Data science Part III

Supervised learning, regression and gradient descent

Agenda

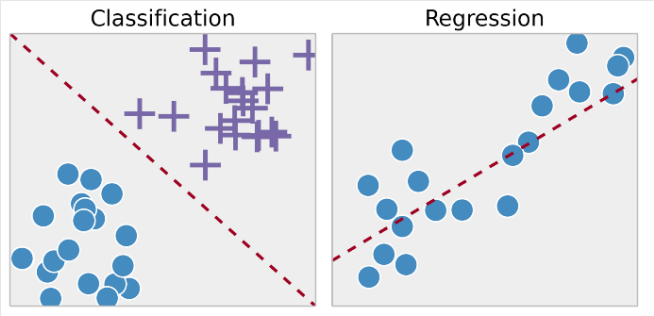
- Introduction to supervised learning
- Classification with k-NN
- Training and testing data
- Regression with gradient descent
- Quick review

Supervised learning

- Given labeled data (X, y), we want to find mapping function y = f(X).

Two common task in supervised learning

- Regression: predict continuous valued output for an example
- Classification: predict a discrete class label output for an example

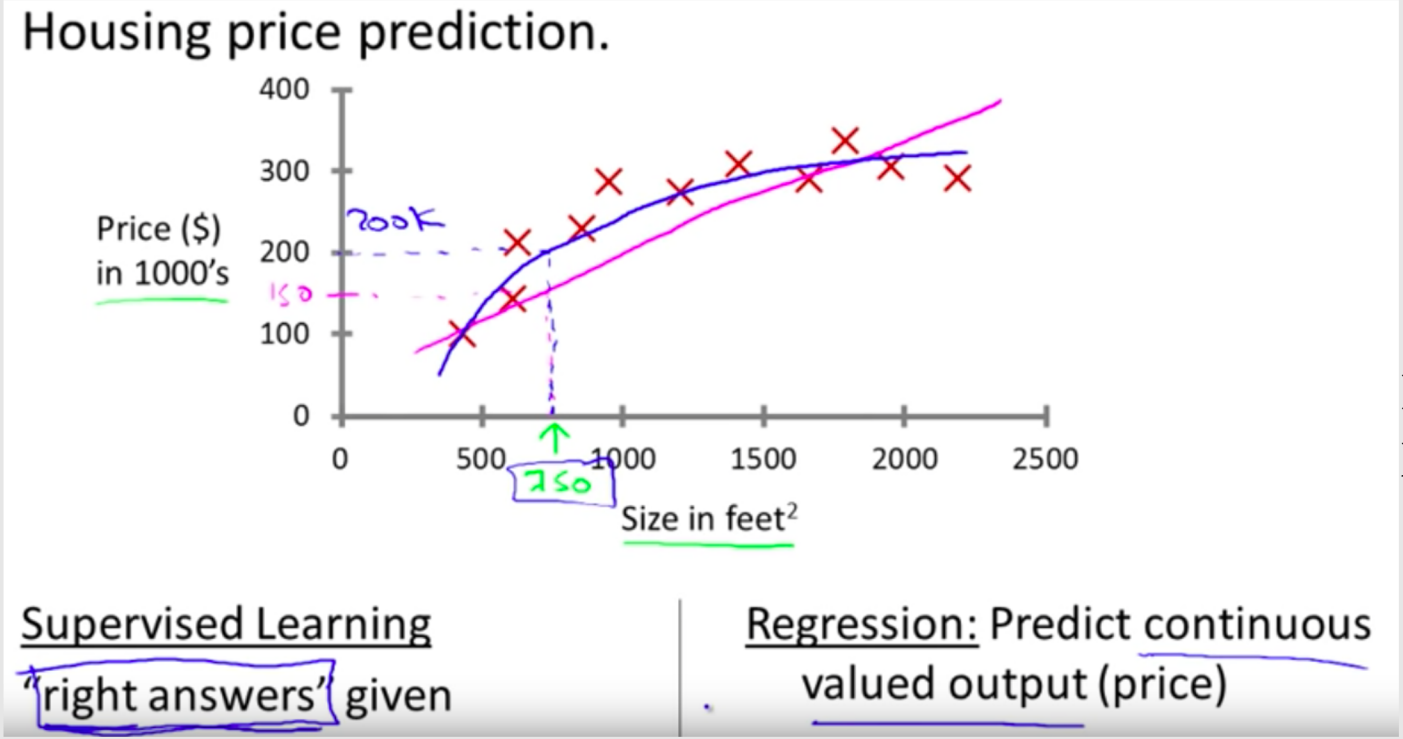


Supervised learning

Andrew Ng Videos about supervised learning

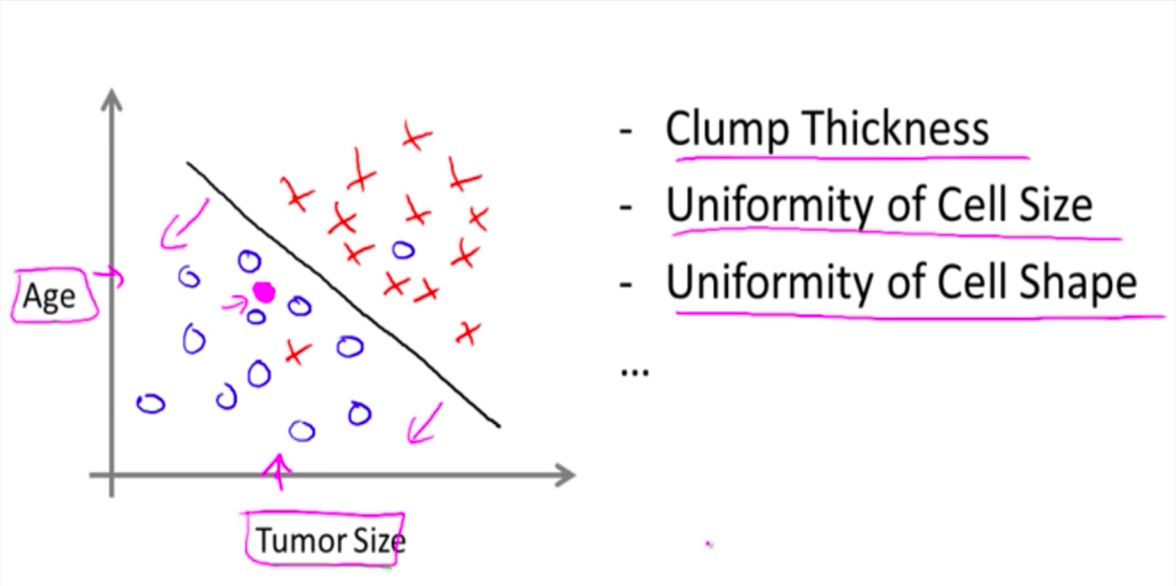
https://www.youtube.com/watch?v=bQI5uDxrFfA&list=PLLssT5z_DsK-h9vYZkQkYNWcItqhlRJLN&index=2

Example: Housing price prediction (regression)



Given a size of a house, we can predict its price.

