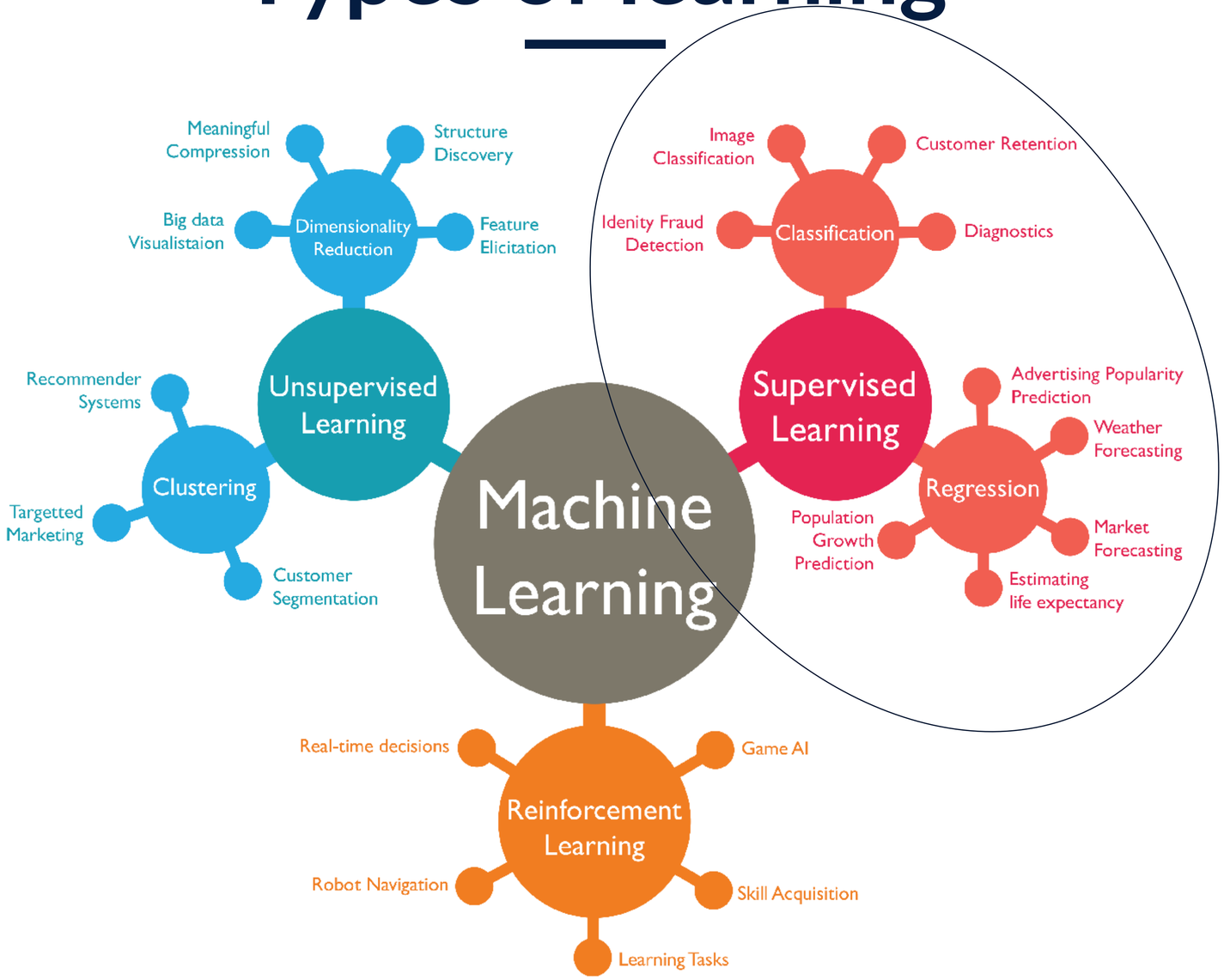
Brief recap

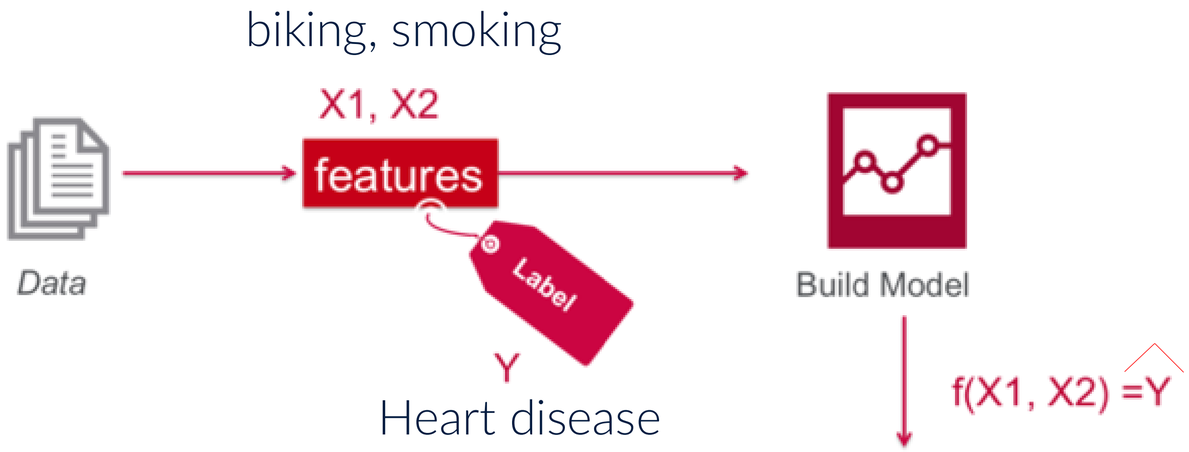
Supervised learning

Linear regression

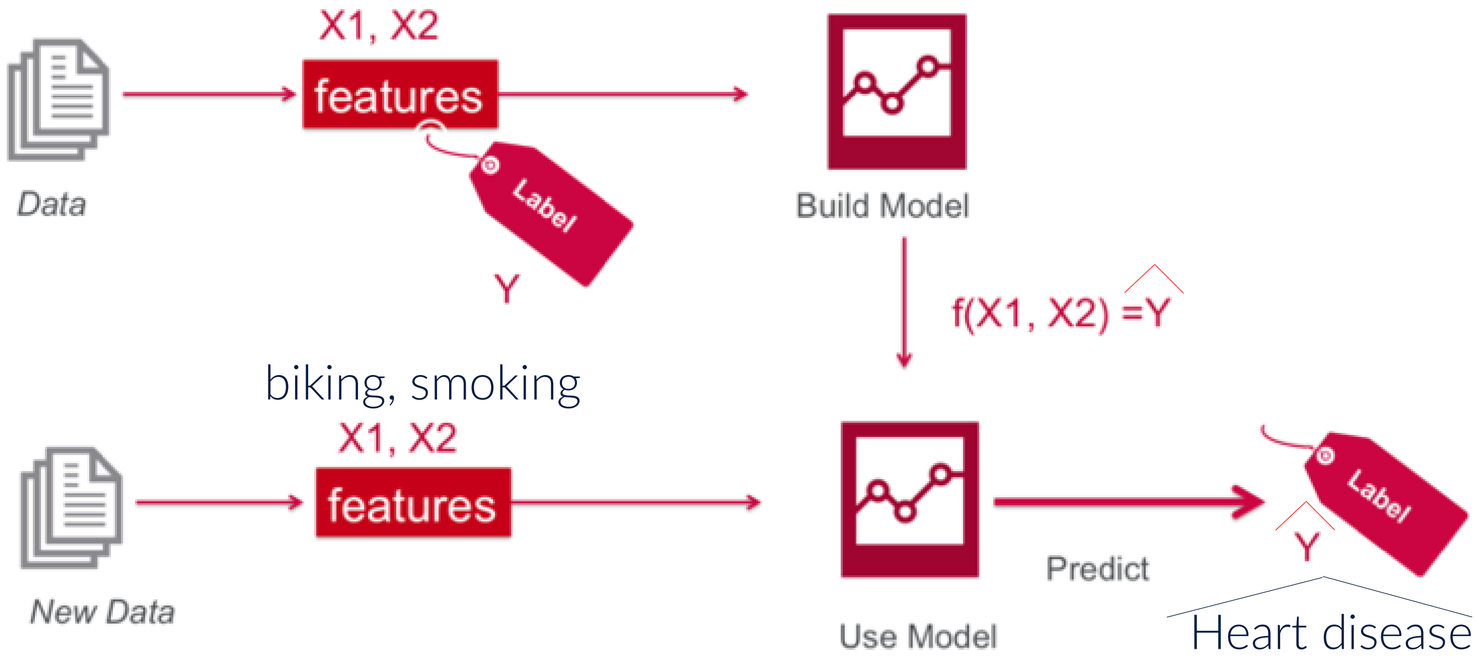
Types of learning



Supervised learning



Supervised learning



Linear regression

	biking (x_1)	smoking (x_2)	heart.disease (y)
0	30.801246	10.896608	11.769423
1	65.129215	2.219563	2.854081
2	1.959665	17.588331	17.177803
3	44.800196	2.802559	6.816647
4	69.428454	15.974505	4.062224

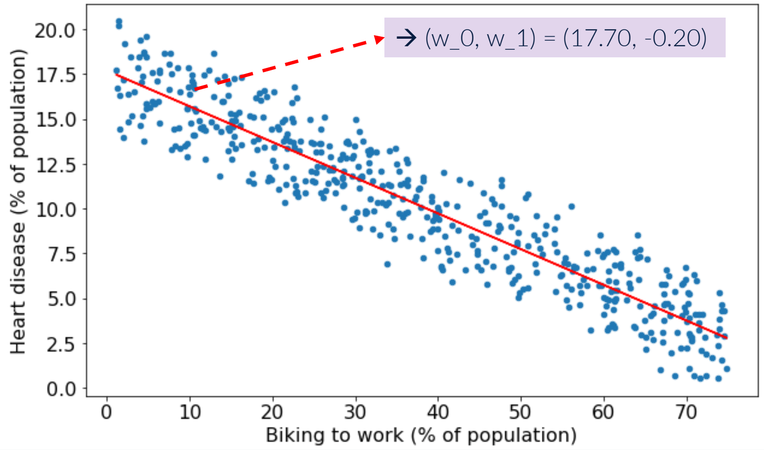
N

$$\text{heart.disease} = \underbrace{w_0 + w_1 \text{biking}}_{\widehat{\text{heart.disease}}} + \epsilon$$

$$\hat{y} = \tilde{\mathbf{X}}\mathbf{w} \text{ where, } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \text{ and } \tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_m^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_m^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^N & x_2^N & \dots & x_m^N \end{bmatrix}$$

OLS solution: $\mathbf{w}^* = \left(\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}}\right)^{-1}\tilde{\mathbf{X}}^\mathbf{T}\mathbf{y}$

