Data visualization

COSC 480B

Reyan Ahmed

rahmed1@colgate.edu

Lecture 18

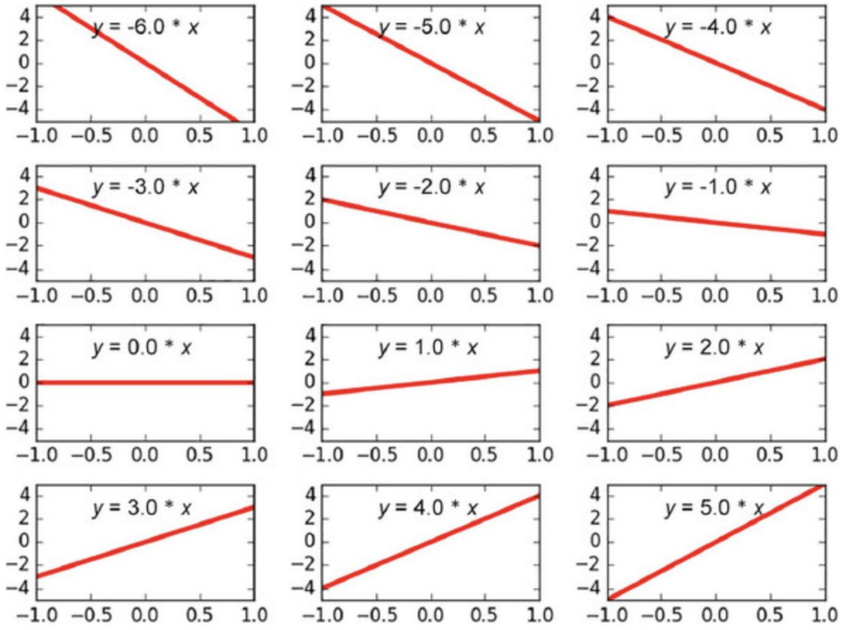
Linear regression and beyond

Overview

- Fitting a line to data points
- Fitting arbitrary curves to data points
- Testing performance of regression algorithms
- Applying regression to real-world data

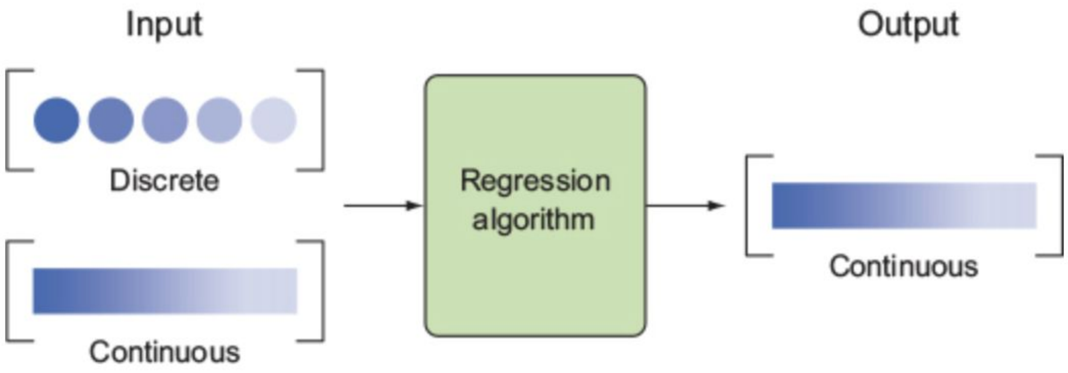
Formal notation

Different values of the parameter w result in different linear equations. The set of all these linear equations is what constitutes the linear model M.



Formal notation

A regression algorithm is meant to produce continuous output. The input is allowed to be discrete or continuous. This distinction is important because discrete-valued outputs are handled better by classification, which is discussed in the next chapter.



Formal notation

Exercise 1: How many possible functions exist that map 10 integers to 10 integers? For example, let $f(x)$ be a function that can take numbers 0 through 9 and produce numbers 0 through 9. One example is the identity function that mimics its input—for example, $f(0) = 0$, $f(1) = 1$, and so on. How many other functions exist?

Formal notation

Exercise 1: How many possible functions exist that map 10 integers to 10 integers? For example, let f(x) be a function that can take numbers 0 through 9 and produce numbers 0 through 9. One example is the identity function that mimics its input—for example, f(0) = 0, f(1) = 1, and so on. How many other functions exist?