Example: Using multiple features



Given patient age, tumor size, we want to predict whether the tumor is benign or malignant

Classification

- **The goal**:
  - “similar” members share the same label
- Common objective:
  - Accuracy: fraction of times correctly classified
  - Generalizability: how well the model performs on unseen data

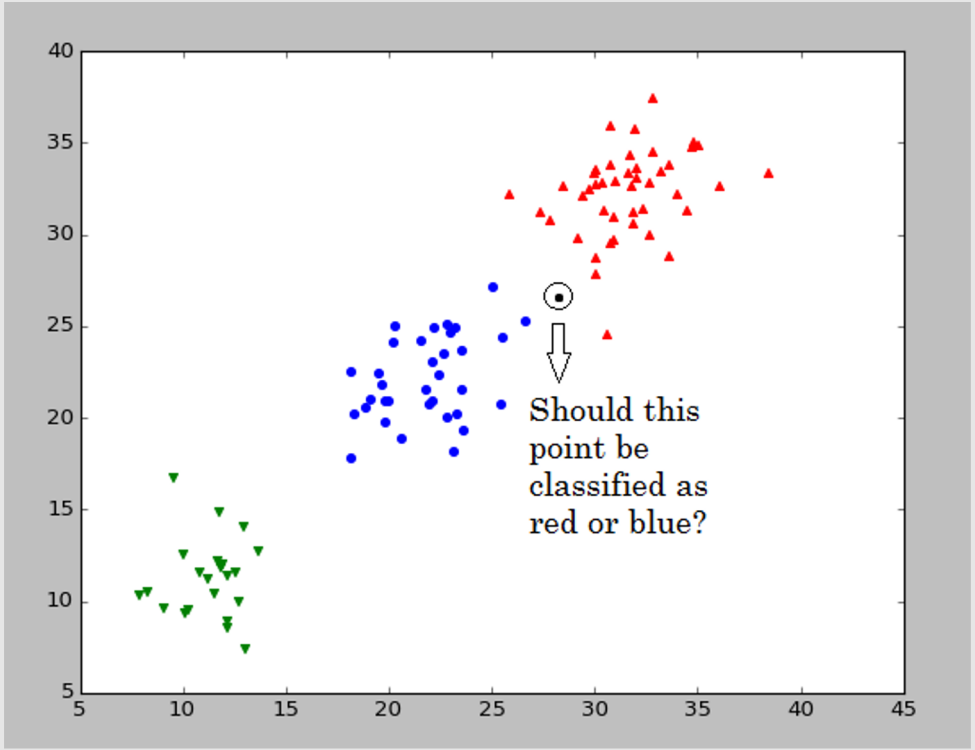
k-Nearest Neighbors (k-NN) classification

An example of “supervised learning”

- Non-parametric method

  f: x → y does not involve any parameter

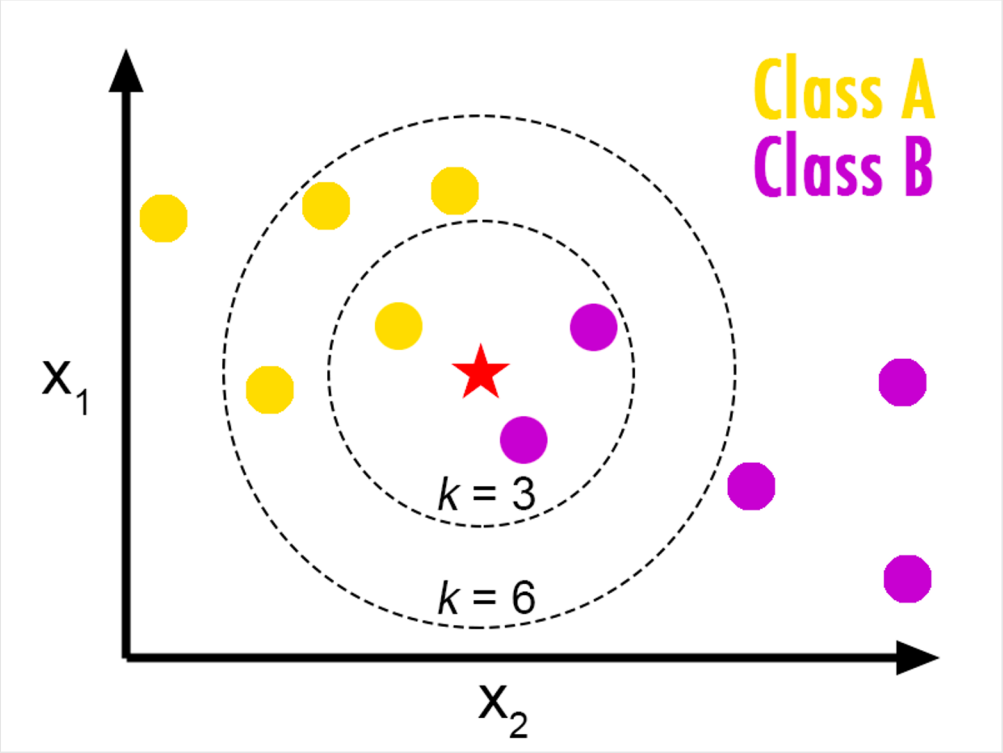
- Lazy learning



k-NN classification

Question:

Which class should our new object (i.e., the star) belong to?



K-NN: Approach

- The idea: **Majority wins**
  - A positive integer k is specified (hyper parameters）
  - Calculate the distance to every known point
  - Find the k nearest ones (sorting)
    - Take Minkowski distance
  - We find the most common classification of these entries

```
3  def minkowskiDist(v1, v2, p):
4      dist = 0.0
5      for i in range(len(v1)):
6          dist += abs(v1[i] - v2[i])**p
7      return dist**(1.0/p)
```

KNN: Programming Activity

```
if __name__ == "__main__":

    # make data
    random.seed(0)
    n = 100
    K = 3
    LABELS = ('a', 'b', 'c')
    all_cluster = []
    data = []
    for i in range(K):
        tmp_data = util.genDistribution(i*2+1, 1, i*2+1, 1, n=20, la
        all_cluster.append(cl.Cluster(tmp_data))
        data += tmp_data

    def onclick(event):
        # Creating a new point and finding the k nearest neighbours
        new = sample.Sample('', [event.xdata, event.ydata], '')
        knn(new, data, K)
```

