Let's Learn about (Machine) Learning!
An Introduction to Machine Learning for Survey Researchers

Adam Eck

Social Intelligence Lab
Computer Science Department, Oberlin College

AAPOR 2019 Short Course

Toronto, Ontario

May 19, 2019

OBERLIN
COLLEGE OF ARTS & SCIENCES
CONSERVATORY OF MUSIC

Acknowledgements

- Two outstanding resources for machine learning:
  - "Machine Learning" by Tom Mitchell (1997)
  - "The Elements of Statistical Learning" by Trevor Hastie, Robert Tibshirani, and Jerome Friedman (2009)

- Some of these slides are based on multiple sources (used with copyright permission)
  - Lectures from CSCI 374 at Oberlin College
  - Previous conference talks
    - (Eck, 2016), (Eck & Soh, 2017), (Eck et al., 2018)

What is Machine Learning?

A computer **learns** if it improves its performance on a task over time based on experience.

Adapted from Mitchell (1997)

- Three key elements
  - **Task:** what we ask the computer to do
  - **Performance:** how well the computer does at that task
  - **Experience:** data we provide to the computer to learn from

- ***Iterative*** process: computer gets better at a task over time (but *not necessarily every time*)

Tasks vs. Performance and Experience

- How we measure **performance** and what **experience** we use depends on the task

Task	Performance	Experience
Predict Visits from Mobile Geolocation Data	Accuracy of Visit Identification	Device data collected from app throughout the day
Automated Assistant	Quality of interview, Helpfulness of Assistance	Outcomes of interactions with people
Predicting Breakoff from Web Surveys	Accuracy of predictions	Paradata collected during prior web surveys
Automating Behavior Coding	Matching between automated/manual codes	Transcripts that are manually coded by a person

What is Machine Learning?

- Machine learning is a **paradigm** for problem solving

- **Traditional approach**: develop an algorithm to solve a given problem
  - List of steps to take (a recipe or plan)
  - Implement those steps in a programming language
  - Engineering-focused approach

- **Learning approach**: provide many examples for the computer to learn from
  - It discovers patterns that solve problems in examples
  - Data-focused approach

Why Do We Want Learning?

- **Computer Science Perspective:**
  - Task requires **too many steps** to program (autonomous cars)
  - Task is **too difficult** to explain (no human expertise)
  - Computer can find a solution **more efficiently** or **effectively**
  - Want software to be able to **adapt** (to changing inputs)

- Long-term impact: will open up the creation of software as tools by non-programmers

Why Do We Want Learning?

- **Data Science Perspective:**
  - Can automatically discover patterns and relationships in data without requiring prior knowledge
    - Might have no theory describing the relationships between predictors and outcomes
    - Especially as the sizes of our data sets grow (organic/Big Data)
  - Can validate existing theory
  - Eventually to automate tasks that a computer is well suited to perform
    - Free up people to work on more complicated tasks

Terminology

Feature or **Attribute**: a particular type of measurement

AKA: covariates, independent variables, spreadsheet column

Instance: a collection of attribute values for one entity

AKA: observation, data point, spreadsheet row

Label: a particular value assigned to an instance

AKA: outcome, dependent variable

Class: a possible value for a label

Categorical: **Classification** Numerical: **Regression**

Terminology

- Example Data Set:

	Attributes/Features				Labels
	Age	Gender	Education	Income	Respond?
$Instance_1$	64	F	College	60-100k	R
$Instance_2$	43	M	HS	<30k	NR
$Instance_3$	39	F	Less than HS	>100k	NR
$Instance_4$	57	M	College	30-60k	R

Classes = Respondent (R) or Non-respondent (NR)

Steps to Setting up Learning

1. Determine **type of learning problem**
   - What task should the computer learn?
   - What data is available?
2. Determine what **representation** you want to use
   - How should the computer "think" about the task?
   - What are the characteristics of the task?
3. Determine the **learning algorithm** you want to use
   - How should the computer create and refine the representation?
   - What are the characteristics of the data?

Supervised Learning Problems

Supervised Learning: a computer learns to assign an output for a given set of inputs

- Tasks: assign some label *Y* to data *X*
  - Stock market prediction; weather forecasting; image recognition; cancer identification
- Performance Measures: accuracy, sensitivity (recall), specificity, precision
  - How many labels *Y* correct? How many of each type (e.g., +, -)?
- Experience: data *X* with predetermined labels *Y*

Supervised Learning Tasks

Sample Frame Construction: identifying locations of interest from Google Maps images or subpopulations from larger sample frames

Response Propensities: predicting whether a respondent will respond to a survey/interview request (responsive design)

Imputing Item Non-Response Values: predicting what respondents might have answered based on their responses to other questions

Supervised Learning Tasks

Automated Coding: coding open-ended responses and respondent/interviewer behaviors from audio/transcripts

Data Analysis: predicting future outcomes based on patterns discovered in collected survey data

Unsupervised Learning Problems

Unsupervised Learning: a computer learns relationships between sets of inputs (features or instances)

- Tasks: find patterns in inputs *X*
  - Discovering categories of customers, finding frequently co-occurring events, creating more compact representations of data
- Performance Measures: similarity, uniqueness
- Experience: data *X* (with no labels *Y*)

Unsupervised Learning Problems

- How to measure similarity? Distance functions!
  - Manhattan and Euclidian
    - How different are the values?
  - Hamming
    - Number of features with different values?
  - Jaccard
    - How many shared over how many values?
    - Useful if instances have different numbers of features (e.g., missing values)

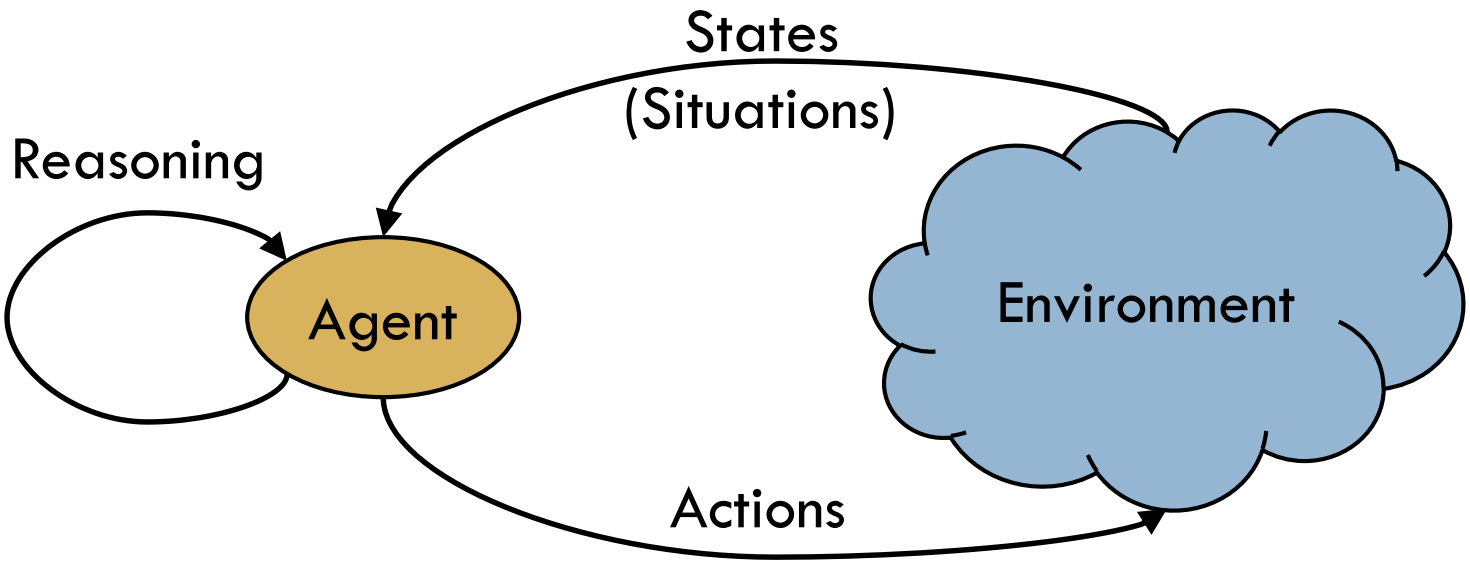
Unsupervised Learning Tasks

Sample Frame Construction: Creating strata of respondents based on shared characteristics for stratified sampling designs

Behavior Modeling: discovering types of behaviors exhibited by respondents and interviewers based on paradata

Questionnaire Design: identifying common responses between questions to reduce questionnaire size (adaptive design)

Reinforcement Learning Problems



Reinforcement Learning Problems

Reinforcement Learning: a computer learns how to choose actions to accomplish one or more goals

- Tasks: find sequences of actions that change the state of the world to ones that are desirable
  - Determining strategies for winning games, robotic movement, autonomous cars, human-agent interactions
- Performance Measures: cumulative rewards/costs, numbers of actions required
- Experience: state/action/reward combinations

Reinforcement Learning Problems

Customized Questionnaires: intelligent surveys that adapt what questions to ask based on data already collected and respondent's behavior

Automated Interviewers: physical devices (e.g., Google Home®, Amazon Alexa®, Apple HomePod®) that interact with respondents and periodically perform interviews

Survey Recommendations: matching respondents to surveys or prior survey questions for particular purposes

