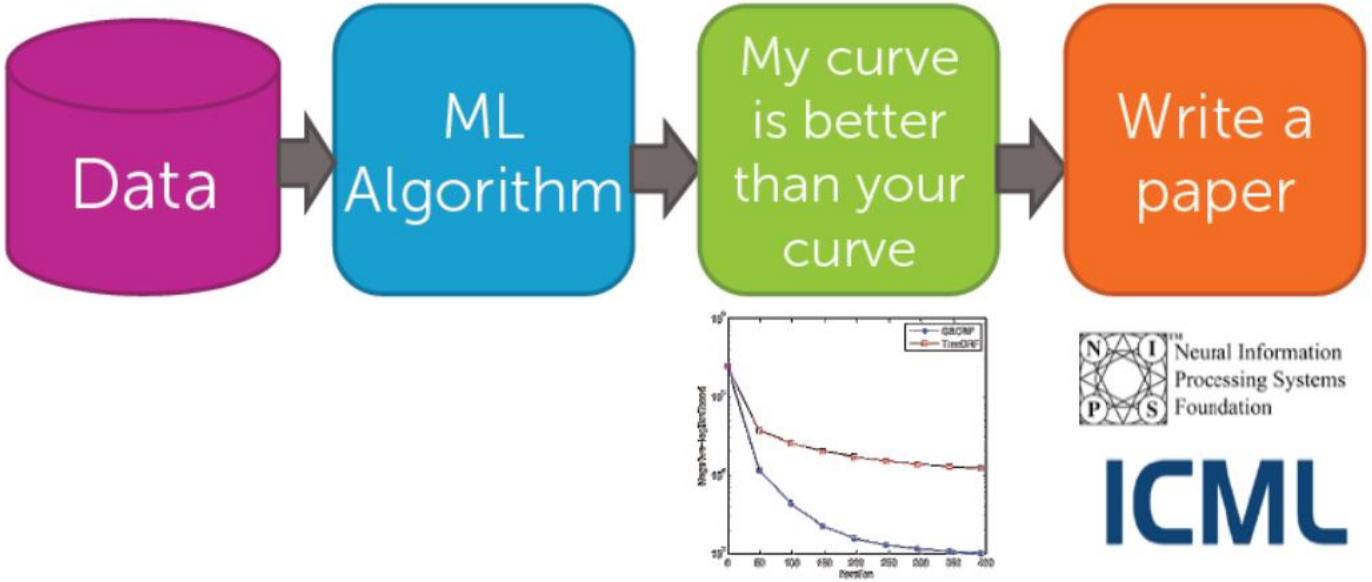
# What is Machine Learning?

- 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.
- Machine learning is a subfield of computer science that is concerned with building algorithms which, to be useful, rely on a collection of examples of some phenomenon. These examples can come from nature, be handcrafted by humans or generated by another algorithm.

- Machine learning teaches computers to do what comes naturally to humans and animals: learn from experience. Machine learning algorithms use computational methods to “learn” information directly from data without relying on a predetermined equation as a model. The algorithms adaptively improve their performance as the number of samples available for learning increases.

# Old view of ML



# There are two sides to machine learning:

- **Practical Machine Learning:** This is about querying databases, cleaning data, writing scripts to transform data and gluing algorithm and libraries together and writing custom code to squeeze reliable answers from data to satisfy difficult and ill defined questions. It's the mess of reality.
- **Theoretical Machine Learning**: This is about math and abstraction and idealized scenarios and limits and beauty and informing what is possible. It is a whole lot neater and cleaner and removed from the mess of reality.

# Real-World Applications

- With the rise in big data, machine learning has become particularly important for solving problems in areas like these:
- Computational finance, for credit scoring and algorithmic trading
- Image processing and computer vision, for face recognition, motion detection, and object detection
- Computational biology, for tumor detection, drug discovery, and DNA sequencing
- Energy production, for price and load forecasting
- Automotive, aerospace, and manufacturing, for predictive maintenance
- Natural language processing

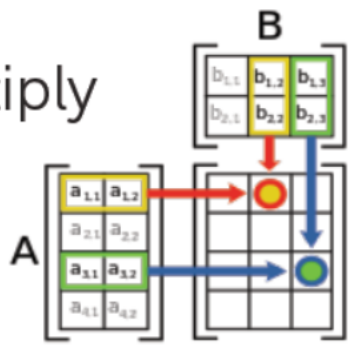

Disruptive companies
differentiated by
INTELLIGENT
APPLICATIONS
using
Machine Learning
amazon
Retail
Google
PageRank
Search
livingsocial
Coupons
NETFLIX
Movie
Distribution
LinkedIn
Networking
PANDORA
Music
Obama'08
Campaigning
Google
Adsense
Advertising
Zillow
Real Estate
glassdoor
Human
Resources
Avvo
Legal
Advice
eHarmony
Dating
fitbit
Wearables
UBER
Taxis
RelateIQ
CRM

- What math skills are required to learn machine learning?
- *You absolutely will need to be comfortable with basic linear algebra (manipulating vectors and matrices) and working with logarithmic and exponential functions.*
- *You'll need to know Linear Algebra through Eigenvectors if you want things to be "easy".*
- Is a strong background in maths a total requisite for ML?
- *You do want to have some familiarity with probability, linear algebra, linear programming, and multivariable calculus.*
- What skills are needed for machine learning jobs?
- *First, you need to have a decent CS/Math background. ML is an advanced topic so most textbooks assume that you have that background.*
- *Statistics, Probability, distributed computing, and Statistics.*

# The machine learning pipeline

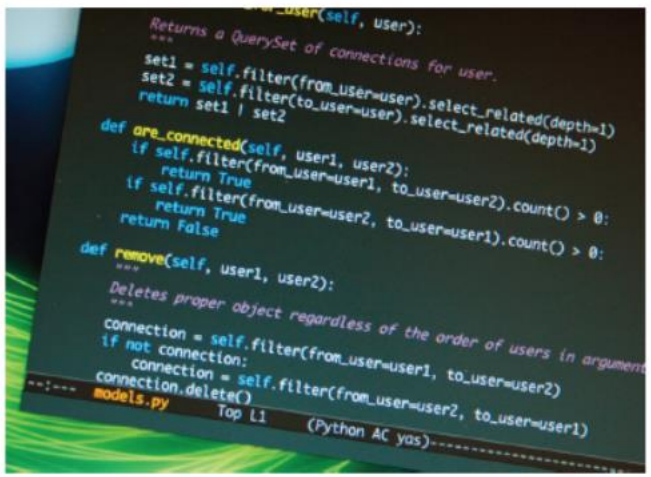
# Math background

- Basic calculus
  - Concept of derivatives
- Basic linear algebra
  - Vectors
  - Matrices
  - Matrix multiply



# Programming experience

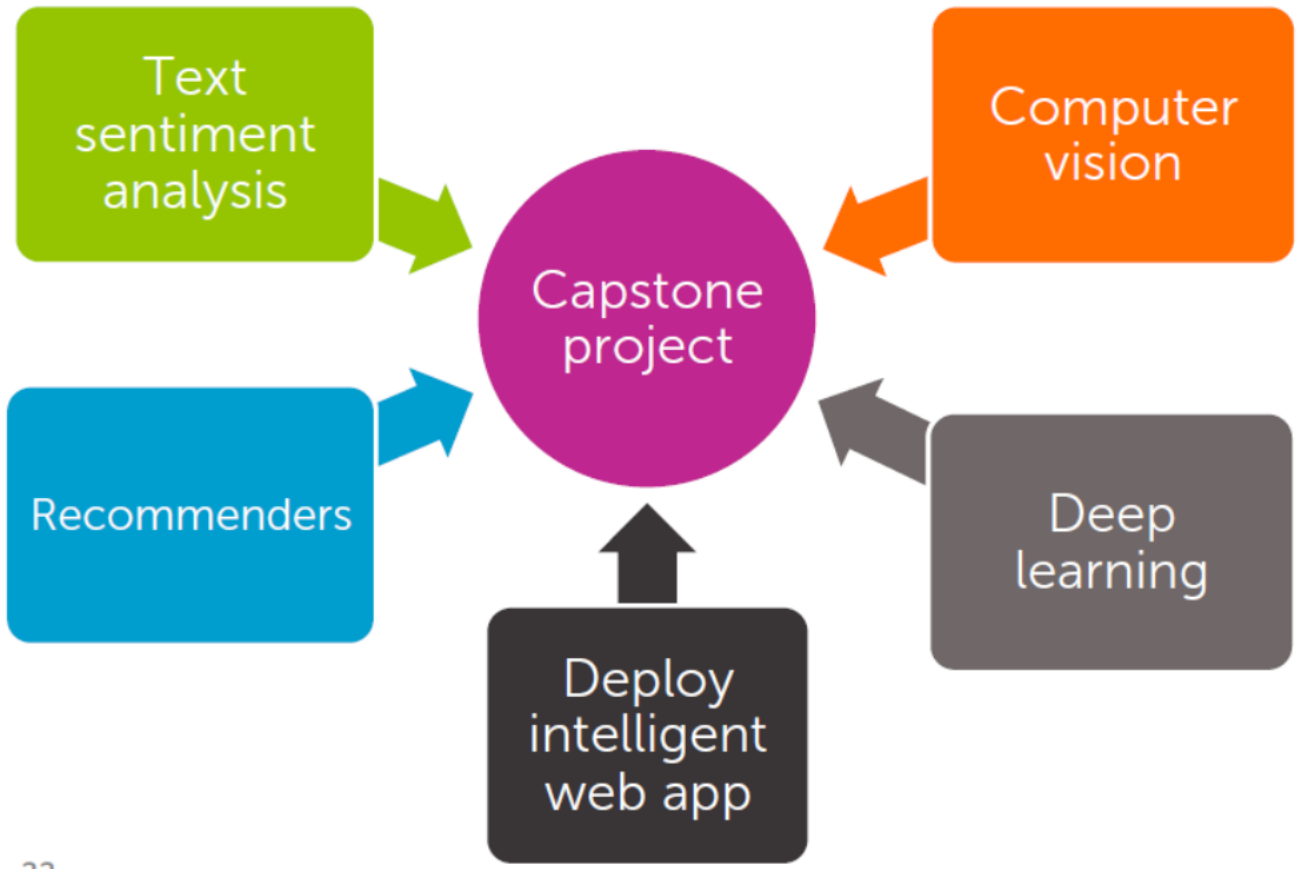
- Basic Python used
  - Can pick up along the way if knowledge of other language



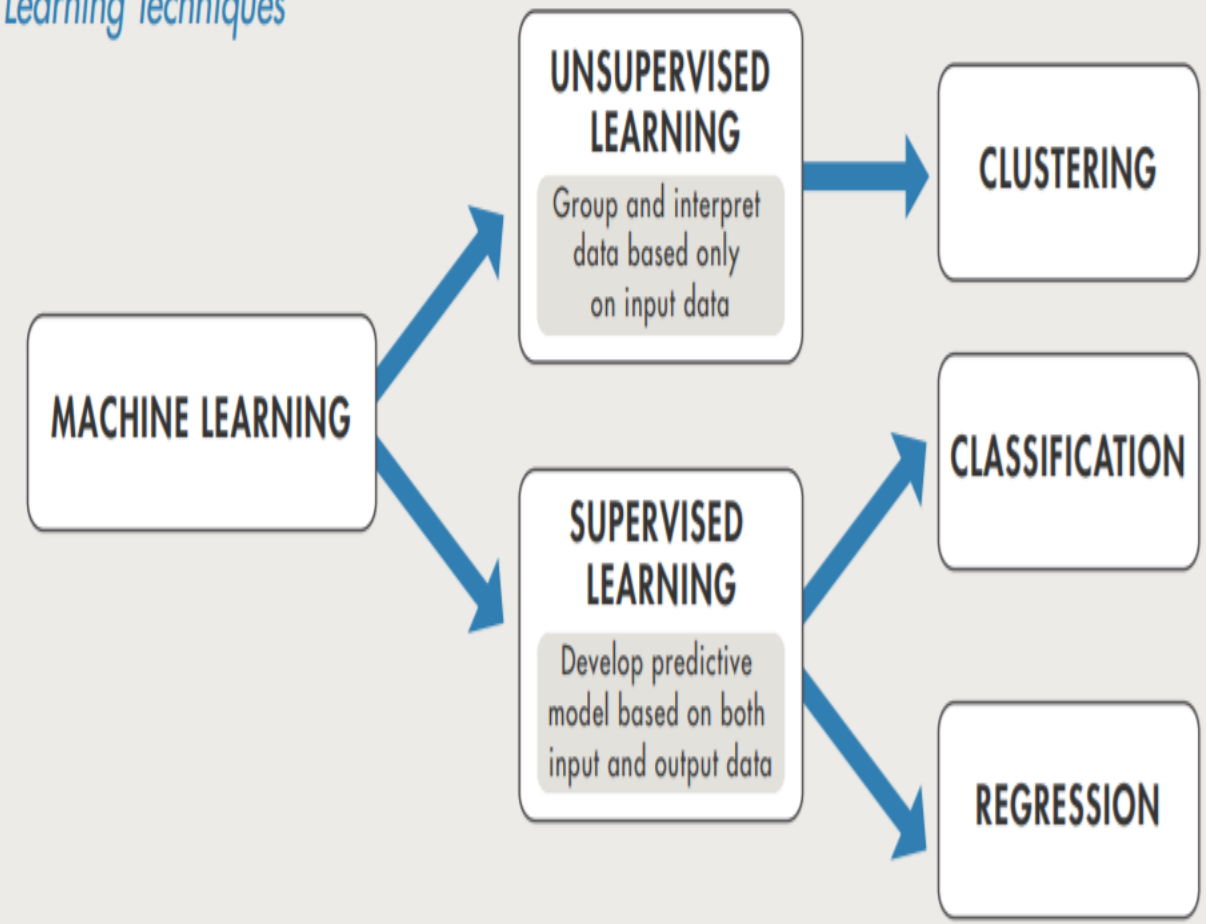

# Tools you'll need

- Python. Python 3 is the best option.
- IPython and the Jupyter Notebook. (FKA IPython and IPython Notebook.)
- Some scientific computing packages:
  - numpy
  - pandas
  - scikit-learn
  - matplotlib
- You can install Python 3 and all of these packages in a few clicks with the Anaconda Python distribution. Anaconda is popular in Data Science and Machine Learning communities.
- If you're using Python 2.7, don't worry. You don't have to migrate to Python 3 just for this guide. Also, if you're using pip/virtualenv instead of Anaconda, that's alright too! And re: installing packages, this is a helpful doc: conda vs. pip vs. virtualenv

# We will do something even more exciting...



## Machine Learning Techniques



# Supervised Learning

The aim of supervised machine learning is to build a model that makes predictions based on evidence in the presence of uncertainty. A supervised learning algorithm takes a known set of input data and known responses to the data (output) and trains a model to generate reasonable predictions for the response to new data.

Supervised learning uses classification and regression techniques to develop predictive models.

- **Classification techniques** predict discrete responses—for example, whether an email is genuine or spam, or whether a tumor is cancerous or benign. Classification models classify input data into categories. Typical applications include medical imaging, speech recognition, and credit scoring.
- **Regression techniques** predict continuous responses—for example, changes in temperature or fluctuations in power demand. Typical applications include electricity load forecasting and algorithmic trading.

## *Using Supervised Learning to Predict Heart Attacks*

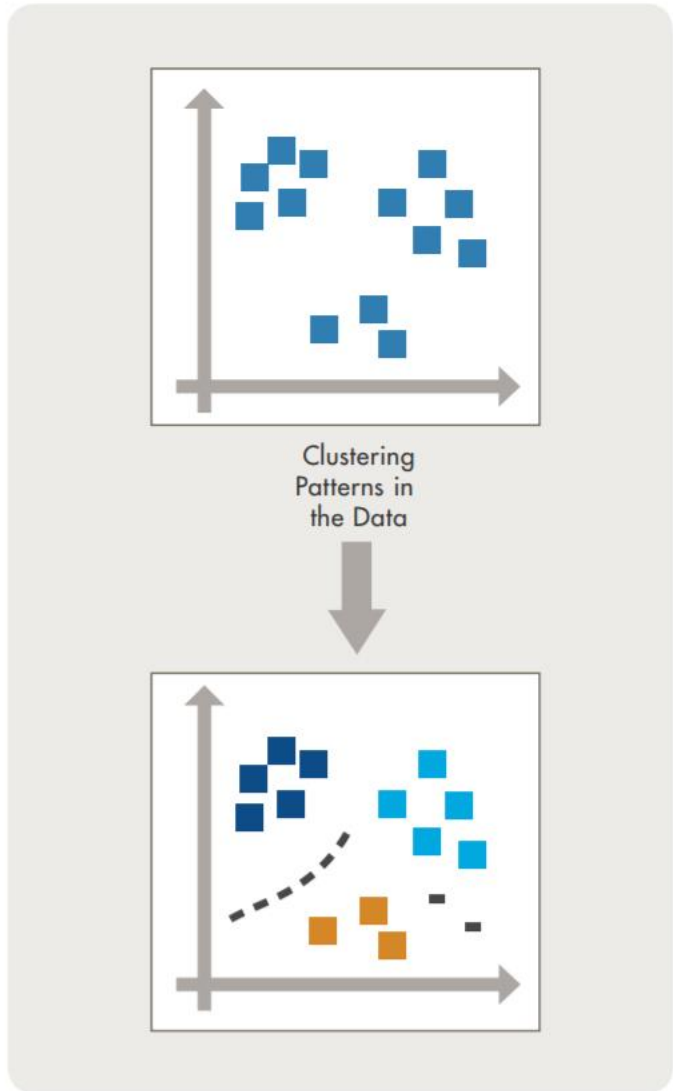
Suppose clinicians want to predict whether someone will have a heart attack within a year. They have data on previous patients, including age, weight, height, and blood pressure. They know whether the previous patients had heart attacks within a year. So the problem is combining the existing data into a model that can predict whether a new person will have a heart attack within a year.

# Unsupervised Learning

Unsupervised learning finds hidden patterns or intrinsic structures in data. It is used to draw inferences from datasets consisting of input data without labeled responses.

**Clustering** is the most common unsupervised learning technique. It is used for exploratory data analysis to find hidden patterns or groupings in data.

Applications for clustering include gene sequence analysis, market research, and object recognition.

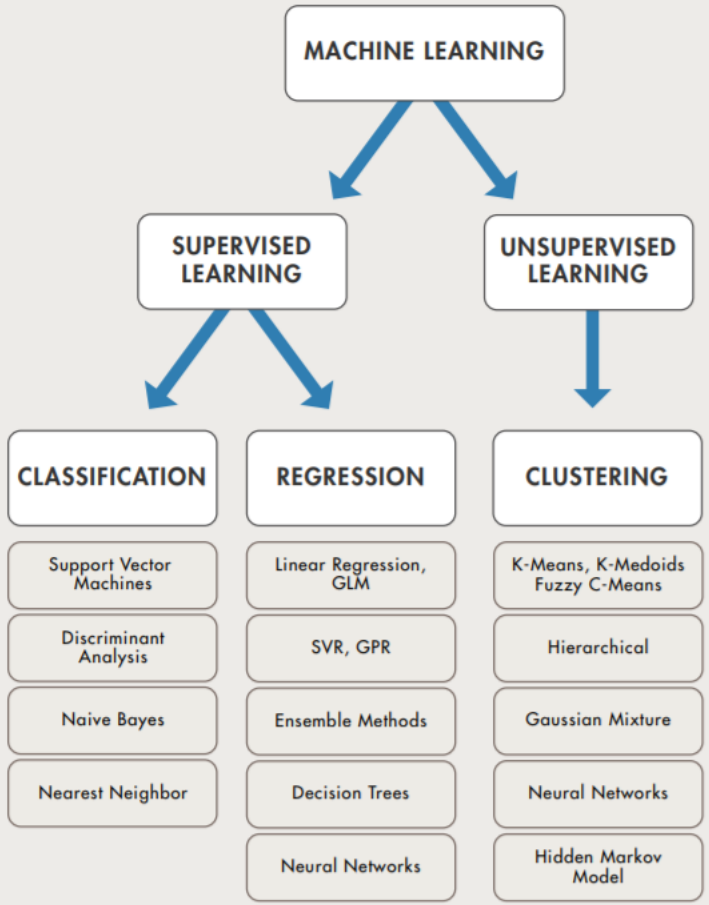


# How Do You Decide Which Algorithm to Use?

Choosing the right algorithm can seem overwhelming—there are dozens of supervised and unsupervised machine learning algorithms, and each takes a different approach to learning.

There is no best method or one size fits all. Finding the right algorithm is partly just trial and error—even highly experienced data scientists can't tell whether an algorithm will work without trying it out. But algorithm selection also depends on the size and type of data you're working with, the insights you want to get from the data, and how those insights will be used.

*Selecting an Algorithm*



# When Should You Use Machine Learning?

Consider using machine learning when you have a complex task or problem involving a large amount of data and lots of variables, but no existing formula or equation. For example, machine learning is a good option if you need to handle situations like these:

*Hand-written rules and equations are too complex—as in face recognition and speech recognition.*

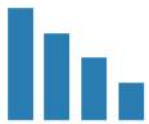
*The rules of a task are constantly changing—as in fraud detection from transaction records.*

*The nature of the data keeps changing, and the program needs to adapt—as in automated trading, energy demand forecasting, and predicting shopping trends.*

# Machine Learning Challenges

Most machine learning challenges relate to handling your data and finding the right model.

**Data comes in all shapes and sizes.** Real-world datasets can be messy, incomplete, and in a variety of formats. You might just have simple numeric data. But sometimes you're combining several different data types, such as sensor signals, text, and streaming images from a camera.



**Preprocessing your data might require specialized knowledge and tools.** For example, to select features to train an object detection algorithm requires specialized knowledge of image processing. Different types of data require different approaches to preprocessing.

