Linear Regression

- [x] Simple
- [x] Multiple
- [x] polynomial
- [x] Bias vs. Variance Tradeoff
- [x] Regularization
- [x] Ridge
- [x] Lasso
- [x] Elastic Net

- [x] Normal Eqn
- [x] Gradient Descent

- [ ] practical Exercises
- [ ] Some Data prep
  - Encoding
  - Scaling

- Simple Linear Reg

$$y = w_0 + w_1 x$$

Get weights

Normal Eqn   LinearRegression()

Closed Solution $< 10^5$ observations

No iterations

No learning rate

→ Complexity ↑

Slower

No need for feature Scaling

Gradient Descent   SGDRegressor()

Iterative Solution $> 10^5$ observations

iterations

learning rate

Complexity ↓

faster

Need feature Scaling

Normal Eqn

Gradient Descent

LinearRegression

class sklearn.linear_model.LinearRegression(*, *fit_intercept*=True, *copy_X*=True, *tol*=1e-06, *n_jobs*=None, *positive*=False) [source]

Ordinary least squares Linear Regression.

LinearRegression 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.

SGDRegressor

class sklearn.linear_model.SGDRegressor(*loss*='squared_error', *, *penalty*='l2', *alpha*=0.0001, *l1_ratio*=0.15, *fit_intercept*=True, *max_iter*=1000, *tol*=0.001, *shuffle*=True, *verbose*=0, *epsilon*=0.1, *random_state*=None, *learning_rate*='invscaling', *eta0*=0.01, *power_t*=0.25, *early_stopping*=False, *validation_fraction*=0.1, *n_iter_no_change*=5, *warm_start*=False, *average*=False) [source]

Linear model fitted by minimizing a regularized empirical loss with SGD.

SGD stands for Stochastic Gradient Descent: the gradient of the loss is estimated each sample at a time and the model is updated along the way with a decreasing strength schedule (aka learning rate).

$$W = (X^T X)^{-1} X^T y$$

w_0 = intercept_

$[w]$ = coef_

iterative

Gradient Descent

$$J(w) = \underbrace{\frac{1}{m}\sum_{i=1}^{m}}_{\text{All data points}} \left(Y_{act}^{(i)} - w_0 - w_1 X^{(i)}\right)^2$$

* Initialize Random weights
* Loop:
  - Calculate grads $\partial J / \partial w$
  - update weights w

During loop
- All data points → Batch GD
- one data point → Stochastic GD
- Some data points (Random) → Mini-Batch GD

