MMA/MMAI 869
Machine Learning and AI

Classification

Stephen Thomas

Updated: October 24, 2022

Smith SCHOOL OF BUSINESS | Queen's University

Outline

- Overview
- Some popular algorithms
  - DT, SVM, NN
- Ensembles
  - Bagging, Boosting
- Which algorithm is best?

Reminder: Machine Learning Terminology

Feature
(inputs, independent variables, X, columns)

Target or **Label**
(response, output, dependent variable, Y)

Age	Income	Married	Citizenship	Default
55	36,765	True	Canada	True
66	87,983	True	Canada	True
21	24,354	False	USA	False
24	56,654	True	Canada	False
34	98,324	False	UK	False
36	132,229	False	Germany	True
28	35,000	True	Canada	False
49	50,334	True	Canada	False

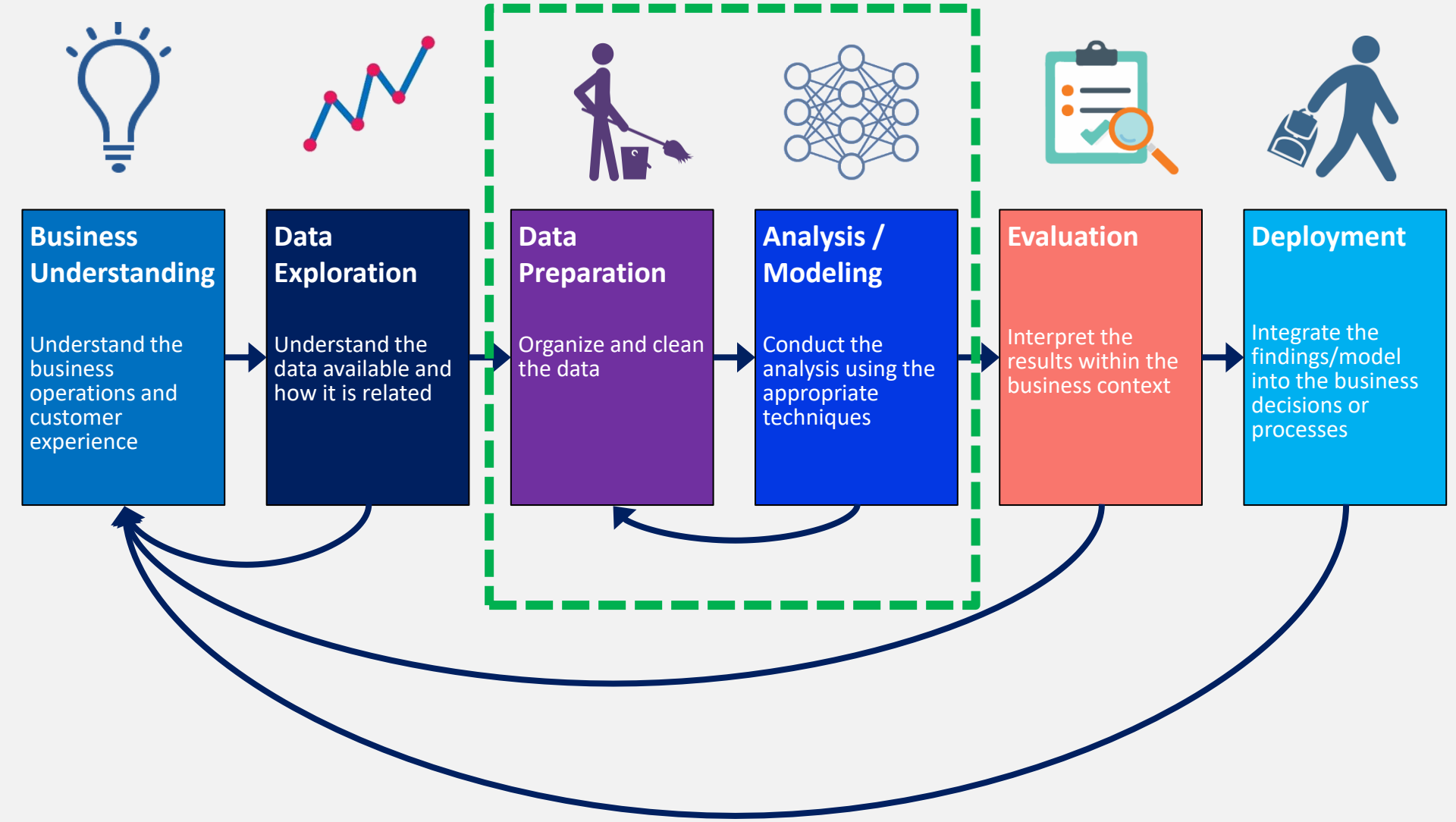
Class

Instance
(rows, cases, records)

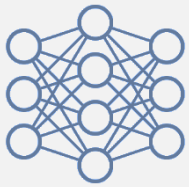
Numeric

Categorical

The Analytics Process: CRISP-DM



More Detail



Data Preparation

Organize and clean the data

Feature Engineering

Normalization
Discretization
Coding
Temporal, text, image

Feature Selection

Filter
Wrapper

Analysis / Modeling

Conduct the analysis using the appropriate techniques

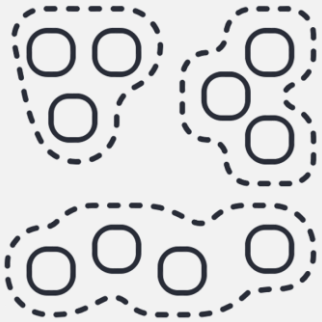
Model Training

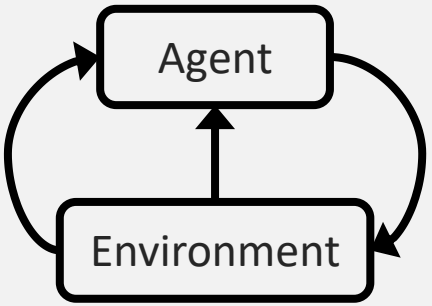
Algorithms
Ensembles
Parameter tuning

Model Selection/ Evaluation

Cross validation
A/B Testing

Three Types of Machine Learning



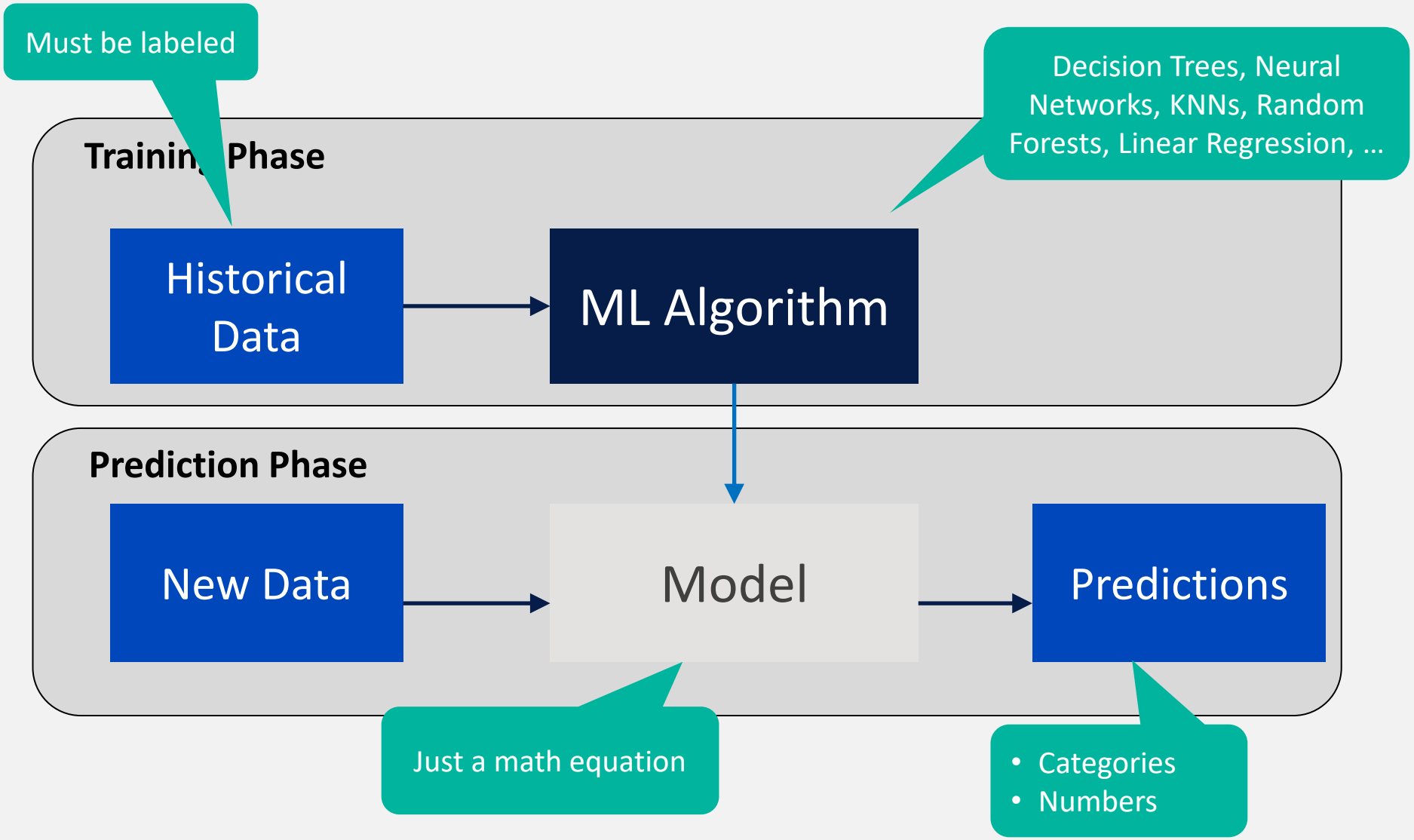


	Supervised	Unsupervised	Reinforcement
What	Predict something in the future	Find relationships	Learn through trial and error
How	Algorithm builds model from past data	Algorithms finds patterns in data	Algorithm takes actions, gets rewards
Data	Labeled	Unlabeled	None
Tasks/ Algorithms	• Classification – Decision Tree, SVM, Naïve Bayes • Regression – Linear, Polynomial, Lasso • Recommenders – Collaborative filtering, matrix decomposition	• Clustering – K-Means, DBSCAN, Hierarchical • Association rules – Apriori, Eclat, FP-Growth • Dimensionality Reduction – PCA, NMF, LDA, GDA, t-SNE	• Q-learning • SARSA • Deep Q Network

OVERVIEW

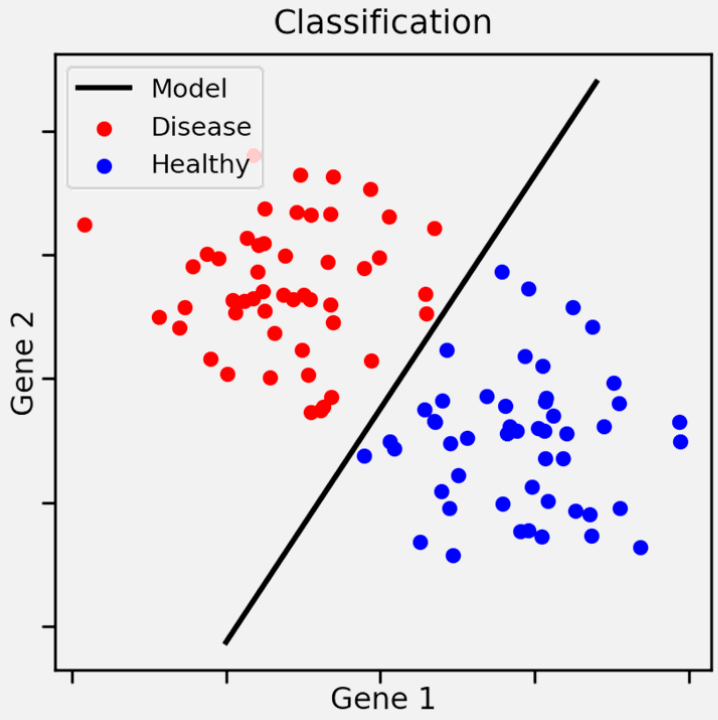
Supervised Machine Learning

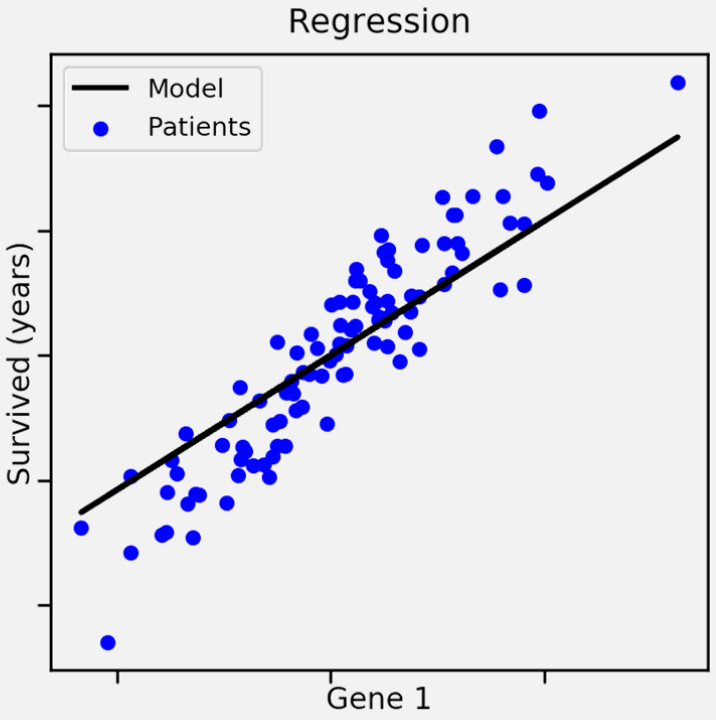
Algorithm **learns** a model that can make predictions



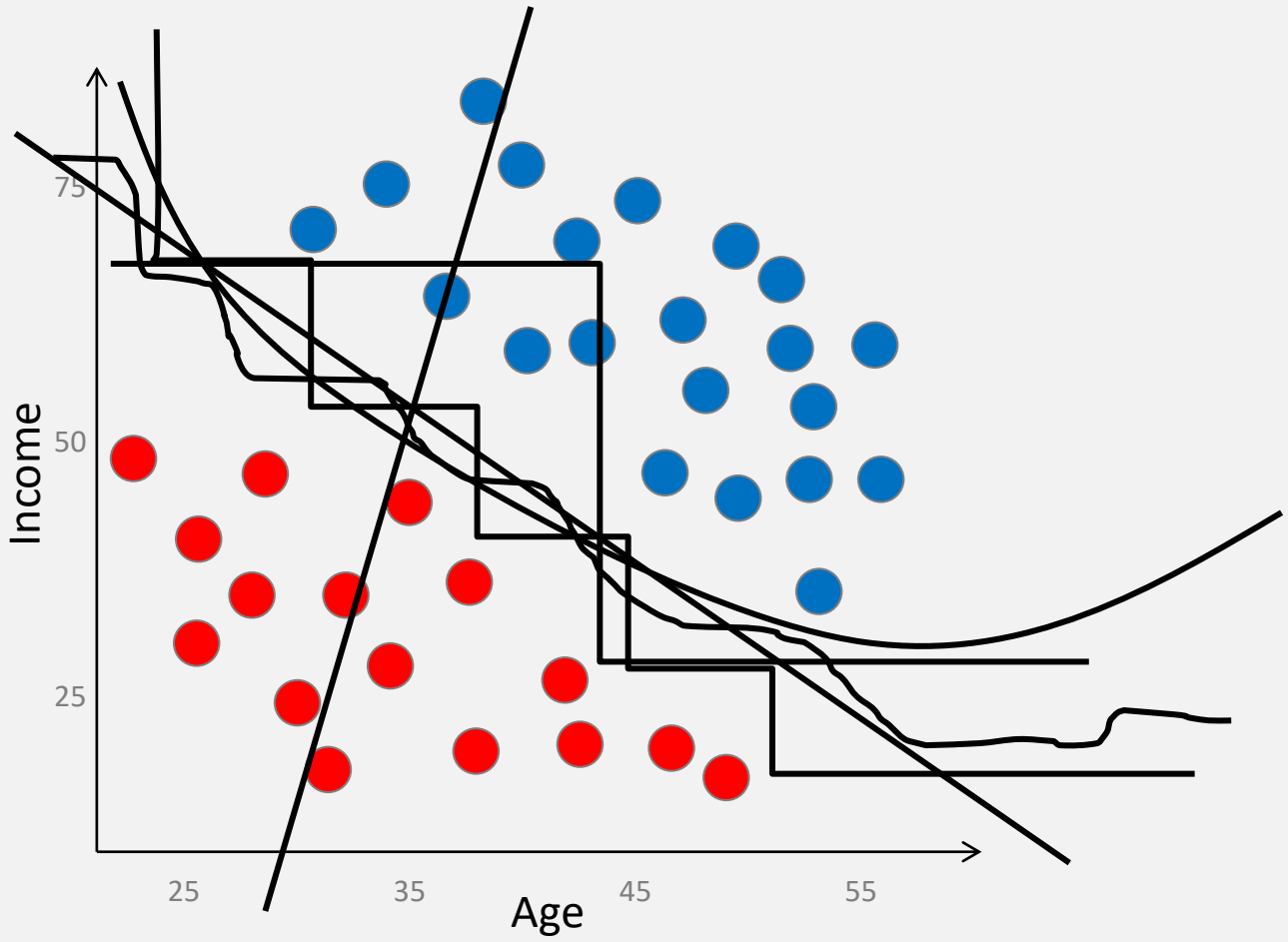
Classification *versus* Regression

- Both use past data to build models that can predict the future
- Classification builds models to predict ***categorical values*** (e.g., "no")
- Regression builds models to predict ***continuous values*** (e.g., 77.5 years)

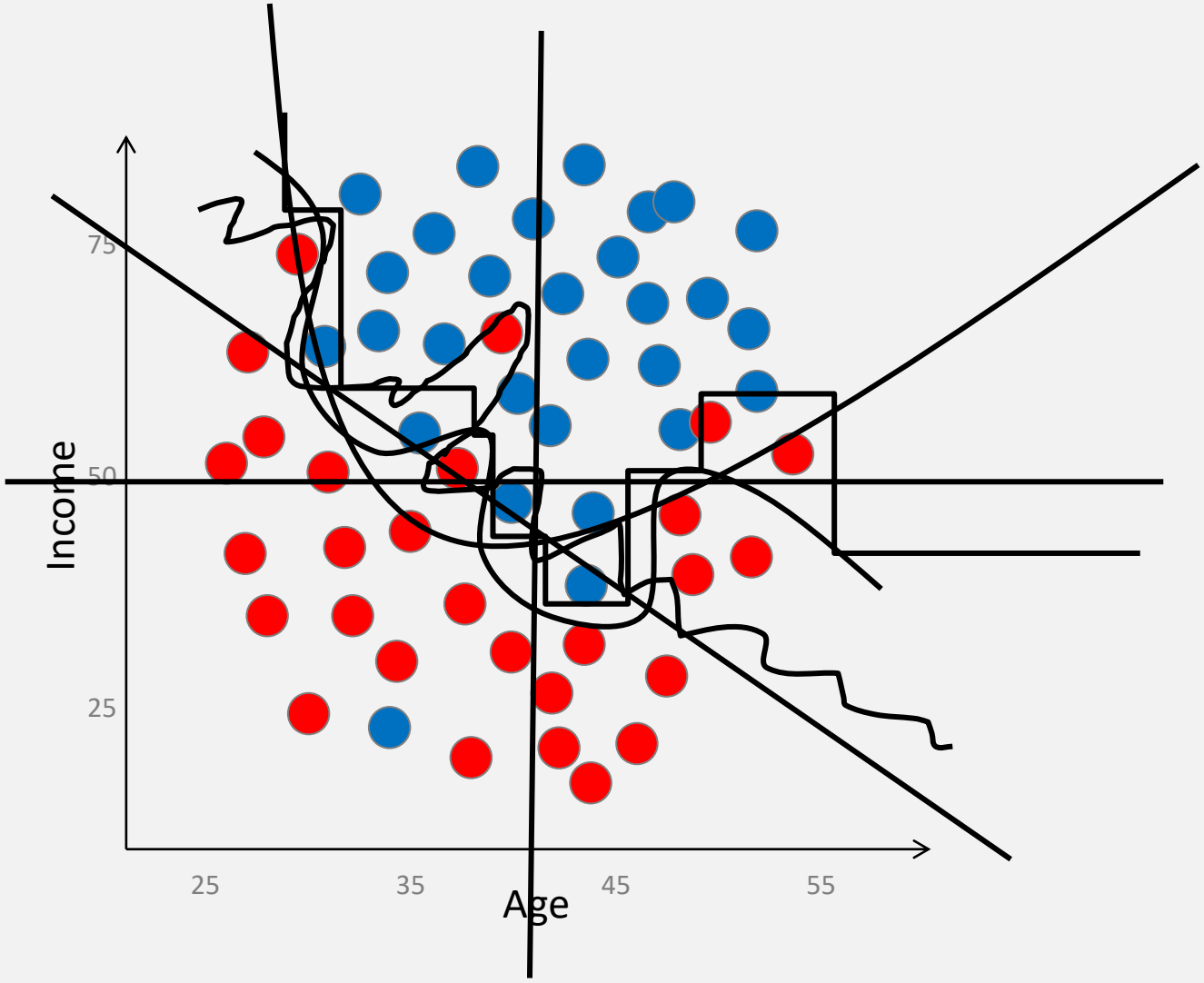




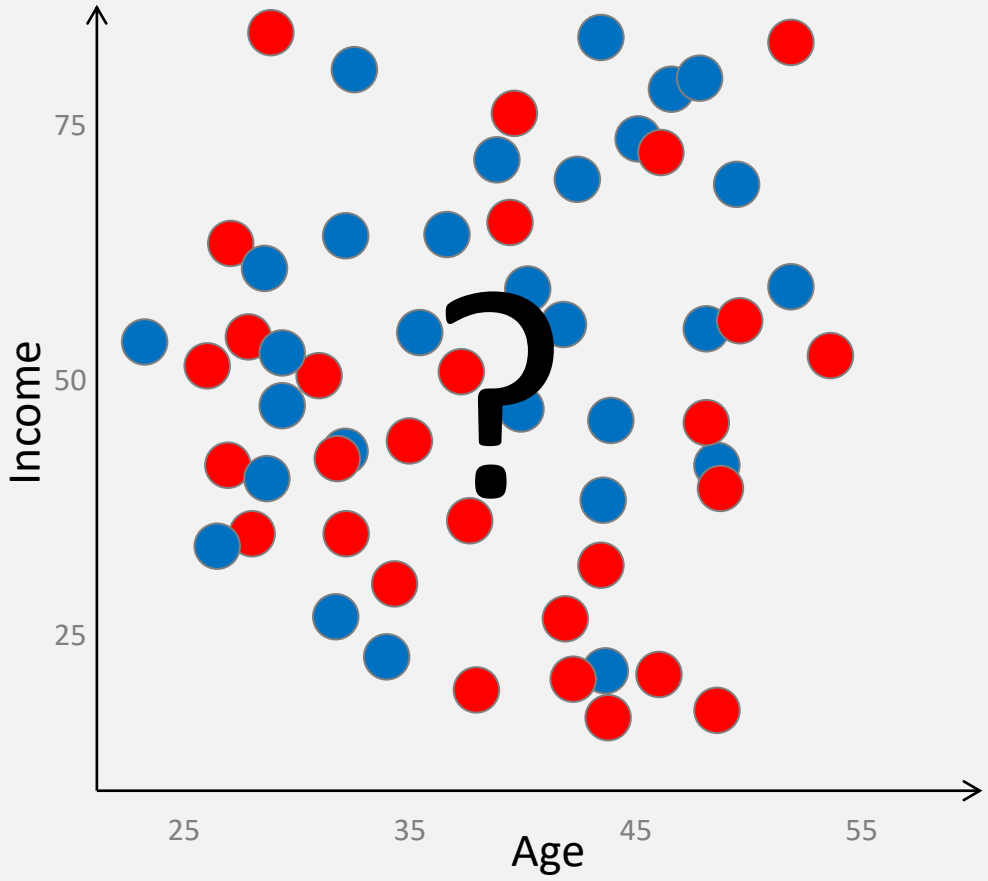
The Decision Boundary – Easy Case



The Decision Boundary – Harder Case



The Decision Boundary – Impossible Case



(Some) Business Applications

Credit Risk

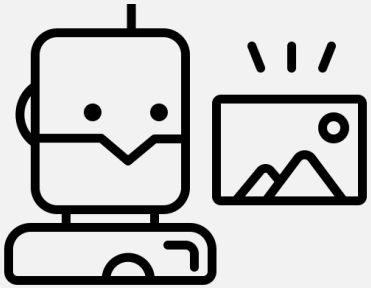
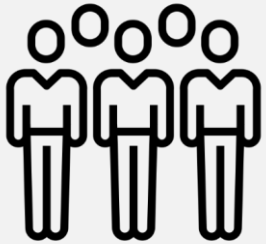


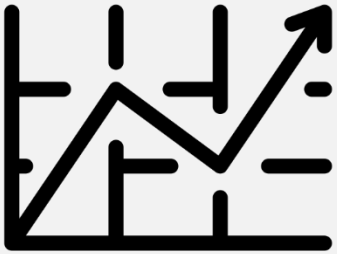
Image Recognition



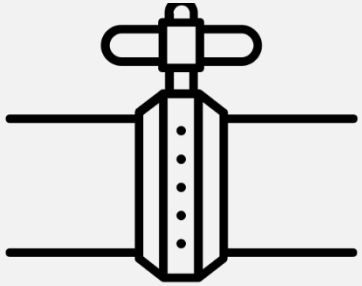
Churn Prediction

Call Center Optimization

Spam Filtering



Stock Market Prediction



Maintenance prediction

ICU Optimization

Employee Attrition

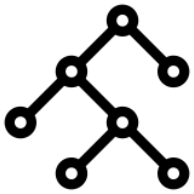
Next Best Offer

Sentiment Analysis

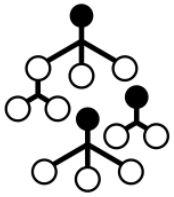
Web Ad Placement

Popular Classification Algorithms

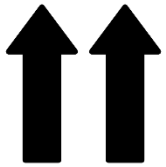
Information-based



Decision Trees



Random Forest



Boosting

Similarity-based

KNN

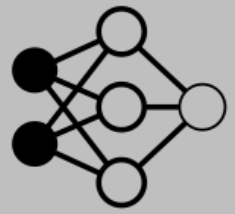
Probability-based

Naïve Bayes

Error-based

Support Vector Machine

Logistic Regression



Neural Networks

Awesome Python Packages

Package	Description	DT	RF	KNN	NB	BST	SVM	LR	NN
scikit-learn	Machine Learning in Python	✓	✓	✓	✓	✓	✓	✓	✓
mlxtend	useful tools for the day-to-day data science tasks							✓	✓
pyspark	Scalable ML library	✓	✓		✓		✓	✓	✓
statsmodels	Provides classes and functions for the estimation of many different statistical models, as well as for conducting statistical tests, and statistical data exploration							✓	
keras	High-level NN API capable of running on top of TensorFlow, CNTK, or Theano								✓
tensorflow	TensorFlow is an end-to-end open source platform for machine learning.								✓
Pycaret	Low-code ML library in Python that automates machine learning workflows.	✓	✓	✓	✓	✓	✓	✓	✓

TLDR: Which is Best?

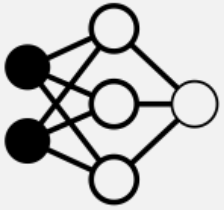
Best for **tabular data**:

Boosting

Best for **tabular data**
(if you need explainability):

Logistic Regression

Best for **unstructured data:**



Neural Networks

All Algorithms Have:

- *Hyperparameters:* settings that control the algorithm's behaviour

```
clf = DecisionTreeClassifier(max_depth=4, criterion="entropy")
```

```
clf = SVC(C=0.4, kernel="rbf", degree=2)
```

```
clf = MultiNomialNB(alpha=1.0, fit_prior=False)
```

- *Ability to underfit or overfit*
  - Depending on the chosen hyperparameter values
  - Choose the right values with *hyperparameter tuning*

FIRST, YOU TRY

Exercise #1

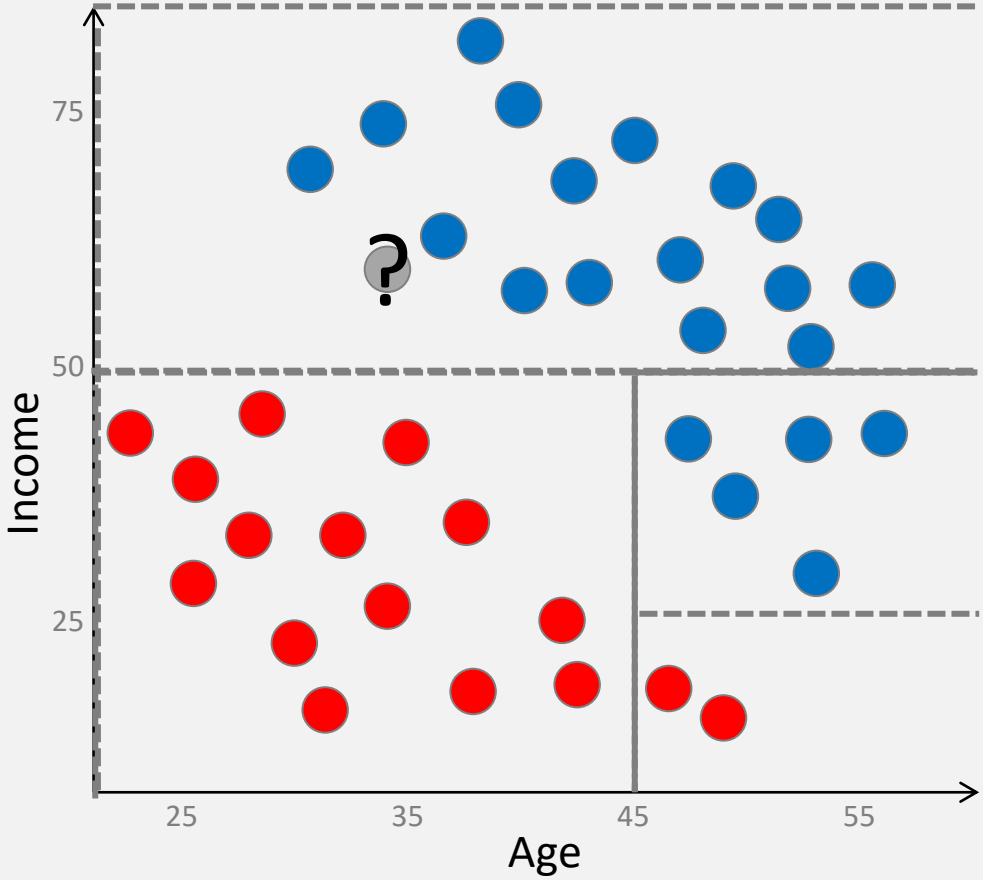
- You work at a local credit union.
- A customer (Susy) walks into your branch and applies for a $20,000 auto loan. Susy is young, unmarried, has a job, does not own a house, and has good credit rating.
- **Question**: Should you give Susy the loan?

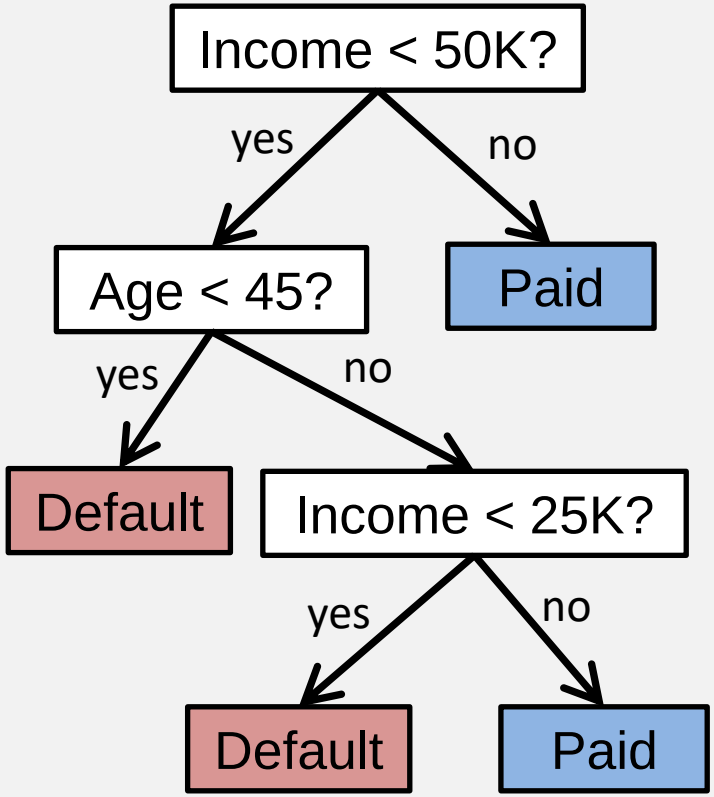
has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid

DECISION TREES

Decision Trees TLDR

- ***Training***: the algorithm builds a tree with rules
- ***Model***: the tree itself
- ***Prediction***: answer questions at each node until you reach a leaf



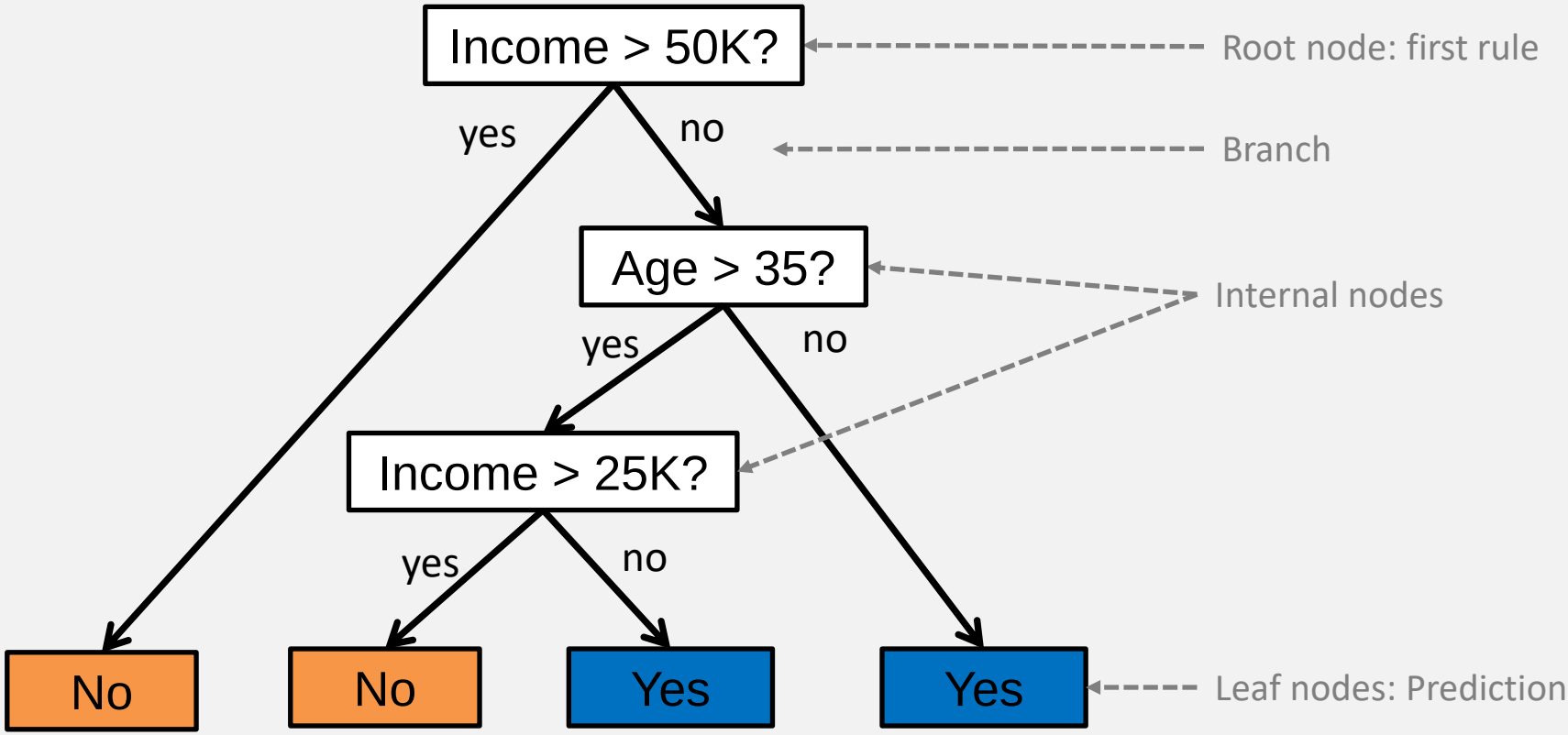


Decision Trees

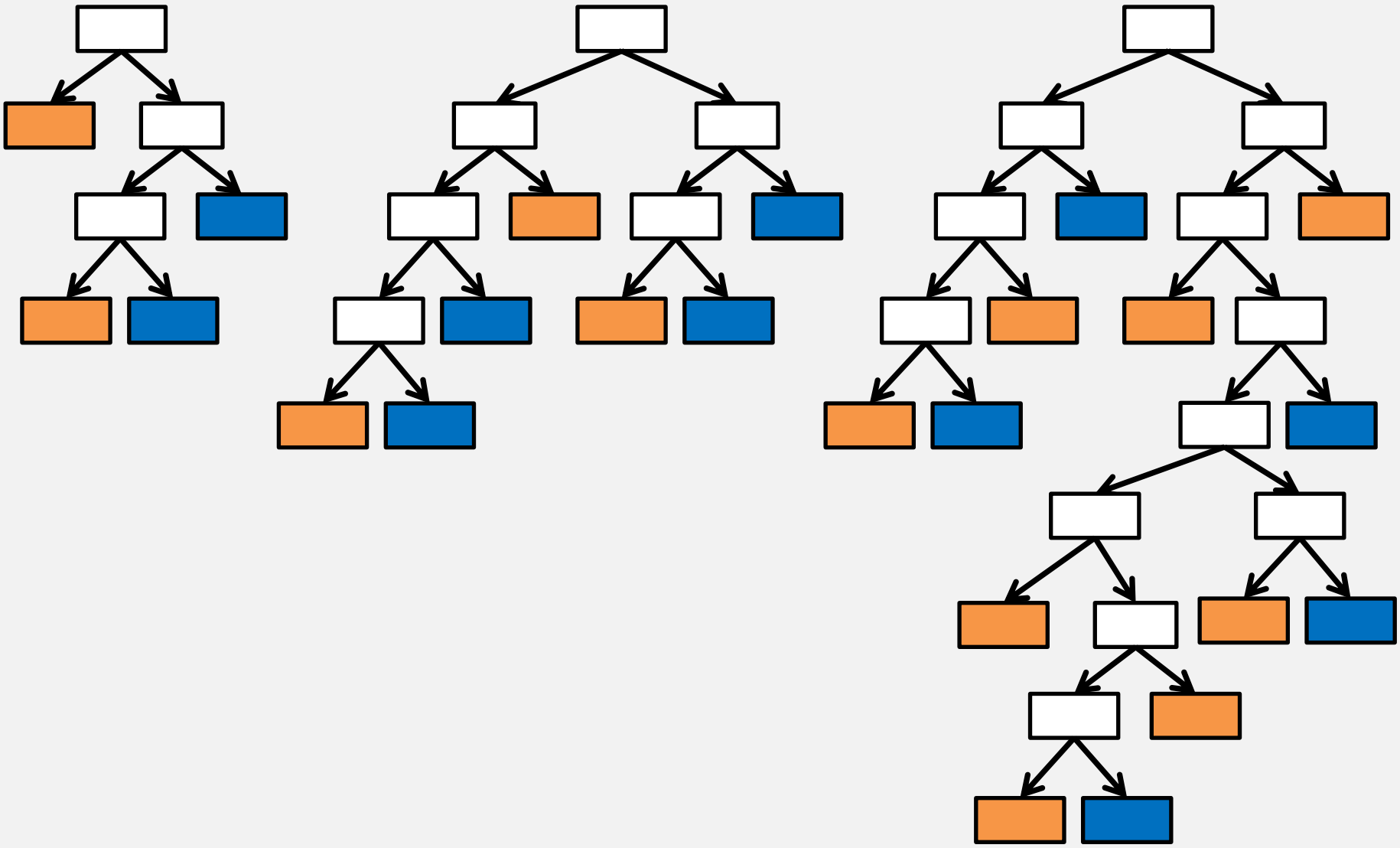
- One of the most widely-used ML algorithms
  - Accuracy is pretty good
  - Runtime is very low
  - Easy (ish?) to understand the model
  - Building block for ensembles (like Random Forests, XGBoost)
- The resulting model is a logical tree, called a ***decision tree***
- There are a few versions of the algorithm:
  - **ID3**: the original from 1986
  - **C4.5 (aka J48):** perhaps the best known and most used
    - Handles categorical and numeric data
    - Handles missing values
    - Uses pruning (error-based)
  - **C5.0:** Commercial version of C4.5
  - **CART**: Supports numeric targets

Model

- A hierarchy of rules/decisions
- Terminology:

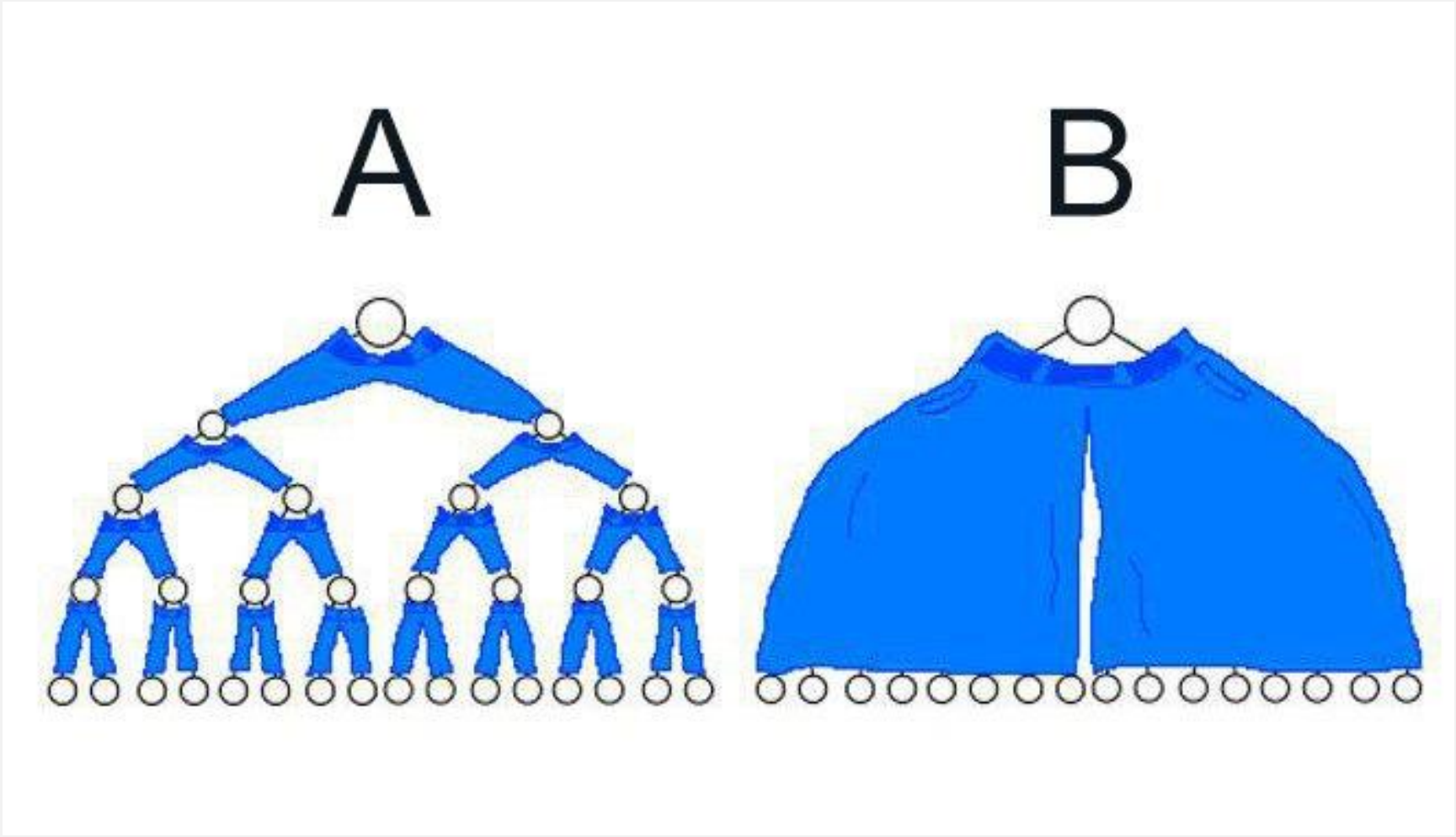


Different Shapes and Sizes

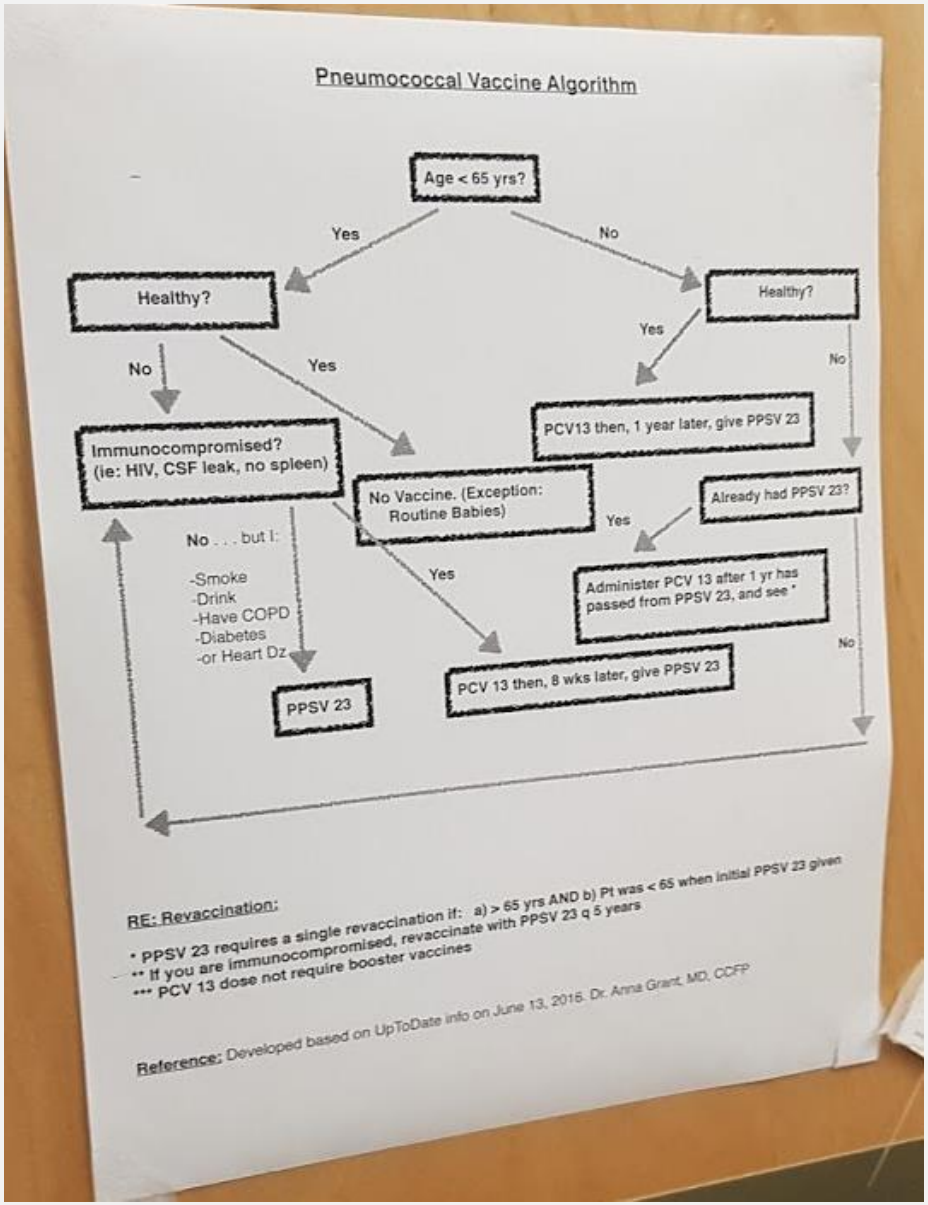


Difficult Questions

- How would a decision tree wear pants?



