# Introduction to Machine Learning (ML)

PLURALSIGHT

# Course Goals

- Understand what ML is and isn’t
- Understand the ML workflow
- Understand and gain hands on experience with some popular ML algorithms
- Learn some Python as it relates to ML if you don’t already know it
- Understand relevant theory, but we won’t dwell on theory or mathematics

# Guiding Principle

"A change in perspective is worth 80 IQ points"

–*Alan Kay*

- When coming from engineering or computer science
  - one right way (+ many inferior ways) to do things
  - one right answer
  - 100% correctness
- ...not so in Machine Learning!
  - "It depends"
  - Goldilocks

# What is Machine Learning (ML)?

- It’s the rocketship by which we travel to Planet AI
  - BTW, what’s the difference between ML and AI?
  - Oh…and what’s fueling that rocketship?
- automating automation
- getting computers to program themselves
- letting the data do the work
- How is ML different from traditional software development?
  - Computers produce output from what input?
  - ML is the reverse: data + output = programs

*The Master Algorithm*
by Pedro Domingos

# What is Machine Learning (cont'd)?

- Algorithms that rely on many examples of some phenomenon
- Where do these examples come from?
    - Nature
    - Handcrafted by humans
    - Generated by another algorithm
- ML is also solving a practical problem by
    - Gathering a dataset
    - Building a statistical model from that dataset
    - The model is used somehow to solve that practical problem

*The Hundred-Page Machine Learning Book*
by Andriy Burkov

# What is Machine Learning (cont'd)?

"The term *machine learning* refers to the automated detection of meaningful patterns in data."

- This definition reminds us that there is nothing sinister or magical about Machine Learning…
    - You too can be an ambassador of ML

*Understanding Machine Learning: From Theory to Algorithms*
by Shavel-Shwartz and Ben-David

# Three (Four) Major Types of ML

- **Supervised** learning
  - We know the inputs and the outputs and generate a mapping function that predicts new outputs from new inputs (e.g., benign/malignant)
- **Unsupervised** learning
  - We only know the inputs and attempt to deduce patterns from the inputs (e.g., clustering, anomaly detection, neural networks)
- **Reinforcement learning**
  - *Intelligent agents* take actions in order to maximize cumulative reward
- **Deep Learning / Neural Networks**
  - Typically used for more "brain-like" problems

# Classification vs. Regression

- Consider that ML models are functions which *approximate* some real world phenomenon (because we rarely if ever know the actual underlying function, so we approximate it)

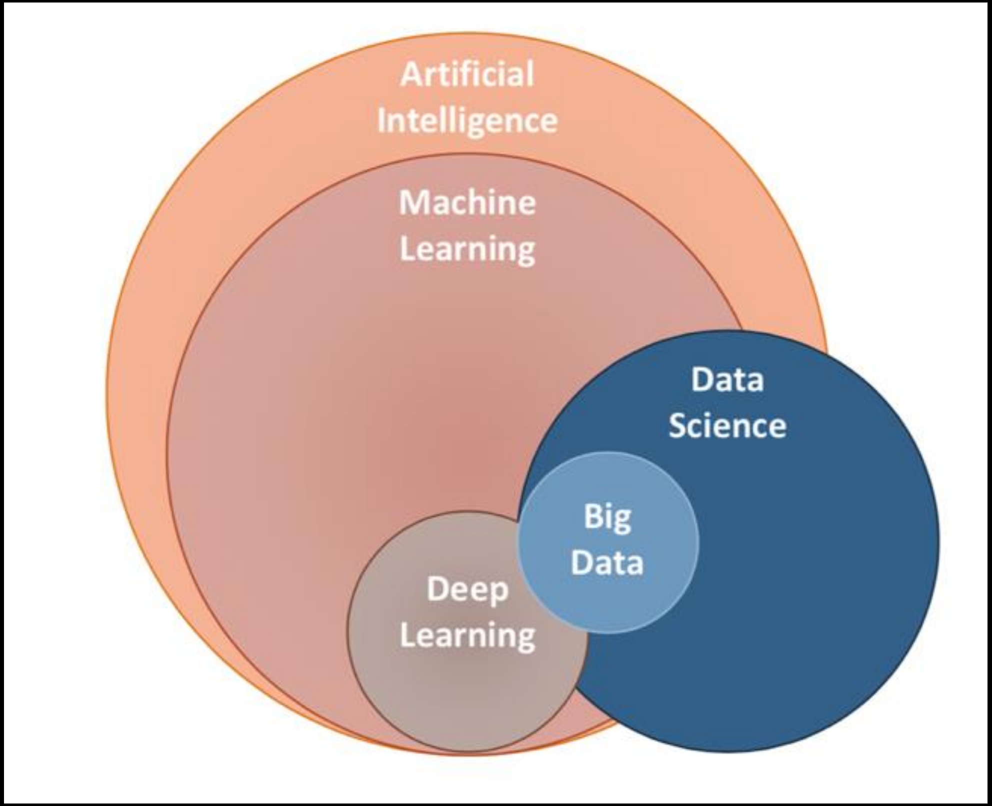
$$\hat{f}(X) = y$$

- X represents the input, and y the output
- Key difference is what form y can be…
    - **Classification**–y is a set of 2+ discrete buckets
        - e.g., cancer/benign, spam/ham, convert/no convert, Yelp ratings
    - **Regression**–y is continuous
        - e.g., selling price of a home,  age