Multiple Linear Regression

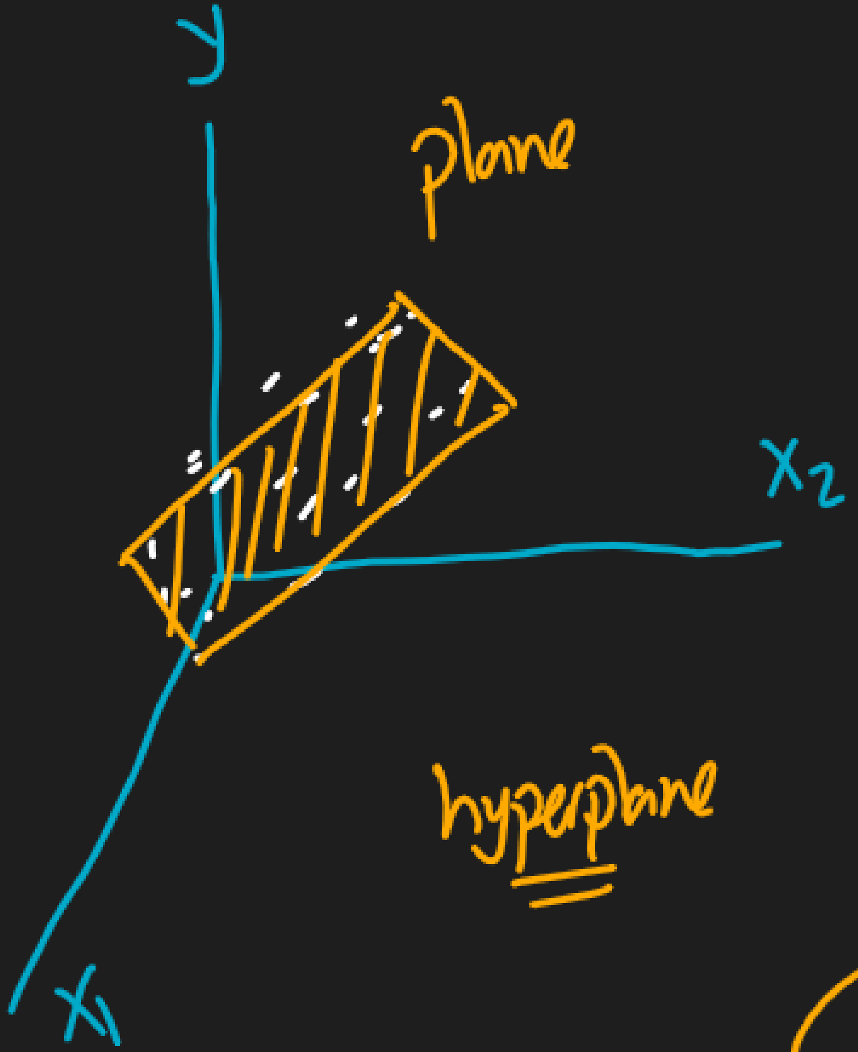
↳ Multiple Features

$$\underline{y} = w_0 x_0 + w_1 x_1 + w_2 x_2 + \dots + w_n x_n$$

$$y = W^T X$$

W: $[w_0 \;\; w_1 \dots w_n]$

X: $[\overset{1}{x_0} \;\; x_1 \dots x_n]$



hyperplane

$$\underset{y}{\text{House\_price}} = w_0 + w_1 * \overset{x_1}{\text{Med\_Income}} + w_2 * \overset{x_2}{\text{House\_Size}} + w_3 * \overset{x_3}{\text{No\_of\_Rooms}}$$