Decision Trees in the Wild



Training Phase

- Algorithm finds best feature for root node
  - Algorithm computes entropy of target
  - Algorithms computes information gain of each feature
  - Algorithm selects feature with highest information gain
- Algorithm splits data
- Algorithm repeats on each subset

$$E(T) = \sum_{i=1}^{n} - p_i \log_2 p_i$$

$$E(T, X) = \sum_{c \in X} P(C) E(C)$$

$$IG(T, X) = E(T) - E(T, X)$$

Wait, What Is Entropy?

- ***Entropy*** is a measure to determine how "mixed" the target is
  - Target mostly different → high entropy
  - Target mostly the same → low entropy
- Which has higher entropy?

status
paid
default
paid
paid
default
paid
default
default
paid
default
paid
paid
default
paid
default

$E(0.53, 0.47) = \mathbf{0.99}$

status
paid
default
paid
paid
default
paid
paid
default
paid
default
paid
paid
default
paid
paid

$E(0.67, 0.33) = \mathbf{0.92}$

status
paid
paid
paid
paid
paid
paid
paid
default
paid
default
paid
paid
paid
paid
paid

$E(0.87, 0.13) = \mathbf{0.56}$

status
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid
paid

$E(1.0, 0.0) = \mathbf{0.0}$

Example

- ***Calculate the entropy of the following target***

t1
Male
Female
Male
Male
Female
Female
Female
Male
Male
Female
Female
Female
Female
Female
Male

15 total
6 Male (40%)
9 Female (60%)

$$E(T) = \sum_{i=1}^{n} - p_i \log_2 p_i$$

$$E(\text{t1}) = E(6, 9) = E(0.40, 0.60) = (-0.40 \log_2 0.40) + (-0.60 \log_2 0.60) = 0.529 + 0.442 = \mathbf{0.971}$$

Exercise

- ***Calculate the entropy of the following target***

t2
Orange
Green
Green
Green
Green
Orange
Red
Red
Red
Red
Green
Green
Green
Green
Green

$$E(T) = \sum_{i=1}^{n} - p_i \log_2 p_i$$

$E(\text{t2}) =$

Running Example: KCU Dataset

has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid

Labeled training data
15 instances

We will use KCU dataset to create models by hand in the slides

Training Phase Example

- Step 1: Algorithm computes entropy of target

has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid

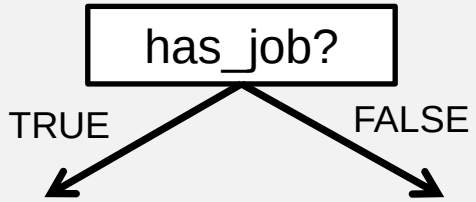
$$E(T) = \sum_{i=1}^{n} - p_i \log_2 p_i$$

$$E(\text{status}) = E(9,6)$$
$$= E(0.6, 0.4)$$
$$= -0.6 \log_2 0.6 - 0.4 \log_2 0.4$$
$$= \mathbf{0.97}$$

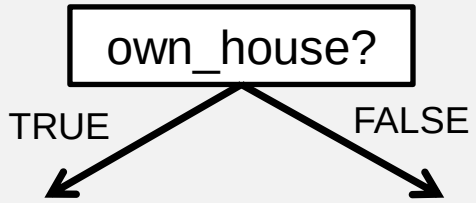
Training Phase Example

- Step 2: Algorithm computes information gain of each feature

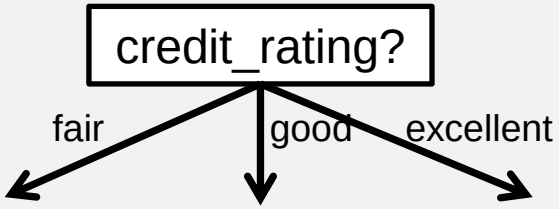
has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid



$IG(\text{status}, \text{has\_job}) = ??$

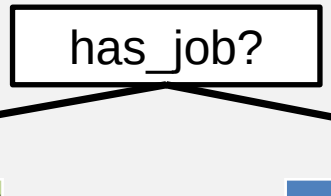


$IG(\text{status}, \text{own\_house}) = ??$



$IG(\text{status}, \text{credit\_rating}) = ??$

Compute IG of *has_job*



has_job	own_house	credit_rating	status
TRUE	FALSE	good	paid
TRUE	FALSE	excellent	paid
TRUE	FALSE	good	paid
TRUE	TRUE	fair	paid
TRUE	TRUE	good	paid

$$E(\text{status, has\_job=True}) = E(5,0)$$
$$= E(1.0\ 0.0)$$
$$= -1.0\log_2 1.0 - 0.0\log_2 0.0 = \mathbf{0.0}$$

$$\text{P(has\_job=True)} = \frac{\mathbf{5}}{\mathbf{15}}$$

has_job	own_house	credit_rating	status
FALSE	FALSE	fair	default
FALSE	FALSE	good	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	good	default
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	good	paid
FALSE	FALSE	fair	default

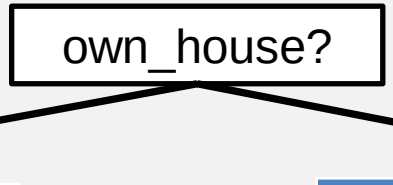
$$E(\text{status, has\_job=False}) = E(4,6)$$
$$= E(0.4\ 0.6)$$
$$= -0.4\log_2 0.4 - 0.6\log_2 0.6 = \mathbf{0.97}$$

$$\text{P(has\_job=False)} = \frac{\mathbf{10}}{\mathbf{15}}$$

$$E(\text{status, has\_job}) = P(\text{has\_job=True})E(\text{status, has\_job=True}) + P(\text{has\_job=False})E(\text{status, has\_job=False})$$
$$= \frac{5}{15} * 0.0 + \frac{10}{15} * 0.97 = \mathbf{0.65}$$

$$IG(\text{status, has\_job}) = E(\text{status}) - E(\text{status, has\_job})$$
$$= .97 - .65 = \mathbf{0.32}$$

Compute IG of *own_house*



has_job	own_house	credit_rating	status
TRUE	TRUE	fair	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	good	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
TRUE	TRUE	good	paid

$$E(\text{status, own\_house=True}) = E(6,0)$$
$$= E(1.0\ 0.0)$$
$$= -1.0\log_2 1.0 - 0.0\log_2 0.0 = \mathbf{0.0}$$

$$\text{P(own\_house=True)} = \frac{\mathbf{6}}{\mathbf{15}}$$

has_job	own_house	credit_rating	status
FALSE	FALSE	fair	default
FALSE	FALSE	good	default
FALSE	FALSE	fair	default
TRUE	FALSE	good	paid
FALSE	FALSE	fair	default
TRUE	FALSE	good	paid
TRUE	FALSE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	good	default

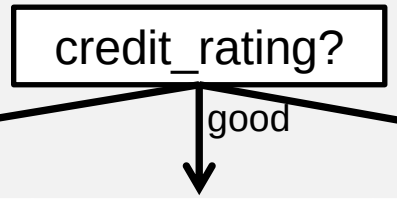
$$E(\text{status,own\_house=False}) = E(6,3)$$
$$= E(0.67\ 0.33)$$
$$= -0.67\log_2 0.67 - 0.33\log_2 0.33 = \mathbf{0.92}$$

$$\text{P(own\_house=False)} = \frac{\mathbf{9}}{\mathbf{15}}$$

$$E(\text{status, own\_house}) = P(\text{own\_house=True})E(\text{status,own\_house=True}) + P(\text{own\_house=False})E(\text{status,own\_house=False})$$
$$= \frac{6}{15} * 0.0 + \frac{9}{15} * 0.92 = \mathbf{0.55}$$

$$IG(\text{status, own\_house}) = E(\text{status}) - E(\text{status, own\_house})$$
$$= .97 - .55 = \mathbf{0.42}$$

Compute IG of *credit_rating*



has_job	own_house	credit_rating	status
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid

$E(\text{status, credit\_rating=fair}) = E(4,1)$
$= E(0.8\ 0.2)$
$= -0.8\log_2 0.8 - 0.2\log_2 0.2 = \mathbf{0.72}$

$P(\text{credit\_rating=fair}) = \frac{5}{15}$

has_job	own_house	credit_rating	status
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid

$E(\text{status, credit\_rating=good}) = E(4,2)$
$= E(0.67\ 0.33)$
$= -0.67\log_2 0.67 - 0.33\log_2 0.33 = \mathbf{0.92}$

$P(\text{credit\_rating=good}) = \frac{6}{15}$

has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid

$E(\text{status, credit\_rating=excellent}) = E(4,0)$
$= E(1.0\ 0.0)$
$= -1.0\log_2 1.0 - 0.0\log_2 0.0 = \mathbf{0.0}$

$P(\text{credit\_rating=excellent}) = \frac{4}{15}$

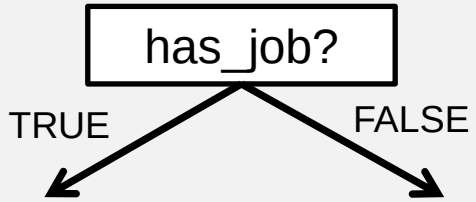
$E(\text{status, credit\_rating}) =$
$= \frac{5}{15} * 0.72 + \frac{6}{15} * 0.92 + \frac{4}{15} * 0.0 = \mathbf{0.60}$

$IG(\text{status, credit\_rating}) = E(\text{status}) - E(\text{status, credit\_rating})$
$= .97 - .60 = \mathbf{0.37}$

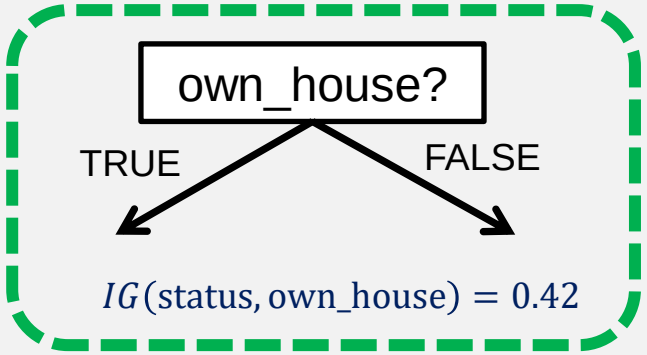
Training Phase Example

- Step 2: Algorithm computes information gain of each feature

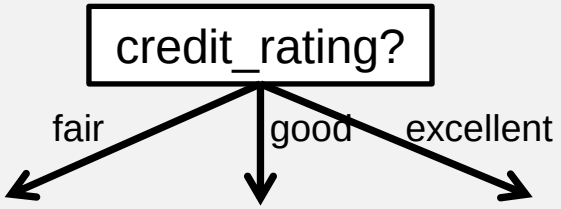
has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
TRUE	TRUE	fair	paid
FALSE	FALSE	good	default
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	FALSE	good	default
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid



$IG(\text{status}, \text{has\_job}) = 0.32$



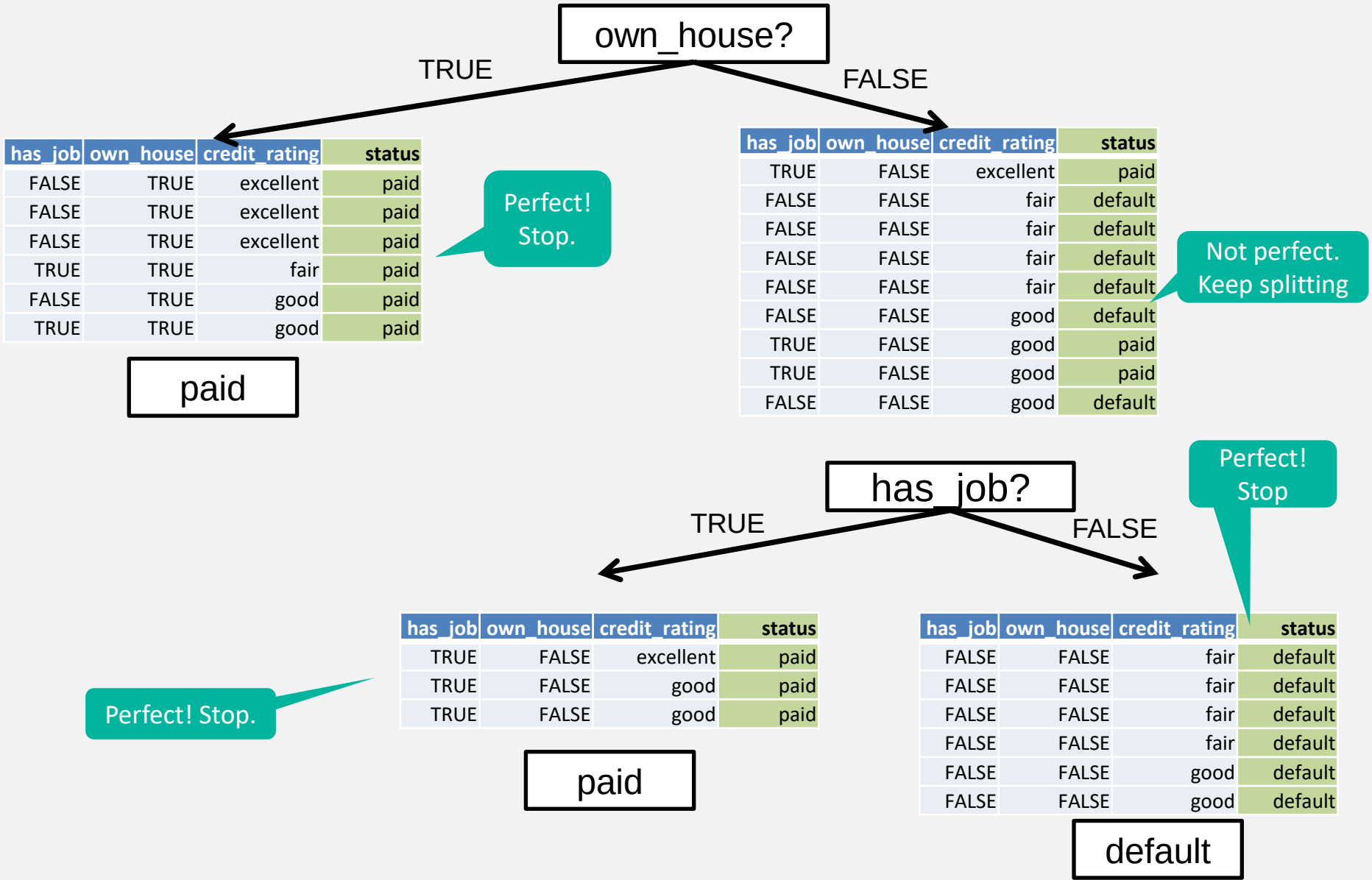
$IG(\text{status}, \text{own\_house}) = 0.42$



$IG(\text{status}, \text{credit\_rating}) = 0.37$

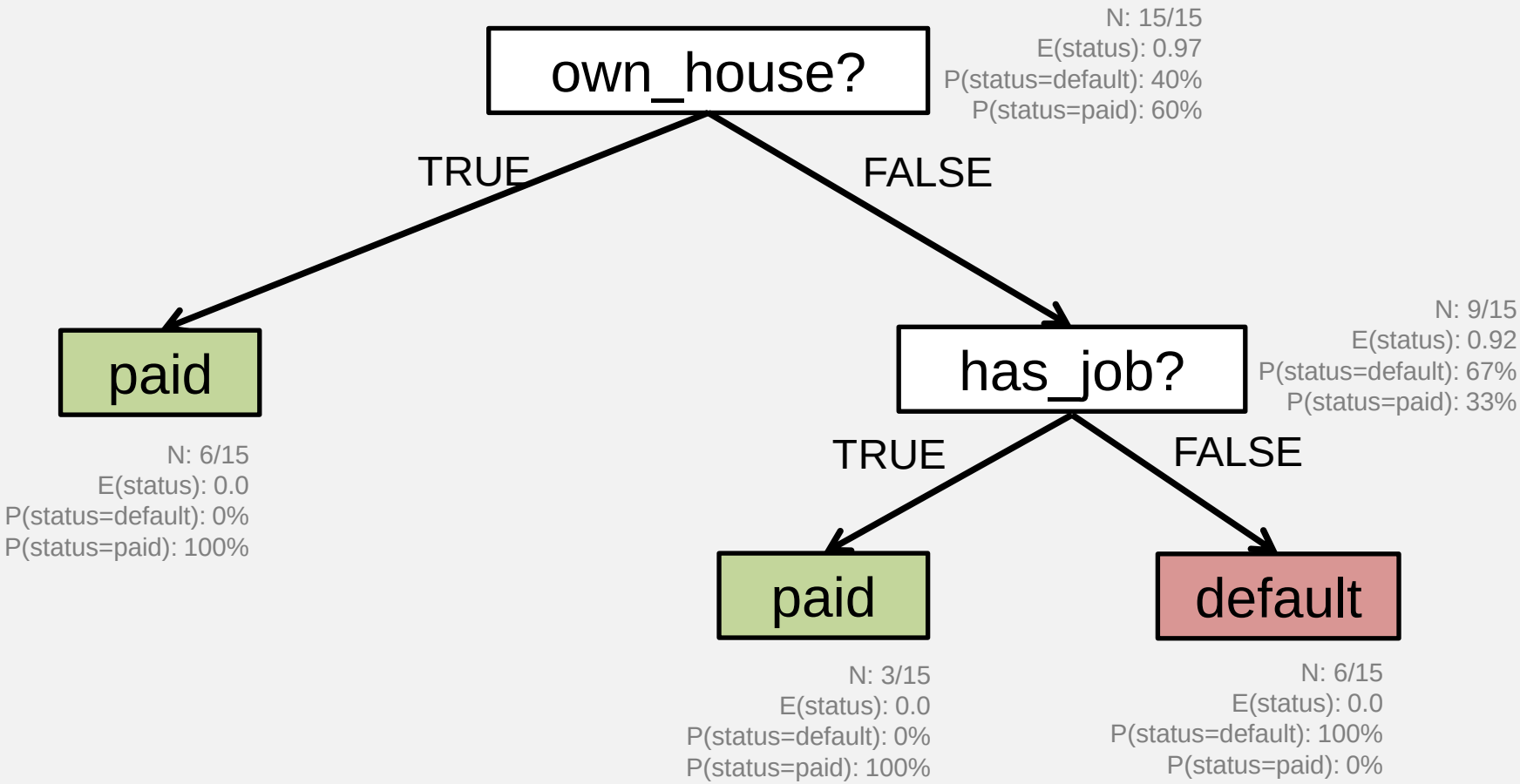
Training Phase Example

Algorithm creates root node with *own_house*



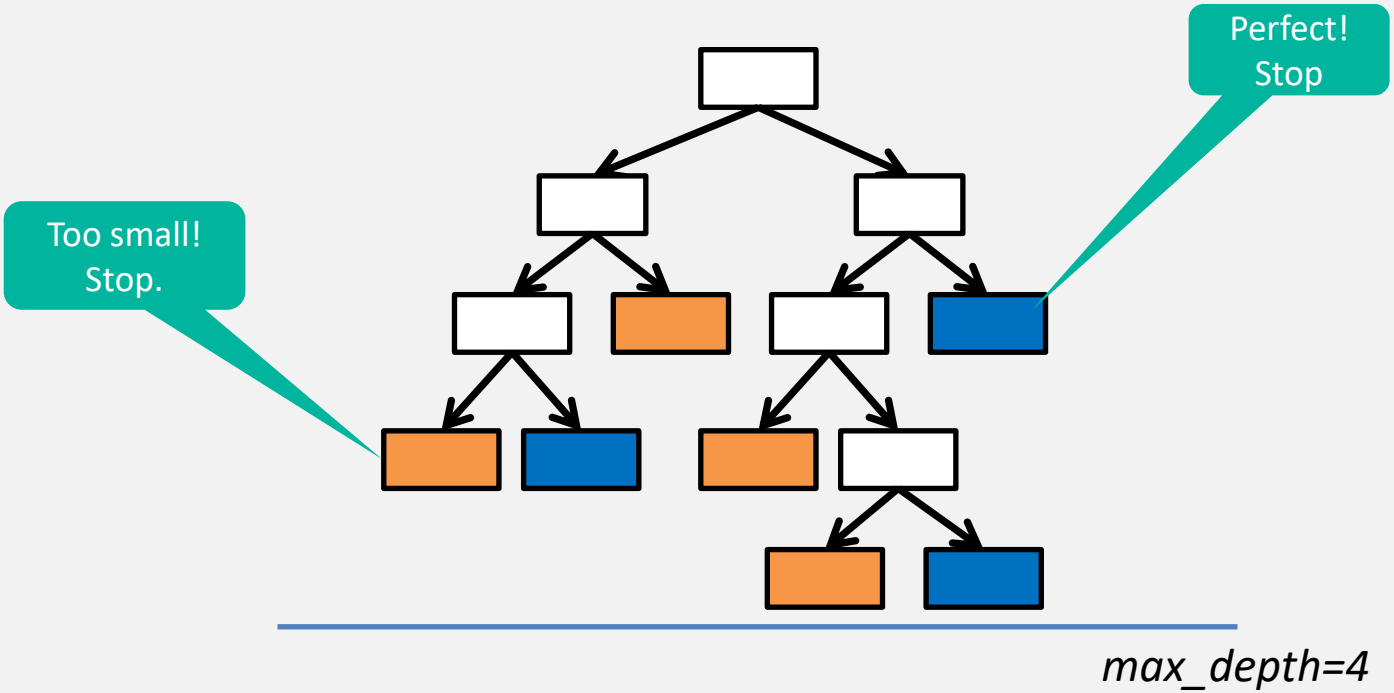
Training Phase Example

Once algorithm is done splitting, final tree is output



Stopping Conditions

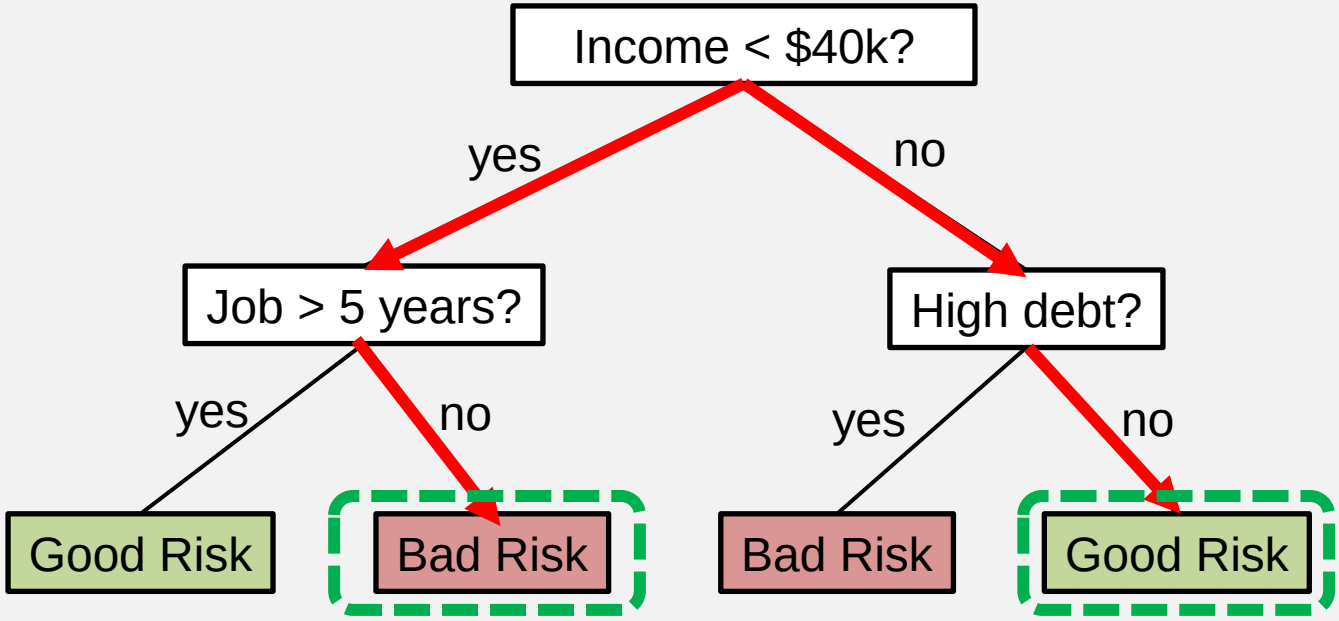
- The tree will keep growing until any of the following:
  - All instances belong to the same target (i.e., entropy=0.0)
  - *min_sample_split* hyperparameter reached
  - *max_depth* hyperparameter reached



Prediction Phase

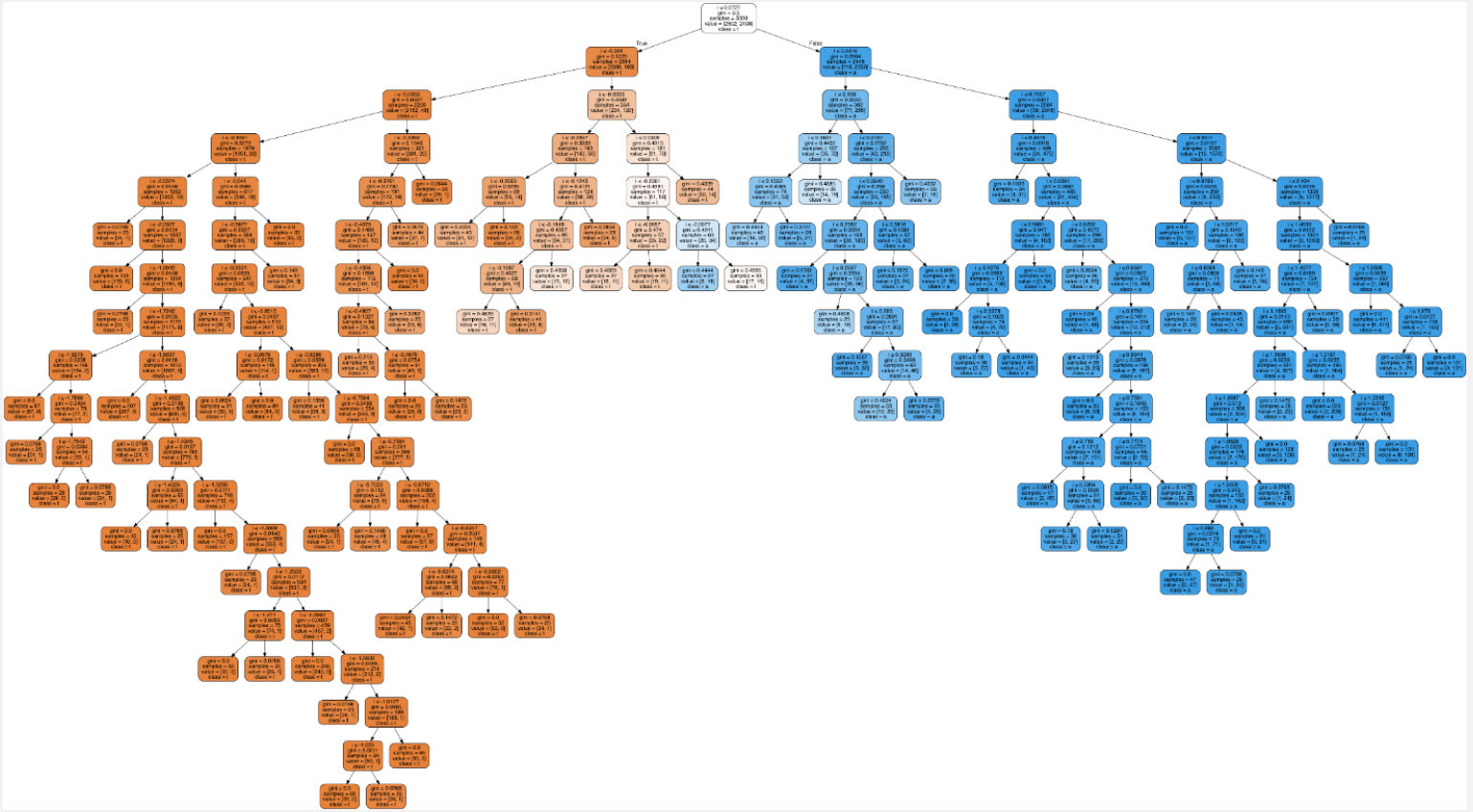
Suppose a new customer applies for a loan

- Age=55 years; Job = 8 years; Debt = Low; and Income = $55K
- Age=29 years; Job = 4 years; Debt = High; and Income = $35K



DTs: Overfitting

- A tree may overfit the training data
  - Tree too deep
  - Tree may reflect anomalies due to noise or outliers
  - Why does this happen?



DTs: Splitting Numeric Features

- DTs can handle numeric features just fine
  - Sort
  - Calculate averages between each adjacent pair
  - Consider each average a candidate split

age
50
14
68
35
19
25
63
85
21
52

sort

age
14
19
21
25
35
50
52
63
68
85

16.5

20

23

30

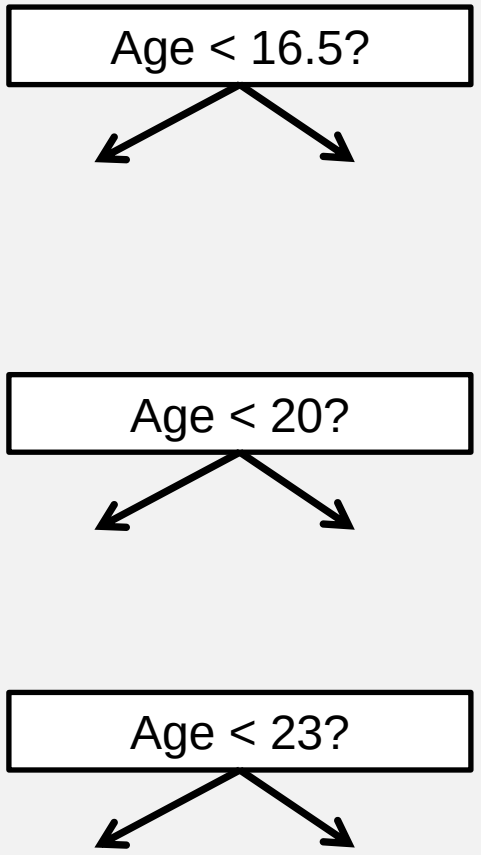
42.5

51

57.5

65.5

76.5



DT Summary

- **Training Phase**: Uses entropy/info gain to find best splits, creating tree along the way
- **Model**: A decision tree
- **Prediction Phase**: Walk the tree.
  - Start at the root node of the tree, answer questions uses instance's feature values, until you get to a leaf
- **Decision boundary**: Looks like step functions/boxes

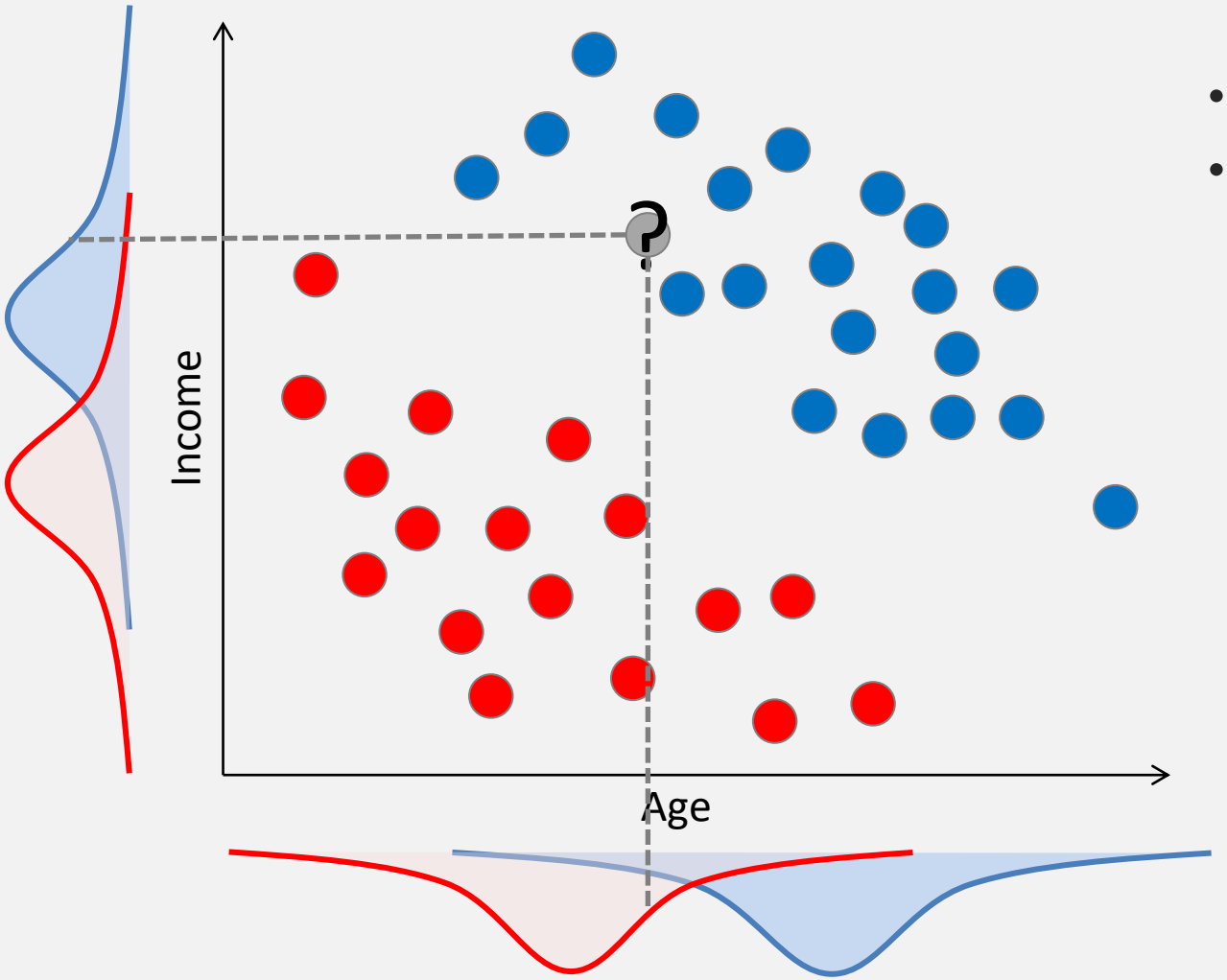
DTs: Pros and Cons

- Pros:
  - Fast and efficient
  - Easy (ish?) to understand the model; can be visualized
  - Little data prep required
    - Can handle nonlinear data
    - Can handle feature interactions
- Cons:
  - Good, not great, predictive performance

NAÏVE BAYES

Naïve Bayes TLDR

Uses Bayes Theorem to see which class is most likely



- ***Training***: calculates distributions per feature/class
- ***Model***: the distributions
- ***Prediction***: calculates probability of each class; choose largest

P(Paid | Income = 51K) = 0.25

P(Default | Income = 51K) = 0.01

P(Paid | Age = 30) = 0.10

P(Default | Age = 30) = 0.22

P(Paid) = 0.25 * 0.10 = 0.025

P(Default) = 0.01 * 0.22 = 0.0022

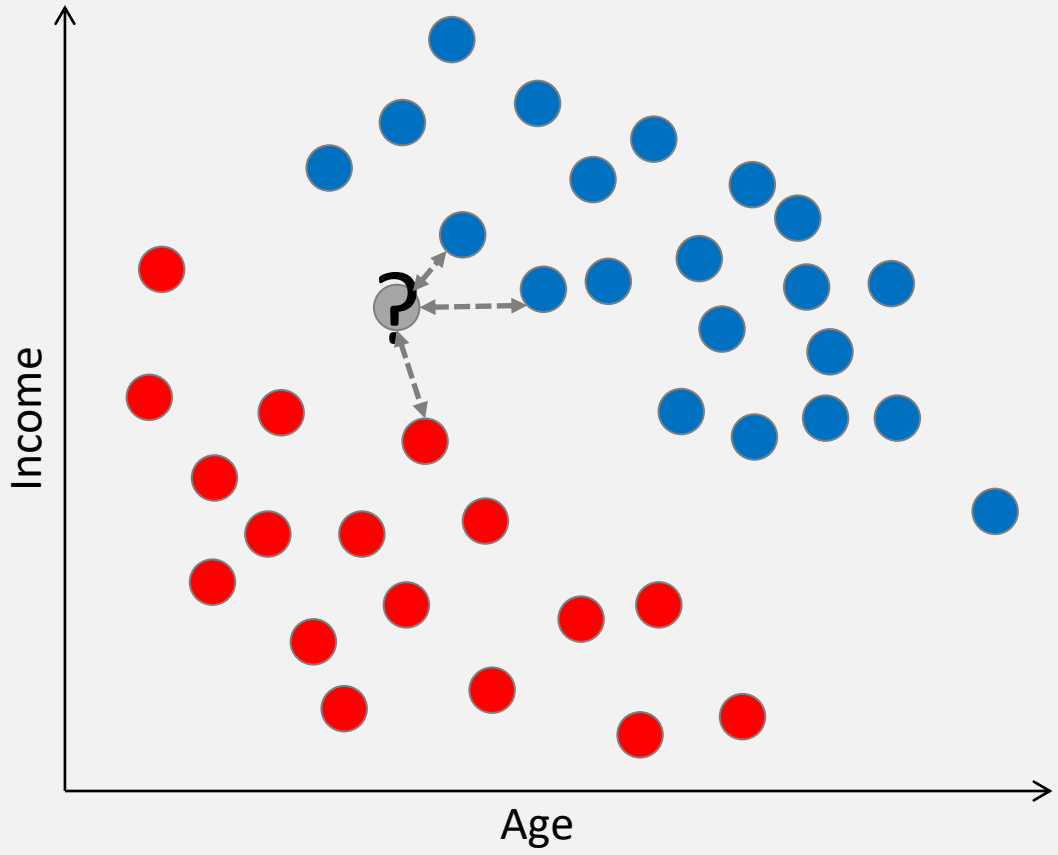
Naïve Bayes: Pros and Cons

- Pros:
  - Fast and efficient
  - Interpretable
  - Resistant to overfitting
- Cons:
  - Makes a strong assumption: features are independent
  - Performance not great

K NEAREST NEIGHBOURS (KNN)

KNN TLDR

A model that classifies new instances by finding the *K* nearest instances in the training data



- ***Training***: None
- ***Model:*** None
- ***Prediction***: Find the *K* nearest instances in the training data; majority wins

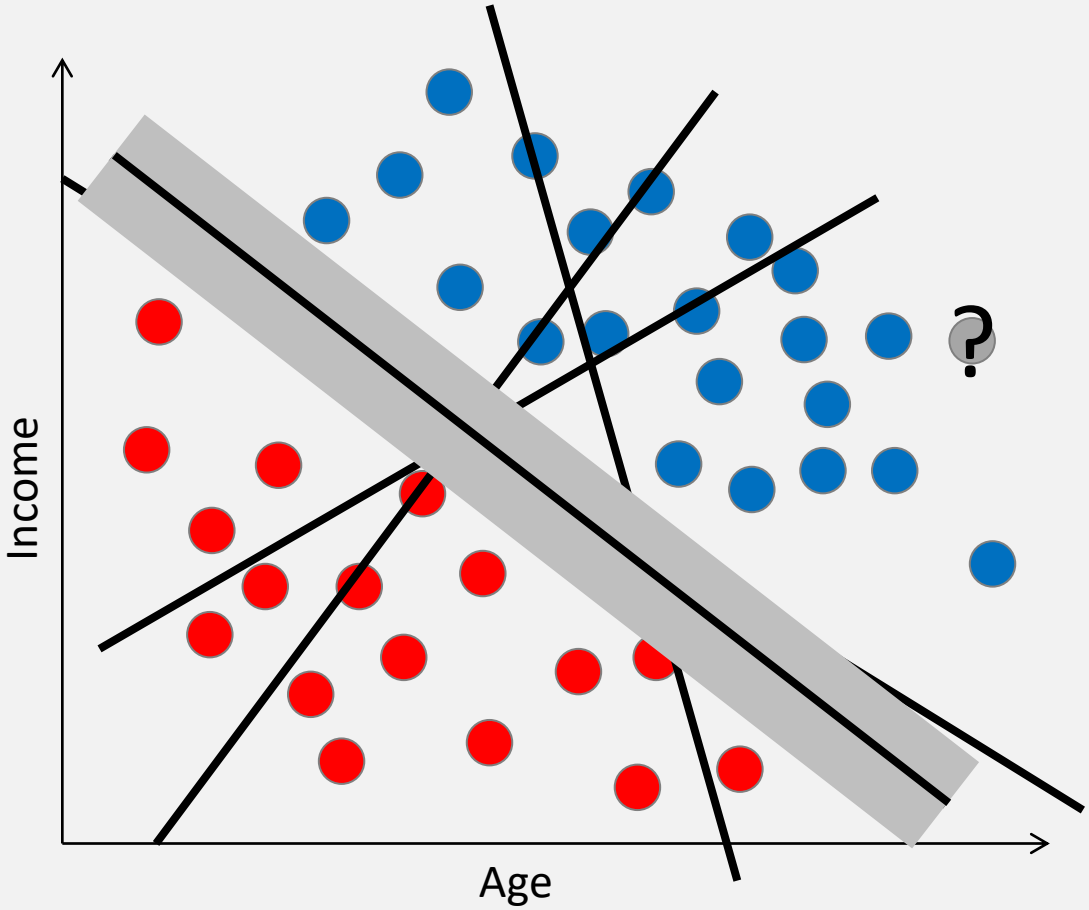
KNN: Pros and Cons

- Pros:
  - Easy to understand, implement, and interpret
  - Training is very fast
  - Usually not bad
- Cons:
  - Memory intensive
  - Prediction can be slow