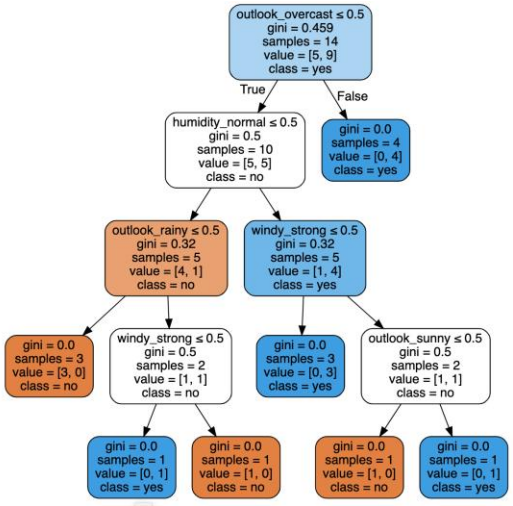
Supervised Machine Learning

- Next, we are going to look **more in-depth** at approaches for **supervised machine learning**
  - This is the most common type of machine learning problem

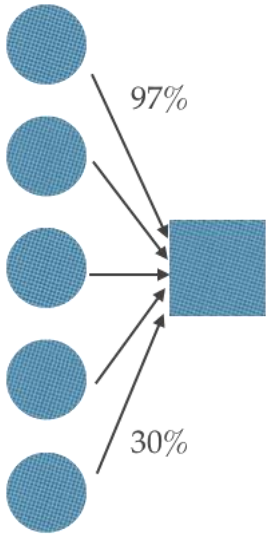
- What are Machine Learning Representations?
- What are Decision Trees and Random Forests?
- What are Neural Networks?
- What are other common approaches?

Supervised Learning Representations

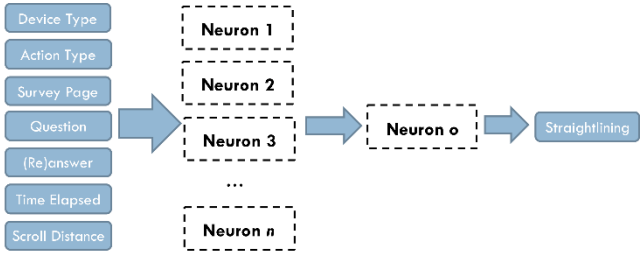
Representation: how the computer thinks about what it is learning (how it organizes information, type of model)



Decision Trees

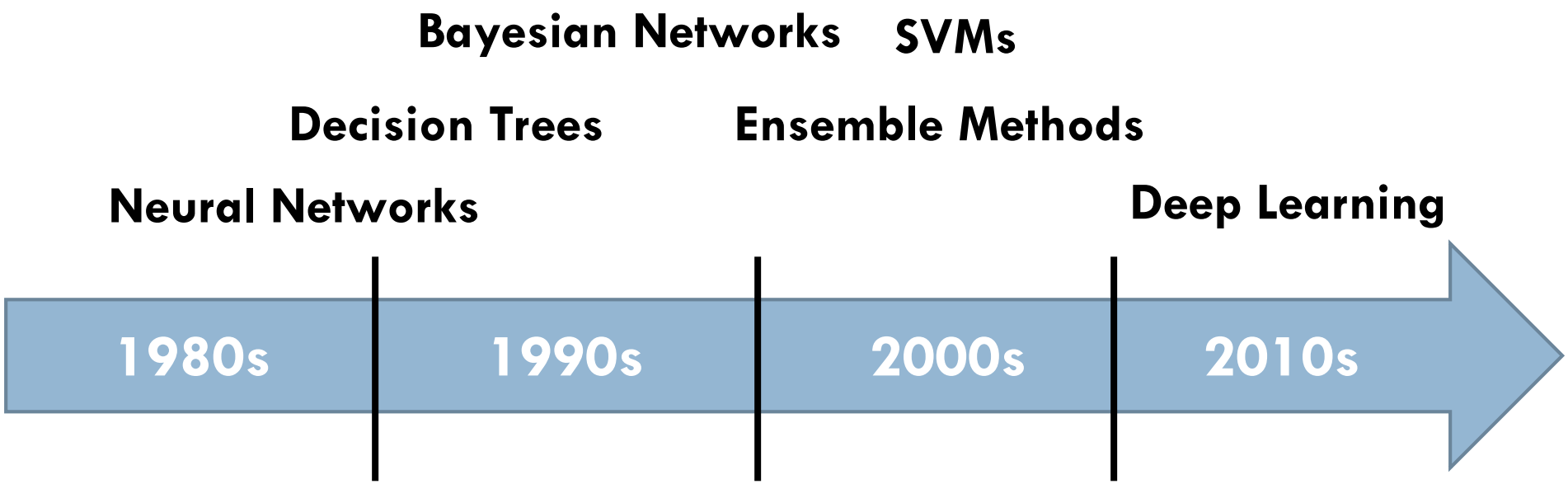


Bayesian Networks



Neural Network

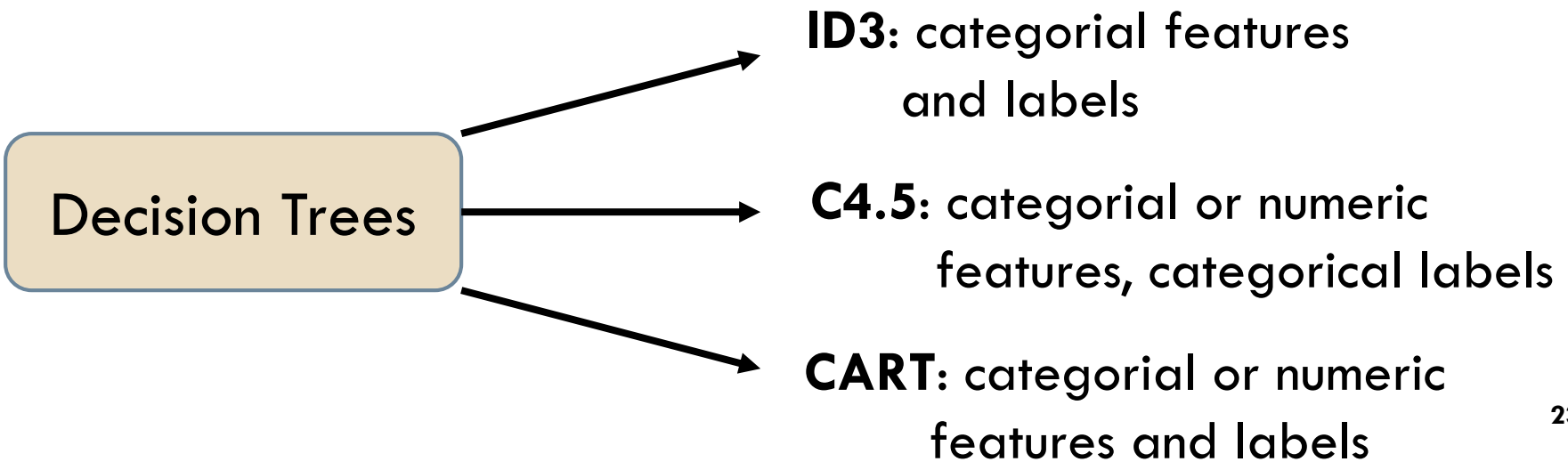
Historical Development



Supervised Learning Algorithms

Algorithm: what sequence of steps the computer follows to learn a model from data

- Often multiple choices for same representation
  - Offer different improvements over other algorithms