Linear Relation

x_1	x_2	x_3	y

$$y = f(X)$$

Multiple Linear Regression

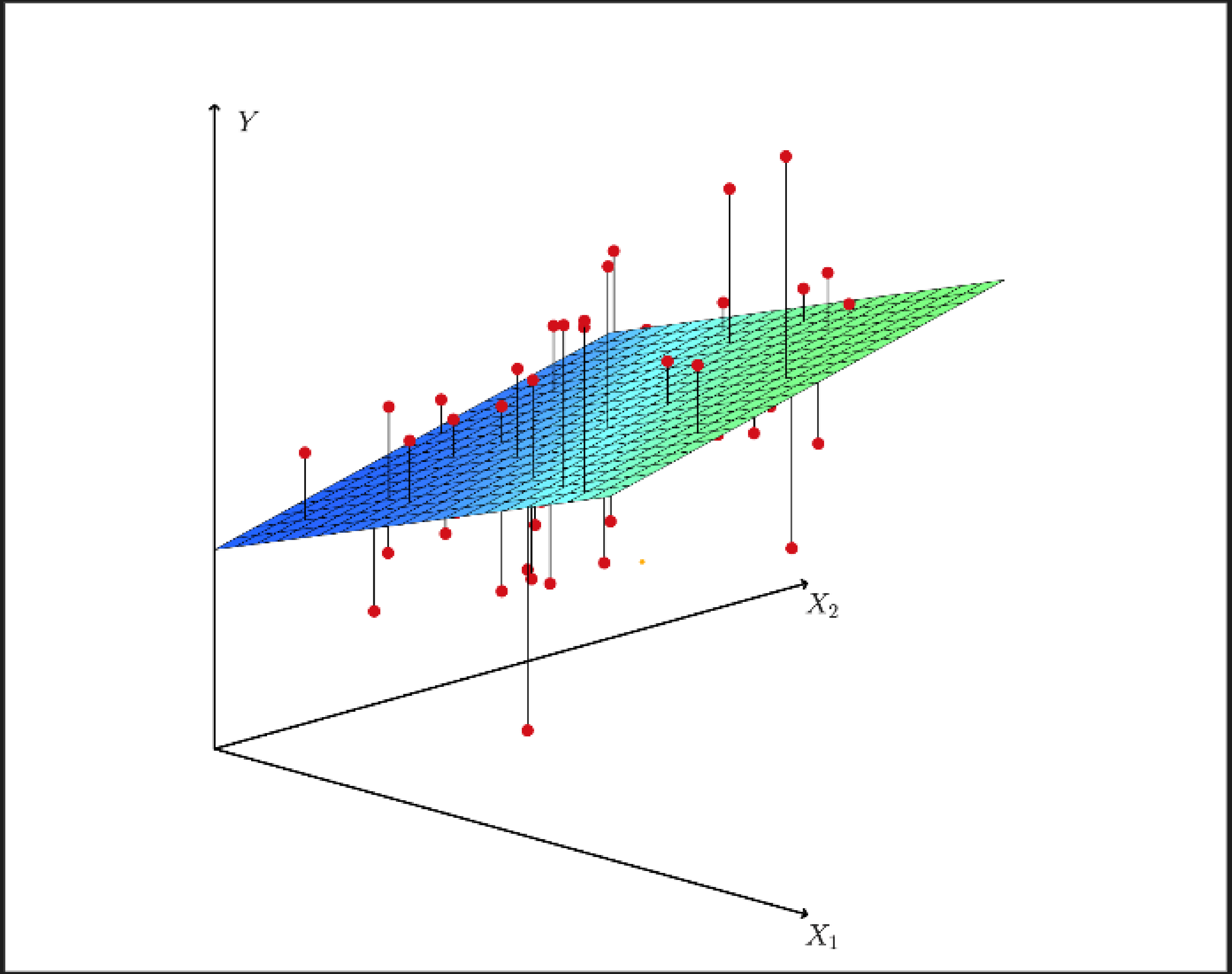
$y = w^T x$

↓

Get weights

Normal Eqn

Grad. Descent



Multiple Linear Regression

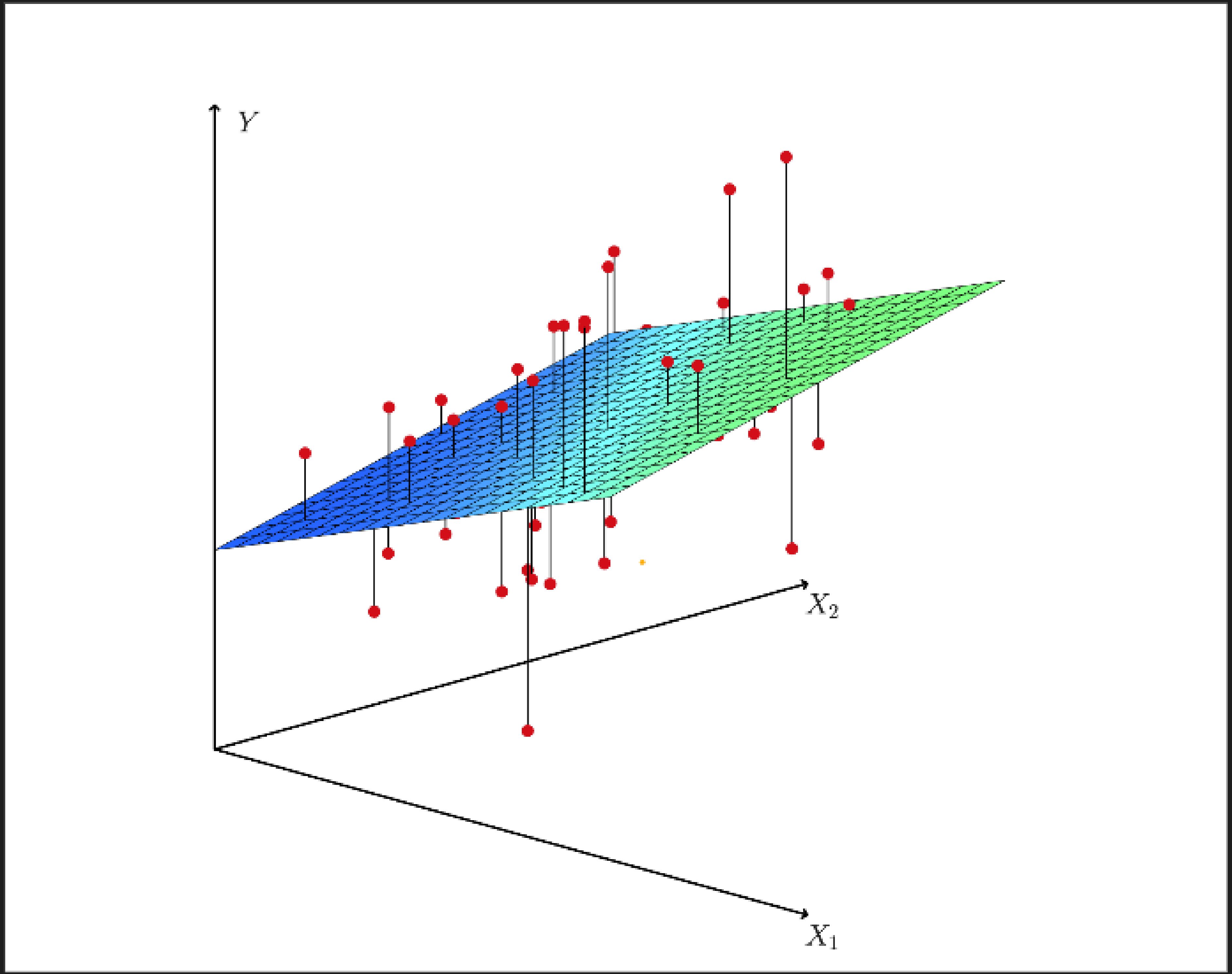
$$y = w^T x$$

$\downarrow$

Get weights

Normal Eqn

Grad. Descent



Room^3	Size^2	Income	$\boxed{\text{Size}}$	$\boxed{\text{Rooms}}$	price
X_4	X_3	X_1	X_2	X_3	y

$$\boxed{\begin{aligned} X_3 &= X_2^2 \\ X_4 &= X_3^3 \end{aligned}}$$

① Polynomial Transformation

$$y = w_0 + w_1 X_1 + w_2 X_2^2 + w_3 X_3^3$$ Non linear

$$y = w_0 + w_1 X_1 + w_2 X_3 + w_3 X_4$$ → Linear

② Linear Model

Sklearn → Polynomial Regression ✗

→ Polynomial features + Linear Regression ☑

Polynomial Linear Regression

PolynomialFeatures

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

Generate polynomial and interaction features.

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].

+

LinearRegression

```
class sklearn.linear_model.LinearRegression(*, fit_intercept=True, copy_X=True, tol=1e-06, n_jobs=None, positive=False)
```
[source]

Ordinary least squares Linear Regression.

LinearRegression 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.