Best line in least squares sense




```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)
```

Linear regression

$$\text{heart.disease} = \underbrace{w_0 1 + w_1 \text{biking} + w_2 \text{smoking}}_{\widehat{\text{heart.disease}}} + \epsilon$$

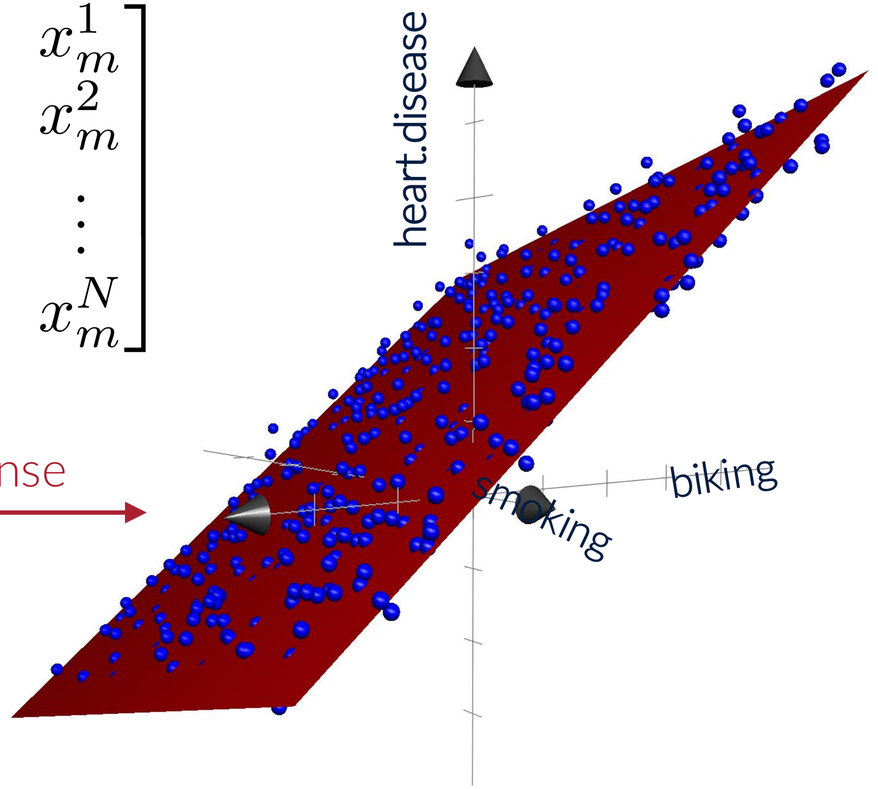
	biking (x_1)	smoking (x_2)	heart.disease (y)
0	30.801246	10.896608	11.769423
1	65.129215	2.219563	2.854081
2	1.959665	17.588331	17.177803
3	44.800196	2.802559	6.816647
4	69.428454	15.974505	4.062224

N

$$\hat{y} = \tilde{\mathbf{X}}\mathbf{w} \text{ where, } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \text{ and } \tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_1^1 & x_2^1 & \dots & x_m^1 \\ 1 & x_1^2 & x_2^2 & \dots & x_m^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^N & x_2^N & \dots & x_m^N \end{bmatrix}$$

OLS solution: $\mathbf{w}^* = \left(\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}}\right)^{-1}\tilde{\mathbf{X}}^\mathbf{T}\mathbf{y}$

Best plane in least squares sense




```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)
```

