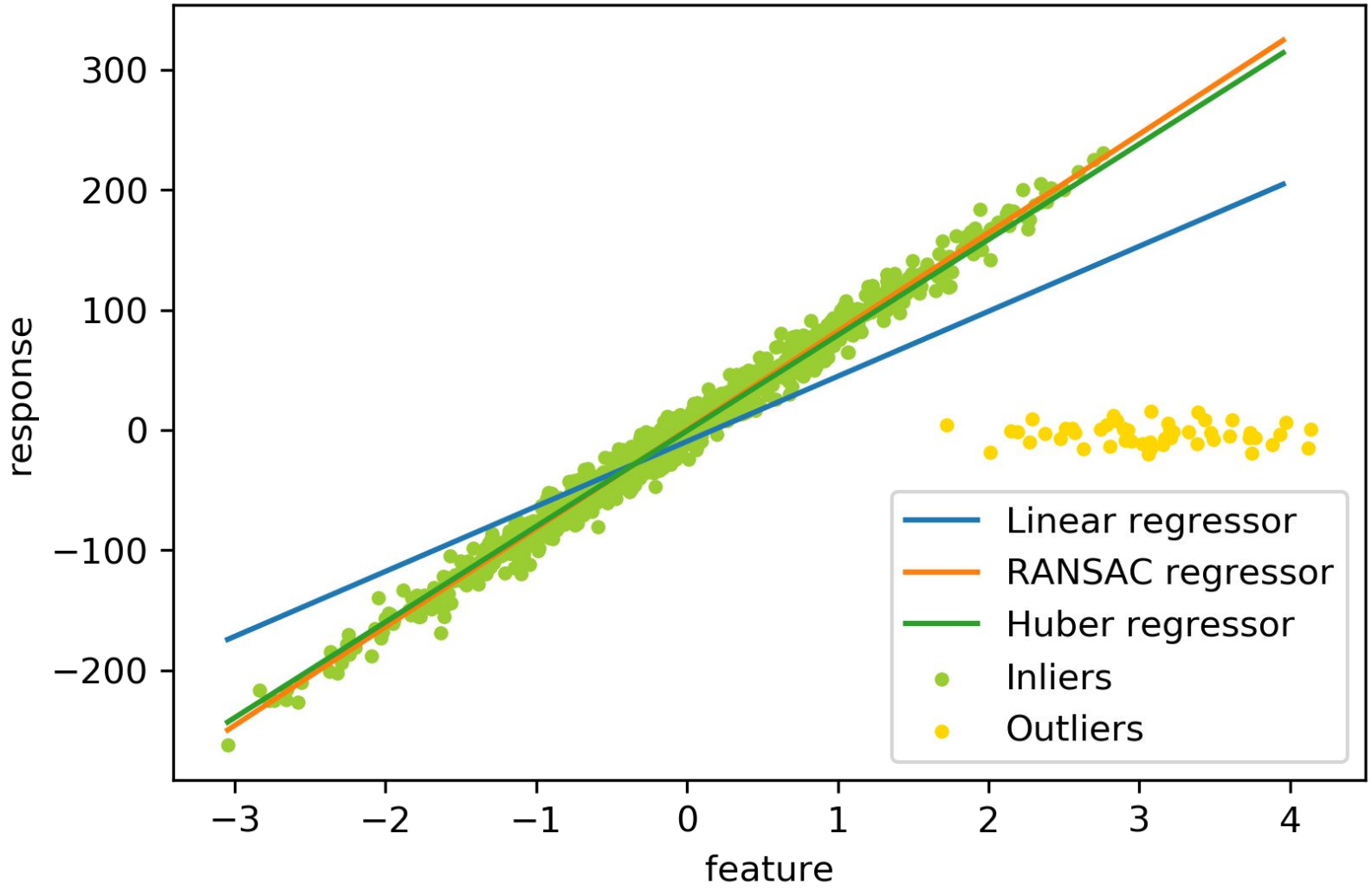
Robust Regression

Aims to fit a regression model in the presence of corrupt data: either outliers, or error in the model.