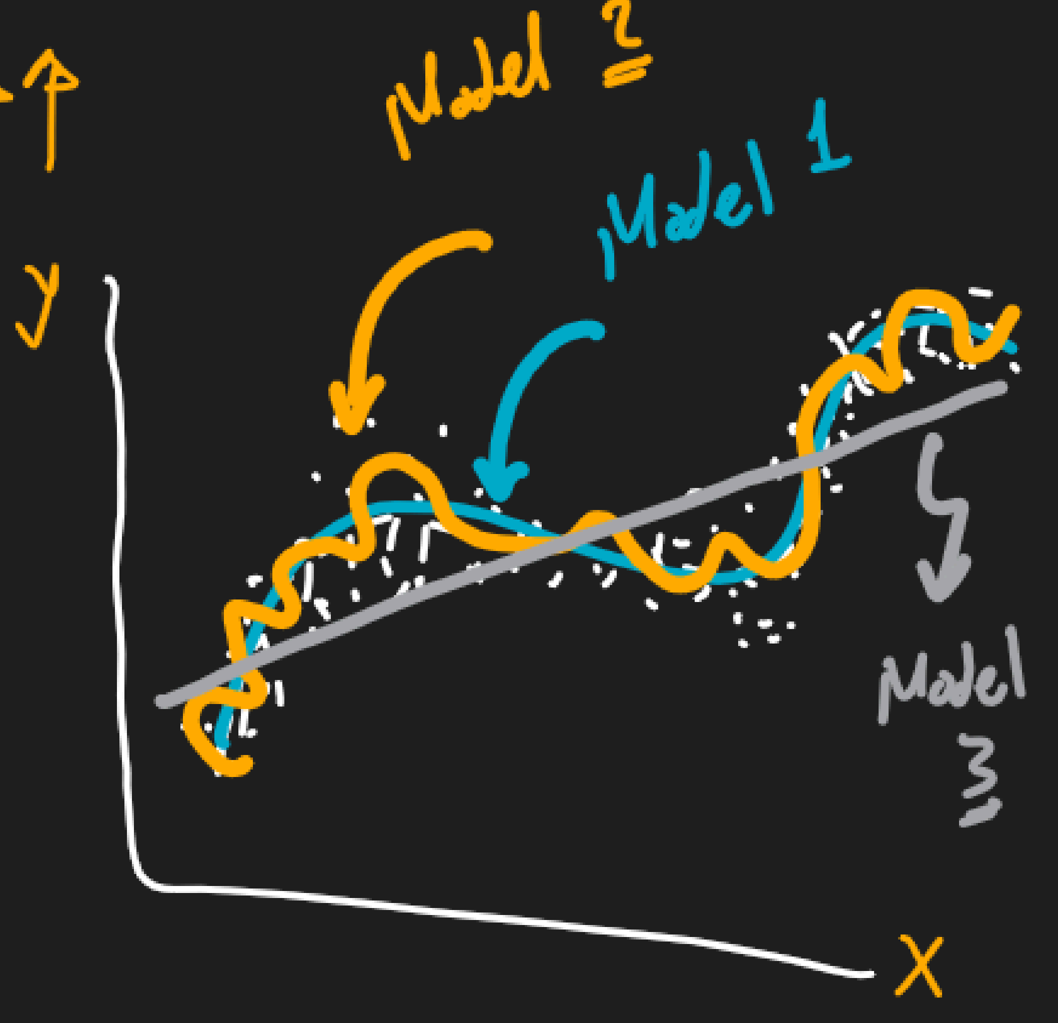
$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1 x_2^2 + w_5 x_1^2 x_2 + w_6 x_1^2 + w_7 x_2^2$$

Polynomial Degree ↑↑ → Model Complexity ↑↑

Variance ↑↑

Best fit Model ??