KNN: Programming Activity

```
def knn(p, data, k):
    """ Compute the distance between p to every sample in data,
        set p's label to the label of the maximum of samples
        within the k nearest neighbors
    """
    """ Steps:
        1. Iterate through samples in data and store the
           distance from p in the dictionary "distance"; key is the
           distance, value is the sample.
        2. Creat a sorted list of samples according to ascending
           order of the distances.
        3. In the dictioary "label_votes", stores number of votes
           in each label among the top-k nearest samples
        4. Assign p the most popular label
    """
    max_label = util.LABELS[0]
    p.setLabel(max_label)
    # above forces a fixed label: remove them
    # replace knn_helper.knn(p, data, k) with your own logic
    print(p)
    knn_helper.knn(p, data, k)
    print(p)
```

p: new data point that we want to predict

data: observed data points that we can use (list of Sample object)

k: pre-specified k value

Program tip: finding the popular vote

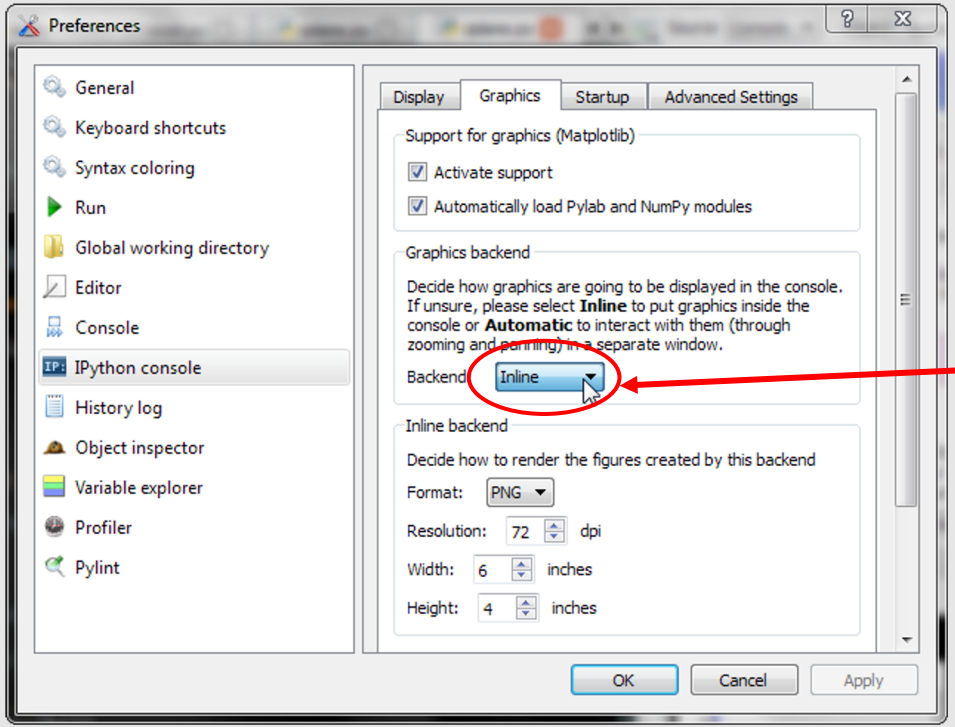
- Okay, I've got label_votes dictionary, can I sort it?
- *No*: dictionary is unordered!
    - But you can sort into a list of keys, but use values (sort-dict.py)

```
 8 mydict = {'carl':40,
 9           'alan':2,
10           'bob':1,
11           'danny':3}
12
13 print(sorted(mydict.keys()))
14
15 for key in sorted(mydict.keys()):
16     print("%s: %s" % (key, mydict[key]))
17
18 print(sorted(mydict.values()))
19
20 print(sorted(mydict, key = mydict.get))
21
```

```
['alan', 'bob', 'carl', 'danny']
alan: 2
bob: 1
carl: 40
danny: 3
[1, 2, 3, 40]
['bob', 'alan', 'danny', 'carl']
```