Answer: 10^{10} = 10,000,000,000

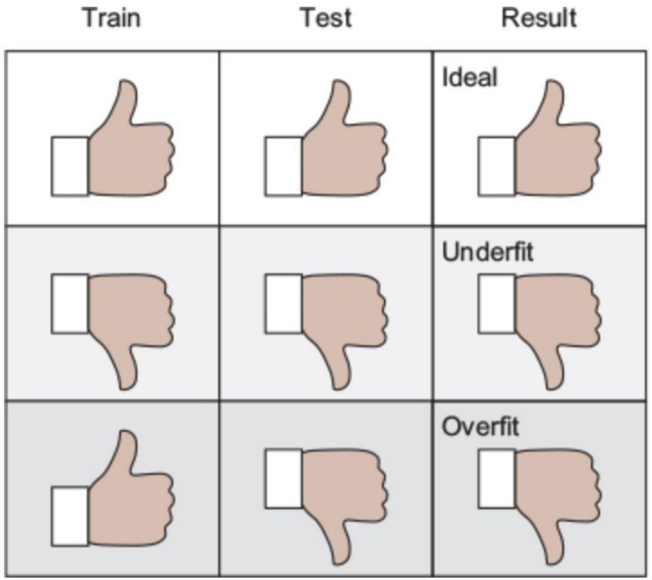
Formal notation

To measure the success of the learning algorithm, you'll need to understand two important concepts, variance and bias:

- Variance indicates how sensitive a prediction is to the training set that was used. Ideally, how you choose the training set shouldn't matter—meaning a lower variance is desired.
- Bias indicates the strength of assumptions made about the training dataset. Making too many assumptions might make the model unable to generalize, so you should prefer low bias as well.

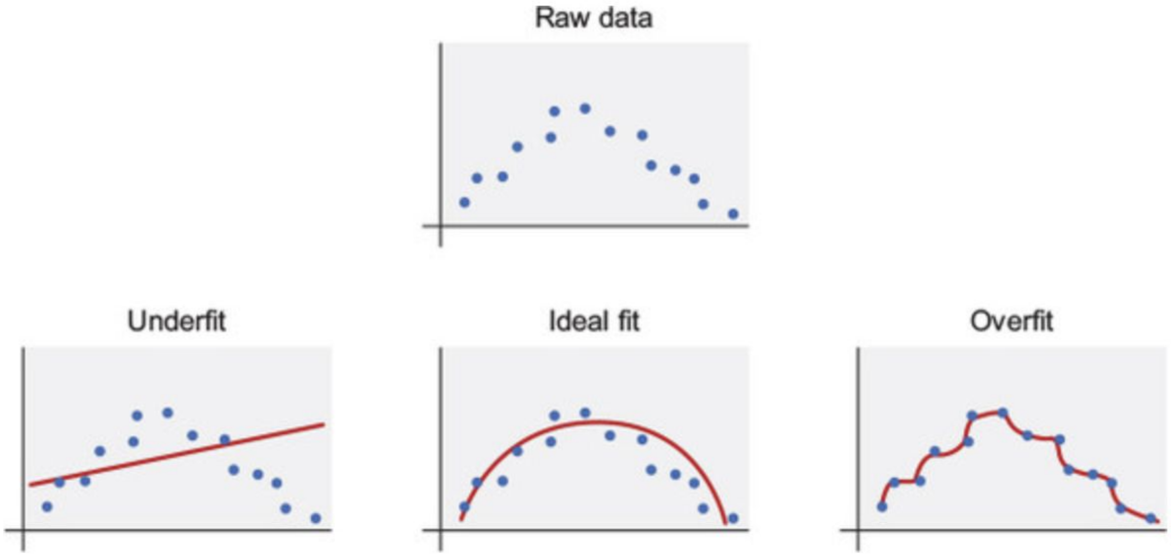
Formal notation

Ideally, the best-fit curve fits well on both the training data and the test data. If we witness it fitting poorly with the test data and the training data, there's a chance that our model is underfitting. On the other hand, if it performs poorly on the test data but well on the training data, we know the model is overfitting.



Formal notation

Examples of underfitting and overfitting the data



Formal notation

Exercise 2: Let's say your model is $M(w)$: $y = wx$. How many possible functions can you generate if the values of the weight parameter w must be integers between 0 and 9 (inclusive)?

Formal notation

Exercise 2: Let's say your model is M(w): y = wx. How many possible functions can you generate if the values of the weight parameter w must be integers between 0 and 9 (inclusive)?

ANSWER: Only 10: {y = 0, y = x, y = 2x, ..., y = 9x}.