**It takes time to find the best model to fit the data.** Choosing the right model is a balancing act. Highly flexible models tend to overfit data by modeling minor variations that could be noise. On the other hand, simple models may assume too much. There are always tradeoffs between model speed, accuracy, and complexity.

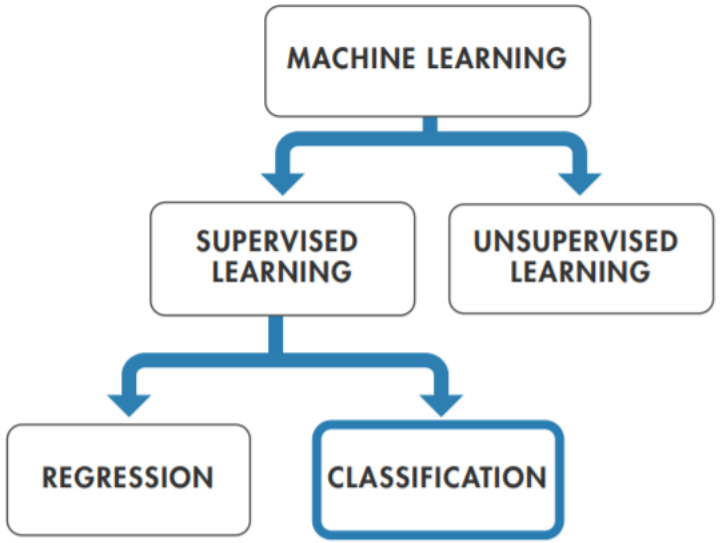
Sounds daunting? Don't be discouraged. Remember that trial and error is at the core of machine learning—if one approach or algorithm doesn't work, you simply try another. But a systematic workflow will help you get off to a smooth start.

# Questions to Consider Before You Start

Every machine learning workflow begins with three questions:

- What kind of data are you working with?
- What insights do you want to get from it?
- How and where will those insights be applied?

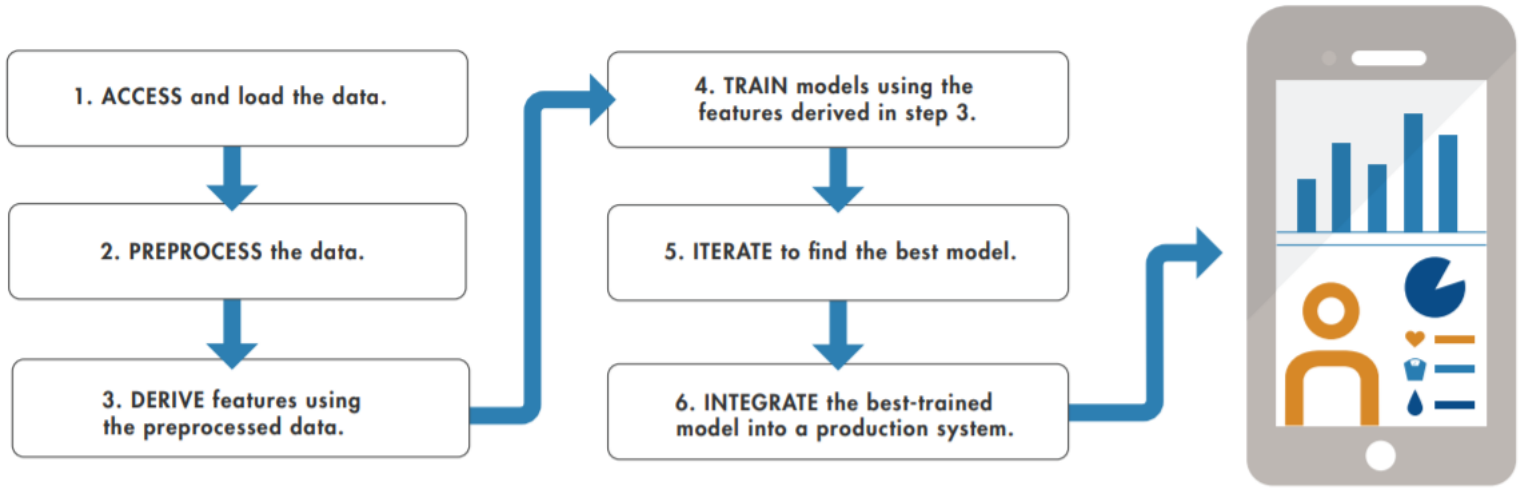
Your answers to these questions help you decide whether to use supervised or unsupervised learning.




Choose supervised learning if you need to train a model to make a prediction--for example, the future value of a continuous variable, such as temperature or a stock price, or a classification—for example, identify makes of cars from webcam video footage.

Choose unsupervised learning if you need to explore your data and want to train a model to find a good internal representation, such as splitting data up into clusters.

# Workflow at a Glance

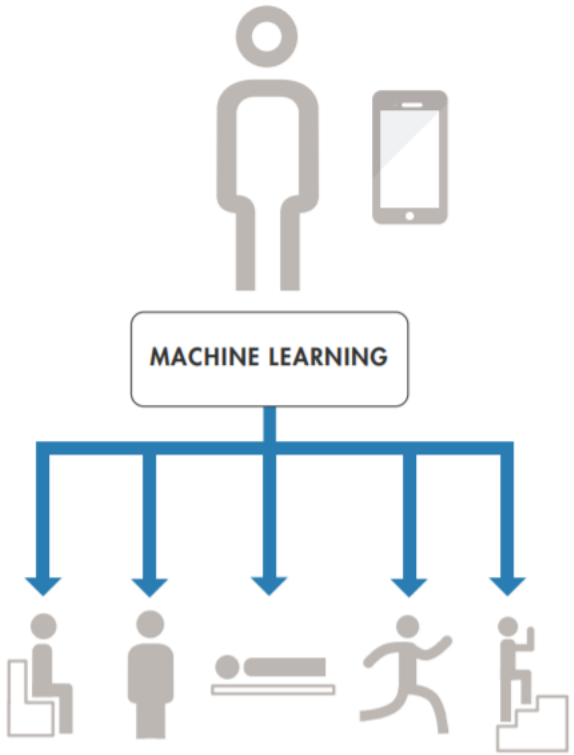


# Training a Model to Classify Physical Activities

This example is based on a cell phone health-monitoring app. The input consists of three-axial sensor data from the phone's accelerometer and gyroscope. The responses, (or output), are the activities performed–walking, standing, running, climbing stairs, or lying down.

We want to use the input data to train a classification model to identify these activities. Since our goal is classification, we'll be applying supervised learning.

The trained model (or classifier) will be integrated into an app to help users track their activity levels throughout the day.



# 1 Step One: Load the Data

To load data from the accelerometer and gyroscope we do the following:

1. Sit down holding the phone, log data from the phone, and store it in a text file labeled "Sitting."
2. Stand up holding the phone, log data from the phone, and store it in a second text file labeled "Standing."
3. Repeat the steps until we have data for each activity we want to classify.

We store the labeled data sets in a text file. A flat file format such as text or CSV is easy to work with and makes it straightforward to import data.

Machine learning algorithms aren't smart enough to tell the difference between noise and valuable information. Before using the data for training, we need to make sure it's clean and complete.

# 2 Step Two: Preprocess the Data

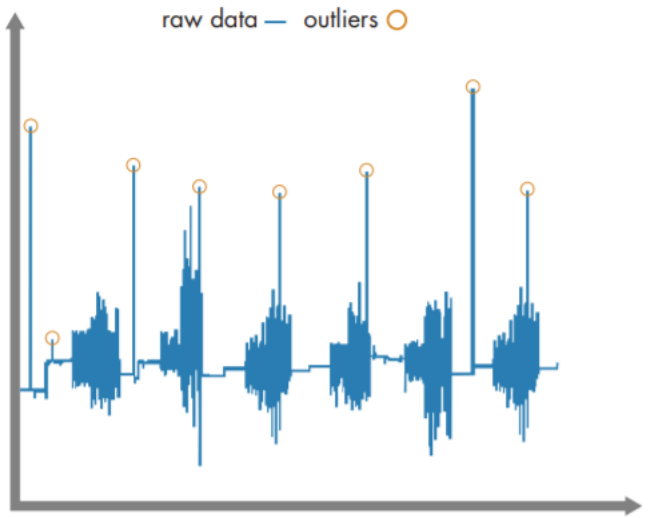
We import the data into and plot each labeled set.
To preprocess the data we do the following:

1. Look for outliers–data points that lie outside the rest of the data.

We must decide whether the outliers can be ignored or whether they indicate a phenomenon that the model should account for. In our example, they can safely be ignored (it turns out that we moved unintentionally while recording the data).

2. Check for missing values (perhaps we lost data because the connection dropped during recording).

We could simply ignore the missing values, but this will reduce the size of the data set. Alternatively, we could substitute approximations for the missing values by interpolating or using comparable data from another sample.




*Outliers in the activity-tracking data.*

In many applications, outliers provide crucial information. For example, in a credit card fraud detection app, they indicate purchases that fall outside a customer's usual buying patterns.

# 2 Step Two: Preprocess the Data *continued*

3. Remove gravitational effects from the accelerometer data so that our algorithm will focus on the movement of the subject, not the movement of the phone. A simple high-pass filter such as a biquad filter is commonly used for this.
4. Divide the data into two sets. We save part of the data for testing (the test set) and use the rest (the training set) to build models. This is referred to as holdout, and is a useful cross-validation technique.

By testing your model against data that wasn't used in the modeling process, you see how it will perform with unknown data.

# 3 Step Three: Derive Features

Deriving features (also known as feature engineering or feature extraction) is one of the most important parts of machine learning. It turns raw data into information that a machine learning algorithm can use.

For the activity tracker, we want to extract features that capture the frequency content of the accelerometer data. These features will help the algorithm distinguish between walking (low frequency) and running (high frequency). We create a new table that includes the selected features.

Use feature selection to:

- Improve the accuracy of a machine learning algorithm
- Boost model performance for high-dimensional data sets
- Improve model interpretability
- Prevent overfitting

# 3 Step Three: Derive Features *continued*

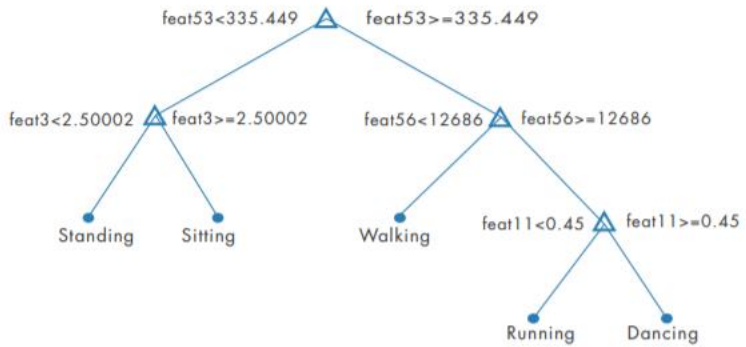
The number of features that you could derive is limited only by your imagination. However, there are a lot of techniques commonly used for different types of data.

| Data Type | Feature Selection Task | Techniques |
|---|---|---|
| **Sensor data** | Extract signal properties from raw sensor data to create higher-level information | **Peak analysis** – perform an fft and identify dominant frequencies<br><br>**Pulse and transition metrics** – derive signal characteristics such as rise time, fall time, and settling time<br><br>**Spectral measurements** – plot signal power, bandwidth, mean frequency, and median frequency |
| **Image and video data** | Extract features such as edge locations, resolution, and color | **Bag of visual words** – create a histogram of local image features, such as edges, corners, and blobs<br><br>**Histogram of oriented gradients (HOG)** – create a histogram of local gradient directions<br><br>**Minimum eigenvalue algorithm** – detect corner locations in images<br><br>**Edge detection** – identify points where the degree of brightness changes sharply |
| **Transactional data** | Calculate derived values that enhance the information in the data | **Timestamp decomposition** – break timestamps down into components such as day and month<br><br>**Aggregate value calculation** – create higher-level features such as the total number of times a particular event occurred |

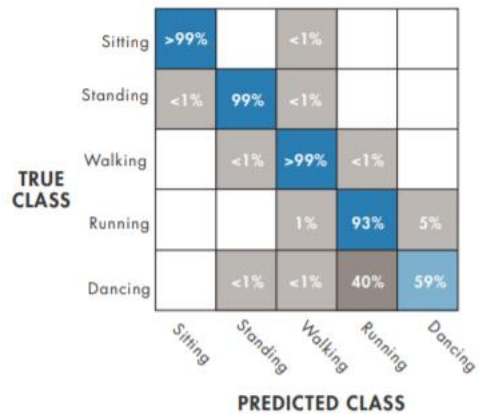
# 4 Step Four: Build and Train the Model

When building a model, it's a good idea to start with something simple; it will be faster to run and easier to interpret.

We start with a basic decision tree.



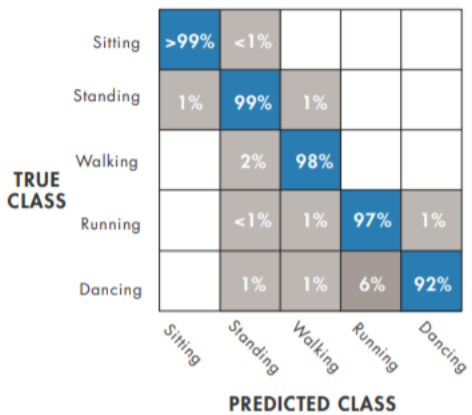

To see how well it performs, we plot the confusion matrix, a table that compares the classifications made by the model with the actual class labels that we created in step 1.




The confusion matrix shows that our model is having trouble distinguishing between dancing and running. Maybe a decision tree doesn't work for this type of data. We'll try a few different algorithms.

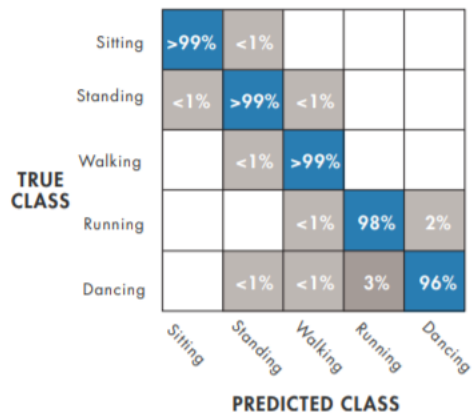
# 4 Step Four: Build and Train the Model *continued*

We start with a K-nearest neighbors (KNN), a simple algorithm that stores all the training data, compares new points to the training data, and returns the most frequent class of the "K" nearest points. That gives us 98% accuracy compared to 94.1% for the simple decision tree. The confusion matrix looks better, too:




However, KNNs take a considerable amount of memory to run, since they require all the training data to make a prediction.

We try a linear discriminant model, but that doesn't improve the results. Finally, we try a multiclass support vector machine (SVM). The SVM does very well—we now get 99% accuracy:




We achieved our goal by iterating on the model and trying different algorithms. If our classifier still couldn't reliably differentiate between dancing and running, we'd look into ways to improve the model.

# 5 Step Five: Improve the Model

Improving a model can take two different directions: make the model simpler or add complexity.

**Simplify**

First, we look for opportunities to reduce the number of features. Popular feature reduction techniques include:

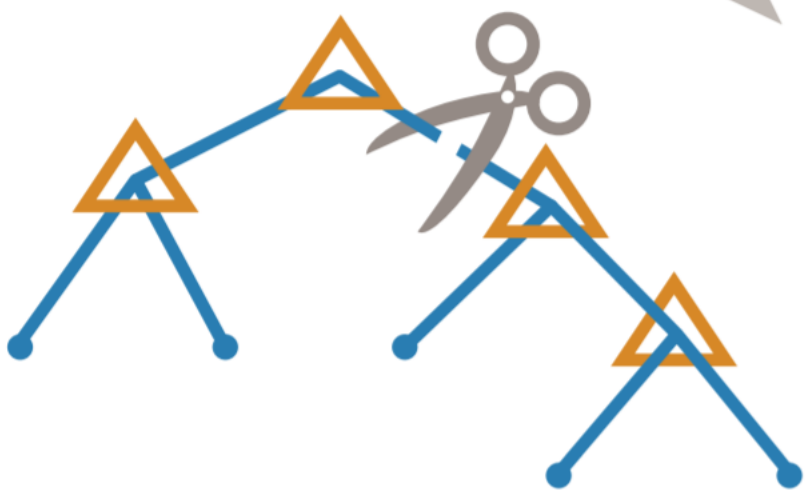
- Correlation matrix – shows the relationship between variables, so that variables (or features) that are not highly correlated can be removed.
- Principal component analysis (PCA) – eliminates redundancy by finding a combination of features that captures key distinctions between the original features and brings out strong patterns in the dataset.
- Sequential feature reduction – reduces features iteratively on the model until there is no improvement in performance.

Next, we look at ways to reduce the model itself. We can do this by:

- Pruning branches from a decision tree
- Removing learners from an ensemble

A good model includes only the features with the most predictive power. A simple model that generalizes well is better than a complex model that may not generalize or train well to new data.

In machine learning, as in many other computational processes, simplifying the model makes it easier to understand, more robust, and more computationally efficient.



# 5 Step Five: Improve the Model *continued*

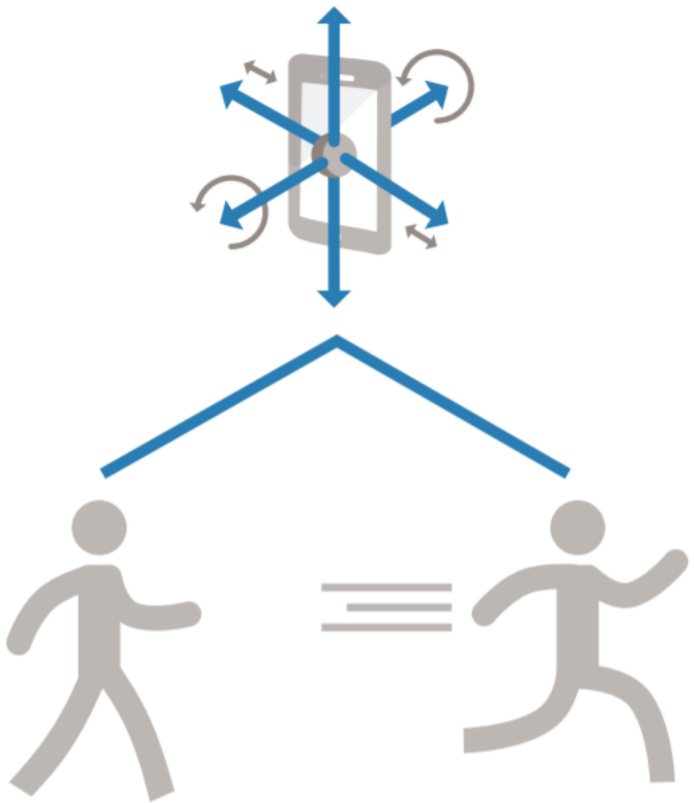
### Add Complexity

If our model can't differentiate dancing from running because it is over-generalizing, then we need find ways to make it more fine-tuned. To do this we can either:

- Use model combination – merge multiple simpler models into a larger model that is better able to represent the trends in the data than any of the simpler models could on their own.
- Add more data sources – look at the gyroscope data as well as the acceleromter data. The gyroscope records the orientation of the cell phone during activity. This data might provide unique signatures for the different activities; for example, there might be a combination of acceleration and rotation that's unique to running.

Once we've adjusted the model, we validate its performance on the test data that we set aside during preprocessing.

If the model can reliably classify activities on the test data, we're ready to move it to the phone and start tracking.



# Predicting House Prices

# How much is my house worth?

# Supervised Learning

- In supervised learning, the dataset is the collection of labelled examples

$$\{(x_i, y_i)\}_{i=1}^{N}$$

*Each element xi among Nis called a **feature vector**.*

A feature vector is a vector in which each **dimension j= 1, . . . , D** contains a value that describes the example some how. That value is called a feature and is denoted as $\mathbf{x}^{(j)}$. For instance, if each example x in our collection represents a person, then the first feature, $\mathbf{x}^{(1)}$, could contain **height** in cm, the second feature, $\mathbf{x}^{(2)}$, could contain weight in kg, $\mathbf{x}^{(3)}$could contain gender, and so on.

For all examples in the dataset, the feature at position j in the feature vector always contains the same kind of information. It means that if $x^{(2)}_i$ contains weight in kg in some example $x_i$, then $x^{(2)}_K$ will also contain weight in kg in every example $x_K$, k= 1, . . . , N. The label $y_i$ can be either an element belonging to a finite set of classes{1,2, . . . , C}, or a real number, or amore complex structure, like a vector, a matrix, a tree, or a Graph.

# Goal Of A Supervised Learning Algorithm

- The goal of a supervised learning algorithm is to use the dataset to produce a model that takes a feature vector x as input and outputs information that allows deducing the label for this feature vector. For instance, the model created using the dataset of people could take as input a feature vector describing a person and output a probability that the person has Cancer.