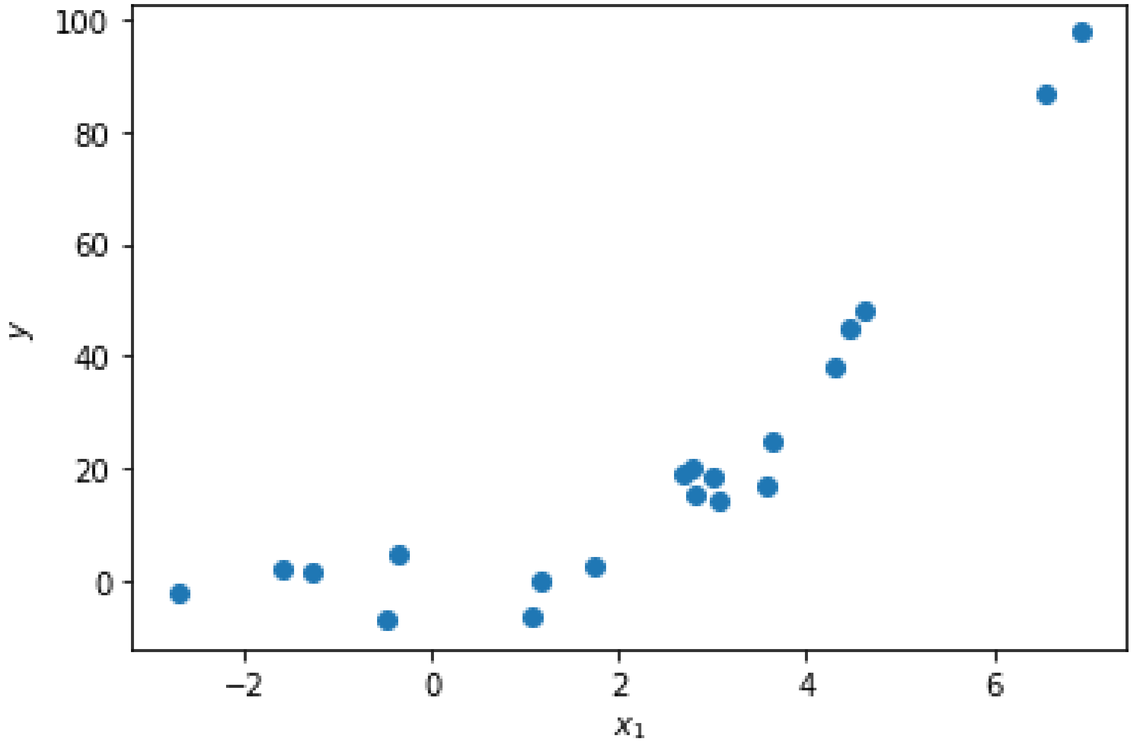
Polynomial regression

Illustrative example


```
np.random.seed(0)

X = 4 - 3 * np.random.normal(0, 1, 20)
X = np.sort(X)

y = 10 + X + 2*(X**2) + np.random.normal(0, 4, 20)

X = X.reshape(-1,1)
y = y.reshape(-1,1)
```

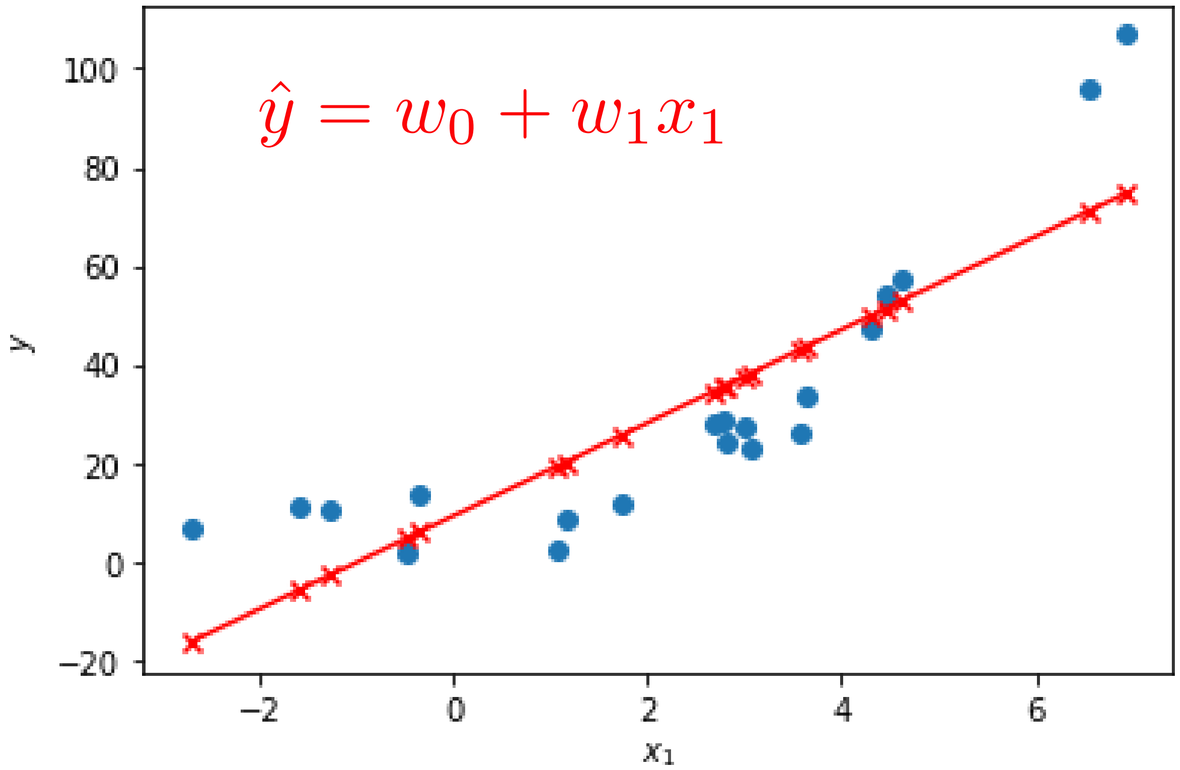
$$y = \underbrace{10 + x_1 + 2x_1^2}_{\widehat{y}} + \underbrace{\epsilon}_{\sim \mathcal{N}(0, 4^2)}$$

```
plt.scatter(X, y)

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```



Linear regression in (x_1, y)


```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

y_pred = model.predict(X)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r',marker='x')
plt.plot(X, y_pred, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```



At first sight, it is not a good model

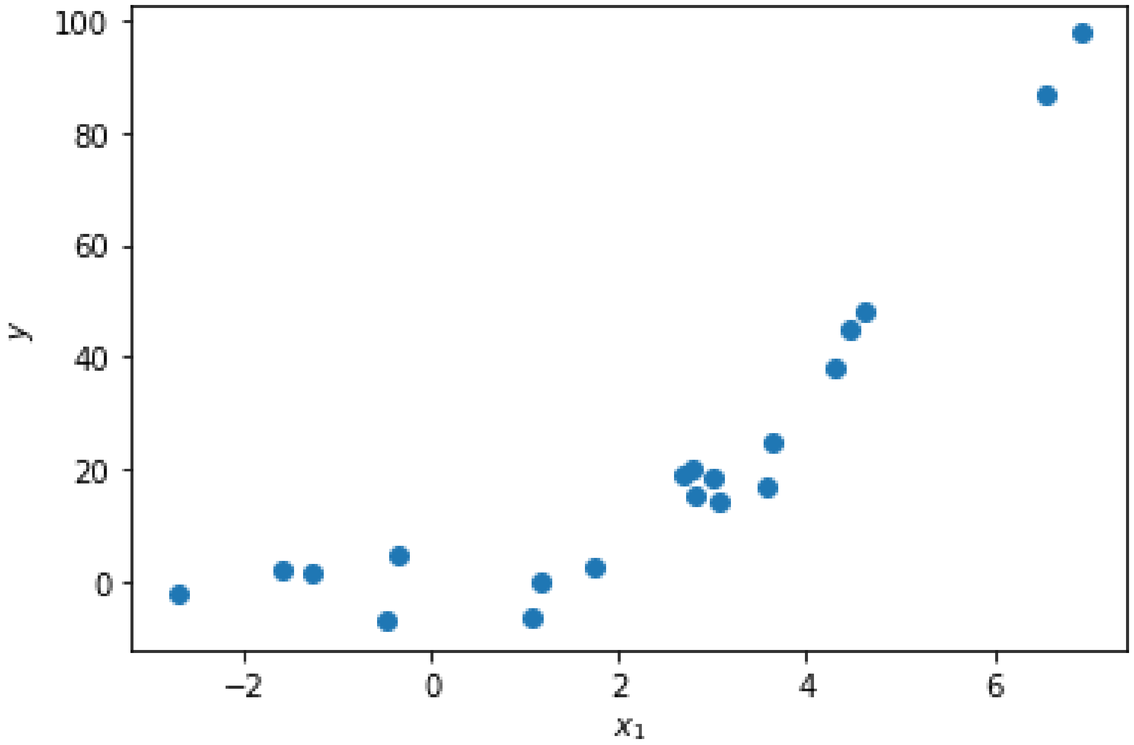
Hint: Observe the pattern of $\epsilon = (y - \hat{y})$