SUPPORT VECTOR MACHINES

Support Vector Machines TLDR

- Main Idea: draw a straight line* to separate the classes as much as possible
- Sometimes called the "***maximal margin classifier***"

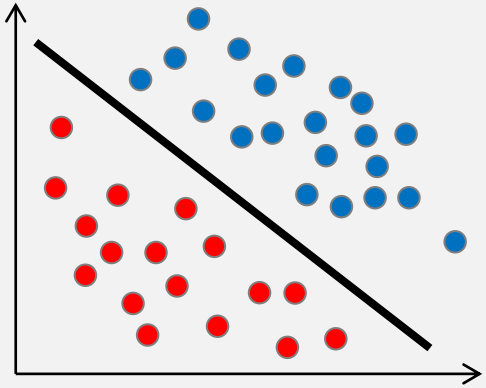


- ***Training***: Algorithm tries a bunch of lines; choose best
- ***Model:*** Equation of the line
- ***Prediction***: Determine which side of the line

Hyperplanes

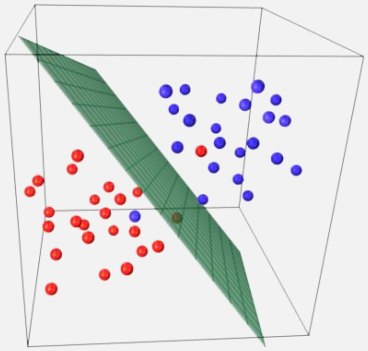
- What if the data has more than two features?
  - No problem, use a hyperplane!
- Conceptually and mathematically the same as a line
  - But, harder for humans to visualize

2 Features



Separate with a 1D hyperplane (i.e., line)

3 Features



Separate with a 2D hyperplane

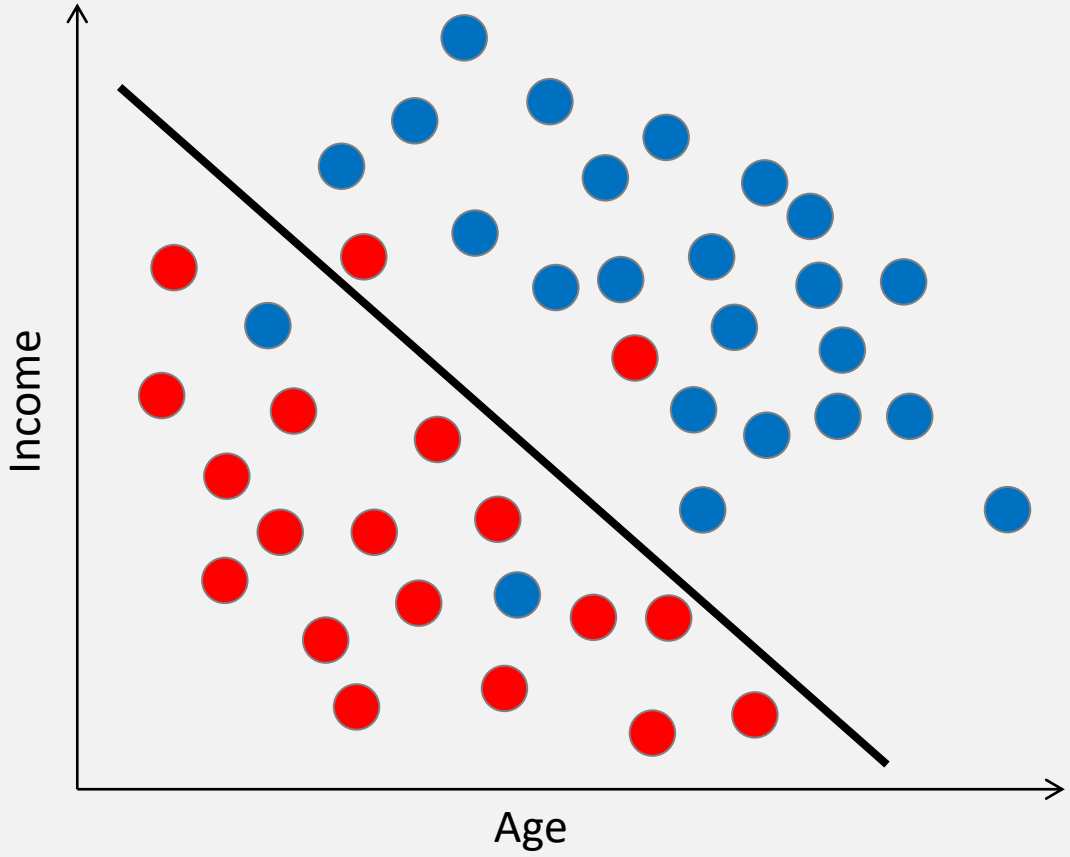
4+ Features

Hard to visualize

Separate with a 3D+ hyperplane

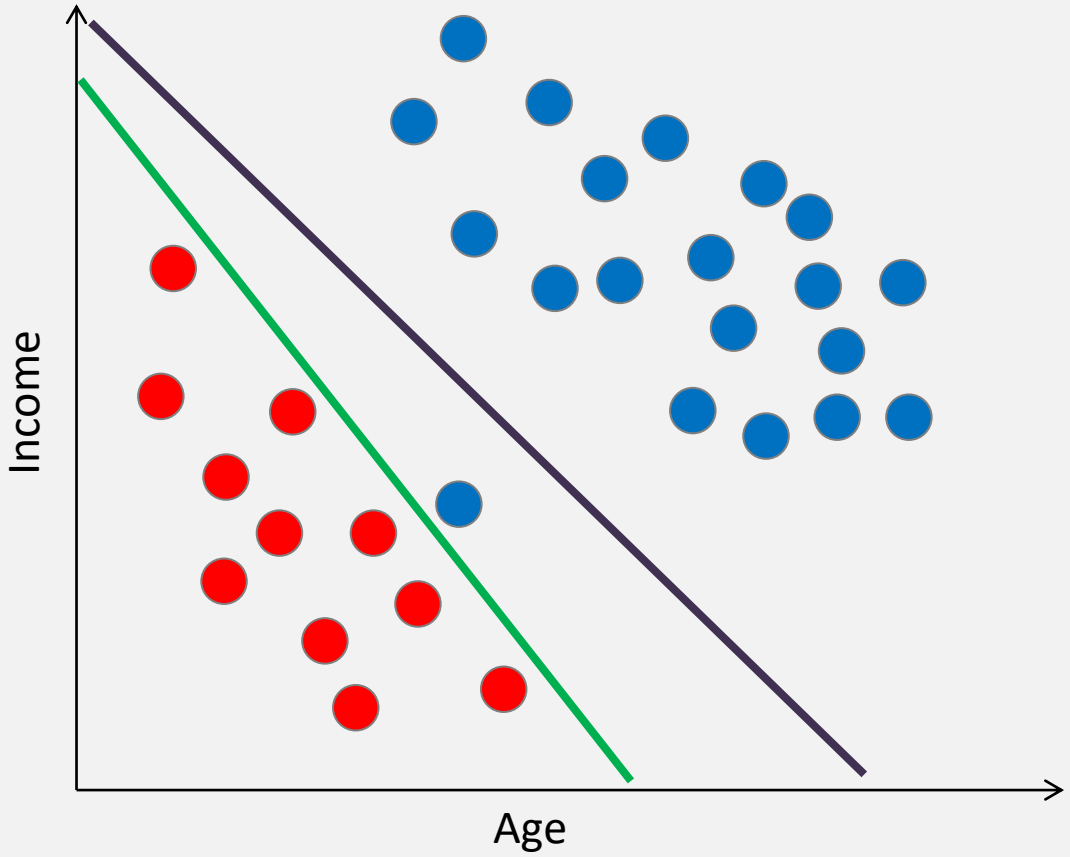
Soft Margins

- What if the classes can't be perfectly separated by a hyperplane?
- No problem: SVM uses a ***soft margin***
  - Allow errors on either side



Soft Margins

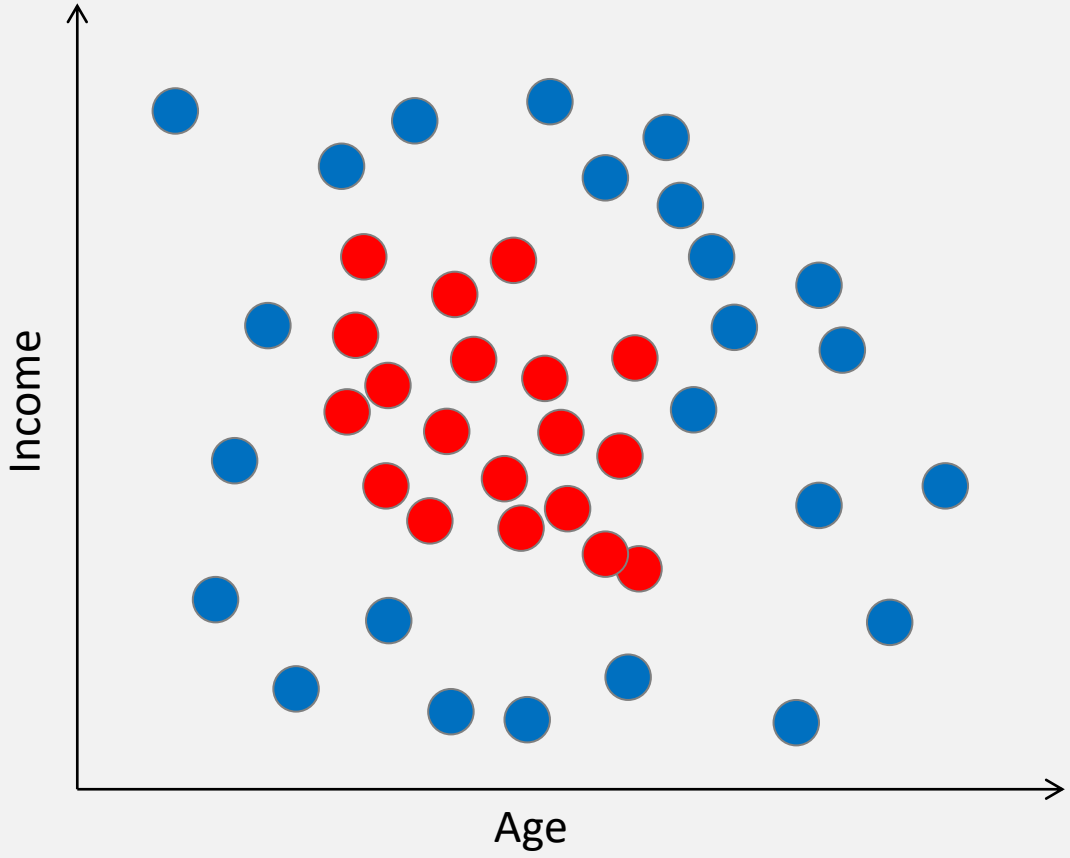
In fact, soft margins are better than hard margins, even if hard margin is possible



- *Green line = hard margin*
- *Purple line = soft margin*
- *Which is better?*

Kernel Trick

- What if the classes just really can't be separated by a straight hyperplane?
- No problem: SVM uses the ***kernel trick*** to create non-linear decision boundary
- The math is complicated; main idea: automated feature engineering

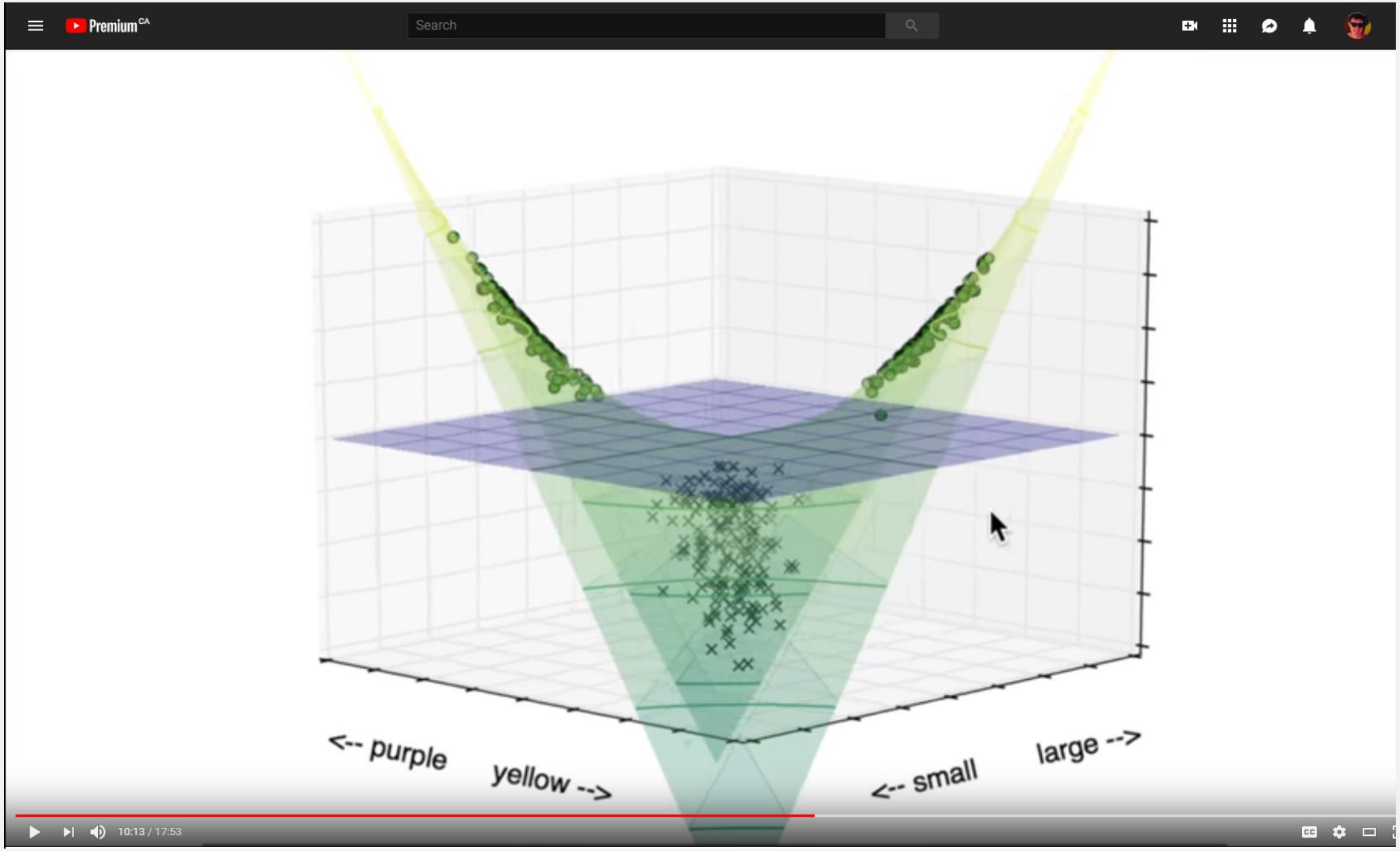


The Kernel Trick

- Engineer new feature(s) (e.g., x^2 + y^2)
- Insert hyperplane in new feature space
- Transform back to original feature space

Source: Support Vector Machines in 2 minutes

The Kernel Trick



Great video: https://youtu.be/-Z4aojJ-pdg?t=9m27s

Understanding Kernel Functions

…and a **Polynomial Kernel** with
***r* = 0** and ***d* = 2** is equal to $(ab)^2$…

$$a^0b^0 + a^1b^1 + a^2b^2 + \cdots + a^\infty b^\infty = (1, a, a^2, \ldots, a^\infty) \cdot (1, b^1, b^2, \ldots, b^\infty)$$

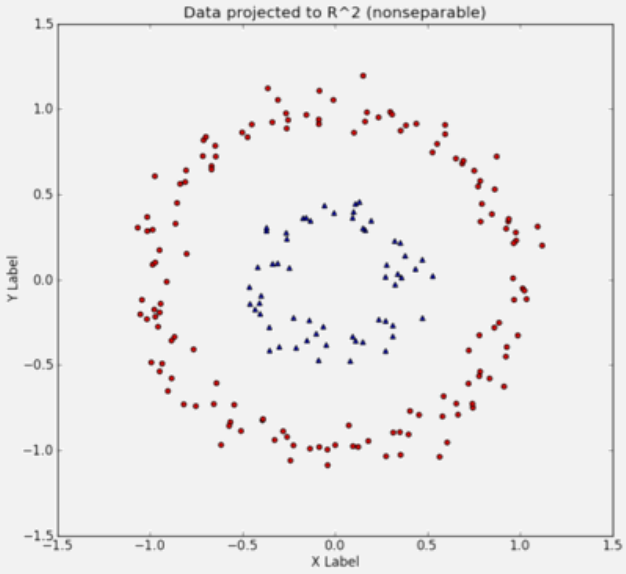
$$e^{ab} = 1 + \frac{1}{1!}ab + \frac{1}{2!}(ab)^2 + \frac{1}{3!}(ab)^3 + \cdots + \frac{1}{\infty!}(ab)^\infty$$

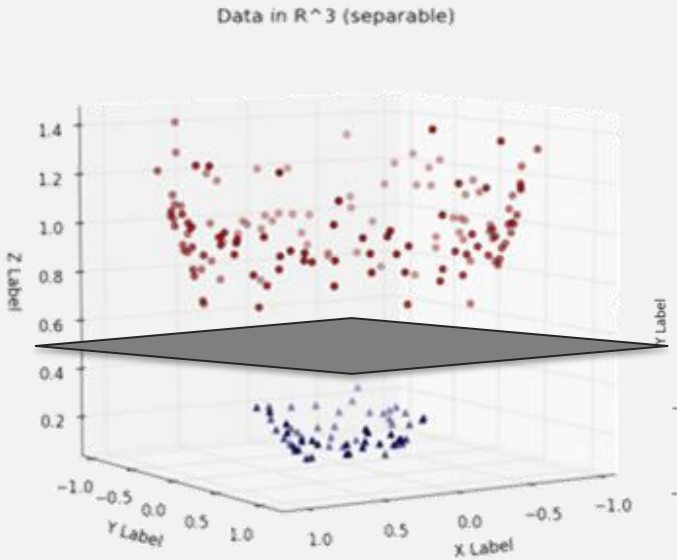
[StatQuest: Support Vector Machines, Clearly Explained!!!]

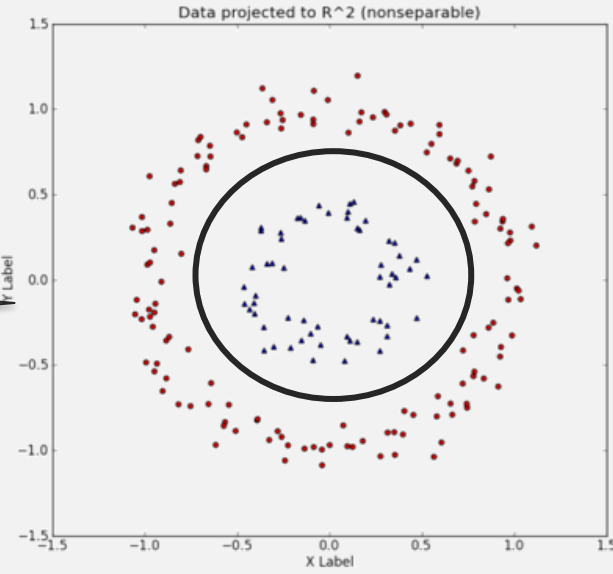
The Kernel Trick

- SVM can use non-linear ***kernel function*** to create new features
  - Goal: new features will make it *easier* to separate
  - Like the opposite of dimensionality reduction (e.g., PCA)

Original Data → (Kernel function) → X1, X2, X3 → (Inverse kernel function) → Original Data

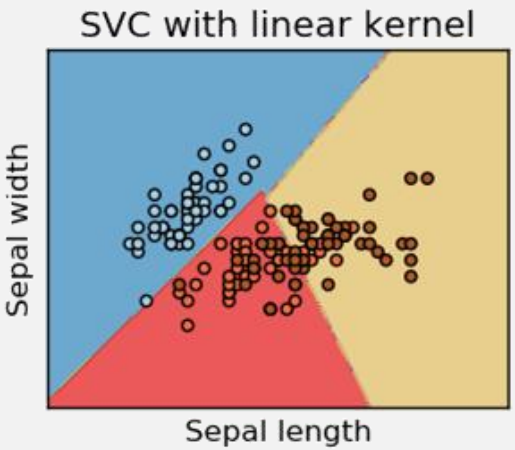




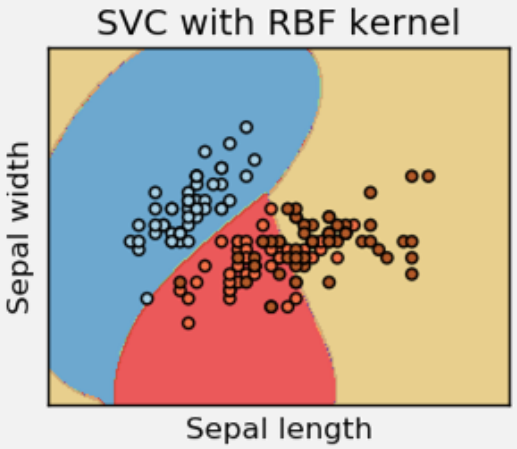


The Kernel Trick

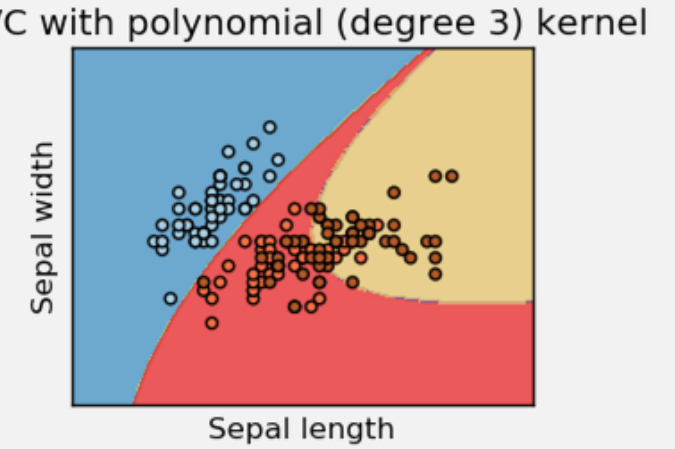
- A few common kernels
- Which one is best for your data?
  - If data is linearly separable → use linear kernel
  - Otherwise, try the others and see which produces best measure



Fastest
Least flexible



Slowest
Most flexible



In the middle
In the middle

Source: Scikit-learn: Plot different SVM classifiers in the iris dataset

SVM Summary

- **Training Phase**: Find hyperplane that best splits training data
- **Model**: Equation of hyperplane
- **Prediction Phase**: Determine which side of hyperplane the instance is on
- **Decision boundary**: anything

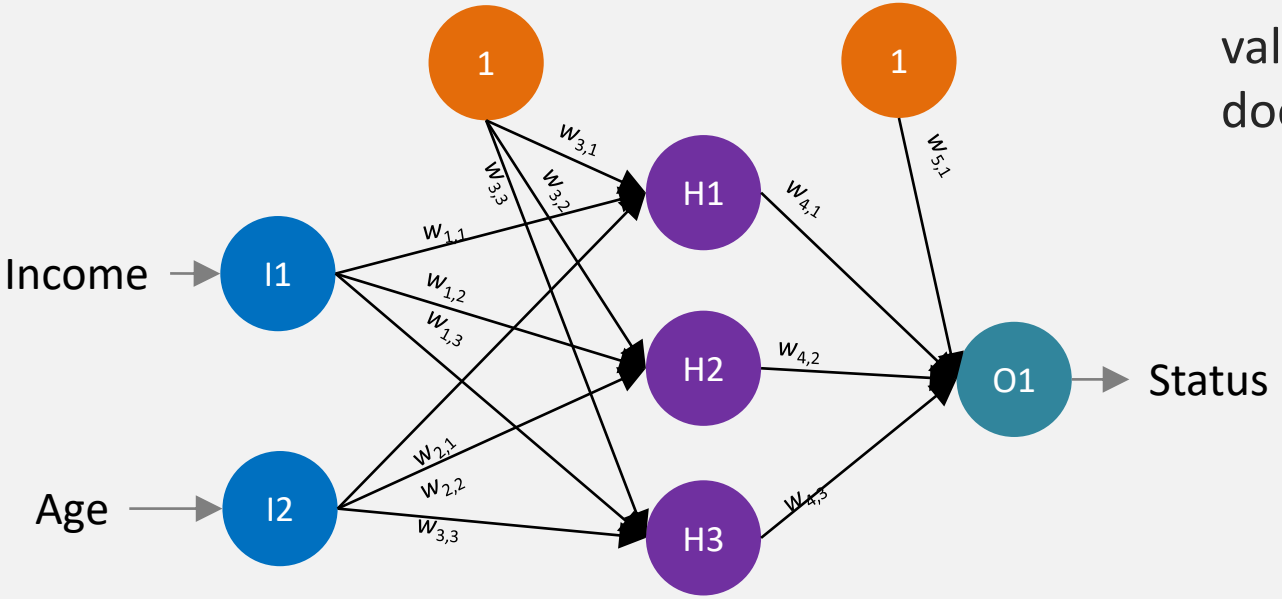
SVM: Pros and Cons

- Pros:
  - High accuracy
  - Can learn non-linear decision boundaries
  - Versatile because of kernel trick
- Cons:
  - Training time is very long
  - Hard to interpret
  - Can overfit

NEURAL NETWORKS

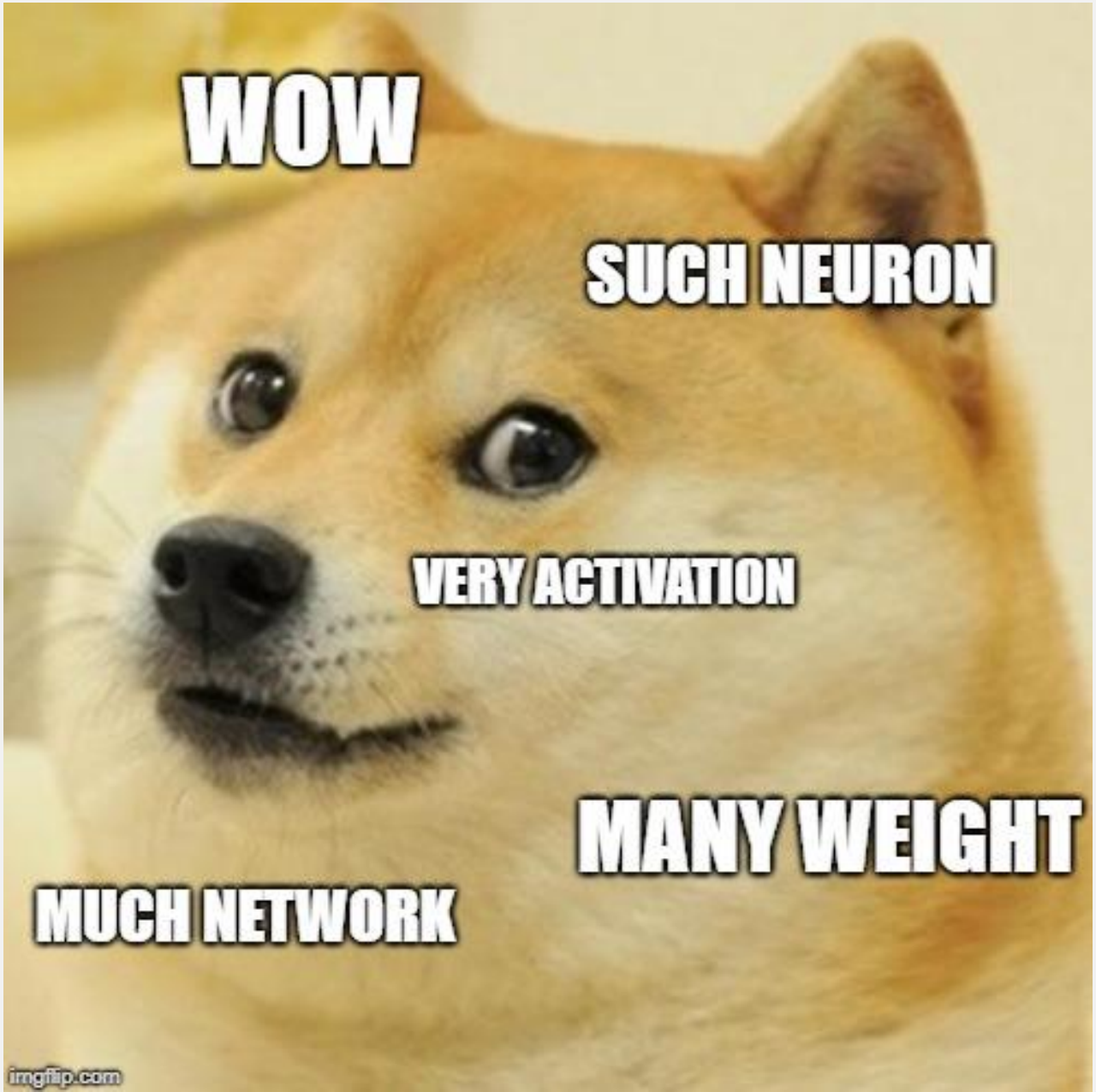
TLDR: Neural Networks

- Model loosely inspired by the human brain
  - A network of ***neurons*** connected by ***weighted edges***
- Architecture is chosen by you, the human

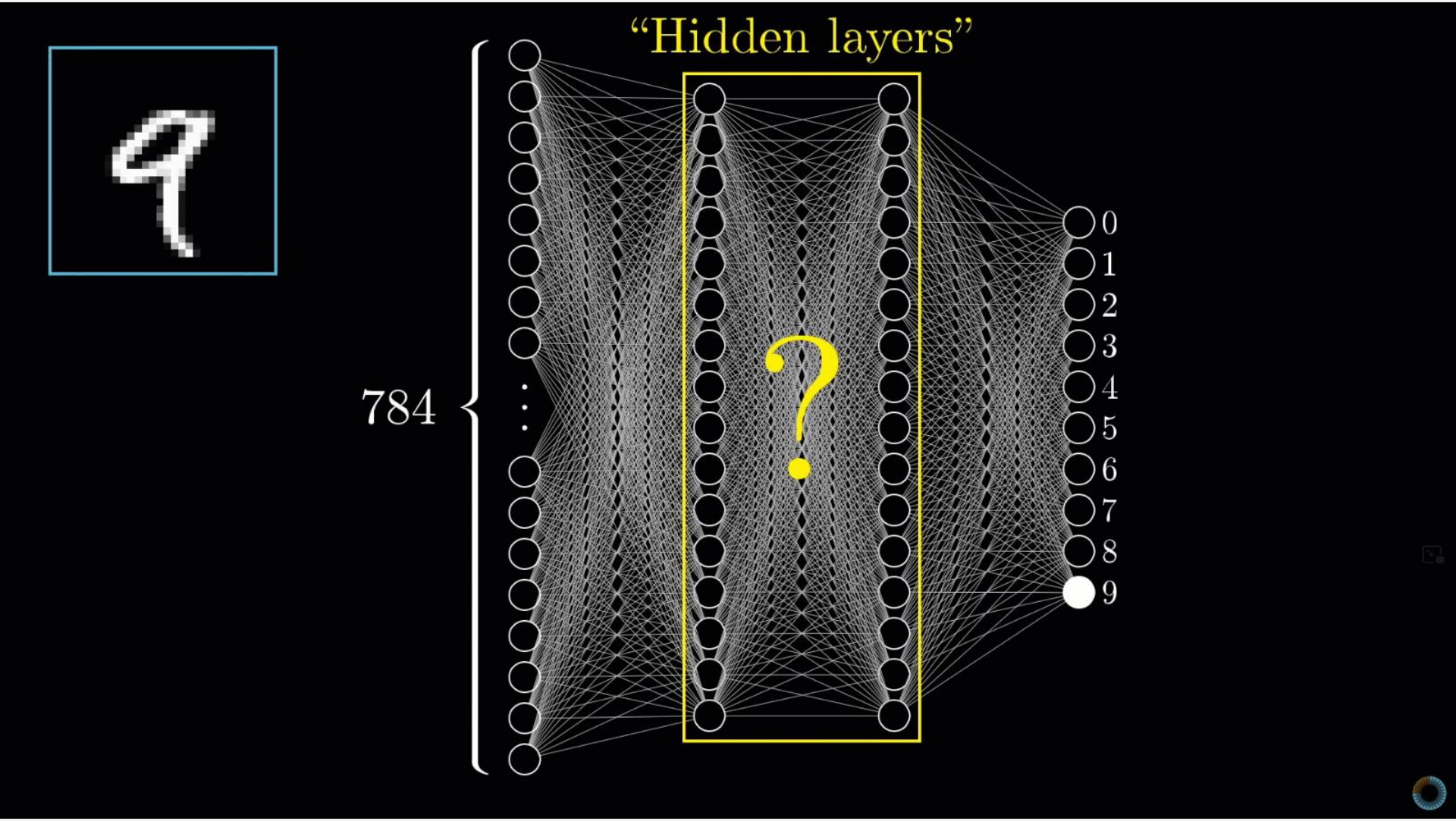


- ***Training***: Algorithm finds best values for weights
- ***Model:*** The weights
- ***Prediction***: Plug instance values into input layer, NN does math

So Hot Right Now



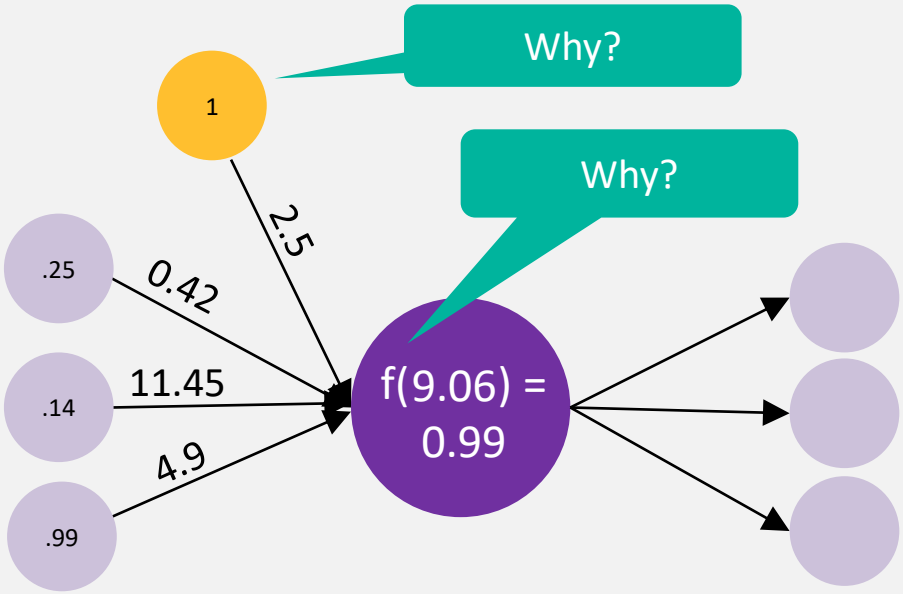
Best for Images



Source: But what is a neural network (3Blue1Brown)

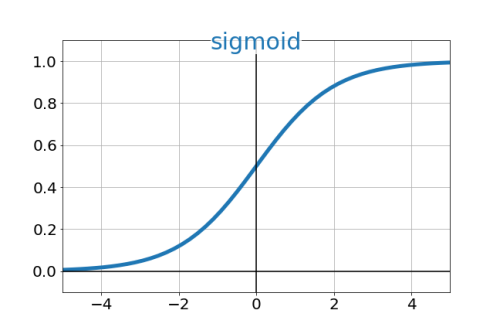
What Math Does a Neuron Do?

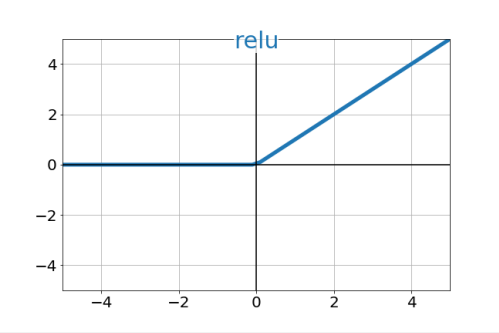
- Receives some numbers from neurons in previous layer (*input)*
- Calculates a weighted sum of its input
  - (0.25*0.42) + (0.14*11.45) + (0.99*4.9) + (1*2.5) = 9.06
- Uses activation function
  - Sigmoid(9.06) = 0.99
- Passes 0.99 along to all the other nodes in next layer

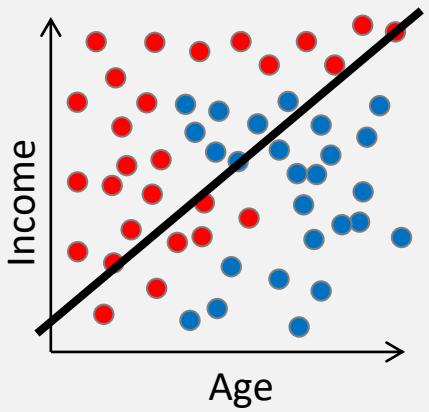


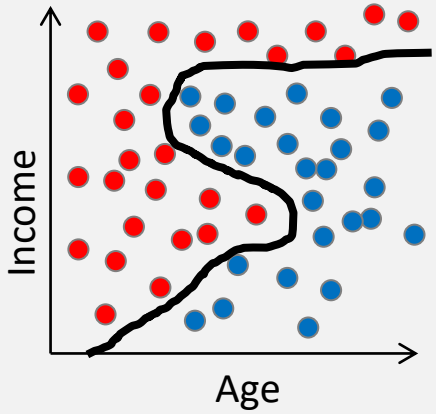
Why Activation Functions?

- Two reasons:
  - Constrain the neuron's output to a certain range
  - Allow NN to learn non-linear boundaries

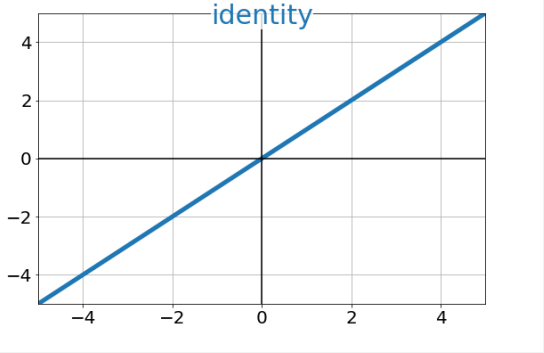


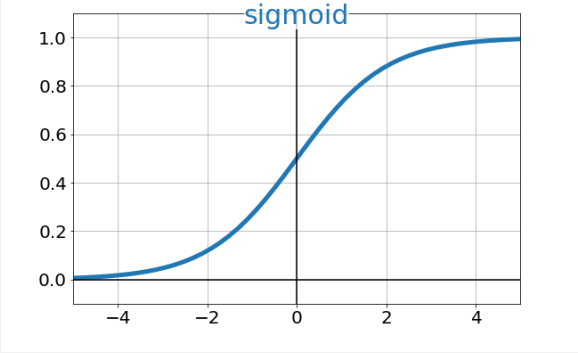


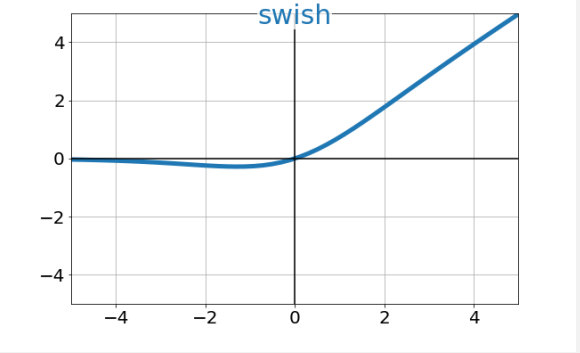


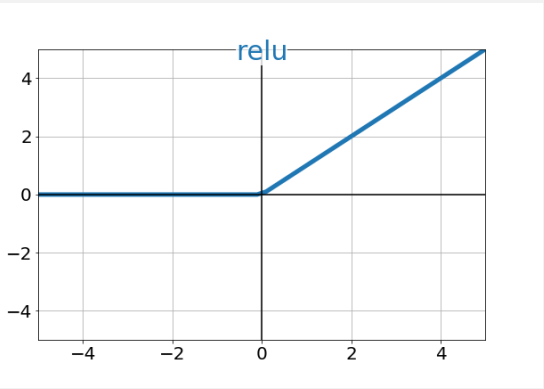


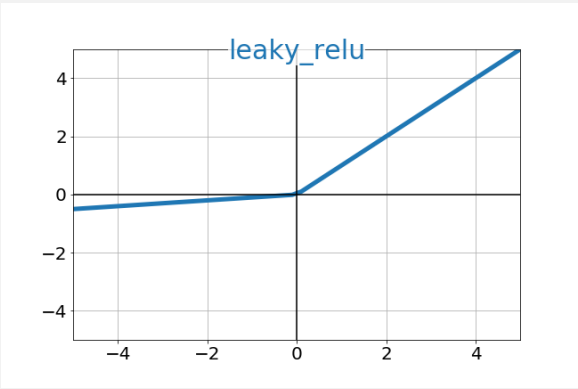
More Activation Functions

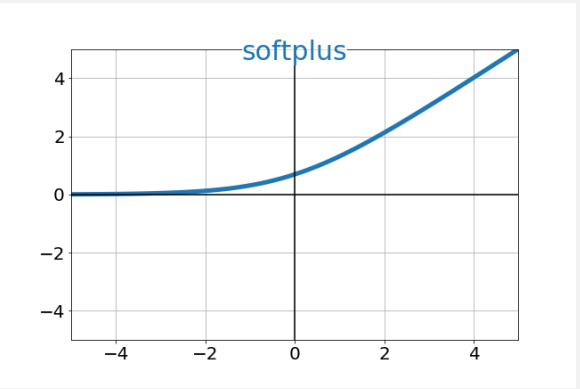






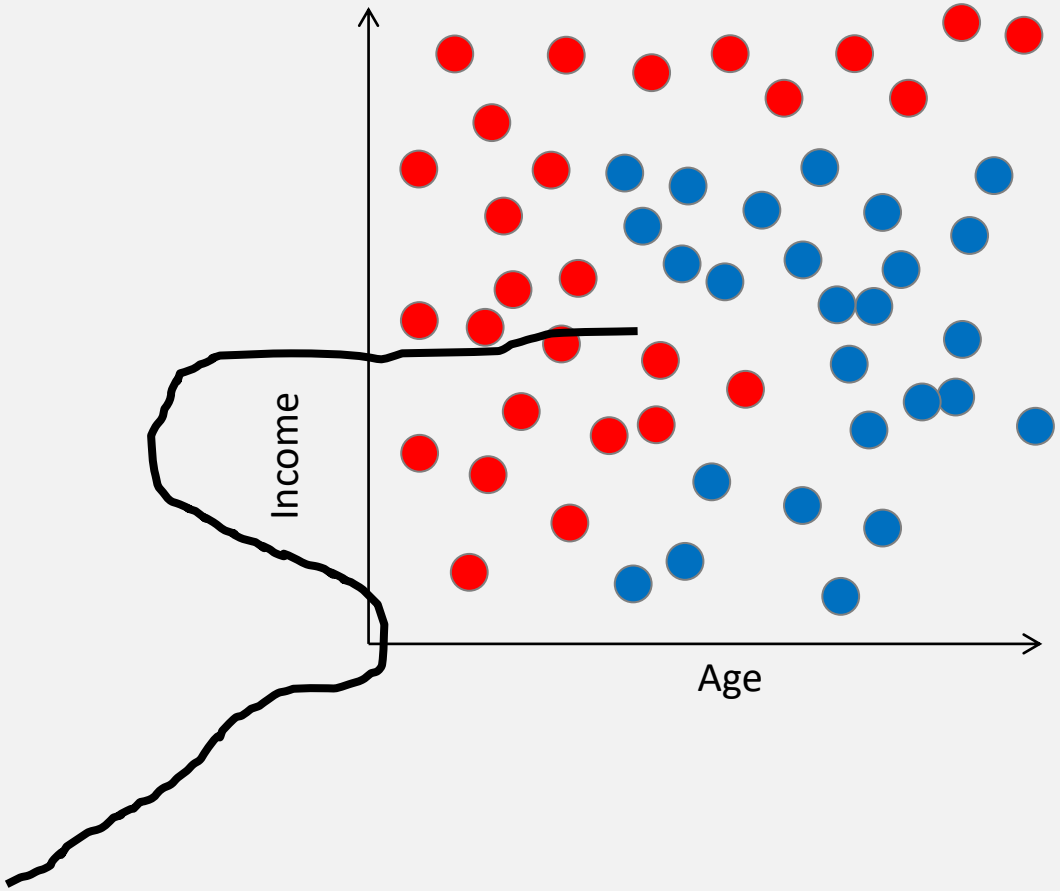






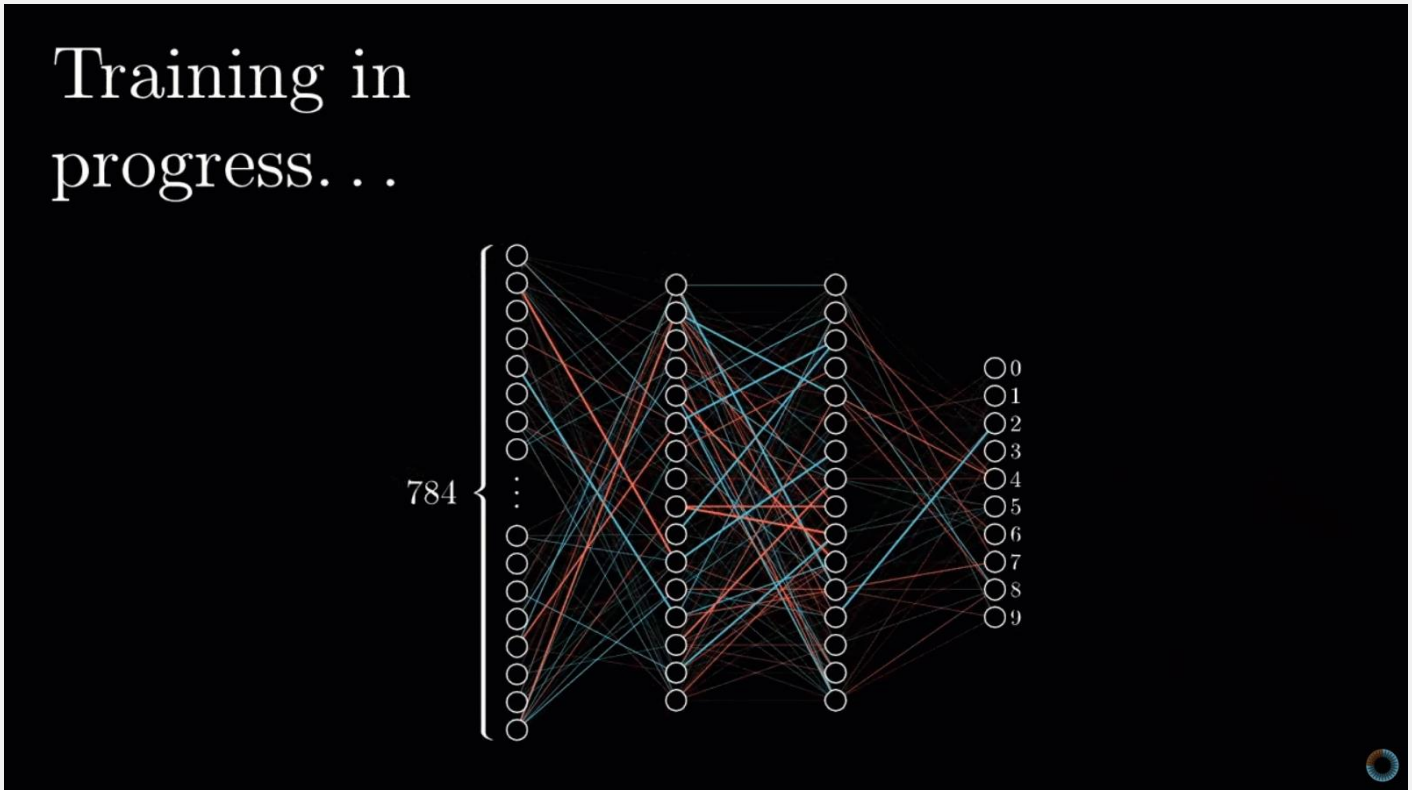
Why Bias?