# What is regression?

| | ENGINESIZE | CYLINDERS | FUELCONSUMPTION_COMB | CO2EMISSIONS |
|---|---|---|---|---|
| 0 | 2.0 | 4 | 8.5 | 196 |
| 1 | 2.4 | 4 | 9.6 | 221 |
| 2 | 1.5 | 4 | 5.9 | 136 |
| 3 | 3.5 | 6 | 11.1 | 255 |
| 4 | 3.5 | 6 | 10.6 | 244 |
| 5 | 3.5 | 6 | 10.0 | 230 |
| 6 | 3.5 | 6 | 10.1 | 232 |
| 7 | 3.7 | 6 | 11.1 | 255 |
| 8 | 3.7 | 6 | 11.6 | 267 |
| 9 | 2.4 | 4 | 9.2 | ? |

# What is regression?

X: Independent variable

Y: Dependent variable

| | ENGINESIZE | CYLINDERS | FUELCONSUMPTION_COMB | CO2EMISSIONS |
|---|---|---|---|---|
| 0 | 2.0 | 4 | 8.5 | 196 |
| 1 | 2.4 | 4 | 9.6 | 221 |
| 2 | 1.5 | 4 | 5.9 | 136 |
| 3 | 3.5 | 6 | 11.1 | 255 |
| 4 | 3.5 | 6 | 10.6 | 244 |
| 5 | 3.5 | 6 | 10.0 | 230 |
| 6 | 3.5 | 6 | 10.1 | 232 |
| 7 | 3.7 | 6 | 11.1 | 255 |
| 8 | 3.7 | 6 | 11.6 | 267 |
| 9 | 2.4 | 4 | 9.2 | ? |

Regression is the process of predicting a continuous value

# Types of regression models

- Simple Regression:
  - Simple Linear Regression
  - Simple Non-linear Regression
- Multiple Regression:
  - Multiple Linear Regression
  - Multiple Non-linear Regression

Predict co2emission vs EngineSize of all cars

Predict co2emission vs EngineSize and Cylinders of all cars

# Regression algorithms

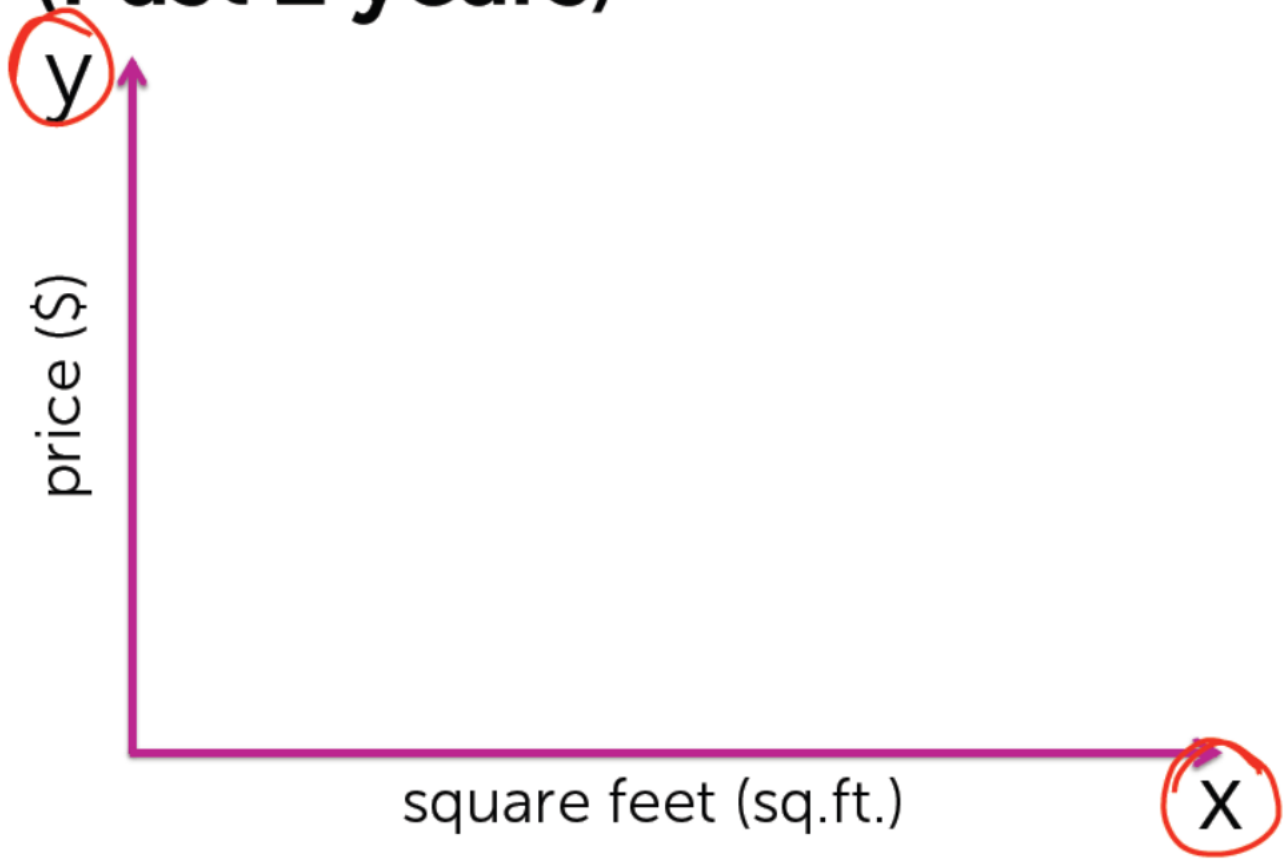
- Ordinal regression
- Poisson regression
- Fast forest quantile regression
- Linear, Polynomial, Lasso, Stepwise, Ridge regression
- Bayesian linear regression
- Neural network regression
- Decision forest regression
- Boosted decision tree regression
- KNN (K-nearest neighbors)

# How much is my house worth?

# Look at recent sales in my neighborhood

- How much did they sell for?

# Plot recent house sales (**Past 2 years**)

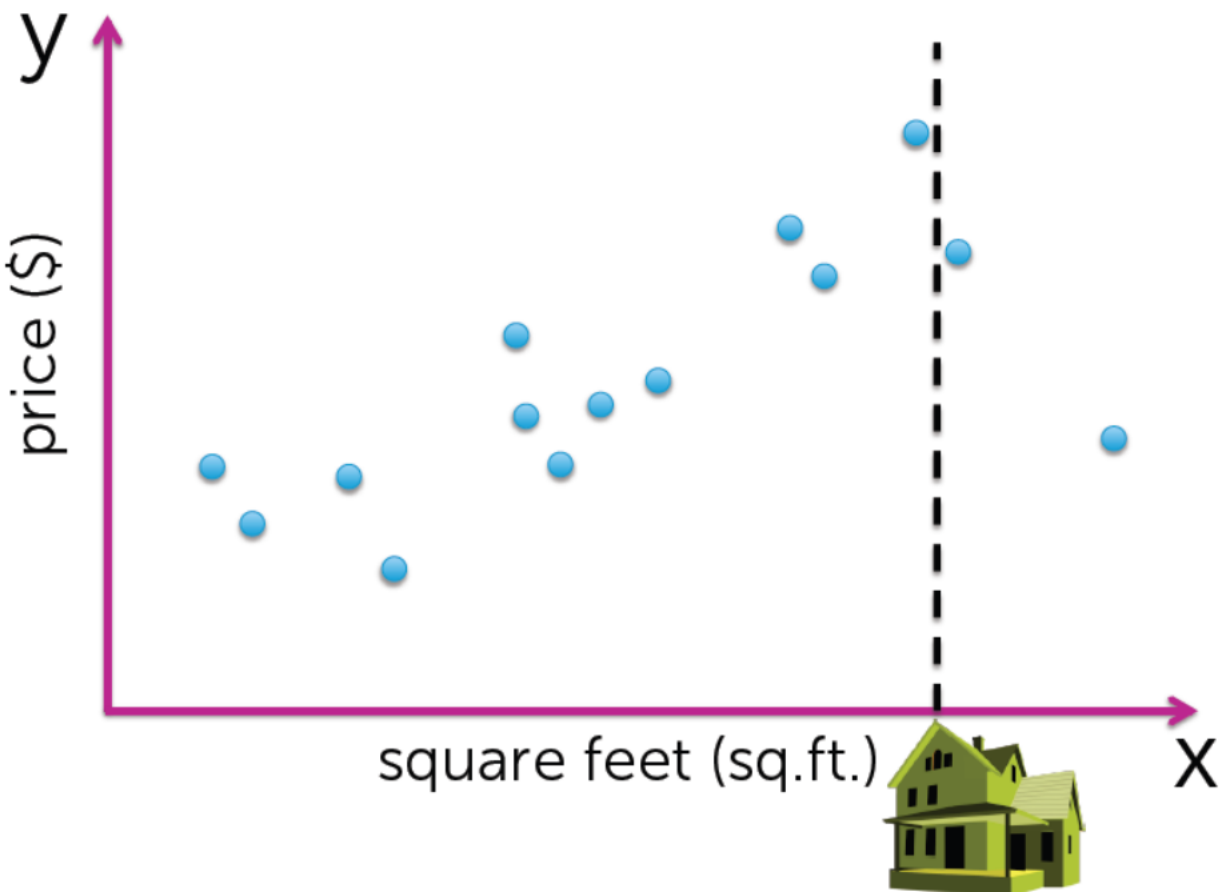



**Terminology:**

x – feature, covariate, or predictor

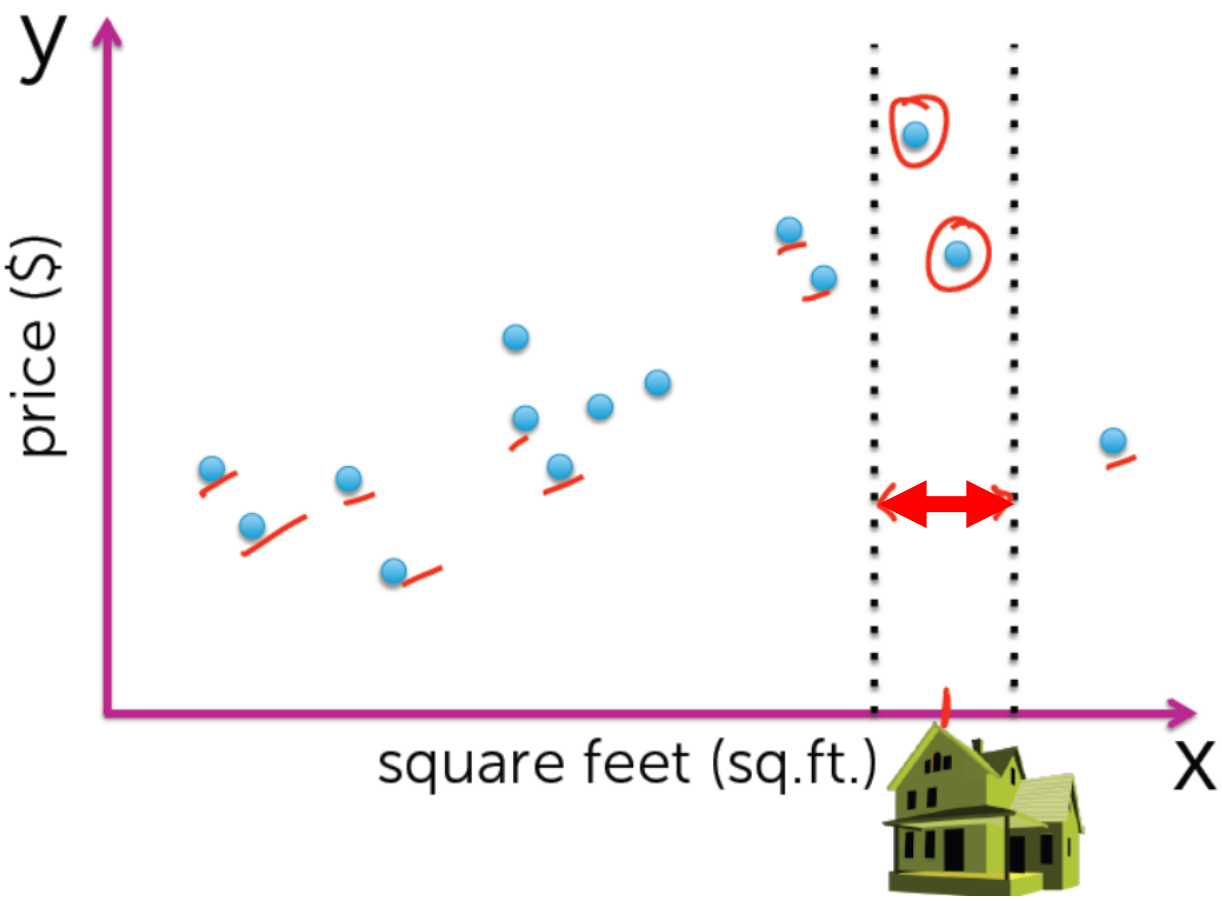
y – observation or response

# Predict your house by similar houses




No house sold recently had *exactly* the same sq.ft.

# Predict your house by similar houses




- Look at average price in range
- **Still only 2 houses!**
- Throwing out info from all other sales

# What is regression?

From features to predictions

# Regression

- Regression is a supervise learning problem, where your given examples of instances whose X and Y value are given, and you have to learn a function, so that given an unknown X, you have to predict Y.
- A function which predicts given X predicts Y; and for regression, Y is continuous
- simple regression where X is a single feature
- multiple regressions, where X comprises a number of features
- So, if x is a single feature then the training examples can be plotted

# Regression

- Linear regression is a popular regression learning algorithm that learns a model which is a linear combination of features of the input example.
- We have a collection of labeled examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$ , where N is the size of the collection, xi is the D-dimensional feature vector of example i = 1, . . . , N, yi is a real-valued1 target and every feature x (j) i , j = 1, . . . , D, is also a real number. We want to build a model $f_{\mathbf{w},b}(\mathbf{x})$ as a linear combination of features of example x:

$$f_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}\mathbf{x} + b,$$

# Regression Problem

- Training data: sample drawn i.i.d. from set $X$ according to some distribution $D$,

$$S = ((x_1, y_1), \ldots, (x_m, y_m)) \in X \times Y,$$

with $Y \subseteq \mathbb{R}$ is a measurable subset.

- Loss function: $L : Y \times Y \to \mathbb{R}_+$ a measure of closeness, typically $L(y, y') = (y' - y)^2$ or $L(y, y') = |y' - y|^p$ for some $p \geq 1$.

- Problem: find hypothesis $h : X \to \mathbb{R}$ in $H$ with small generalization error with respect to target $f$

$$R_D(h) = \mathop{\mathrm{E}}_{x \sim D} \left[ L\big(h(x), f(x)\big) \right].$$

# What is regression?

From features to predictions

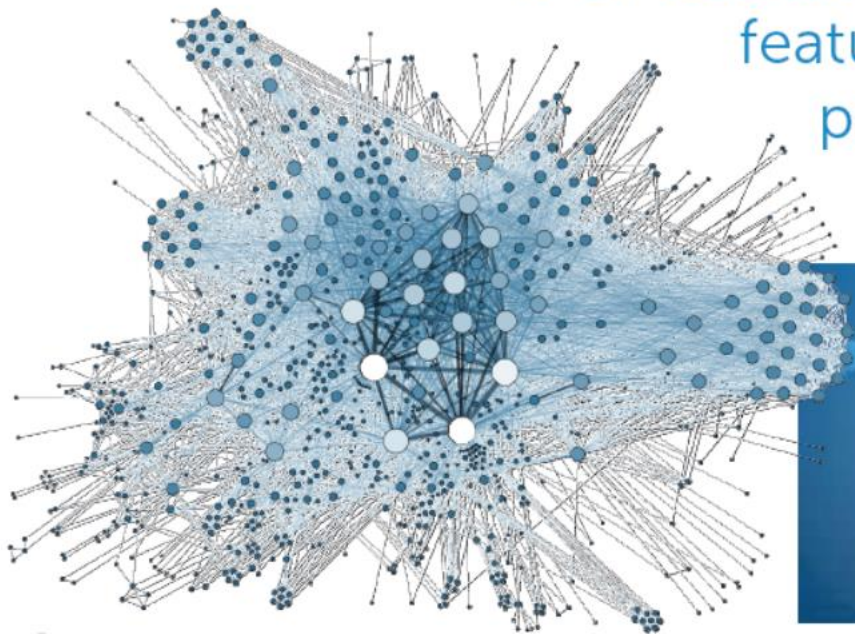

**Input x:**
features derived from data

**Learn x→y relationship**

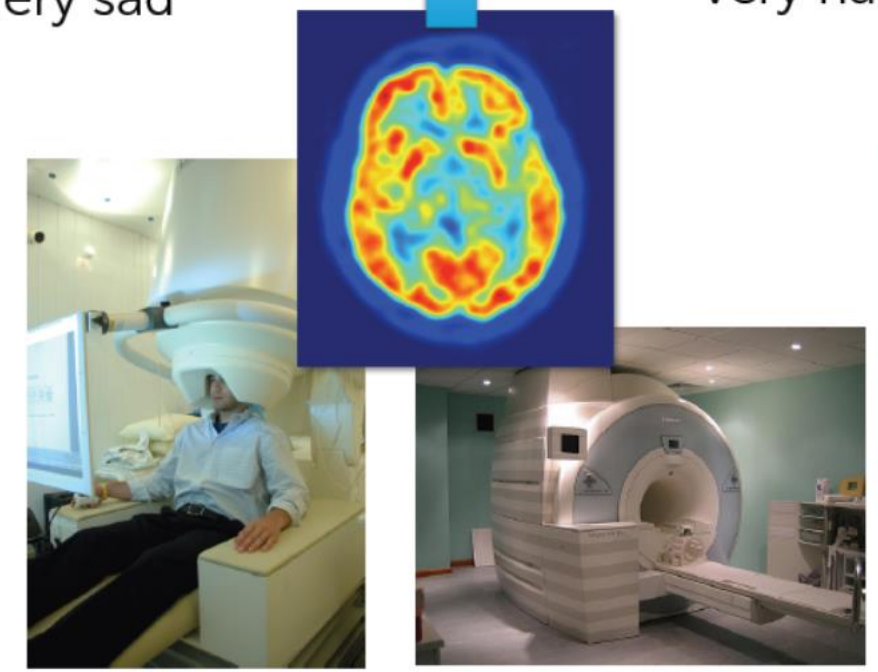
**Predict y:**
continuous "output" or "response" to input

# Tweet popularity

- How many people will retweet your tweet? (y)
- Depends on **x** = # followers,
  # of followers of followers,
  features of text tweeted,
  popularity of hashtag,
  # of past retweets,...