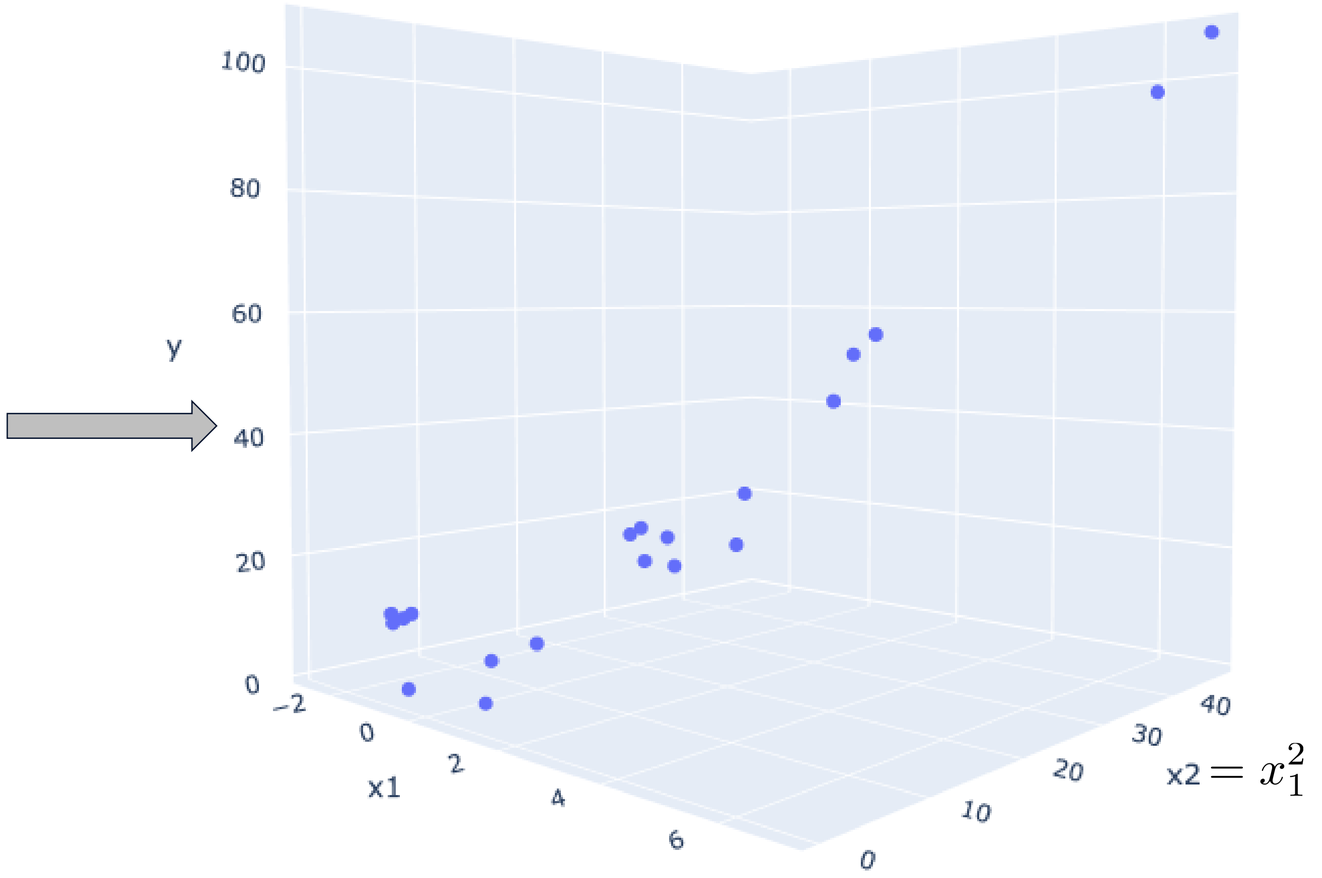
Expanding original feature space

- Observe that: $y = 10 + x_1 + 2x_1^2 + \epsilon$
- Can be written as: $y = 10 + x_1 + 2x_2 + \epsilon, \text{ with } x_2 = x_1^2$

- The original feature space $[x_1]$ has been expanded $[x_1, x_2]$
- We end up with a **linear regression problem** in (x_1, x_2, y)

Visual interpretation





OLS solution

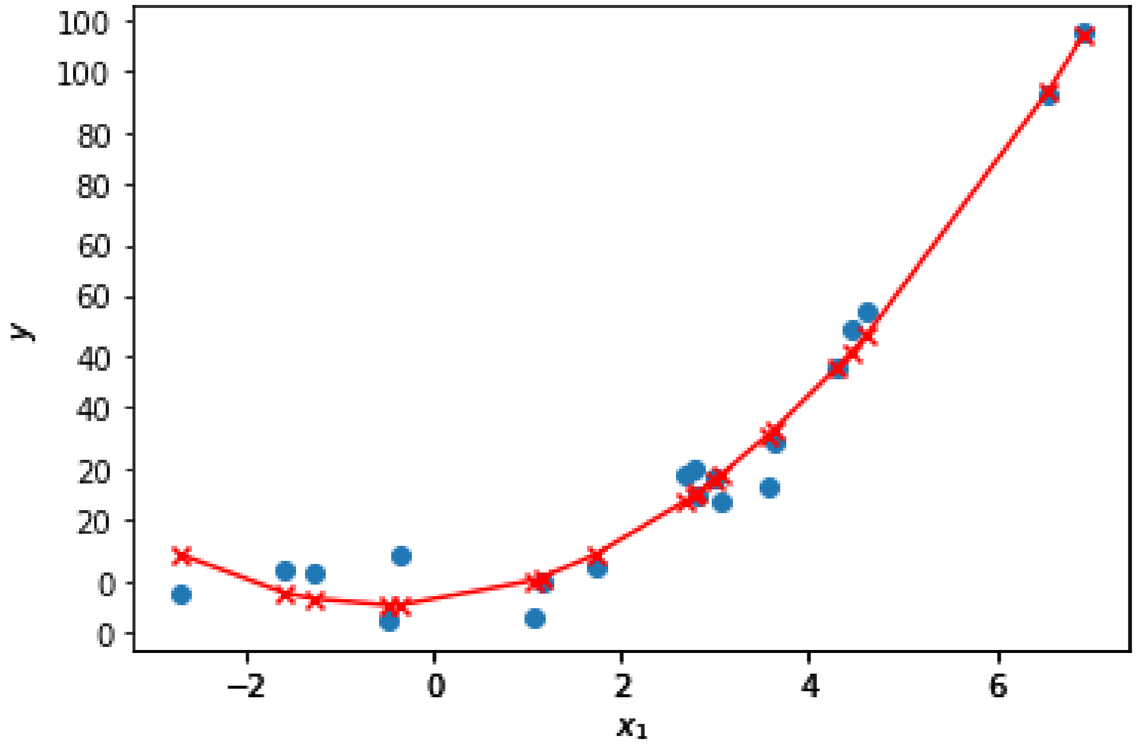
$$y = \underbrace{w_0 + w_1 x_1 + w_2 x_1^2}_{\hat{y}} + \epsilon$$

