Linear Regression

Hongchang Gao

Spring 2024

Summarization

- 1. Build model

$$f(x_i) = w_0 + w_1 x_{i,1} + w_2 x_{i,2} + \cdots + w_d x_{i,d}$$

- 2. Build loss function

$$\min_w \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

- 3. Optimize model

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

- 4. Make prediction

$$\hat{y} = w^T x_{n+1}$$

Gradient Descent

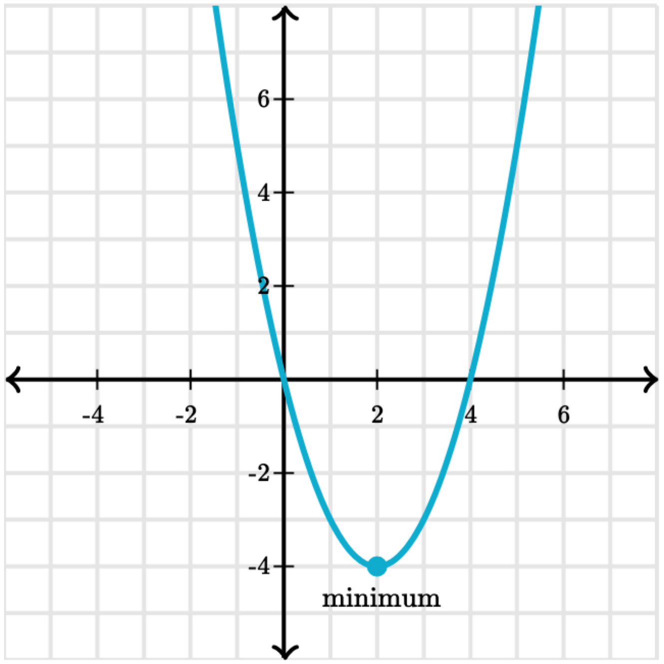
- Machine learning model is an optimization problem

$$\min_x \; f(x)$$

  - f(x) is the objective function, or loss function.
  - x is the model parameter.
  - Goal: find a model parameter to minimize the objective function

Gradient Descent

- Gradient Descent method:
  - Decrease the loss function along the direction of the negative gradient
    - The gradient represents the slope of the loss function curve



Gradient Descent

- Gradient Descent method:
  - Decrease the loss function along the direction of the negative gradient

$$x_{t+1} = x_t - \eta \nabla f(x_t)$$

- x_t is the model parameter in the t-th iteration
- $\nabla f(x_t)$ is the gradient
- η is the learning rate, or step size

Gradient Descent

- Example: Linear Regression

$$\min_w \|y - Xw\|^2$$

  - Gradient:

$$-2X^T(y - Xw)$$

  - Gradient descent:

$$w_{t+1} = w_t - \eta(2X^T(Xw_t - y))$$

Example

- House price prediction

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41	880	129.0	322	126	8.3252	452600	NEAR BAY
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	358500	NEAR BAY
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	352100	NEAR BAY
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	341300	NEAR BAY
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	342200	NEAR BAY

Practical Step 1

- 1. Data preprocessing

```
df = pd.read_csv('housing.csv')

# 0. fill in missing values
mean_val = df['total_bedrooms'].mean()
df['total_bedrooms'] = df['total_bedrooms'].fillna(mean_val)
print(df.isnull().sum())

# 1. convert categorical features to numerical values
labelencoder = LabelEncoder()
df['ocean_proximity'] = labelencoder.fit_transform(df['ocean_proximity'])
print(df.info())
```

Practical Step 2

- 2. Split dataset

Training phase

Given n samples: $\{(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)\}$

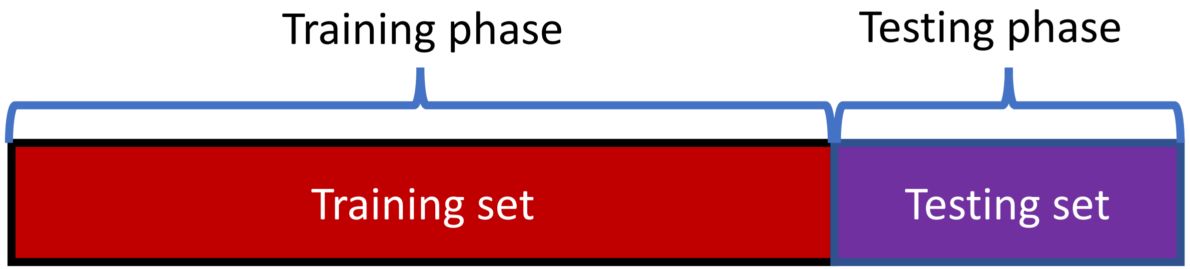
Learn a mapping function $x_i \xrightarrow{f(x)} y_i$