Formal notation

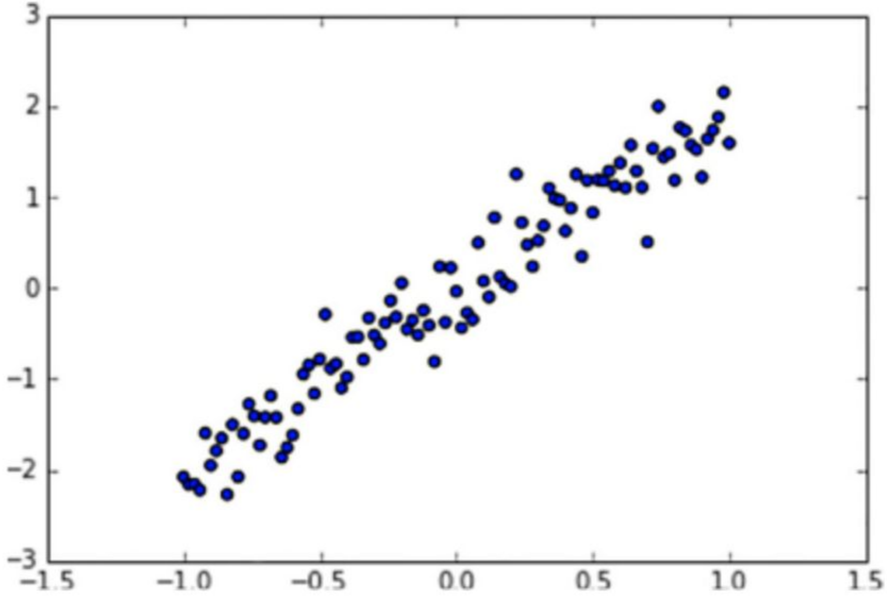
Visualizing raw input

```
import numpy as np                                   1
import matplotlib.pyplot as plt                        2

x_train = np.linspace(-1, 1, 101)                       3
y_train = 2 * x_train +
np.random.randn(*x_train.shape) * 0.33     4

plt.scatter(x_train, y_train)                         5
plt.show()                                         5
```

1 Imports NumPy to help generate initial raw data
2 Uses matplotlib to visualize the data
3 The input values are 101 evenly spaced numbers between –1 and 1.
4 The output values are proportional to the input but with added noise.
5 Uses matplotlib's function to generate a scatter plot of the data

Formal notation

Scatter plot of y = x + (noise)



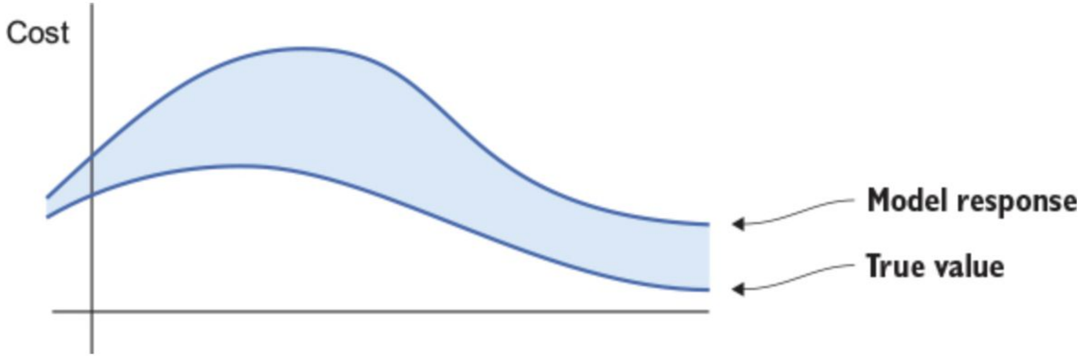
Formal notation

Whichever parameter w minimizes, the cost is optimal. Cost is defined as the norm of the error between the ideal value with the model response. And, lastly, the response value is calculated from the function in the model set.

$$w* = \arg\ min_w\ \underbrace{cost(Y_{model}, Y_{ideal})}_{|\underbrace{Y_{model}}_{M(w, X)} - Y_{ideal}|}$$

Formal notation

The cost is the norm of the point-wise difference between the model response and the true value.