k-NN demo: knn_main.py

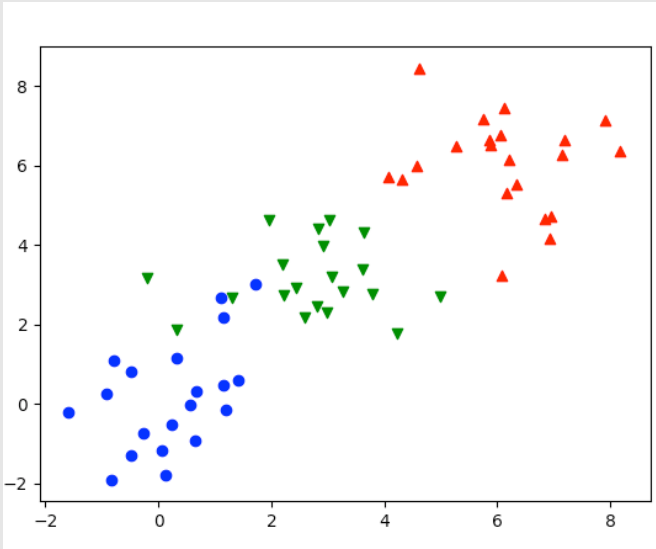
Run it in IDLE; or on command line



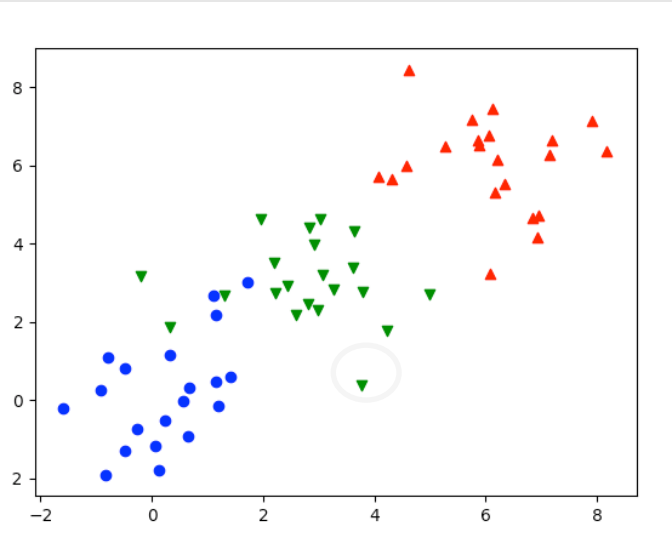
Set this to “automatic” for Spyder

KNN: Programming Activity

before



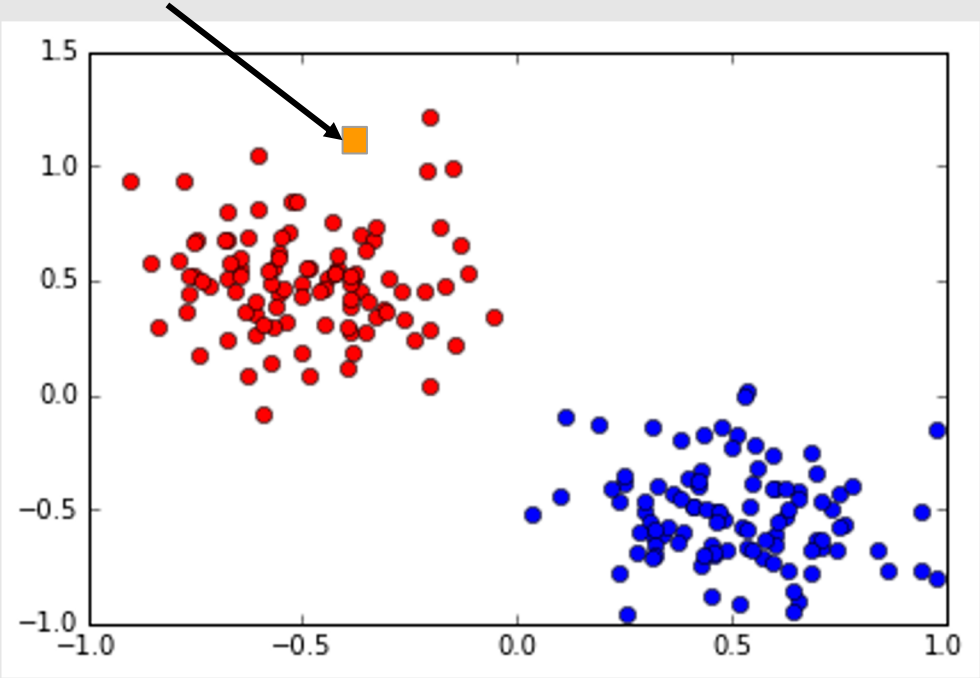
after



Training and testing

When there are multiple solutions

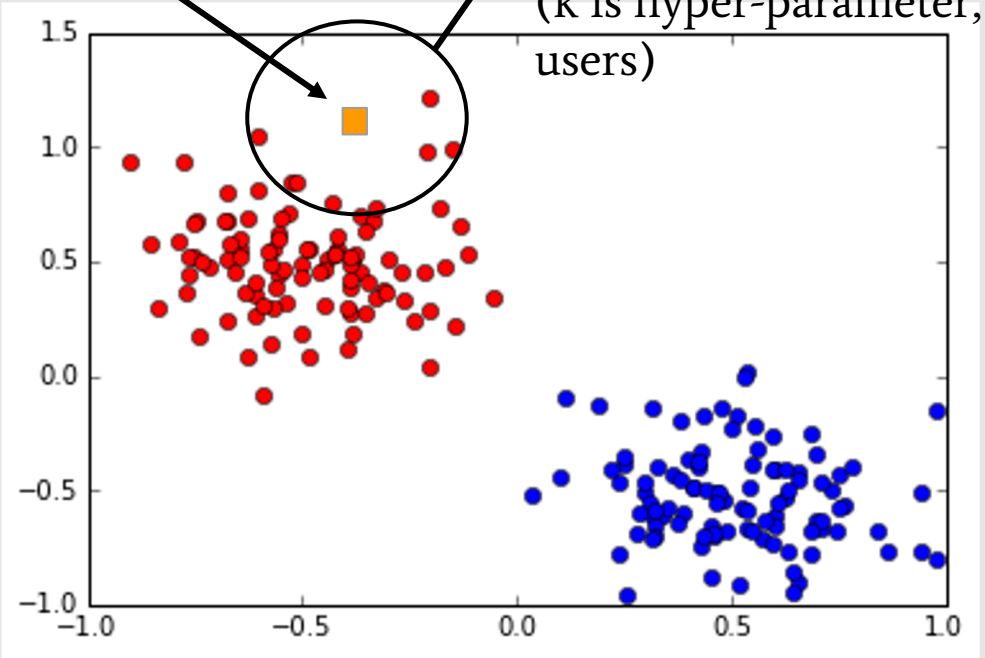
What label does this data has?



Solution 1: k-NN

What label does this data has?

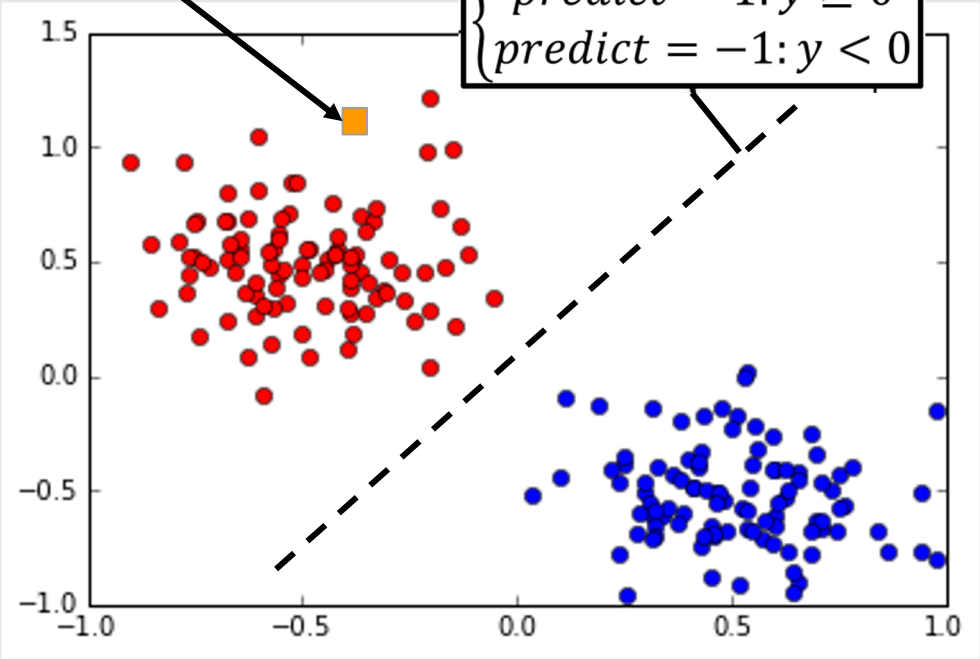
Using k-NN
A baseline model with no parameters
(k is hyper-parameter, given by the users)



Solution 2: linear classifier

What label does this data has?

$$y = w_1 \times x_1 + w_2 \times x_2$$
$$\begin{cases} predict = 1 : y \geq 0 \\ predict = -1 : y < 0 \end{cases}$$