- Allows the decision boundary to shift



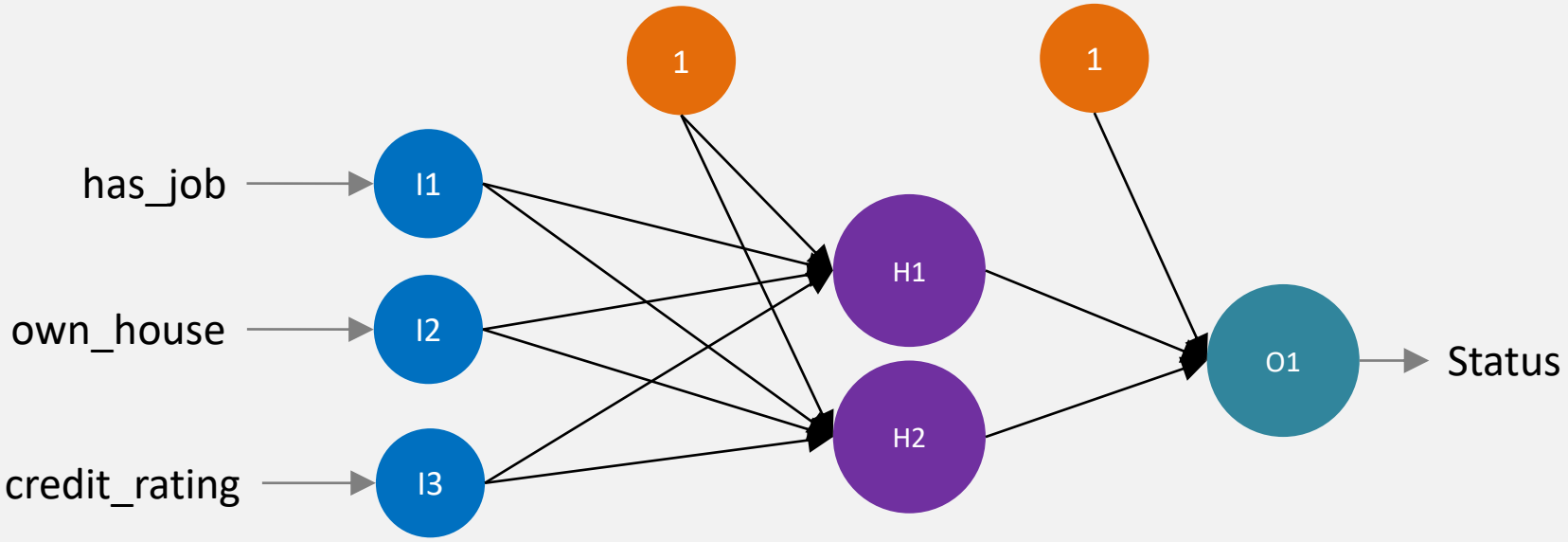
Training Phase: Learning the Weights

- ***Backpropagation*** is the most common training algorithm
- Big idea: nudge weights up and down repeatedly to increase accuracy
  - Randomly initialize all the weights
  - Feed in the training instances, and see if output is correct
  - If not, nudge/adjust the weights a little
  - Repeat for all training instances until weights converge



Training Phase Example

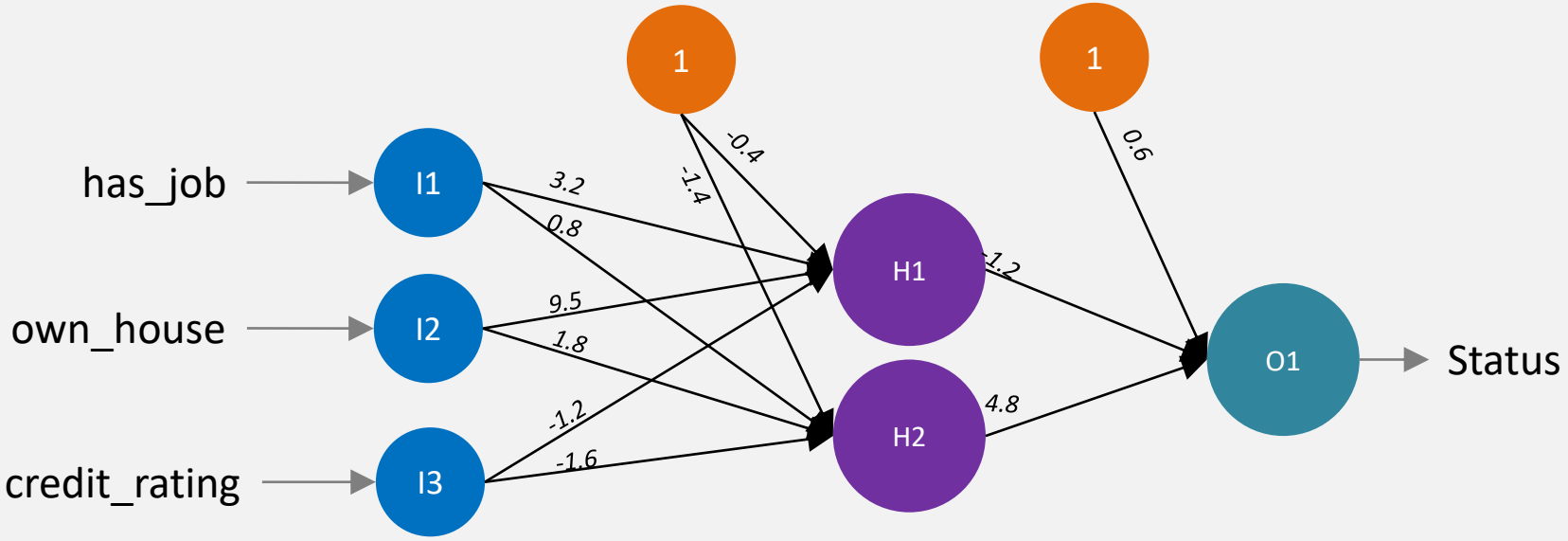
- Choose architecture
  - 3 input neurons, 2 hidden neurons, 1 output neuron
  - Activation function for hidden neurons = relu
    - f(x) = {x if x >= 0, 0 otherwise}
  - Activation function for output neuron = sigmoid
    - f(x) = 1/(1+exp(-x))



Training Phase Example

has_job	own_house	credit_rating	status
1	0	2	1
0	1	2	1
0	1	2	1
0	1	2	1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	1	0	1
0	0	1	0
1	0	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	1

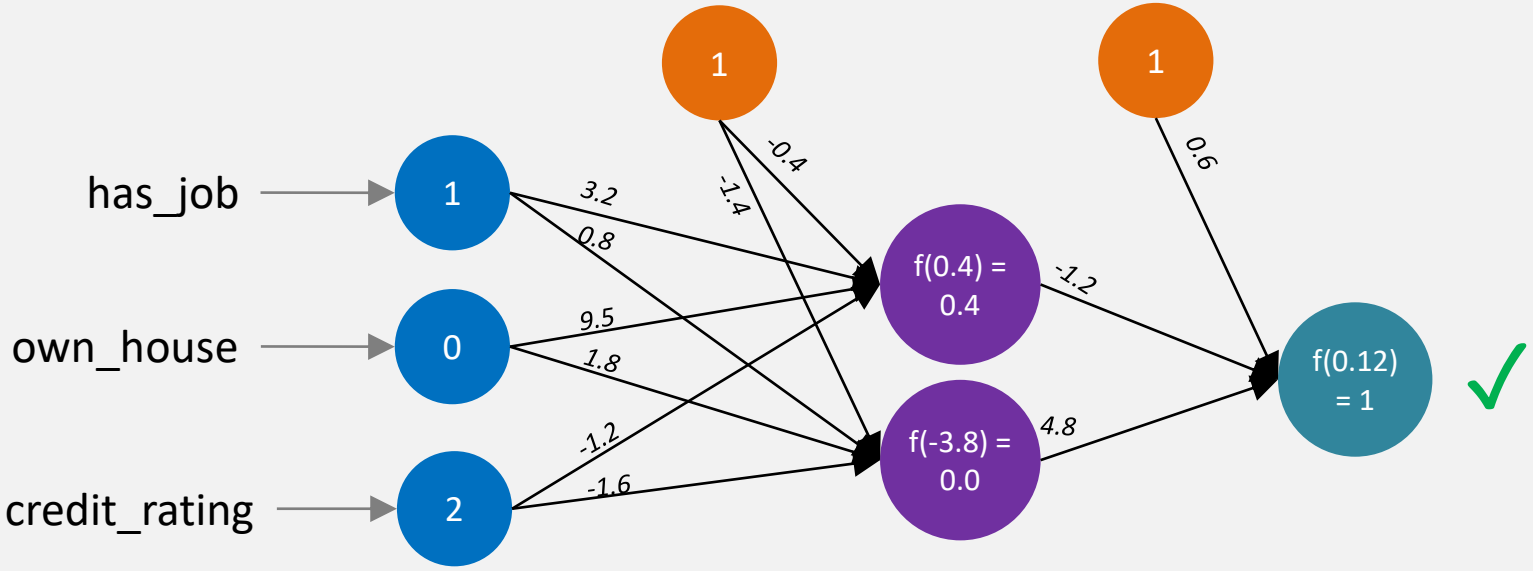
- Randomly initialize weights



Training Phase Example

has_job	own_house	credit_rating	status
1	0	2	1
0	1	2	1
0	1	2	1
0	1	2	1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	1	0	1
0	0	1	0
1	0	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	1

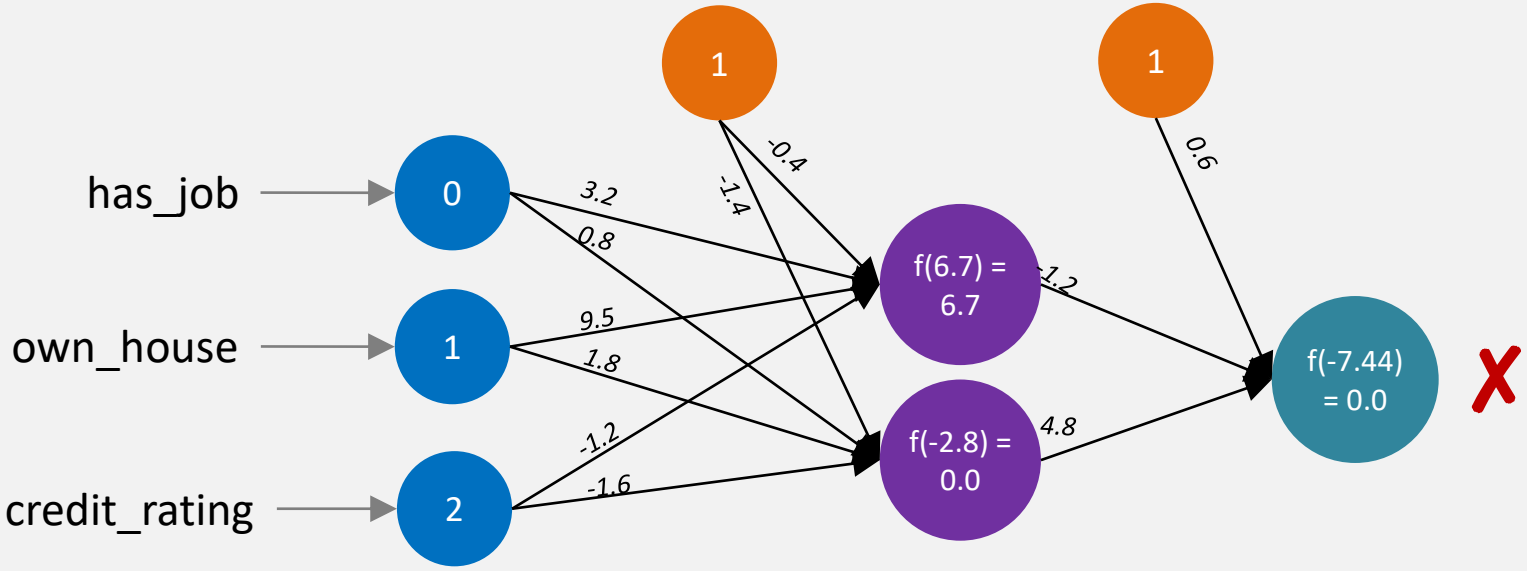
- Randomly initialize weights
- Feed the training data, compute errors



Training Phase Example

has_job	own_house	credit_rating	status	
1	0	2	1	✓
0	1	2	1	✗
0	1	2	1	
0	1	2	1	
0	0	0	0	
0	0	0	0	
0	0	0	0	
0	0	0	0	
1	1	0	1	
0	0	1	0	
1	0	1	1	
1	0	1	1	
0	0	1	0	
0	1	1	1	
1	1	1	1	

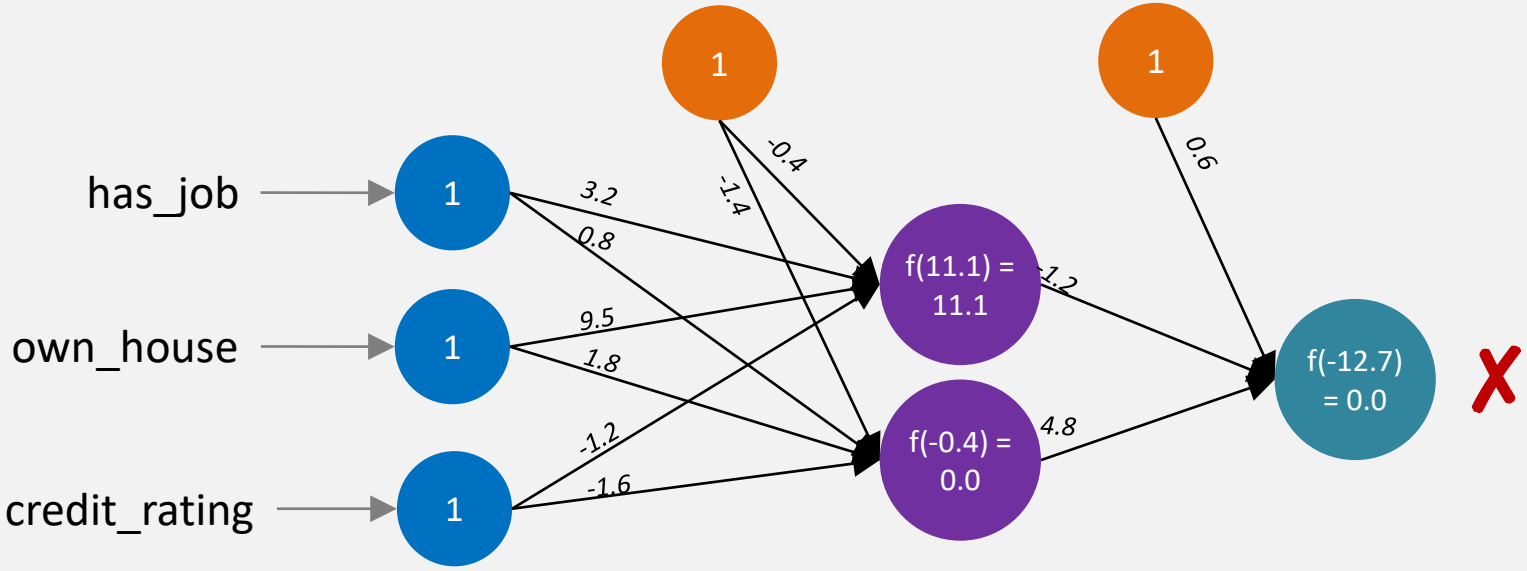
- Randomly initialize weights
- Feed the training data, compute errors



Training Phase Example

has_job	own_house	credit_rating	status	
1	0	2	1	✓
0	1	2	1	✗
0	1	2	1	✗
0	1	2	1	✗
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
1	1	0	1	✗
0	0	1	0	✗
1	0	1	1	✗
1	0	1	1	✗
0	0	1	0	✗
0	1	1	1	✗
1	1	1	1	✗

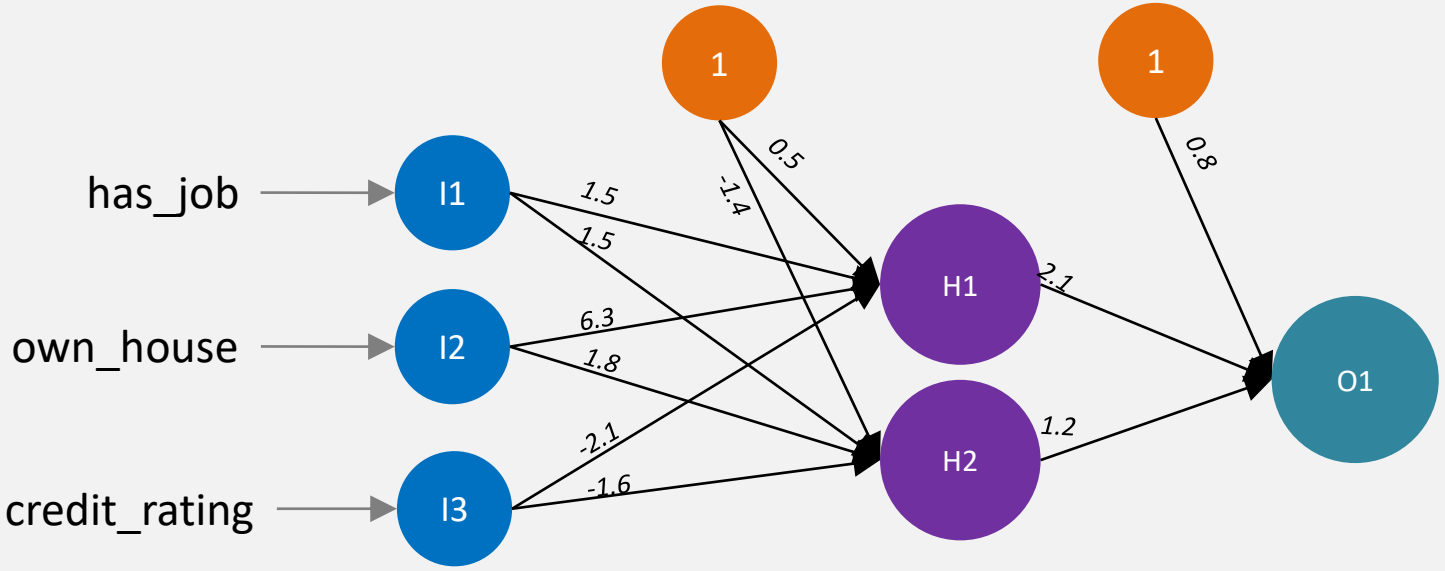
- Randomly initialize weights
- Feed the training data, compute errors



Training Phase Example

has_job	own_house	credit_rating	status	
1	0	2	1	✓
0	1	2	1	✗
0	1	2	1	✗
0	1	2	1	✗
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
1	1	0	1	✗
0	0	1	0	✗
1	0	1	1	✗
1	0	1	1	✗
0	0	1	0	✗
0	1	1	1	✗
1	1	1	1	✗

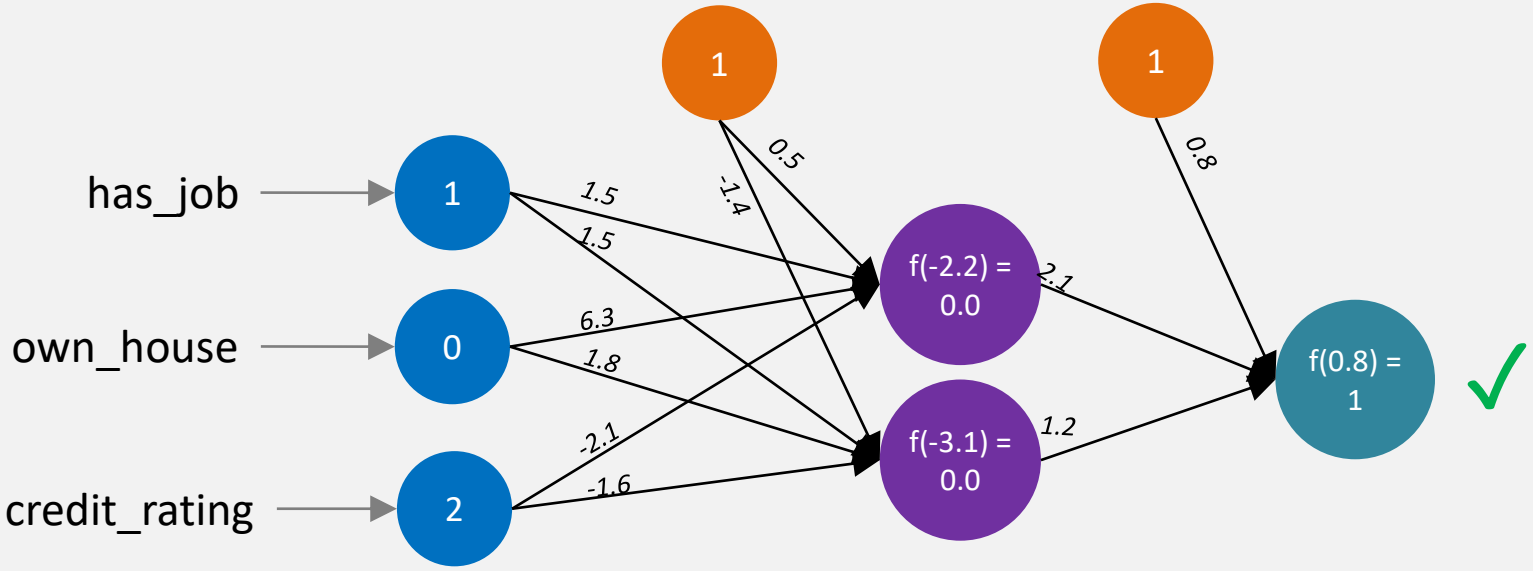
- Randomly initialize weights
- Feed the training data, compute errors
- **Propagate the errors back** to adjust weights



Training Phase Example

has_job	own_house	credit_rating	status
1	0	2	1
0	1	2	1
0	1	2	1
0	1	2	1
0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0
1	1	0	1
0	0	1	0
1	0	1	1
1	0	1	1
0	0	1	0
0	1	1	1
1	1	1	1

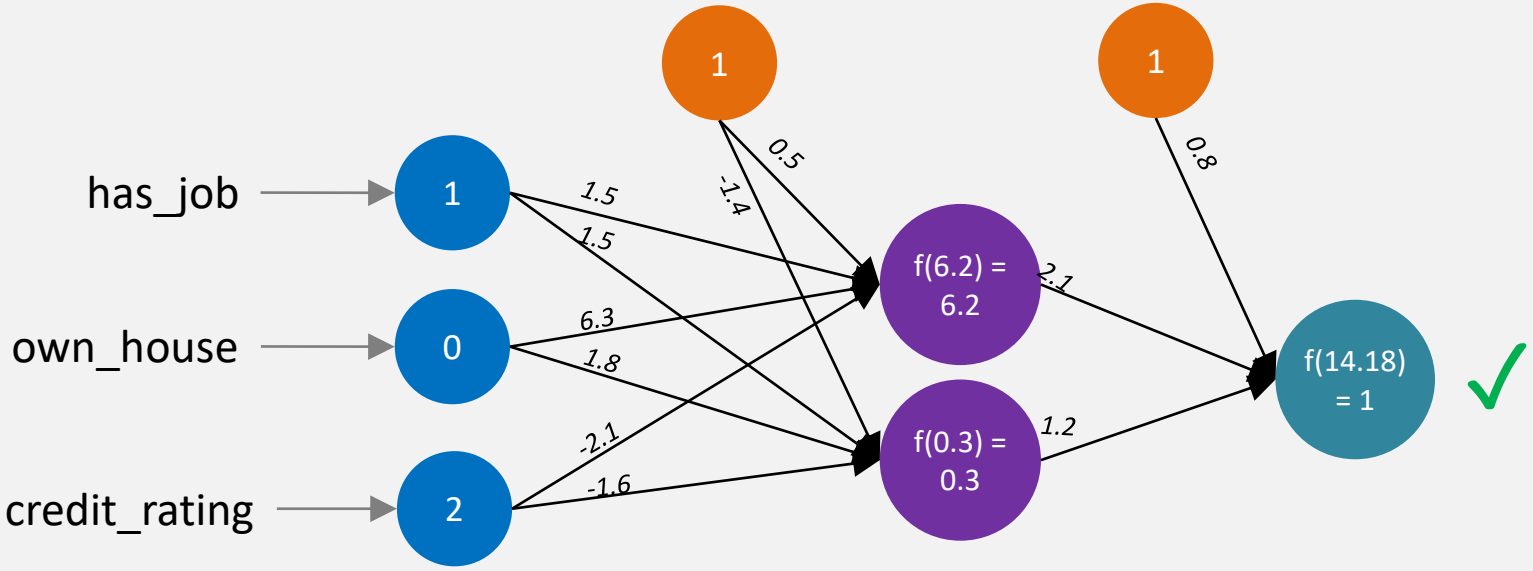
- Randomly initialize weights
- Feed the training data, compute errors
- **Propagate the errors back** to adjust weights
- Repeat



Training Phase Example

has_job	own_house	credit_rating	status	
1	0	2	1	✓
0	1	2	1	✓
0	1	2	1	✓
0	1	2	1	✓
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
0	0	0	0	✗
1	1	0	1	✓
0	0	1	0	✗
1	0	1	1	✓
1	0	1	1	✓
0	0	1	0	✗
0	1	1	1	✓
1	1	1	1	✓

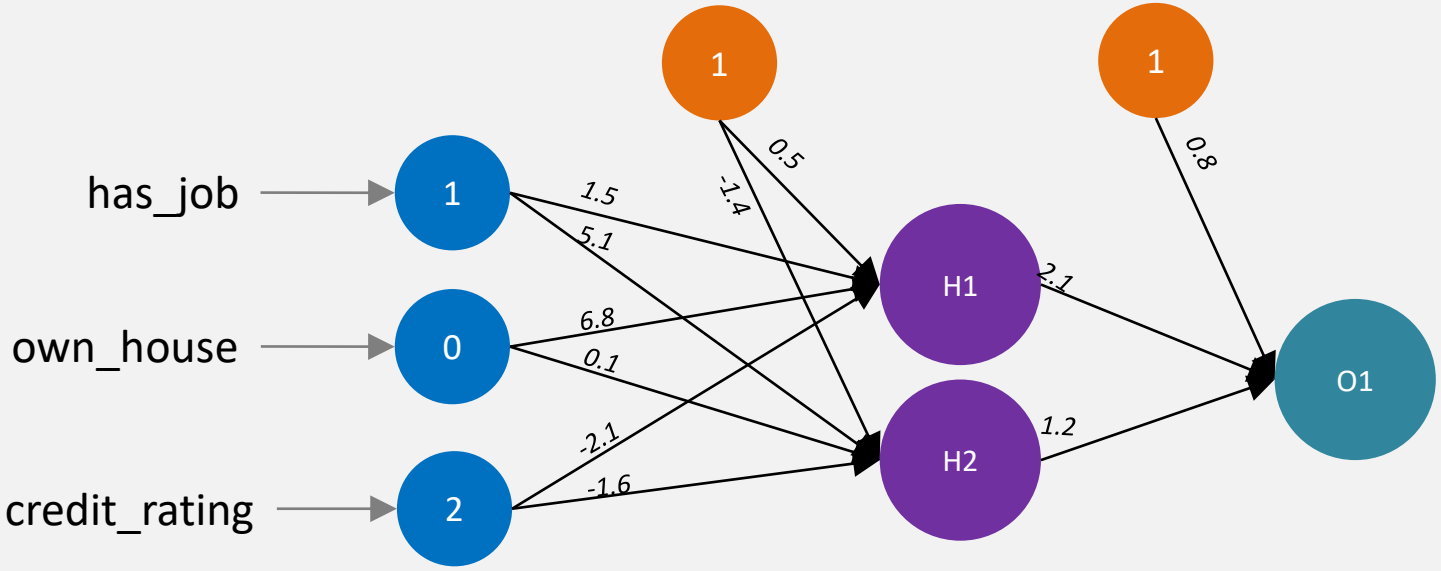
- Randomly initialize weights
- Feed the training data, compute errors
- **Propagate the errors back** to adjust weights
- Repeat



Training Phase Example

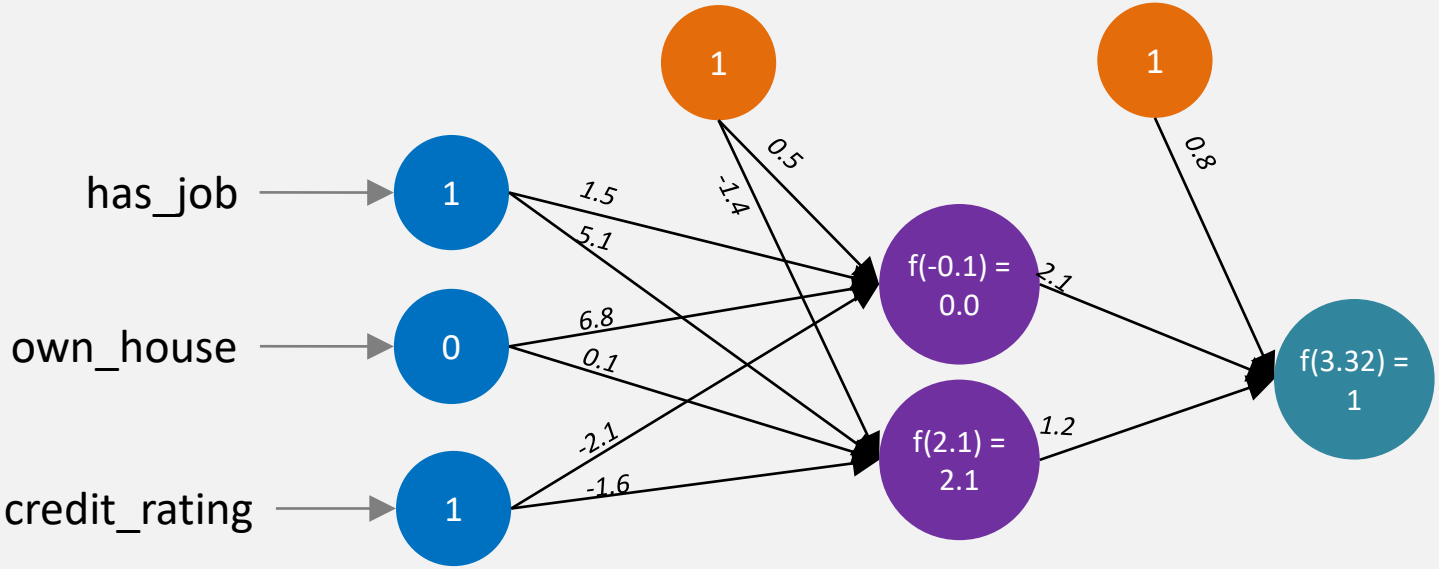
has_job	own_house	credit_rating	status	
1	0	2	1	✓
0	1	2	1	✓
0	1	2	1	✓
0	1	2	1	✓
0	0	0	0	✓
0	0	0	0	✓
0	0	0	0	✓
0	0	0	0	✓
1	1	0	1	✓
0	0	1	0	✓
1	0	1	1	✓
1	0	1	1	✗
0	0	1	0	✓
0	1	1	1	✓
1	1	1	1	✓

- Randomly initialize weights
- Feed the training data, compute errors
- **Propagate the errors back** to adjust weights
- Repeat
- Eventually, weights will converge

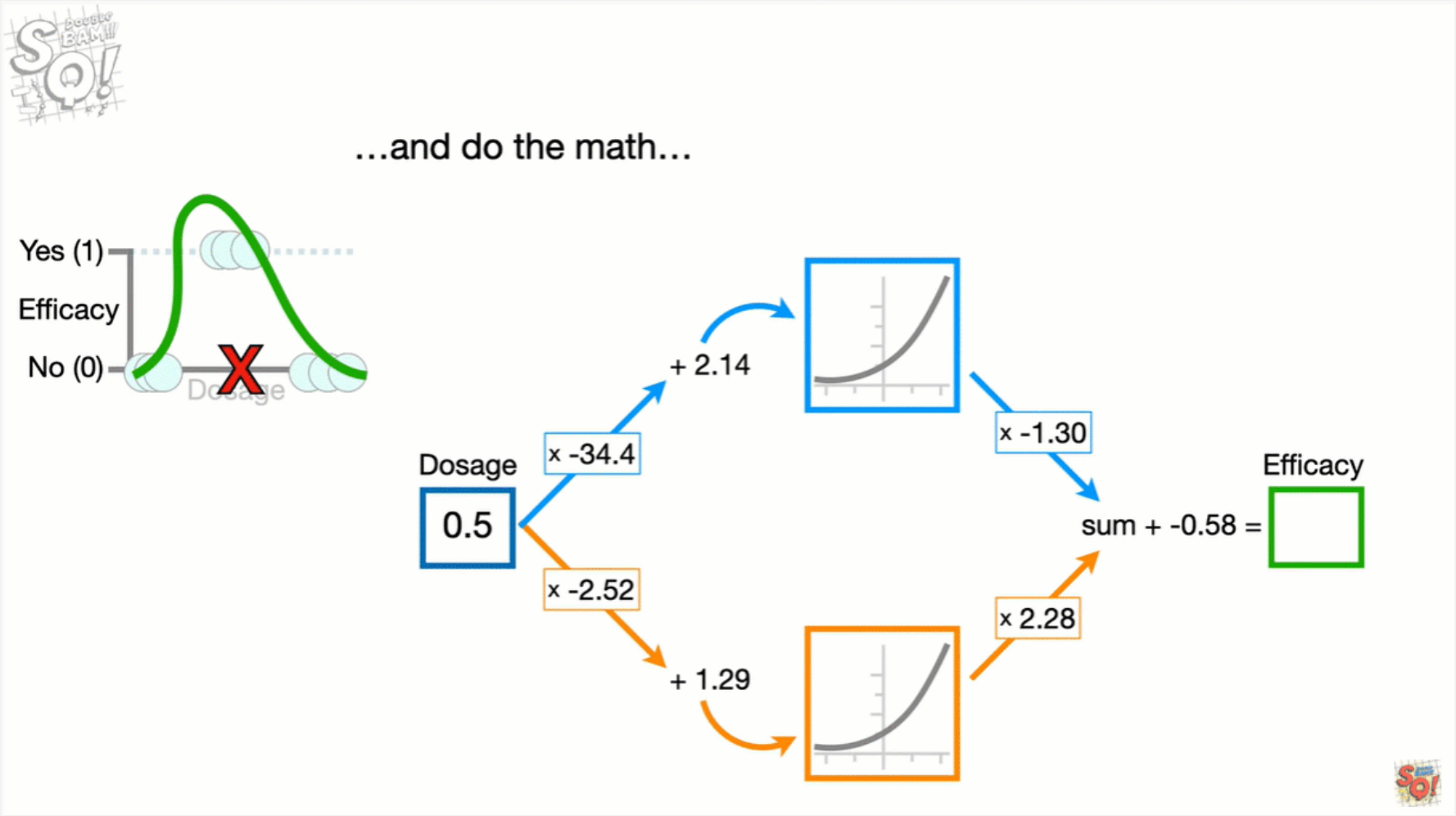


Prediction Phase Example

- Suppose a new instance is:
  - has_job=1, own_house = 0, and credit_rating = 1



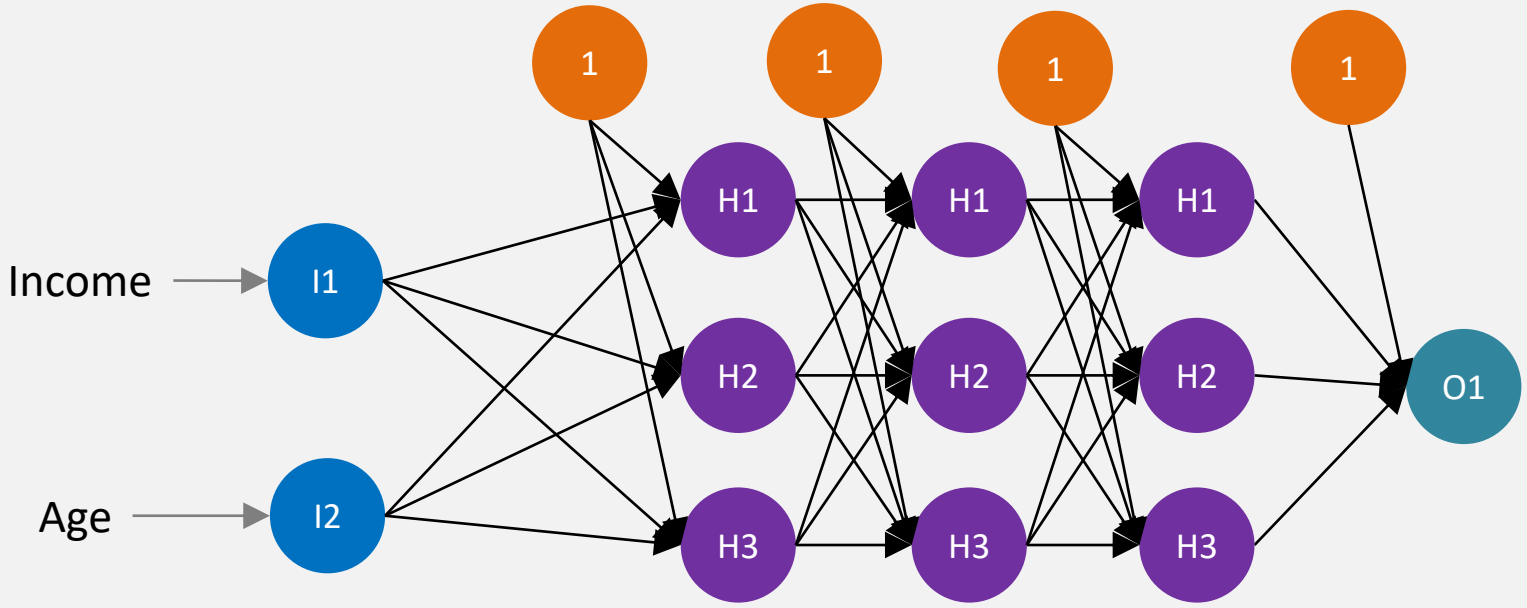
StatQuest Video



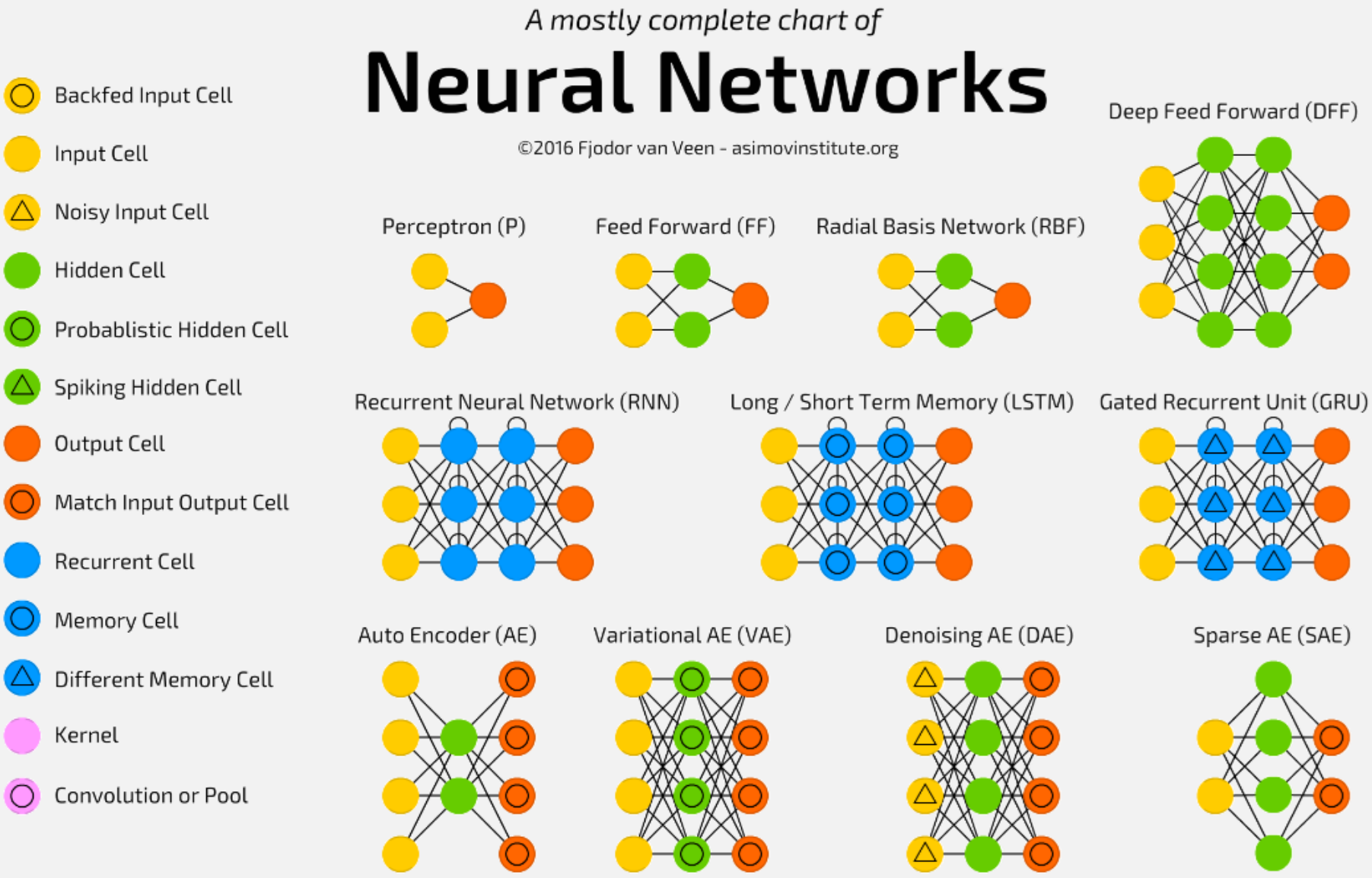
[StatQuest: Neural Networks Part 1]

Why Multiple Layers?