Model 3 | Model 1 | Model 2



Model 3

Training Error ↑↑

Testing Error ↑↑

Underfitting

High Bias

Model 1 ☑

Training Error →

Testing Error ↓

Good fit

Model 2

Training Error ↓↓

Testing Error ↑↑

Overfitting

High Variance

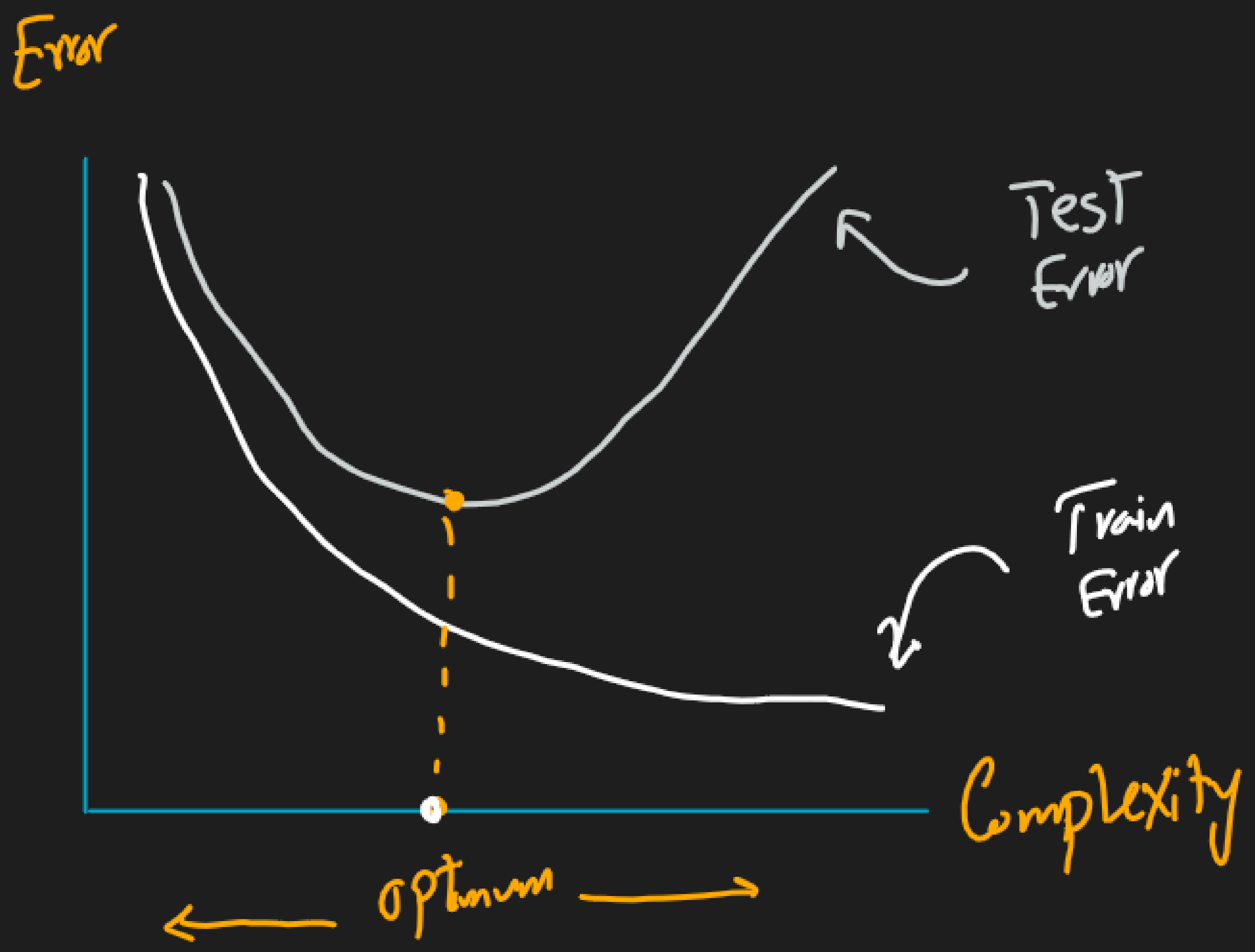
Data Complex

Model Complexity

Balance

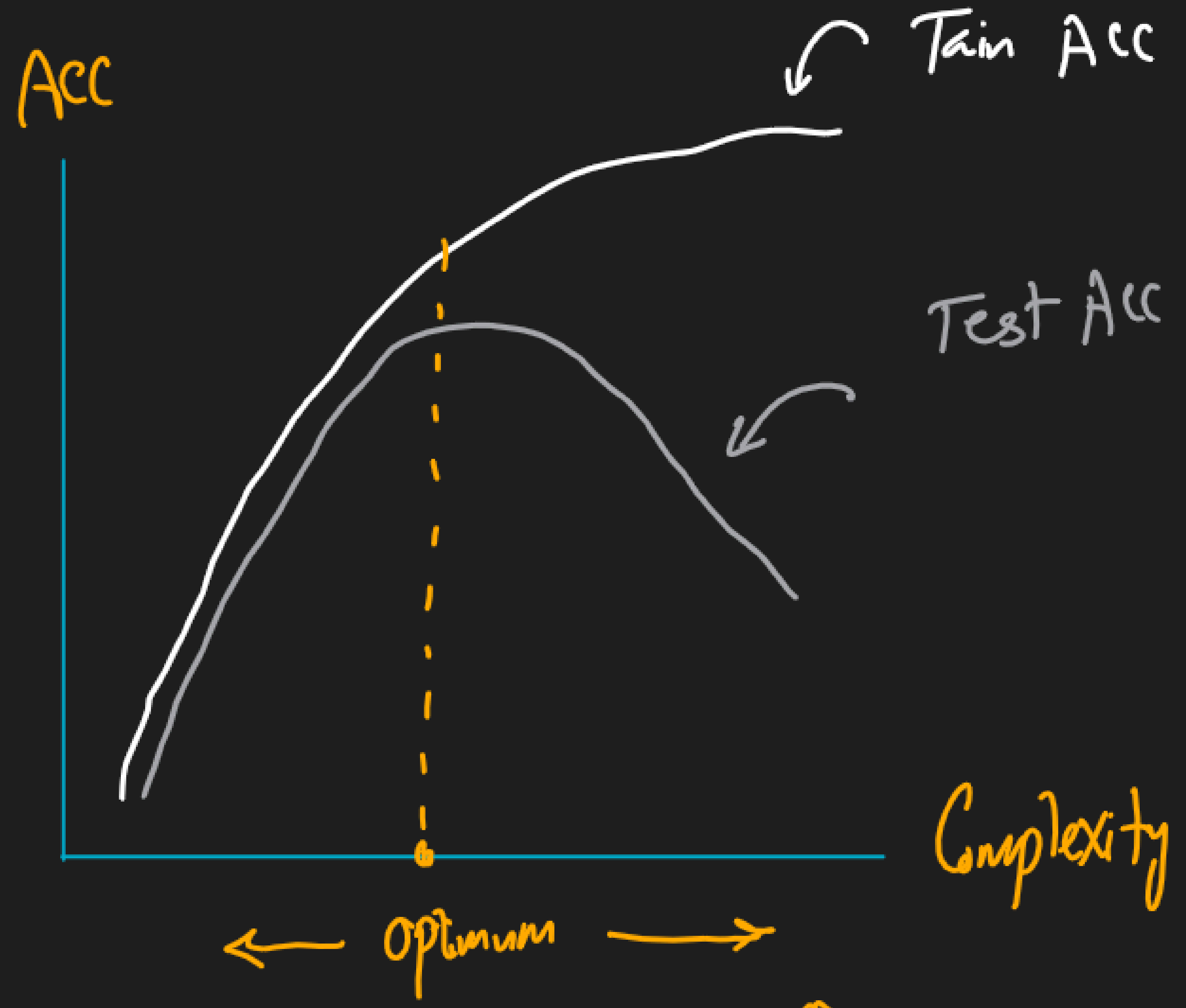
* Model Complexity >> Data Complexity → overfitting (High Variance)
* Model Complexity << Data Complexity → underfitting (High Bias)
* Model Complexity ≈ Data Complexity → Good fit

Bias vs. Varience Tradeoff



Train Error ↑
Test Error ↑
underfitting
(High Bias)

Train Error ↓
Test Error ↑
overfitting
(High variance)



Train Acc ↓
Test Acc ↓
underfitting
(High Bias)

Train Acc ↑
Test Acc ↓
overfitting
(High variance)

Model Complexity ↑↑ → Overfitting

↓

Regularization

Penalty

L1 L2

Neglect (reduce) Contribution

underfitting

$$y = \underbrace{\overbrace{W_0 + W_1 X_1 + W_2 X_2}^{\text{underfitting}} + W_3 X_1^2 + W_4 X_2^2}_{\text{Good fit}} + \underbrace{W_5 X_1^3 + W_6 X_2^3}_{\text{overfitting}}$$

~ 0 ~ 0

Good fit

overfitting

Weights → learned by Model

- Regularization with Gradient Descent

L2 Regularization

Ridge

$$J(\underline{W}) = MSE + \alpha \sum \|W\|^2$$

Regularization Term

$\alpha = 0$ → No regularization

$\alpha \uparrow$ → Regularized

$\alpha \uparrow\uparrow\uparrow$ → underfitting

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
```

Ridge

Linear Model with L2 Penalty

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
```

$$J(w) = MSE + \alpha \sum \|w\|$$

L1 Regularization

Lasso

↓

Linear Model with L1 Penalty

$$J(w) = MSE + \alpha_1 \sum \|w\|^2 + \alpha_2 \sum \|w\|$$

L2 L1

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:

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

Elastic Net

↳ Linear Model with L1 & L2 Penalty

L2 Penalty

$\alpha \uparrow \rightarrow w \downarrow$

(Impossible) $w \neq 0$

L1 Penalty

$\alpha \uparrow \rightarrow w \downarrow$

$w = 0$ (May be)

$\downarrow$

Feature Selection