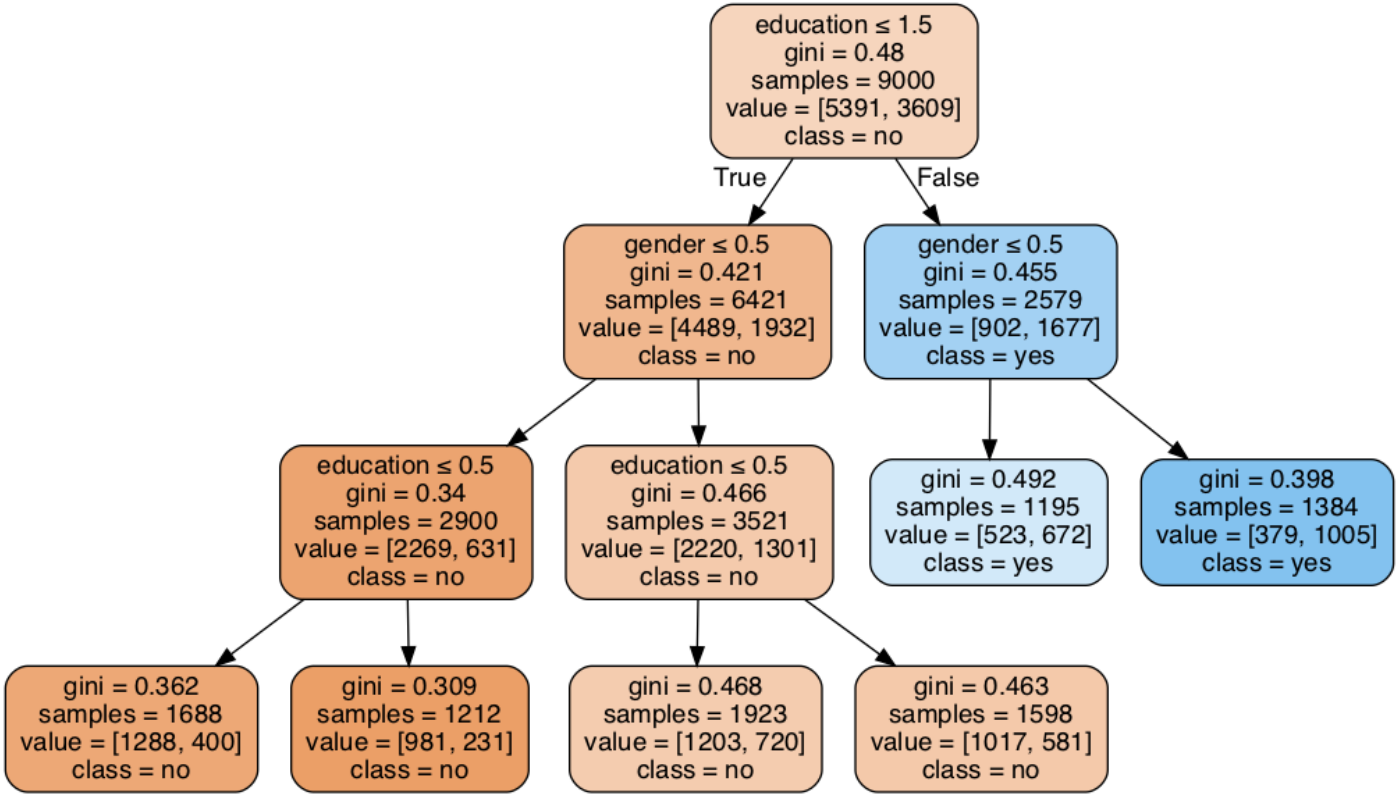
Illustrative Example

Response Propensities: predicting whether a respondent will respond to a survey/interview request (responsive design)

- Dataset from "An Introduction to Machine Learning Methods for Survey Researchers" in *Survey Practice* (Buskirk et al., 2018)
- Considering attributes:
  - Age, Cellphone, Education, Gender, Income, Landline, Region
- Labels: Respondent or Non-Respondent

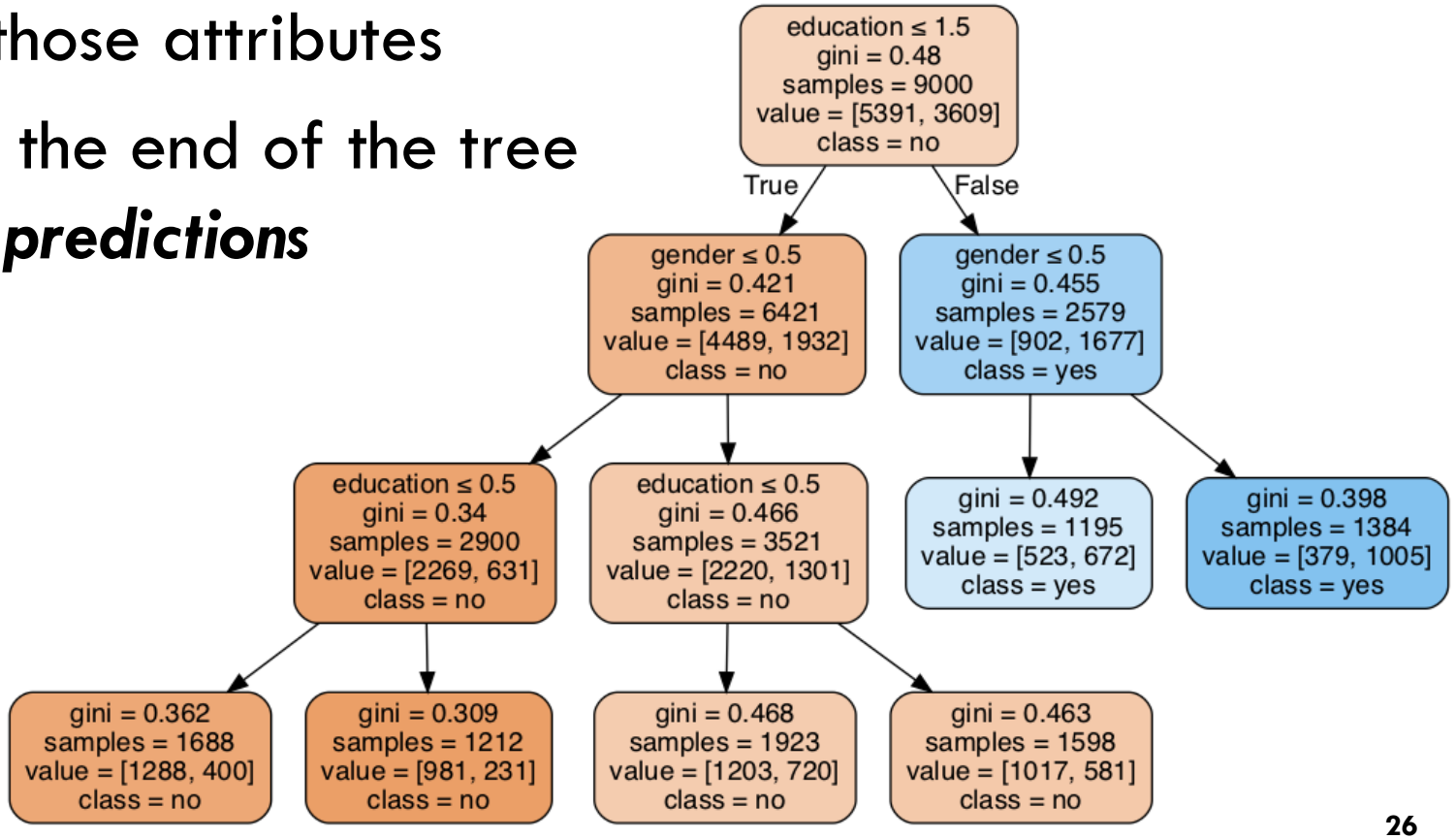
Decision Trees (CART)

- **Decision Tree:** classifier that looks at combinations of attribute values to make a prediction



Decision Trees (CART)

- **Nodes** in the tree (big squares) represent ***attributes***
- **Branches** in the tree (lines between nodes) represent ***values*** of those attributes
- **Leaves** at the end of the tree represent ***predictions***



When to Use Decision Trees

- Generally for **classification** problems
  - Exception: CART for regression
- When we want to be able to **understand what the machine learned** (transparency)
  - More trust in predictions, new human knowledge
- When there are **disjunctive rules** generating labels
  - Respondent when Educ. = College AND Landline = Yes
- When the **attributes are categorical** (or numeric)
- When there might be **measurement error**
- When there might be **missing** data

Learning a Tree

- To learn a tree from a set of data *S* (using CART)
  - Else find the attribute *A* that best informs the decision tree on the current data *S*
    - Create a node for attribute *A* with two branches
    - Split the data *S* into two sets S_L and S_R, one for each branch
    - Repeat the process for each each new data set

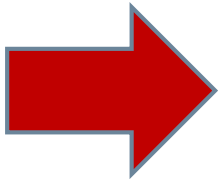
Picking the Best Attributes

- How do we determine the "best" attribute?
  - The one that **reduces the uncertainty** in the label the most!
- Best case scenario: all instances in S_L have same label, also all instances in S_R have the same label
  - Then there is **no more uncertainty** about which label to predict (and the next nodes will be leaf nodes)

Data *S*

5 instances with Label ***No***

5 instances with Label ***Yes***



Data S_L

5 instances with Label ***No***

0 instances with Label ***Yes***

Data S_R

0 instances with Label ***No***

5 instances with Label ***Yes***

Picking the Best Attributes

- How do we measure "uncertainty" in data *S*?
  - For **classification**, we consider the sum of the variances in the proportions of the labels
  - Also called the Gini index

$$Gini(S) = \sum_{labels\ y} P_y(S) * (1 - P_y(S)) \qquad P_y(S) = \frac{\#\ instances\ in\ S\ with\ label\ y}{\#\ instances\ in\ S}$$

Data *S*

5 instances with Label ***No***

5 instances with Label ***Yes***

$$P_{No}(S) = \frac{5}{10} = 0.5 \quad P_{Yes}(S) = \frac{5}{10} = 0.5$$

$$Gini(S) = \left[P_{No}(S) * \left(1 - P_{No}(S)\right)\right] + \left[P_{Yes}(S) * \left(1 - P_{Yes}(S)\right)\right]$$
$$= [0.5 * 0.5] + [0.5 * 0.5] = \mathbf{0.5}$$

Attribute Importance

- The higher up an attribute is in a tree, the more "important" it is for prediction (i.e., its predictive power)

- Looking at the hierarchy of a tree shows how relevant each attribute is in predicting the labels (might depend on combinations of features)
  - Combinations come from the branches already taken
    - Top attribute = its value AND second attribute = its value AND …

Special Features

- If some attributes have missing values in some instances:
  - CART automatically finds the best alternative attribute to use (highest correlation between values) to decide whether that instances goes in S_L or S_R
- Prunes the tree after it is finished learning
  - We prefer smaller trees because they are easier to work with and might generalize better (Occam's Razor)
  - CART removes some nodes after it is finished if they don't improve the accuracy of predictions very much

Drawbacks to Decision Trees

- Sensitive to **overfitting**
  - Learns nuances of data used in learning that does not generalize to all data
  - Pruning only helps so much
- CART nodes always have exactly two children
  - Requires an attribute to appear several times to handle 3+ categories (increases depth of the tree)
    - Especially problematic for numeric variables (≤ splits)
- Require **retraining** when new data is available