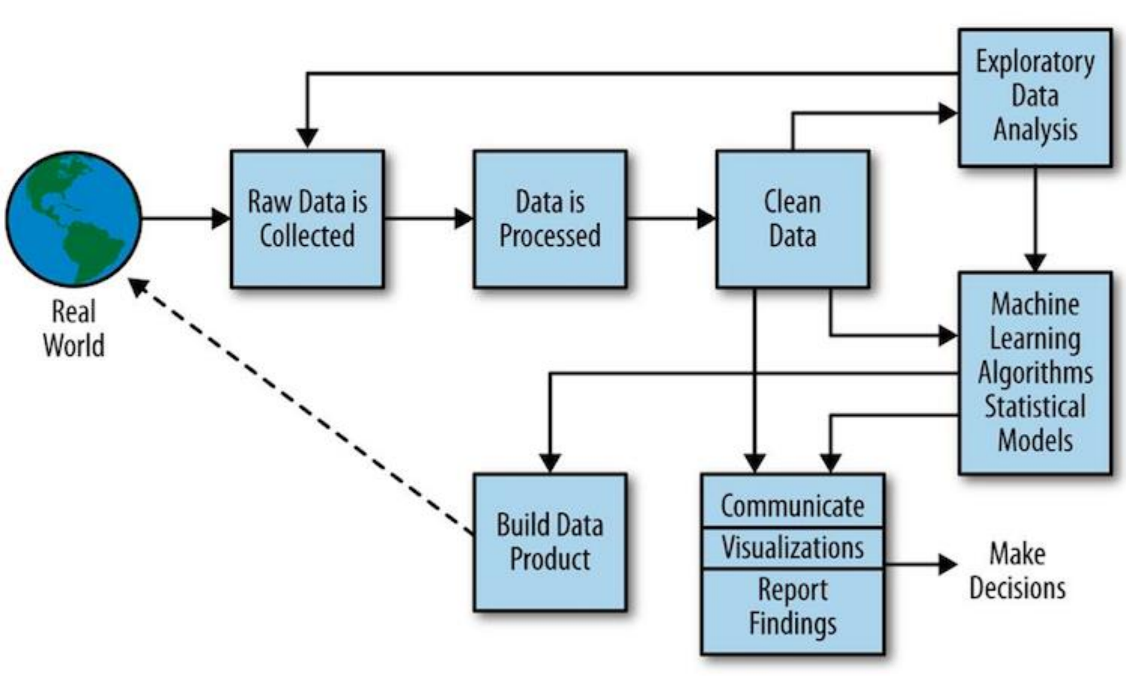
# Demo: A Visual Intro to Machine Learning

- Before we dive in, let's go through a nice visual intro to ML without worrying about all the terminology, which we will get into soon enough…
- r2d3 — A visual introduction to machine learning
- Shortly we'll revisit what they did in a Jupyter notebook

# Data Science vs. Machine Learning



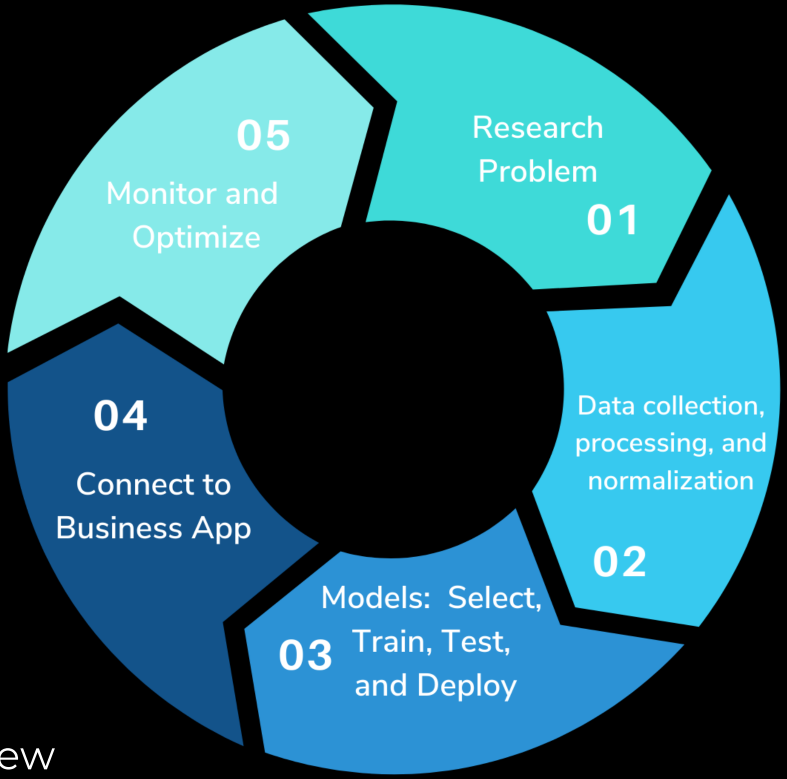
# Data Science Pipeline




*Doing Data Science*
by Cathy O'Neil & Rachel Schutt

# ML Product Lifecycle

- Define the problem
  - Is ML even right for it?
- Collect the data
  - Examine the data!
- Preprocess the data
  - Create test and training sets
- Define models
  - Build a few promising models
- Evaluate
  - Define “success” beforehand
- Deploy
  - Continuous or batch training?
- Monitor
  - Turn data about predictions into new training data if possible!
- Lather, rinse, repeat



# Let's get started with Jupyter notebooks...

- We'll be using sandboxes to do our work for this course
- You should have a link to use at the signup page
- Once we are in our sandbox, we're going to open up the notebook named **`Demo - Jupyter.ipynb`**
- After that we'll revisit the SF vs. NY problem by opening **`Demo - SF vs. NY.ipynb`**

# Demo/Exercise: Pandas

- ML will not succeed unless we
  - Clean our data
  - Visualize our data
  - Understand our data
- Our goal is to get a basic overview of what Pandas can do and how we can use it to ask questions of our data and therefore understand our data
- Pandas can be used to clean our data as well, but we will forego that in this introductory course
  - …but don't think data cleansing is unimportant or something we ignore in real world ML

# Demo/Exercise: scikit-learn

- De facto ML package in Python and what we will use for this course
  - Other popular tools are Keras and PyTorch
- Source code: https://github.com/scikit-learn/scikit-learn
- Before we start…a bit more terminology

# Features vs. Labels

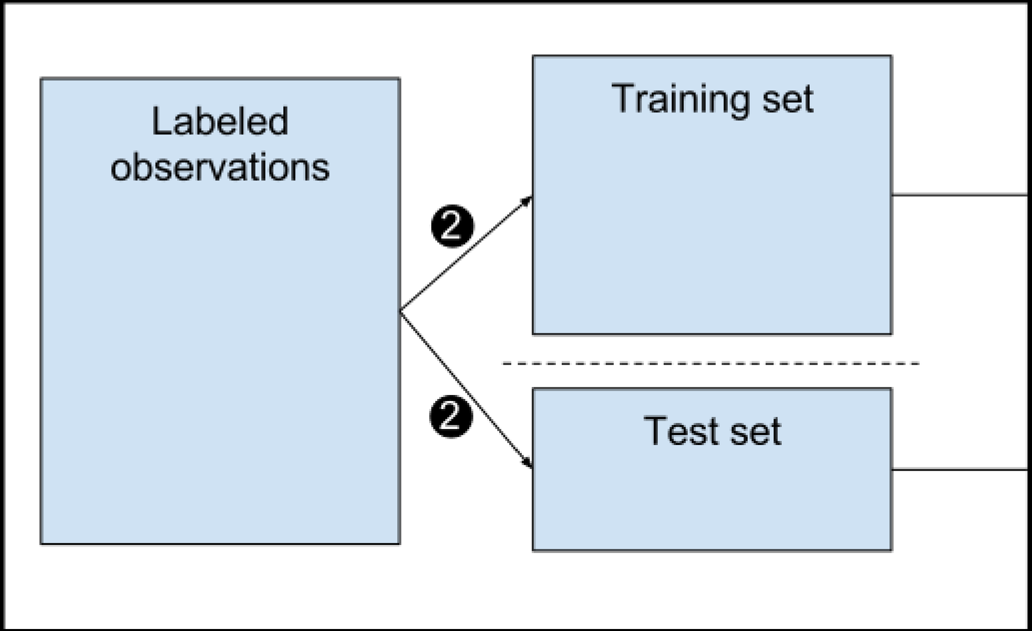
- **Features** (also known as *attributes*, or *variables*)
  - Characteristic or property of your data (e.g., if our data represent people, our features might be age, height, number of years of education, etc.)
  - Features are *input* to an ML algorithm
- **Labels** (also known as *targets*)
  - The data that the algorithm is supposed to predict, e.g., cancer/benign, dog/cat
  - In supervised learning, the input data are *labeled* and the algorithm consumes those labels along with the input features
  - Labels are *output* from an ML model
    - A *model* is a *trained algorithm*