Testing phase

Given a new sample: $x_{n+1} \xrightarrow{f(x)} ?$

Practical Step 2

- 2. Split dataset
  - Training set
    - Used for training the model, during the training phase
  - Testing set
    - Used for evaluating the performance of the model, after obtaining the model



Practical Step 2

- ```
  # 2. split samples
  house_fea = df.drop('median_house_value', axis=1).values
  house_price = df['median_house_value'].values

  X_train,X_test,y_train,y_test = train_test_split(house_fea,
                                                   house_price,
                                                   test_size=0.2,
                                                   random_state=42)

  print(X_train.shape)      (16512, 9)
  print(X_test.shape)       (4128, 9)


  # normalize features
  normalizer = StandardScaler()
  X_train = normalizer.fit_transform(X_train)
  X_test = normalizer.transform(X_test)
  ```
  ```

Practical Step 3

- 3. Train the model
  - Find the optimal model parameter

```
#3. train the model
lr = LinearRegression()

lr.fit(X_train,y_train)

print("bias is "+str(lr.intercept_))
print("coefficients  is "+str(lr.coef_))
```

```
bias is 207194.69373786208
coefficients  is [-85854.94724101 -90946.06271148  14924.30655143 -17693.23405277
  48767.60670995 -43884.16852449  17601.31495096  77144.10164179
   -451.52015229]
```

Practical Step 3

- 3. Train the model

```
y_train_pred = lr.predict(X_train)

mae = mean_absolute_error(y_train_pred,y_train)
mse = mean_squared_error(y_train_pred,y_train)
rmse = np.sqrt(mse)

print('prediction for training set:')
print('MAE is: {}'.format(mae))
print('MSE is: {}'.format(mse))
print('RMSE is: {}'.format(rmse))
```

```
prediction for training set:
MAE is: 50626.9285430206
MSE is: 4810958229.787787
RMSE is: 69361.0714290645
```

Practical Step 4

- 4. Evaluate the model

```
#4. evaluate the model
y_test_pred = lr.predict(X_test)

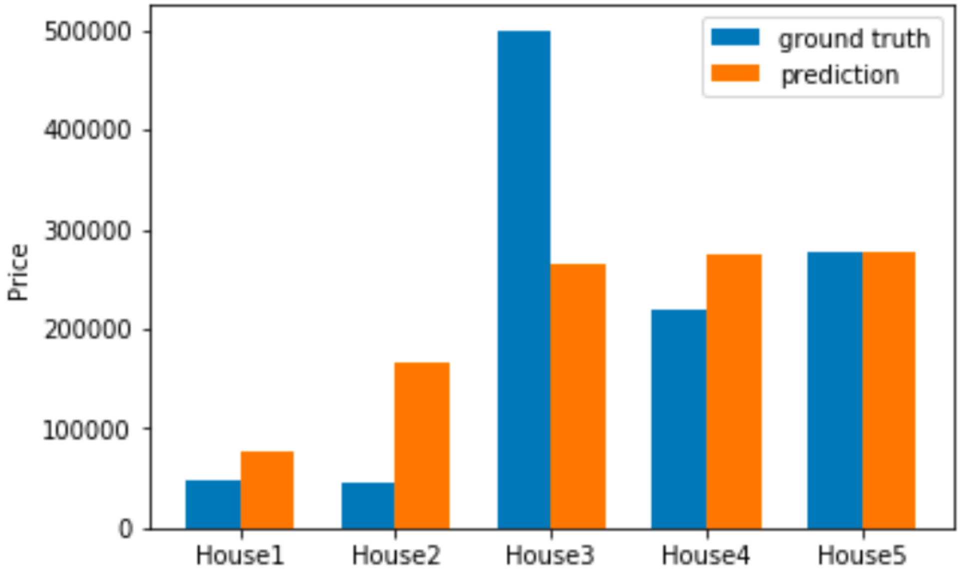
mae = mean_absolute_error(y_test_pred,y_test)
mse = mean_squared_error(y_test_pred,y_test)
rmse = np.sqrt(mse)

print('prediction for testing set:')
print('MAE is: {}'.format(mae))
print('MSE is: {}'.format(mse))
print('RMSE is: {}'.format(rmse))
```