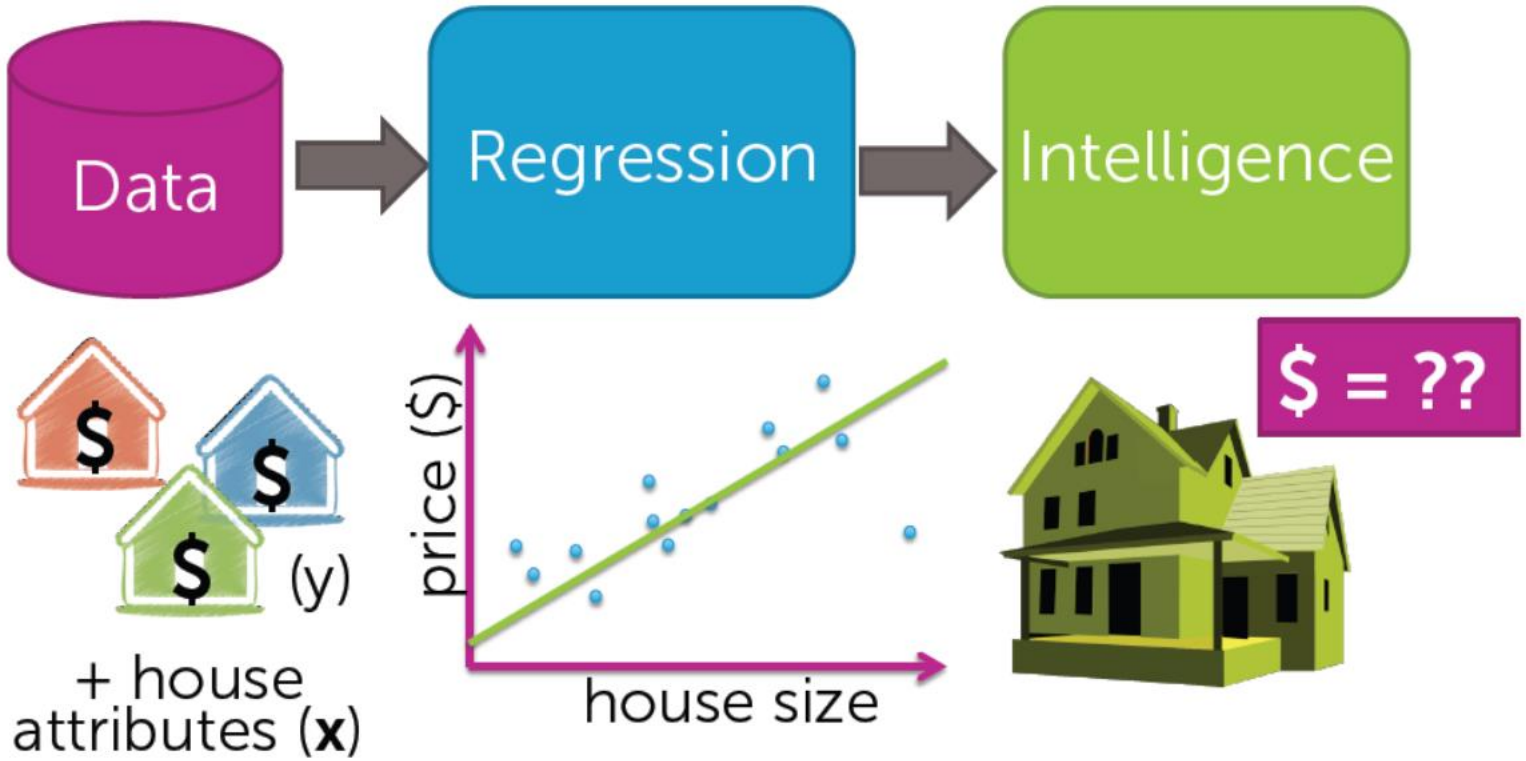


# Reading your mind

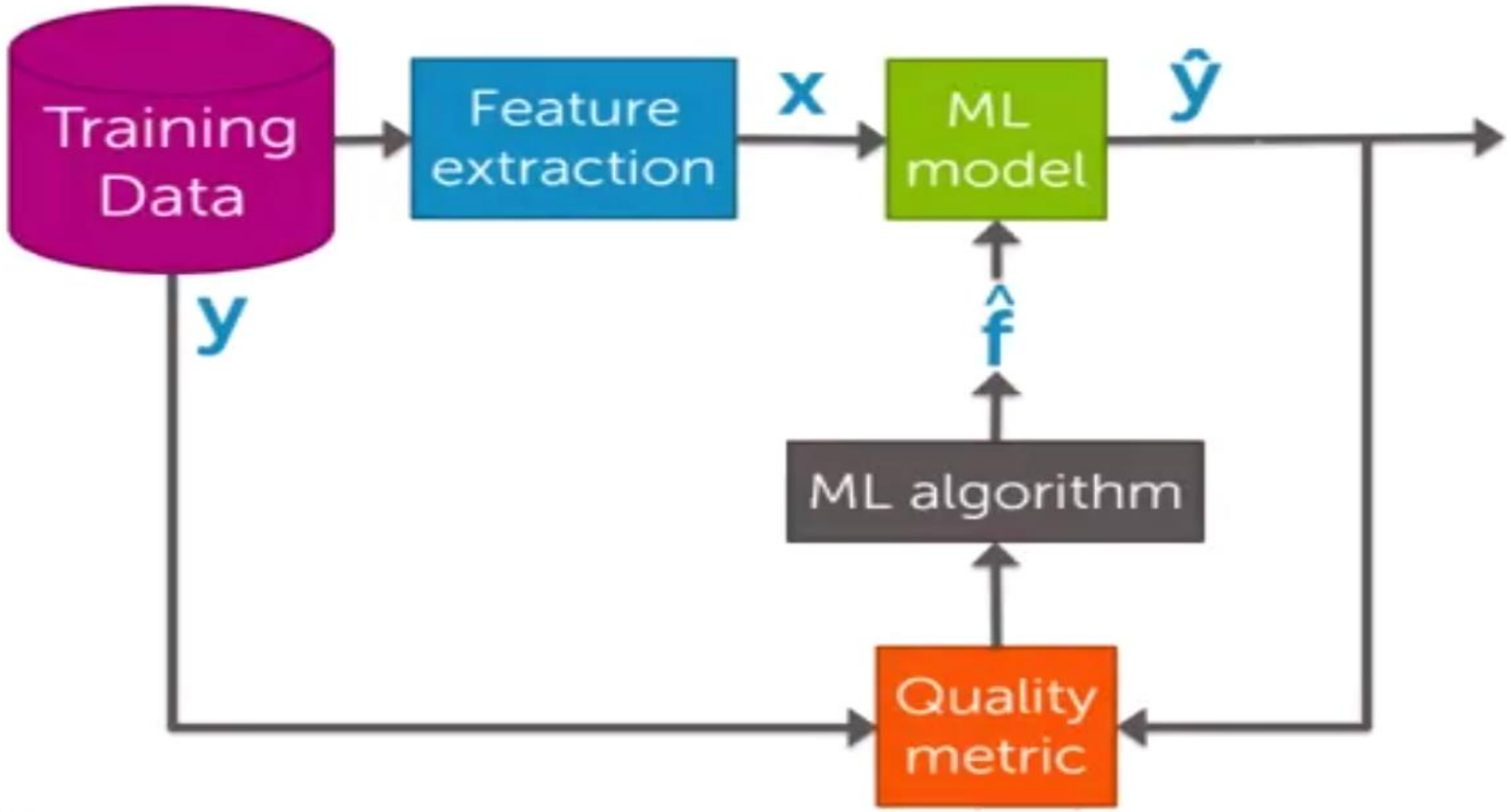



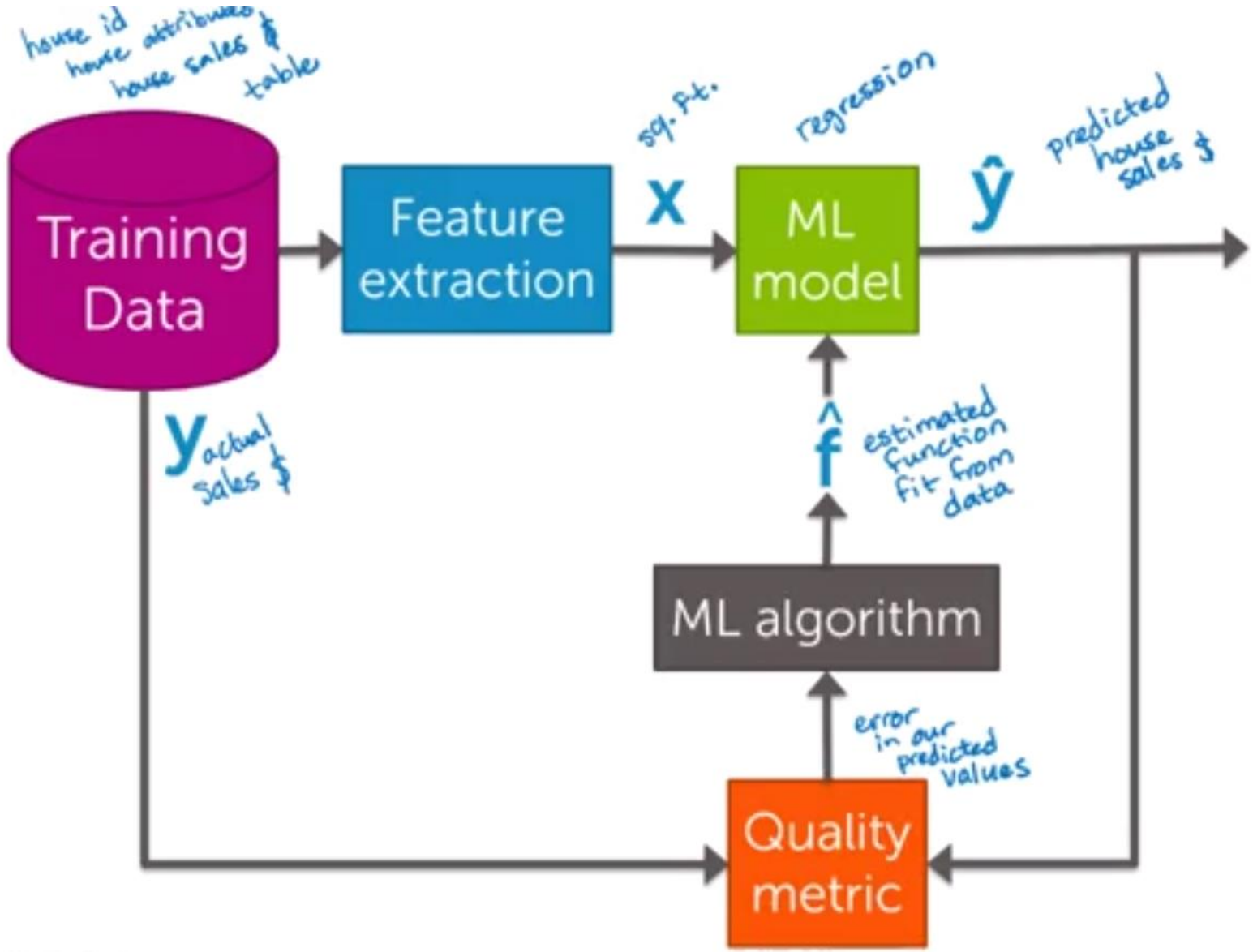
Inputs **x** are brain region intensities

# Case Study:
# Predicting house prices



# Workflow of Machine Learning

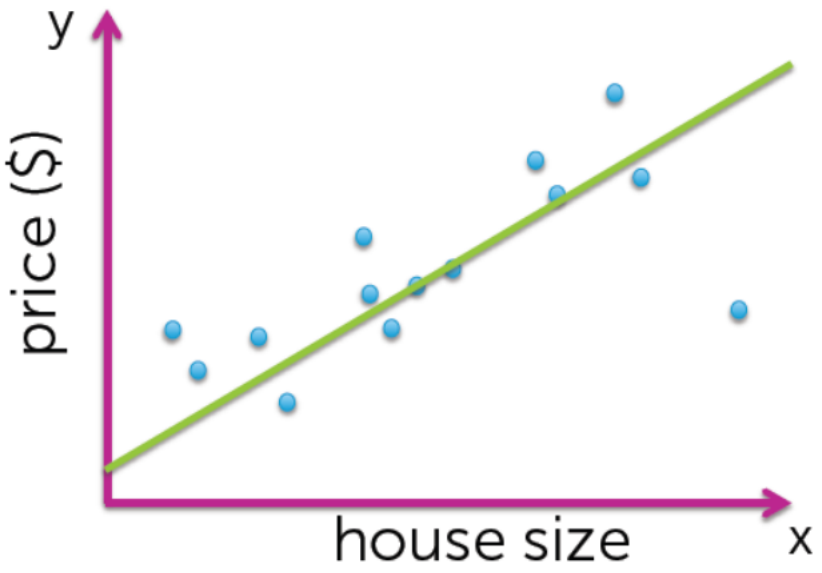



house id
house attributes
house sales $
table
Training Data
Feature extraction
sq. ft.
x
regression
ML model
ŷ
predicted house sales $
y actual sales $
f̂
estimated function fit from data
ML algorithm
error in our predicted values
Quality metric

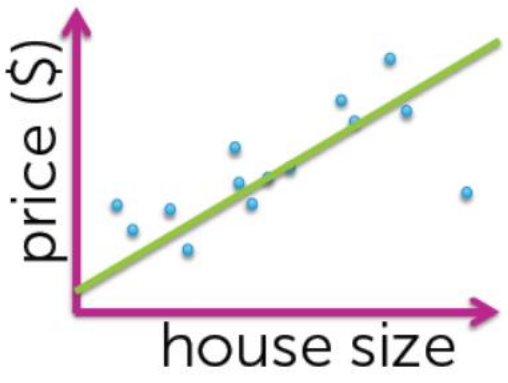
# Simple Regression

**What makes it simple?**

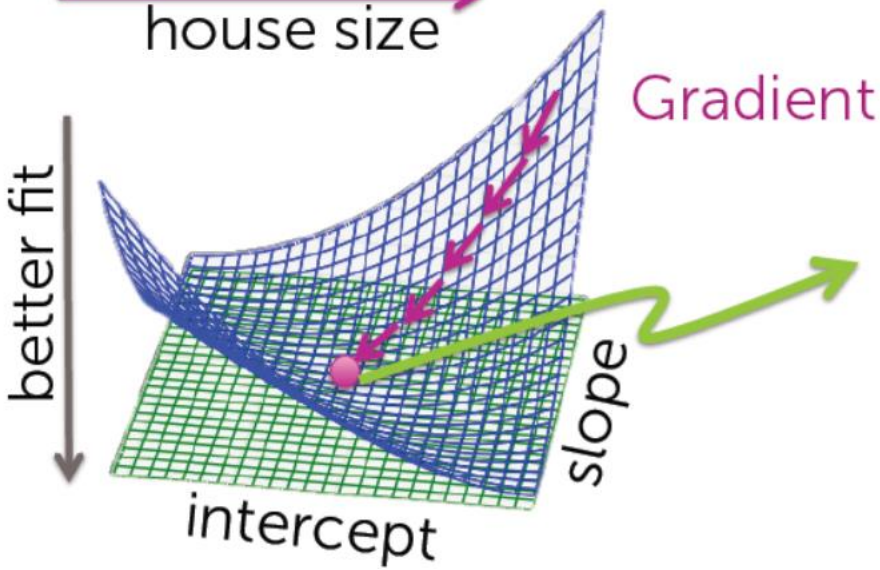
1 input and just fit a line to data



# Simple Regression



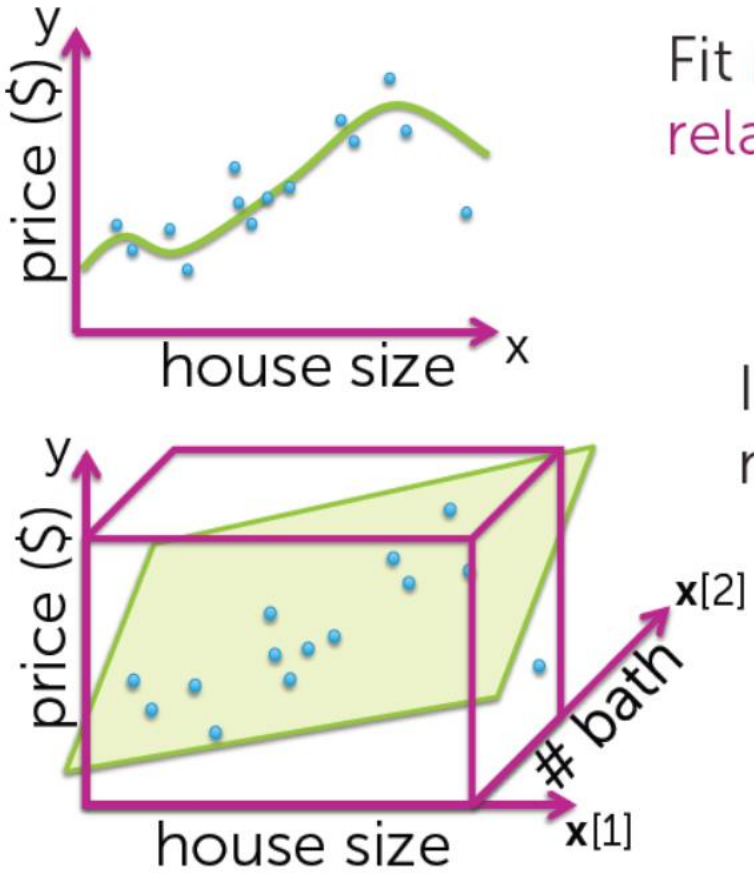

Define goodness-of-fit metric for each possible line




Gradient descent algorithm

**Get estimated parameters**
- interpret
- use to form predictions

# Multiple Regression

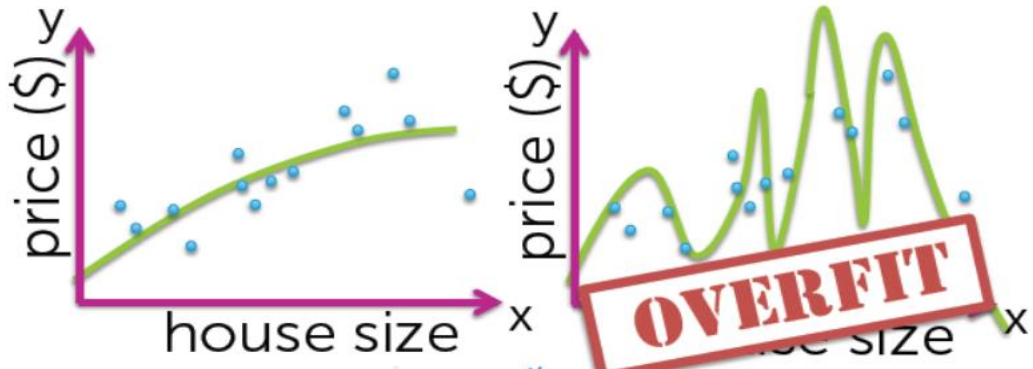


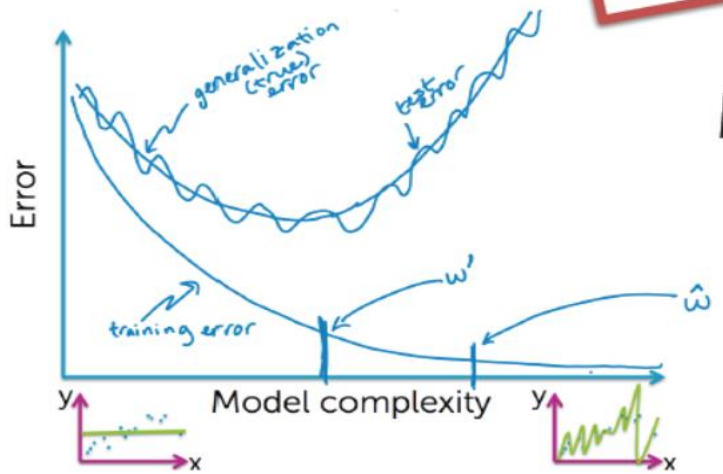

Fit more complex relationships than just a line

Incorporate more inputs

- Square feet
- # bathrooms
- # bedrooms
- Lot size
- Year built
- ...

# Assessing Performance

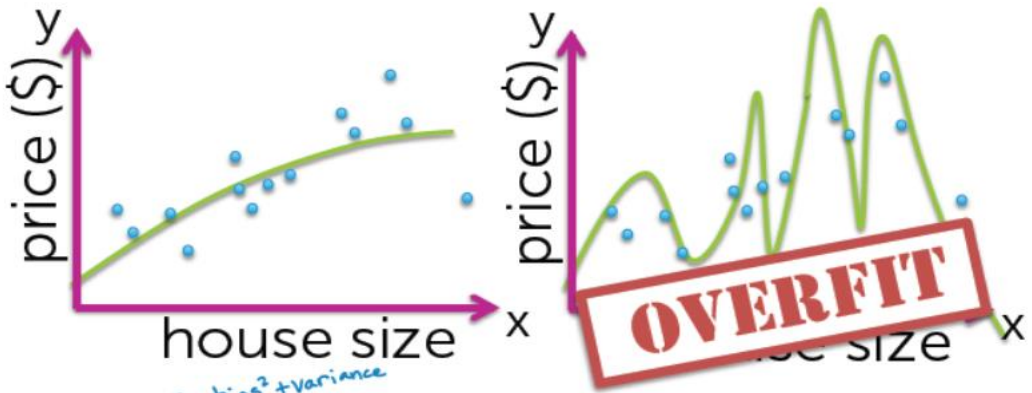


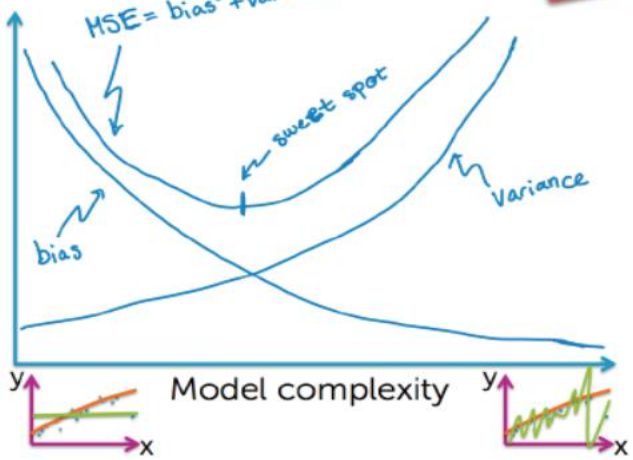




Measures of error:

- Training
- Test
- True (generalization)