# Training Data vs. Test Data

- **Training Data**–data we use to train our ML algorithm, thereby creating a *model*
- **Test Data**–data we hold back, i.e., data the model doesn't see during training
  - Performance on the test data lets us see how well our model performs on unseen data
- How much should be in training vs. test?
  - 70%/30%
  - 80%/20%
  - These days datasets can be LARGE, so even a test set of 5% could be a LOT of data!

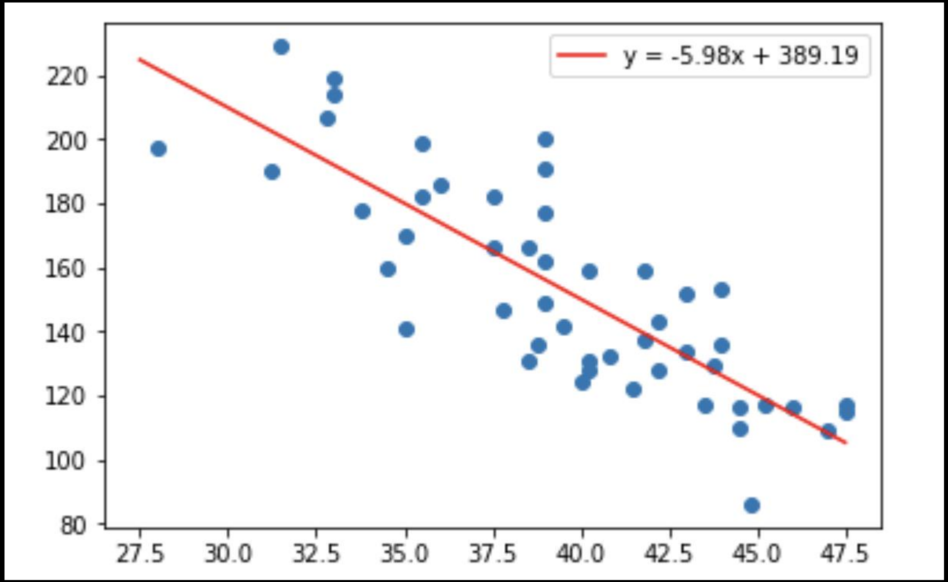
# Training Data vs. Test Data (cont'd)

- For Supervised Learning, it would look like this…



# Our first algorithm: Linear Regression

- relates your input (sometimes called "independent variable") to your output ("dependent variable") by fitting a straight line through your data

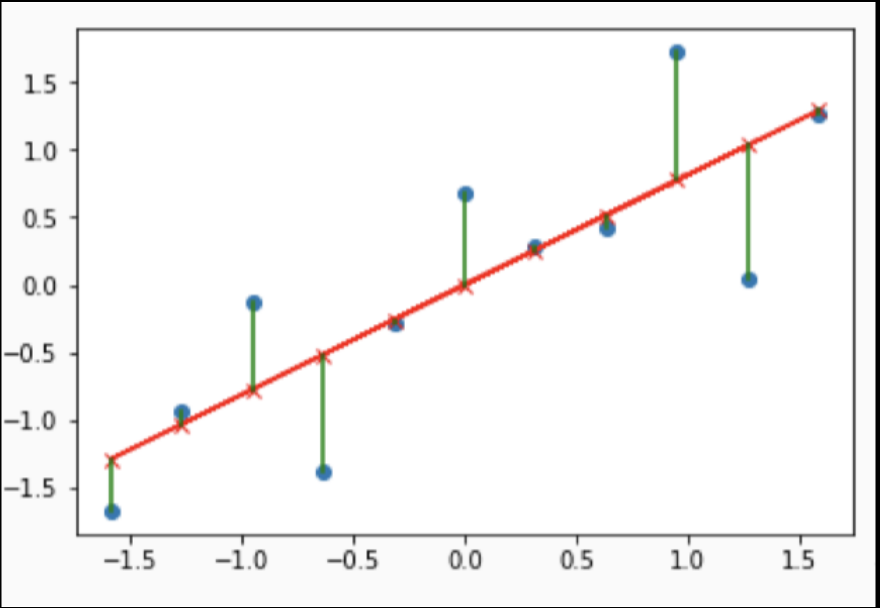


# Our first algorithm: Linear Regression (cont'd)

- Can be expressed using a simple formula that will take you back to elementary school: $\mathbf{y = mx + b}$
- In two dimensions $\mathbf{x}$, $\mathbf{y}$, $\mathbf{m}$, and $\mathbf{b}$ are all single numbers
- In multidimensional problems not much changes, we simply add more $\mathbf{m}$'s and $\mathbf{x}$'s — one for each feature in the training data: $\mathbf{y = m_1x_1 + m_2x_2 + m_3x_3 \ldots m_nx_n + b}$
- Note that we still have only a single "intercept" but one coefficient ($\mathbf{m}$) per input feature ($\mathbf{x}$)

# Our first algorithm: Linear Regression (cont'd)

- Works by minimizing the sum of squared errors (technically called *residuals*) between the data points and the regression line



# Linear Regression: Pros and Cons

- **Pros**
  - Common approach for numeric data
  - Easily interpretable
  - Estimates strength and size of relationships among features and targets
- **Cons**
  - Strong assumptions about the data (i.e., linearity)
  - Sensitive to outliers