```
prediction for testing set:
MAE is: 51846.87784903816
MSE is: 5055025116.165613
RMSE is: 71098.6998205033
```

Practical Step 4

- 4. Evaluate the model



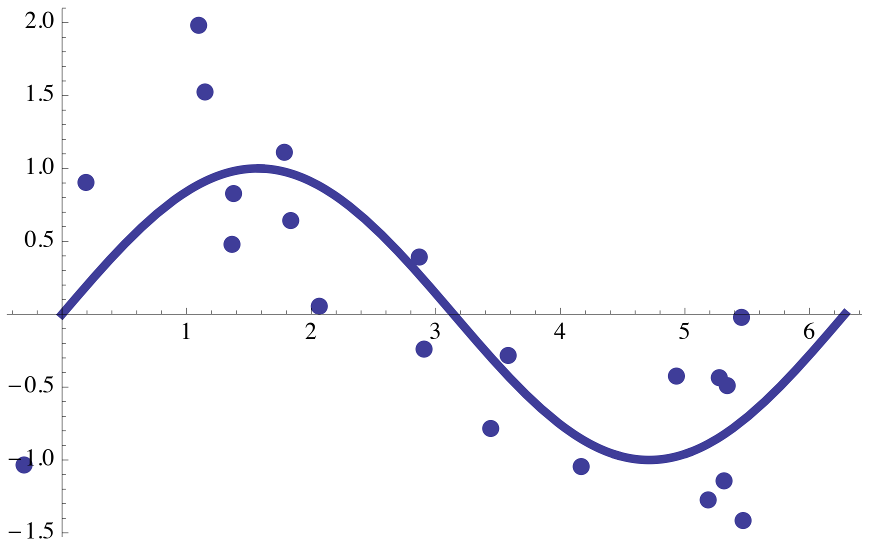

```
labels = ['House1', 'House2', 'House3', 'House4', 'House5']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars

fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, y_test[0:5], width, label='ground truth')
rects2 = ax.bar(x + width/2, y_test_pred[0:5], width, label='prediction')