Model parameter:
w_1, w_2

Other Classification Problems

{X, Y}: X is the data set, Y is the respective label set

A model maps some x to some y: $f(x; \Theta) \rightarrow y$

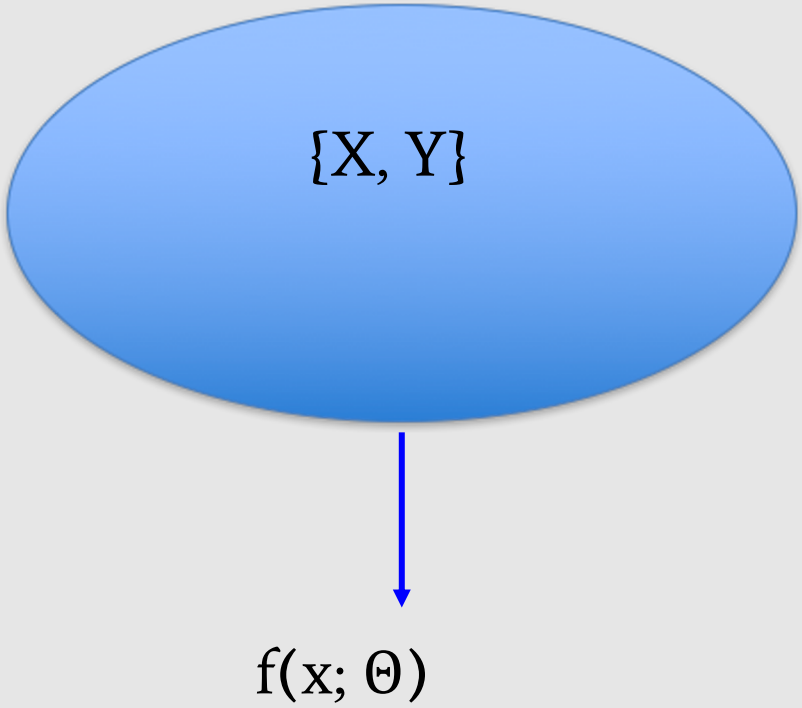
$f(x; \Theta)$

2, 9, 6, 1, 3
3, 9, 4, 0, 3
6, 9, 4, 1, 9
9, 5, 0, 8, 5
8, 8, 3, 5, 0

To make a choice, we need to know

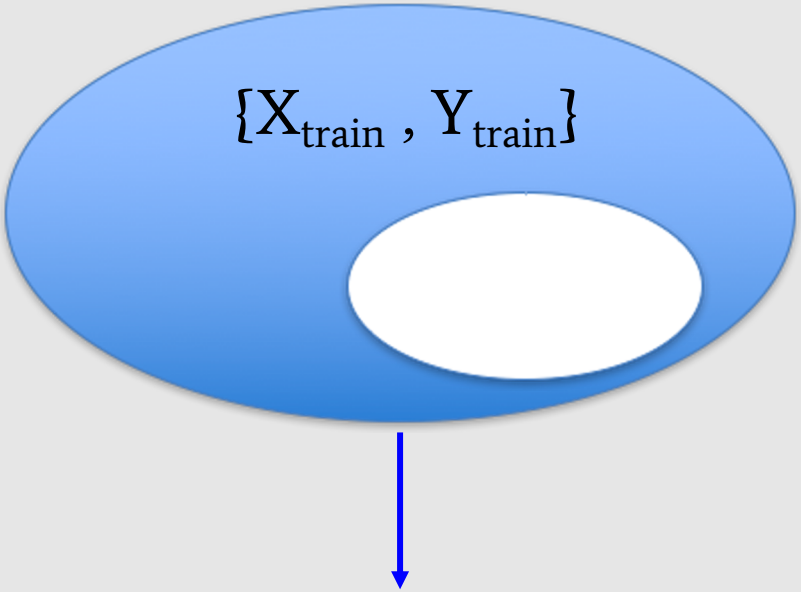
- How good is each solution?
- How can we measure the accuracy?

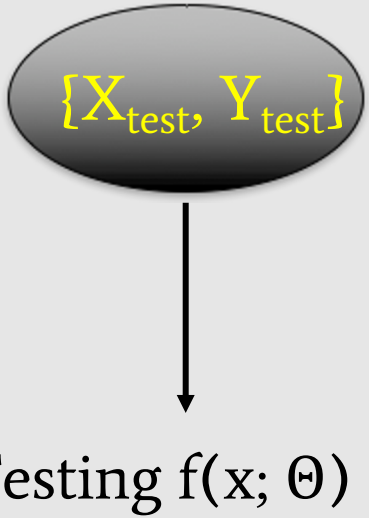
Training and testing



Training and testing

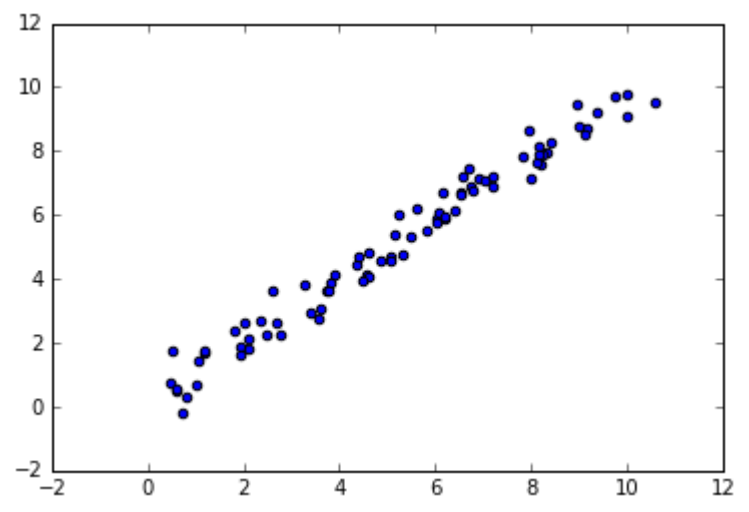
We learn the model from training data and verify it with the testing data.
A model ***overfits*** if it performs well on training data, but poorly on testing data



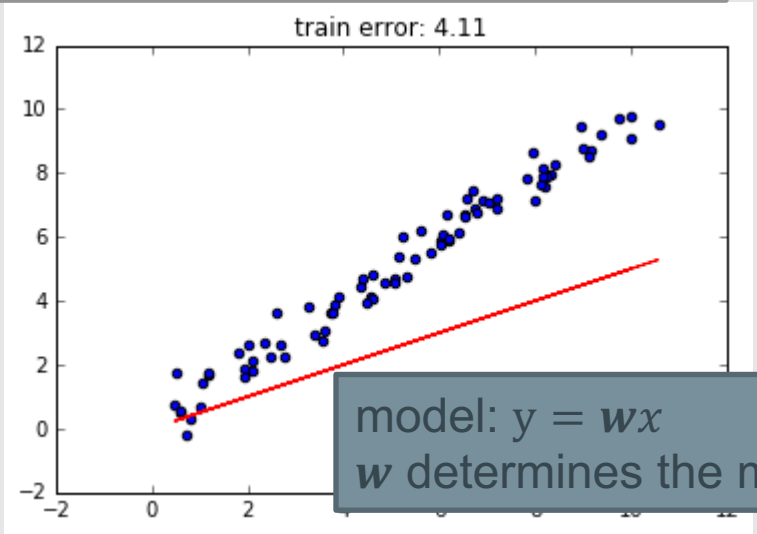


Regression with gradient descent

Regression to fit a line



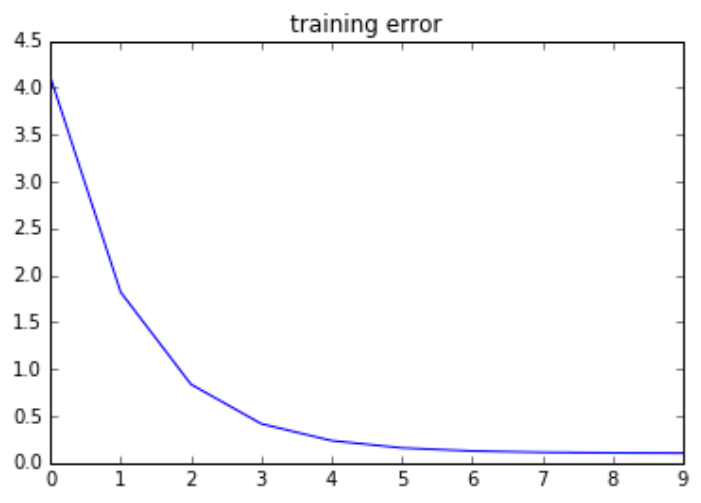
Data:
- Generated with $y = x$
- Plus noises (say from observation)

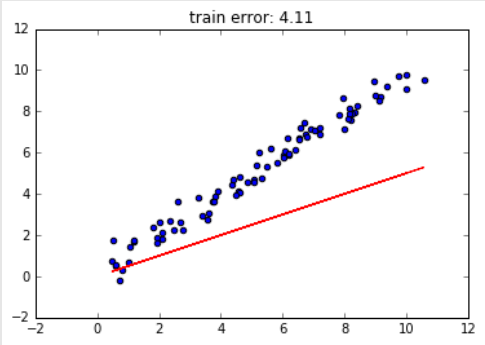


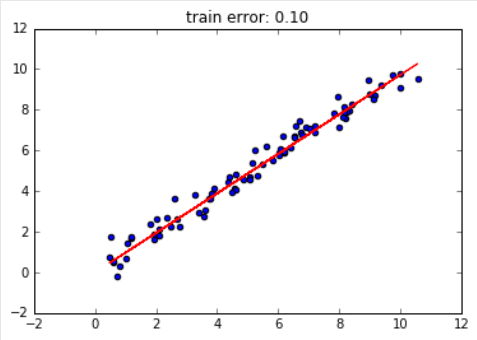
model: $y = \boldsymbol{w}x$
$\boldsymbol{w}$ determines the model!