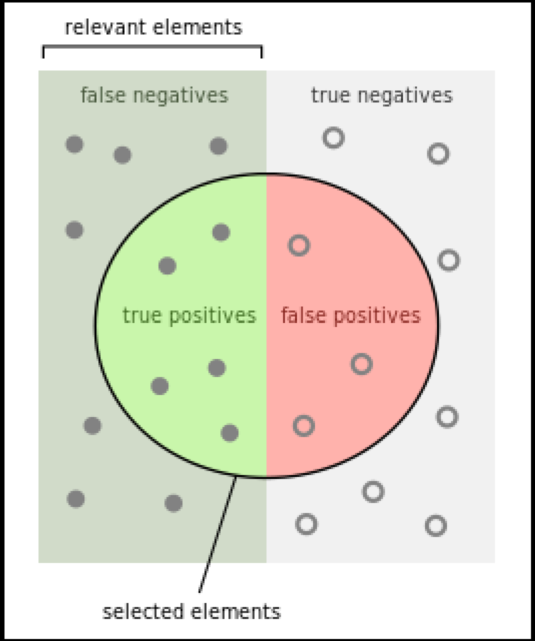
# Demo/Exercise:
# Linear Regression

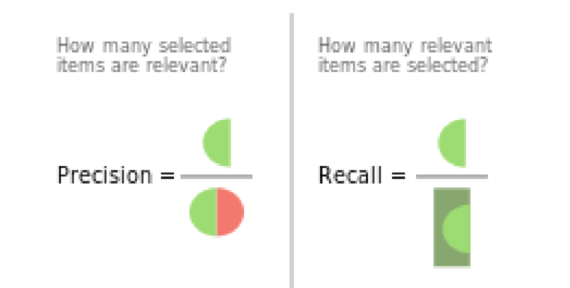
# “That’s not ML, it’s just a formula.”

# How are we doing? Metrics...

- There are many ways to measure our model's performance
- Two common metrics are *precision* and *recall*
- Helpful to think in terms of items return via a search:
  - **Precision** = how many presented items are relevant?
  - **Recall** = how many relevant items were presented?

# Precision vs. Recall




$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

In what kind of real-world system would we prefer recall over precision?

# Hypothetical COVID Identification System

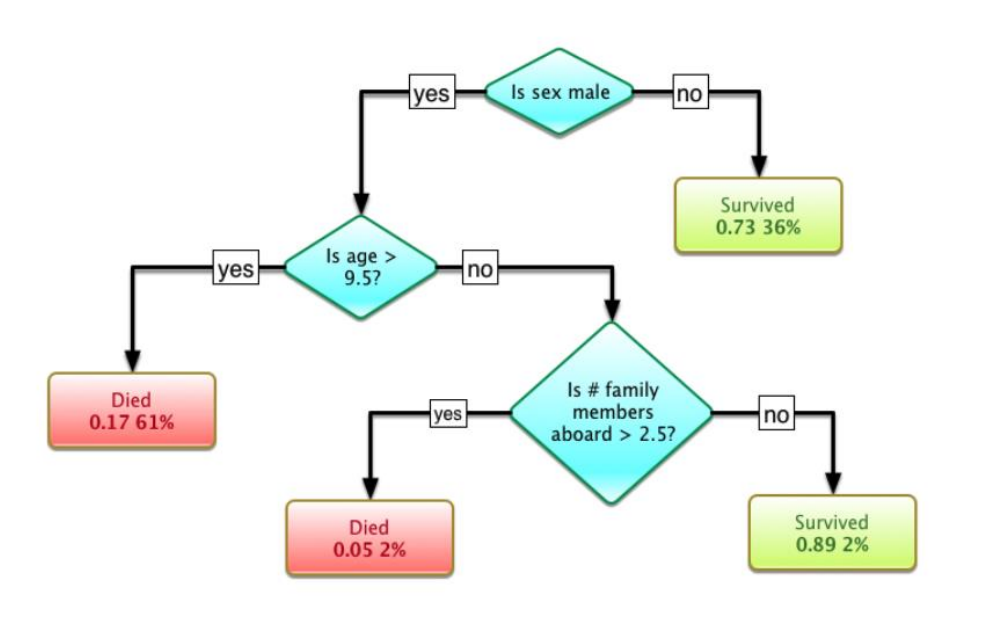
- For airports and other locations where people gather
- Guaranteed not to miss a single person with COVID
- Here it is...
  - Are you ready?
    - **Everyone has COVID!**
- ANALYSIS: We didn’t miss a single person with COVID
- PROBLEM: Too many false positives
- Let's say there are 5M people with COVID in the USA at any given time...
  - Our system identified all, but not very precise!
  - Recall is a function of how many we got right– we got all 5M who are sick so recall was great!

# Other Metrics: MSE and MAE

- MAE = Mean Absolute Error
- MSE = Mean Squared Error
- What do they “mean”?
- When might we use them?
- There are others, but our goal is not to delve into the mathy details, but rather, have a high-level understand of some of these metrics

# Next algorithm: Decision Trees

- Tree-based classifier (like a bunch of if-then statements)
- Models relationships between features and targets



# Decision Trees (cont'd)

- Easy to explain to users
- Can be turned into external representation (i.e., a picture)
- Builds tree where each node divides the set of items based on the value of a feature
- The feature and feature value are chosen based by which one "best" splits the set of items
- two common ways to determine best split are *gini impurity* (default in scikit-learn) and information gain
  - Gini Impurity seeks to maximize homogeneity of subnodes
  - Information Gain seeks to minimize entropy of subnodes