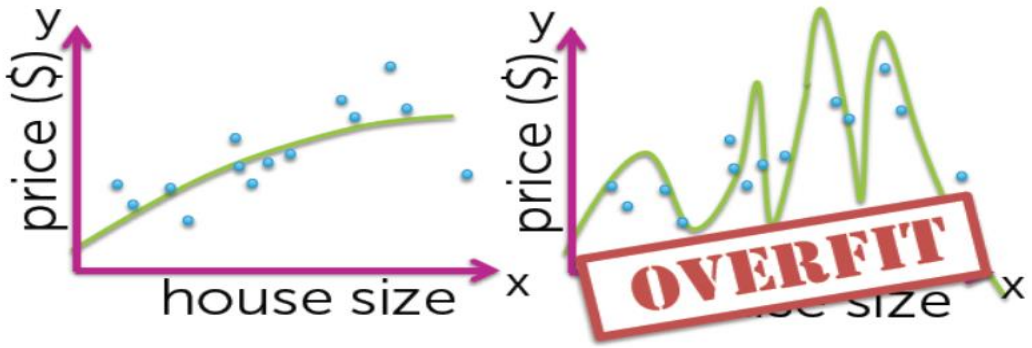
# Assessing Performance




Bias-variance tradeoff

# Ridge Regression



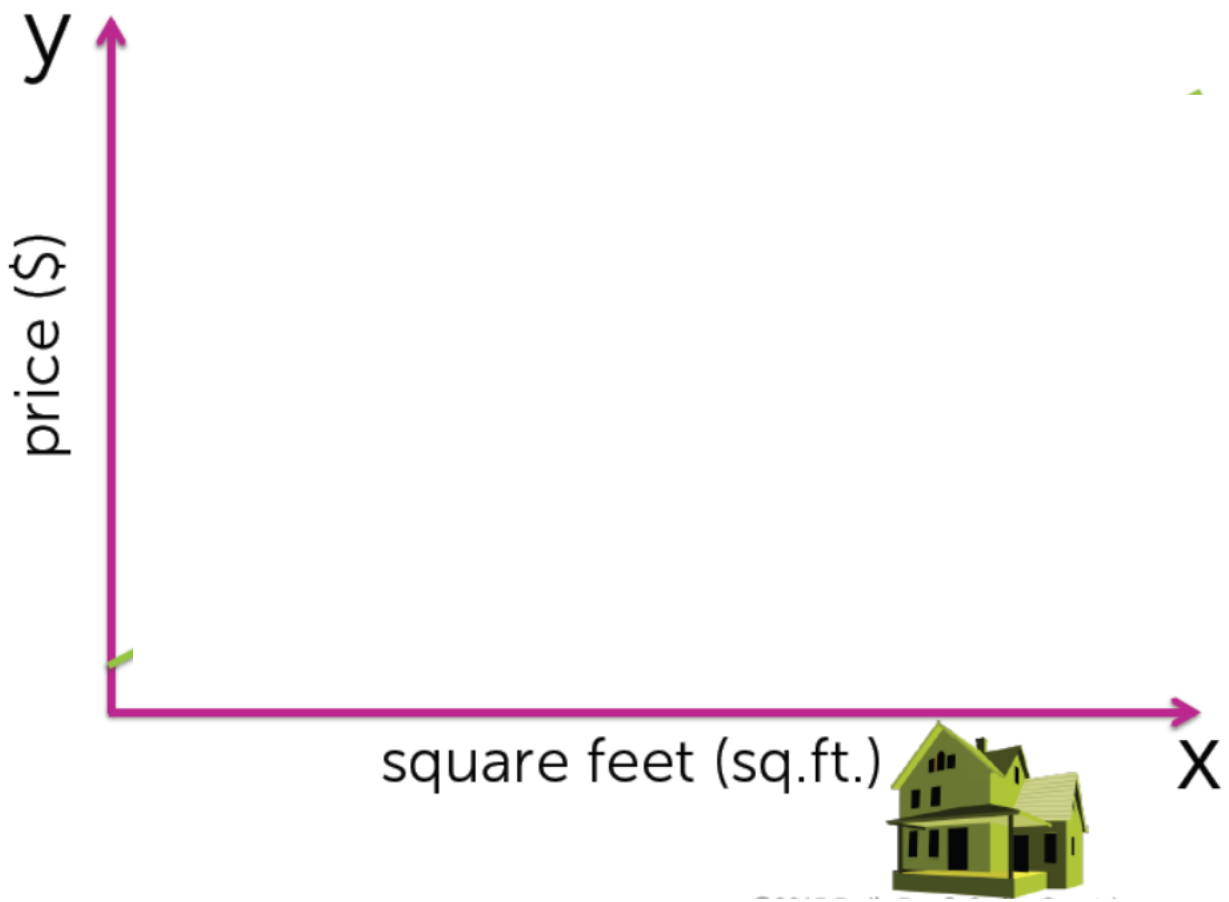

**Ridge total cost** =

measure of fit + measure of model complexity

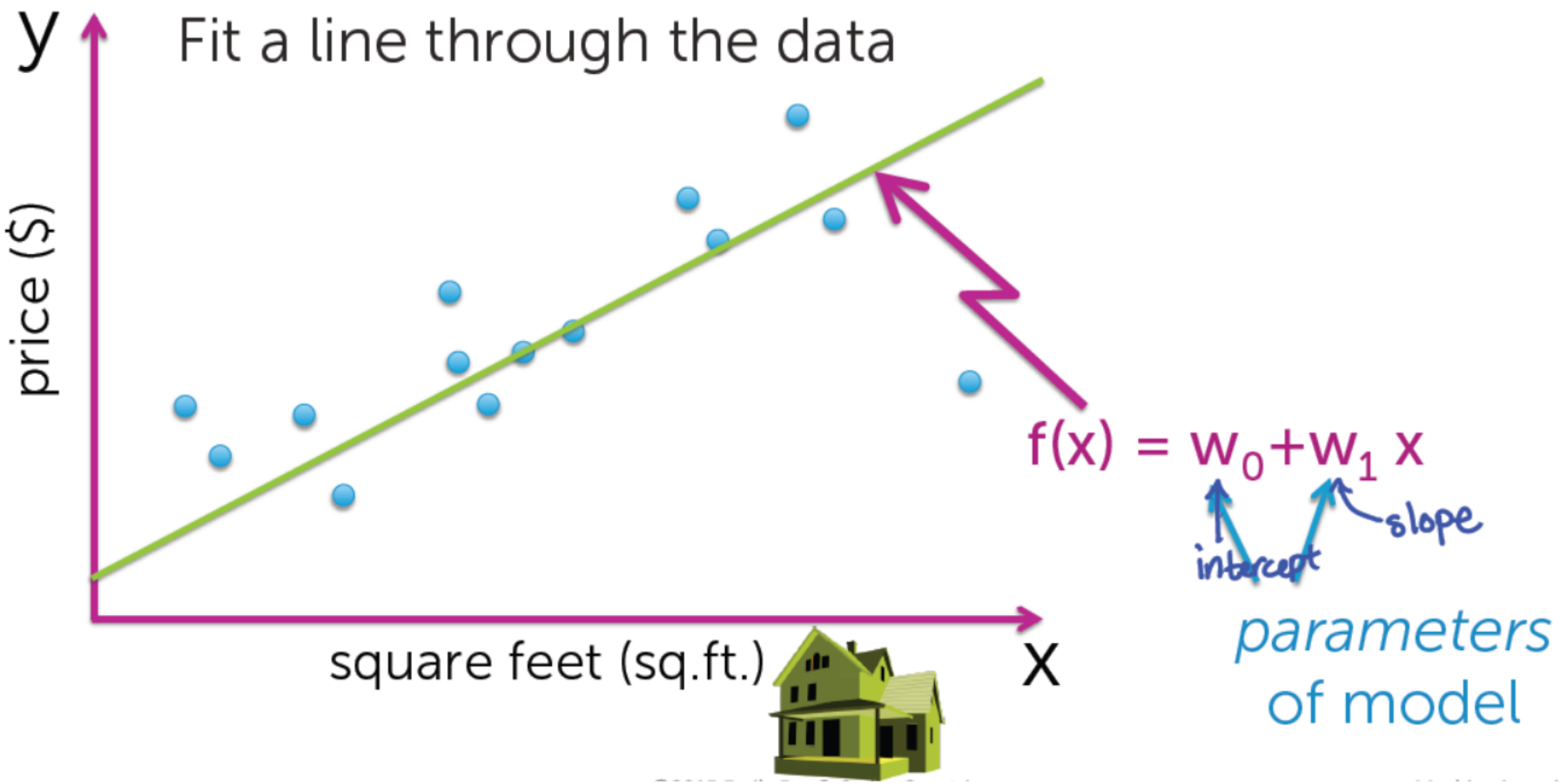
bias-variance tradeoff

# Linear regression

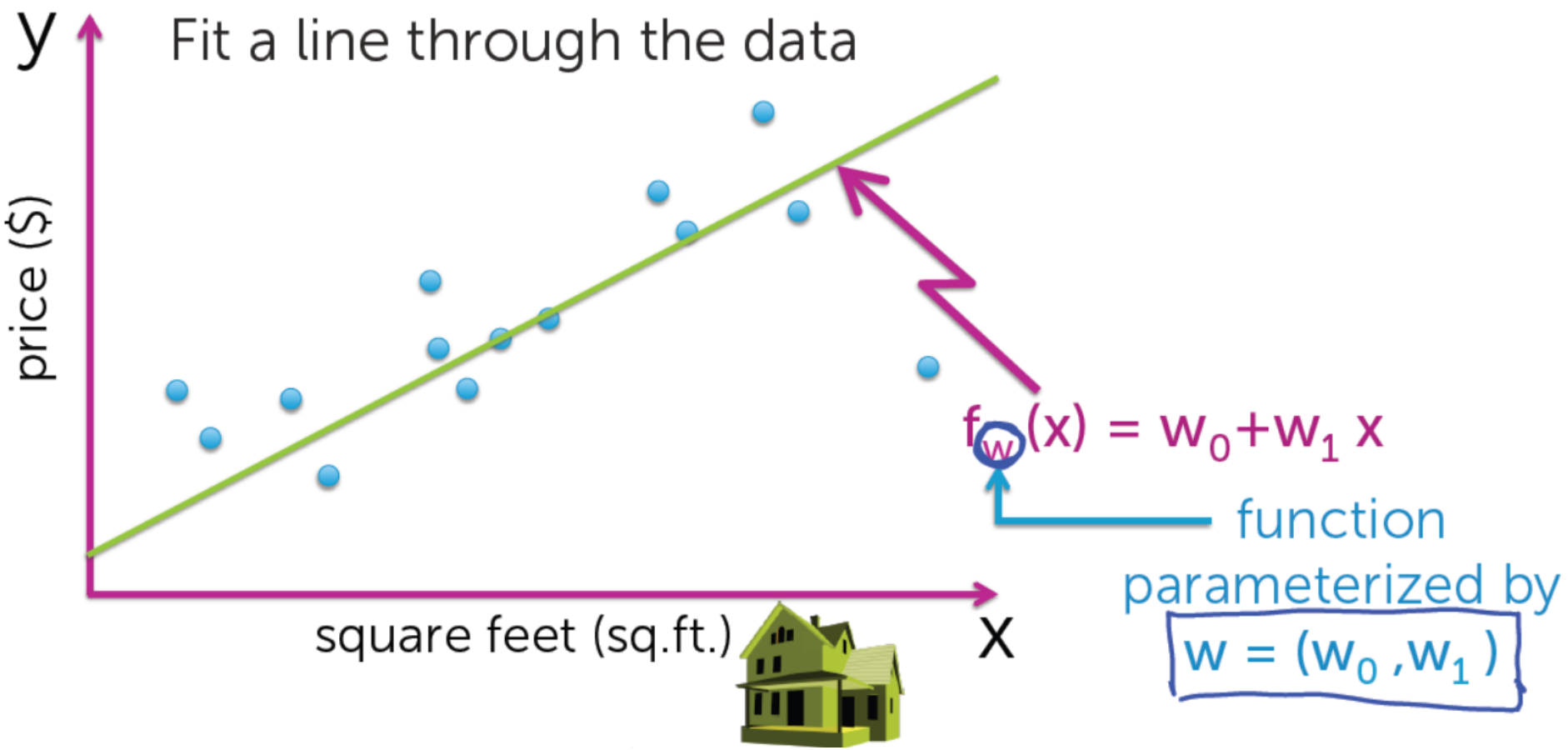
# Use a **linear** regression model



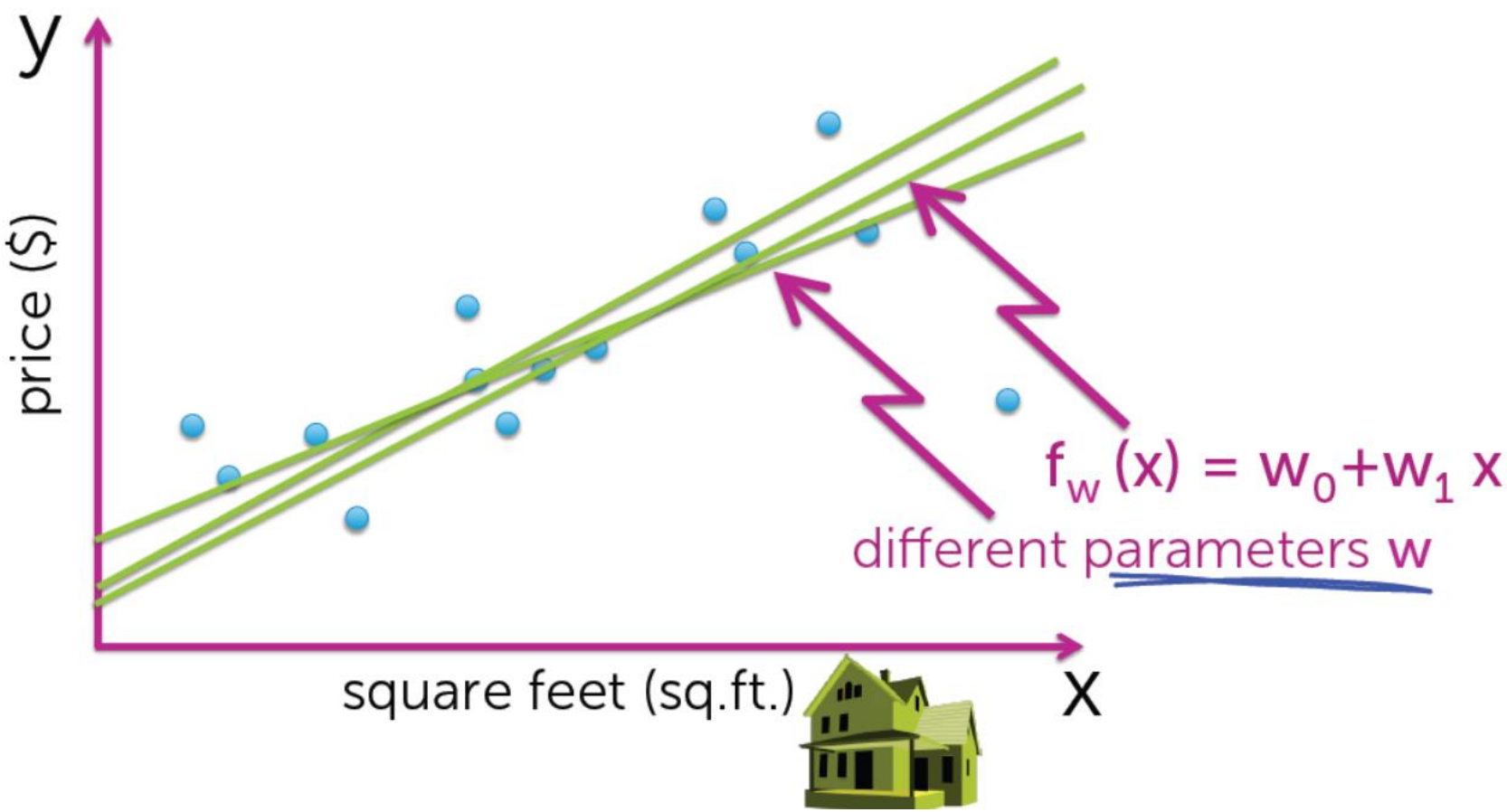
# Use a **linear** regression model



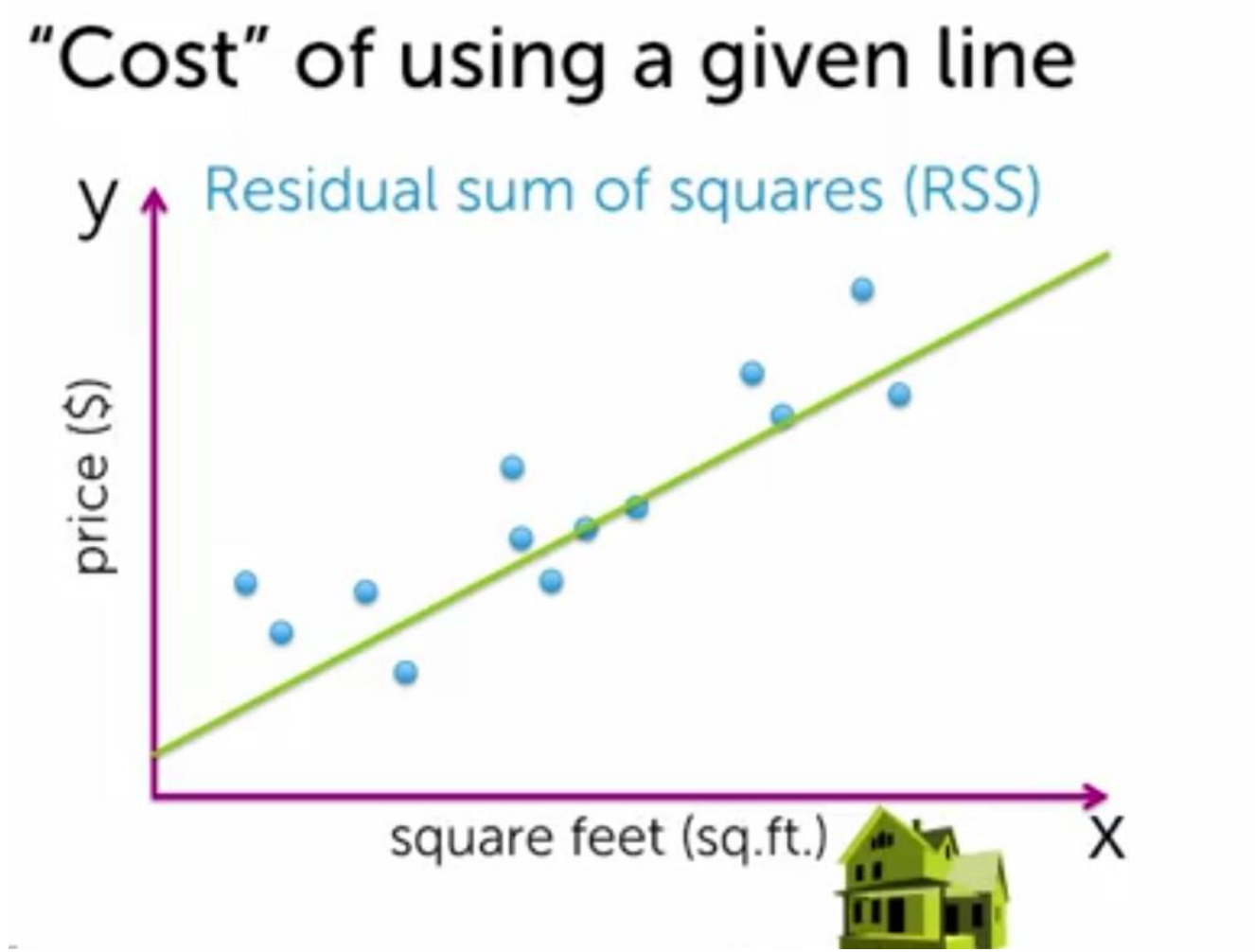
# Use a **linear** regression model



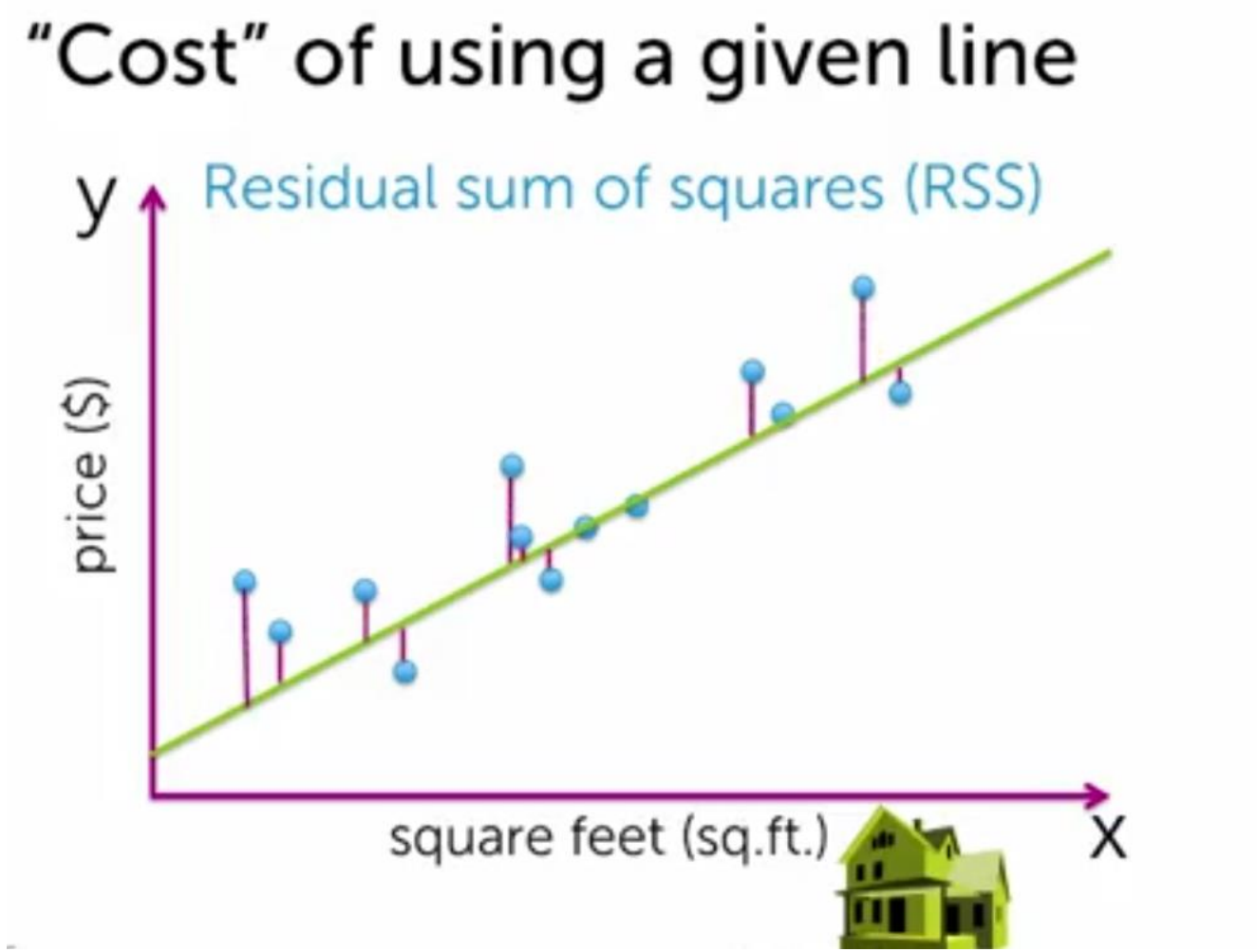
# Which line?