Error and gradient

- Model: y = wx
- Model parameter: w
- Sum of error: $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- Goal: find w such that E is minimized.



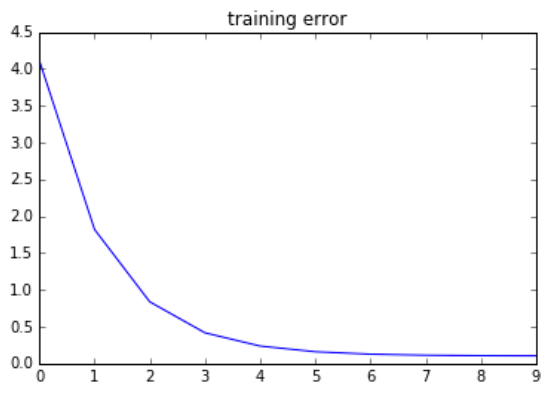


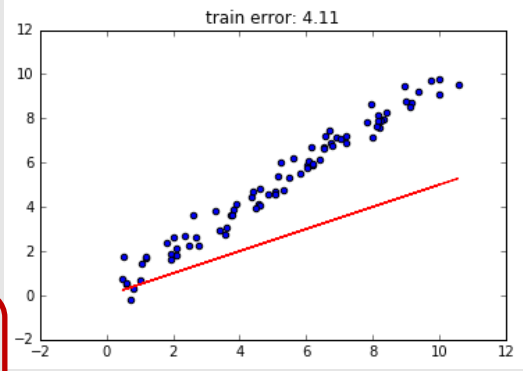


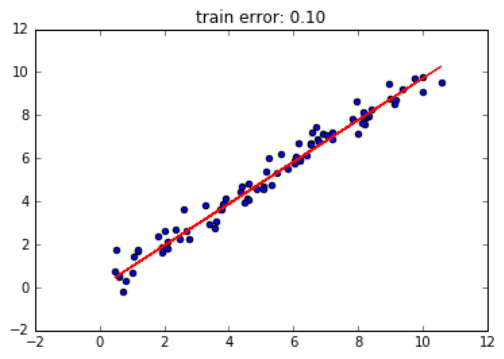
Error and gradient

- Model: $pred = wx$
- Model parameter: w
- Sum of error: $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- **Goal: find w such that E is minimized.**

2 is not necessary, only to make calculation in later steps easier.

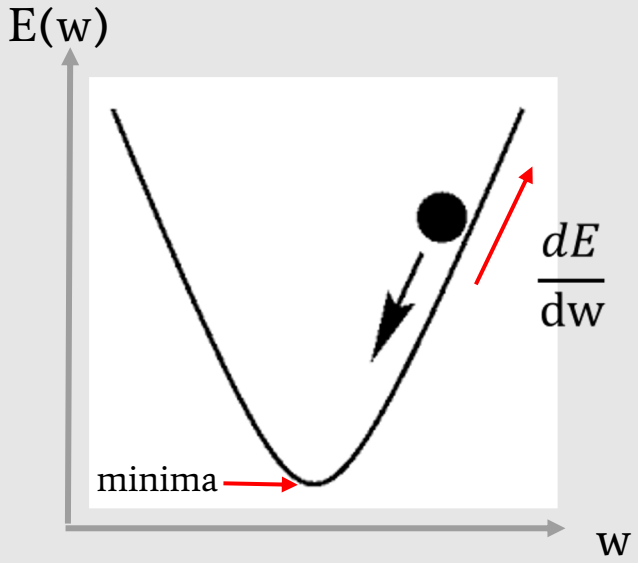






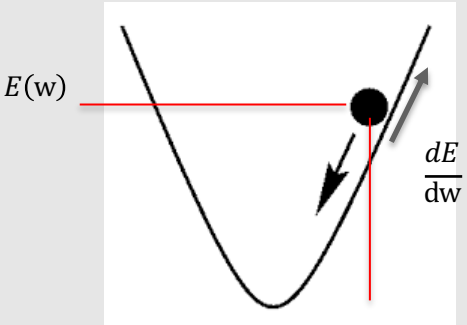
Gradient descent

- Optimization problem: Minimize E(w)
- E(w) is quadratic
- Can we search [-∞, +∞] for w, finding the one that gives the minimum of E?



Gradient descent: training is learning

A *global* optimization problem : Minimize $E(\mathrm{w})$



- $E(\mathrm{w})$: total error (or cost function)

$$\Delta E = \frac{dE}{dw}\Delta w \text{ , } * \text{ assume } \Delta w \text{ is very small}$$

where $\frac{dE}{dw}$ is the rate of change in total error (or cost)

- Question: how to choose Δw to guarantee ΔE is negative, i.e., guarantee that total error will reduce

w

Gradient descent: training is learning

A *global* optimization problem : Minimize $E(\mathrm{w})$

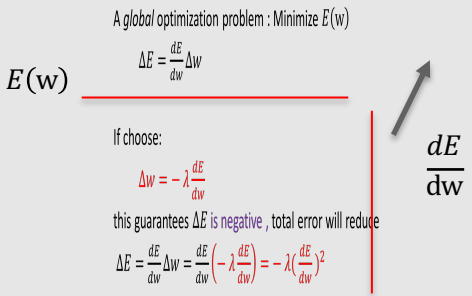
$$\Delta E = \frac{dE}{dw}\Delta w$$

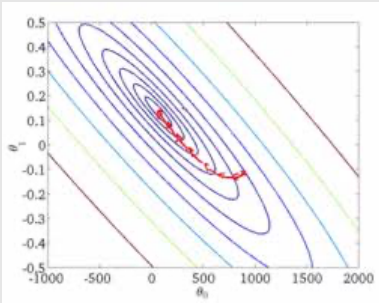
If choose:

$$\Delta w = -\lambda \frac{dE}{dw}$$

this guarantees ΔE is negative , total error will reduce

$$\Delta E = \frac{dE}{dw}\Delta w = \frac{dE}{dw}\left(-\lambda \frac{dE}{dw}\right) = -\lambda\left(\frac{dE}{dw}\right)^2$$





w

Gradient derivation (calculus)

$$f(w) = (5w - 9)^2$$

$$\frac{df}{dw} = ?$$

Gradient derivation (calculus)

$$f(w) = (5w - 9)^2$$

$$\frac{df}{dw} = 2(5w - 9) \times 5$$

Gradient derivation (calculus)