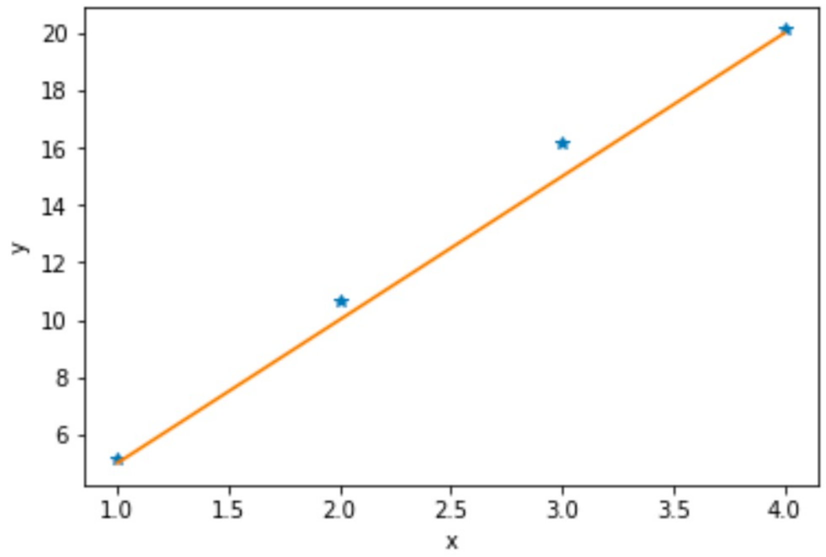
ax.set_ylabel('Price')
ax.set_xticks(x)
ax.set_xticklabels(labels)
ax.legend()
```

How to fit non-linear data?

- Y has a non-linear response to X



Non-linear model



Linear model

How to fit non-linear data?

- Apply non-linear transformation to features

One-dimensional input: $\mathbf{x} = (x) \xrightarrow{\phi(\mathbf{x})} \{1, x, x^2, x^3, \cdots, x^k\}$

$$f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_k x^k$$

Two-dimensional input: $\mathbf{x} = (x_1, x_2) \xrightarrow{\phi(\mathbf{x})} \{1, x_1, x_1^2, x_2, x_2^2, x_1 x_2\}$

$$f(x) = w_0 + w_1 x_1 + w_2 x_1^2 + w_3 x_2 + w_4 x_2^2 + w_5 x_1 x_2$$

How to fit non-linear data?

- Linear model

x	y
1	5.14
2	10.67
3	16.17
4	20.12

$$f(x_1) = w_0 + w_1 * 1$$
$$f(x_2) = w_0 + w_1 * 2$$
$$f(x_3) = w_0 + w_1 * 3$$
$$f(x_4) = w_0 + w_1 * 4$$

- Non-linear model

x	x^2	x^3	x^4	y
1	1^2	1^3	1^4	5.14
2	2^2	2^3	2^4	10.67
3	3^2	3^3	3^4	16.17
4	4^2	4^3	4^4	20.12

$$f(x_1) = w_0 + w_1 * 1 + w_2 * 1^2 + w_3 * 1^3 + w_4 * 1^4$$
$$f(x_2) = w_0 + w_1 * 2 + w_2 * 2^2 + w_3 * 2^3 + w_4 * 2^4$$
$$f(x_3) = w_0 + w_1 * 3 + w_2 * 3^2 + w_3 * 3^3 + w_4 * 3^4$$
$$f(x_4) = w_0 + w_1 * 4 + w_2 * 4^2 + w_3 * 4^3 + w_4 * 4^4$$

How to fit non-linear data?

- Linear model

X_1	X_2	y
1	-1	11
2	-2	12
3	-3	13
4	-4	14