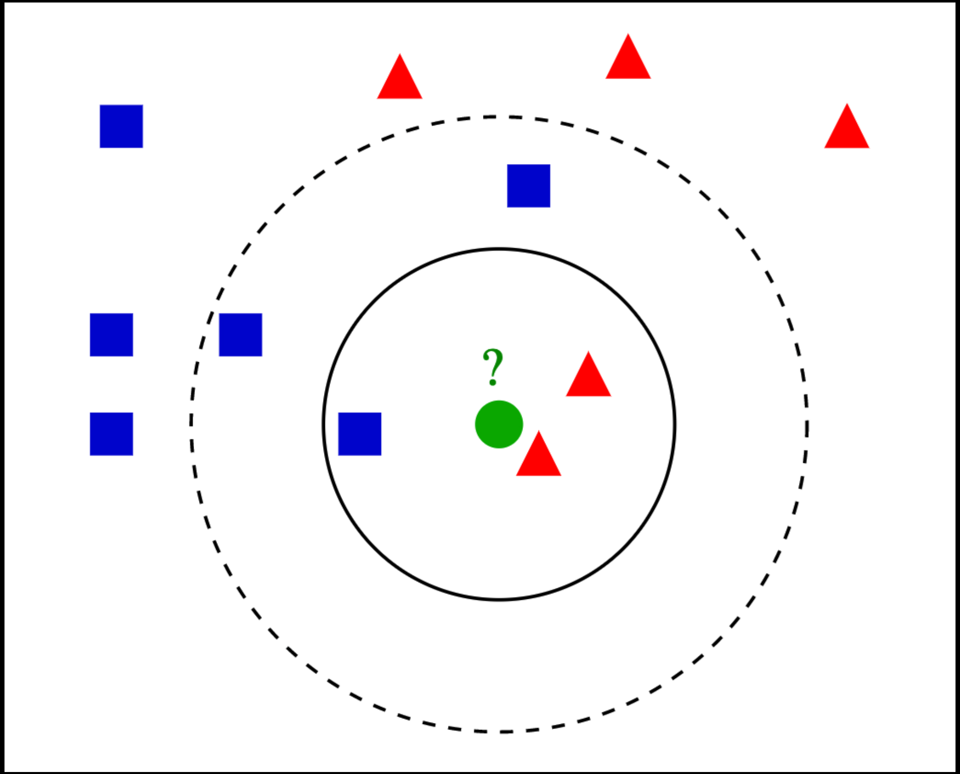
# Demo/Exercise: Decision Trees

# Demo+Exercise: Titanic

# Next algorithm: k-Nearest Neighbors (kNN)

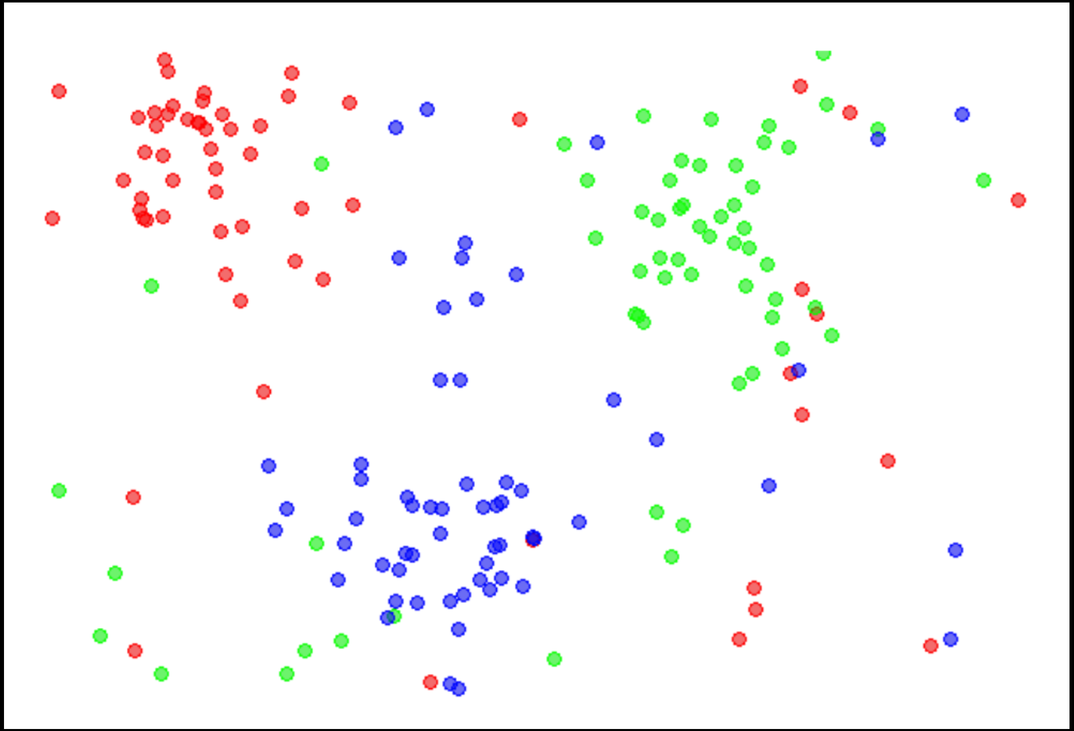
- k-NN *Classification*
  - output is class membership
  - object is classified by a majority vote of its neighbors
  - for k = 1, then the object is simply assigned to the class of that single nearest neighbor
- k-NN *Regression*
  - output is the property value for the object
  - this value is the average of the values of its k nearest neighbors
- Examples?

# k-Nearest Neighbors (kNN)



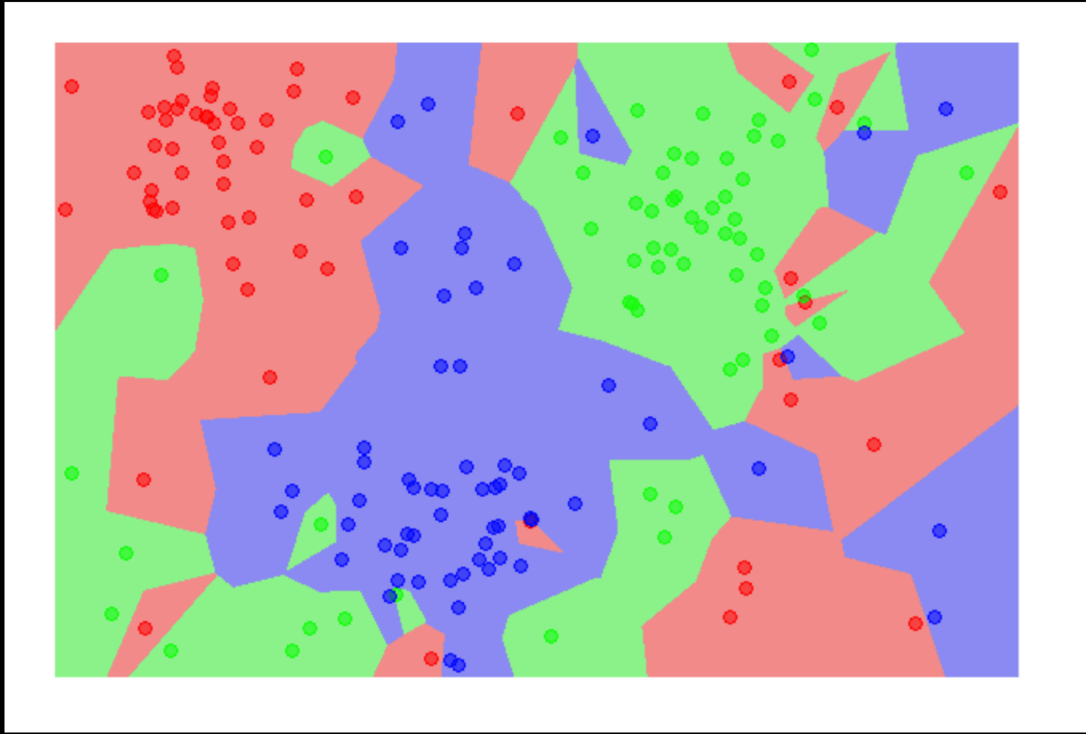

- Green dot is unknown, i.e., is it a red triangle or a blue square?
- Pick a k, and look at that many nearest neighbors
- What if k = 3?
- What if k = 5?