- Allows NN to perform (automated, advanced, black-box) feature engineering
- NN have proven very useful for:
  - Image classification
  - Speech recognition
  - NLP

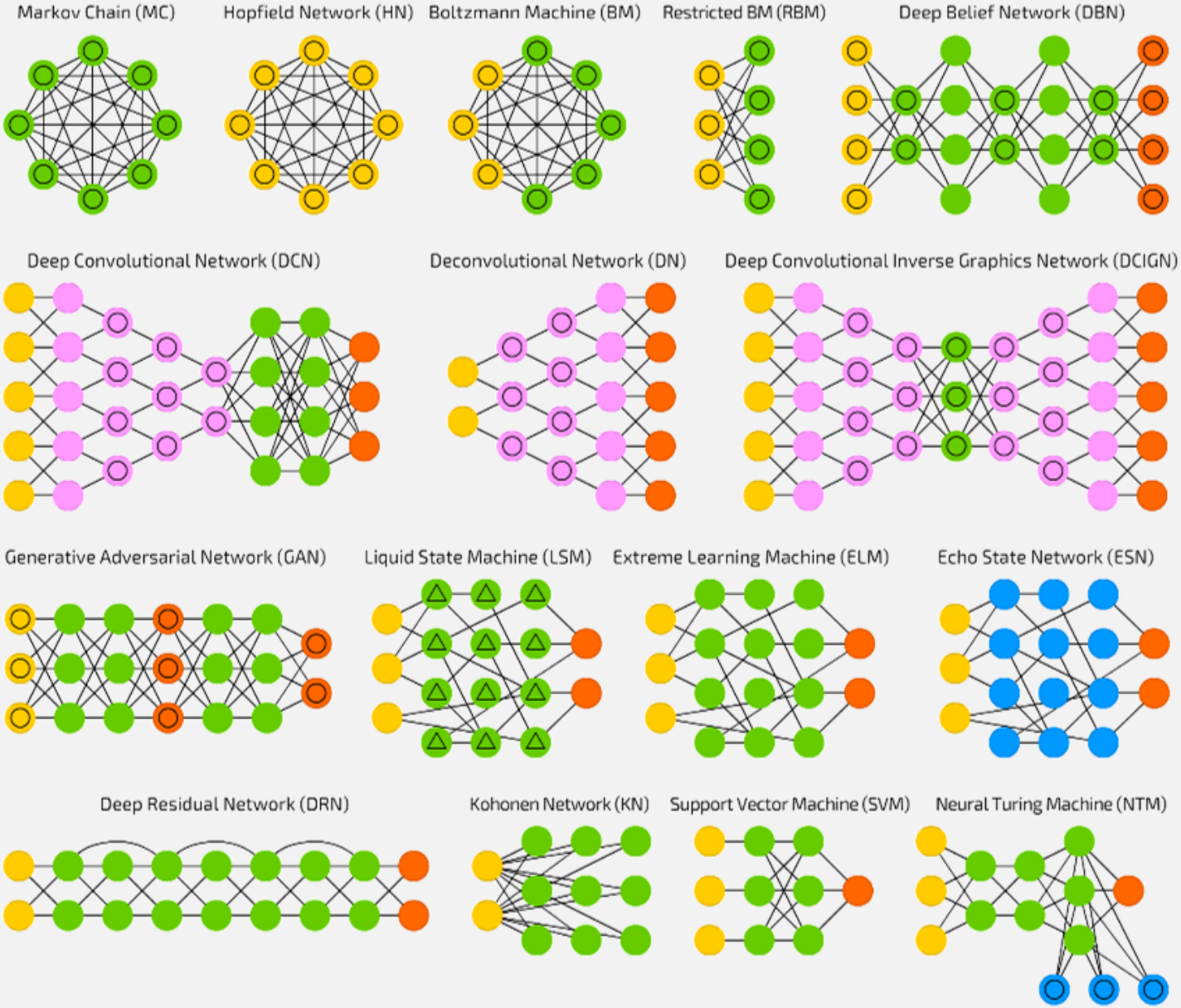


Different NN Architectures



Source: http://www.asimovinstitute.org/neural-network-zoo/

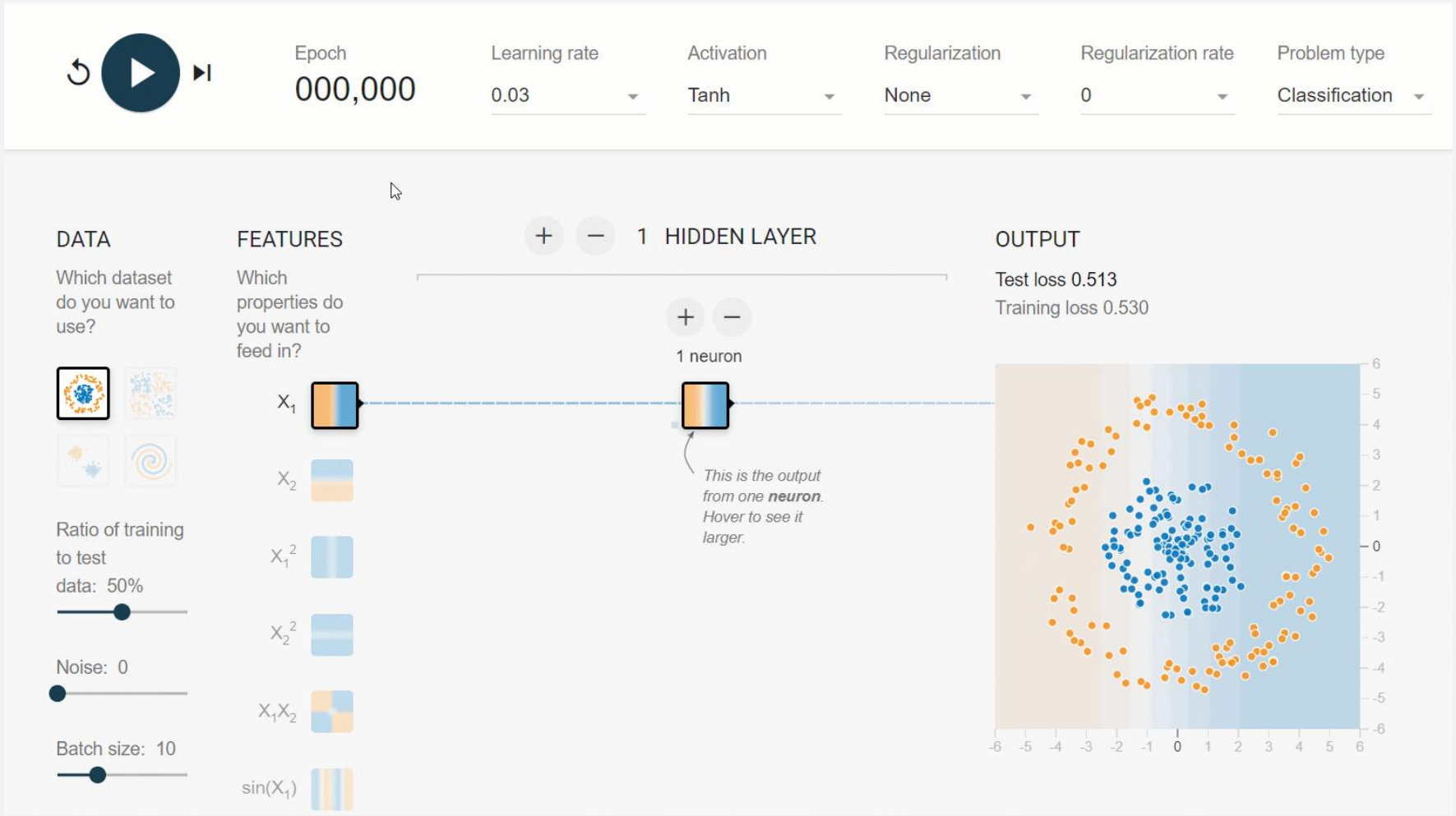
Different NN Architectures



Source: http://www.asimovinstitute.org/neural-network-zoo/

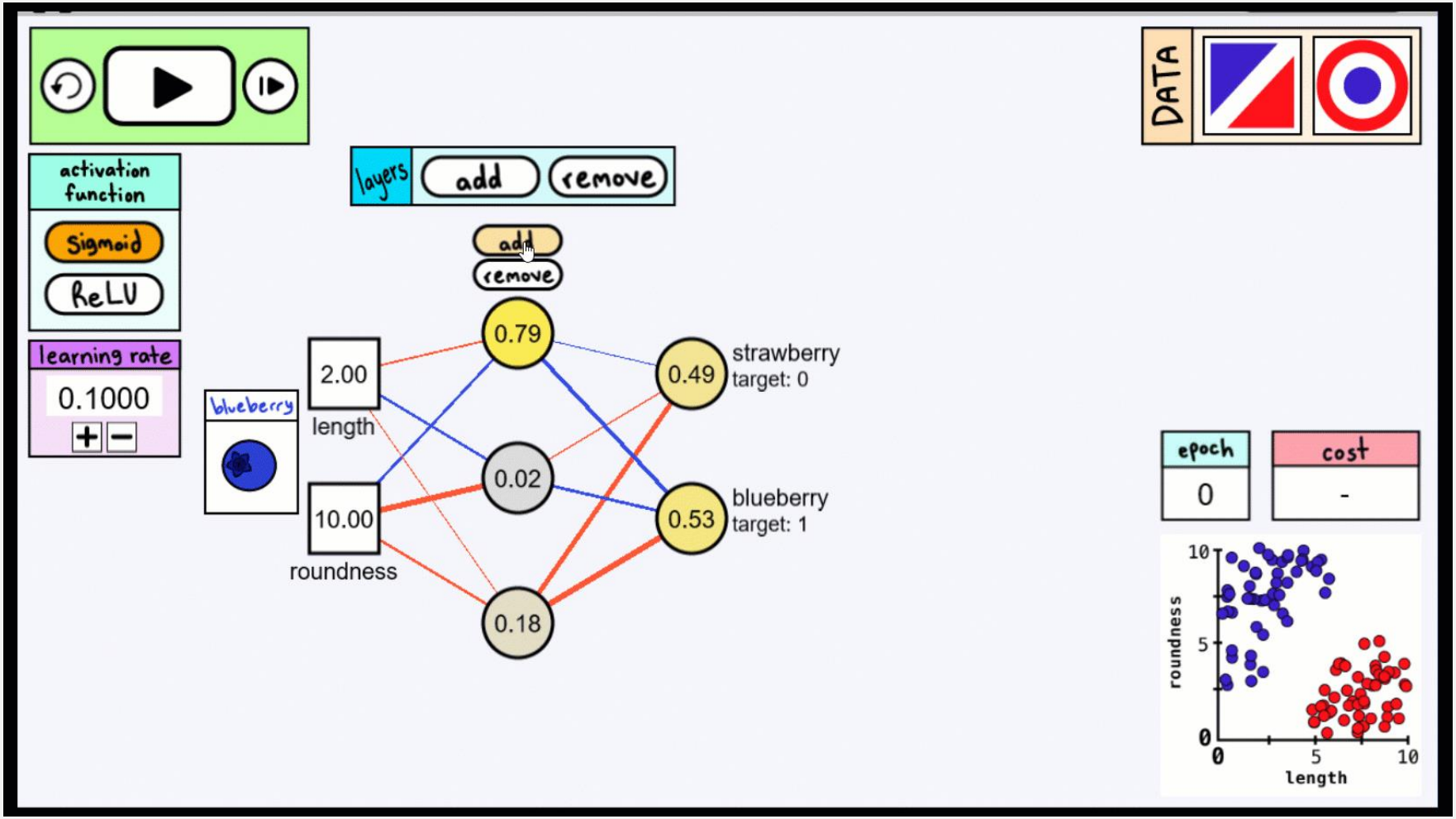
Resource: Neural Network Playground

[Neural Network Playground (Tensorflow)]



Resource: Neural Networks from Scratch

[Neural Networks from scratch]



NN Summary

- **Training Phase**: Learn weights using backpropagation
- **Model**: The weights
- **Prediction Phase**: Insert instance's feature values into input neurons and see what pops out in the output neurons
- **Decision boundary**: anything

NN: Pros and Cons

- Pros:
  - Great predictive performance
  - Best performance for unstructured data
    - Text, audio, images, video
  - Sexy (instant startup funding)
- Cons:
  - Hard to interpret
  - Hard to design good architecture
  - Needs lots of data
  - Long training time

Research on DL for Tabular Data

Tabular Data: Deep Learning is Not All You Need

Deep Neural Networks and Tabular Data: A Survey

Vadim Borisov[a,1], Tobias Leemann[a,1], Kathrin Seßler[a], Johannes Haug[a], Martin Pawelczyk[a] and Gjergji Kas

[a]The University o
[b]SCHUFA Holdin

ARTICLE I

Keywords:
Deep learning
Tabular data
Heterogeneous da
Discrete data
Tabular data gene
Probabilistic mod
Survey

[cs.LG] 5 Oct 2021

1. Introduc

The succes
based on conv
(Hochreiter an
(Vaswani et al.

Why do tree-based models still outperform deep learning on tabular data?

Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
ISIR, CNRS, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (~10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and Neural Networks (NNs). This leads to a series of challenges which should guide researchers aiming to build tabular-specific NNs: **1.** be robust to uninformative features, **2.** preserve the orientation of the data, and **3.** be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20 000 compute hours hyperparameter search for each learner.

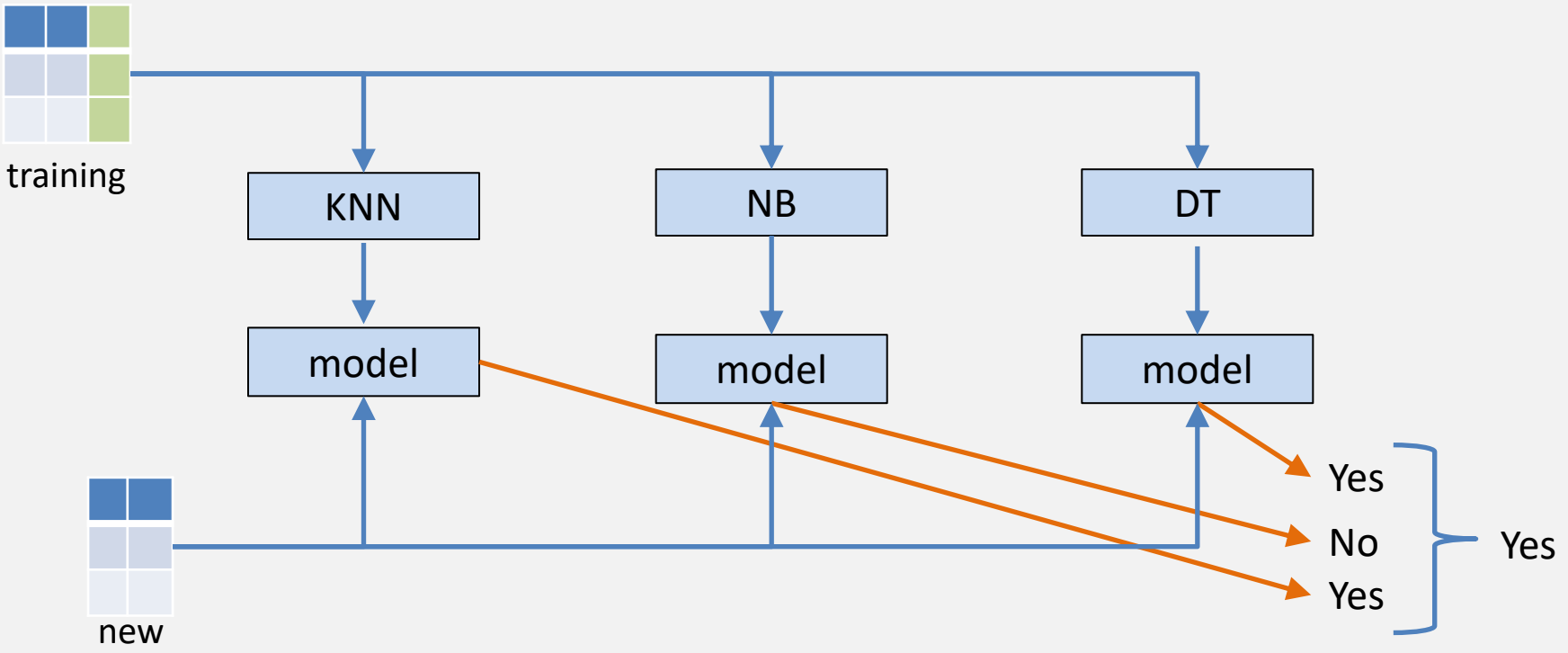
ENSEMBLES (PREVIEW)

Ensembles

- **Can we combine multiple models to produce a better model?**
  - Yes!
  - Called "ensembles"
- In practice, ensembles are ***very*** effective
- Many ways:
  - **Committee**, *aka* **Voting**
  - **Bagging** (incl. Random Forests, Extra Trees)
  - **Boosting** (incl. XGBoost, LGBM, CatBoost)
  - **Stacking**
- While you can create an ensemble manually, usually you just use a function/package like RandomForest or XGBoost

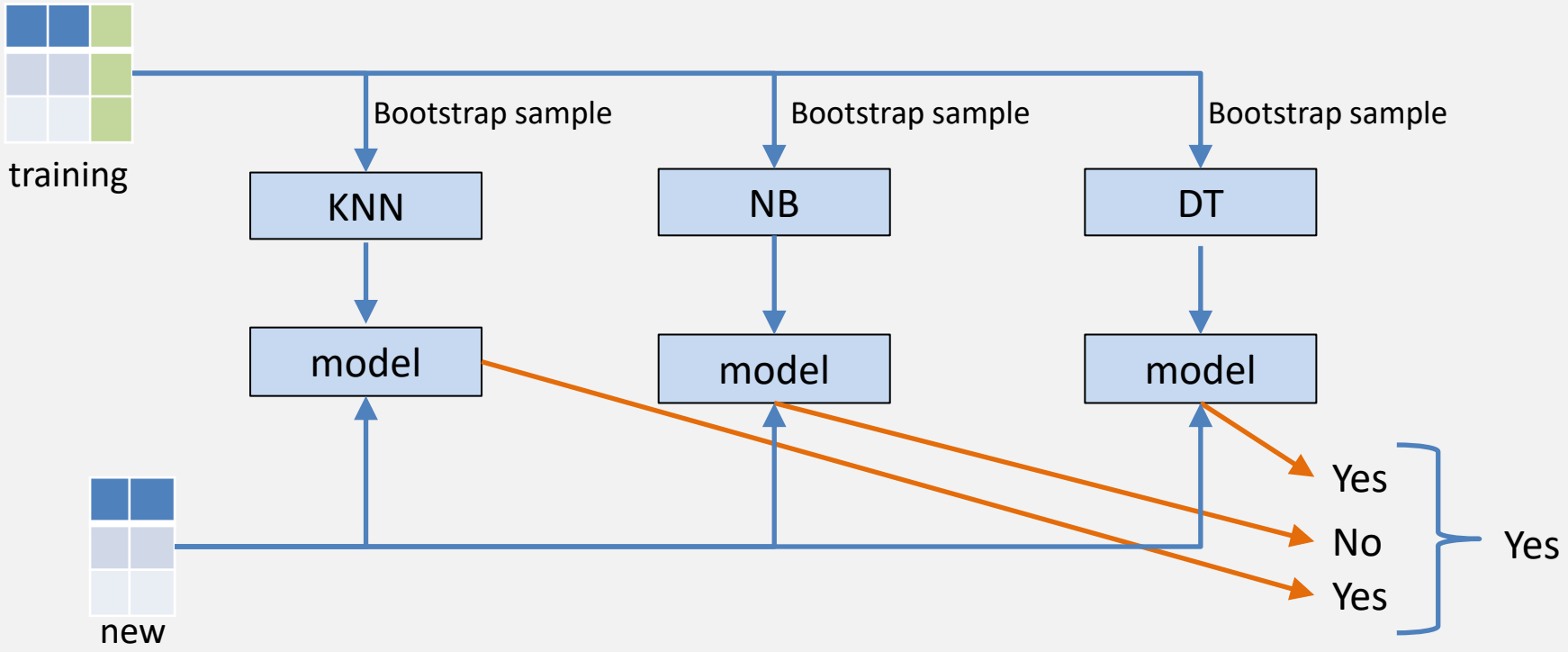
Committee

- Train several models as normal
  - Decision trees, NB, SVMs, whatever you want!
  - Each algorithm gets full training data as normal
- Prediction: Majority answer (or average) wins

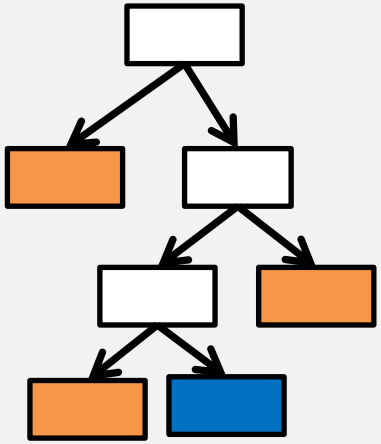


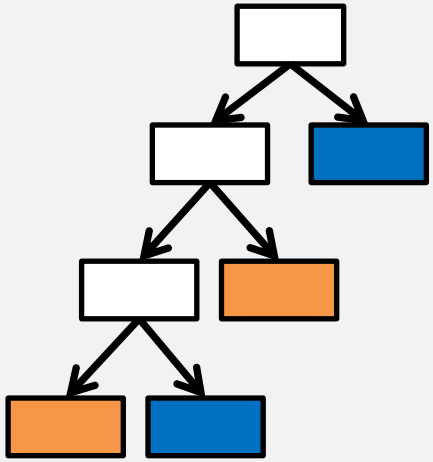
Bagging

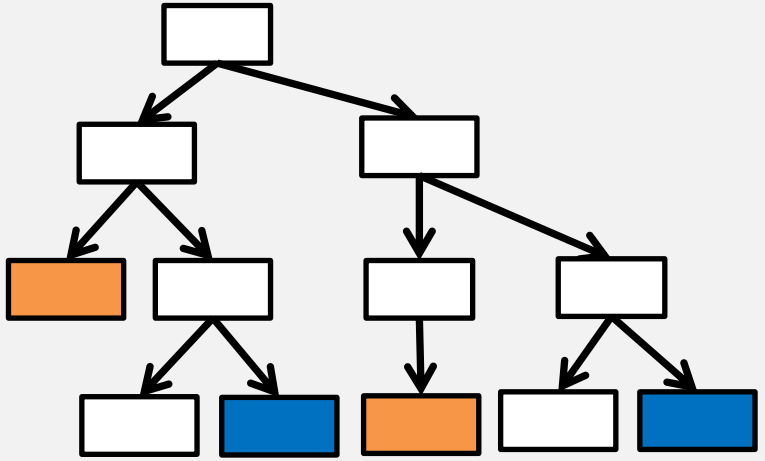
- ***Bagging*** is the same as committees, except:
  - Instead of getting the full training data, each model only gets a ***bootstrap sample*** of the training data
- Popular examples: Random Forests*, Extra Trees*
  - *A few other tweaks

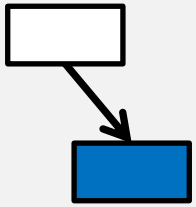


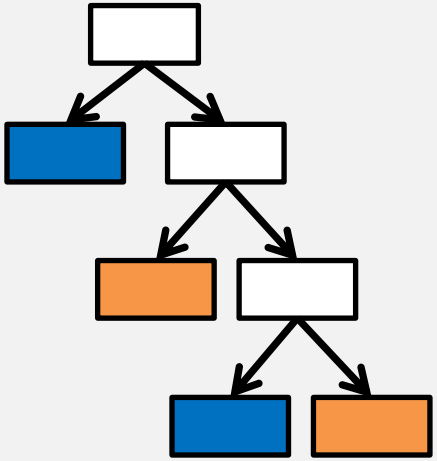
Tree Bagging

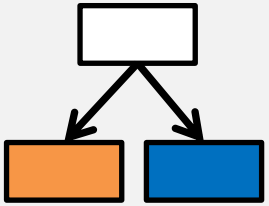






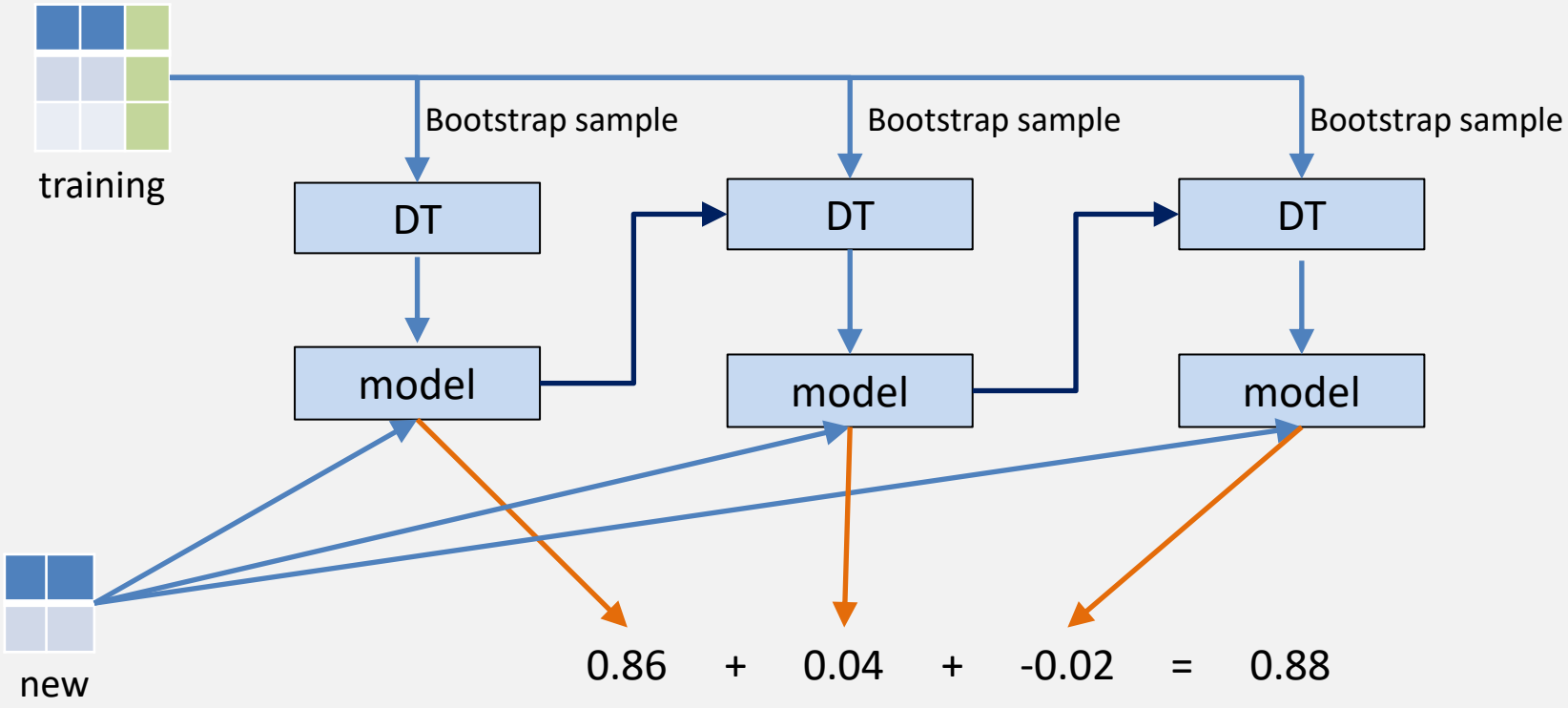






Boosting

- ***Boosting***: train models sequentially to improve existing models
- Very popular! Great results in many domains
- Popular implementations: XGBoost, LightGBM, CatBoost



Ensembles: Pros and Cons

- Pros:
  - Great predictive performance
  - Relatively fast
  - Usually tree-based, so some indication of feature importance
- Cons:
  - Harder to interpret
  - Lots of hyperparameters to tune

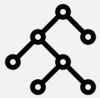
WHICH ALGORITHM SHOULD I USE?

Which Algorithm Should I Use?

- **Hard question!** Algorithms all differ in their:
  - Accuracy
  - Explainability: can you understand a given prediction?
  - Interpretability: can you understand what the model is doing?
  - Efficiency
    - Training speed and memory required
    - Prediction speed and memory required
  - Robustness: handling noise and missing values
  - …
- There is No Free Lunch®

Uncle Steve's Ultimate Comparison Guide




	KNN	LR	NB	DT	NN	SVM	BOOST
Accuracy on structured data	Lower	Medium	Lower	Lower	Medium	High	High
Accuracy on unstructured data	Lower	Medium	Medium	Medium	High	Medium	Medium
Explainability	Good	Good	Medium	Depends	Bad	Bad	Medium
Interpretability	Low	Good	Good	Medium	Bad	Bad	Medium
Training speed	Fast	Fast	Fast	Fast	Slow	Slow	Medium
Prediction speed	Slow	Fast	Fast	Fast	Fast	Fast	Medium
Hyperparameter tuning required	Low	Low	None	Medium	Huge	Medium	Medium
Works well with small data	No	Yes	Yes	No	No	Yes	Medium
Robust to irrelevant features	No	No	Yes	Yes	Yes	Yes	Yes
Learns interactions	No	No	No	Yes	Yes	Yes	Yes
Handles categorical features	Yes	No	Yes	Yes	No	No	Yes
Handles non-linear data	Yes	Possibly	No	Yes	Yes	Yes	Yes
Handles correlated features	No	No	No	Yes	Yes	No	Yes
Handles missing values	Yes	No	Yes	Yes	No	No	Yes

Uncle Steve's TLDR: Which is Best?

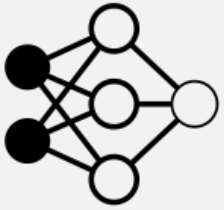
Best for **tabular data**:

Boosting

Best for **tabular data**
(if you need explainability):

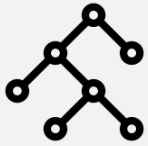
Logistic Regression

Best for **unstructured data:**



Neural Networks

TLDR: What about the others?

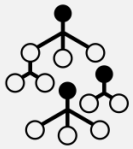


Decision Trees → Accuracy not high enough

KNN → Slow, accuracy not high enough

Naïve Bayes → Too many assumptions, accuracy not high enough

Support Vector Machine → Way too slow



Random Forest → Good, but not as good as boosting

Applied Predictive Modeling's Guide

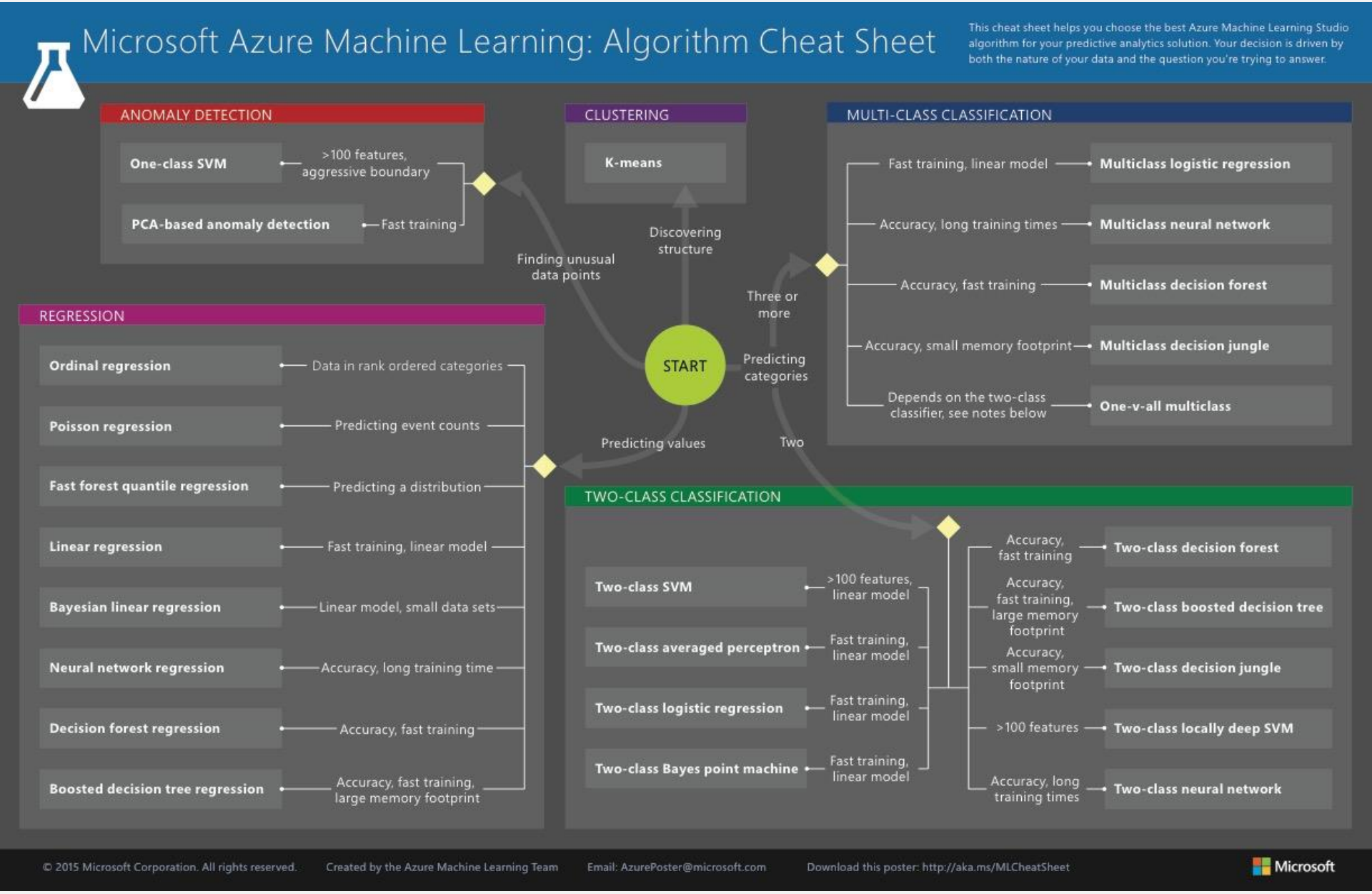
Model	Allows $n < p$	Pre-processing	Interpretable	Automatic feature selection	# Tuning parameters	Robust to predictor noise	Computation time
Linear regression†	×	CS, NZV, Corr	✓	×	0	×	✓
Partial least squares	✓	CS	✓	○	1	×	✓
Ridge regression	×	CS, NZV	✓	×	1	×	✓
Elastic net/lasso	×	CS, NZV	✓	✓	1–2	×	✓
Neural networks	✓	CS, NZV, Corr	×	×	2	×	×
Support vector machines	✓	CS	×	×	1–3	×	×
MARS/FDA	✓		○	✓	1–2	○	○
K-nearest neighbors	✓	CS, NZV	×	×	1	○	✓
Single trees	✓		○	✓	1	✓	✓
Model trees/rules†	✓		○	✓	1–2	✓	✓
Bagged trees	✓		×	✓	0	✓	○
Random forest	✓		×	○	0–1	✓	×
Boosted trees	✓		×	✓	3	✓	×
Cubist†	✓		×	○	2	✓	×
Logistic regression*	×	CS, NZV, Corr	✓	×	0	×	✓
{LQRM}DA*	×	NZV	○	×	0–2	×	✓
Nearest shrunken centroids*	✓	NZV	○	✓	1	×	✓
Naïve Bayes*	✓	NZV	×	×	0–1	○	○
C5.0*	✓		○	✓	0–3	✓	×

†regression only *classification only

Symbols represent affirmative (✓), negative (×), and somewhere in between (○)

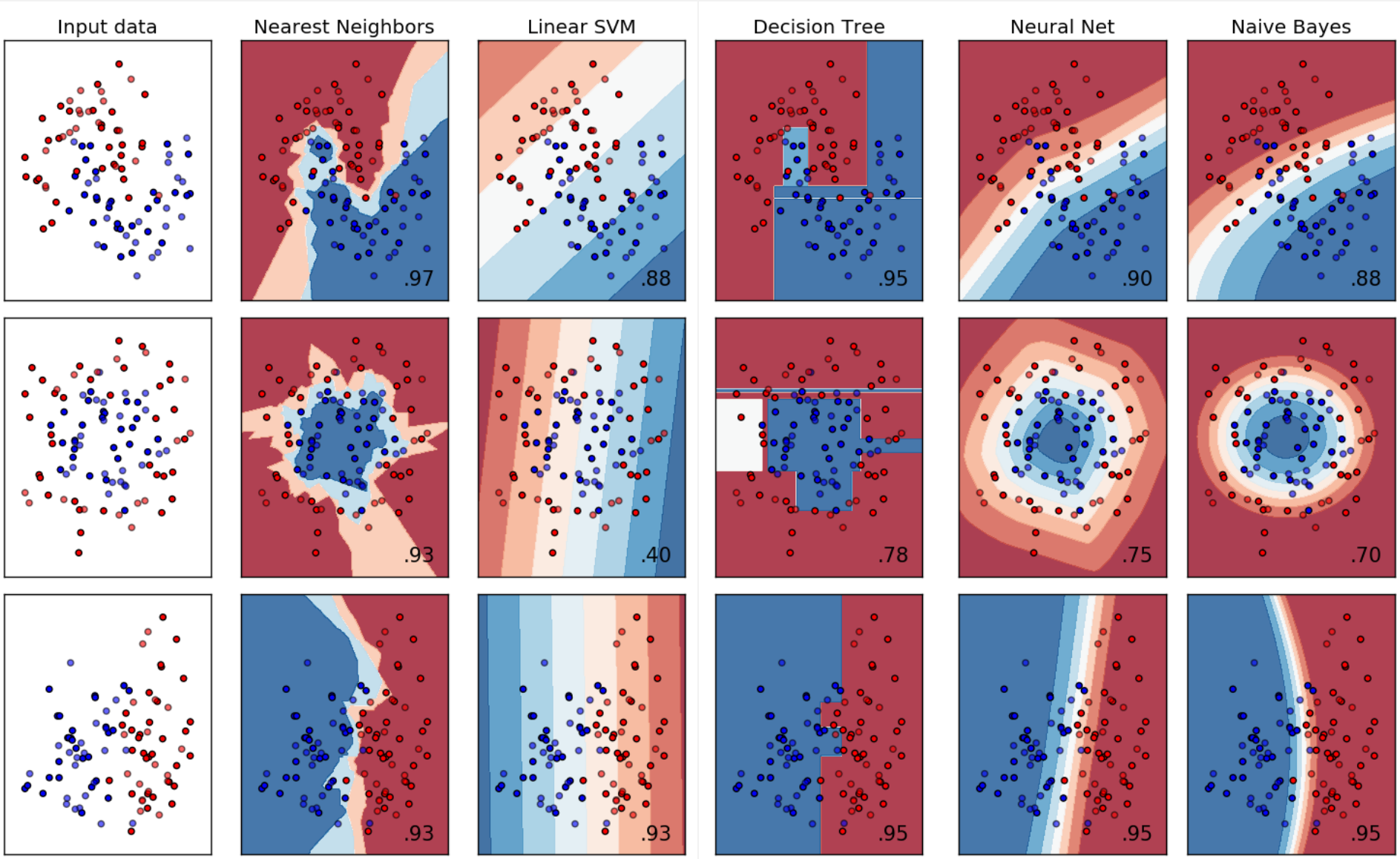
CS = centering and scaling; NZV = remove near-zero predictors; Corr = remove highly correlated predictors

Microsoft Azure's Flowchart



Created by the Azure Machine Learning Team
Email: AzurePoster@microsoft.com
Download this poster: http://aka.ms/MLCheatSheet

Scikit-Learn's Visual Comparison

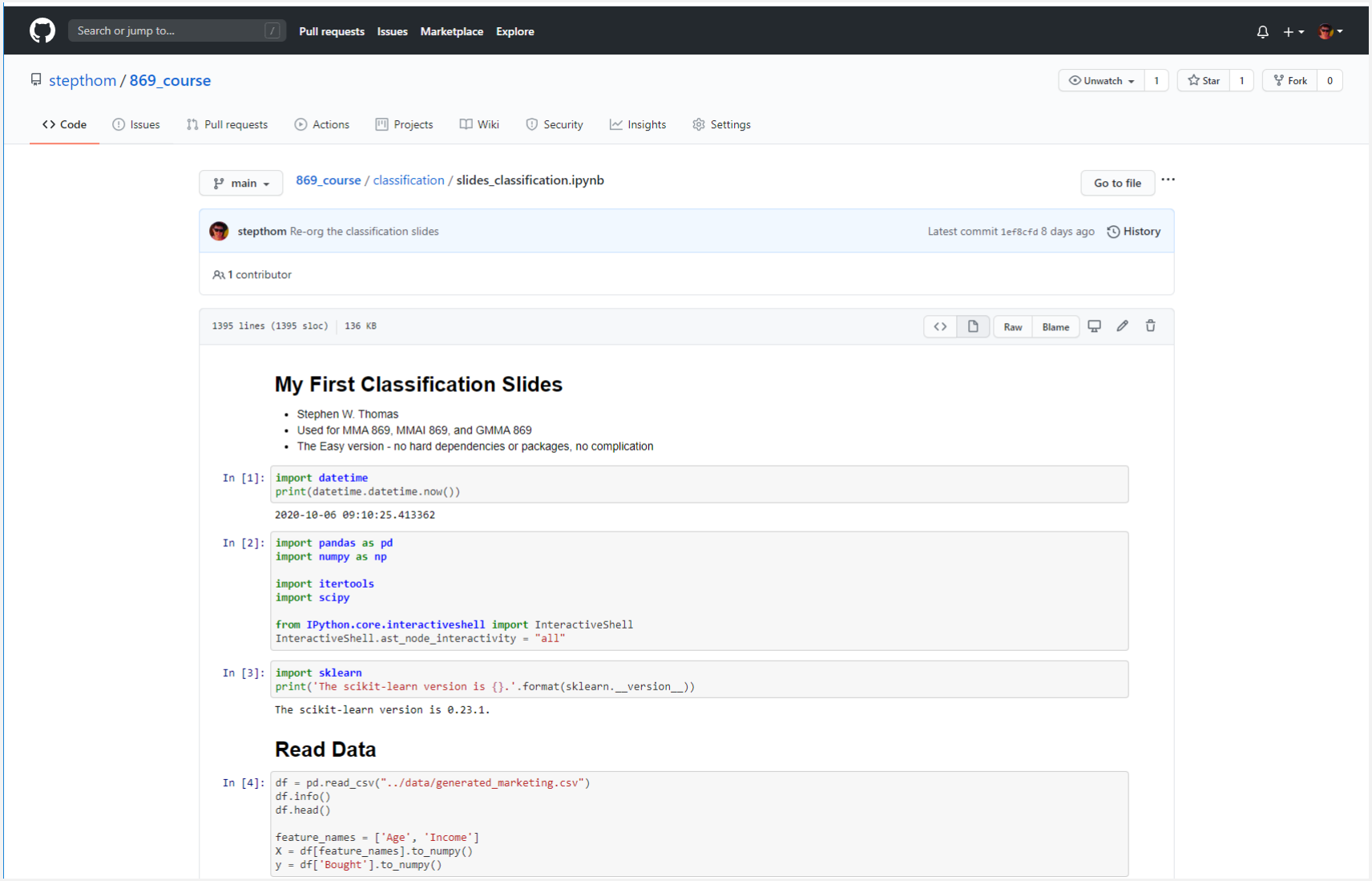


Production Line Exercise

- Want to build a prediction model for factory QA
- The model will predict whether widgets are "defective" or "not"
- The model will be integrated into factory's production line
- Assume features of the widgets are gathered in real time
- Design goals:
  - The model does not slow down the production line
  - The possibility of defective widgets being passed by the model should be minimized
  - If the model makes a mistake, it should be re-trained immediately
  - If the model makes a mistake, it should be possible for operators to query the model to understand why the prediction was made
  - A large set of historical labeled data is available for training
- Discuss the different issues that should be taken into account when evaluating the suitability of different classification algorithms
- For this task, discuss the suitability of the (a) decision tree, (b) KNN, (c) Naïve Bayes, (d) SVM, and (e) NN models. Which one is most appropriate?