- Non-linear model

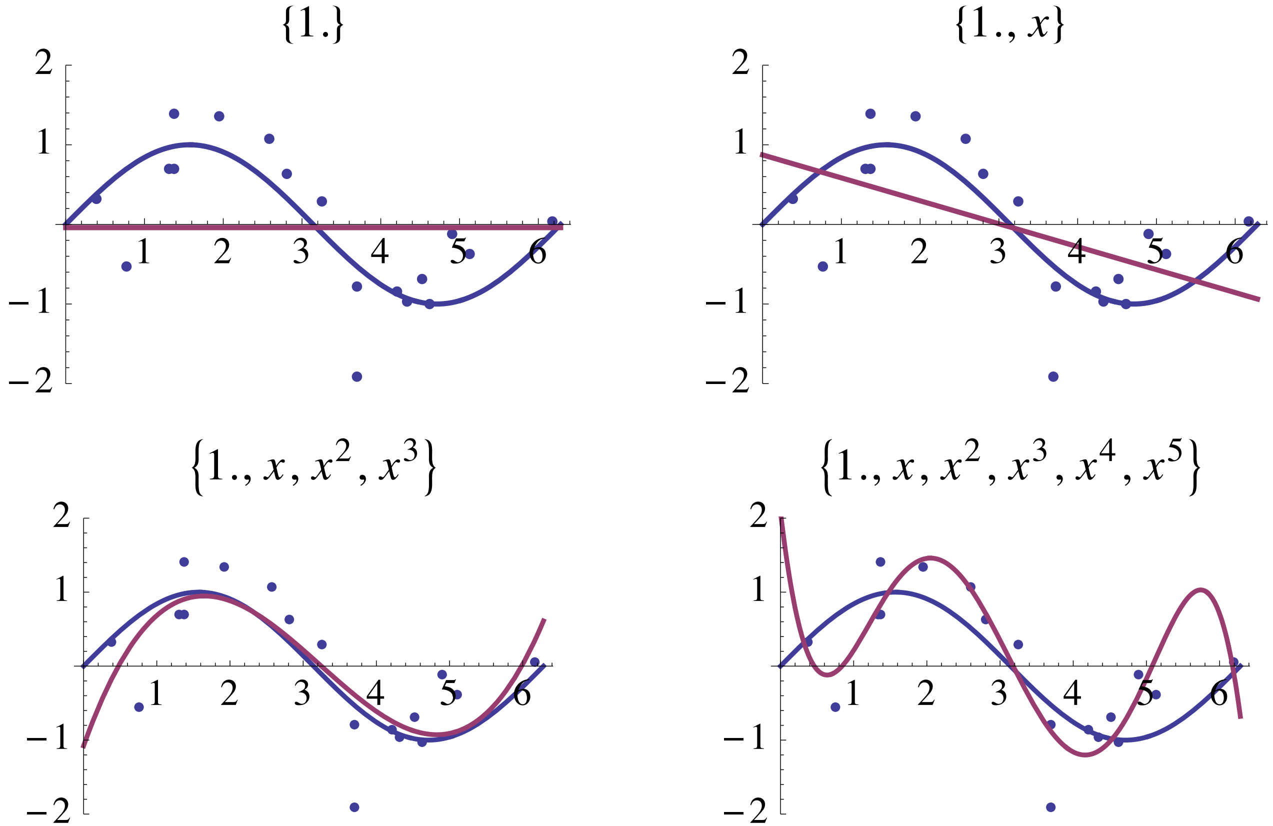
Two-dimensional input: $\mathbf{x} = (x_1, x_2) \xrightarrow{\phi(\mathbf{x})} \{1, x_1, x_1^2, x_2, x_2^2, x_1 x_2\}$

X_1	X_2				y
1	-1				11
2	-2				12
3	-3				13
4	-4				14

How to fit non-linear data?

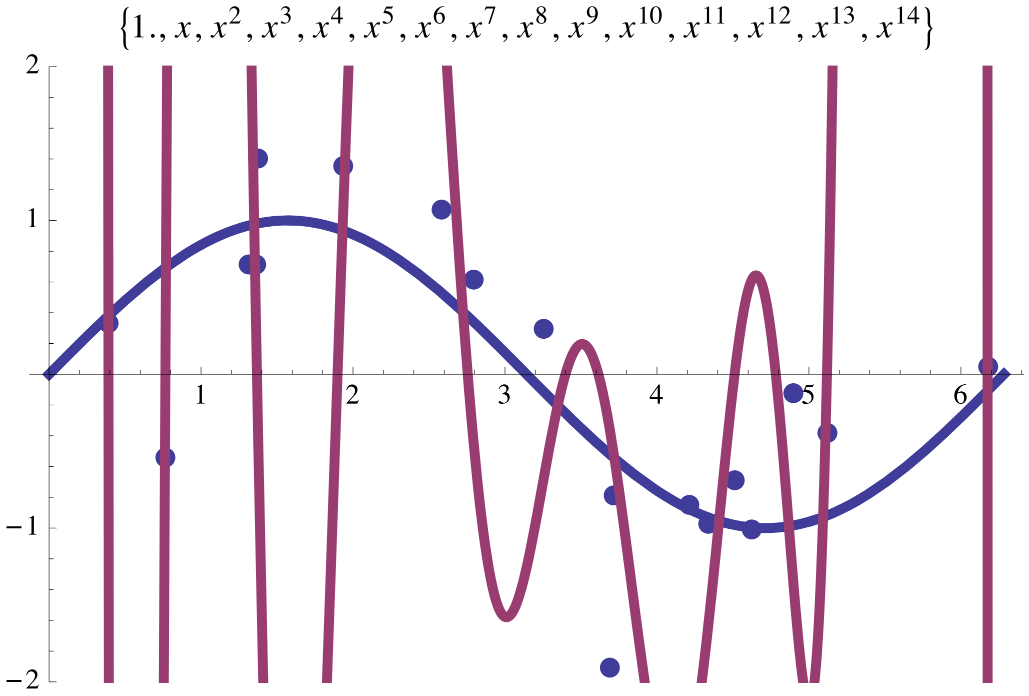
- Non-linear transformation $\phi(x) = \{1, x, x^2, \ldots, x^k\}$

$$f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_k x^k$$



Overfitting

- Errors on training data are small
- But errors on new points are likely to be large



How to avoid overfitting?

- Add a regularization term
  - Make some w_i very small or approach to zero

$$f(x) = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_k x^k$$

$$\min_w \boxed{\frac{1}{n}\|y - Xw\|_2^2} + \boxed{\lambda\|w\|_2^2}$$

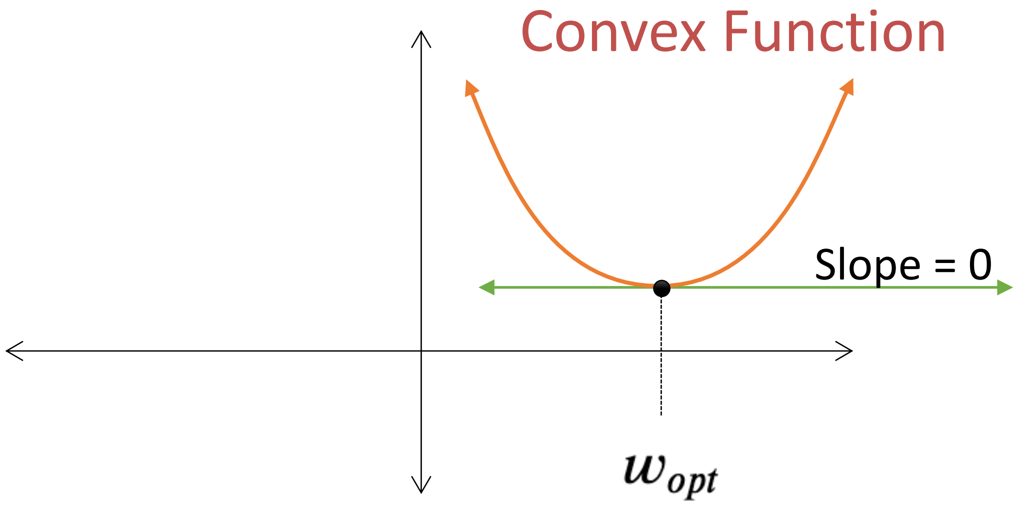
Fit the data   Control the model complexity

w is the model parameter

λ is the hyperparameter

How to avoid overfitting?

- Optimization



$$\min_w \frac{1}{n}\|y - Xw\|_2^2 + \lambda\|w\|_2^2$$

$$\downarrow$$

$$\frac{\partial\mathcal{L}(w)}{w} = -\frac{2}{n}X^T(y - Xw) + 2\lambda w = 0$$

$$\downarrow$$

$$(X^TX + \lambda nI)w = X^Ty$$

$$\downarrow$$

$$w = (X^TX + \lambda nI)^{-1}X^Ty$$

$$\mathcal{L}(w) = \frac{1}{n}\|y - Xw\|_2^2$$

$$\downarrow$$