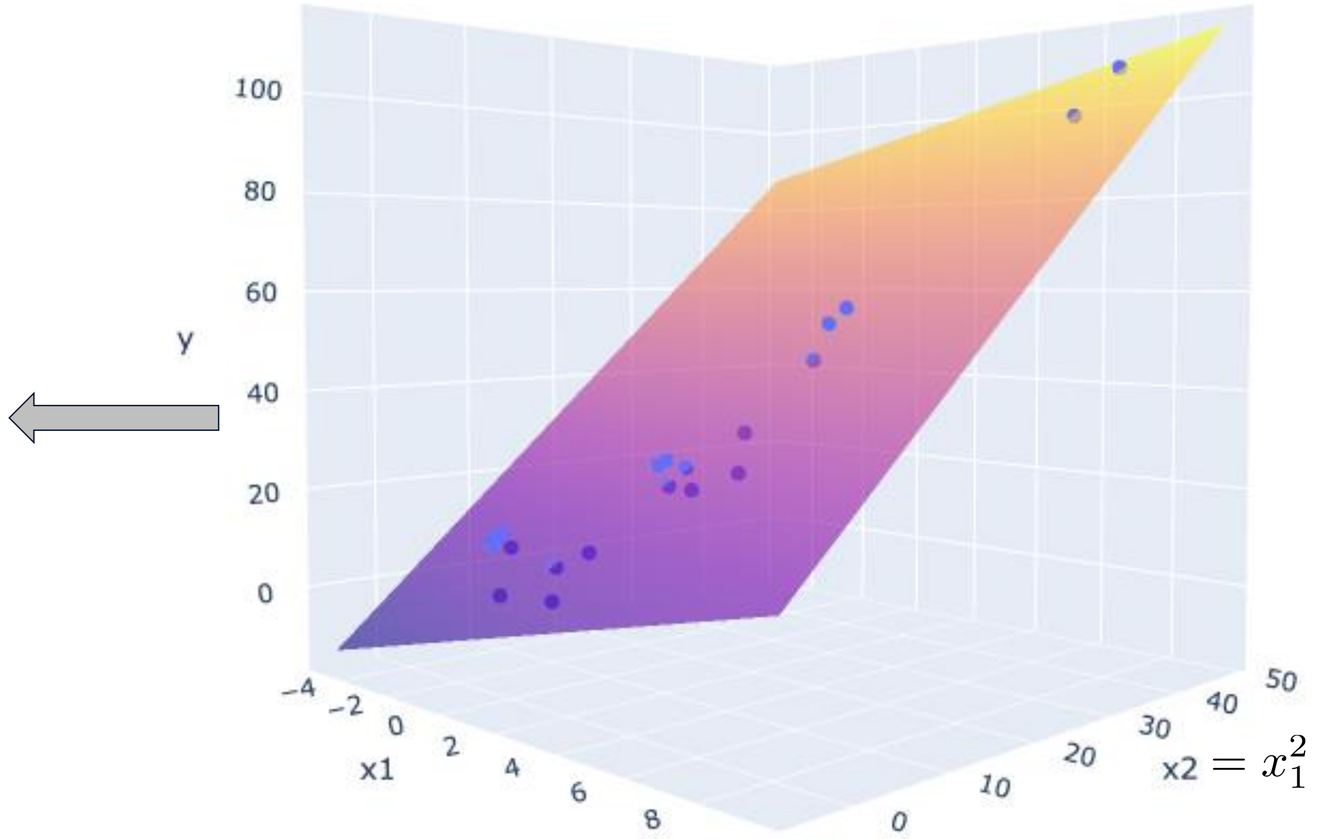
$$= \underbrace{w_0 + w_1 x_1 + w_2 x_2}_{\hat{y}} + \epsilon, \text{ with } x_2 = x_1^2$$

$$\hat{y} = \tilde{\mathbf{X}}\mathbf{w} \text{ where, } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \text{ and } \tilde{\mathbf{X}} = \begin{bmatrix} 1 & x_1^1 & x_2^1 \\ 1 & x_1^2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_1^N & x_2^N \end{bmatrix}$$

$$\mathbf{w}^* = \left(\tilde{\mathbf{X}}^\mathbf{T}\tilde{\mathbf{X}}\right)^{-1}\tilde{\mathbf{X}}^\mathbf{T}\mathbf{y}$$

Visual interpretation





Creating polynomial features

- Note that the **true parametric form** is not necessarily known
- And it's tedious to generate all sorts of polynomial features manually
- The library scikit-learn has a method to generate features automatically

```
from sklearn.preprocessing import PolynomialFeatures

polynomial_features= PolynomialFeatures(degree=2, include_bias=False)

X_transformed = polynomial_features.fit_transform(X)
```

- Say $X = [x_1, x_2]$. That is, **m**=2 independent variables (or features)

➢ For **degree**=2, $X\_\text{transformed} = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$