Random Forests

"Alone we can do so little, together we can do so much." – Helen Keller

"None of us is as smart as all of us." – Ken Blanchard

- We can improve on the performance of decision trees by **not using *only one tree* at a time**
  - Instead, we create a **forest of trees** and combine the predictions of individual trees
- Helps reduce the variance in the predictions made
  - One tree might have learned some knowledge about the world
  - Many trees collectively learn more

When to Use Random Forests

- Similar to when we use decision trees
  - Classification problems with categorical (or numeric) attributes
  - When we want to understand how predictions are made
  - When we have noisy/missing data

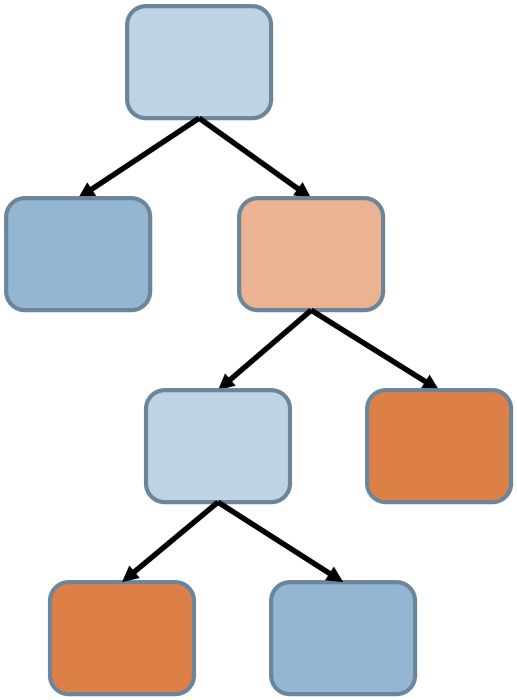
- Why forests over trees?
  - Often **better accuracy** (tradeoff time for performance)
  - **Reduces overfitting** (so don't have to throw out information)

Predicting Labels

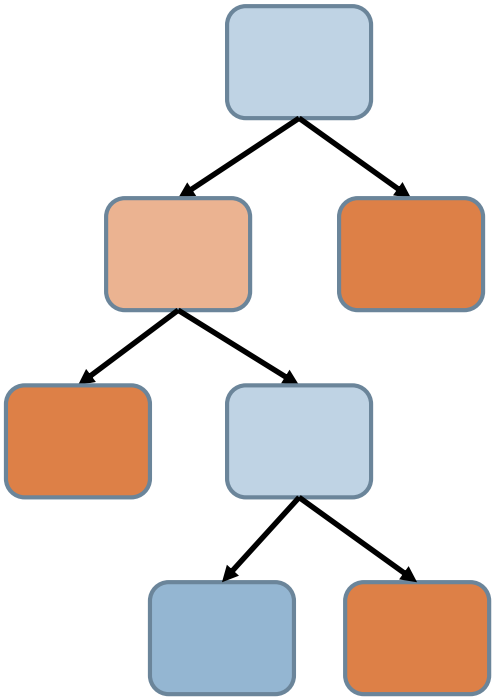
- **Classification:** Take a majority vote of the individual trees, predict the **most common label** voted

Final Prediction: Yes (2 Yes, 1 No)

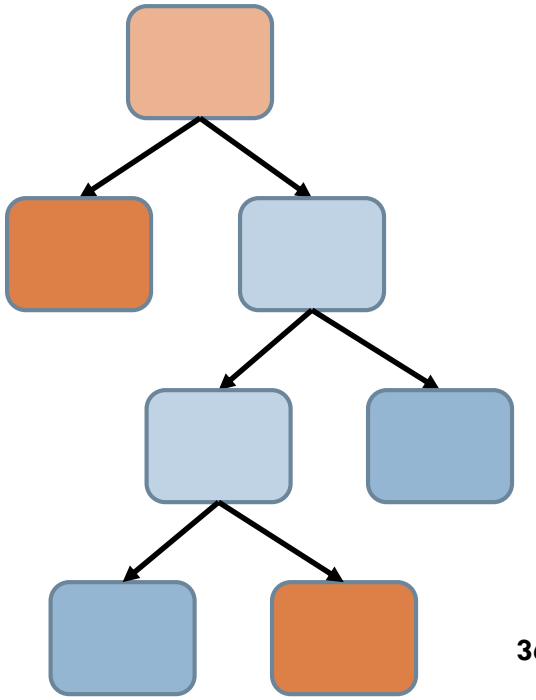
$Tree_1$ Prediction: Yes



$Tree_2$ Prediction: No



$Tree_3$ Prediction: Yes

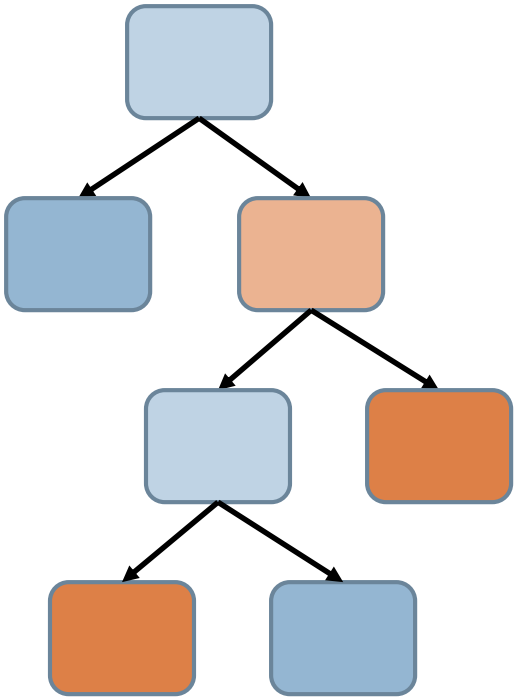


Predicting Labels

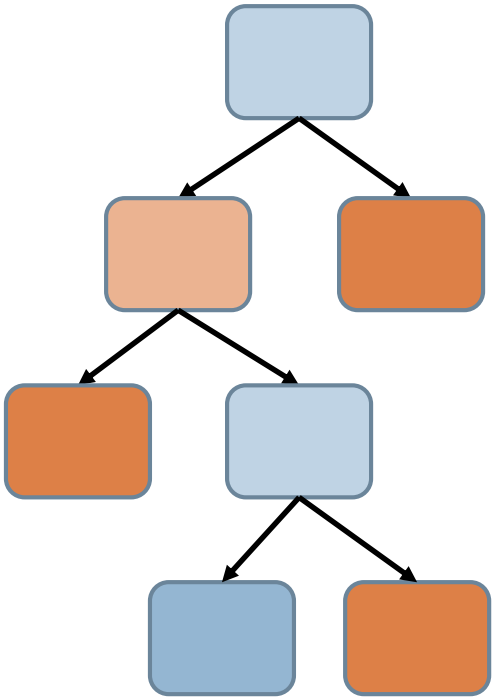
- **Regression:** Predict the **average value** predicted by the individual trees

Final Prediction: 3.1 = (3.2 + 2.1 + 4.0) / 3

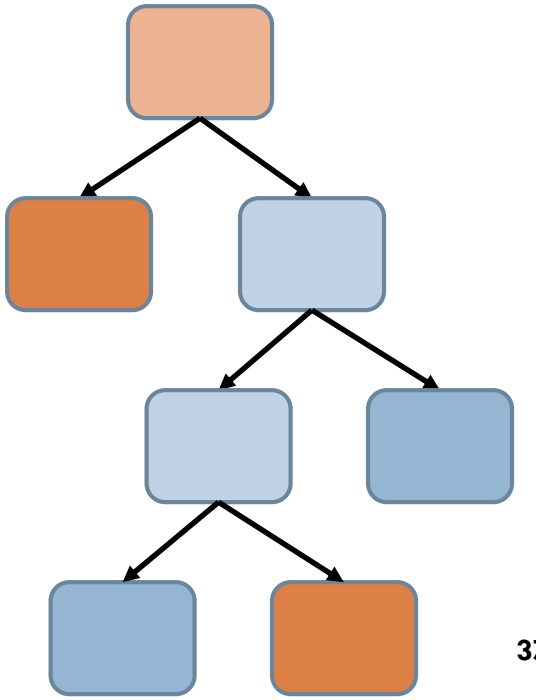
$Tree_1$ Prediction: 3.2



$Tree_2$ Prediction: 2.1



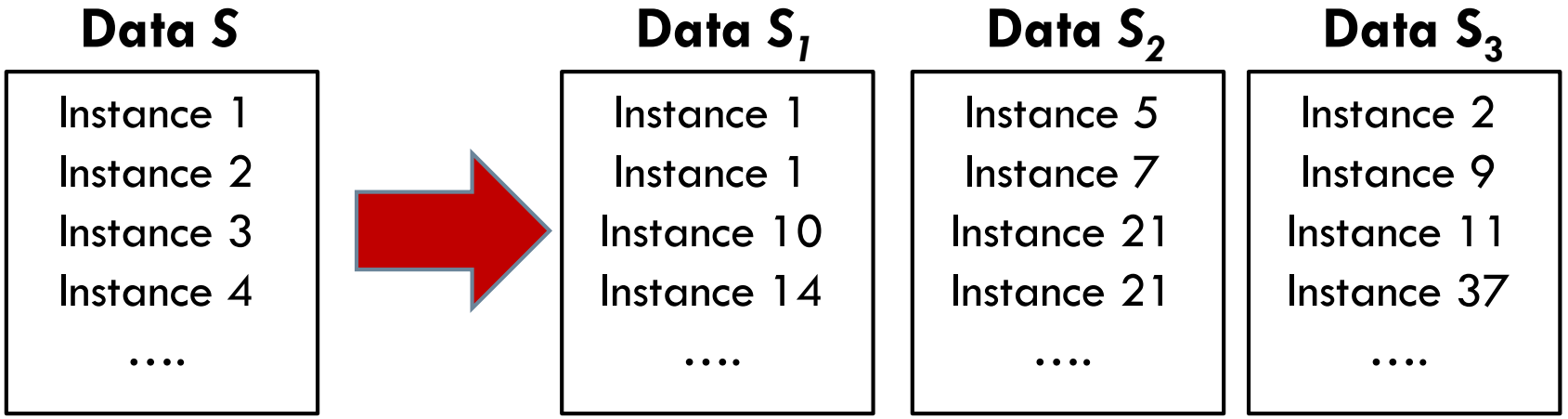
$Tree_3$ Prediction: 4.0



Learning a Forest

- How do we build multiple trees that are different?