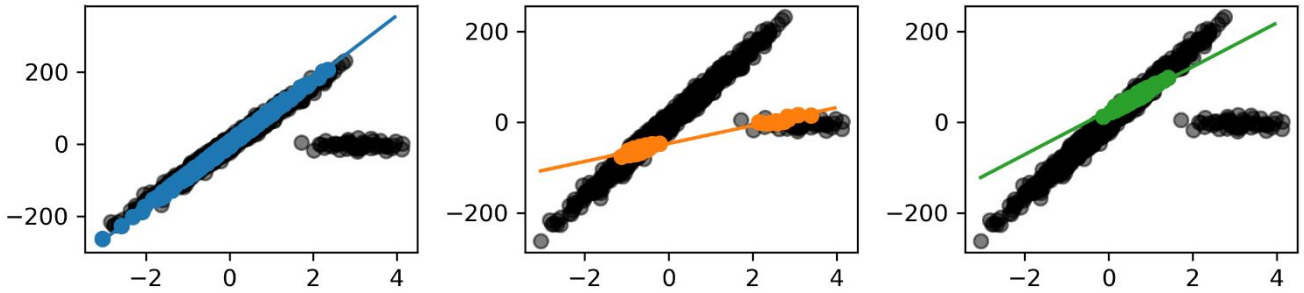
Least Squares Fit to Outlier Data

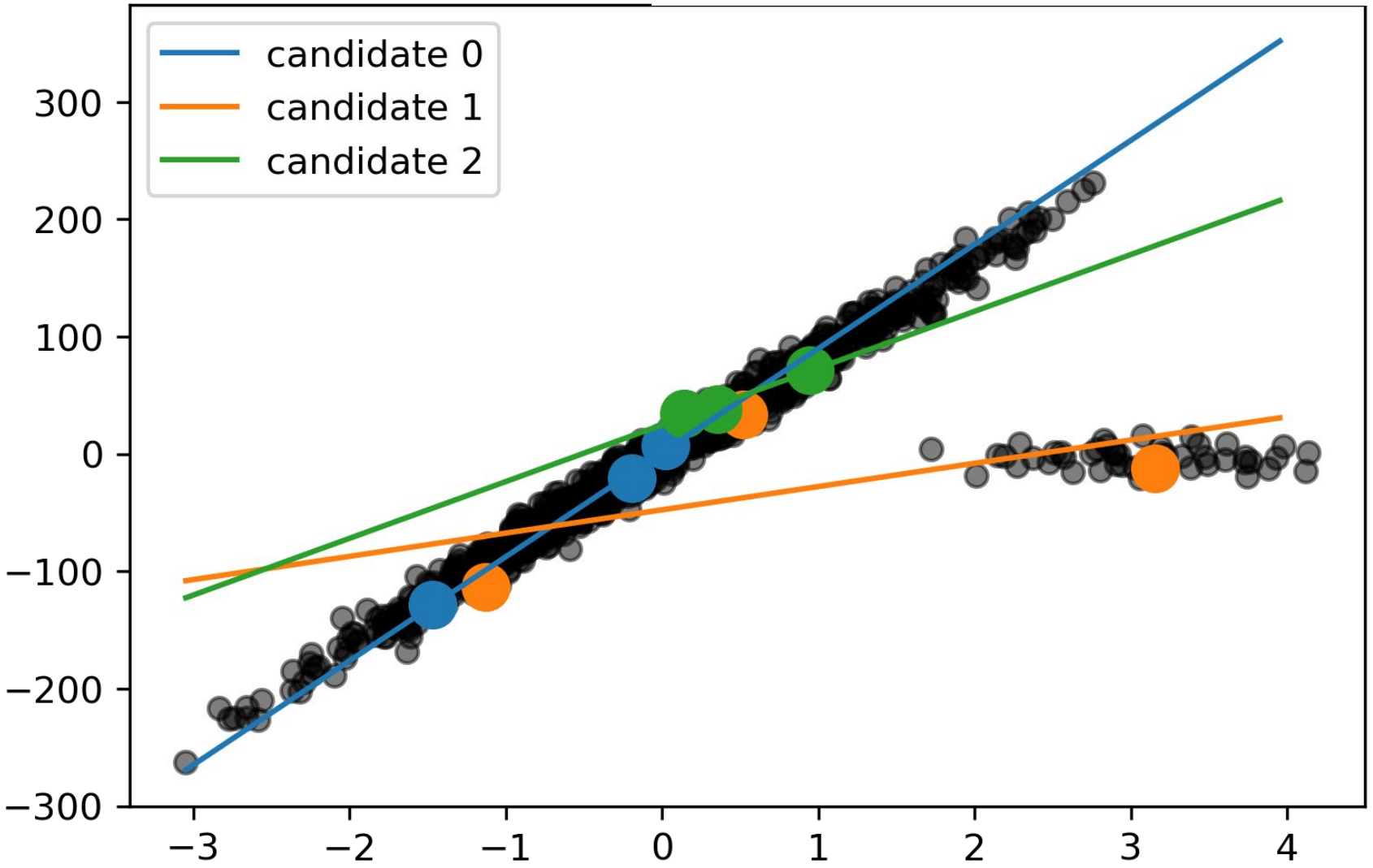
Robust Fit


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

Random Sample Consensus

Robust Fit - RANSAC





References

- ISLR Ch 3.
- Andreas C. Müller and Sarah Guido. ***Introduction to Machine Learning with Python: A Guide for Data Scientists***. O'Reilly Media; 1st edition.
- **ML501 Machine Learning**, Intel AI Academy, 2017.