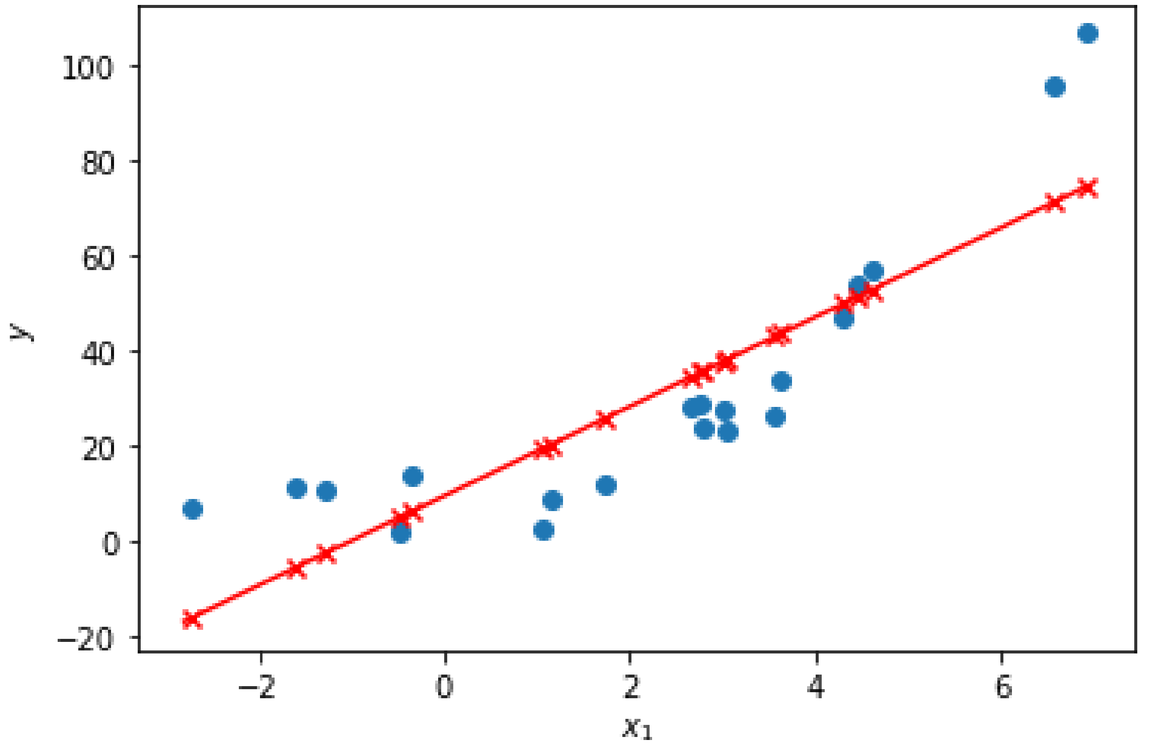
"interaction" feature

⚠ Dimensionality of feature space; e.g. (m=8, degree=5) -> 1286 features

OLS solution in Python


```
model = LinearRegression()
model.fit(X_transformed, y)

y_pred = model.predict(X_transformed)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r', marker='x')

x = np.arange(-2.8, 7.0, 0.05).reshape(-1,1)
x_transformed = polynomial_features.fit_transform(x)
y_pred_x = model.predict(x_transformed)

plt.plot(x_1, y_pred_x, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```

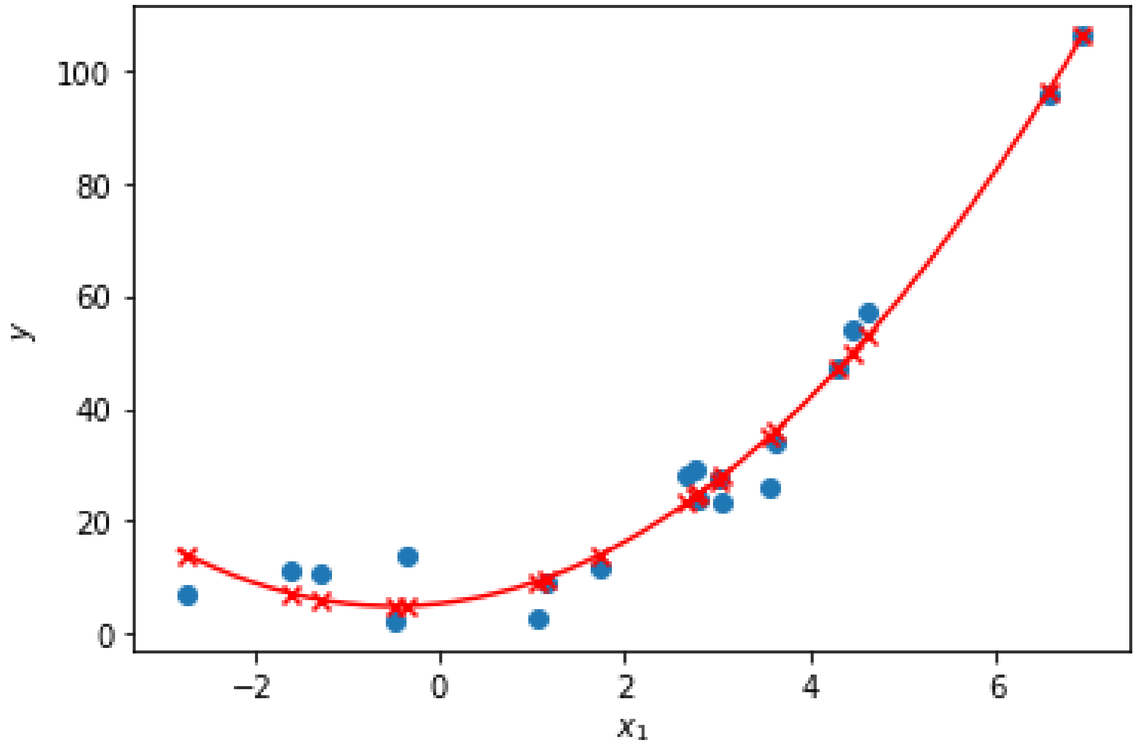
PolynomialFeatures(degree=1, include_bias=False)



OLS solution in Python


```
model = LinearRegression()
model.fit(X_transformed, y)

y_pred = model.predict(X_transformed)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r', marker='x')

x = np.arange(-2.8, 7.0, 0.05).reshape(-1,1)
x_transformed = polynomial_features.fit_transform(x)
y_pred_x = model.predict(x_transformed)

plt.plot(x_1, y_pred_x, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```

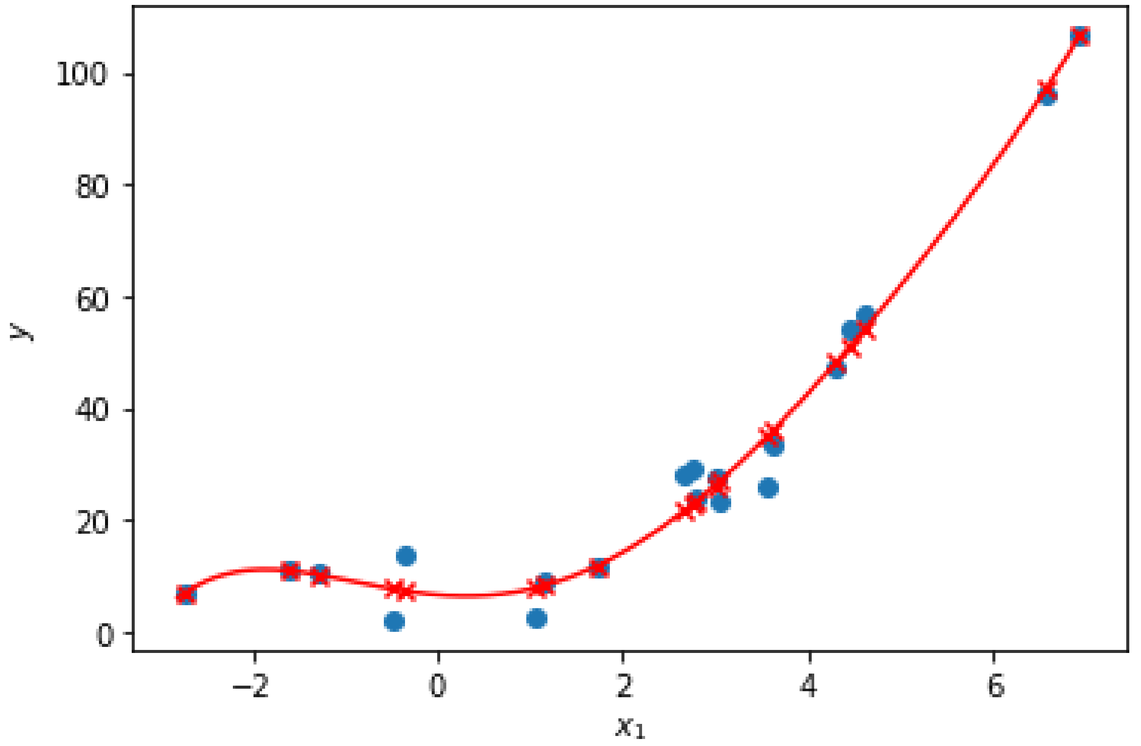
PolynomialFeatures(degree=2, include_bias=False)