- Give them different data sets *S*
  - Different sets of instances using **bagging** (bootstrap aggregating)
    - Randomly sample instances *with replacement*
    - Different trees will learn special knowledge about different input data



Learning a Forest

- How do we build multiple trees that are different?

- Give them different data sets S
  - Different **sets of attributes**
    - Only consider a random set of attributes (of count m) for each node in the tree
      - $m = \sqrt{\#\ attributes}$ for classification, $\frac{\#\ attributes}{3}$ for regression
    - Different trees will consider each attribute at different times (learning different combinations of values)

Attribute Importance

- Similar to decision trees, we can evaluate how important each attribute was in the prediction process
  - Average the **reduction in uncertainty** (Gini for classification, variance for regression) **each time the attribute is used** anywhere in a tree
  - Can then rank attributes based on this average uncertainty reduction

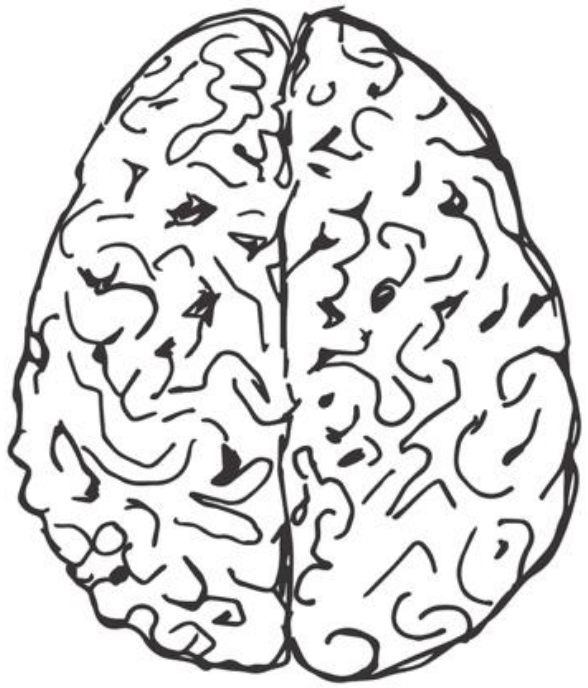
Drawbacks to Random Forests

- More **hyperparameters to tune** than decision trees
  - Number of trees, depth of trees, number of attributes, etc.
  - Requires finding a good combination (additional effort for better performance)

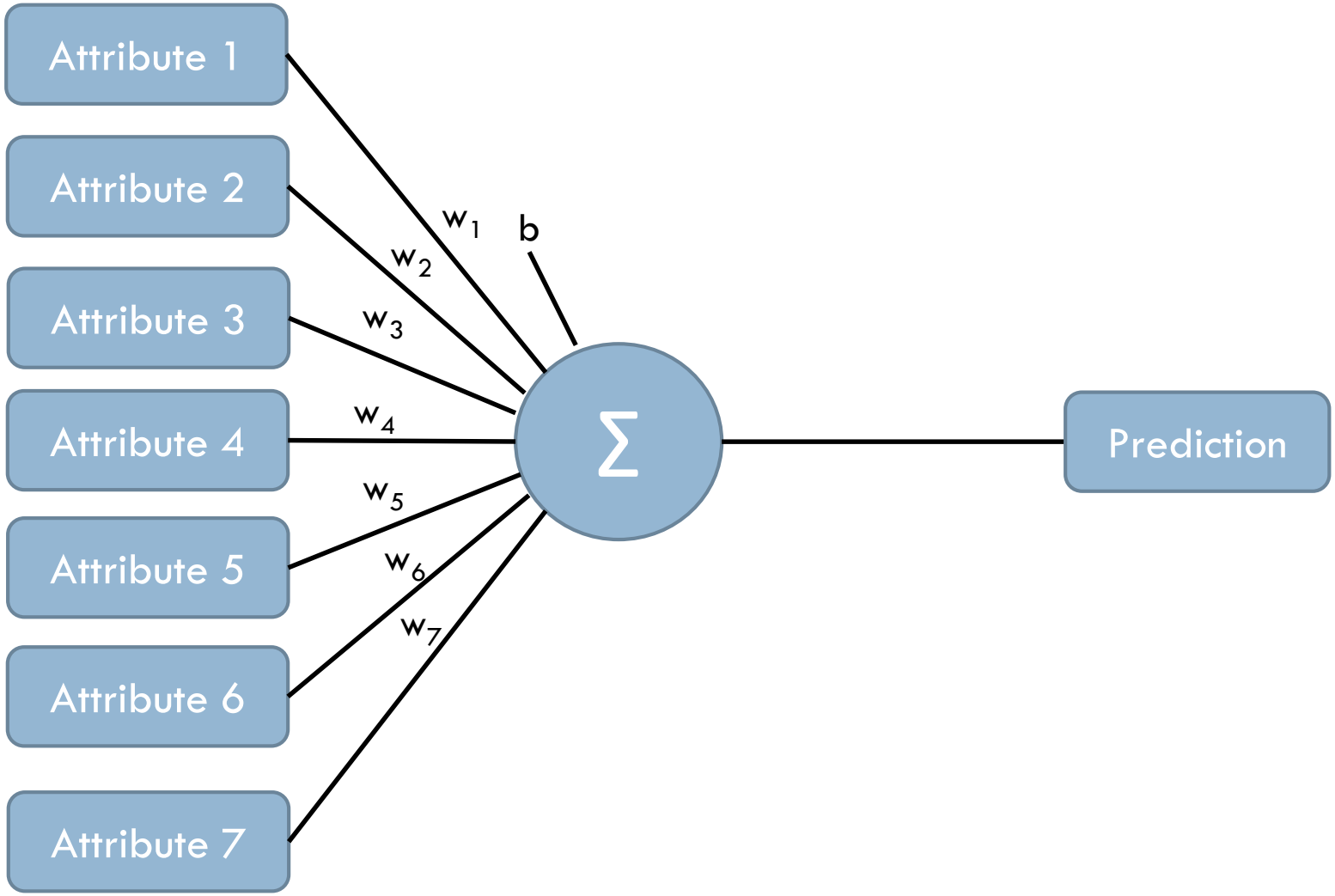
- **Increases the bias** in the predictions, but ***reduces the variance*** (compared to decision trees)
  - More bias because considering less data (instances and attributes)
  - Still usually better overall results