Formal notation

Solving linear regression

```
import tensorflow as tf                          1
import numpy as np                               1
import matplotlib.pyplot as plt                  1

learning_rate = 0.01                             2
training_epochs = 100                            2

x_train = np.linspace(-1, 1, 101)
3
y_train = 2 * x_train + np.random.randn(*x_train.shape) *
0.33          3

X = tf.placeholder(tf.float32)                   4
Y = tf.placeholder(tf.float32)                   4
```

1 Imports TensorFlow for the learning algorithm. You'll need NumPy to set up the initial data. And you'll use matplotlib to visualize your data.
2 Defines constants used by the learning algorithm. They're called hyperparameters.
3 Sets up fake data that you'll use to find a best-fit line
4 Sets up the input and output nodes as placeholders because the value will be injected by x_train and y_train

Formal notation

```
def model(X, w):                                    5
    return tf.multiply(X, w)

w = tf.Variable(0.0, name="weights")                6

y_model = model(X, w)                               7
cost = tf.square(Y-y_model)                         7

train_op =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
8

sess = tf.Session()                                 9
init = tf.global_variables_initializer()            9
sess.run(init)                                      9
```

5 Defines the model as y = w*X
6 Sets up the weights variable
7 Defines the cost function
8 Defines the operation that will be called on each iteration of the learning algorithm
9 Sets up a session and initializes all variables

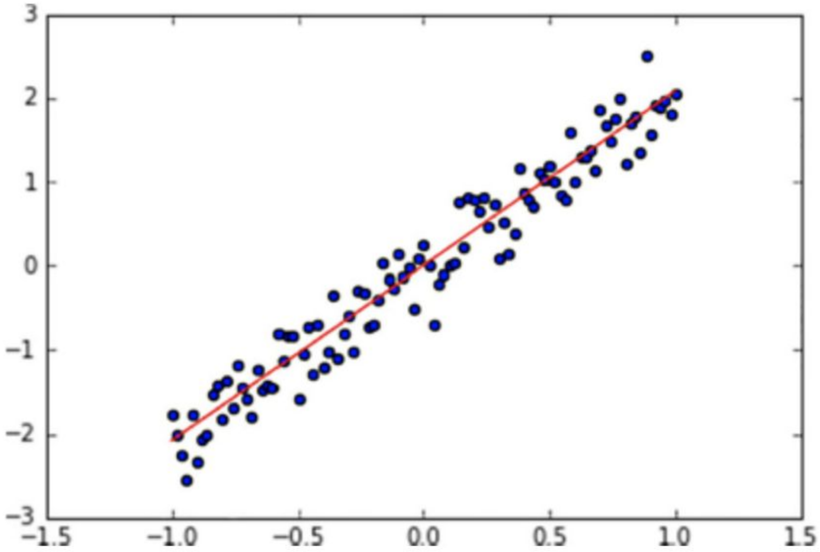
Formal notation

```
for epoch in range(training_epochs):                    10
  for (x, y) in zip(x_train, y_train):                  11
    sess.run(train_op, feed_dict={X: x, Y: y})          12

w_val = sess.run(w)                                     13

sess.close()                                            14
plt.scatter(x_train, y_train)                           15
y_learned = x_train*w_val                               16
plt.plot(x_train, y_learned, 'r')                       16
plt.show(    )                                          16
```

10 Loops through the dataset multiple times
11 Loops through each item in the dataset
12 Updates the model parameter(s) to try to minimize the cost function
13 Obtains the final parameter value
14 Closes the session
15 Plots the original data
16 Plots the best-fit line

Formal notation

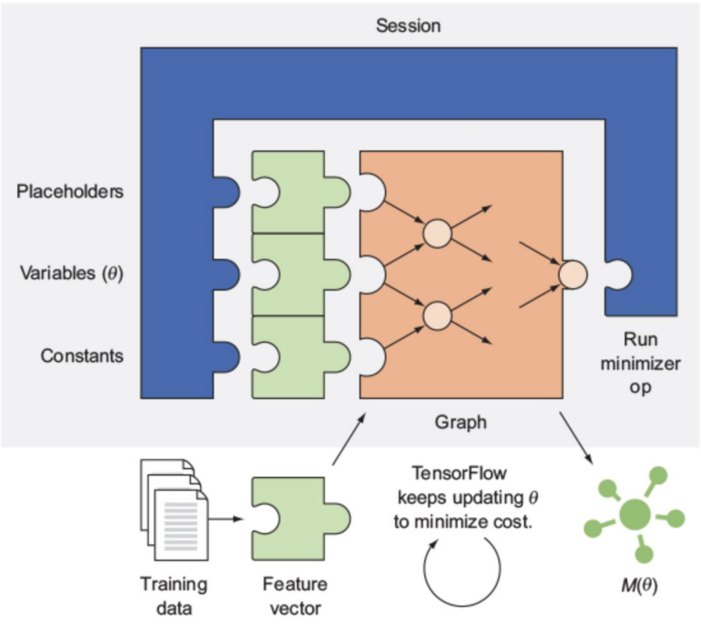
Linear regression estimate shown by running previous code.