OLS solution in Python


```
model = LinearRegression()
model.fit(X_transformed, y)

y_pred = model.predict(X_transformed)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r', marker='x')

x = np.arange(-2.8, 7.0, 0.05).reshape(-1,1)
x_transformed = polynomial_features.fit_transform(x)
y_pred_x = model.predict(x_transformed)

plt.plot(x_1, y_pred_x, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```

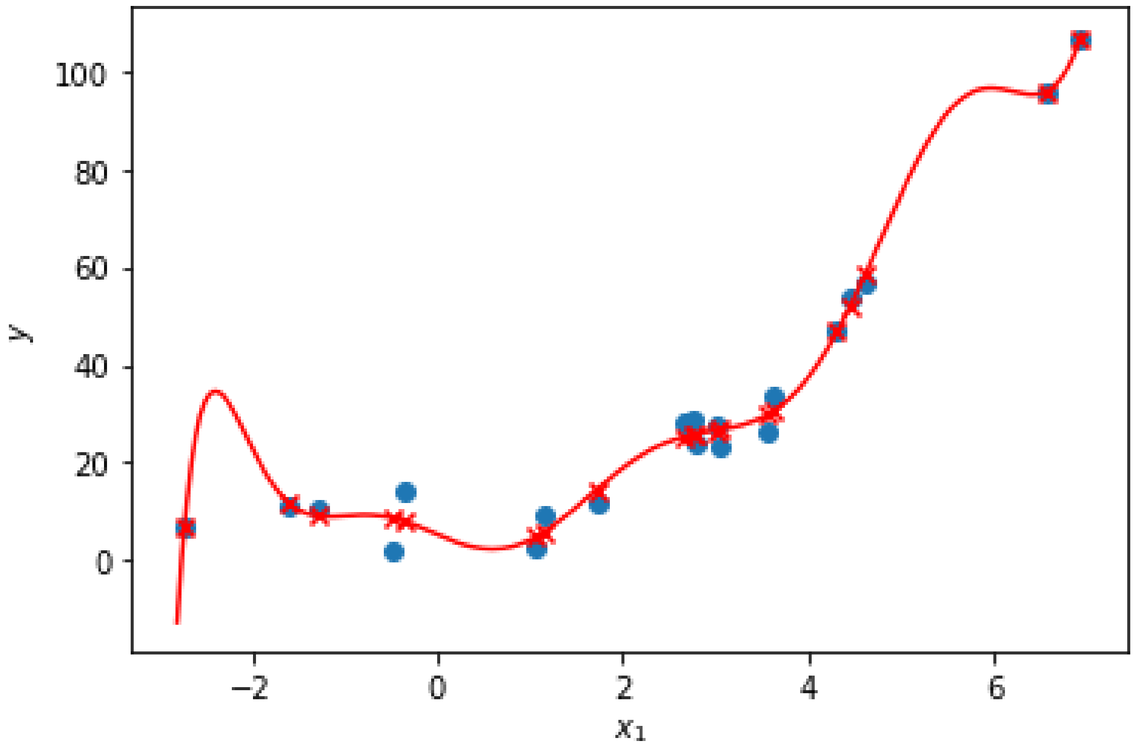
PolynomialFeatures(degree=5, include_bias=False)



OLS solution in Python


```
model = LinearRegression()
model.fit(X_transformed, y)

y_pred = model.predict(X_transformed)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r', marker='x')

x = np.arange(-2.8, 7.0, 0.05).reshape(-1,1)
x_transformed = polynomial_features.fit_transform(x)
y_pred_x = model.predict(x_transformed)

plt.plot(x_1, y_pred_x, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```

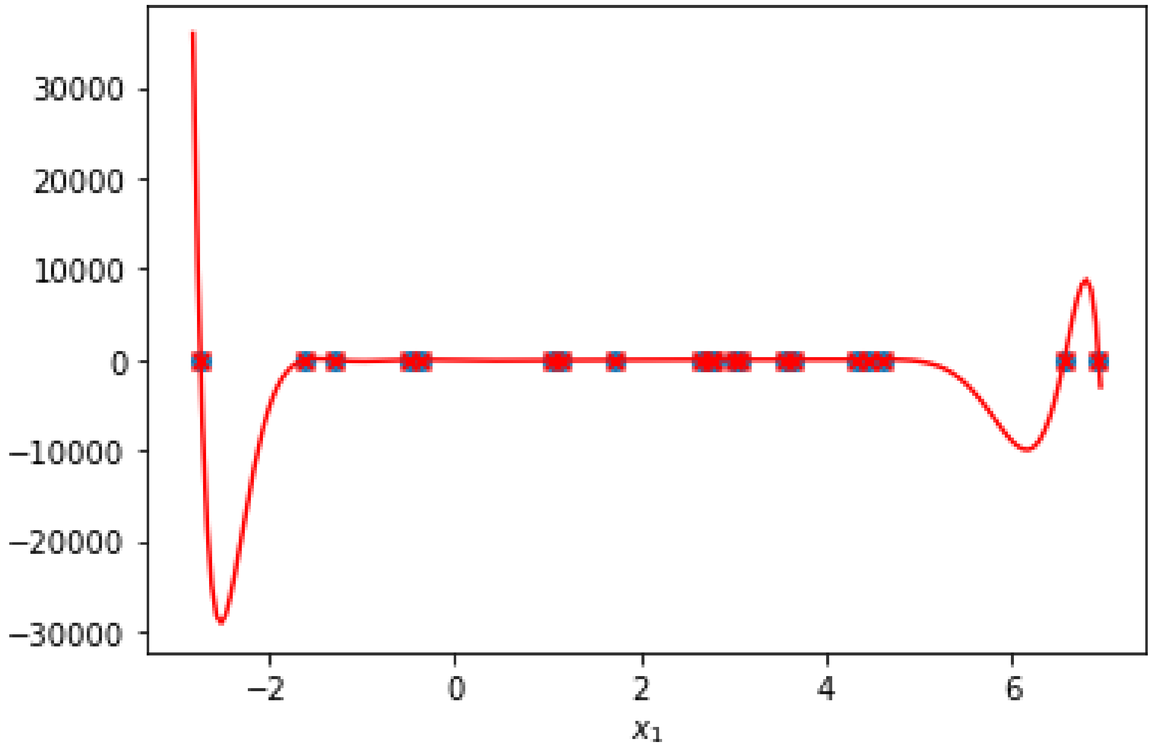
PolynomialFeatures(degree=10, include_bias=False)



OLS solution in Python

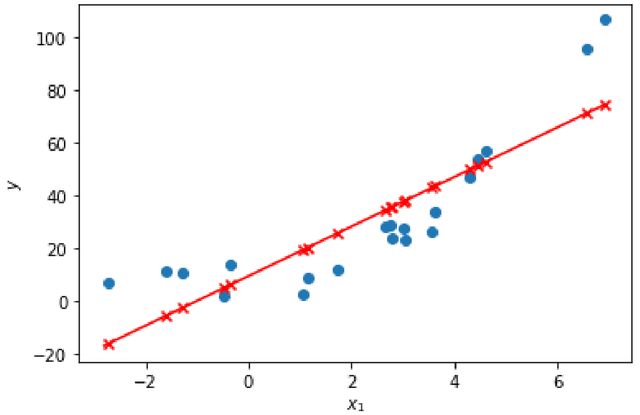

```
model = LinearRegression()
model.fit(X_transformed, y)

y_pred = model.predict(X_transformed)

plt.scatter(X, y)
plt.scatter(X, y_pred, color='r', marker='x')

x = np.arange(-2.8, 7.0, 0.05).reshape(-1,1)
x_transformed = polynomial_features.fit_transform(x)
y_pred_x = model.predict(x_transformed)

plt.plot(x_1, y_pred_x, color='r')

plt.xlabel("$x_1$")
plt.ylabel("$y$")

plt.show()
```