Neural Networks

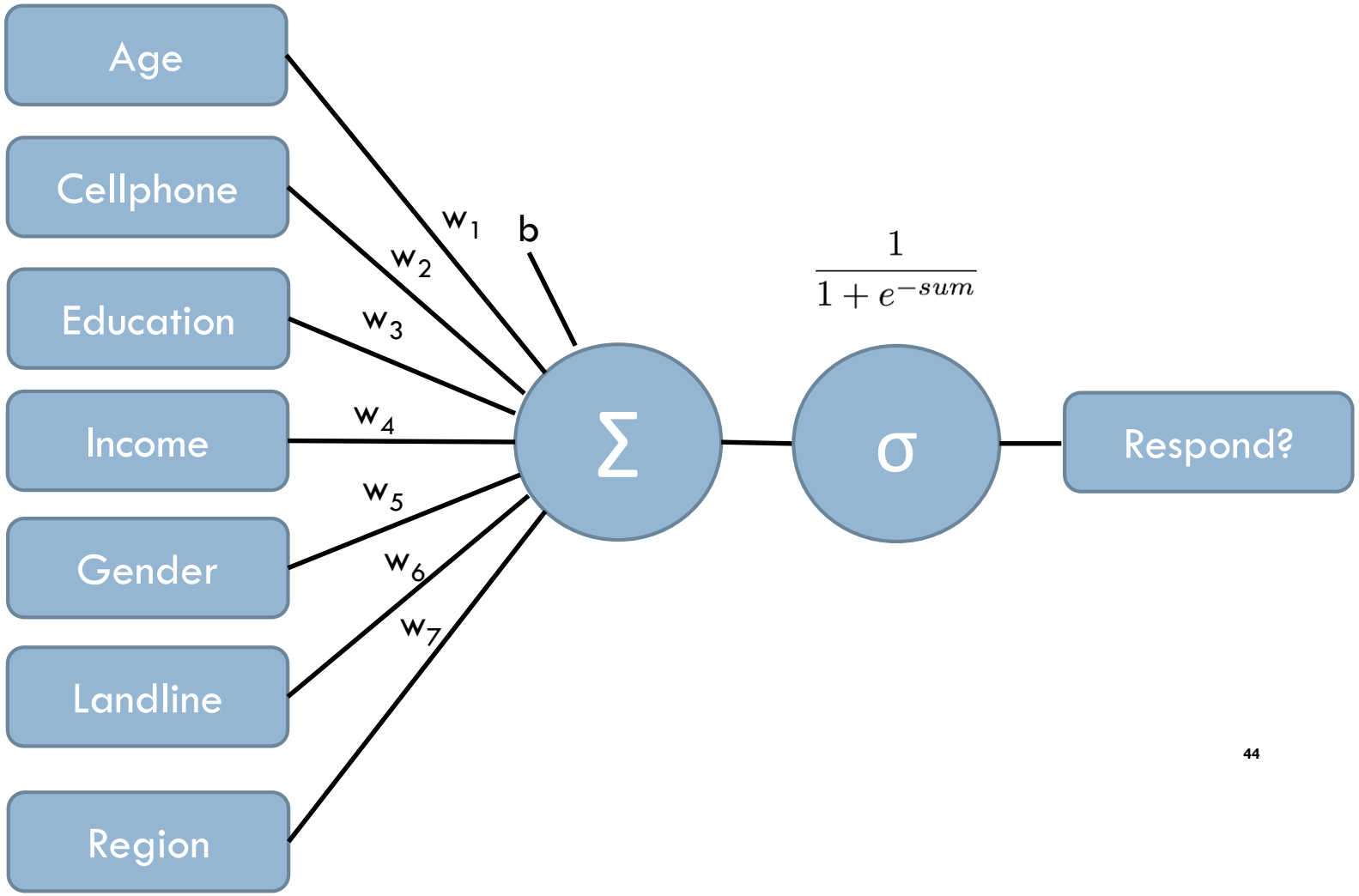
- Inspired by biology and physical underpinnings of human learning
- Human brain:
  - Composed of around 10^{10} neurons
  - Average connections per neuron: 10^4-10^5 other neurons
  - Time to recognize a scene: around 0.1 seconds
    - Only ~100 processing steps
    - Implies brain is very parallelized!



Linear Regression

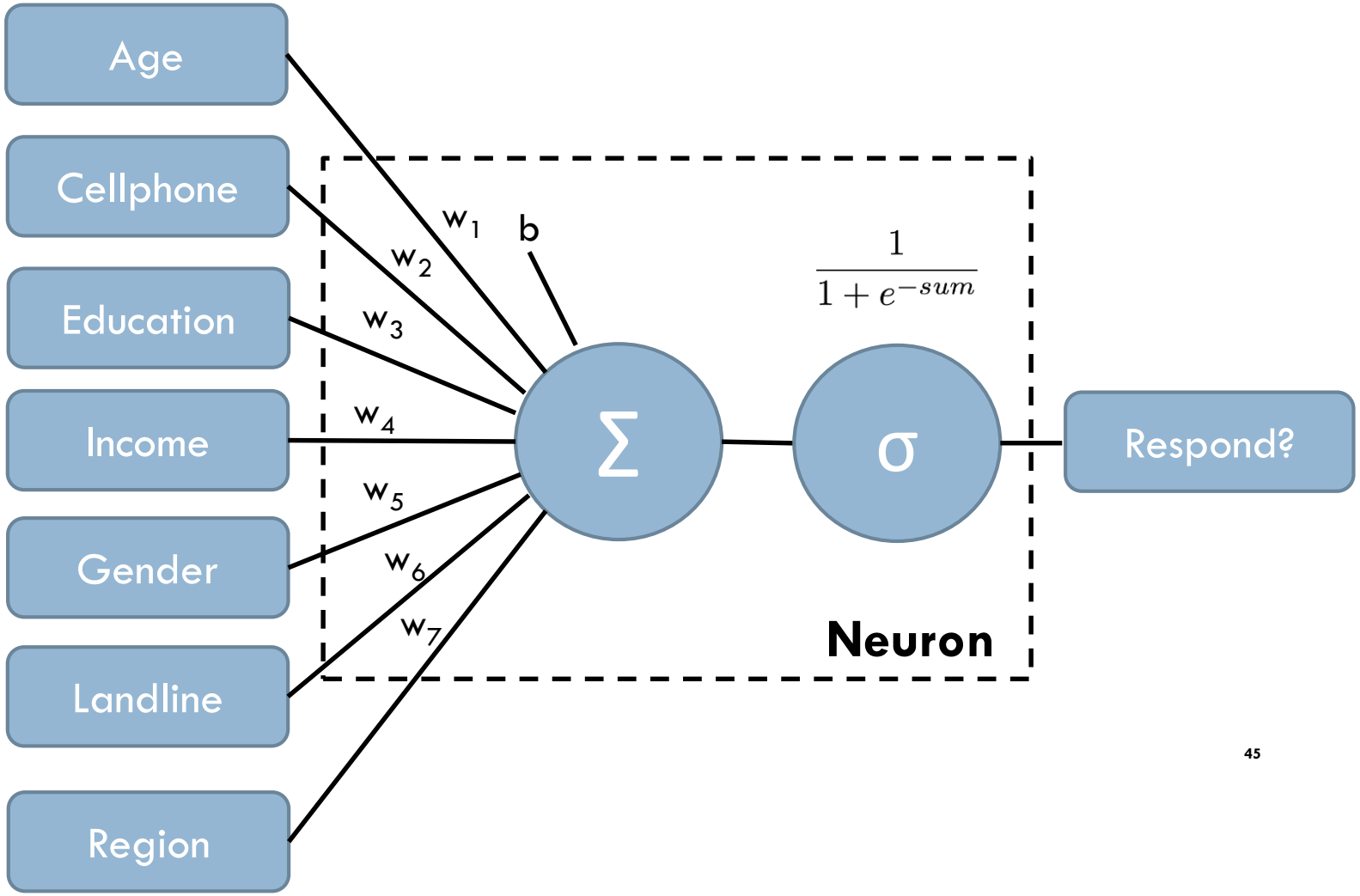


Logistic Regression



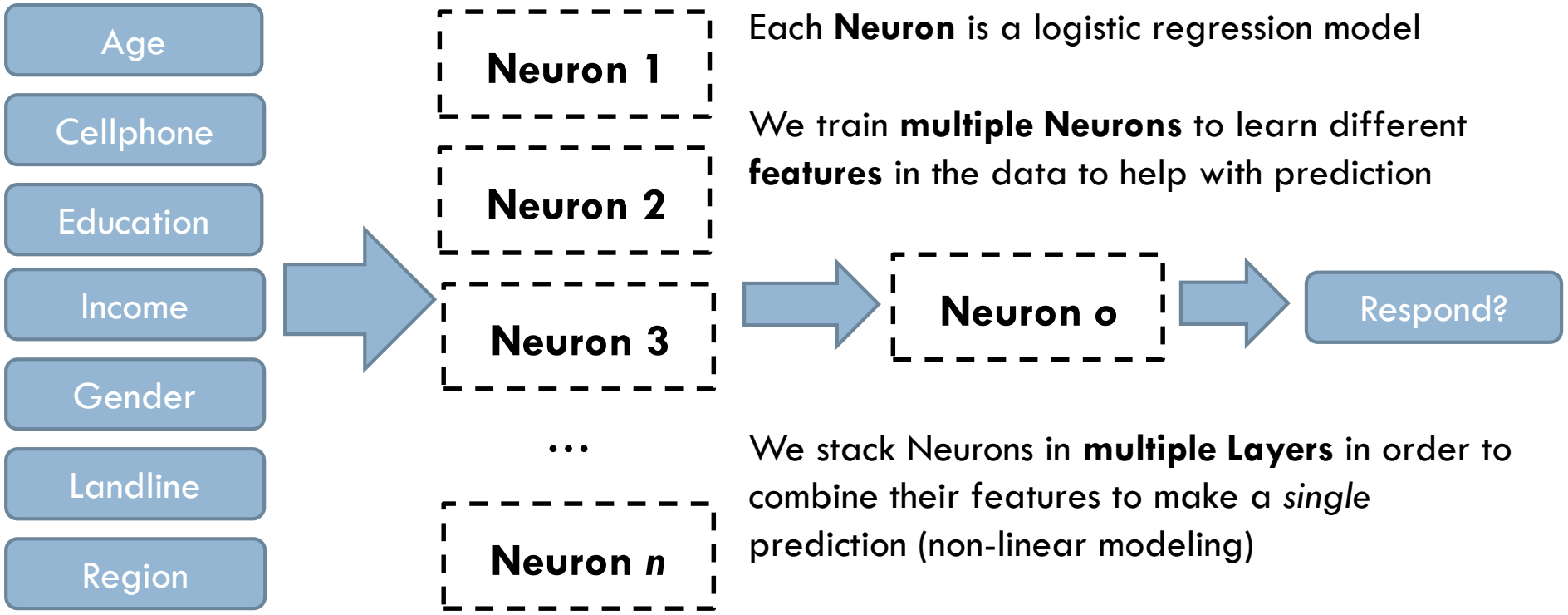
44

Logistic Regression as a Neuron



45

Neural Networks



Each **Neuron** is a logistic regression model

We train **multiple Neurons** to learn different **features** in the data to help with prediction