Formal notation

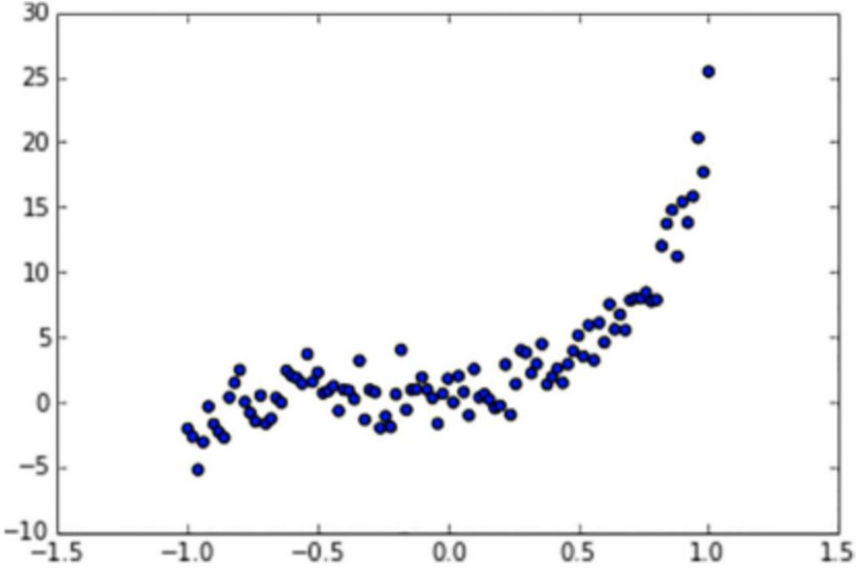
The learning algorithm updates the model’s parameters to minimize the given cost function.

Learning algorithm in TensorFlow



Polynomial model

Data points like this aren't suitable for a linear model.



Polynomial model

Using a polynomial model

```
import tensorflow as tf                                  1
import numpy as np                                       1
import matplotlib.pyplot as plt                          1

learning_rate = 0.01                                     1
training_epochs = 40                                     1

trX = np.linspace(-1, 1, 101)                            2

num_coeffs = 6                                           3
trY_coeffs = [1, 2, 3, 4, 5, 6]                          3
trY = 0                                                  3
for i in range(num_coeffs):                              3
    trY += trY_coeffs[i] * np.power(trX, i)              3

trY += np.random.randn(*trX.shape) * 1.5                 4

plt.scatter(trX, trY)                                    5
plt.show()                                               5
```

1 Imports the relevant libraries and initializes the hyperparameters
2 Sets up fake raw input data
3 Sets up raw output data based on a fifth-degree polynomial
4 Adds noise
5 Shows a scatter plot of the raw data

Polynomial model

```
X = tf.placeholder(tf.float32)                         6
Y = tf.placeholder(tf.float32)                         6

def model(X, w):                                       7
    terms = []                                         7
    for i in range(num_coeffs):                        7
        term = tf.multiply(w[i], tf.pow(X, i))         7
        terms.append(term)                             7
    return tf.add_n(terms)                             7

w = tf.Variable([0.] * num_coeffs, name="parameters")
8
y_model = model(X, w)                                  8

cost = (tf.pow(Y-y_model, 2))                          9
train_op =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost) 9
```

6 Defines the nodes to hold values for input/output pairs
7 Defines your polynomial model
8 Sets up the parameter vector to all zeros
9 Defines the cost function just as before

Polynomial model

```
sess = tf.Session()                                         10
init = tf.global_variables_initializer()                    10
sess.run(init)                                              10
                                                            10
for epoch in range(training_epochs):                        10
    for (x, y) in zip(trX, trY):                            10
        sess.run(train_op, feed_dict={X: x, Y: y})          10
                                                            10
w_val = sess.run(w)                                         10
print(w_val)                                                10

sess.close()                                                11
```