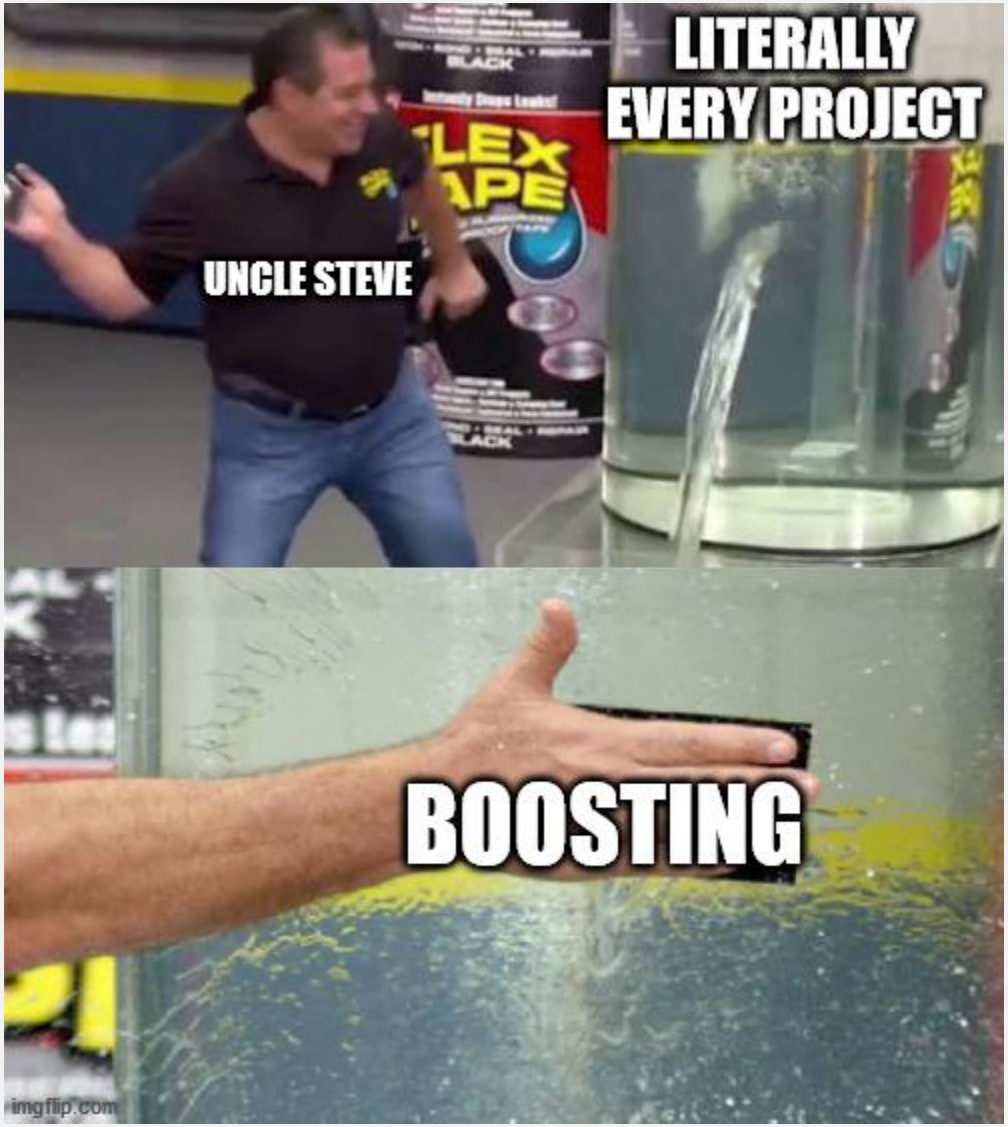
RESOURCES

Resources



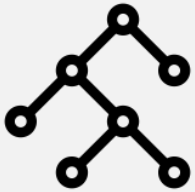
https://github.com/stepthom/869_course/blob/main/classification/slides_classification.ipynb

SUMMARY


UNCLE STEVE
LITERALLY
EVERY PROJECT
BOOSTING
imgflip.com

Summary

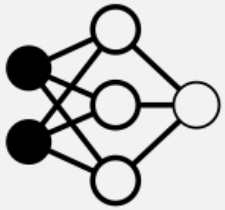
- ***Classification:*** supervised ML algorithms to build models to predict categorical features
- ***Algorithms***:


Decision Trees


Naïve Bayes


KNN


Support Vector Machine


Neural Networks

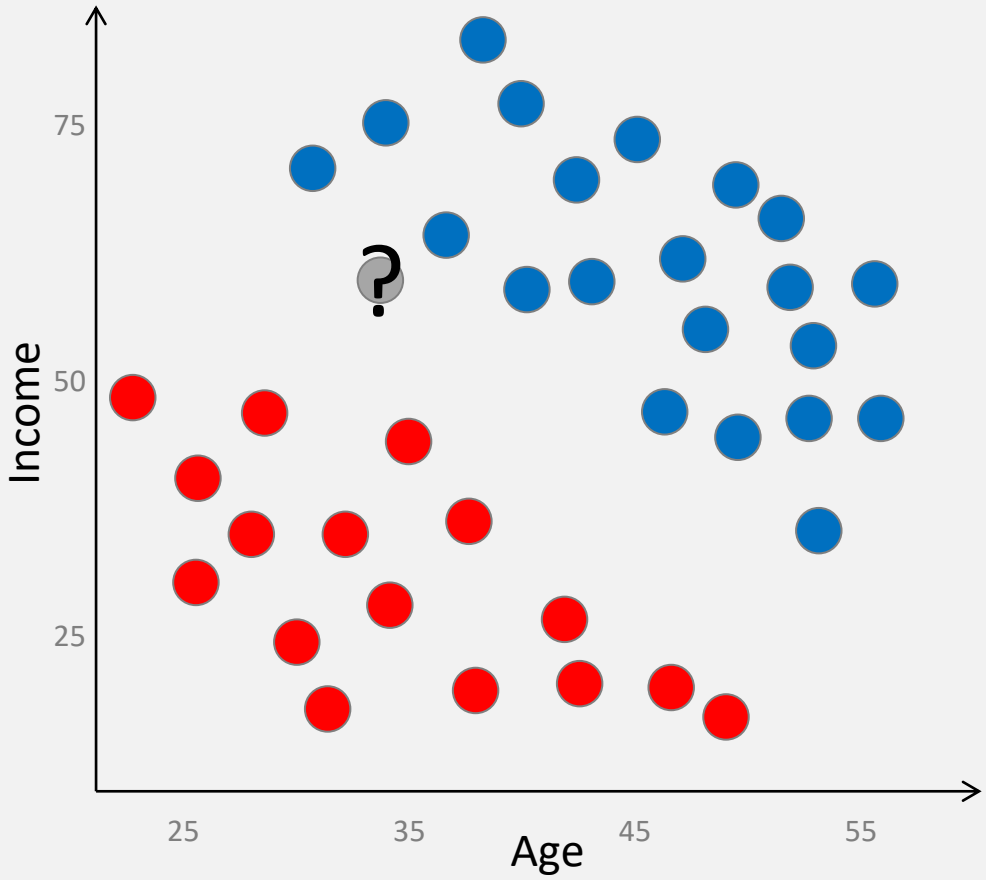
- ***Which algorithm to use***?
  - Depends - There's No Free Lunch
- Still coming:
  - Performance measurement
  - Hyperparameter tuning
  - Ensembles
  - Feature engineering
  - Feature selection
  - Class imbalance
  - Pipelines

APPENDIX

Exercise #2

- You work at a local credit union.
- A customer (Susy) walks into your branch and applies for a $20,000 auto loan. Susy is 30 years old and makes $59K per year.
- **Question**: Should you give Susy the loan?

Age	Income	Paid Back
49.0	57.0	yes
39.0	66.0	yes
50.0	20.0	no
42.0	55.0	yes
42.0	34.0	no
51.0	34.0	yes
35.0	39.0	no
32.0	32.0	no
40.0	32.0	no
27.0	60.0	yes
44.0	29.0	no
46.0	36.0	yes
39.0	30.0	no
50.0	24.0	no
48.0	51.0	yes



Quick Primer: Abstract Notation

- $x = x^{(1)}, \ldots, x^{(m)}$: Feature vectors
- $y = y^{(1)}, \ldots, y^{(m)}$: Truth target values
- $z = z^{(1)}, \ldots, z^{(m)}$: Predicted target values
- A classification algorithm takes x and y and builds a model to predict z from x

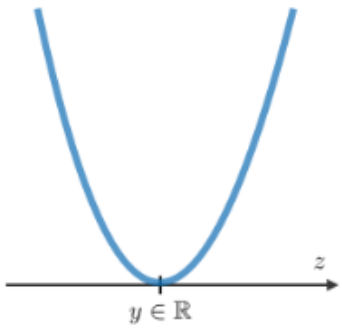
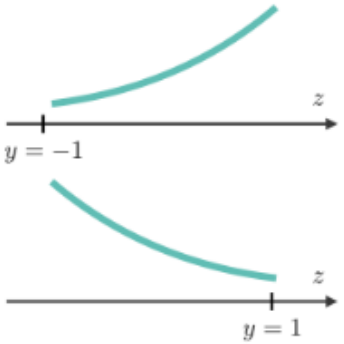
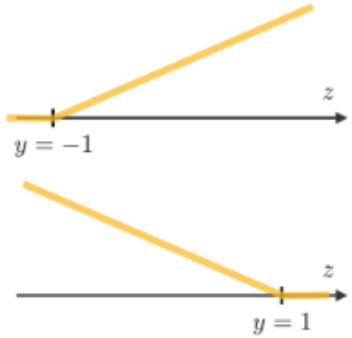
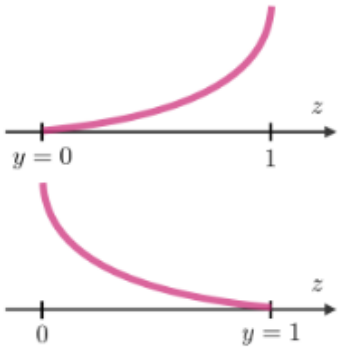
	Age	Income	Married	Citizenship
$x^{(1)}$	55	36,765	True	Canada
$x^{(2)}$	66	87,983	True	Canada
$x^{(3)}$	21	24,354	False	USA
$x^{(4)}$	24	56,654	True	Canada
$x^{(5)}$	34	98,324	False	UK
$x^{(6)}$	36	132,229	False	Germany
$x^{(7)}$	28	35,000	True	Canada
$x^{(8)}$	49	50,334	True	Canada

	Truth
$y^{(1)}$	True
$y^{(2)}$	True
$y^{(3)}$	False
$y^{(4)}$	False
$y^{(5)}$	False
$y^{(6)}$	True
$y^{(7)}$	False
$y^{(8)}$	False

	Predicted
$z^{(1)}$	True
$z^{(2)}$	False
$z^{(3)}$	True
$z^{(4)}$	True
$z^{(5)}$	False
$z^{(6)}$	True
$z^{(7)}$	False
$z^{(8)}$	True

Quick Primer: Loss Functions

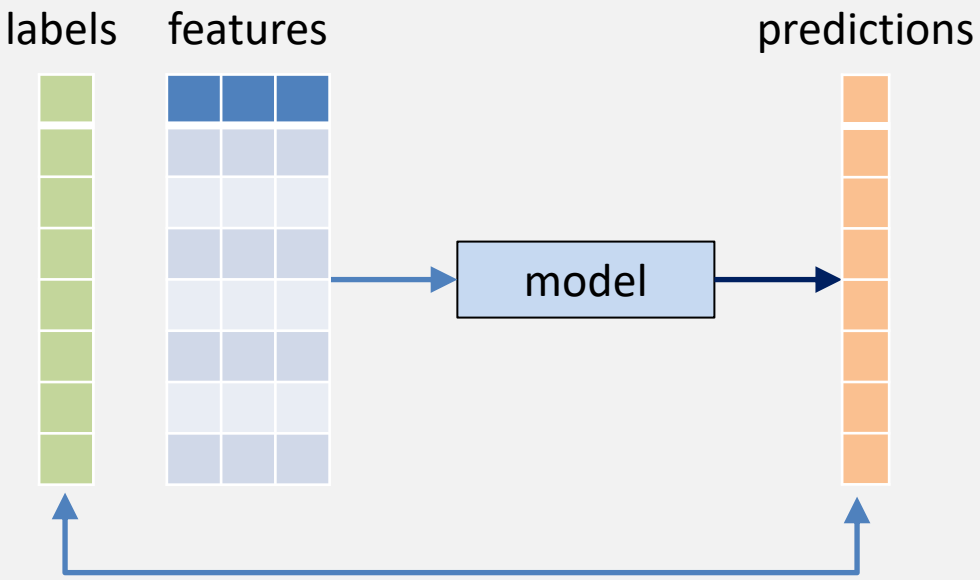
- All ML algorithms seek to minimize a ***loss function***
- An algorithm uses the loss function to iteratively fit the data
  - The loss function says how different the predictions z are from the truth y
- There are many loss functions. Examples:
  - Least squared error (used by regression)
  - Information gain (used by decision trees)
  - Joint likelihood (used by Naïve Bayes)
  - SVM Loss

Least squared error	Logistic loss	Hinge loss	Cross-entropy
$\frac{1}{2}(y-z)^2$	$\log(1+\exp(-yz))$	$\max(0, 1-yz)$	$-\left[y\log(z)+(1-y)\log(1-z)\right]$
			
Linear regression	Logistic regression	SVM	Neural Network

Stanford CS 229

Quick Primer: Performance Metrics

- How good are a model's predictions?

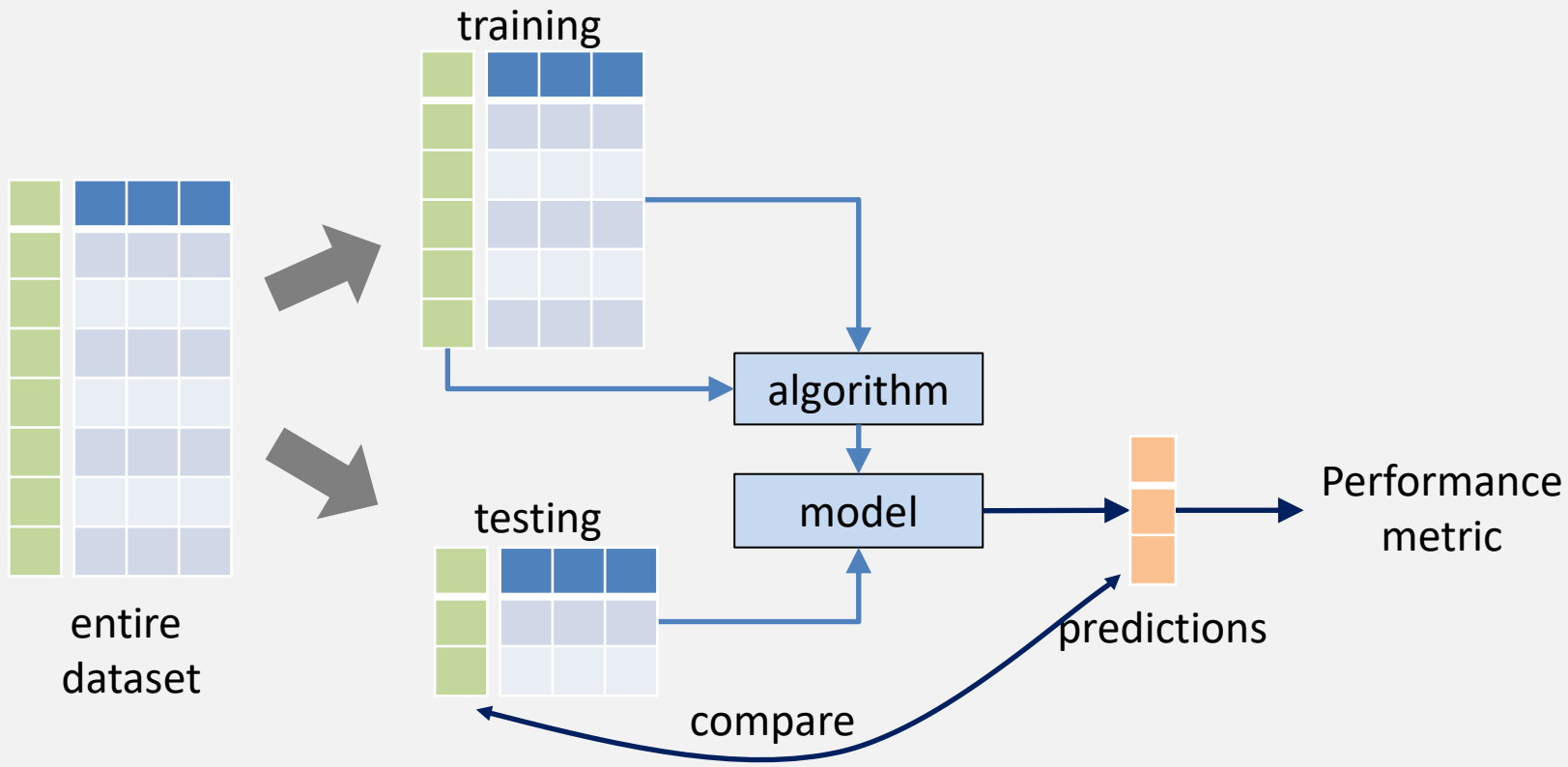


Compare using metric:

- Accuracy and Error
- Precision, Recall, and F1 score
- Sensitivity and Specificity
- ROC Curve and AUC
- Log Loss

Quick Primer: Model Validation

- How can we tell if a model will perform well in the future?
- Typical process:
  - Randomly split data into training and testing data sets
  - Train the model using training set; evaluation using test set
- (Covered in next lecture)

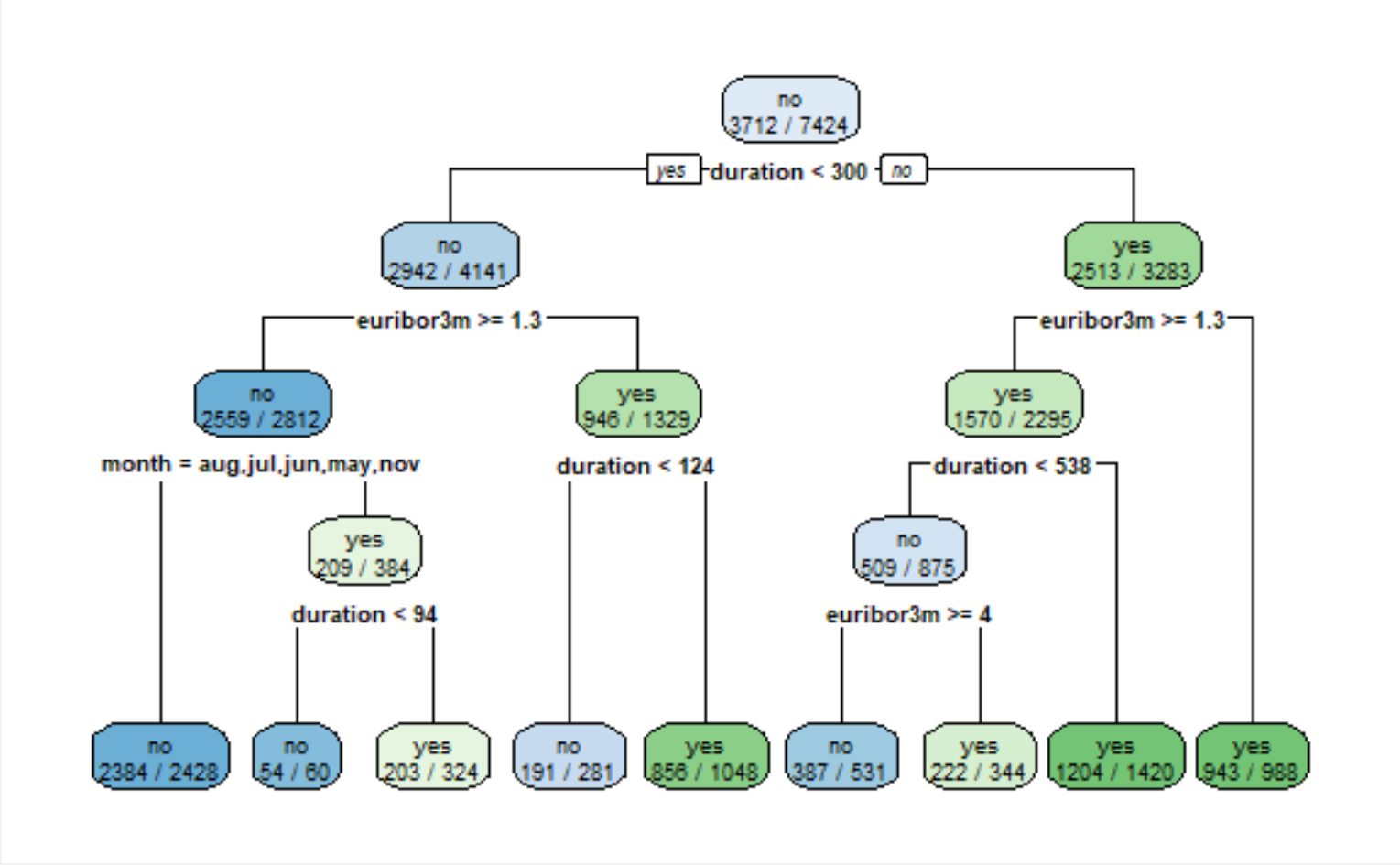


DT

Example

```
train.index <- createDataPartition(df$bought, p = .8, list = FALSE)
train <- df[ train.index,]
test  <- df[-train.index,]
```

```
tree <- rpart(bought ~ ., method="class", data=train)
rpart.plot(tree, extra=2, type=2)
```



DT: Missing Values

- Decision trees can handle missing values
- Just use present values for impurity calculation

age
young
young
young
young
young
old
young
young
old
young
young
young

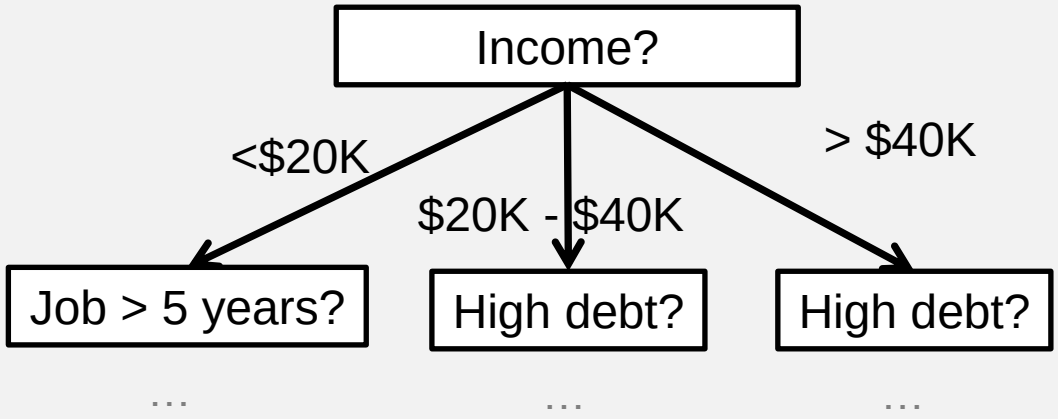
Example

```
pred = predict(tree, test, type="class")
confusionMatrix(data=pred, reference=test$bought,
                positive=positive,
                dnn=c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

         Actual
Predicted  no yes
      no  763  79
      yes 165 849

               Accuracy : 0.8685
                 95% CI : (0.8523, 0.8836)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7371
 Mcnemar's Test P-Value : 5.281e-08

            Sensitivity : 0.9149
            Specificity : 0.8222
         Pos Pred Value : 0.8373
         Neg Pred Value : 0.9062
             Prevalence : 0.5000
         Detection Rate : 0.4574
   Detection Prevalence : 0.5463
      Balanced Accuracy : 0.8685

       'Positive' Class : yes
```

DT: Trees with Multi Way Splits

- Most algorithms split each node two ways
- However, some algorithms can split a node multiple ways



NB

Naïve Bayes

- Another popular classification algorithm
  - Fast
  - Easy to understand
  - Performs pretty well
- Based on Bayes Theorem of conditional probability

$$p(B|A) = \frac{p(A|B)p(B)}{p(A)}$$

Remember Bayes Theorem?

- Bayesian reasoning is revising beliefs in the light of new evidence
- Quick refresher:
  - A doctor knows that meningitis causes stiff neck 50% of the time
  - Prior probability of any patient having meningitis is 1/50,000
  - Prior probability of any patient having stiff neck is 1/20
- If a patient has stiff neck, what's the probability he/she has meningitis?

$$P(M \mid S) = \frac{P(S \mid M)P(M)}{P(S)} = \frac{0.5 \times 1/50000}{1/20} = 0.0002$$

Note: For NB classifiers, it's common to ignore the denominator, since it will be the same for each target, and we only care about the relative values, not the absolute values.

Training Phase

- Algorithm computes distributions of each feature/class
  - Numeric features: probability distribution
  - Categorical features: Counts/tabulations
- The distributions are the "model"
- Future predictions are made by plugging numbers into the Bayes Theorem

Training Phase Example

- Algorithm calculates all the necessary probabilities

$$P(\text{status} = \text{paid}) = \frac{9}{15} \qquad P(\text{status} = \text{default}) = \frac{6}{15}$$

has_job	own_house	credit_rating	status
TRUE	FALSE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
FALSE	TRUE	excellent	paid
TRUE	TRUE	fair	paid
TRUE	FALSE	good	paid
TRUE	FALSE	good	paid
FALSE	TRUE	good	paid
TRUE	TRUE	good	paid

$$P(\text{has\_job=FALSE} \mid \text{status} = \text{paid}) = \frac{4}{9}$$
$$P(\text{has\_job=TRUE} \mid \text{status} = \text{paid}) = \frac{5}{9}$$
$$P(\text{own\_house=FALSE} \mid \text{status} = \text{paid}) = \frac{3}{9}$$
$$P(\text{own\_house=TRUE} \mid \text{status} = \text{paid}) = \frac{6}{9}$$
$$P(\text{credit\_rating=excellent} \mid \text{status} = \text{paid}) = \frac{4}{9}$$
$$P(\text{credit\_rating=good} \mid \text{status} = \text{paid}) = \frac{4}{9}$$
$$P(\text{credit\_rating=fair} \mid \text{status} = \text{paid}) = \frac{1}{9}$$

has_job	own_house	credit_rating	status
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	fair	default
FALSE	FALSE	good	default
FALSE	FALSE	good	default

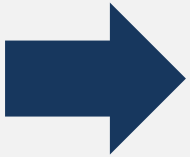
$$P(\text{has\_job=FALSE} \mid \text{status} = \text{default}) = \frac{6}{6}$$
$$P(\text{has\_job=TRUE} \mid \text{status} = \text{default}) = \frac{0}{6}$$
$$P(\text{own\_house=FALSE} \mid \text{status} = \text{default}) = \frac{6}{6}$$
$$P(\text{own\_house=TRUE} \mid \text{status} = \text{default}) = \frac{0}{6}$$
$$P(\text{credit\_rating=excellent} \mid \text{status} = \text{default}) = \frac{0}{6}$$
$$P(\text{credit\_rating=good} \mid \text{status} = \text{default}) = \frac{2}{6}$$
$$P(\text{credit\_rating=fair} \mid \text{status} = \text{default}) = \frac{4}{6}$$

Additive Smoothing

- Some probabilities are 0! Yuck.
- Trick: use ***additive smoothing***. (Just add 1 to all entries.)

$P(\text{has\_job=FALSE} \mid \text{status} = \text{paid}) = \frac{4}{9}$
$P(\text{has\_job=TRUE} \mid \text{status} = \text{paid}) = \frac{5}{9}$
$P(\text{own\_house=FALSE} \mid \text{status} = \text{paid}) = \frac{3}{9}$
$P(\text{own\_house=TRUE} \mid \text{status} = \text{paid}) = \frac{6}{9}$
$P(\text{credit\_rating=excellent} \mid \text{status} = \text{paid}) = \frac{4}{9}$
$P(\text{credit\_rating=good} \mid \text{status} = \text{paid}) = \frac{4}{9}$
$P(\text{credit\_rating=fair} \mid \text{status} = \text{paid}) = \frac{1}{9}$

$P(\text{has\_job=FALSE} \mid \text{status} = \text{default}) = \frac{6}{6}$
$P(\text{has\_job=TRUE} \mid \text{status} = \text{default}) = \frac{0}{6}$
$P(\text{own\_house=FALSE} \mid \text{status} = \text{default}) = \frac{6}{6}$
$P(\text{own\_house=TRUE} \mid \text{status} = \text{default}) = \frac{0}{6}$
$P(\text{credit\_rating=excellent} \mid \text{status} = \text{default}) = \frac{0}{6}$
$P(\text{credit\_rating=good} \mid \text{status} = \text{default}) = \frac{2}{6}$
$P(\text{credit\_rating=fair} \mid \text{status} = \text{default}) = \frac{4}{6}$



$P(\text{has\_job=FALSE} \mid \text{status} = \text{paid}) = \frac{5}{11}$
$P(\text{has\_job=TRUE} \mid \text{status} = \text{paid}) = \frac{6}{11}$
$P(\text{own\_house=FALSE} \mid \text{status} = \text{paid}) = \frac{4}{11}$
$P(\text{own\_house=TRUE} \mid \text{status} = \text{paid}) = \frac{7}{11}$
$P(\text{credit\_rating=excellent} \mid \text{status} = \text{paid}) = \frac{5}{12}$
$P(\text{credit\_rating=good} \mid \text{status} = \text{paid}) = \frac{5}{12}$
$P(\text{credit\_rating=fair} \mid \text{status} = \text{paid}) = \frac{2}{12}$

$P(\text{has\_job=FALSE} \mid \text{status} = \text{default}) = \frac{7}{8}$
$P(\text{has\_job=TRUE} \mid \text{status} = \text{default}) = \frac{1}{8}$
$P(\text{own\_house=FALSE} \mid \text{status} = \text{default}) = \frac{7}{8}$
$P(\text{own\_house=TRUE} \mid \text{status} = \text{default}) = \frac{1}{8}$
$P(\text{credit\_rating=excellent} \mid \text{status} = \text{default}) = \frac{1}{9}$
$P(\text{credit\_rating=good} \mid \text{status} = \text{default}) = \frac{3}{9}$
$P(\text{credit\_rating=fair} \mid \text{status} = \text{default}) = \frac{5}{9}$

Prediction Phase Example

- Suppose a new instance is:
  - has_job=TRUE, own_house = FALSE, and credit_rating = good
- To make a prediction, use Bayes Theorem twice (i.e., once for each target level)

P(status=default | has_job=TRUE,own_house=FALSE,credit_rating=good)

= P(status=default) * = 6/15 * = 0.015
P(has_job=TRUE | status=default) * 1/8 *
P(own_house=FALSE| status=default) * 7/8 *
P(credit_rating=good| status=default) 3/9

Predict: status=paid

P(status=paid | has_job=TRUE,own_house=FALSE,credit_rating=good)

= P(status=paid) * = 9/15 * = 0.049
P(has_job=TRUE | status=paid) * 6/11 *
P(own_house=FALSE| status=paid) * 4/11 *
P(credit_rating=good| status=paid) 5/12

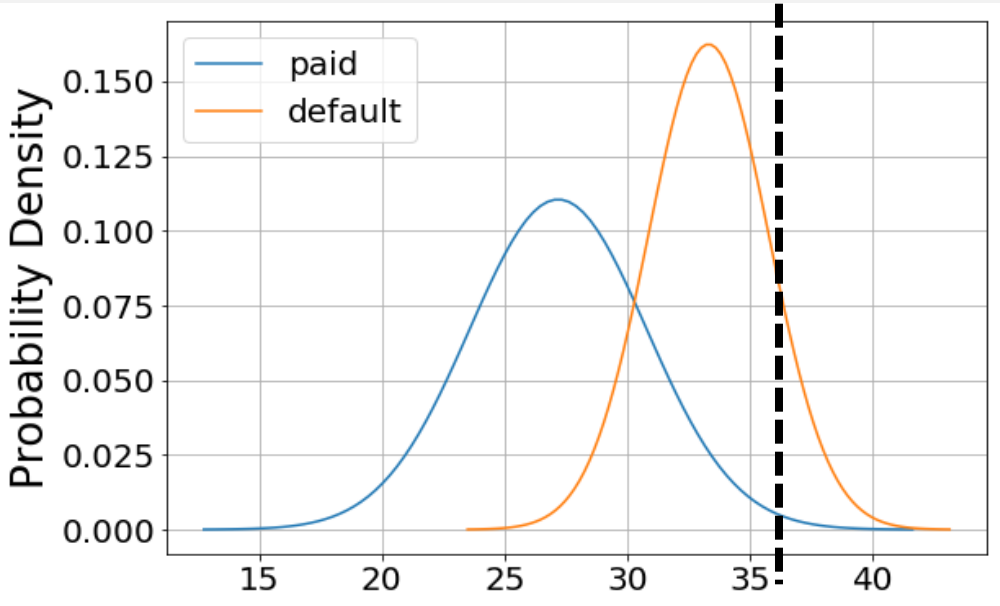
Example of Numeric Feature

age	status
27	paid
24	paid
28	paid
33	paid
24	paid
24	paid
33	paid
29	paid
23	paid

mean = μ = 27.1
std = σ = 3.6

age	status
37	default
34	default
34	default
36	default
29	default
30	default
33	default
32	default
36	default

mean = μ = 33.3
std = σ = 2.4



$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Suppose Suzy is 36 years old.

$$P(\text{age}=36 \mid \text{status} = \text{paid}) = \frac{1}{3.6\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{36-27.1}{3.6}\right)^2} = 0.005$$

$$P(\text{age}=36 \mid \text{status} = \text{default}) = \frac{1}{2.4\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{36-33.3}{2.4}\right)^2} = 0.085$$

Naïve Bayes: Pros and Cons

- Pros:
  - Easy to implement
  - Very efficient, can handle huge data easily
  - Good results obtained in many applications
  - Can handle missing values
  - Resistant to overfitting
- Cons:
  - Makes a strong assumption: features are independent

Naïve Bayes Summary

- **Model**: Probabilities for each class/feature
- **Training Phase**: Calculate probabilities from training data
- **Prediction Phase**: Apply Bayes Theorem

Example

Bank Marketing Dataset: 20 features, 9280 instances

```
nb_fit = naiveBayes(bought ~ ., data=train)
nb_fit
```

```
A-priori probabilities:
Y
 no yes
0.5 0.5

Conditional probabilities:
     age
Y         [,1]      [,2]
  no  39.84052  9.867126
  yes 40.85911 13.910397

     job
Y          admin. blue-collar entrepreneur   housemaid  management     retired
  no  0.248383621 0.226562500  0.035829741 0.024784483 0.074892241 0.029094828
  yes 0.293642241 0.136314655  0.026400862 0.026131466 0.068965517 0.092133621
     job
Y     self-employed    services     student  technician  unemployed     unknown
  no    0.034213362 0.099676724 0.018588362 0.174568966 0.024515086 0.008890086
  yes   0.031250000 0.070851293 0.059267241 0.155172414 0.031788793 0.008081897

     marital
Y        divorced     married      single     unknown
  no  0.115301724 0.602370690 0.280172414 0.002155172
  yes 0.099676724 0.550377155 0.347252155 0.002693966
```

...

Example

```
pred = predict(nb_fit, test, type="class")
confusionMatrix(data=pred, reference=test$bought,
                positive=positive,
                dnn=c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

          Actual
Predicted  no yes
      no  805 267
      yes 123 661

               Accuracy : 0.7899
                 95% CI : (0.7706, 0.8082)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5797
 Mcnemar's Test P-Value : 4.451e-13

            Sensitivity : 0.7123
            Specificity : 0.8675
         Pos Pred Value : 0.8431
         Neg Pred Value : 0.7509
             Prevalence : 0.5000
         Detection Rate : 0.3561
   Detection Prevalence : 0.4224
      Balanced Accuracy : 0.7899

       'Positive' Class : yes
```

Marketing Dataset Example

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=42, criterion="entropy",
                             min_samples_split=10, min_samples_leaf=10, max_depth=3, max_leaf_nodes=5)
clf.fit(X_train, y_train)

y_pred_dt = clf.predict(X_test)
```

```
clf.predict_proba([[2, 2]])
clf.predict([[2, 2]])
```

```
array([[0.93922652, 0.06077348]])
```

Marketing Dataset Example

```
from pandas_ml import ConfusionMatrix

print(ConfusionMatrix(y_test, y_pred_dt))
```

```
Predicted  False  True  __all__
Actual
False         50     5       55
True           5    40       45
__all__       55    45      100
```

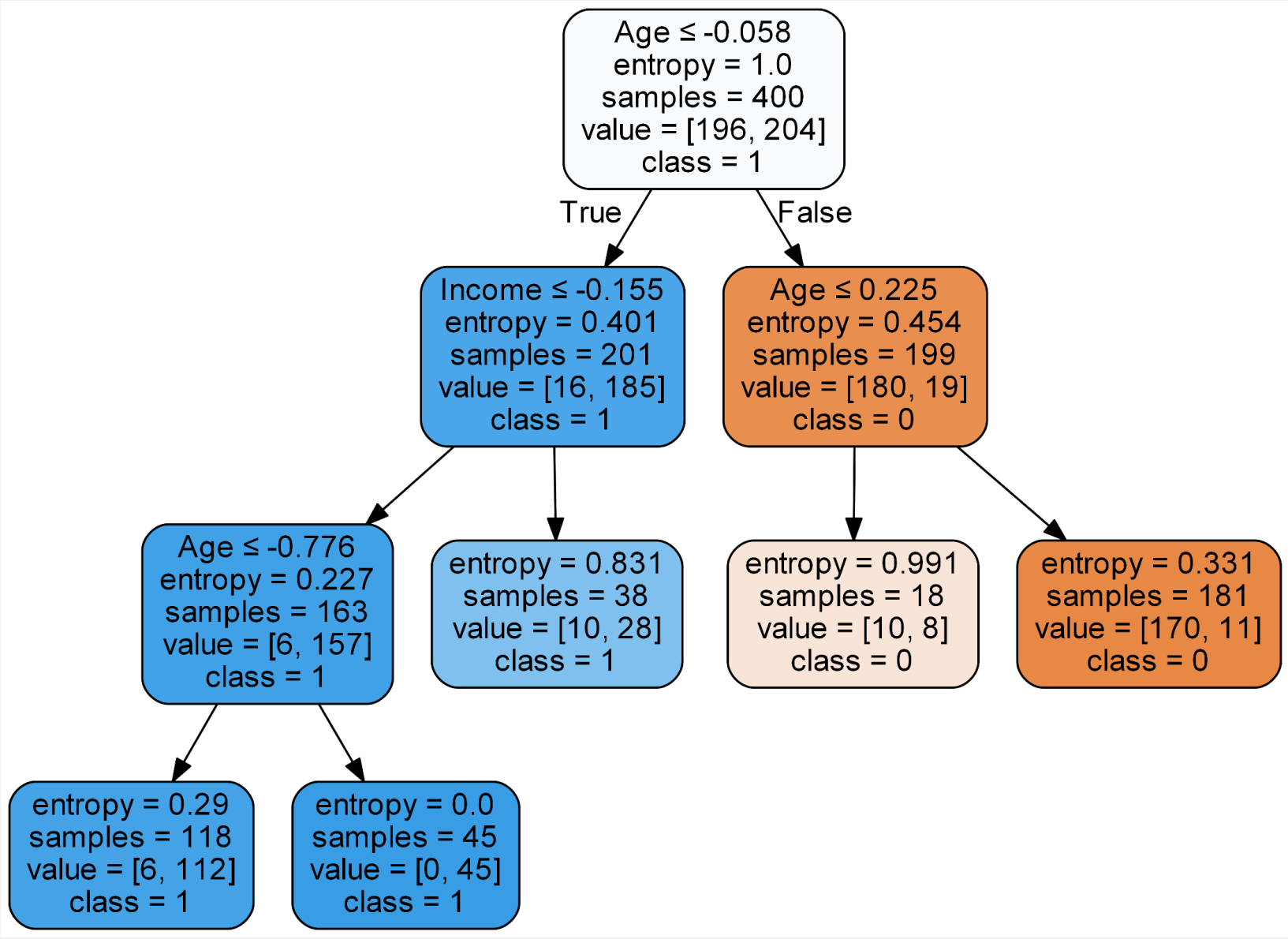
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_dt, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.91      0.91      0.91        55
          1       0.89      0.89      0.89        45

avg / total       0.90      0.90      0.90       100
```

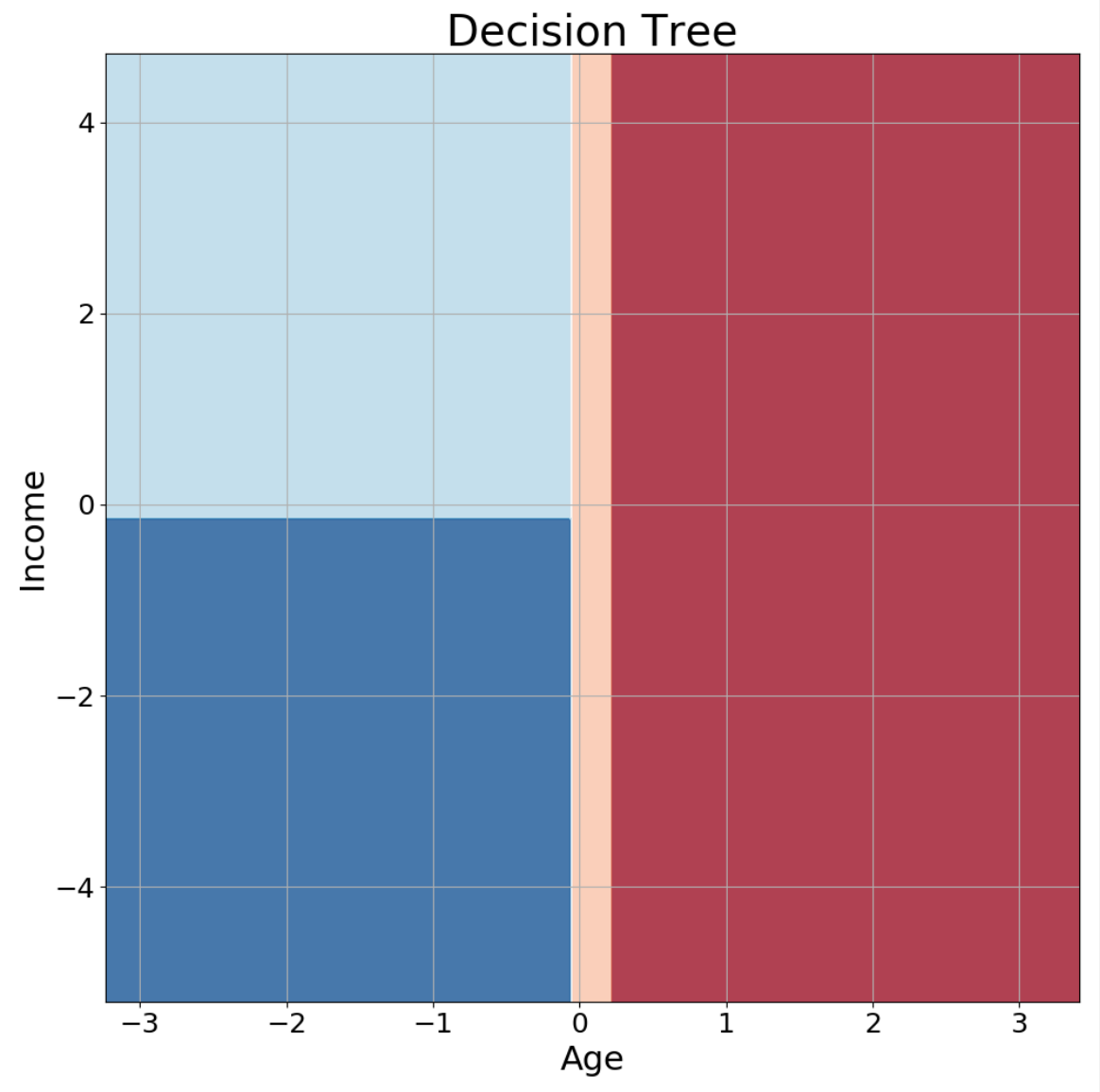
```
from sklearn.metrics import accuracy_score, cohen_kappa_score, f1_score, log_loss

print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_dt)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_dt)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_dt)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_dt)))
```

```
Accuracy = 0.90
Kappa = 0.80
F1 Score = 0.89
Log Loss = 3.45
```