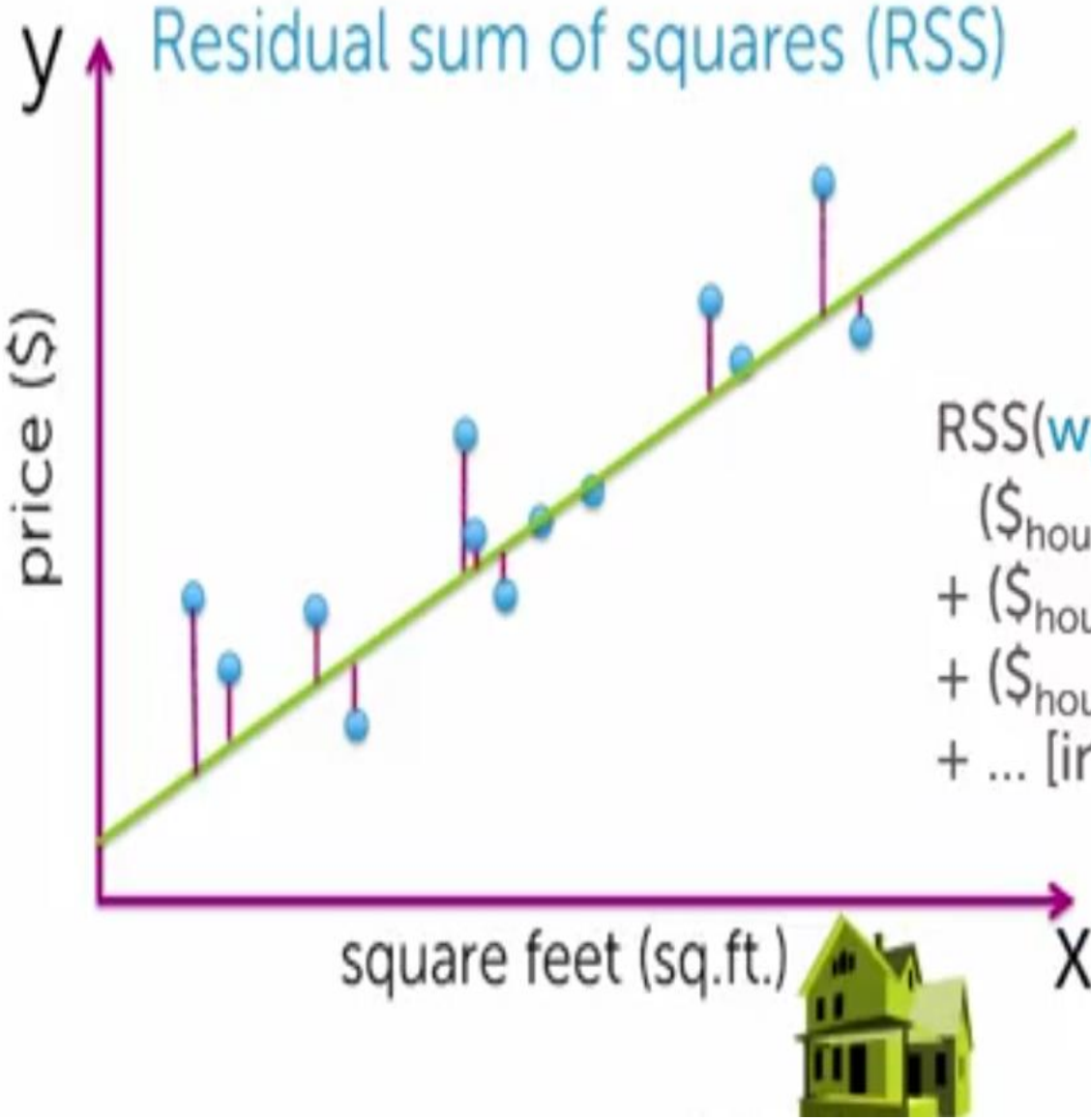
# "Cost" of using a given line



# "Cost" of using a given line

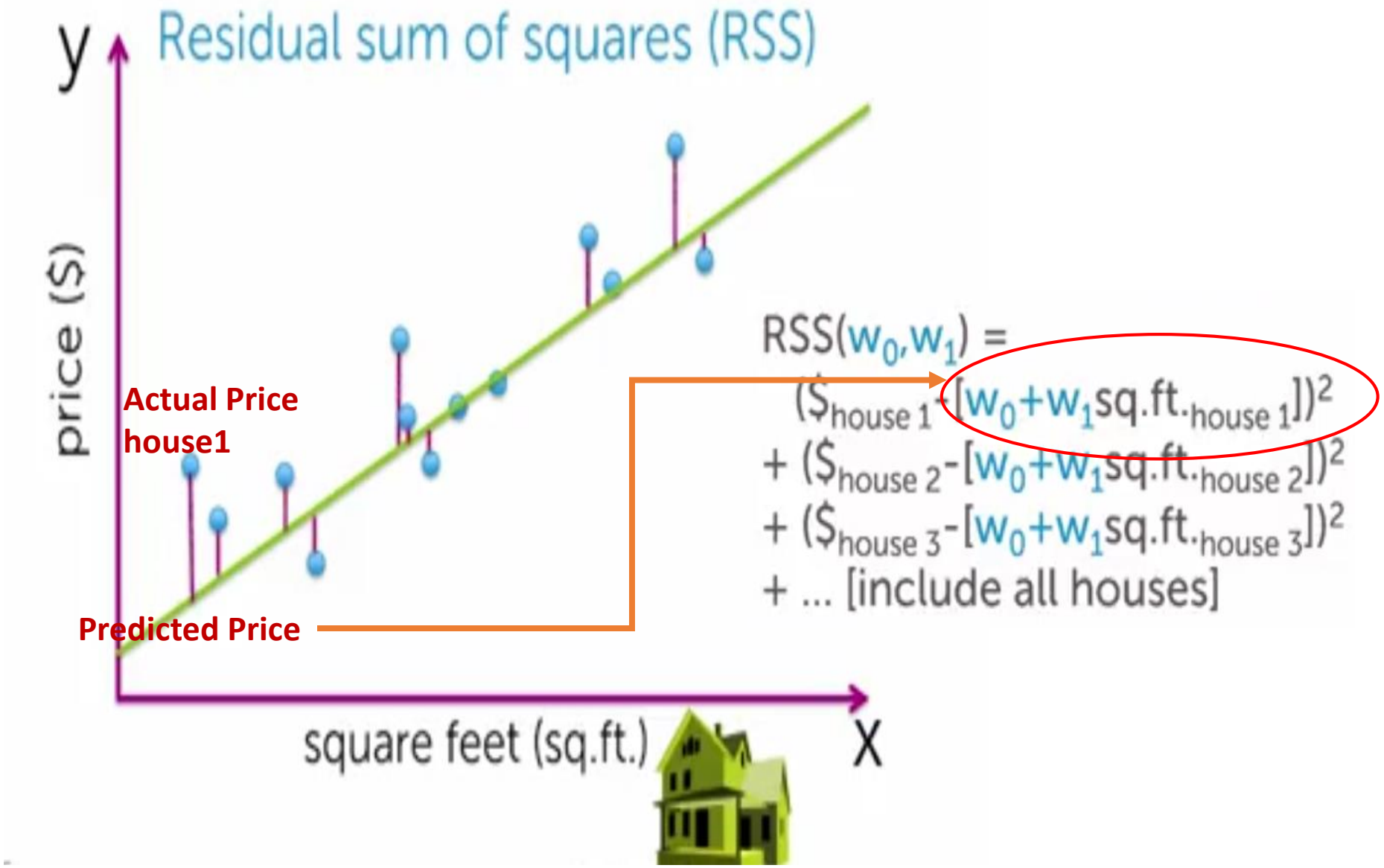


# "Cost" of using a given line

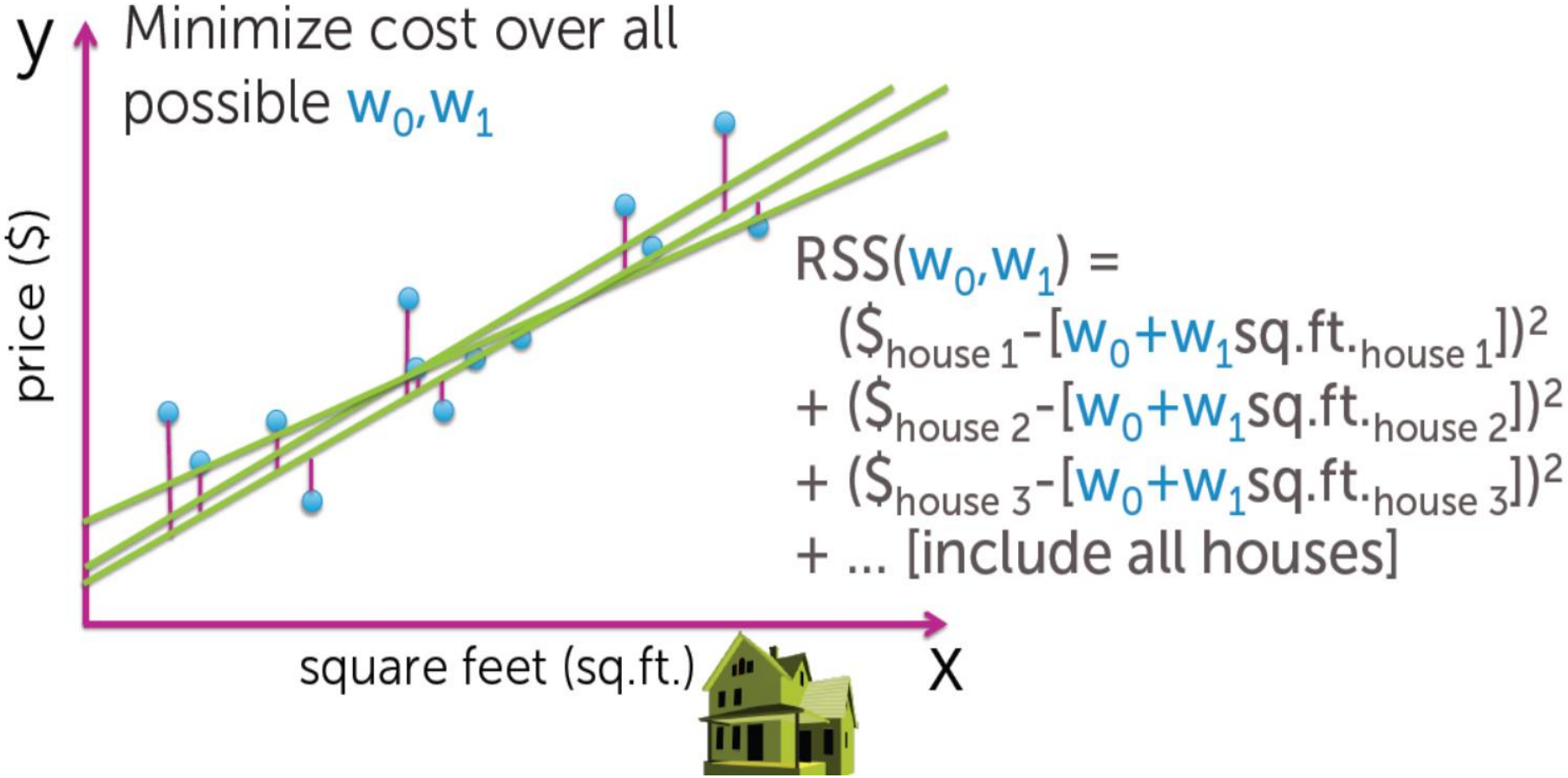



$$RSS(w_0, w_1) = (\$_{house\ 1} - [w_0 + w_1 sq.ft._{house\ 1}])^2 + (\$_{house\ 2} - [w_0 + w_1 sq.ft._{house\ 2}])^2 + (\$_{house\ 3} - [w_0 + w_1 sq.ft._{house\ 3}])^2 + \ldots \text{ [include all houses]}$$

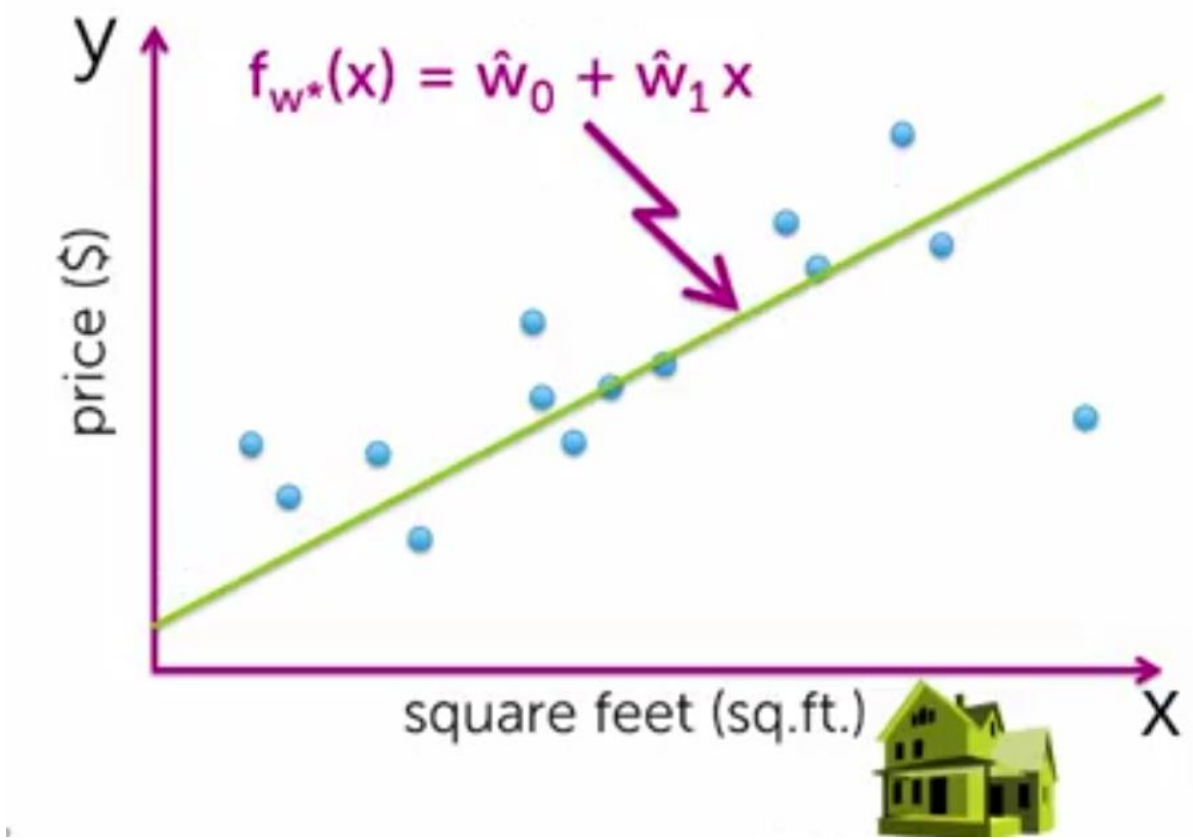
# "Cost" of using a given line



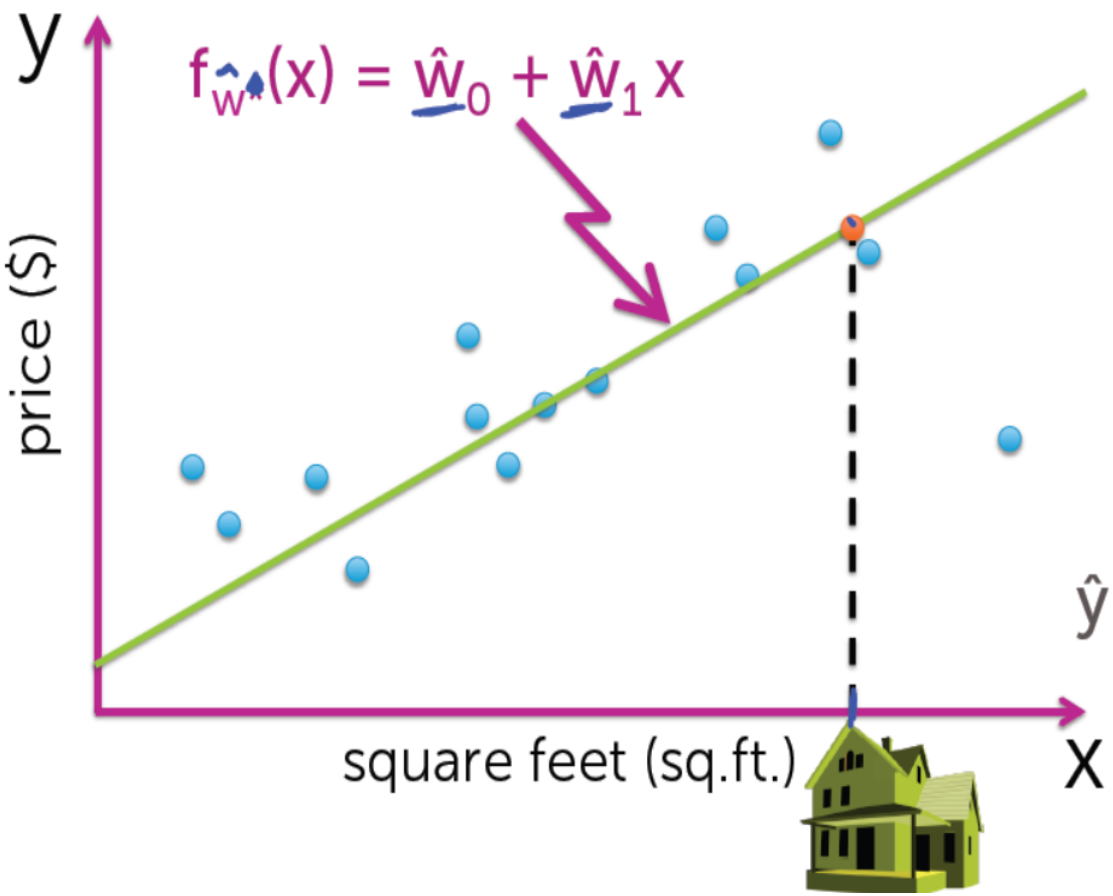
# Find "best" line



# Predicting your house price



# Predicting your house price

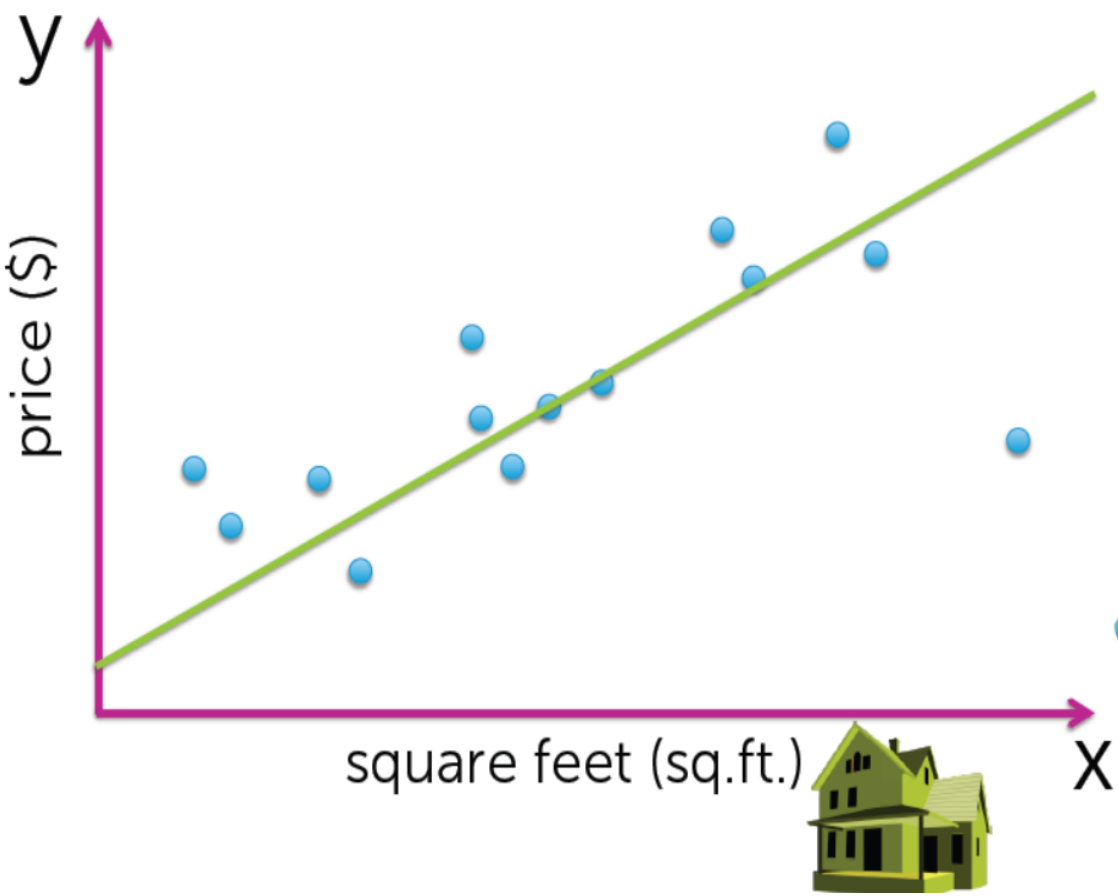



Best guess of your house price:

$\hat{y} = \hat{w}_0 + \hat{w}_1 \text{ sq.ft.}_{\text{your house}}$

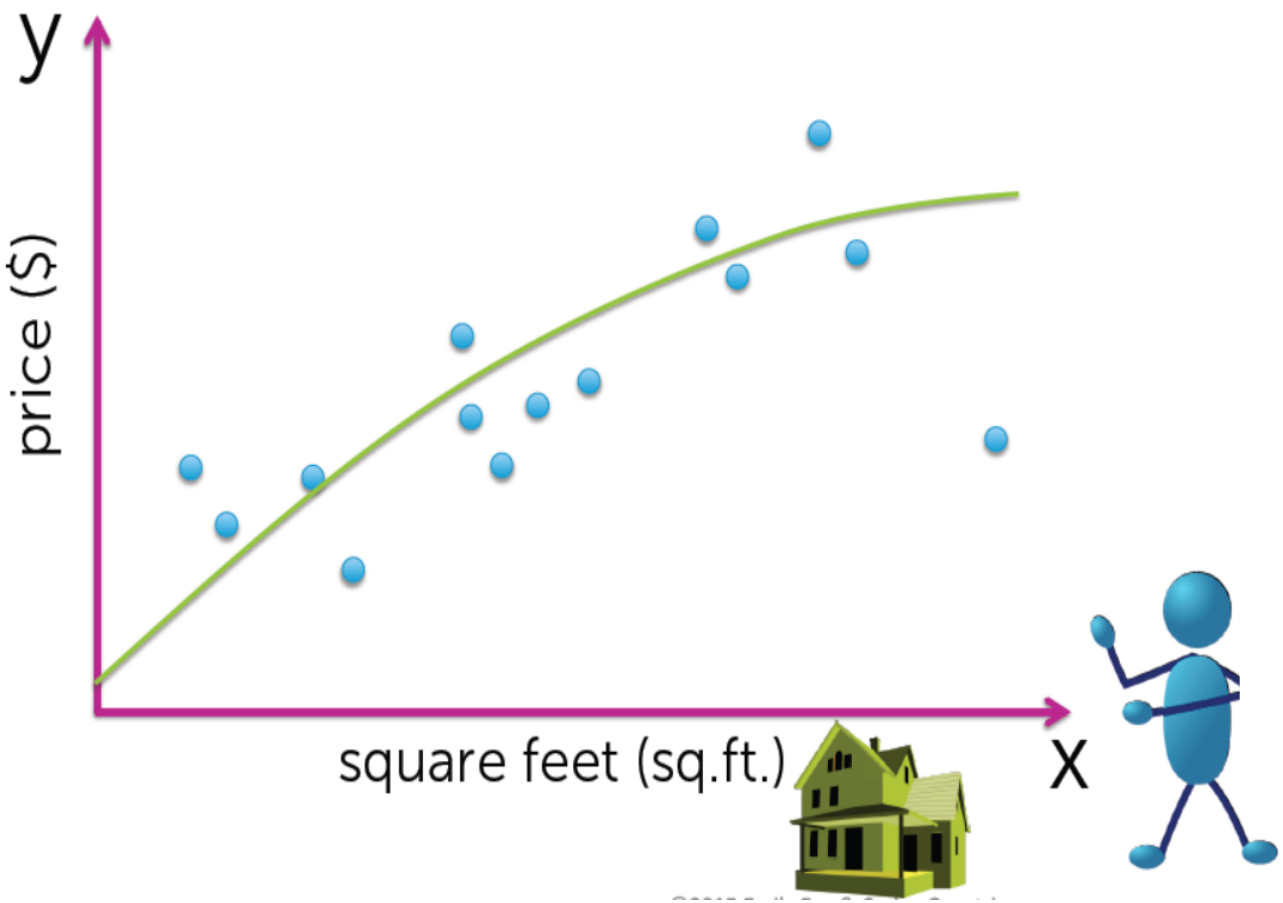
# ADDING HIGHER ORDER EFFECTS

# Fit data with a line or … ?

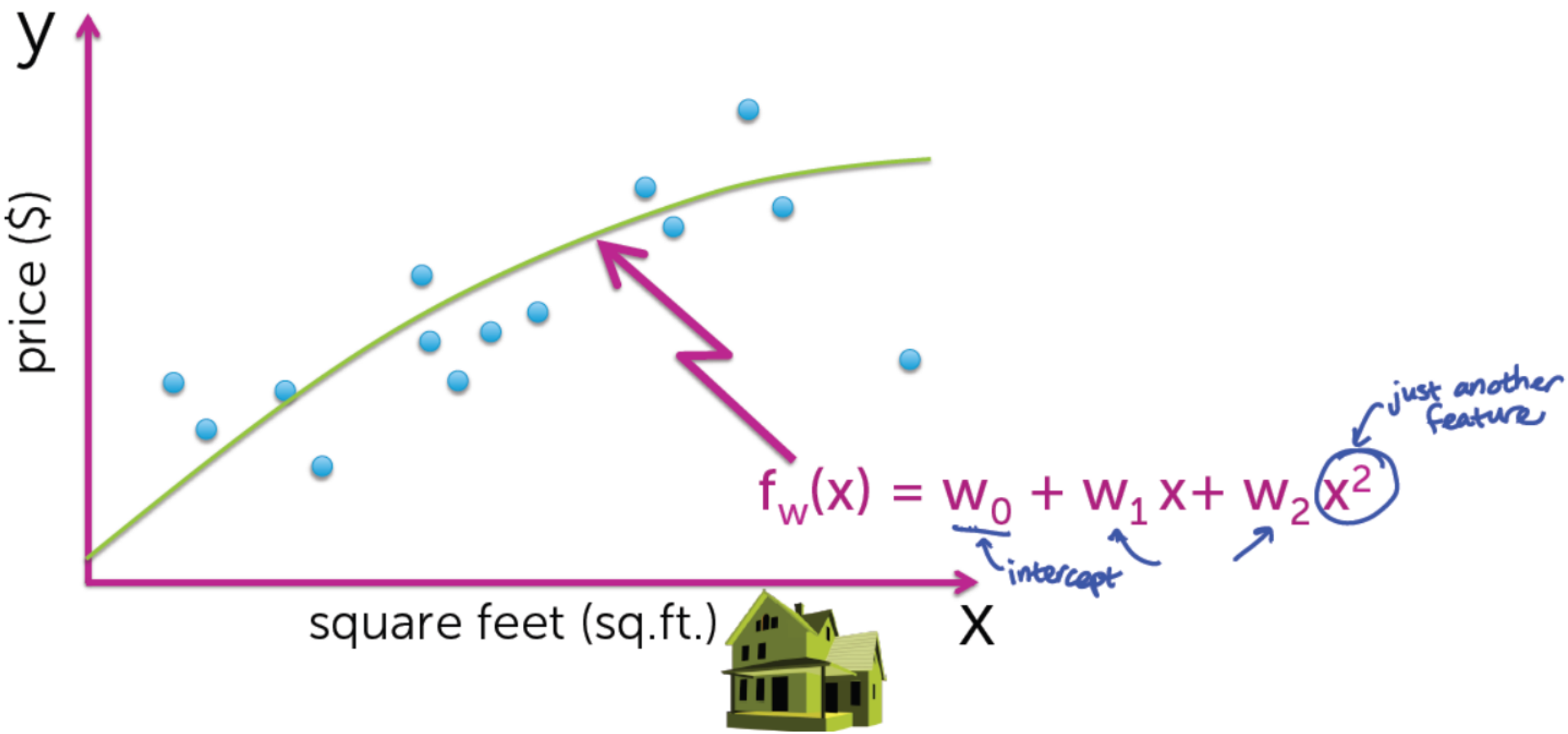



You show your friend your analysis

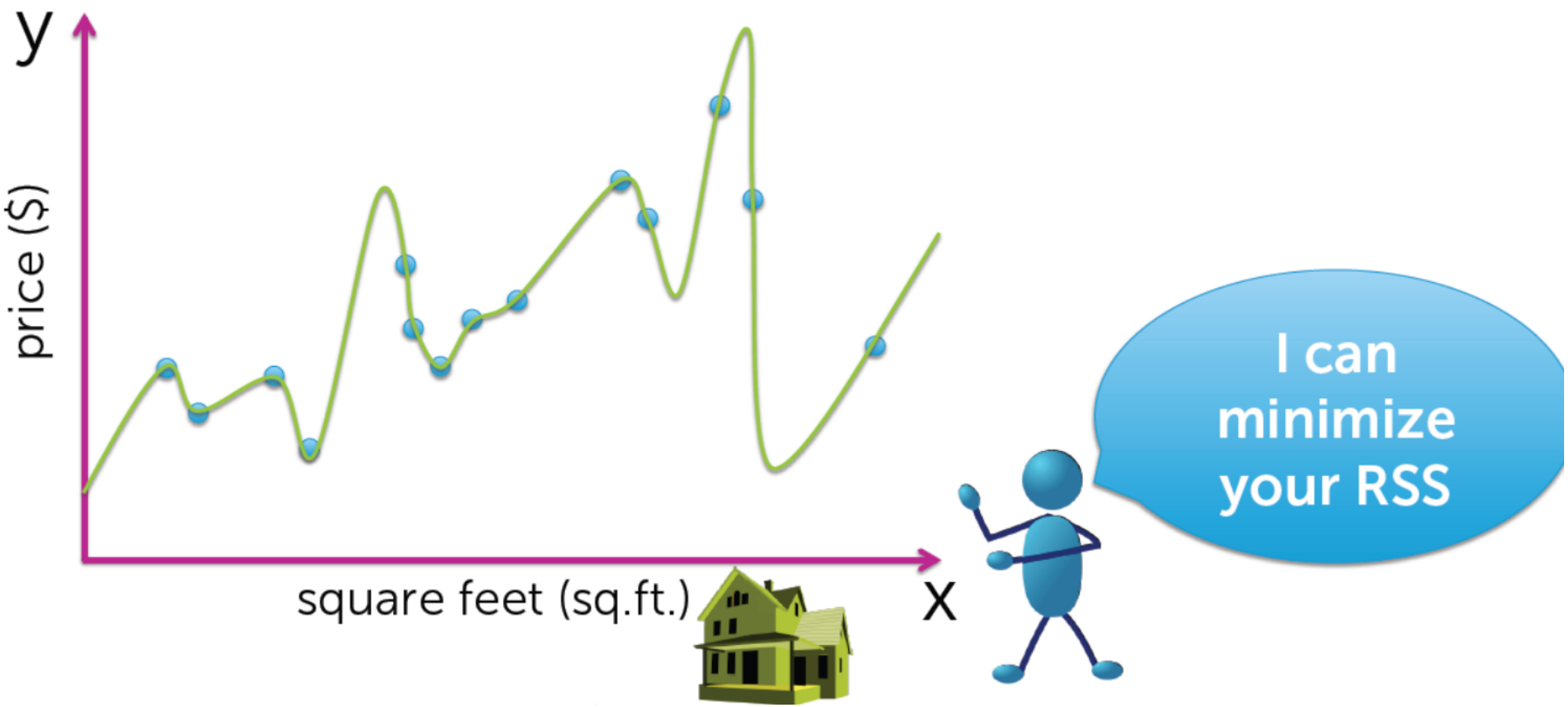
# What about a quadratic function?



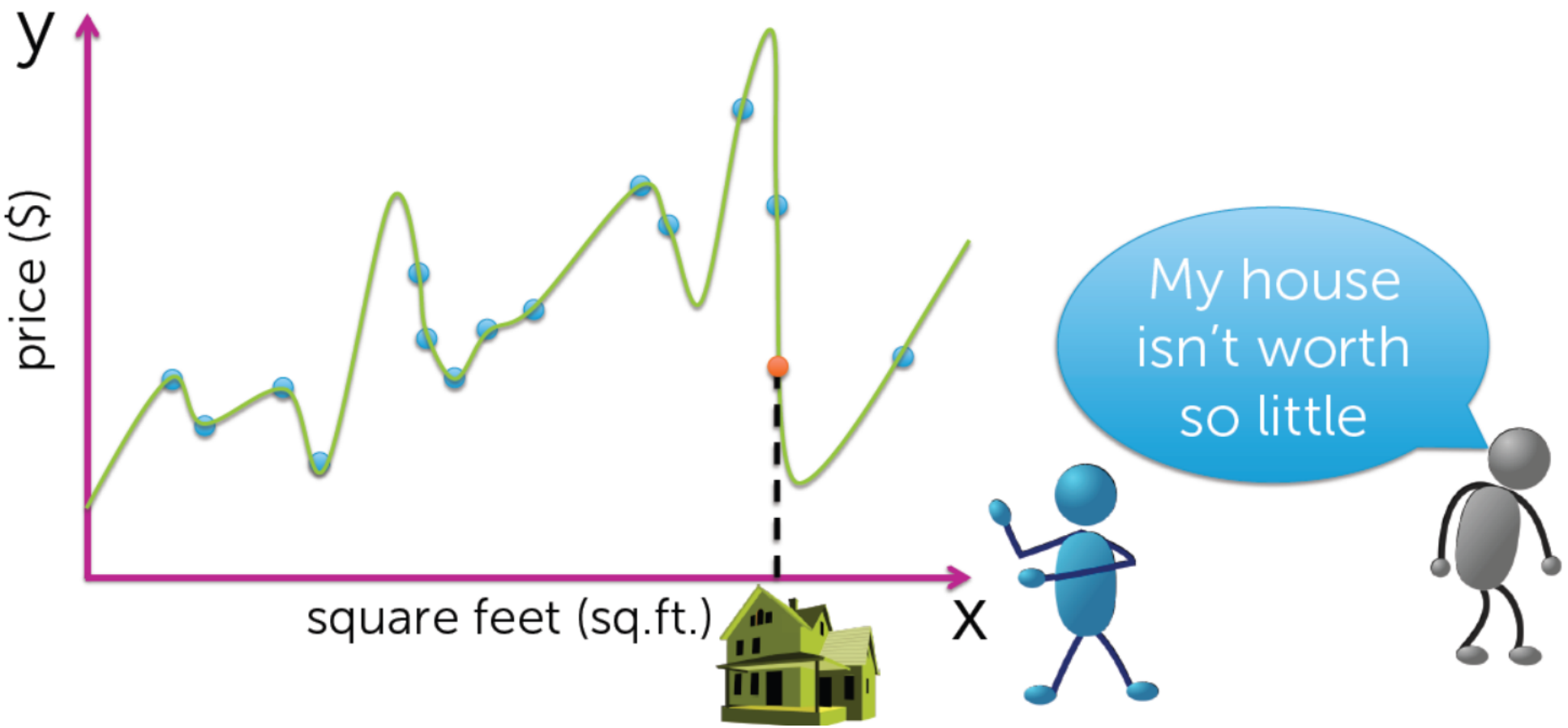
# What about a quadratic function?



# Even higher order polynomial

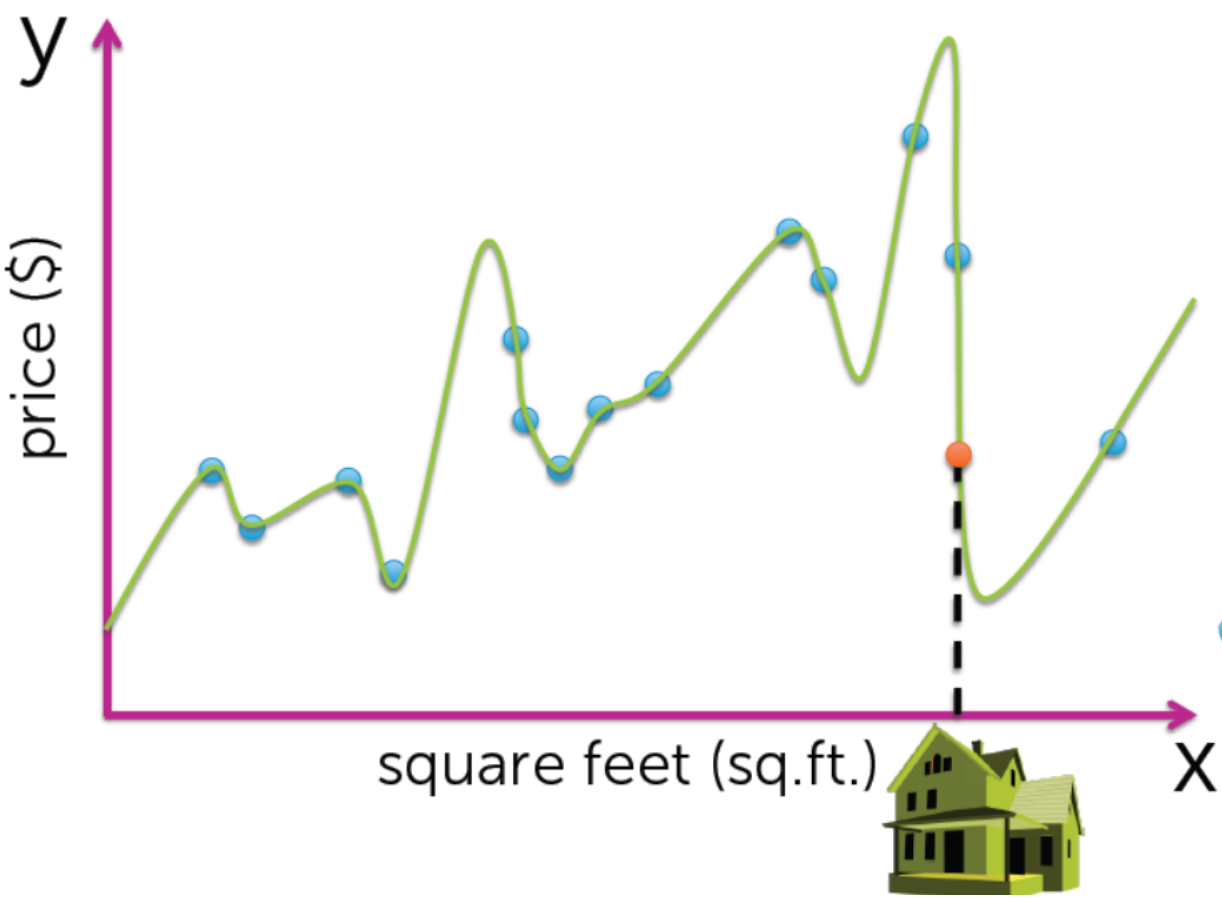


# Do you believe this fit?



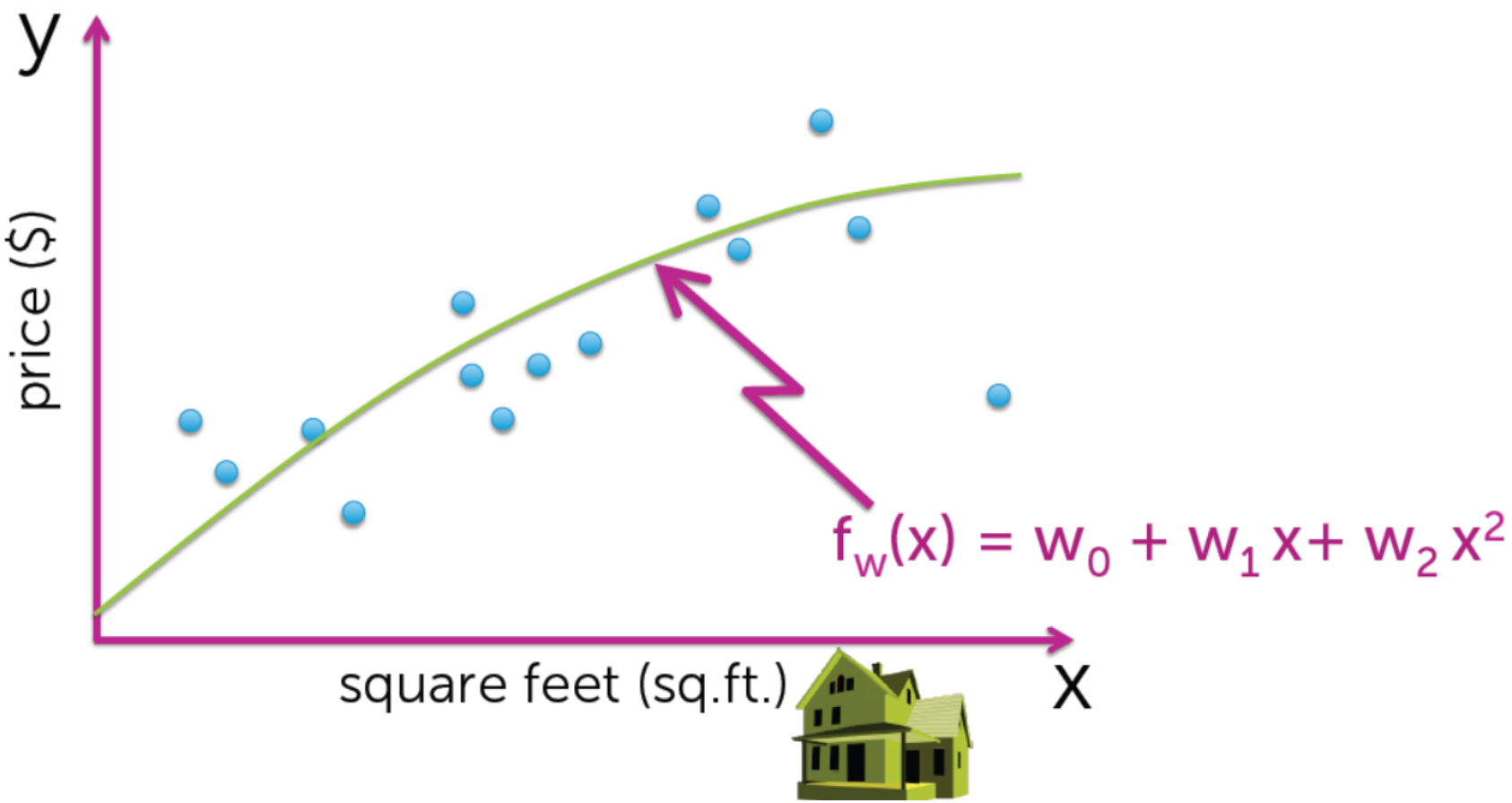
# Evaluating overfitting via training/test split