We stack Neurons in **multiple Layers** in order to combine their features to make a *single* prediction (non-linear modeling)

When to Use Neural Networks

- Current trend in machine learning: **for everything**

Universal Approximation Theorem (paraphrased): almost every function mapping some set of features to a label can be approximated well with a neural network

- When there are **complex relationships** between features and labels
- When many features are **numeric**
- When labels are numeric (**regression**) or there are **3+ possible labels** (or **more than one label**)

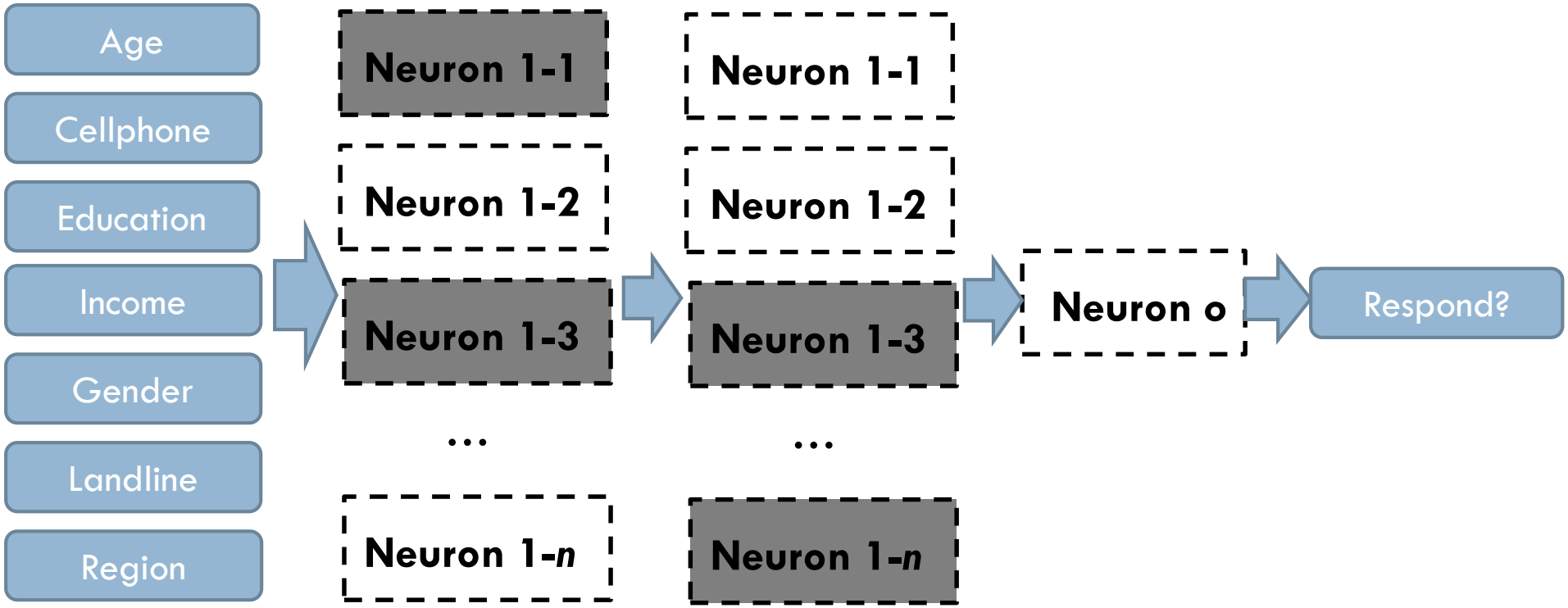
Learning a Network

- Learning = finding a good set of weights for each input into each neuron
- More difficult than logistic regression
  - Many more interconnected weights, so **not guaranteed** to find optimal set of weights
- Instead, **iteratively adjust weights** after looking at data multiple times
  - Start with random weights, then adjust towards better performance (based on blame for errors in prediction)
  - Process: backpropagation (several different algorithms of varying complexity)

Dropout

- Similar to decision trees, neural networks are prone to overfit the data used for learning
  - Learn nuances not present in all data
- Common approach: turn off a random subset of neurons each time we train
  - Spread learning across all neurons so none specialize too much (knowledge is shared throughout network)
  - Similar to how human brain learns after injury

Dropout


Age
Cellphone
Education
Income
Gender
Landline
Region
Neuron 1-1
Neuron 1-2
Neuron 1-3
...
Neuron 1-n
Neuron 1-1
Neuron 1-2
Neuron 1-3
...
Neuron 1-n
Neuron o
Respond?

Hyperparameters

- Neural networks suffer the most hyperparameters of any representation

- Four most common:
  1. Number of layers (how many abstractions)
  2. Number of neurons in each hidden layer (how many patterns)
  3. Learning rate (how aggressively to change weights)
  4. Dropout proportion (how robust to overfitting)

- Often have to do a search over all combinations
  - Difficult to optimize one at a time due to interdependence

Drawbacks to Neural Networks

- Inputs can become **very large** when working with categorical attributes (need to convert)
- Cannot handle **missing data**, and **assumes all attributes are relevant** to predicting the label
- Often the **longest time spent** learning the models
  - Speedups from special hardware (video cards)
- Generally **opaque models**
  - Cannot interpret weights as easily as in logistic regression

Comparing Representations

	General Performance	Transparent	Fast	Domain Expertise	Retraining
Decision Trees	**3**	✓	✓	✗	✗
Random Forests	**2**	✓		✗	✗
Neural Networks	**1**	✗	✗	✗	✓
Bayesian Networks	**2-3**	✓		✓	✓
SVMs	**2**		✗	✓	✗

Machine Learning Process

- Now, we are going to take a look at how the machine learning process occurs
  - What are the primary steps we take to help a computer learn a model of a given representation

Data Preprocessing

Data Preprocessing: transforming data before performing machine learning to aid in the learning process

- Common Steps (especially for neural networks):
  - Normalizing numeric attributes
  - Creating one-hot attributes for categorical attributes
  - Converting text data to n-grams
  - Filtering missing data

Feature Selection

Feature Selection: selecting the attributes that are most relevant to the machine learning process

- **Problem:** Often, we have many attributes and we aren't sure *a priori* which ones are most relevant
  - Especially when data comes from Big Data/Organic sources
- If we have established theory, this step isn't necessary
  - Unless you want to validate the theory

Feature Selection

- Common approaches to feature selection
  - **Principle Component Analysis (PCA)**: finding linear combinations of attributes that account for the most variance in the data
  - **Forward Search**: start with only one attribute, select the one that is most predictive of the label. Then add each other variable to find the most predictive pair, etc.
    - Continue until adding variables doesn't improve performance
  - **Backward Search**: start with all attributes, remove one at a time until performance is worsened (opposite of forward search!)

Splitting Up Data

- When testing machine learning to see if it works for our task, we often **do *not* learn from *all*** data

- Instead, we split it into three data sets
  - **Training Set:** data used to learn the model
  - **Validation Set:** data used to tune parameters
  - **Testing Set:** data used to evaluate the model

Splitting Up Data

- Why multiple data sets?
  - Need a separate testing set so that we can see if our model generalizes to unseen data
    - Evaluating on the training set only verifies that the model can ***memorize*** information
    - In practice, the data we make predictions for (e.g., future respondents) ***would not be available*** during training
  - Use a third (validation) set for tuning parameters
    - Do not want to use training set for the same reasons above
    - Not fair to use testing set since we wouldn't know that data when tuning the models

Splitting Up Data

- How do we split the data?
  - First, we randomize it (prevent order effects)
  - Then, we have two options:

1. Grab first T% for training, next V% for validation, and remaining (100 – T – V)% for testing
   - Common values: T = 60%, V = 20%
   - Each data point is in one and only one set
2. Use k-fold cross-validation
   - Allows each data point to rotate between each set
   - Offers additional statistical power

Splitting Up Data

k-fold Cross-Validation: splitting the data into k equally sized folds, then rotating how each fold is used

Testing	Training	Validation

- First time: first fold is validation, second is testing, rest for training
- Second time: second fold is validation, third is testing, rest is training
- Last time: last fold is validation, first is testing, rest is training

Evaluating Machine Learning

Confusion Matrix: contingency table showing how often each label was ***predicted*** for each ***actual*** label

- Classification: create a **confusion matrix**
  - Rows = actual labels
  - Columns = predicted labels
  - Values = counts of how often each pair occurred for the test (or validation) set

Actual Label \ *Predicted Label*	Completed	Breakoff
Completed	1000	50
Breakoff	100	200

Evaluating Machine Learning

- **Accuracy** tells us how well we performed across all labels
  - Accuracy = proportion of all instances correctly classified
    = (sum on diagonal) / (sum of all cells)

Accuracy = (1000 + 200) / (1000 + 50 + 100 + 200)
= 1200 / 1350 = 0.8888 = **88.88%**

Actual Label		*Predicted Label*	
		Completed	**Breakoff**
	Completed	1000	50
	Breakoff	100	200

Evaluating Machine Learning

- **Recall** tells us how well we predicted a particular label (finer grained inspection than accuracy)
  - Recall_y = proportion of instances with label y correctly predicted
    = (cell YY) / (sum of row Y)

$\text{Recall}_{\text{Breakoff}} = 200 / (200 + 100) = 0.6667 = \mathbf{66.67\%}$