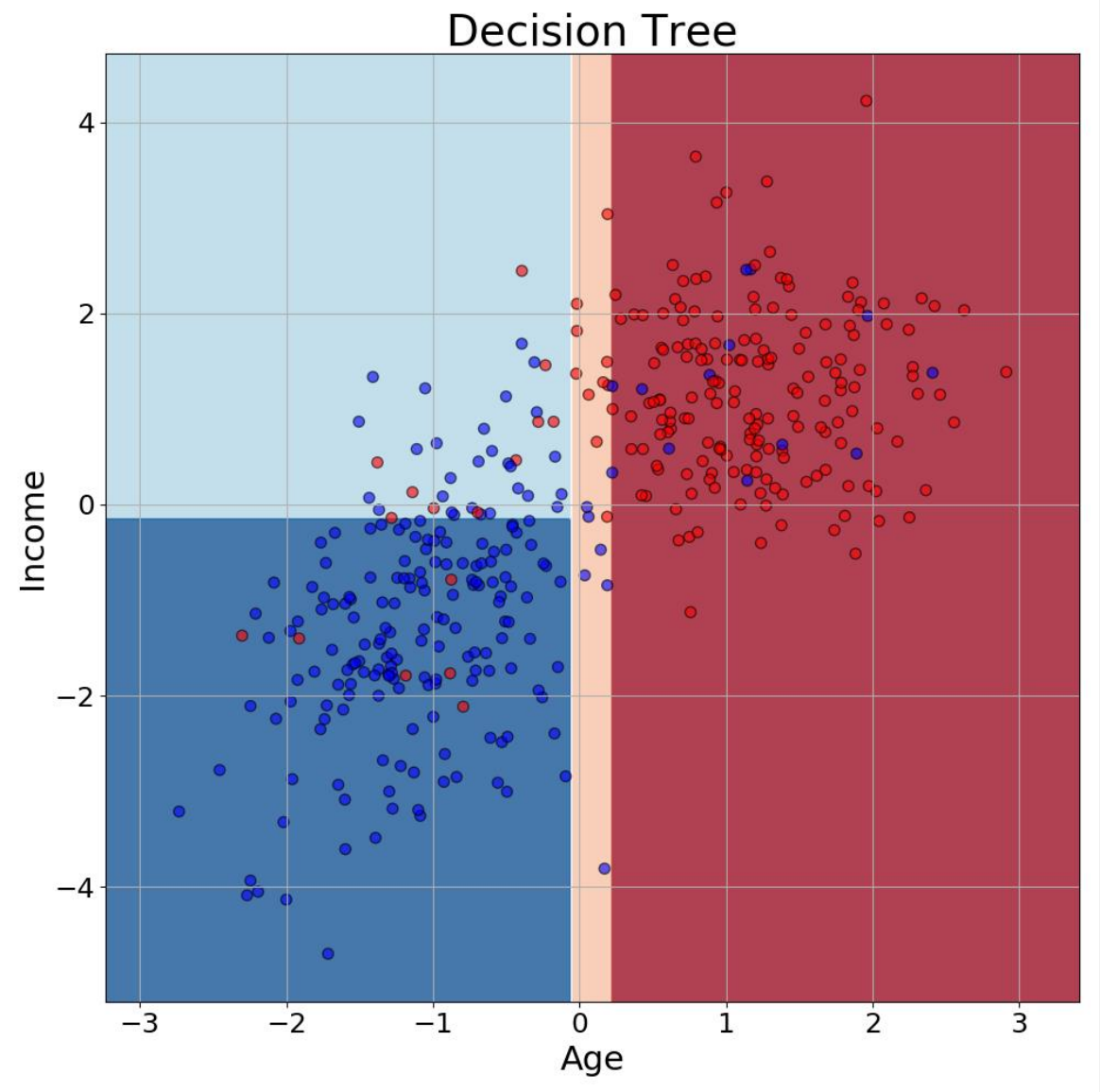
Marketing Dataset Example



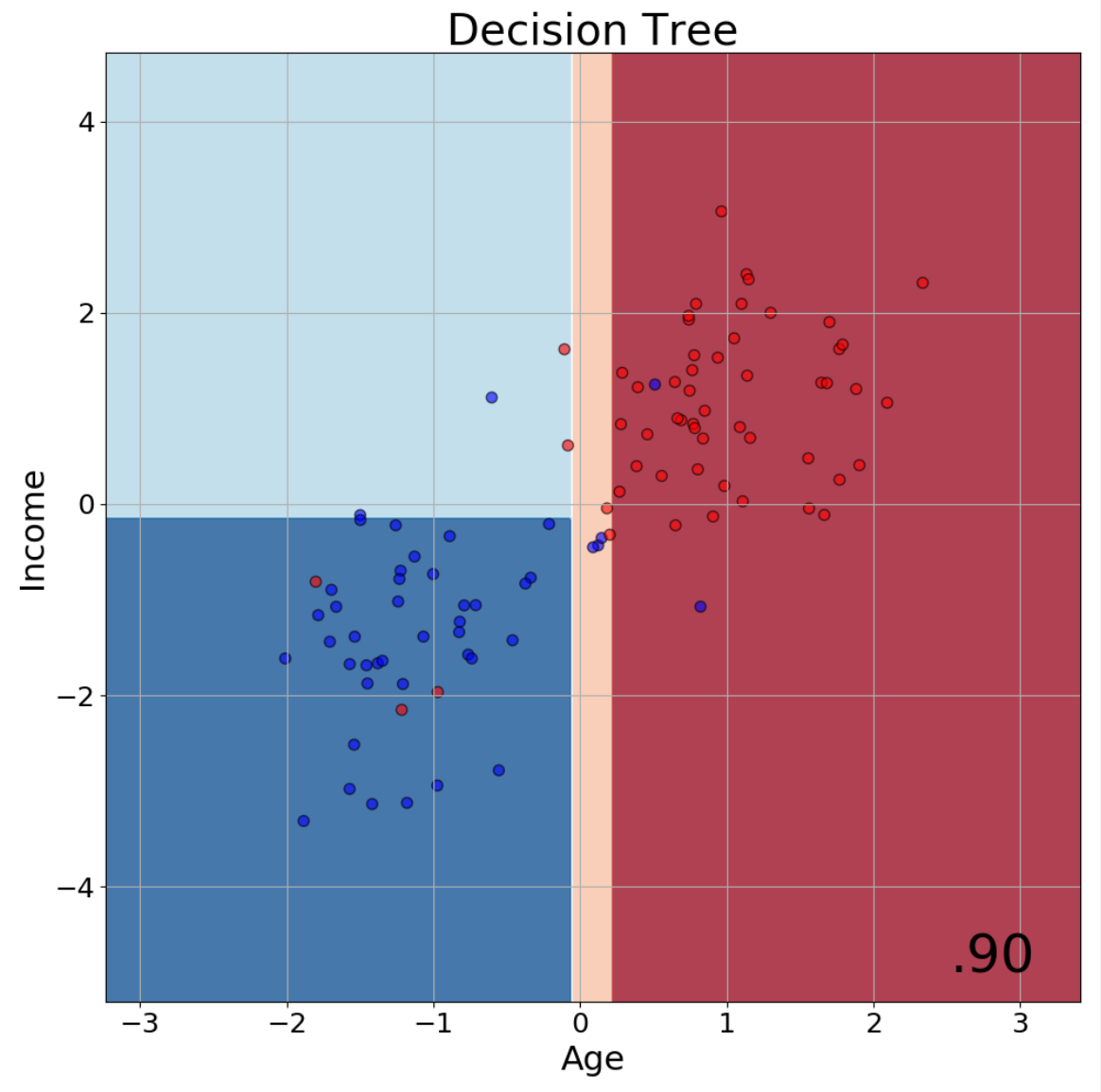
Decision Boundaries



Decision Boundaries (Training Data Overlaid)



Decision Boundaries (Testing Data Overlaid)



Marketing Dataset Example

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb = gnb.fit(X_train, y_train)
gnb

y_pred_gnb = gnb.predict(X_test)
```

```
gnb.theta_ # Mean of each feature per class
gnb.sigma_ # Variance of each feature per class
```

```
array([[ 0.99887777,  1.06512859],
       [-0.91827636, -1.08561939]])
array([[0.70142311, 0.949339  ],
       [0.62737657, 1.6885857 ]])
```

Marketing Dataset Example

```
print(ConfusionMatrix(y_test, y_pred_gnb))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

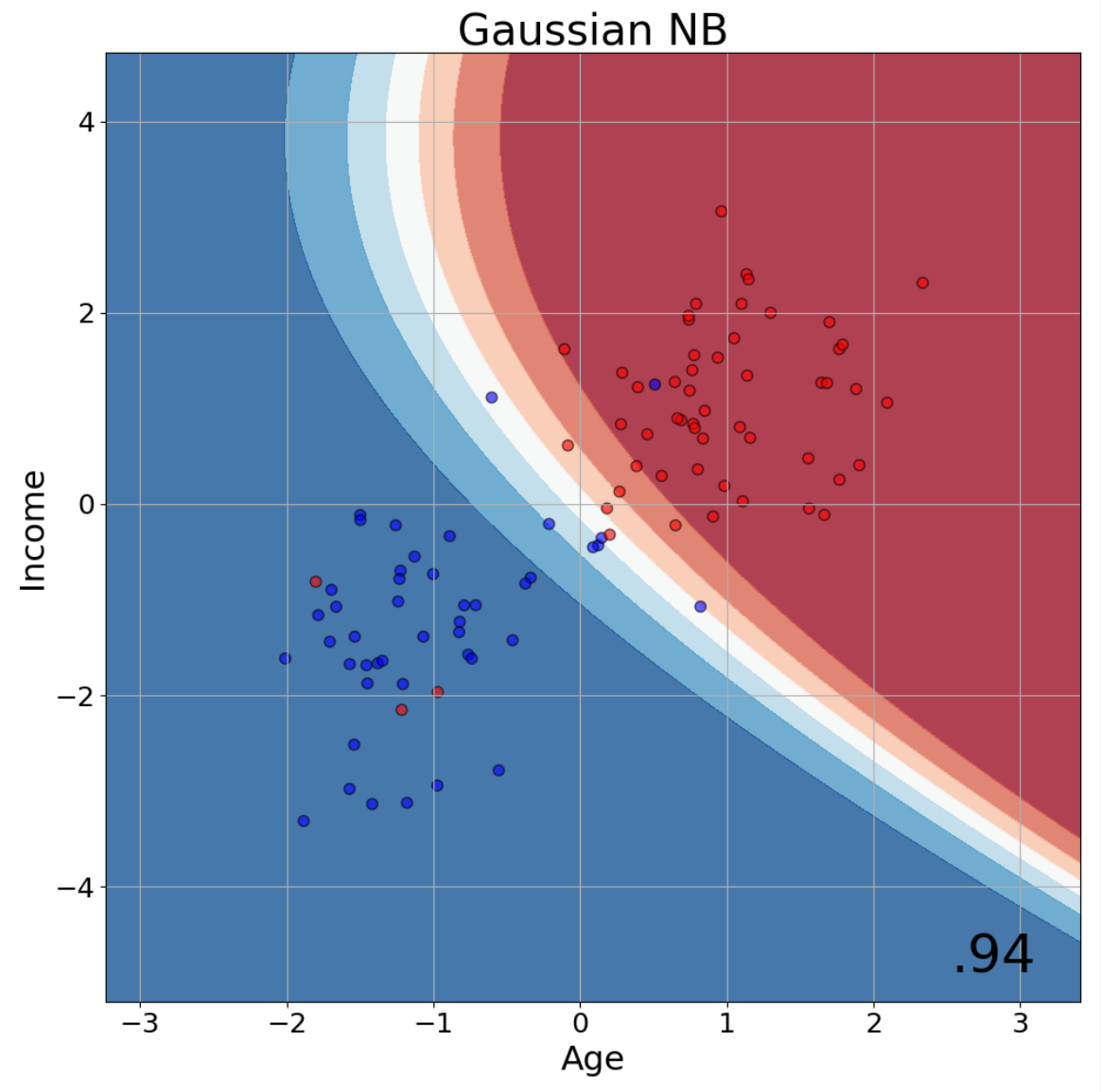
```
print(classification_report(y_test, y_pred_gnb, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.96      0.93      0.94        55
          1       0.91      0.96      0.93        45

avg / total       0.94      0.94      0.94       100
```

```
print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_gnb)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_gnb)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_gnb)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_gnb)))
```

```
Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

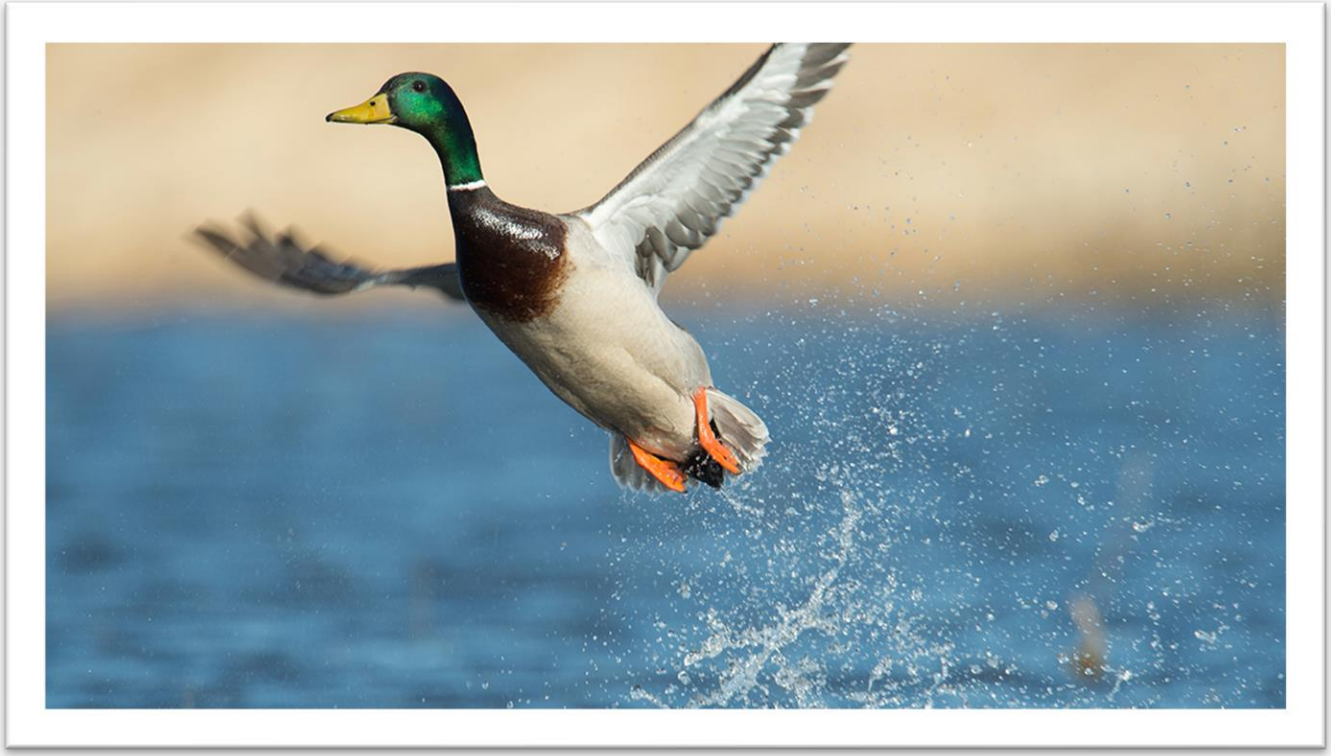
Marketing Dataset Example



KNN

K Nearest Neighbors (KNN)

If it walks like a duck, swims like a duck, and quacks like a duck, then let's call it a _____!



Big idea: find instances in training data that "look" like the current instance

How Does it Work?

- Model:
  - None
- Training phase
  - Nothing to do: Just store all the training data
- Prediction phase
  - This is where all the work happens
  - For a new instance, find the *K* instances in the training data that are "closest" to the new instance
    - Euclidean distance, hamming distance, cosine distance…
  - Predict the target of the new instance using the *K* nearest instances' target
    - Even better: weighted by similarity of each instance

Prediction Phase Example

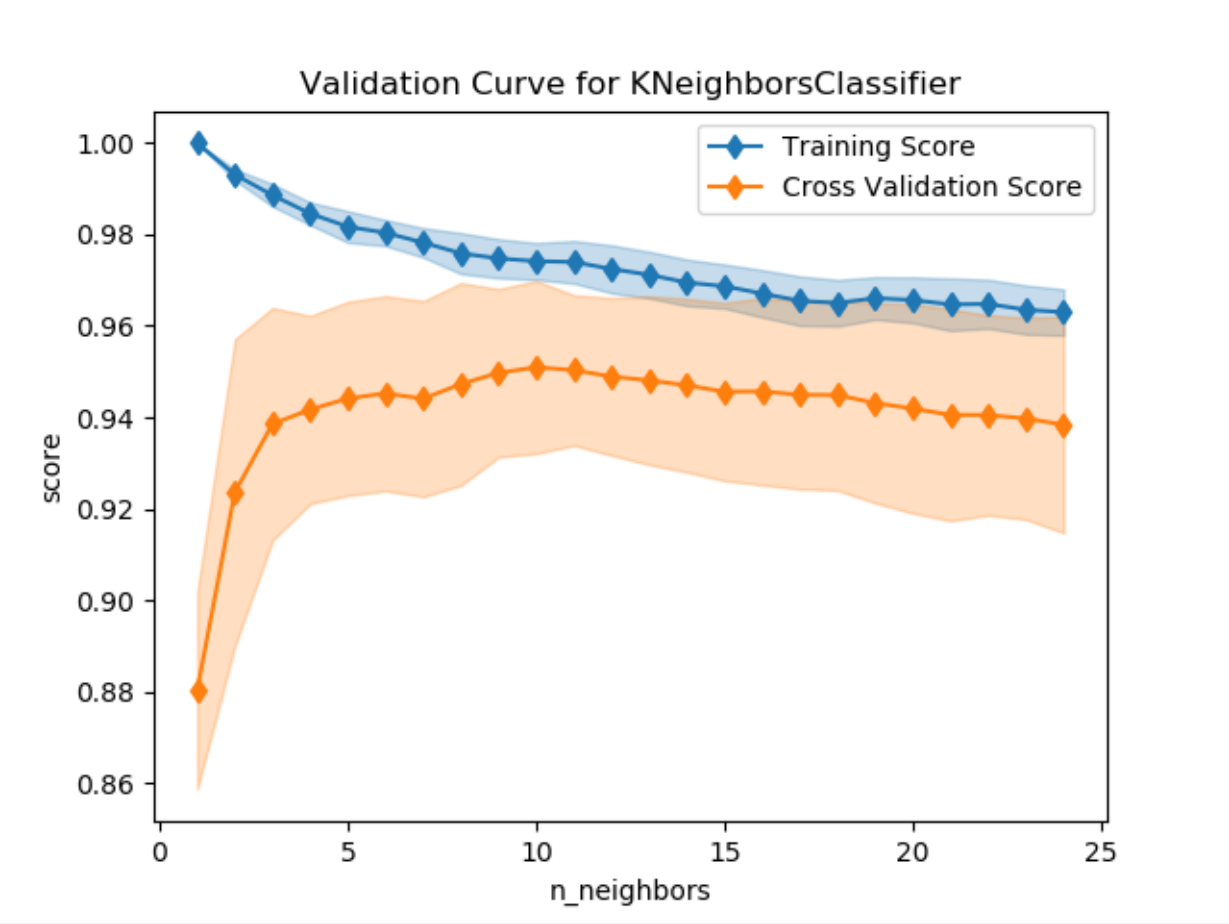
- Suppose K = 3, Hamming Distance
- Suppose a new instance is:
  - has_job=TRUE, own_house = FALSE, and credit_rating = good

has_job	own_house	credit_rating	status	
TRUE	FALSE	excellent	paid	0.67
FALSE	TRUE	excellent	paid	0.00
FALSE	TRUE	excellent	paid	0.00
FALSE	TRUE	excellent	paid	0.00
FALSE	FALSE	fair	default	0.33
FALSE	FALSE	fair	default	0.33
FALSE	FALSE	fair	default	0.33
FALSE	FALSE	fair	default	0.33
TRUE	TRUE	fair	paid	0.33
FALSE	FALSE	good	default	0.67 ←
TRUE	FALSE	good	paid	1.00 ←
TRUE	FALSE	good	paid	1.00 ←
FALSE	FALSE	good	default	0.67
FALSE	TRUE	good	paid	0.33
TRUE	TRUE	good	paid	0.67

Predict status=paid

Choice of K

- Popular heuristic: K = sqrt(N)
- Better: Parameter tuning with cross validation (next lecture)
  - Training error rate and test error rate as K decreases



KNN Summary

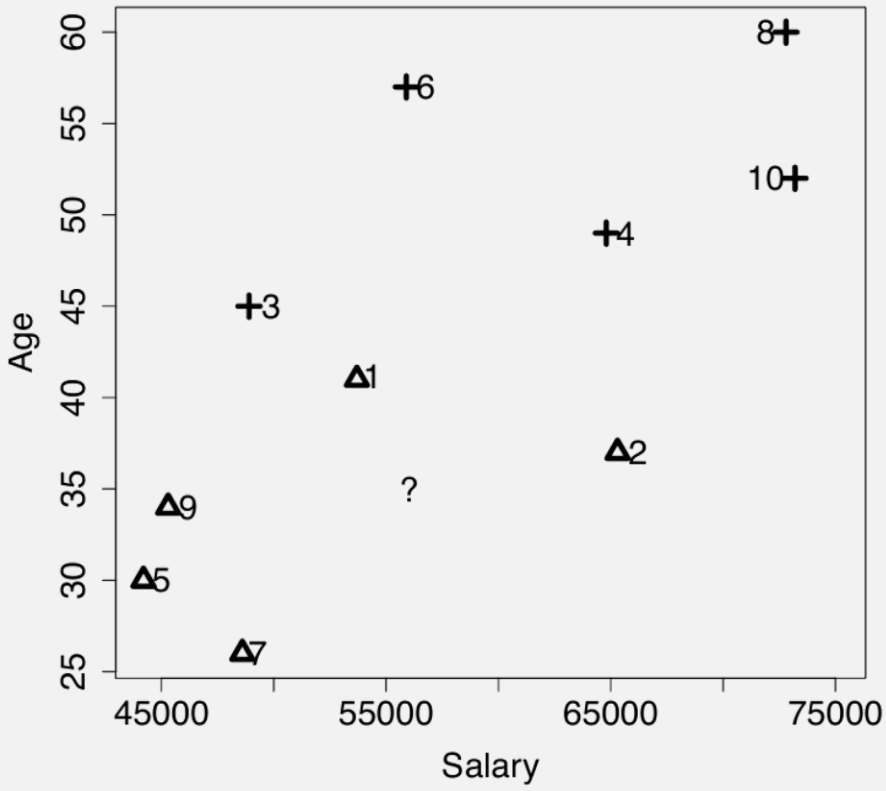
- **Model**: None.
- **Training Phase**: None.
- **Prediction Phase**: Find *K* nearest neighbors to training instance; take majority class

Data Normalization

- Often useful/necessary to normalize data for KNN
- Example:
  - Financial institution wants to run a marketing campaign to sell a pension product
  - Creates a 1NN model, using Euclidean distance, to predict which customers will respond to the campaign
    - Trained from historical data: Salary, Age, Purchase?
  - Offers will only be sent to customers that are predicted to respond
  - Using model, marketing team wants to know whether it should contact the following customer:
    - Salary = 56,000, Age = 35

A Few Rows from the Training Data

ID	SALARY	AGE	PURCH
1	53,700	41	no
2	65,300	37	no
3	48,900	45	yes
4	64,800	49	yes
5	44,200	30	no
6	55,900	57	yes
7	48,600	26	no
8	72,800	60	yes
9	45,300	34	no
10	73,200	52	yes



- What would the model predict for Salary=56,000, Age=35?
- Visually, it looks like ID=1 is closest (PURCH=no)
- However, model would actually predict yes! Why?

Look at the Distances

Dataset				Salary and Age		Salary Only		Age Only	
ID	Salary	Age	Purch	Dist.	Rank	Dist.	Rank	Dist.	Rank
1	53,700	41	no	2,300.0078	2	2,300	2	6	4
2	65,300	37	no	9,300.0002	6	9,300	6	2	2
3	48,900	45	yes	7,100.0070	3	7,100	3	10	6
4	64,800	49	yes	8,800.0111	5	8,800	5	14	7
5	44,200	30	no	11,800.0011	8	11,800	8	5	5
6	55,900	57	yes	102.3914	1	100	1	22	9
7	48,600	26	no	7,400.0055	4	7,400	4	9	3
8	72,800	60	yes	16,800.0186	9	16v800	9	25	10
9	45,300	34	no	10,700.0000	7	10,700	7	1	1
10	73,200	52	yes	17,200.0084	10	17,200	10	17	8

- ID=6 (PURCH=yes) is the closest when both Salary and Age are used
- Why?
- Salary values are much larger than Age values, and therefore dominate the Euclidean distance calculation

Let's Normalize!

Use ***range normalization*** to make values fall between 0 and 1

E.g., ID=1:

$$\text{SALARY: } \left(\frac{53{,}700 - 44{,}200}{73{,}200 - 44{,}200}\right) \times (1.0 - 0.0) + 0 = 0.3276$$

$$\text{AGE: } \left(\frac{41 - 26}{60 - 26}\right) \times (1.0 - 0.0) + 0 = 0.4412$$

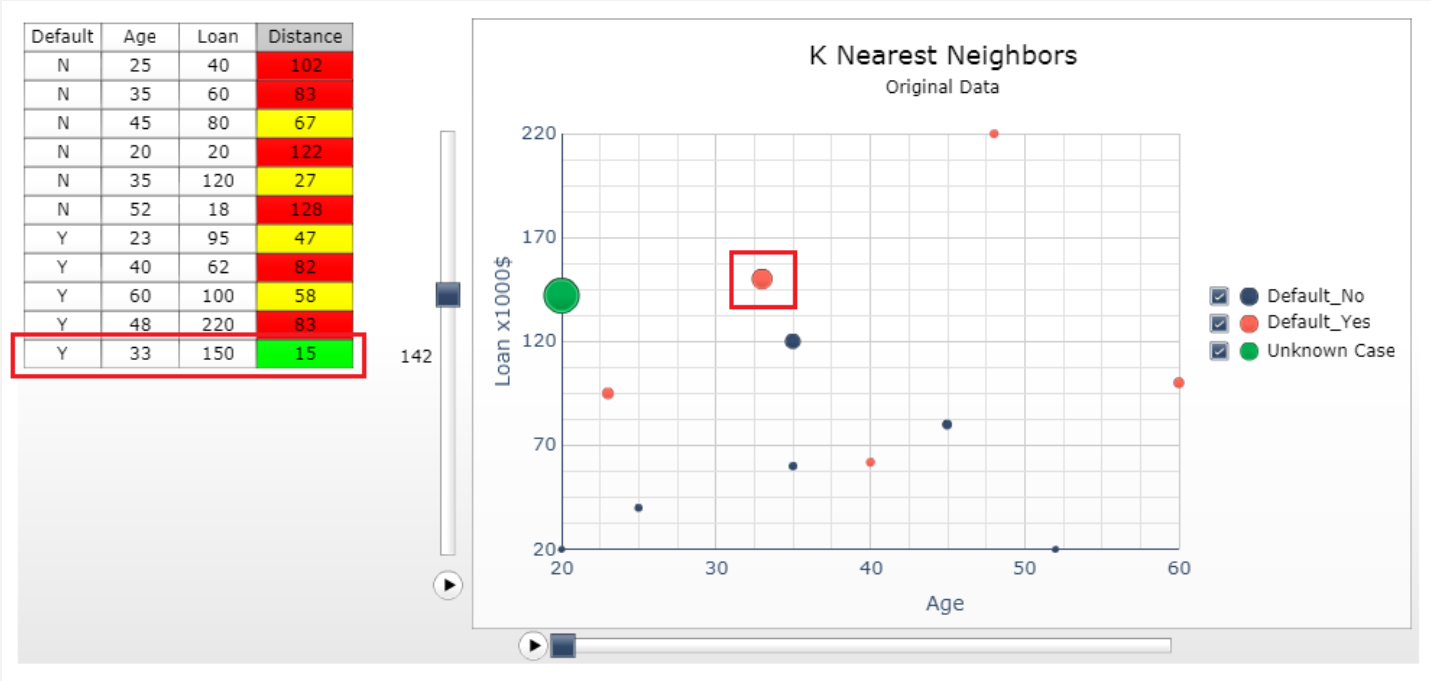
Normalized Dataset				SALARY and AGE		SALARY Only		AGE Only	
ID	SALARY	AGE	PURCH	Dist.	Rank	Dist.	Rank	Dist.	Rank
1	0.3276	0.4412	no	0.1935	1	0.0793	2	0.17647	4
2	0.7276	0.3235	no	0.3260	2	0.3207	6	0.05882	2
3	0.1621	0.5588	yes	0.3827	5	0.2448	3	0.29412	6
4	0.7103	0.6765	yes	0.5115	7	0.3034	5	0.41176	7
5	0.0000	0.1176	no	0.4327	6	0.4069	8	0.14706	3
6	0.4034	0.9118	yes	0.6471	8	0.0034	1	0.64706	9
7	0.1517	0.0000	no	0.3677	3	0.2552	4	0.26471	5
8	0.9862	1.0000	yes	0.9361	10	0.5793	9	0.73529	10
9	0.0379	0.2353	no	0.3701	4	0.3690	7	0.02941	1
10	1.0000	0.7647	yes	0.7757	9	0.5931	10	0.50000	8

- Now, Salary values do not dominate ➔better model
- Marketing team will not send offer to customer

Exercise

Use the link below to explore this phenomenon visually:

http://www.saedsayad.com/flash/KNN_flash.html



Default	Age	Loan	Distance
N	25	40	102
N	35	60	83
N	45	80	67
N	20	20	122
N	35	120	27
N	52	18	128
Y	23	95	47
Y	40	62	82
Y	60	100	58
Y	48	220	83
Y	33	150	15

Use the slider at the bottom to change the Age of the unknown case. Notice how the point in the red box is always the closest by distance?

Exercise

- ***Why does this matter in the real world?***

- In groups of 3-4 around you, come up with a few scenarios where failure to normalize your data could cause issues.

- Some ideas to get you started:
  - Classifying illnesses
  - Detecting bank fraud
  - Identifying defective car parts

- What are the practical implications? Why should you care?

Example

Bank Marketing Dataset: 20 features, 9280 instances

```
kknn_fit = train.kknn(bought ~ ., train, kmax=7)
summary(kknn_fit)
```

```
Call:
train.kknn(formula = bought ~ ., data = train, kmax = 7)

Type of response variable: nominal
Minimal misclassification: 0.1920797414
Best kernel: optimal
Best k: 6
```

Example

```
pred = predict(kknn_fit, test)
confusionMatrix(data=pred, reference=test$bought,
                positive=positive,
                dnn=c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

          Actual
Predicted  no yes
      no  776 200
      yes 152 728

               Accuracy : 0.8103448
                 95% CI : (0.7917552, 0.827948)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 0.00000000000000222

                  Kappa : 0.6206897
 Mcnemar's Test P-Value : 0.01224139

            Sensitivity : 0.7844828
            Specificity : 0.8362069
         Pos Pred Value : 0.8272727
         Neg Pred Value : 0.7950820
             Prevalence : 0.5000000
         Detection Rate : 0.3922414
   Detection Prevalence : 0.4741379
      Balanced Accuracy : 0.8103448

       'Positive' Class : yes
```

Marketing Dataset Example

```
from sklearn.neighbors import KNeighborsClassifier

knn_clf = KNeighborsClassifier(n_neighbors=3)
knn_clf.fit(X_train, y_train)

y_pred_knn = knn_clf.predict(X_test)
```

```
print(ConfusionMatrix(y_test, y_pred_knn))
```

```
Predicted  False  True  __all__
Actual
False         47     8       55
True           2    43       45
__all__       49    51      100
```

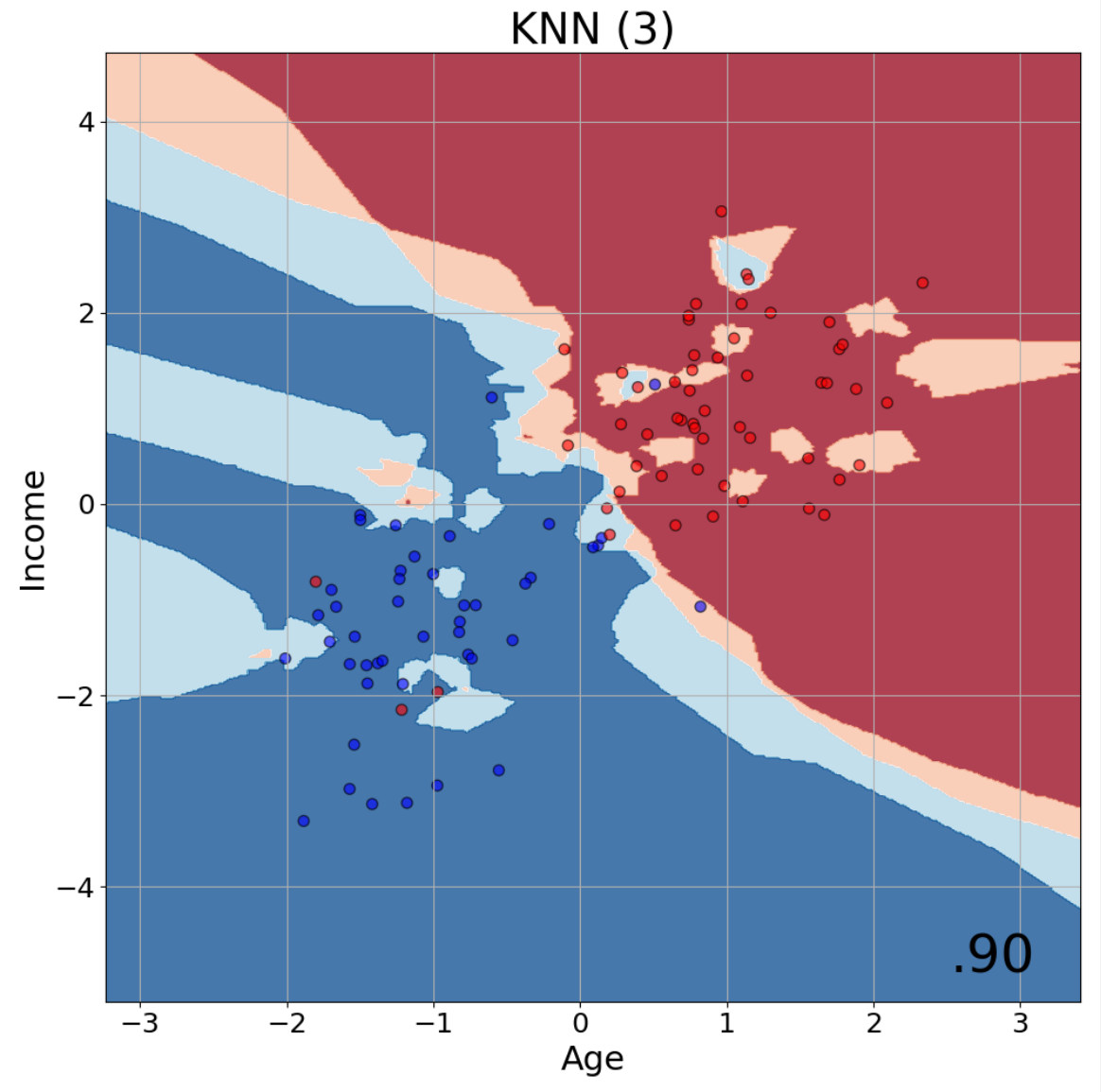
```
print(classification_report(y_test, y_pred_knn, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.96      0.85      0.90        55
          1       0.84      0.96      0.90        45

avg / total       0.91      0.90      0.90       100
```

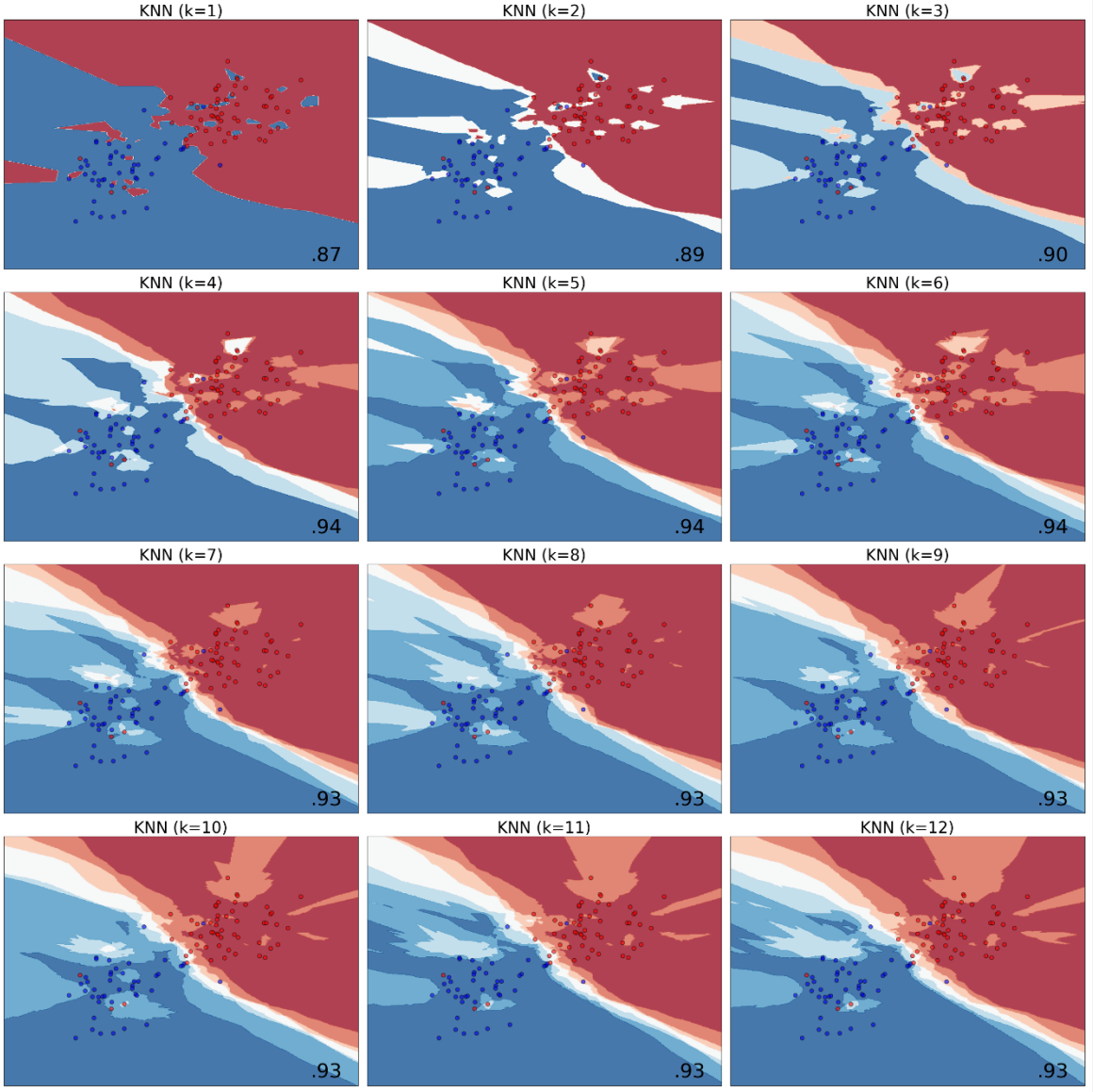
```
print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_knn)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_knn)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_knn)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_knn)))
```