Therefore, with $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$

$$\frac{dE}{dw} = ?$$

Gradient derivation (calculus)

- Therefore, with $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)x_i$

Gradient derivation (calculus)

- Therefore, with $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)x_i$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i}^{n}(pred_i - y_i)x_i$

- Interpretation: average prediction error weighted by input contribution

Gradient derivation (calculus)

- Therefore, with $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i=1}^{n}(wx_i - y_i)x_i$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i}^{n}(pred_i - y_i)x_i$
- Interpretation: average prediction error weighted by input contribution

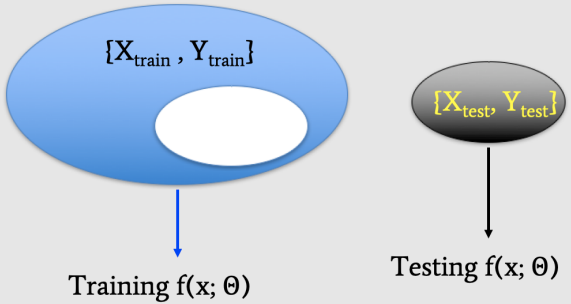
- Assign initial value for w (w=0.5)
- $w = w - \lambda\frac{dE}{dW}$
- do it until converge

Data: training and testing

```
def make_data(n, w=1, scale=1):
    """ A simple y = wx curve, with noisy displacement on both
        both x and y axis; change scale to change the range
    """
    linear_data = [sample.Sample('', [float(x)/scale, float(x)/scale * w], '') for x in range(n)]
    noise = util.genDistribution(xSD=0.3, ySD=0.3, n=n)
    data = [linear_data[i] + noise[i] for i in range(n)]
    return data
```

```
# Generating random data
data = make_data(num_samples, w=1, scale=10)

# Make a random split of the data
random.shuffle(data)
train_data = data[:80]
test_data = data[80:]
```



Regression with gradient descent

```
63      # training loop
64      while True:
65
66          # predict and compute error
67          pred_y = [w*x for x in d_x]
68          error = compute_error(pred_y, d_y)
69          error_history.append(error)
70
71          plt.scatter(d_x, d_y)
72          plt.plot(d_x, pred_y, 'r')
73          title_str = "train error: %0.2f" % error
74          plt.title(title_str)
75          plt.show()
76
77          steps += 1
78          if error <= margin or steps >= num_iter:
79              break
80
81          # gradient descent
82          grad = compute_grad(pred_y, d_y, d_x)
83          w -= learning_rate * grad
84
85          if VERBOSE and input("cont? (y/n) ") == 'n':
86              break
```

$$w = w - \lambda \frac{dE}{dW}$$

Regression with gradient descent

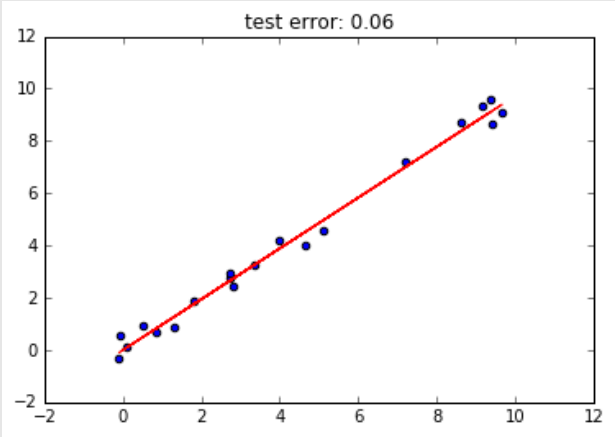
Complete comptue_error and compute_grad:

- $E(w) = \frac{1}{2n}\sum_{i=1}^{n}(wx_i - y_i)^2$
- $\frac{dE}{dw} = \frac{1}{n}\sum_{i}^{n}(pred_i - y_i)x_i$

```
24 def compute_error(pred, truth):
25     """ mean square error:
26         sum of (pred[i] - truth[i])^2, divided by 2*length
27     """
28     return regression_helper.compute_error(pred, truth)
29
30 def compute_grad(pred, truth, x):
31     """ gradient is mean of (pred[i] - truth[i]) * x[i]
32     """
33     return regression_helper.compute_grad(pred, truth, x)
```

The model never looks at test data

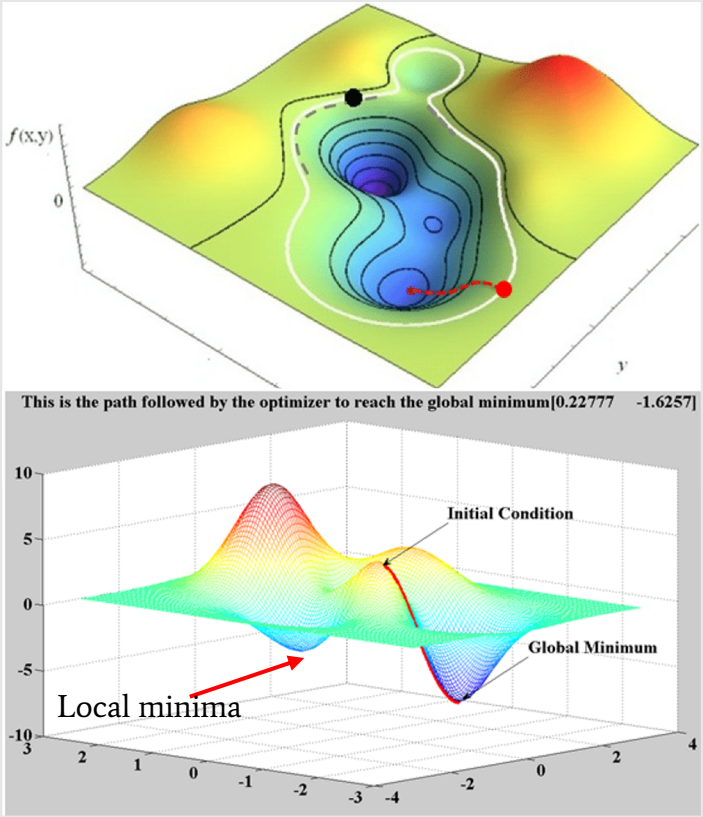
```
88    # testing on test data
89    d_x = [d.getFeatures()[0] for d in test_data]
90    d_y = [d.getFeatures()[1] for d in test_data]
91    pred_y = [w*x for x in d_x]
92    error = compute_error(pred_y, d_y)
93
94    plt.scatter(d_x, d_y)
95    plt.plot(d_x, pred_y, 'r')
96    title_str = "test error: %0.2f" % error
97    plt.title(title_str)
98    plt.show()
99
100   plt.plot(error_history)
101   plt.title('training error')
102   plt.show()
```



Exercise

- Can you write your own code that can fit a line with 2 parameters y = $ax + b$?
- You need to use gradient decent.