Actual Label		*Predicted Label*	
		Completed	**Breakoff**
	Completed	1000	50
	Breakoff	100	200

Evaluating Machine Learning

- If we have only two labels, then:
  - Recall for the "positive" label is called **Sensitivity**
    - Also True Positive Rate
  - Recall for the "negative" label is called **Specificity**
    - Also True Negative Rate
- Example: When predicting breakoff vs. completed, breakoff is "positive" label since we are trying to anticipate (and ultimately prevent) breakoff

		Predicted Label	
		Completed	**Breakoff**
Actual Label	**Completed**	1000	50
	Breakoff	100	200

Evaluating Machine Learning

- **Precision** tells us how often we were correct if we predicted a particular label
  - "Crying wolf" performance
  - Precision_y = proportion of times we predicted label y that were correct
    = (cell YY) / (sum of column Y)

$\text{Precision}_{\text{Breakoff}} = 200 / (200 + 50) = 0.8 = \mathbf{80.00\%}$

Actual Label		*Predicted Label*	
		Completed	**Breakoff**
	Completed	1000	50
	Breakoff	100	200

Evaluating Machine Learning

- If we care most about identifying all cases of something (e.g., breakoff) and can afford some false positives
  - Then we care more about maximizing recall than precision

- If false positives are expensive (e.g., adaptive design)
  - Then we might try to balance both precision and recall (or even favor precision)

Evaluating Machine Learning

- For **regression** problems, we cannot calculate a confusion matrix

- Instead, we evaluate using different performance measures. Two common ones:
  - **Mean Squared Error:** how close are the predicted values to the actual labels? (squared penalizes outliers)

$$MSE = \frac{1}{\#\ of\ instances} \sum_{i=1}^{\#\ of\ instances} (actual_i - predicted_i)^2$$

  - **R^2 Goodness of Fit:** how much of the variance in the labels were accounted for by the attributes in the model

Previewing the Future

A computer **learns** if it improves its performance on a task over time based on experience.

Adapted from Mitchell (1997)

- Emerging research in **Deep Learning** will enable more tasks to be automated/enhanced in Survey Research
  - Working with ***Image Data***
  - Working with ***Sequential Data***

Learning about Images

- Images store information in two or three dimensions:
  - Two dimensions for width and height
  - Third dimension captures color
    - 1 channel each for Red, Green, and Blue (RGB)
    - Only 2D if image is in grayscale

Learning about Images

- In images, location matters
  - Pixels near each other share more information
  - Pixels far apart are often completely unrelated

- **Implication:** if we make instances for learning where attributes are each pixel's color values
  - Then considering all attributes at the same time is inefficient
  - Also confusing if we assume all attributes are relevant

Learning about Images

- State-of-the-art in Image Machine Learning: **Convolutional Neural Networks**
  - Popular topic in deep learning
  - Best models outperform humans in deciding what is in an image

- Difference from traditional neural networks
  - Consider a sliding window over an image to only look at some pixels at a time (find objects within images)
  - Combine information from all locations of sliding window to make a prediction (still consider all pixels)

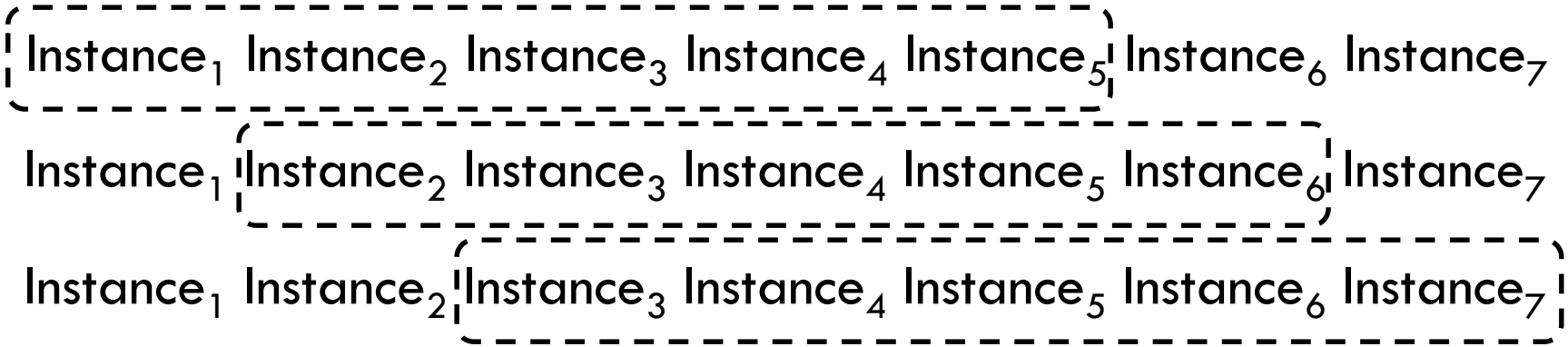
Learning over Observations

- However, life **unfolds over time**
  - Speech is a sequence of sounds
  - Traveling is a sequence of locations
  - Days are sequences of activities

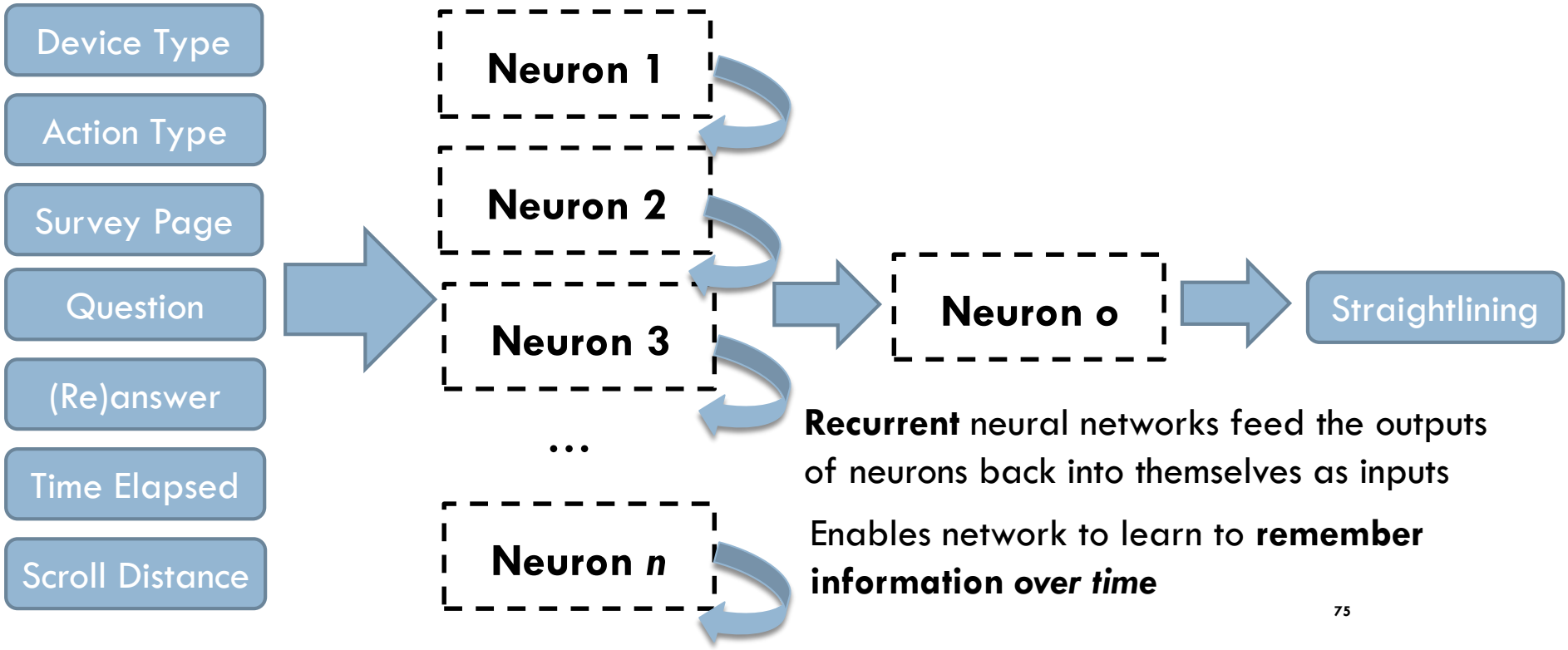
Learning over Observations

- Two common approaches to learning over observations

- **Autoregression**: create instances that combine the attributes of the last *k* observations
  - Represent a sliding window over time of size *k*



Recurrent Neural Networks



Recurrent neural networks feed the outputs of neurons back into themselves as inputs

Enables network to learn to **remember information *over time***

Learning over Observations

- Two common approaches to learning over observations

- Autoregression works with any machine learning representation/algorithm
  - RNNs are a special kind of neural network and have require more computational resources
- Autoregression assumes a fixed time window size *k*
  - RNNs learn over variable length sequences (more flexible)

- RNNs are the standard in speech data

Summary

- Machine learning has many uses for Survey Research (now and in the future)
- Three main types of learning: supervised, unsupervised, reinforcement
- Popular supervised learning approaches: decision trees, random forests, neural networks
- Process for machine learning: preprocess data, feature selection, split data, train, evaluate performance

Email: aeck [at] oberlin.edu

Website: http://cs.oberlin.edu/~aeck

OBERLIN
COLLEGE OF ARTS & SCIENCES
CONSERVATORY OF MUSIC