10 Sets up the session and runs the learning algorithm just as before
11 Closes the session when done

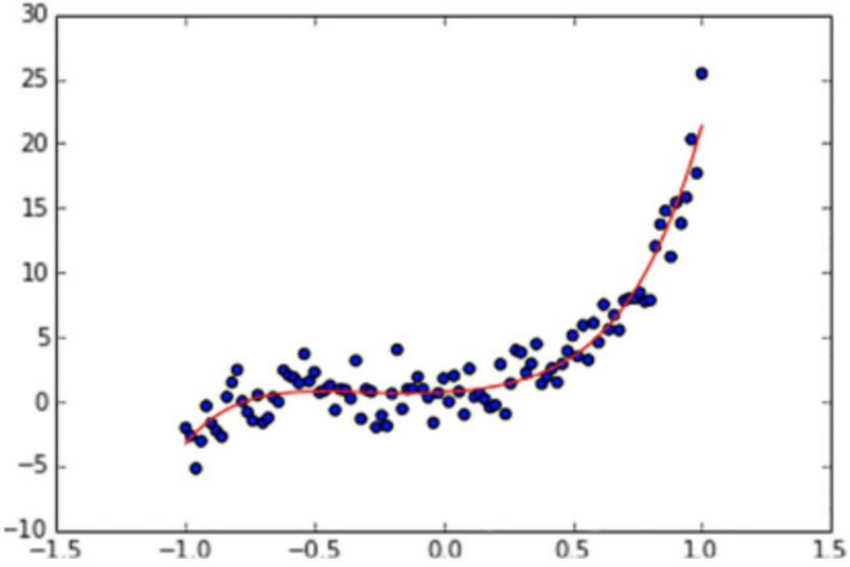
Polynomial model

```
plt.scatter(trX, trY)                               12
trY2 = 0                                          12
for i in range(num_coeffs):                             12
    trY2 += w_val[i] * np.power(trX, i)                   12
                                               12
plt.plot(trX, trY2, 'r')                             12
plt.show()                                         12
```

12 Plots the result

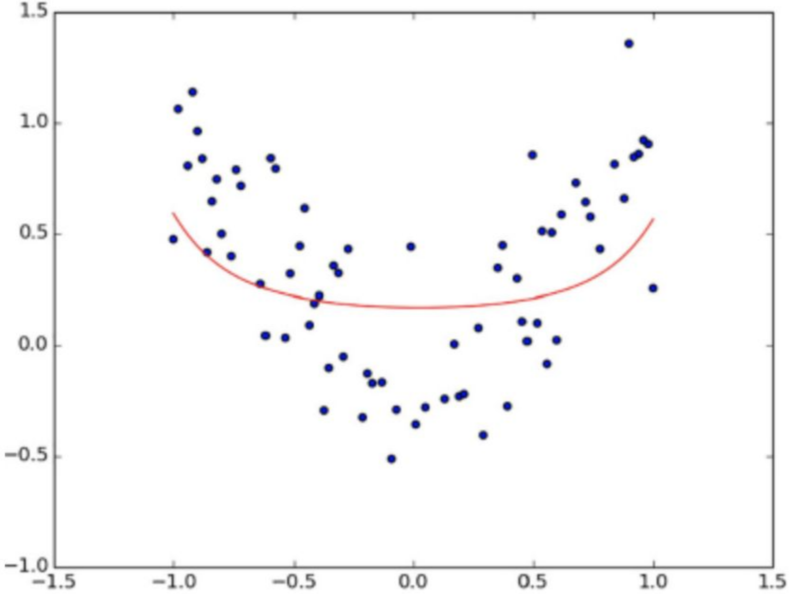
Polynomial model

The best-fit curve smoothly aligns with the nonlinear data.



Regularization

When the model is too flexible, a best-fit curve can look awkwardly complicated or unintuitive. We need to use regularization to improve the fit, so that the learned model performs well against test data.



Regularization

To influence the learning algorithm to produce a smaller coefficient vector (let's call it w), you add that penalty to the loss term.

$$\text{Cost}(X, Y) = \text{Loss}(X, Y) + \lambda|\omega|$$

Regularization

Splitting the dataset into testing and training sets

```
def split_dataset(x_dataset, y_dataset, ratio):                 1
    arr = np.arange(x_dataset.size)                             2
    np.random.shuffle(arr)                                      2
    num_train = int(ratio * x_dataset.size)                     3
    x_train = x_dataset[arr[0:num_train]]                       4
    x_test = x_dataset[arr[num_train:x_dataset.size]]           4
    y_train = y_dataset[arr[0:num_train]]                       5
    y_test = y_dataset[arr[num_train:x_dataset.size]]           5
    return x_train, x_test, y_train, y_test                     6
```

1 Takes the input and output dataset as well as the desired split ratio
2 Shuffles a list of numbers
3 Calculates the number of training examples
4 Uses the shuffled list to split the x_dataset
5 Likewise, splits the y_dataset
6 Returns the split x and y datasets

Regularization

Exercise 3: A Python library called scikit-learn supports many useful data-preprocessing algorithms. You can call a function in scikit-learn to do exactly what the previous code achieves. Can you find this function on the library's documentation? Hint: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection.