$$\frac{\partial\mathcal{L}(w)}{w} = -\frac{2}{n}X^T(y - Xw) = 0$$

$$\downarrow$$

$$X^TXw = X^Ty$$

$$\downarrow$$

$$w = (X^TX)^{-1}X^Ty$$

Ridge Regression vs Lasso

- Ridge regression
  - ℓ_2-norm regularization

$$\min_w \frac{1}{n}\|y - Xw\|_2^2 + \lambda\|w\|_2^2$$

- Lasso
  - ℓ_1-norm regularization

$$\min_w \frac{1}{n}\|y - Xw\|_2^2 + \lambda\|w\|_1$$

Ridge Regression vs Lasso

- Ridge Regression

```
#3. train the model
lr = Ridge(alpha=0.1)

lr.fit(X_train,y_train)
```

- Lasso

```
#3. train the model
lr = Lasso(alpha=0.1)

lr.fit(X_train,y_train)
```

Ridge Regression vs Lasso

```
x = np.random.rand(10, 1)
noise = np.random.rand(10, 1)*0.2
y = 3*x + noise

# linear regression
lr = LinearRegression()
lr.fit(x,y)
print("coefficients  is "+str(lr.coef_))

# ridge regression
lr = Ridge(alpha=0.1)
lr.fit(x,y)
print("coefficients  is "+str(lr.coef_))

# lasso
lr = Lasso(alpha=0.1)
lr.fit(x,y)
print("coefficients  is "+str(lr.coef_))
```

```
coefficients  is [[2.95186796]]
coefficients  is [[2.64284605]]
coefficients  is [1.78259097]
```

Online Resources

- https://cims.nyu.edu/~cfgranda/pages/OBDA_fall17/notes/linear_models.pdf
- https://stat.ethz.ch/education/semesters/ss2016/regression/Regression.pdf
- https://www.cs.toronto.edu/~rgrosse/courses/csc311_f20/readings/notes_on_linear_regression.pdf