# k-Nearest Neighbors (cont’d)



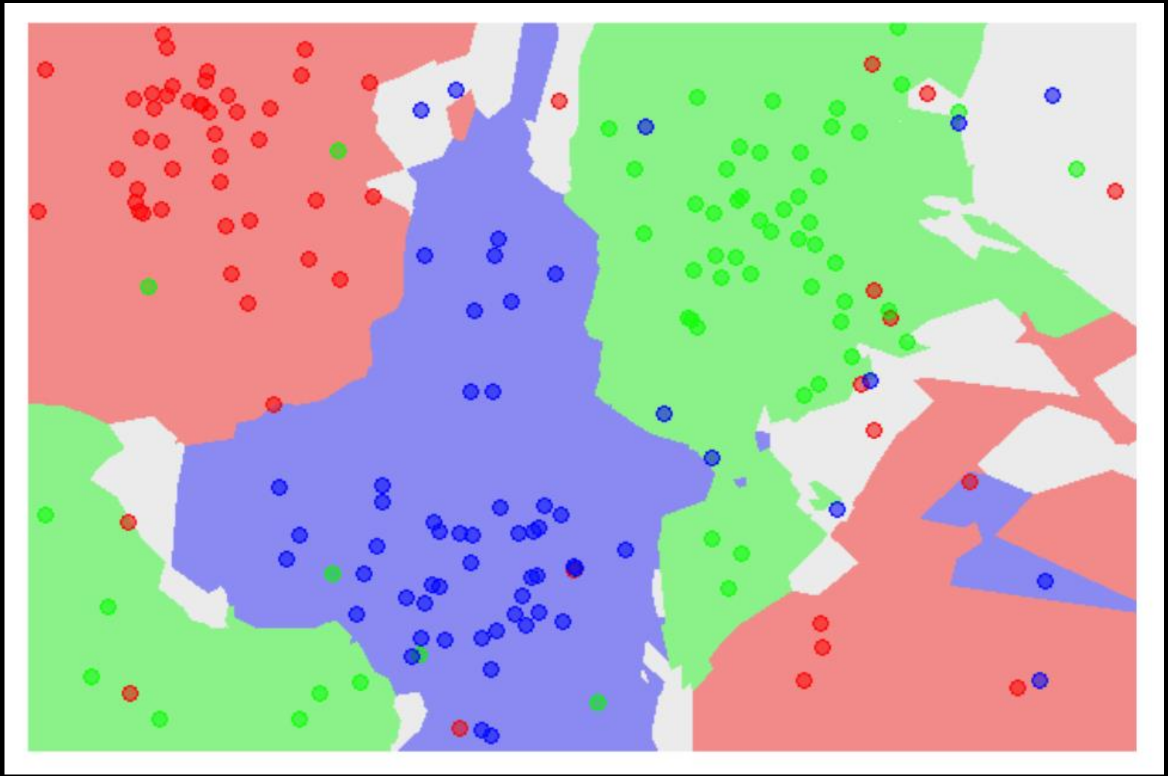
- Suppose we have 3 classes of input data
- How would we classify an unknown point using kNN?

# k-Nearest Neighbors (cont'd)



- k = 1
- Each unknown point is "pulled" towards its nearest neighbor
- If nearest neighbor is "weird", points may be misclassified

# k-Nearest Neighbors (cont’d)



- k = 5
- What happens as k increases?
- What are the white areas?

# k-NN Demos/Exercise

- Interactive K-Nearest Neighbors Demo
- Demo + Exercise notebooks

# A bit more theory...

# Covariance

- Measure of joint variability
  - i.e., how does one variable change as the other changes?
  - this draws our attention to a connection
- Non-zero covariance implies?
  - there is some connection
- Zero covariance implies?
  - Independence
- “Correlation is not causation”

  –Anyone who has ever taken statistics
- Correlation is not causation, but it sure is a hint”

# Correlation/Causation

- What are the possible causal relationships?
  - could be unrelated, i.e., coincidence
  - reverse causation
    - every time windmills are spinning it's really windy
- missed variable, i.e., some other factor causes both
  - whenever I go to sleep with my shoes on I wake up with a headache
  - when it rains, every time I see a flash I hear a boom
- bi-directional relationships
  - temperature and pressure
- or they are actually the same thing
  - °F goes up as °C goes up

# Choosing an ML Algorithm

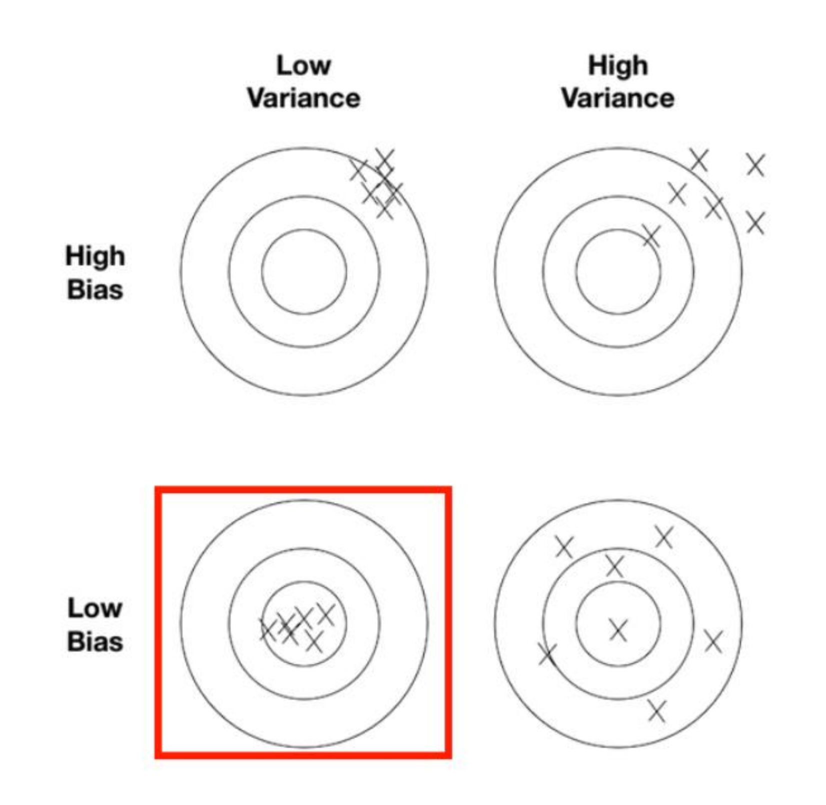
- Now that we've seen a few (and there are MANY more), how do we know what to choose?
- from an exploratory data analysis standpoint, we might try things and see what works
- but if we can't map it back to the business function, it doesn't matter
- if it does work, but doesn't work from an operational standpoint, you can't use it either
    - e.g., can't run on a phone, or breaks email
- therefore, not everything that works is available to us
- sometimes things work well in one domain but don't work well in another … in ML we say:

"THERE IS NO FREE LUNCH!"

# Bias/Variance Tradeoff

- A *model* is the "function" which is produced from an ML algorithm
- *bias* refers to errors made when our model is too simple–i.e., it doesn't consider enough features
- high bias means our model is making errors because it doesn't know enough...also called *underfitting*
    - Credit scoring...what if we consider only income?
- *variance* refers to poor generalization to new, unseen data
    - Our model knows too much or we are feeding it too many features–some of them aren't predictive of the target in general
    - ...also called *overfitting*
- Goldilocks once again
    - As we add more features, bias *decreases*, but when we add too many, variance increases

# Bias/Variance Tradeoff (cont'd)




- **Lower left**: where we'd like to be…training data reflects the population we are trying to predict, good selection of features, good choice of algorithm, etc.
- **Upper left**: underfit…we're consistently wrong (our model is too simple)
- **Lower right**: overfit, we do well on our test data, but we'll generalize poorly to new data
- **Upper right**: good algorithms + bad data = garbage …imagine building a model of credit worthiness based on favorite color

# Feature Selection

- Given that features can make or break our model, we need to be thoughtful about which features to include (or exclude)
- Too many/few features result in overfitting/underfitting
- We've seen some scatterplots where we are able to identify the features which are *predictive* of our target–that's a good place to start
  - Discard features that aren't predictive of the target
  - Our intuition is often bad–look at the data!
- Be careful of correlations between features, which is known as multicollinearity–we typically would discard a feature which is highly correlated with another

# Demo/Exercise: Feature Selection in the Boston Housing Data

# Demo/Exercise: Date Handling + Feature Engineering

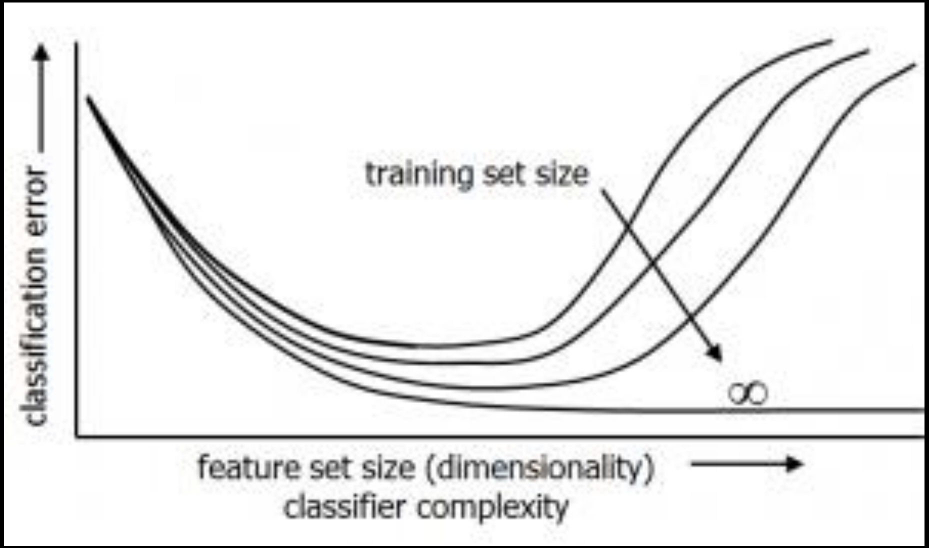
# Some More Theory...

# Curse of Dimensionality

- Models which don't consider "enough" features do not make good predictions (e.g., credit scoring)
    - Technically "underfitting"...but hold that thought
- Our intuition is that considering more features is always better, but that is not always true
- The more features we consider, the more data we need to train our model
- In other words, as dimensionality increases, the volume of the space increases and the data become sparse

# Curse of Dimensionality (cont'd)

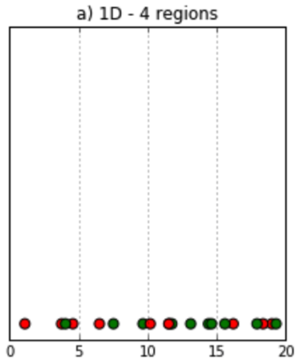
- Hughes Phenomenon: predictive power of a classifier or regressor increases as number of dimensions/features considered increases, then decreases



# Curse of Dimensionality (cont'd)

- Suppose we have 20 data points, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{20}$
  - e.g., perhaps they are gross revenue for companies that are in the trial phase with our SaaS product
- We also have 20 target values, $\mathbf{t}_1, \mathbf{t}_2, \ldots, \mathbf{t}_{20}$
  - **red** = won't convert, i.e., won't pay for our product
  - **green** = converts, i.e., will be a paying customer
- We want an ML classifier to tell us the probability of a customer converting, $\mathbf{p}(\mathbf{t}_n = \mathbf{g}|\mathbf{x}_n)$
- First cut, we'll partition $\mathbf{x}$ into four equal-sized regions