Regularization

Exercise 3: A Python library called scikit-learn supports many useful data-preprocessing algorithms. You can call a function in scikit-learn to do exactly what the previous code achieves. Can you find this function on the library's documentation? Hint: http://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection.

ANSWER: It's called sklearn.model_selection.train_test_split.

Regularization

Evaluating regularization parameters

```
import tensorflow as tf                                        1
import numpy as np                                             1
import matplotlib.pyplot as plt                                1
                                                               1
learning_rate = 0.001                                          1
training_epochs = 1000                                         1
reg_lambda = 0.                                                1

x_dataset = np.linspace(-1, 1, 100)                            2
                                                               2
num_coeffs = 9                                                 2
y_dataset_params = [0.] * num_coeffs                           2
y_dataset_params[2] = 1                                        2
y_dataset = 0                                                  2
for i in range(num_coeffs):                                    2
    y_dataset += y_dataset_params[i] * np.power(x_dataset, i)  2
y_dataset += np.random.randn(*x_dataset.shape) * 0.3           2
```

1 Imports the relevant libraries and initializes the hyperparameters
2 Creates a fake dataset, $y = x^2$

Regularization

```
(x_train, x_test, y_train, y_test) = split_dataset(x_dataset, y_dataset, 0.7)3

X = tf.placeholder(tf.float32)                                         4
Y = tf.placeholder(tf.float32)                                         4

def model(X, w):                                                      5
    terms = []                                                        5
    for i in range(num_coeffs):                                       5
        term = tf.multiply(w[i], tf.pow(X, i))                        5
        terms.append(term)                                            5
    return tf.add_n(terms)                                            5

w = tf.Variable([0.] * num_coeffs, name="parameters")                 6
y_model = model(X, w)                                                 6
cost = tf.div(tf.add(tf.reduce_sum(tf.square(Y-y_model)),             6
                     tf.multiply(reg_lambda, tf.reduce_sum(tf.square(w)))),  6
              2*x_train.size)                                         6
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
6
```

3 Splits the dataset into 70% training and 30% testing using listing 3.4
4 Sets up the input/output placeholders
5 Defines your model
6 Defines the regularized cost function

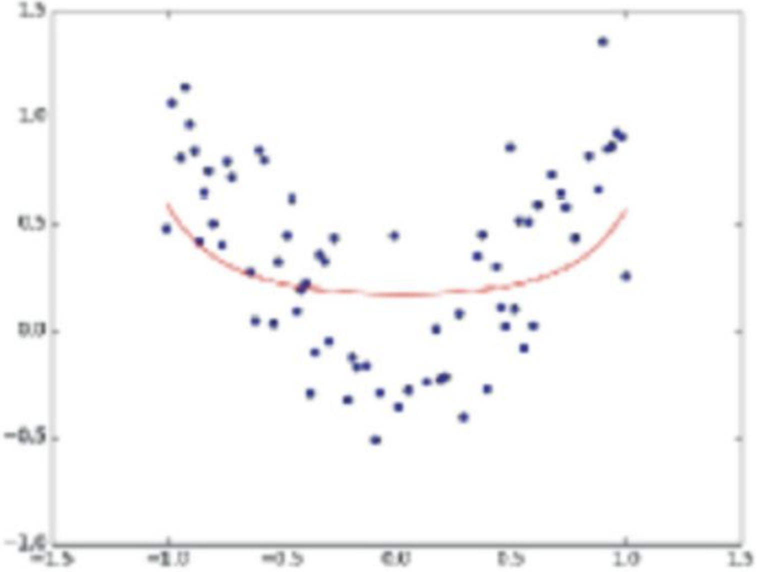
Regularization

```
sess = tf.Session()                                          7
init = tf.global_variables_initializer()                     7
sess.run(init)                                               7

for reg_lambda in np.linspace(0,1,100):                      8
    for epoch in range(training_epochs):                     8
        sess.run(train_op, feed_dict={X: x_train, Y: y_train})     8
    final_cost = sess.run(cost, feed_dict={X: x_test, Y:y_test})     8
    print('reg lambda', reg_lambda)                          8
    print('final cost', final_cost)                          8

sess.close()                                                 9
```