PolynomialFeatures(degree=15, include_bias=False)

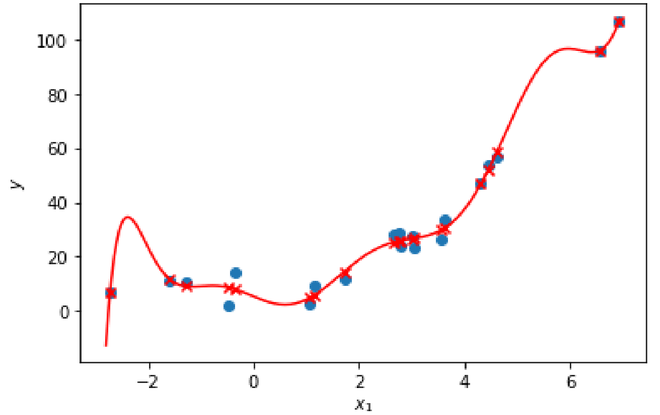


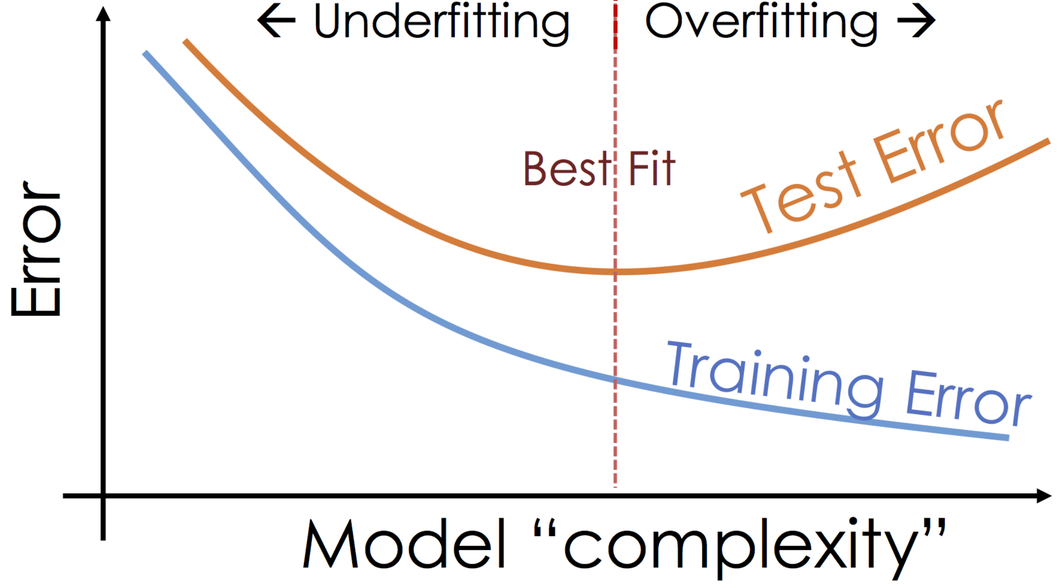
Underfitting & Overfitting

PolynomialFeatures(degree=1, include_bias=False)



PolynomialFeatures(degree=10, include_bias=False)





→ Use train-test split procedure

Preventing overfitting via regularization

- We penalize the **cost function** by adding a penalty that regularizes or shrinks the coefficient estimates w_i^* to zero
- Reminder: the cost function for linear regression in the least square sense

$$g(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^i - \tilde{\mathbf{X}}^i\mathbf{w}\right)^2$$

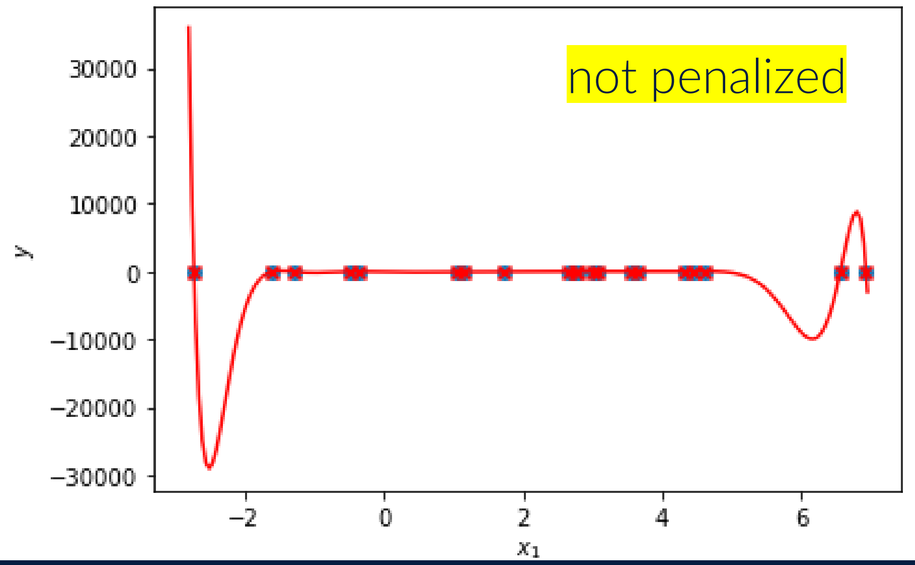
- Let's add a L_1 penalty term to induce fewer non-zero coefficients

$$g(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}\left(y^i - \tilde{\mathbf{X}}^i\mathbf{w}\right)^2 + \alpha\sum_{j=1}^{m}|w_j|$$

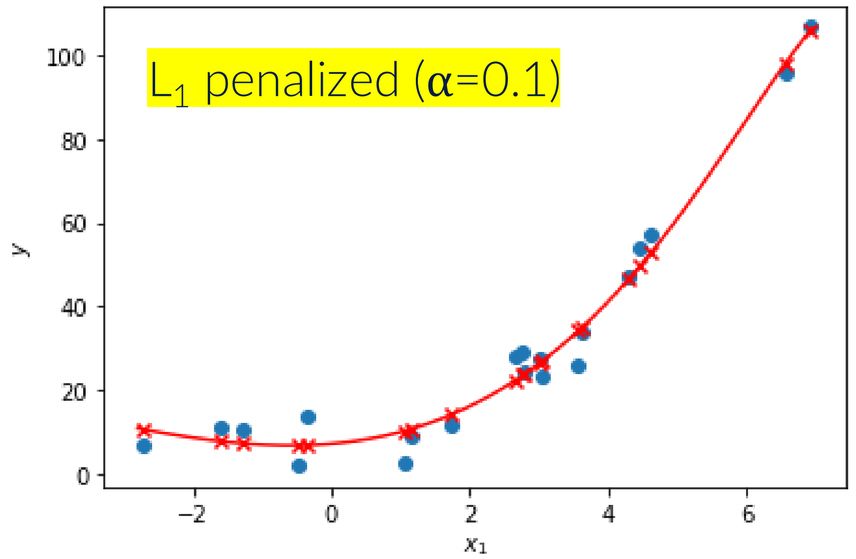
LASSO Regression

- The implementation in the class **Lasso** (scikit-learn) uses coordinate descent to minimize the cost junction $g(\boldsymbol{w})$
- Note that iterative optimization greatly benefits from a standardization of the dependent and independent variables

PolynomialFeatures(degree=**15**, include_bias=**False**)



PolynomialFeatures(degree=**15**, include_bias=**False**)



What did I learn?

- Expanding feature space for polynomial regression
- Curse of dimensionality
- Underfitting and overfitting
- L1 regularization (Lasso) to prevent overfitting

QUESTIONS ?