Widely used in solving optimization problems

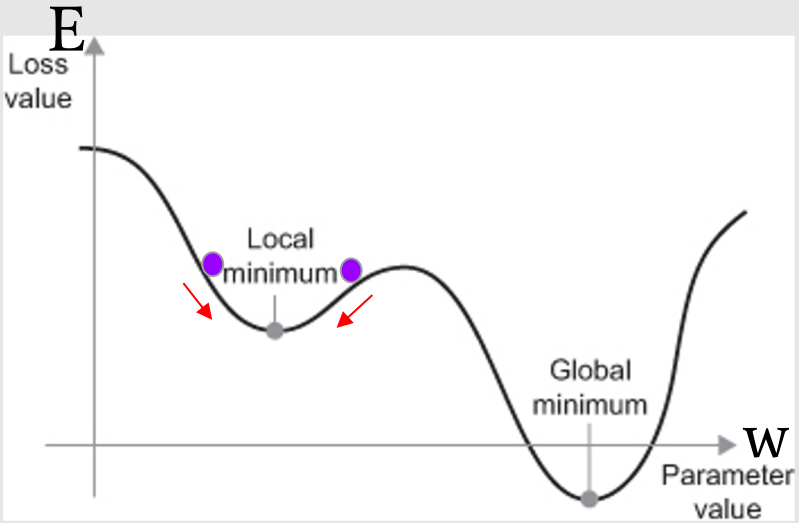


Gradient descent also works with more intricate surfaces (e.g., multi-variable functions).

- Not always be able to find the global optima, as the descending process may be trapped by local minima.

Question:
What can we do to avoid being trapped?

What can we do to avoid being trapped?



$$\Delta W = -\lambda \frac{dE}{dw}$$

- One key is the learning rate λ
  - A large λ may help the ball rolling over the local minima.
  - But if λ is too lager, we might jump over the global minima also.

Data Science: a review

Data science review

Probability

- Joint probability $P(A \cap B)$
- Unions of events $P(A \cup B)$
- Conditional probability $P(A|B)$
  - We build a dice by OOP
- Monte Carlo simulation (approximation by multiple simulating and averaging)
  - We verify it by measuring Pi
- Normal distribution and law of large numbers
  - μ: the mean of the data points
  - σ: the standard deviation;

Data science review

Machine learning

- Machine learning is PET
  - Tom Mitchell: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

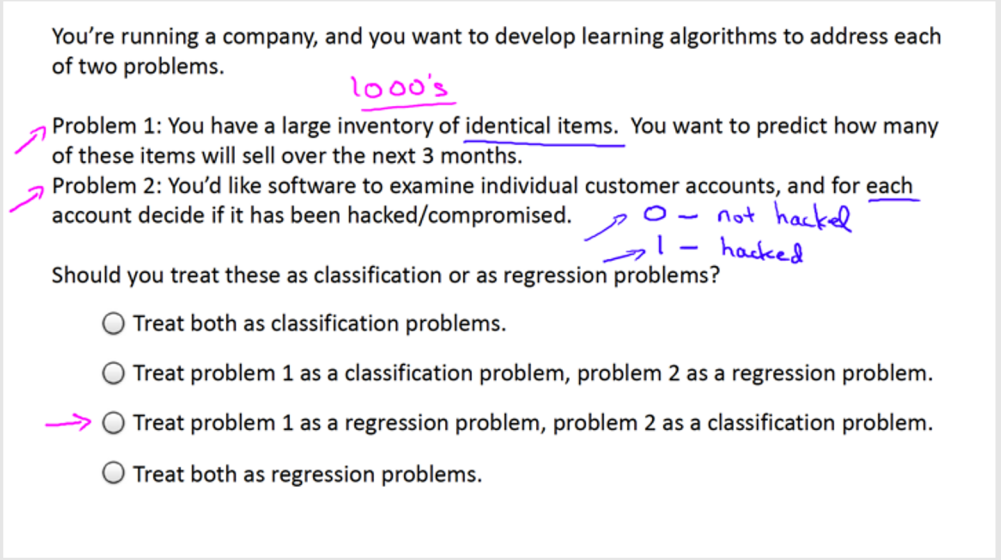
"A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

- ○ Classifying emails as spam or not spam. T ←
- ○ Watching you label emails as spam or not spam. E ←
- ○ The number (or fraction) of emails correctly classified as spam/not spam. P ←
- ○ None of the above—this is not a machine learning problem.

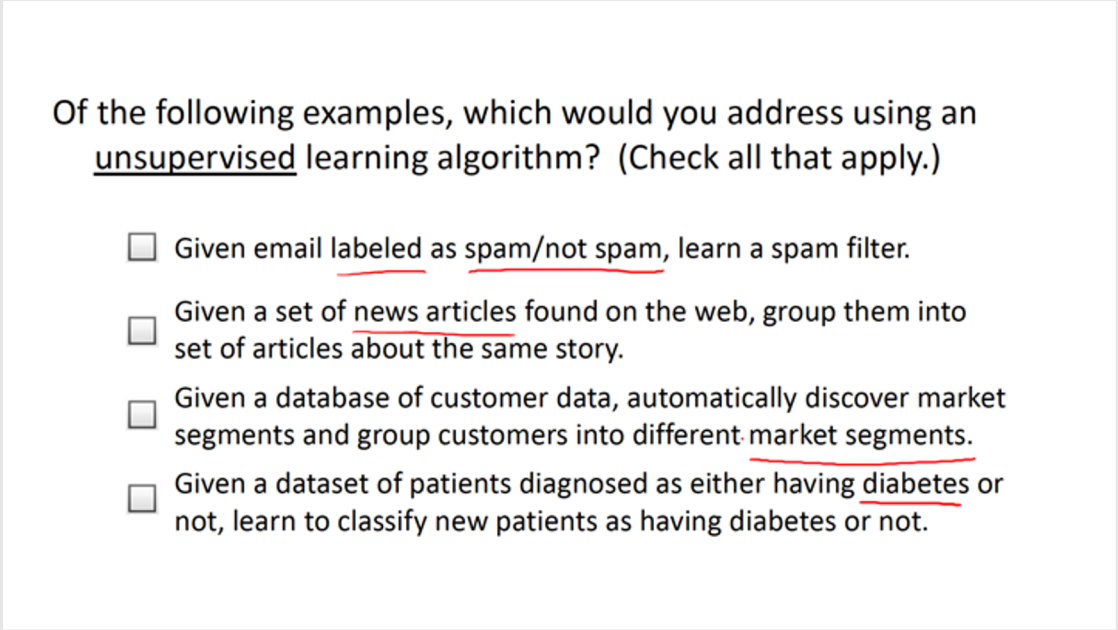
Data science review

- Supervised learning
  - The data set given contains labels (“right answers”)
  - Model parameters obtain by training
  - Two tasks: regression, classification
  - We tried k-NN and regression
    - We use the sample class
  - We use gradient descent to find the parameters.



Data science review

- Unsupervised learning
  - The given data set have no labels
  - Probe the structure of the data set
    - e.g., Clustering
  - K-means algorithm
    - How it works?
    - The sample class is used.



Reminders

- Make sure that **matplotlib** works properly with your computers since we may use it in the exams.
  - The plots in these in-class exercises can be plotted correctly in your laptops