7 Sets up the session
8 Tries various regularization parameters
9 Closes the session

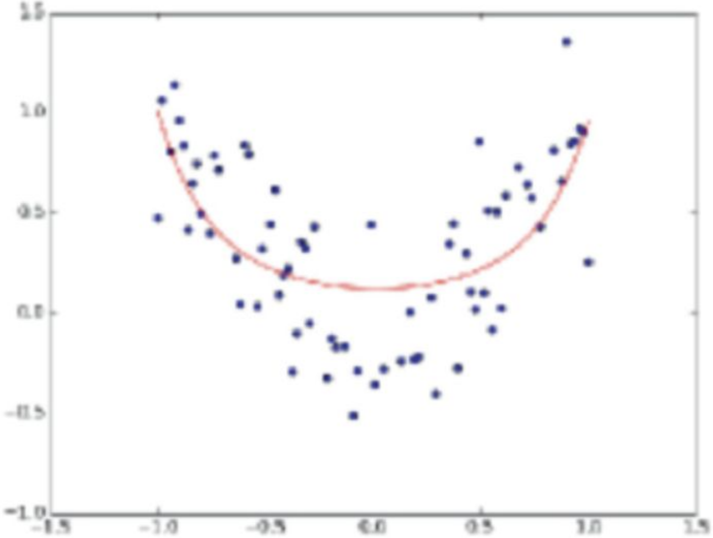
Regularization

As you increase the regularization parameter to some extent, the cost decreases. This implies that the model was originally overfitting the data, and regularization helped add structure.



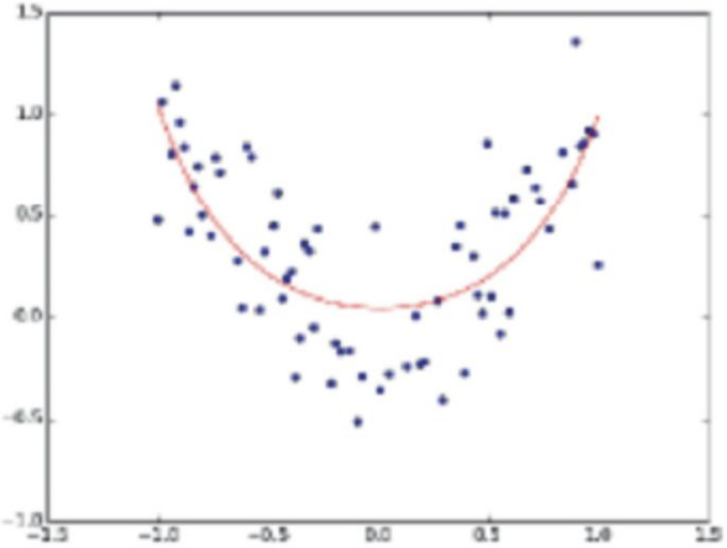
λ = 0.0
Cost = 0.032031

Regularization



λ = 0.05
Cost = 0.24077

Regularization



λ = 0.20
Cost = 0.212215

Application of linear regression

Many datasets are available online to test your newfound knowledge of regression:

- The University of Massachusetts Amherst supplies small datasets of various types: www.umass.edu/statdata/statdata.
- Kaggle contains all types of large-scale data for machine-learning competitions: www.kaggle.com/datasets.
- Data.gov is an open data initiative by the US government that contains many interesting and practical datasets: https://catalog.data.gov.
- A good number of datasets contain dates. For example, there’s a dataset of all phone calls to the 3-1-1 non-emergency line in Los Angeles, California. You can obtain it at http://mng.bz/6vHx.

Application of linear regression

Parsing raw CSV datasets

```
import csv                                          1
import time                                         2

def read(filename, date_idx, date_parse, year, bucket=7):

    days_in_year = 365

    freq = {}                                       3
    for period in range(0, int(days_in_year / bucket)):
        freq[period] = 0

    with open(filename, 'rb') as csvfile:           4
        csvreader = csv.reader(csvfile)
        csvreader.next()
        for row in csvreader:
            if row[date_idx] == '':
                continue
            t = time.strptime(row[date_idx], date_parse)
            if t.tm_year == year and t.tm_yday < (days_in_year-1):
                freq[int(t.tm_yday / bucket)] += 1

    return freq

freq = read('311.csv', 0, '%m/%d/%Y', 2014)         5
```

1 For easily reading CSV files
2 For using useful date functions
3 Sets up initial frequency map
4 Reads data and aggregates count per period
5 Obtains a weekly frequency count of 3-1-1 phone calls in 2014