# Curse of Dimensionality (cont'd)




$R_1$: 4 dots, 1 green, 3 red

$R_2$: 3 dots, 2 green, 1 red

$R_3$: 7 dots, 4 green, 3 red

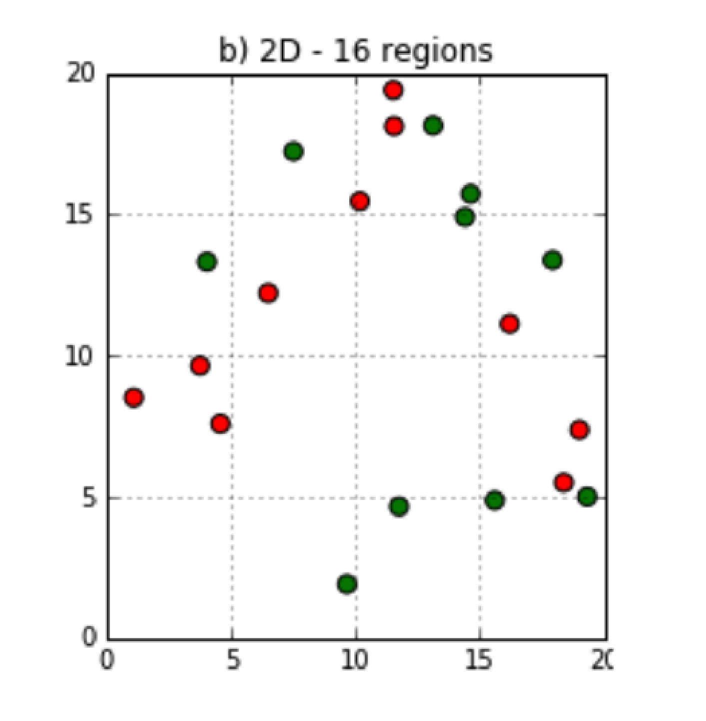
$R_4$: 6 dots, 3 green, 3 red

$$p(t_n = g|R_1) = \frac{1}{4} = 0.25$$

$$\frac{20}{4} = 5 \text{ obs per region on average}$$

# Curse of Dimensionality (cont'd)

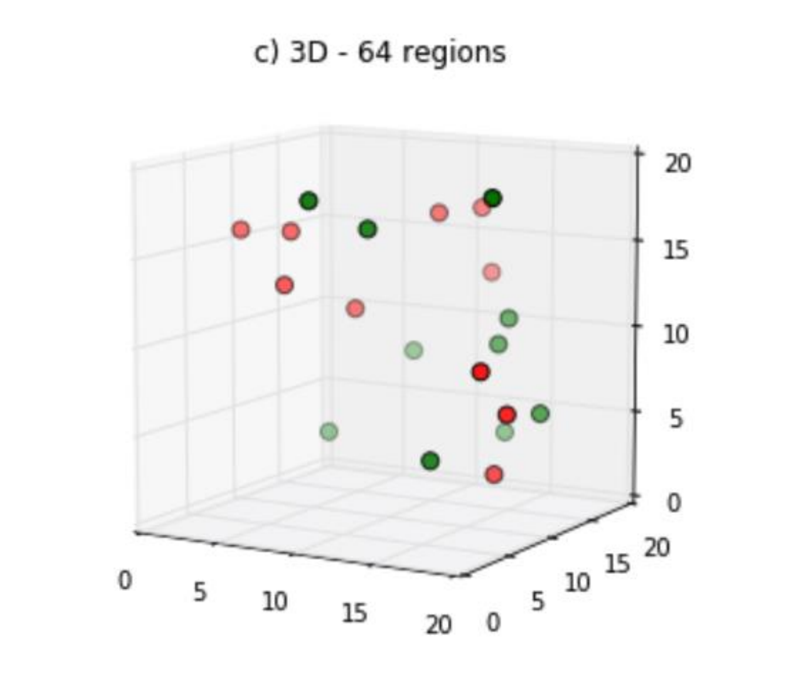
- Now let's consider an additional dimension (e.g., # of employees)




$$\frac{20}{16} = 1.25 \text{ obs per region}$$

# Curse of Dimensionality (cont'd)

- Now let's consider an additional dimension (e.g., years in business)




$$\frac{20}{64} \approx 0.31 \text{ obs per region}$$

Sampling density is proportional to $N^{\frac{1}{D}}$

$$20^{\frac{1}{1}} = 8000^{\frac{1}{3}}$$

# Curse of Dimensionality (cont'd)

- As the number of dimensions increases, number of samples we need grows exponentially
- We need enough dimensions to have ample variation to detect the classification of each input–too many dimensions requires too much data, two few does not have enough feature variation to detect anything
  - In addition, considering too many features can lead to "overfitting"–our model learns idiosyncrasies of our training data and does not generalize well
- Goldilocks is the answer yet again...

# Enough theory...for now

# Model Evaluation

- So far, we haven't considered *hyperparameters*, the parameters to the ML algorithm itself
- We'll want to fine tune our model by tweaking the hyperparameters, but that can be a lot of work if we're doing it by hand
- Instead we'll use GridSearch, a method of automating the search for the best hyperparameters

# Demo: Model Evaluation

# Next algorithm: Logistic Regression

- Despite the name, it’s a *classification* algorithm
- It shares some properties with Linear Regression
  - It's a linear model, that is, it determine a linear combination of all the features using coefficients to scale each feature
  - The coefficients are what is being "learned"
  - Ultimately the model can be represented by a fairly simple mathematical function

# Differences b/w Logistic & Linear Regression

- The shape of the function is not linear, but is *logistic*–also known as a *sigmoid* function
    - These functions have an S-shape
- The function is bounded between 0 and 1, and all of the labels for our dataset must be exactly 0 or 1
    - ...at least for binary classification
    - For multiclass logistic regression this isn't exactly true

# Demo/Exercise: Logistic Regression

# Interesting uses of ML/AI

- JP Morgan has software to find anomalies in contracts https://www.imaginovation.net/blog/ai-in-banking-jp-morgan-case-study-benefits-to-businesses/
- This X does not exist: https://thisxdoesnotexist.com/
- What breed is that dog?
- Cancer detection: https://web.archive.org/web/20210226200143/https://www.analyticsvidhya.com/blog/2018/04/googles-machine-learning-model-can-detect-cancer-real-time/
- https://www.youtube.com/watch?v=toK1OSLep3s

# Prominent AI Failures

- Amazon scraps secret AI recruiting tool that showed bias against women
- Wikipedia – Tay (bot)
- Husky vs. Wolf

# Prominent AI Failures (cont'd)

A husky (on the left) is confused with a wolf, because the pixels (on the right) characterizing wolves are those of the snowy background. This artifact is due to a learning base that was insufficiently representative.

# Further Study

- Ensemble methods (Random Forest, bagging, boosting)
- Model evaluation (Cross Validation, Grid Search)
- Feature reduction
  - Principal Component Analysis
- Additional Models
  - Logistic Regression
  - Support Vector Machines

# Thank you!

PLURALSIGHT