Ridge

```
class sklearn.linear_model.Ridge(alpha=1.0, *, fit_intercept=True, copy_X=True,
max_iter=None, tol=0.0001, solver='auto', positive=False, random_state=None)
```
[source]

- Linear least squares with l2 regularization.
- Minimizes the objective function: `||y - Xw||^2_2 + alpha * ||w||^2_2`

Attributes:

- **coef_ndarray of shape (n_features,) or (n_targets, n_features):** Weight vector(s).
- **intercept_float or ndarray of shape (n_targets,):** Independent term in decision function. Set to 0.0 if `fit_intercept = False`.

Lasso

```
class sklearn.linear_model.Lasso(alpha=1.0, *, fit_intercept=True, precompute=False,
copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False,
random_state=None, selection='cyclic')
```

[source]

- Linear Model trained with L1 prior as regularizer (aka the Lasso).
- The optimization objective for Lasso is: `(1 / (2 * n_samples)) * ||y - Xw||^2_2 + alpha * ||w||_1`

Attributes:

- **coef_ndarray of shape (n_features,) or (n_targets, n_features):** Parameter vector (w in the cost function formula).
- **dual_gap_float or ndarray of shape (n_targets,):** Given param alpha, the dual gaps at the end of the optimization, same shape as each observation of y.
- **intercept_float or ndarray of shape (n_targets,):** Independent term in decision function.
- **n_iter_int or list of int:** Number of iterations run by the coordinate descent solver to reach the specified tolerance.

ElasticNet

```
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True,
precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False,
positive=False, random_state=None, selection='cyclic')
```

[source]

- Linear regression with combined L1 and L2 priors as regularizer.
- Minimizes the objective function:
- l1_ratio (default=0.5):
  - The ElasticNet mixing parameter, with 0 <= l1_ratio <= 1.
  - l1_ratio = 0 the penalty is an L2
  - l1_ratio = 1 it is an L1 penalty
  - For 0 < l1_ratio < 1, the penalty is a combination of L1 and L2.

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

ElasticNet

```
class sklearn.linear_model.ElasticNet(alpha=1.0, *, l1_ratio=0.5, fit_intercept=True,
precompute=False, max_iter=1000, copy_X=True, tol=0.0001, warm_start=False,
positive=False, random_state=None, selection='cyclic')
```

[source]

- Linear regression with combined L1 and L2 priors as regularizer.
- Minimizes the objective function:
- l1_ratio = 1 is the lasso penalty

```
1 / (2 * n_samples) * ||y - Xw||^2_2
+ alpha * l1_ratio * ||w||_1
+ 0.5 * alpha * (1 - l1_ratio) * ||w||^2_2
```

Attributes:

- **coef_ndarray of shape (n_features,) or (n_targets, n_features):** Parameter vector (w in the cost function formula).
- **dual_gap_float or ndarray of shape (n_targets,):** Given param alpha, the dual gaps at the end of the optimization, same shape as each observation of y.
- **intercept_float or ndarray of shape (n_targets,):** Independent term in decision function.
- **n_iter_int or list of int:** Number of iterations run by the coordinate descent solver to reach the specified tolerance.

RANSACRegressor

```
class sklearn.linear_model.RANSACRegressor(estimator=None, *, min_samples=None,
residual_threshold=None, is_data_valid=None, is_model_valid=None, max_trials=100,
max_skips=inf, stop_n_inliers=inf, stop_score=inf, stop_probability=0.99,
loss='absolute_error', random_state=None)
```

[source]

- RANSAC (RANdom SAmple Consensus) algorithm.
- RANSAC is an iterative algorithm for the robust estimation of parameters from a subset of inliers from the complete data set.
- Based on repeated random sub-sampling technique
- This is done by fitting linear models to several random samplings of the data and returning the model that has the best fit to a subset of the data.
- Assumption: The inliers tend to be more linearly related than a random mixture of inliers and outliers, a random subset that consists entirely of inliers will have the best model fit.

HuberRegressor

```
class sklearn.linear_model.HuberRegressor(*, epsilon=1.35, max_iter=100,
alpha=0.0001, warm_start=False, fit_intercept=True, tol=1e-05)
```

[source]

- L2-regularized linear regression model that is robust to outliers.
- alpha: Strength of the squared L2 regularization.
- The Huber Regressor optimizes the squared loss for the samples where |(y - Xw - c) / sigma| < epsilon (small residual) and the absolute loss for the samples where |(y - Xw - c) / sigma| > epsilon (large residual)
- The model coefficients w, the intercept c and the scale sigma are parameters to be optimized.
- The parameter sigma makes sure that if y is scaled up or down by a certain factor, one does not need to rescale epsilon to achieve the same robustness. Note that this does not take into account the fact that the different features of X may be of different scales.