# Do you believe this fit?




Minimizes RSS,
but bad predictions

# What about a quadratic function?



# How to choose model order/complexity

- Want good predictions, but can't observe future
- **Simulate predictions**

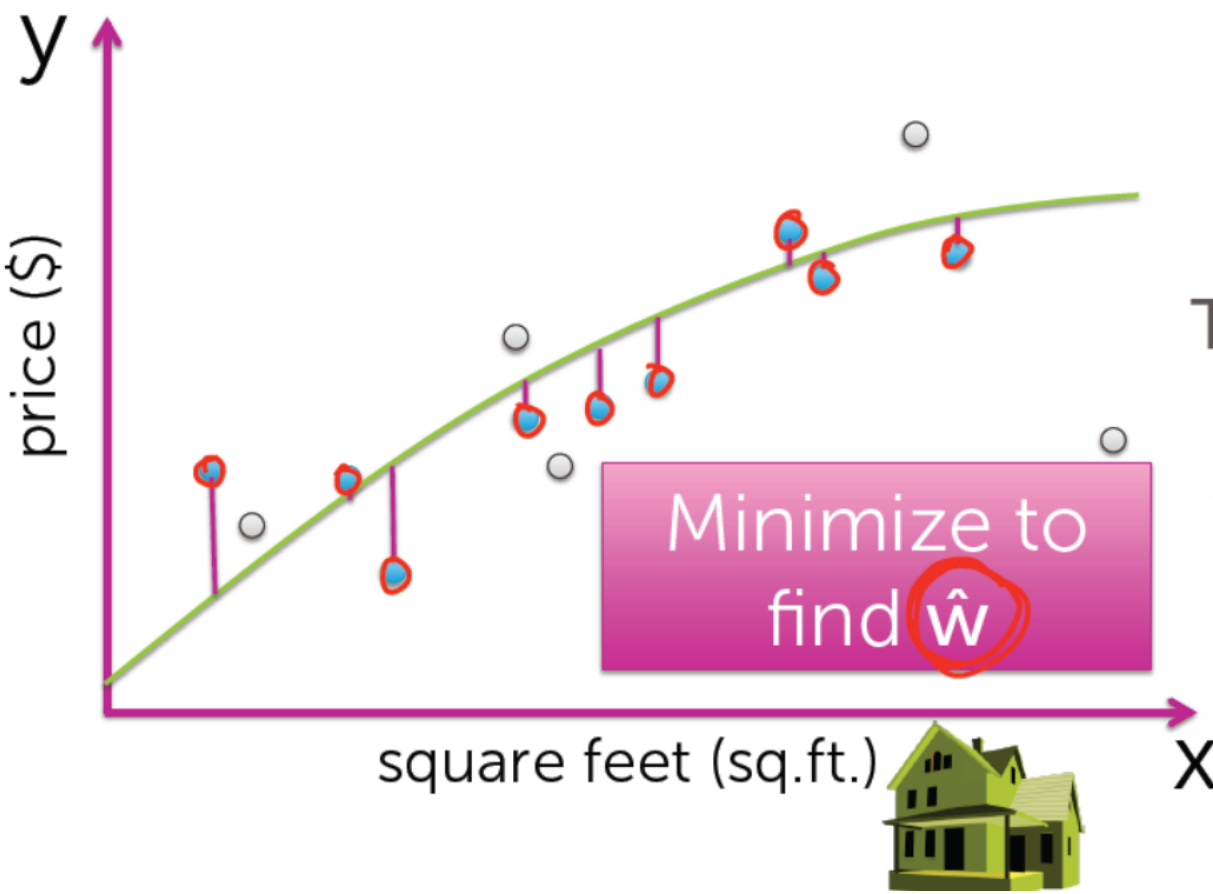
1. Remove some houses
2. Fit model on remaining
3. Predict heldout houses

# Training/test split

**Terminology:** – training set

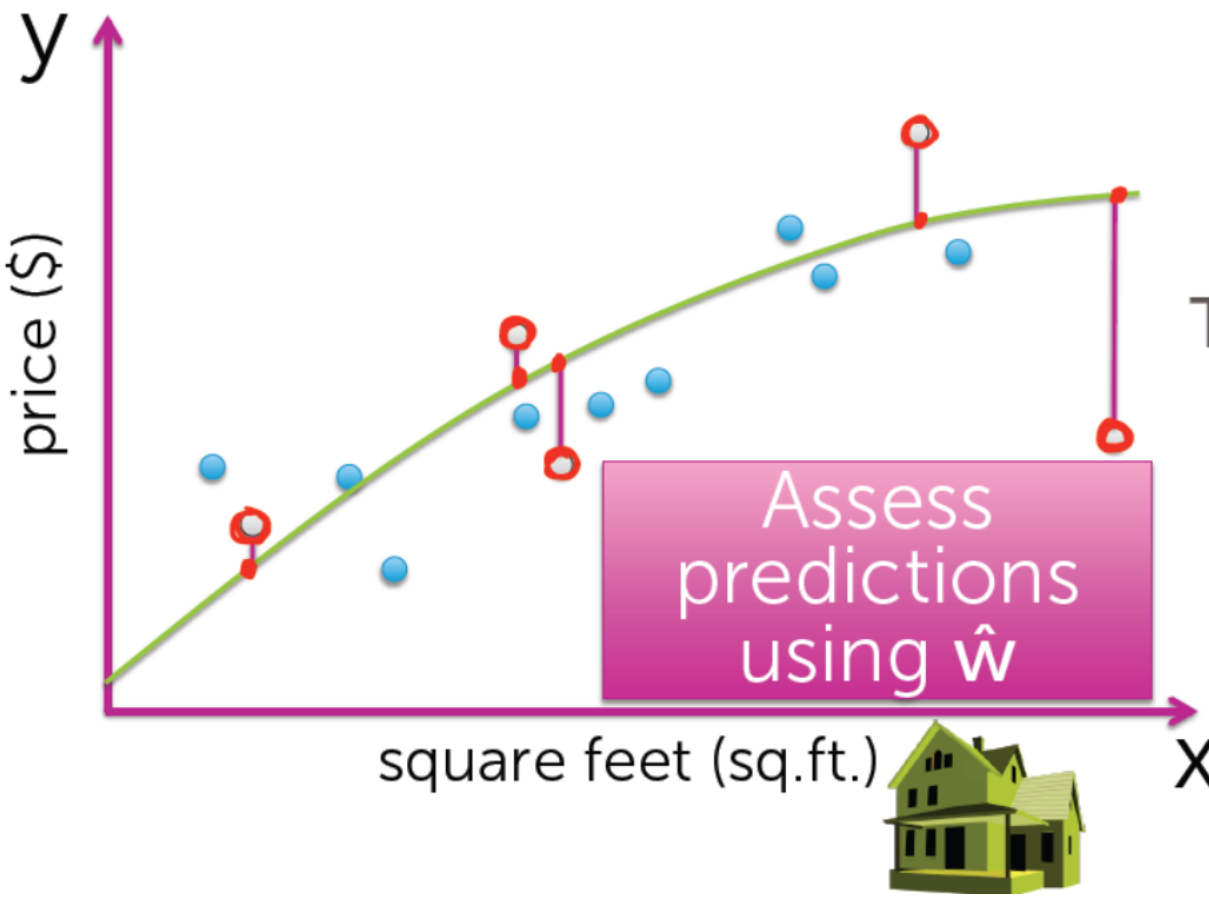
– test set

# Training error



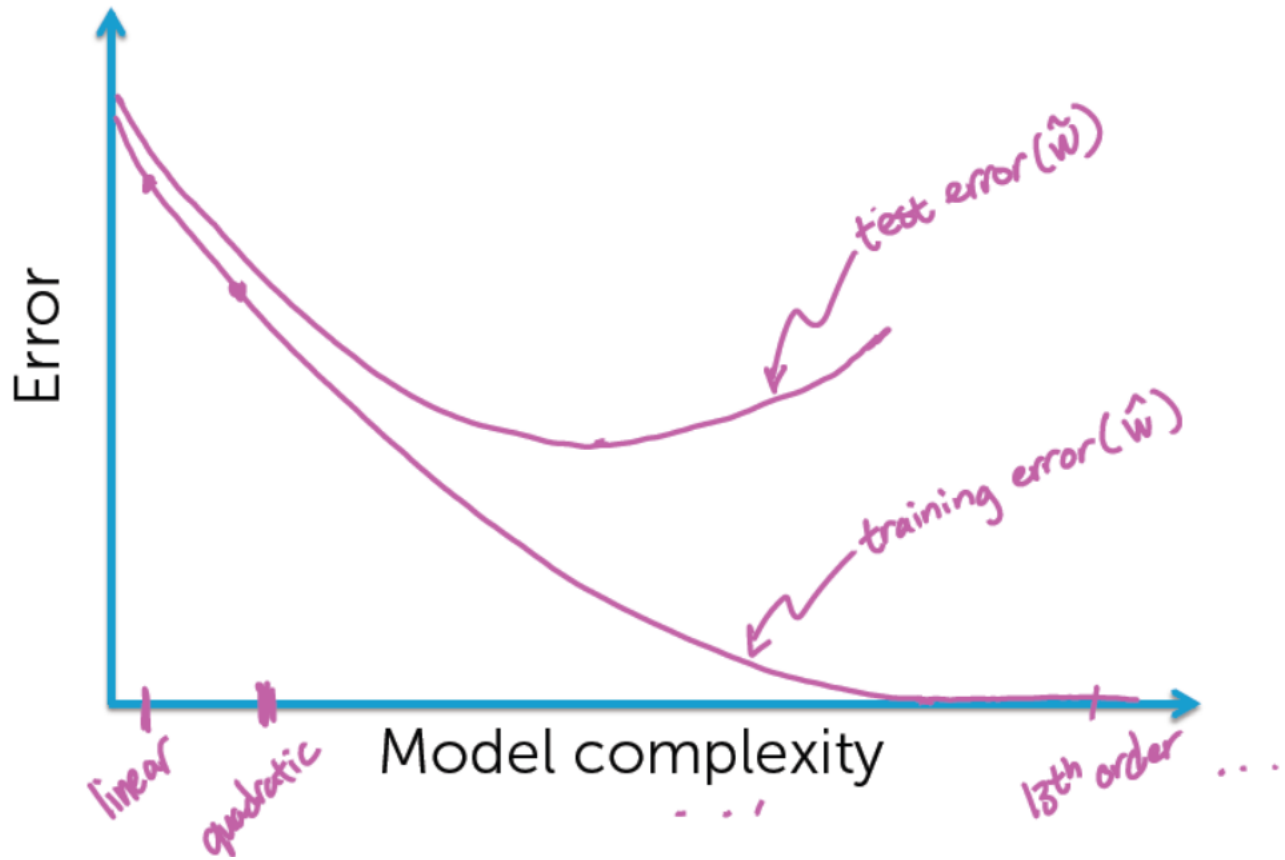

Training error ($w$) =
$(\$_{train\ 1} - f_w(sq.ft._{train\ 1}))^2$
$+ (\$_{train\ 2} - f_w(sq.ft._{train\ 2}))^2$
$+ (\$_{train\ 3} - f_w(sq.ft._{train\ 3}))^2$
+ ... [include all training houses]

# Test error



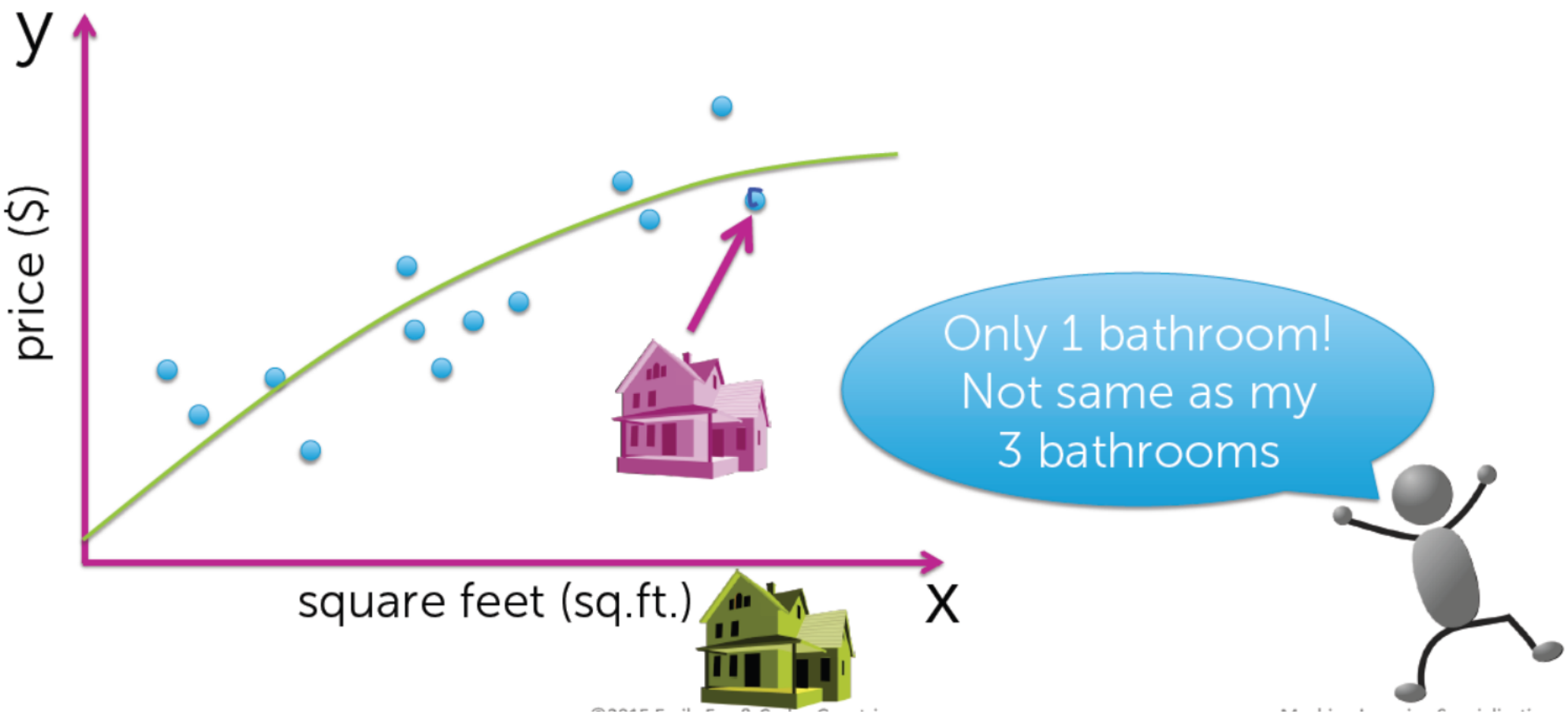

Test error ($\hat{w}$) =
$(\$_{test\ 1} - f_{\hat{w}}(sq.ft._{test\ 1}))^2$
$+ (\$_{test\ 2} - f_{\hat{w}}(sq.ft._{test\ 2}))^2$
$+ (\$_{test\ 3} - f_{\hat{w}}(sq.ft._{test\ 3}))^2$
+ … [include all test houses]

# Training/Test Curves

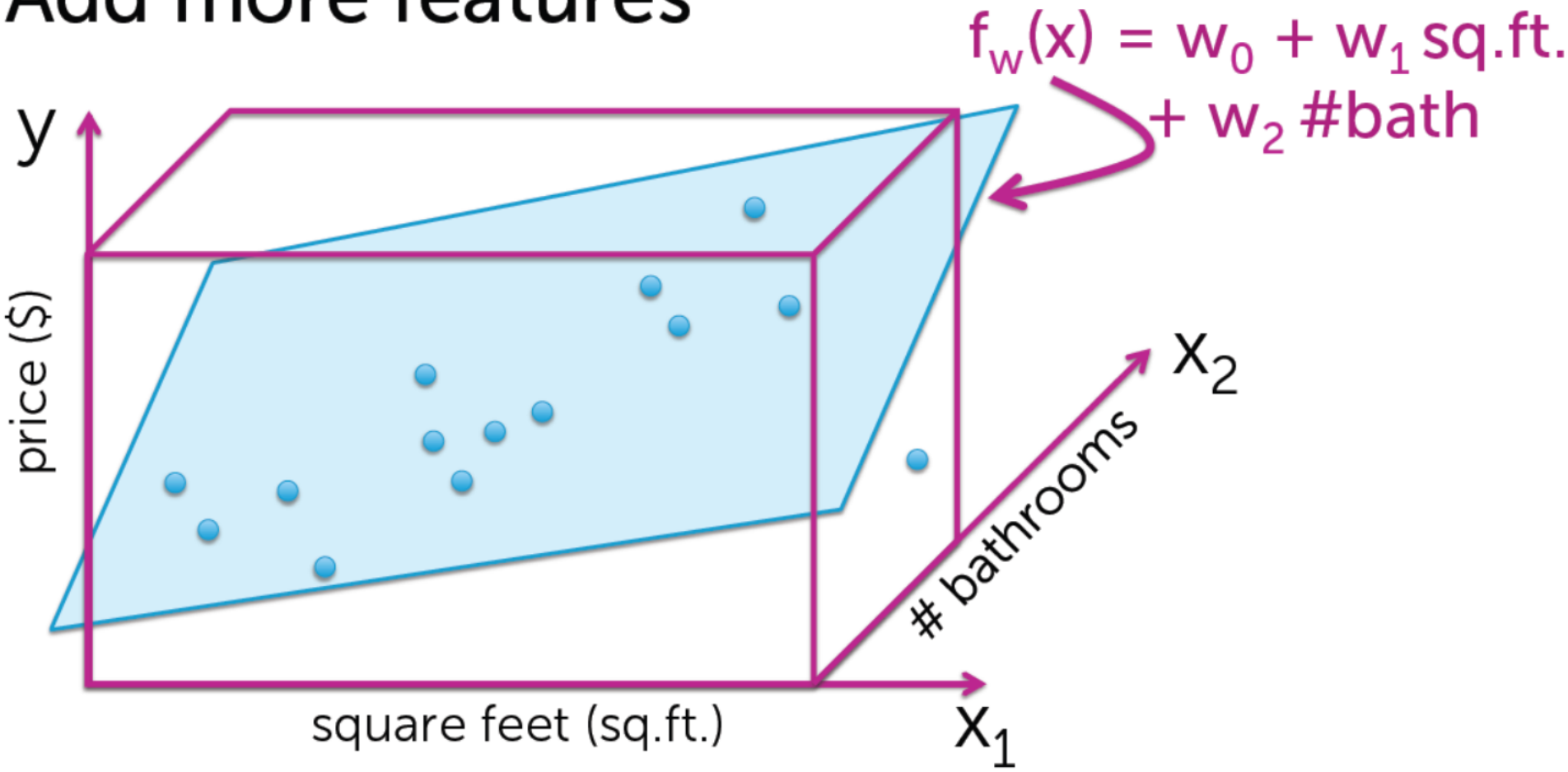


# Adding other features

# Predictions just based on house size



# Add more features

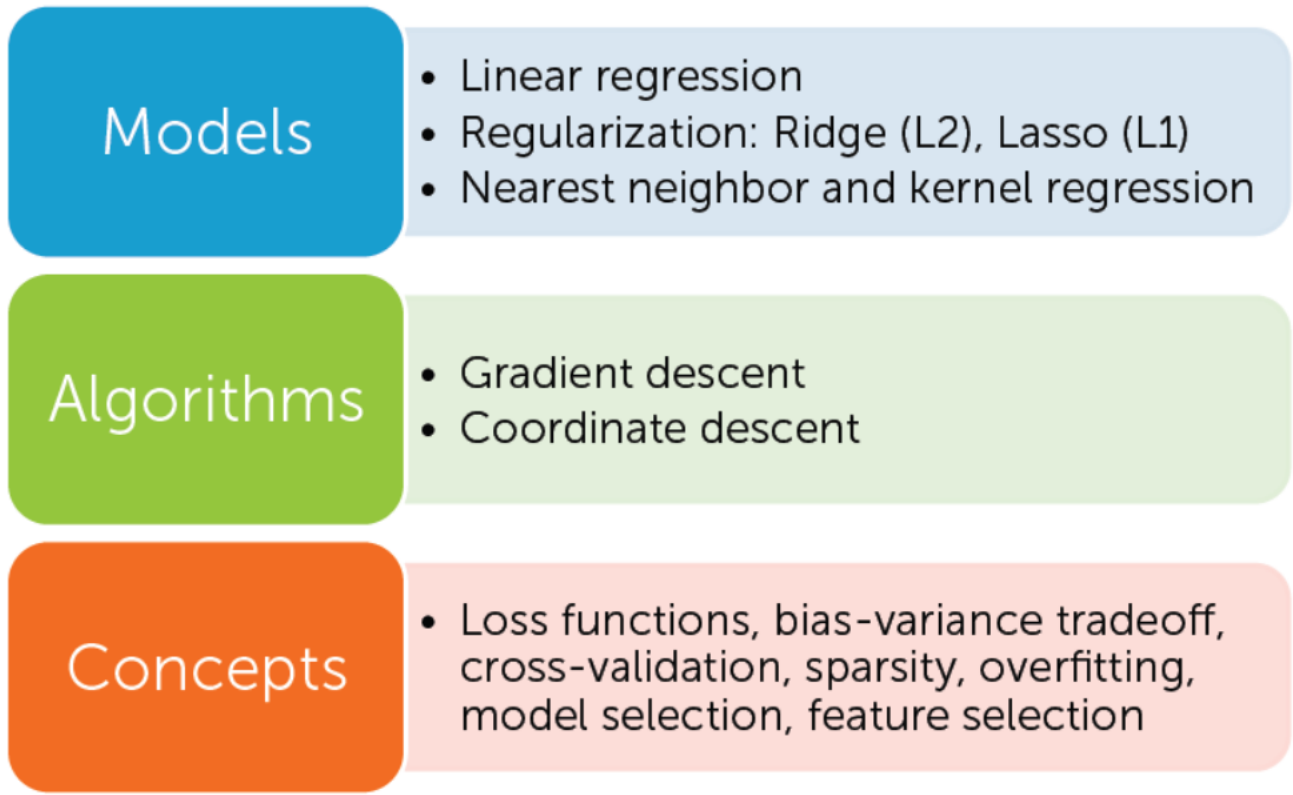


# How many features to use?

- Possible choices:
  - Square feet
  - # bathrooms
  - # bedrooms
  - Lot size
  - Year built
  - ...

# What you can do now...

- Describe the input (features) and output (real-valued predictions) of a regression model
- Calculate a goodness-of-fit metric (e.g., RSS)
- Estimate model parameters by minimizing RSS (algorithms to come...)
- Exploit the estimated model to form predictions
- Perform a training/test split of the data
- Analyze performance of various regression models in terms of test error
- Use test error to avoid overfitting when selecting amongst candidate models
- Describe a regression model using multiple features
- Describe other applications where regression is useful


Models
• Linear regression
• Regularization: Ridge (L2), Lasso (L1)
• Nearest neighbor and kernel regression
Algorithms
• Gradient descent
• Coordinate descent
Concepts
• Loss functions, bias-variance tradeoff, cross-validation, sparsity, overfitting, model selection, feature selection