```
Accuracy = 0.90
Kappa = 0.80
F1 Score = 0.90
Log Loss = 3.45
```

Marketing Dataset

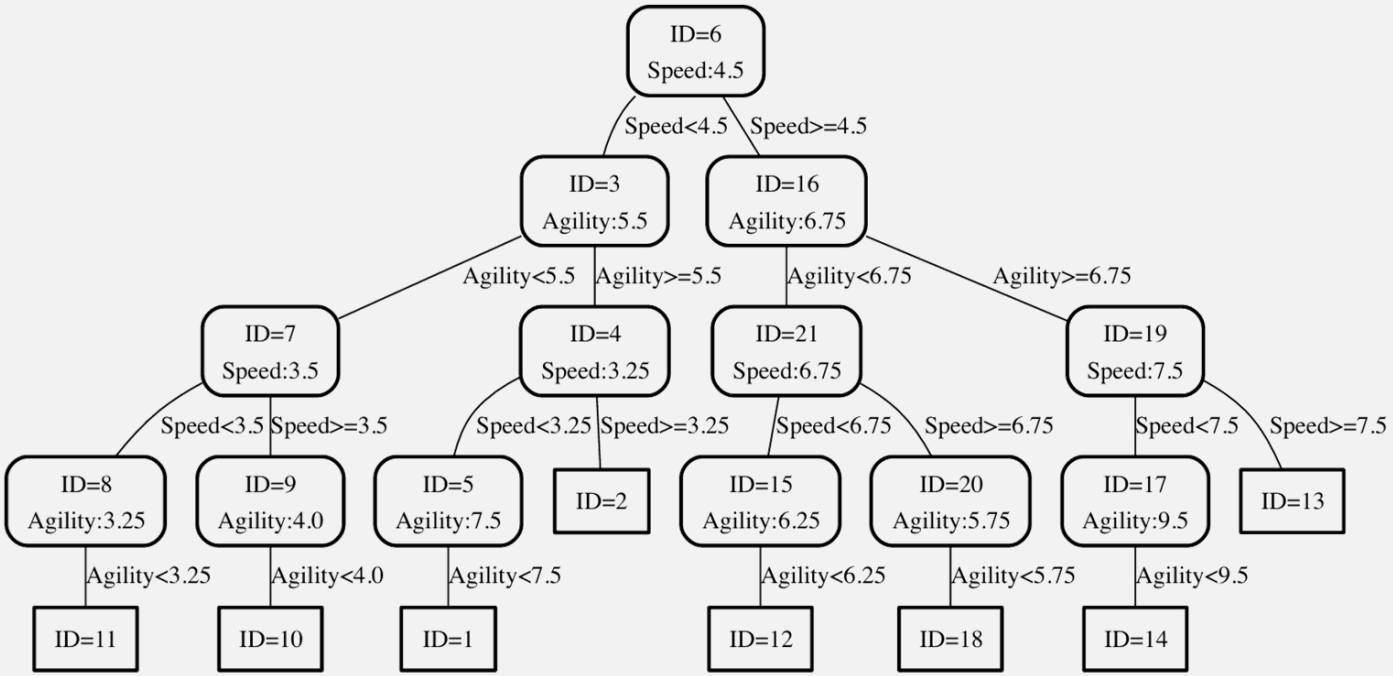


Marketing Dataset (Lots of Ks)



Efficient Memory Search

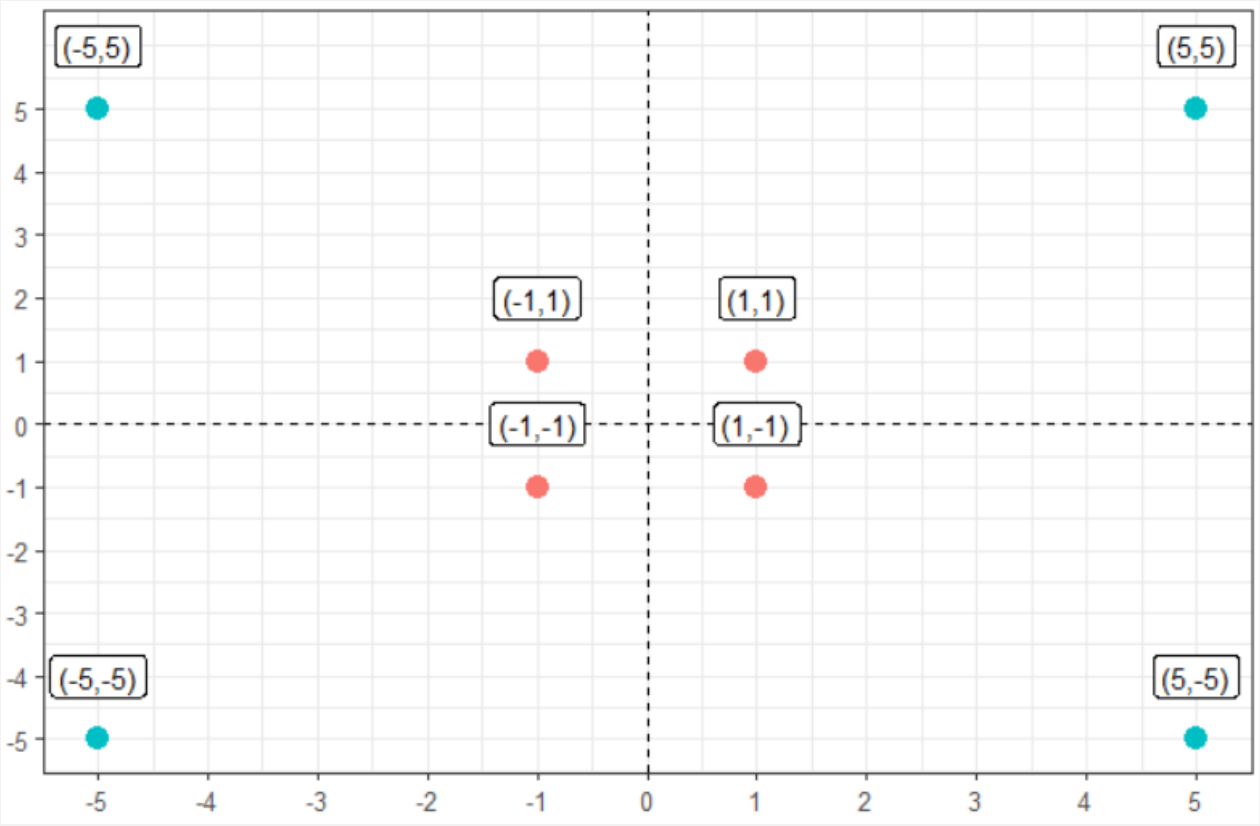
- If training data is large, then cost of prediction could be HUGE
  - Must compute similarity between new instance and every training sample
- Better way: pre-compute a special data structure to reduce computation necessary during prediction phase
- One example: ***k-d tree***



SVM

Exercise: Kernel Intuition

- Given the following points, how would you use a polynomial kernel function to separate the red from the green?
- ***Hint: Can we use the equation of a circle somehow?***

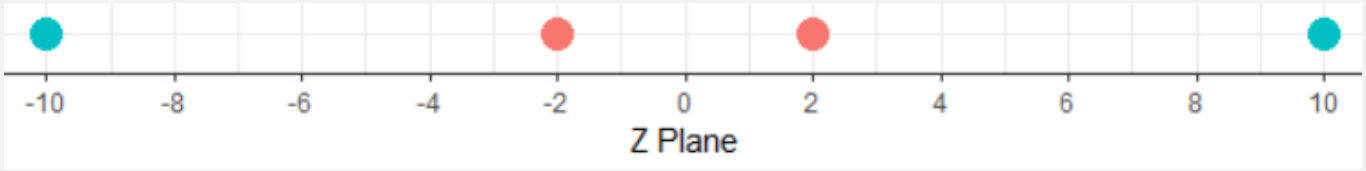


Exercise: Kernel Intuition

- Let's start by defining another feature called Z
  - We'll apply a function to X and Y to project into Z plane
  - But what function should we use? Z = X + Y?

Colour	X	Y	Z = X + Y
Blue	-5	5	0
Blue	5	5	10
Blue	-5	-5	-10
Blue	5	-5	0
Red	-1	1	0
Red	1	1	2
Red	-1	-1	-2
Red	1	-1	0

- Are we any better off? Can we separate the data using a single cut?

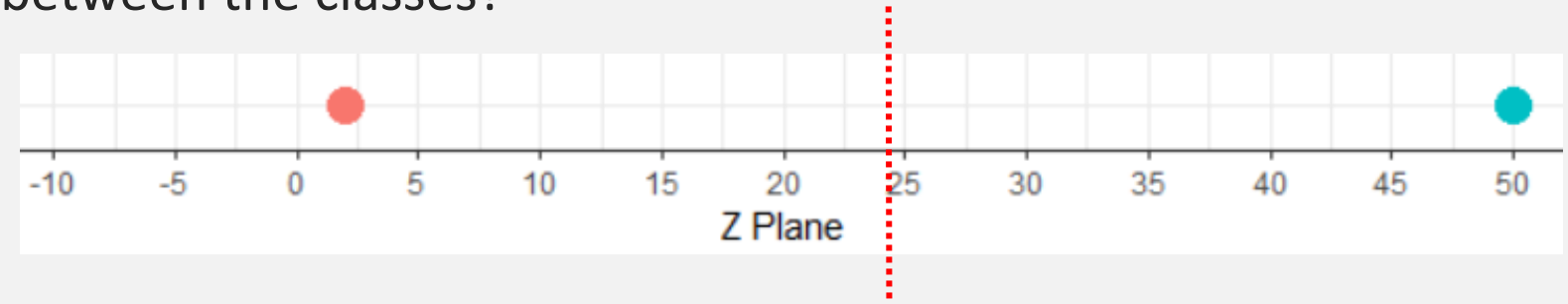


Exercise: Kernel Intuition

- What if we try: Z = X^2 + Y^2

Colour	X	Y	Z = X^2 + Y^2
Blue	-5	5	50
Blue	5	5	50
Blue	-5	-5	50
Blue	5	-5	50
Red	-1	1	2
Red	1	1	2
Red	-1	-1	2
Red	1	-1	2

- Now can you separate the red from the blue with a single cut?
- Where would you cut in the Z-plane to maximize the distance between the classes?

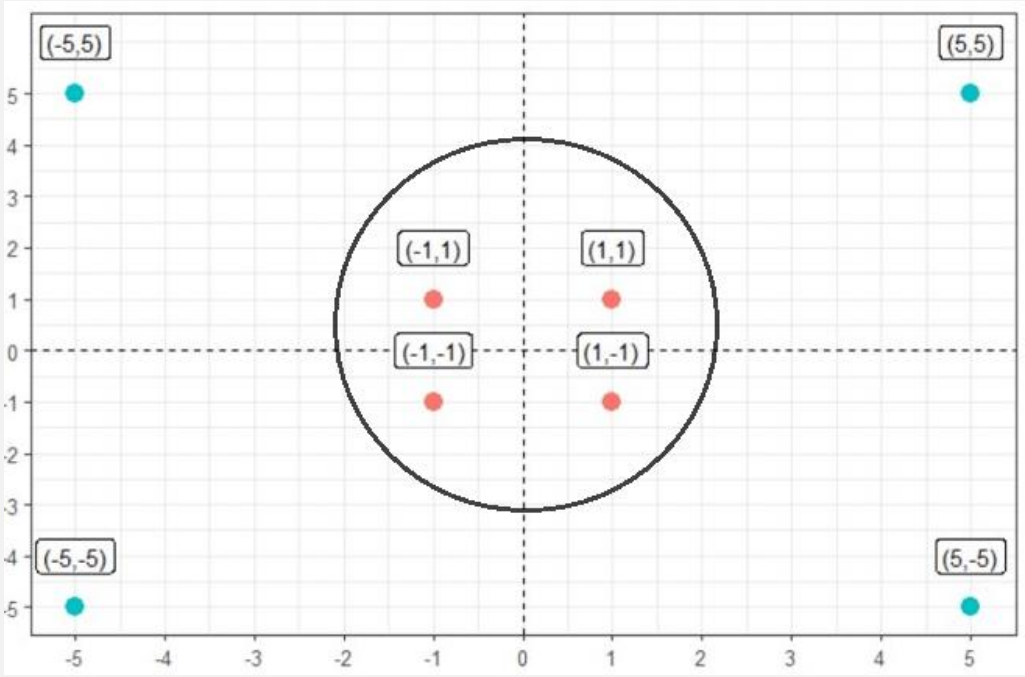


Z = (50 – 2) / 2 = 24

Exercise: Kernel Intuition

Recap:

- Used the polynomial kernel Z = X^2 + Y^2 to project our data into the Z plane (i.e., to create another feature)
- Found the value of Z that splits the data optimally
- Projected 24 = X^2 + Y^2 back down to the X, Y plane to separate the data



Example

Bank Marketing Dataset: 20 features, 9280 instances

```
library(e1071)
svm_fit <- svm(bought ~ ., data = train)
svm_fit
summary(svm_fit)
```

```
Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  1
      gamma:  0.01612903

Number of Support Vectors:  2578

 ( 1276 1302 )


Number of Classes:  2

Levels:
 no yes
```

Example

```
pred = predict(svm_fit, test)
confusionMatrix(data=pred, reference=test$bought,
                positive=positive,
                dnn=c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

         Actual
Predicted  no yes
      no  796  92
      yes 132 836

               Accuracy : 0.8793
                 95% CI : (0.8636, 0.8938)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7586
 Mcnemar's Test P-Value : 0.009166

            Sensitivity : 0.9009
            Specificity : 0.8578
         Pos Pred Value : 0.8636
         Neg Pred Value : 0.8964
             Prevalence : 0.5000
         Detection Rate : 0.4504
   Detection Prevalence : 0.5216
      Balanced Accuracy : 0.8793

       'Positive' Class : yes
```

Marketing Dataset Example

```
from sklearn.svm import SVC

svm_clf = SVC(kernel="linear", C=0.025)
svm_clf.fit(X_train, y_train)

y_pred_svm = svm_clf.predict(X_test)
```

```
svm_clf.n_support_
```

```
array([70, 69])
```

```
svm_clf.support_vectors_
```

```
array([[-2.30188062e+00, -1.37169280e+00],
       [-1.18486685e+00, -1.79040549e+00],
       [ 6.59102878e-01, -4.92594995e-02],
       [ 1.88977492e-01,  1.49340633e+00],
       [ 6.32255404e-02,  1.15016778e+00],
       [ 1.23912826e+00, -4.01893060e-01],
       [ 7.49980152e-01, -3.38768128e-01],
       [ 1.23622216e+00,  1.18585351e-01],
       [ 7.19460491e-01,  9.02795935e-01],
       [ 1.95852933e-01,  1.24920912e+00],
       [ 6.24289241e-01,  7.92790019e-01],
       [ 6.76905028e-01, -3.75127816e-01],
       [ 9.03792039e-01,  3.30215452e-01],
       [ 1.27385324e+00, -1.12810817e-02],
       [-8.82249296e-01, -1.76640750e+00],
       [-2.19281611e-02,  1.37013900e+00],
       [ 6.18149343e-01,  8.72452885e-01],
       [ 7.33559474e-01,  3.18524739e-01],
       [ 5.64698788e-01,  8.86571007e-01],
       [-1.13932834e+00,  1.30141500e-01],
       [ 8.88865699e-01,  2.62250487e-01],
```

Marketing Dataset Example

```
print(ConfusionMatrix(y_test, y_pred_svm))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

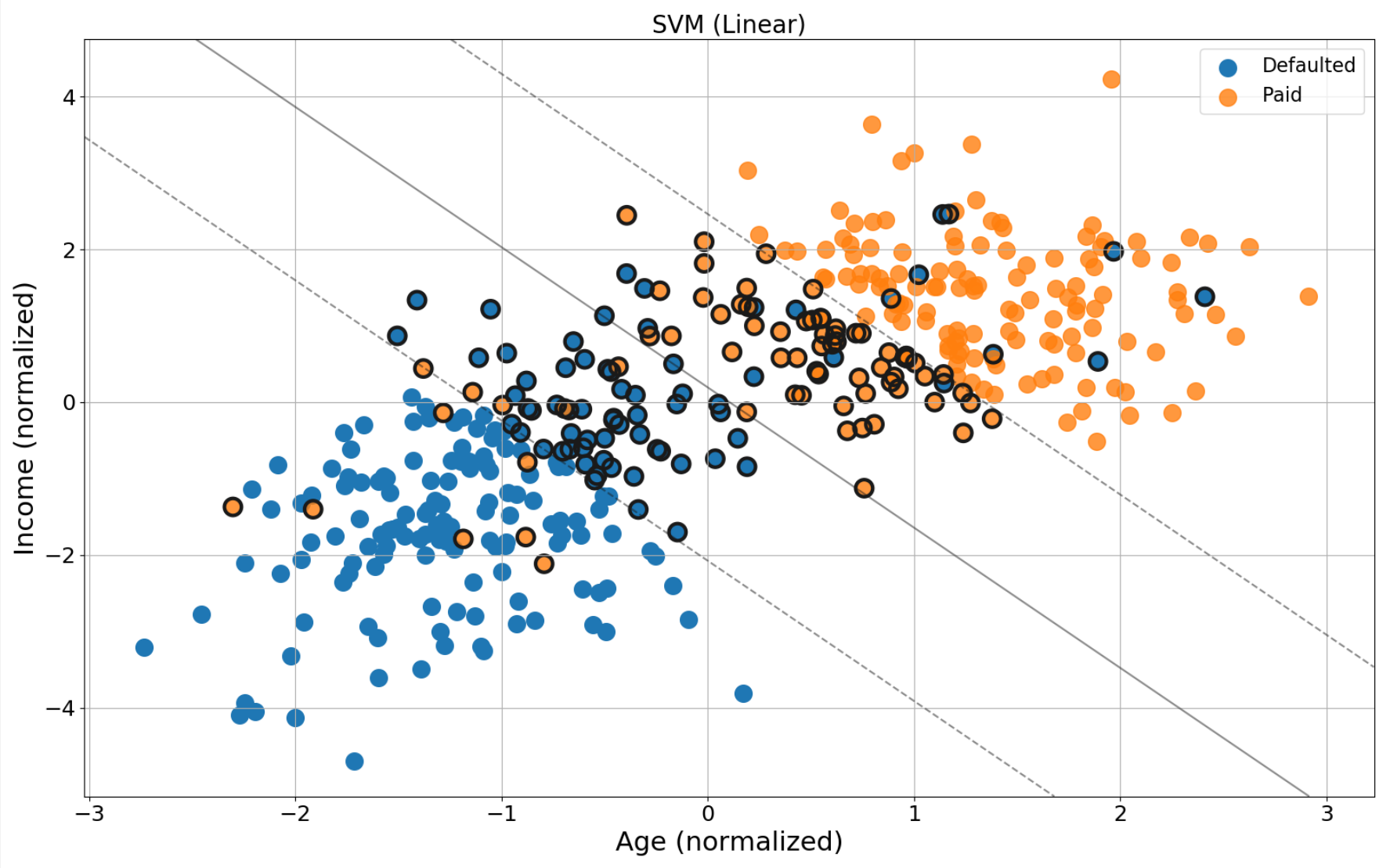
```
print(classification_report(y_test, y_pred_svm, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.96      0.93      0.94        55
          1       0.91      0.96      0.93        45

avg / total       0.94      0.94      0.94       100
```

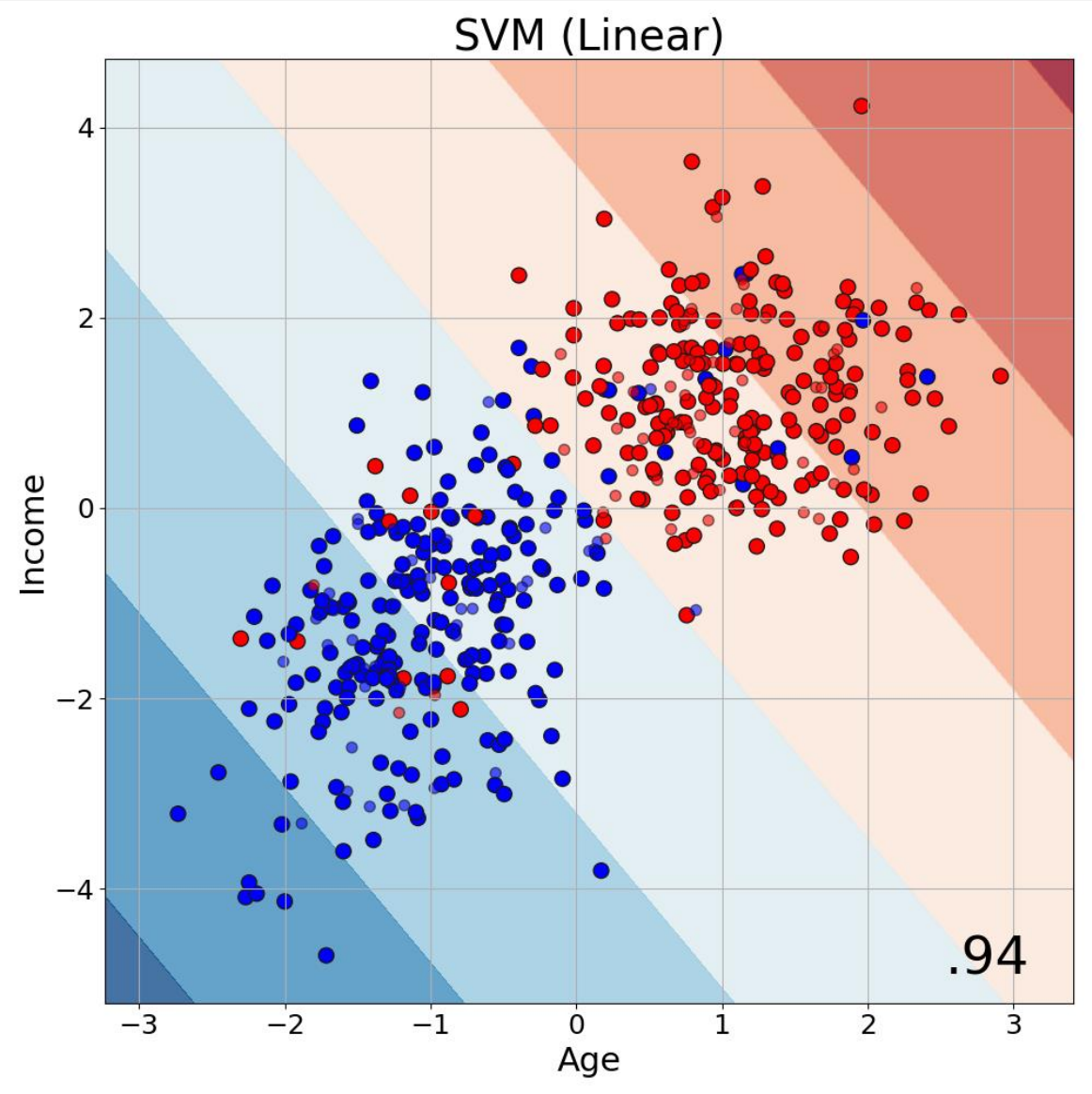
```
print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_svm)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_svm)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_svm)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_svm)))
```

```
Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

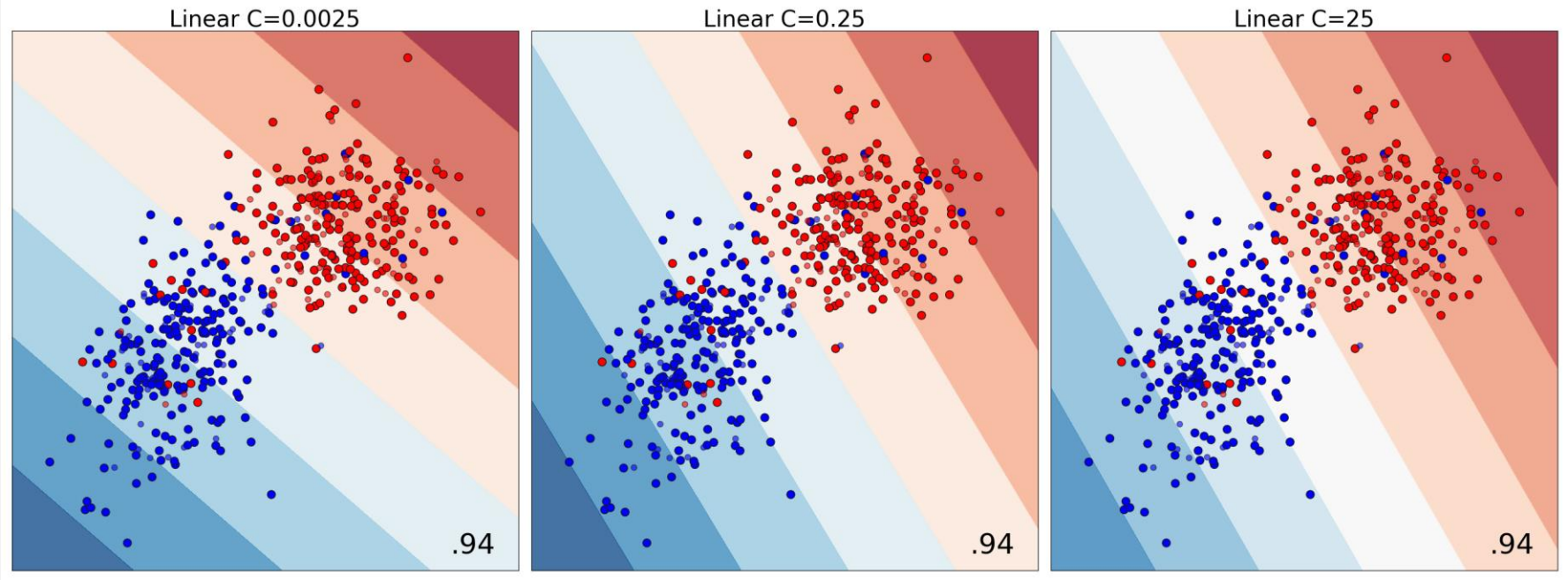
Marketing Dataset Example



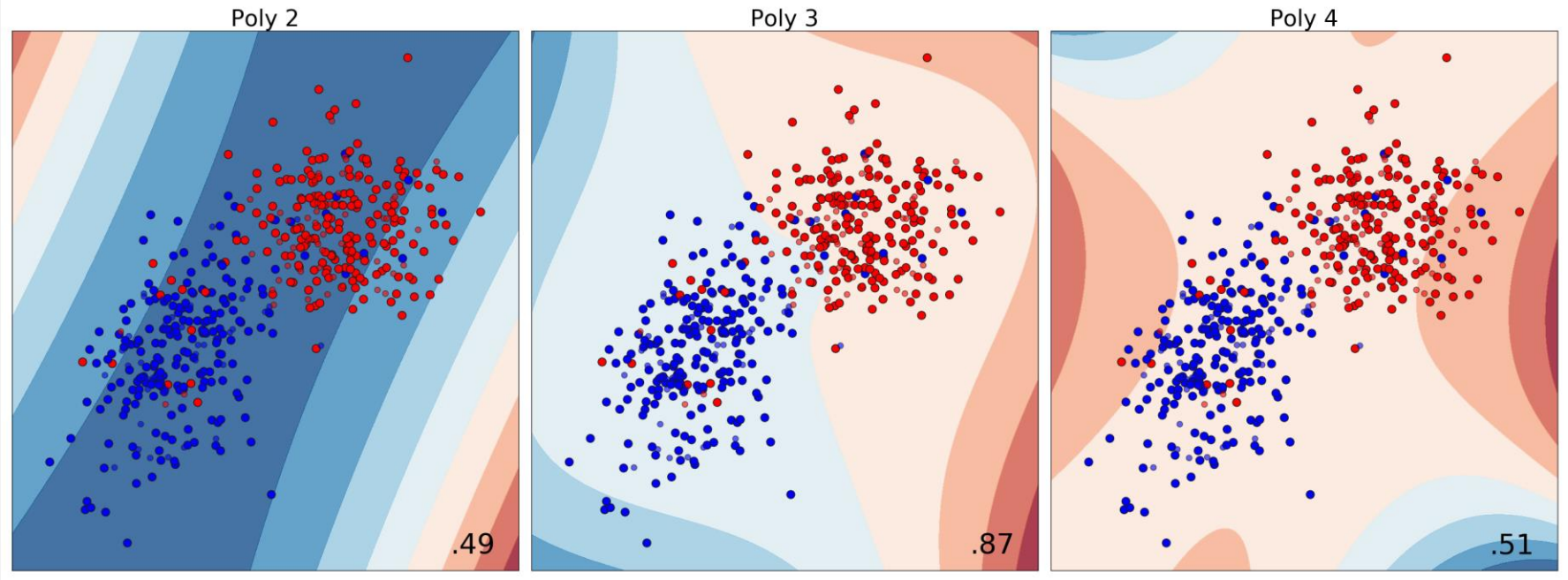
Marketing Dataset Example



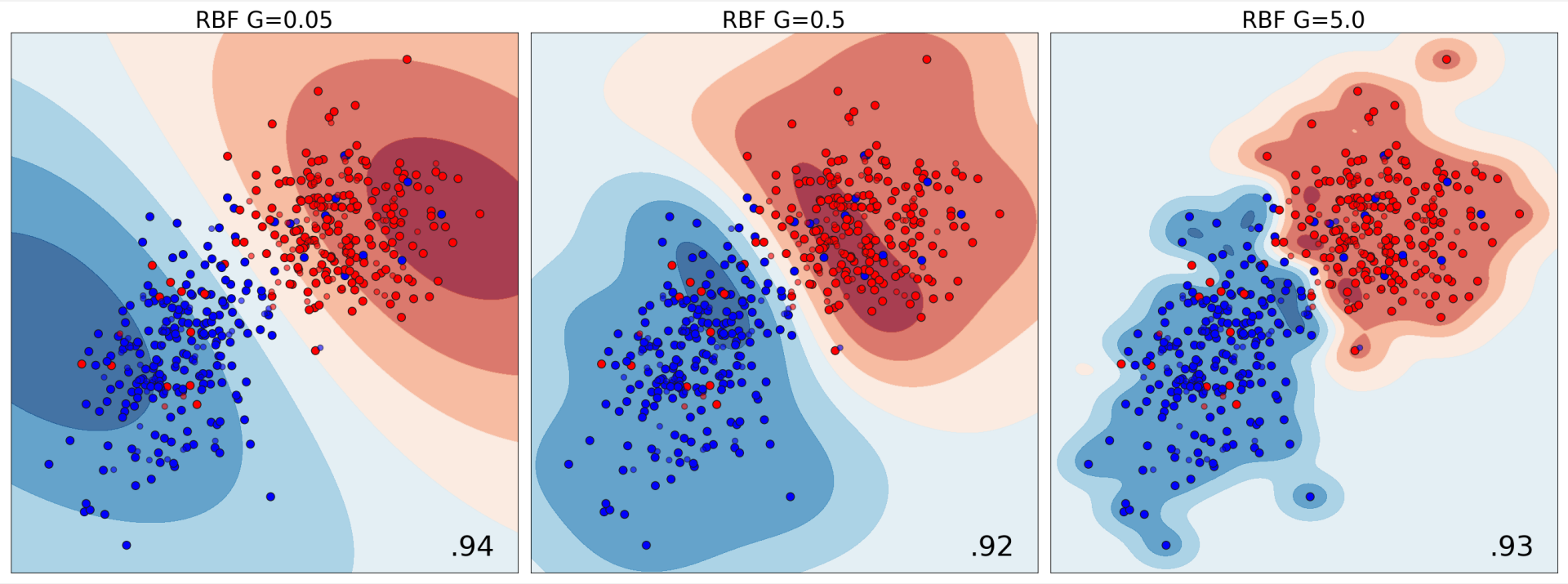
Marketing Dataset Example



Marketing Dataset Example



Marketing Dataset Example



NN

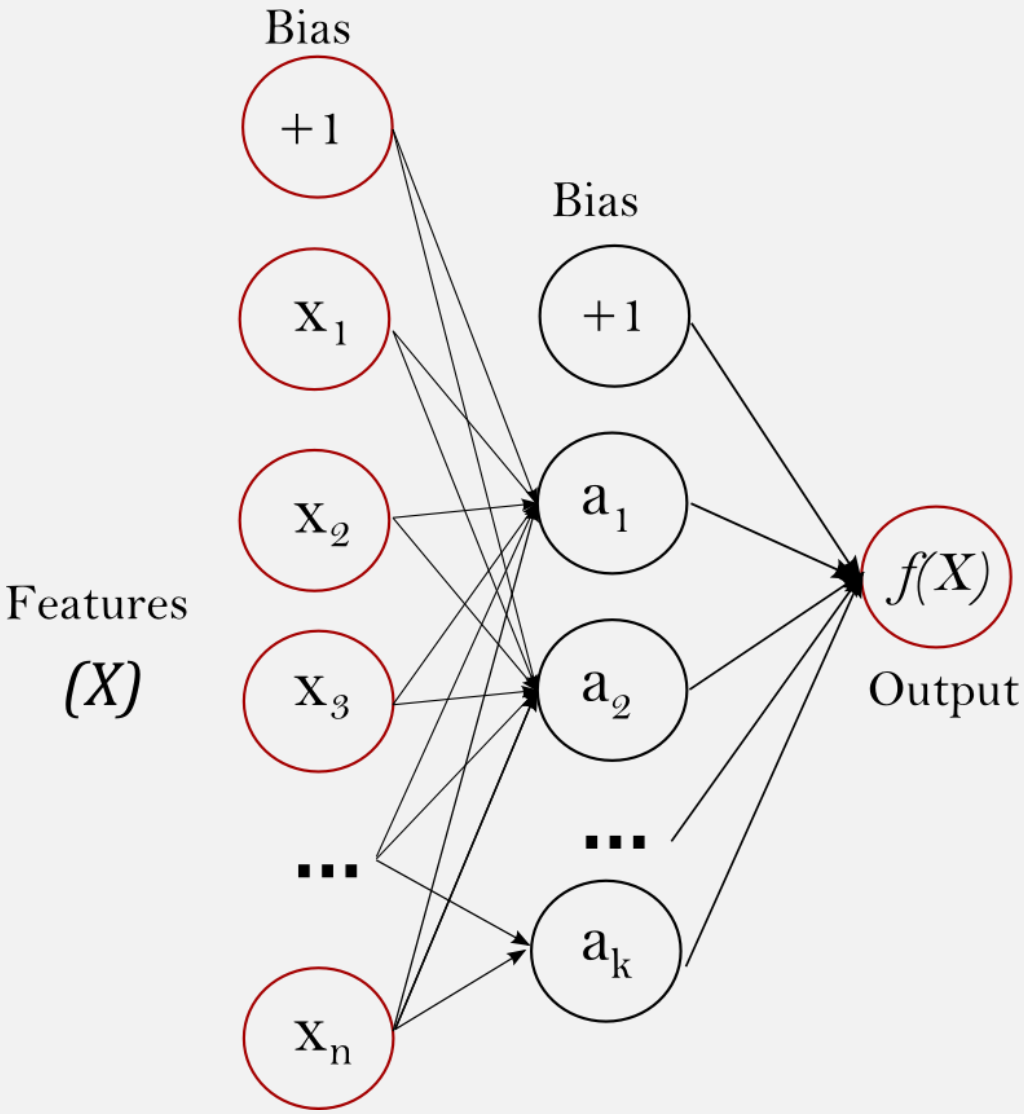
Example

Bank Marketing Dataset: 20 features, 9280 instances

```
library(nnet)
nnet_fit <- nnet(bought ~ ., data=train, size=5)
summary(nnet_fit)
```

```
a 62-5-1 network with 321 weights
options were - entropy fitting
  b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1 i10->h1 i11->h1
   0.34    1.81    0.23    0.16   -0.88   -0.01    0.10   -1.33    0.74   -1.07   -0.11    0.76
i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1 i19->h1 i20->h1 i21->h1 i22->h1 i23->h1
   0.24   -0.59   -1.88    3.31   -2.08    0.28    0.79   -0.11   -2.07   -1.32   -0.63    0.30
i24->h1 i25->h1 i26->h1 i27->h1 i28->h1 i29->h1 i30->h1 i31->h1 i32->h1 i33->h1 i34->h1 i35->h1
  -1.73    1.62    0.83   -0.60   -0.05   -5.27    0.08    4.93    1.00   -0.18   -0.86   -1.01
i36->h1 i37->h1 i38->h1 i39->h1 i40->h1 i41->h1 i42->h1 i43->h1 i44->h1 i45->h1 i46->h1 i47->h1
   1.13    1.32    0.88    0.05   -1.48   -0.21   -1.55    1.55   -1.62   -0.97    1.01   -0.36
i48->h1 i49->h1 i50->h1 i51->h1 i52->h1 i53->h1 i54->h1 i55->h1 i56->h1 i57->h1 i58->h1 i59->h1
   3.75   -2.08   -0.07    0.10   -5.53   -6.83    2.18    0.69   -0.81    0.60    1.21    3.63
i60->h1 i61->h1 i62->h1
   7.36    0.25    4.27
  b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2 i10->h2 i11->h2
  -0.71    0.16    1.46   -7.98   -4.76    0.45   -0.44   13.60    1.53   -2.41    7.11   -3.69
i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2 i19->h2 i20->h2 i21->h2 i22->h2 i23->h2
  -6.68    1.98   -0.33    1.28    1.16   -2.63   -1.51    2.99    0.47   -5.29    2.92   -2.68
i24->h2 i25->h2 i26->h2 i27->h2 i28->h2 i29->h2 i30->h2 i31->h2 i32->h2 i33->h2 i34->h2 i35->h2
  -0.09    1.08    5.01   -5.81   -0.47   -2.28    1.41    0.15   -0.98    1.66   -2.31   -3.11
i36->h2 i37->h2 i38->h2 i39->h2 i40->h2 i41->h2 i42->h2 i43->h2 i44->h2 i45->h2 i46->h2 i47->h2
   2.10    1.82    5.61  -10.60    2.40    2.71    5.92  -14.39   -6.49    8.10    4.07   -3.49
i48->h2 i49->h2 i50->h2 i51->h2 i52->h2 i53->h2 i54->h2 i55->h2 i56->h2 i57->h2 i58->h2 i59->h2
   1.10    0.64    0.21    0.42    0.15   -0.67    0.00    2.00   -5.50   -0.99    6.84   -8.26
i60->h2 i61->h2 i62->h2
  -0.49   -0.34   -4.28
```

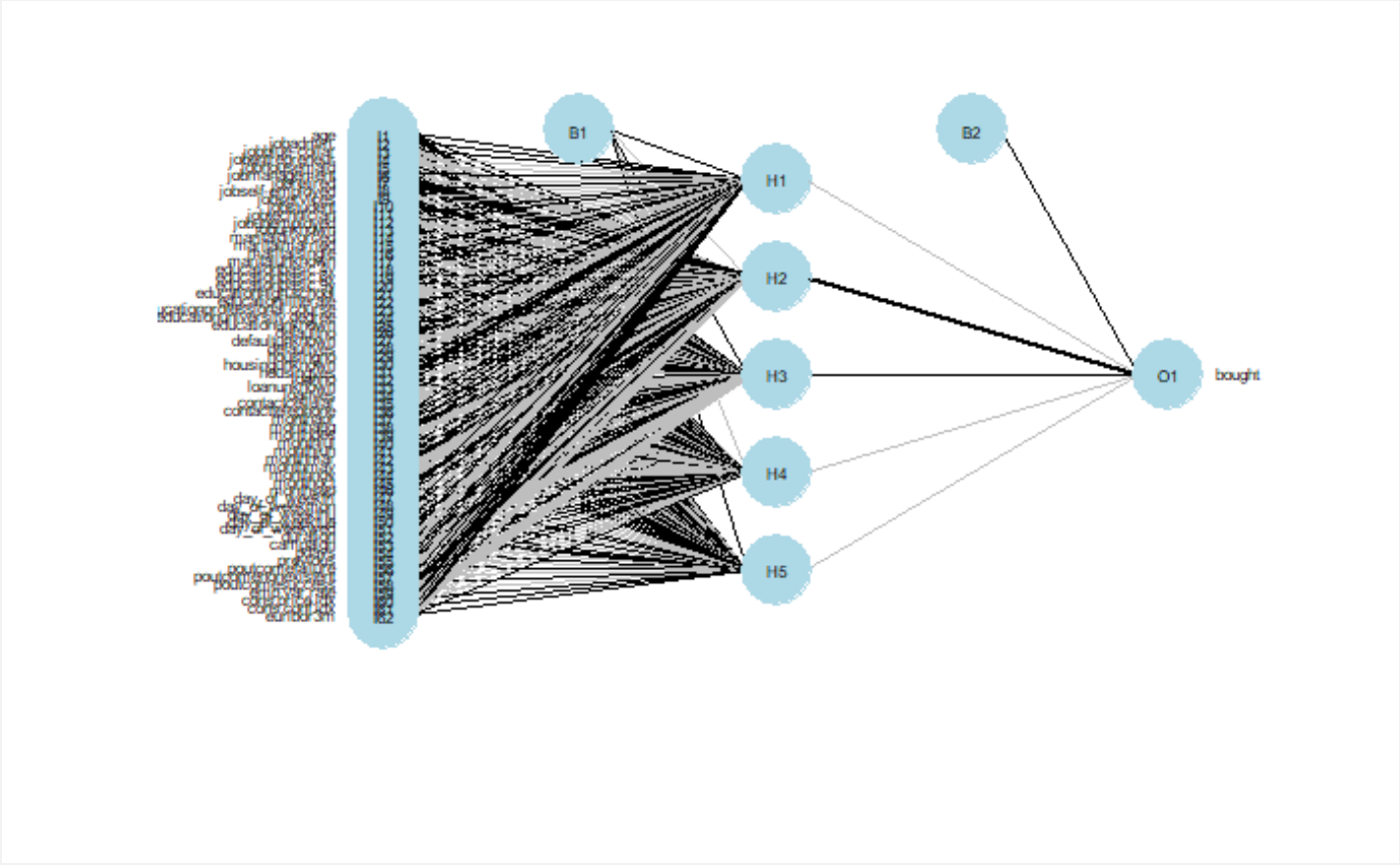
Another View



Source: https://scikit-learn.org/stable/modules/neural_networks_supervised.html

Example

```
library(gamlss.add)
plot(nnet_fit, cex.val = 0.5, max.sp = TRUE)
```



Example

```
pred = ifelse(predict(nnet_fit, test) > 0.5, "yes", "no")
confusionMatrix(data=pred, reference=test$bought,
                positive=positive,
                dnn=c("Predicted", "Actual"))
```

```
Confusion Matrix and Statistics

          Actual
Predicted  no yes
      no  736  58
      yes 192 870

               Accuracy : 0.8653017
                 95% CI : (0.8489226, 0.8805165)
    No Information Rate : 0.5
    P-Value [Acc > NIR] : < 0.00000000000000022204

                  Kappa : 0.7306034
 Mcnemar's Test P-Value : < 0.00000000000000022204

            Sensitivity : 0.9375000
            Specificity : 0.7931034
         Pos Pred Value : 0.8192090
         Neg Pred Value : 0.9269521
             Prevalence : 0.5000000
         Detection Rate : 0.4687500
   Detection Prevalence : 0.5721983
      Balanced Accuracy : 0.8653017

       'Positive' Class : yes
```

Other Terminology

- ***Regularization***: ways to prevent overfitting
  - ***L1***: basically performs feature selection
  - ***L2*** (weight decay): helps to keep weights small (which leads to simpler models), usually preferred
  - ***Dropout***: randomly ignoring some neurons during a particular forward or backward pass
- ***Epoch:*** one forward pass and one backward pass of all the training instances
- ***Batch size***: the number of training instances in one forward/backward pass
  - Weights are updated after each batch
  - The higher the batch size, the more memory you'll need
  - Example: if you have 1000 training instances, and your batch size is 50, then it will take 20 iterations to complete 1 epoch
- ***Learning rate***: how big the "step" is in the gradient descent algorithm

Other Ways to Think About ML Algorithms

- Earlier, we categorized algorithms around the approaches to learning:
  - Information-based → based on entropy
  - Similarity-based → based on distance measures
  - Probability-based → based on Bayes' Theorem
  - Error-based → based on sum of squared errors
- But, there are other ways to categorize algorithms
  - Parametric *versus* Non-parametric
  - Generative *versus* Discriminative

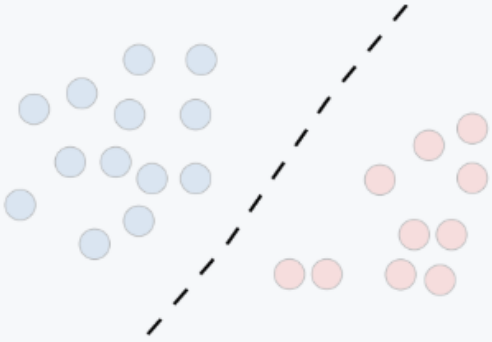
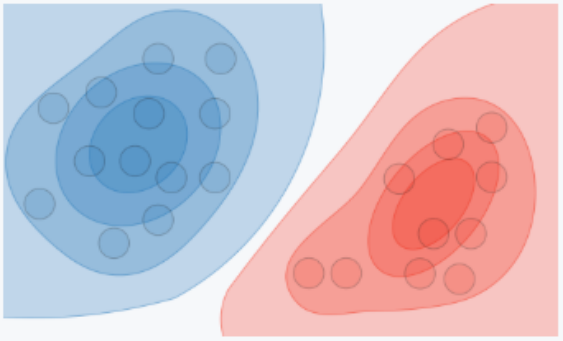
Parametric *versus* Non-parametric

Type	Summary	Examples	Pros	Cons
Parametric	Summarizes data with a set number of parameters (independent of the number of training instances)	• Logistic Regression • Neural network • Naïve Bayes	✓ Simpler ✓ Faster ✓ Need less data	× Constrained × Limited complexity × Poor fit
Non-parametric	Number of parameters increases as the number of training samples. More samples → more parameters in the model	• KNN • Decision Trees • SVM	✓ Flexibility ✓ Power ✓ Performance	× Need more data × Slower × Can overfit

Generative *versus* Discriminative

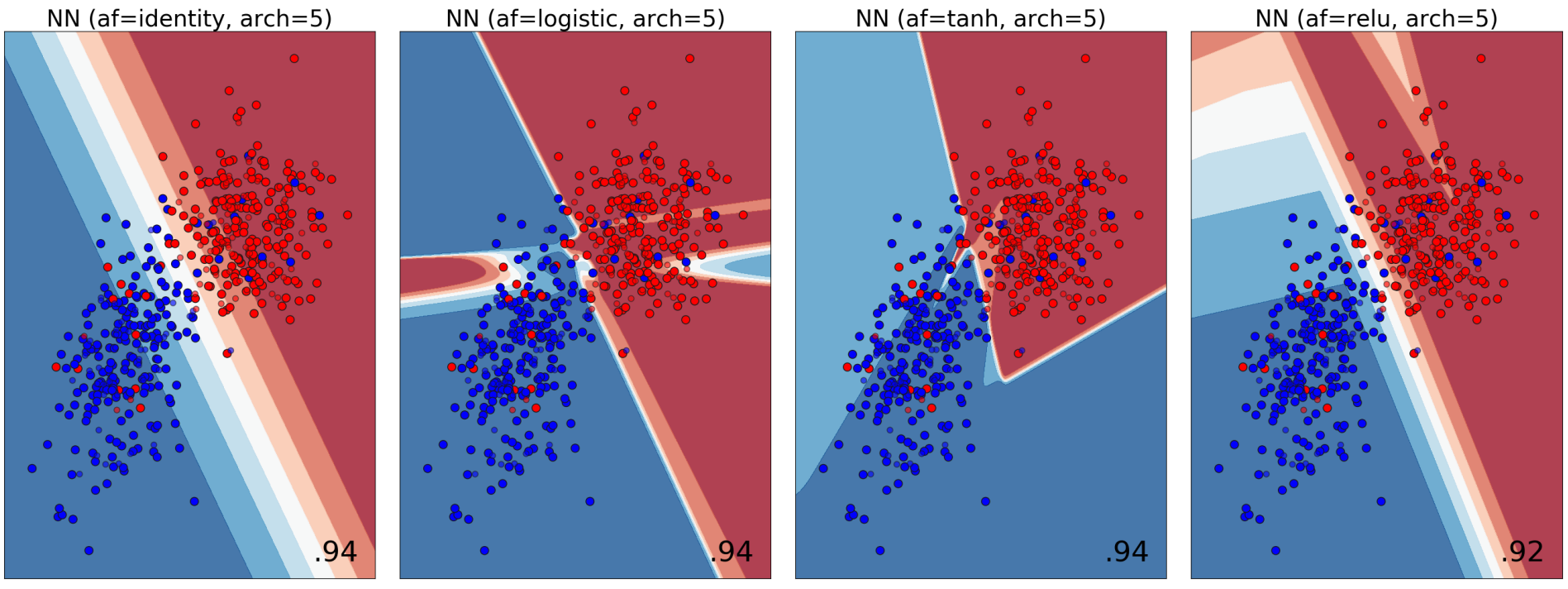
Type	Summary	Examples	Pros	Cons
Generative	Tries to figure out the underlying distribution of all features. Uses that knowledge to guess decision boundaries.	• Naïve Bayes	✓ Needs less data ✓ Can be used to generate data (e.g., imputation)	× Less accurate
Discriminative	Doesn't care about underlying distribution. Guesses decision boundaries directly.	• KNN • Decision Trees • SVM • Logistic regression • Neural Network	✓ More accurate	× Needs more data

Type of model — The different models are summed up in the table below:

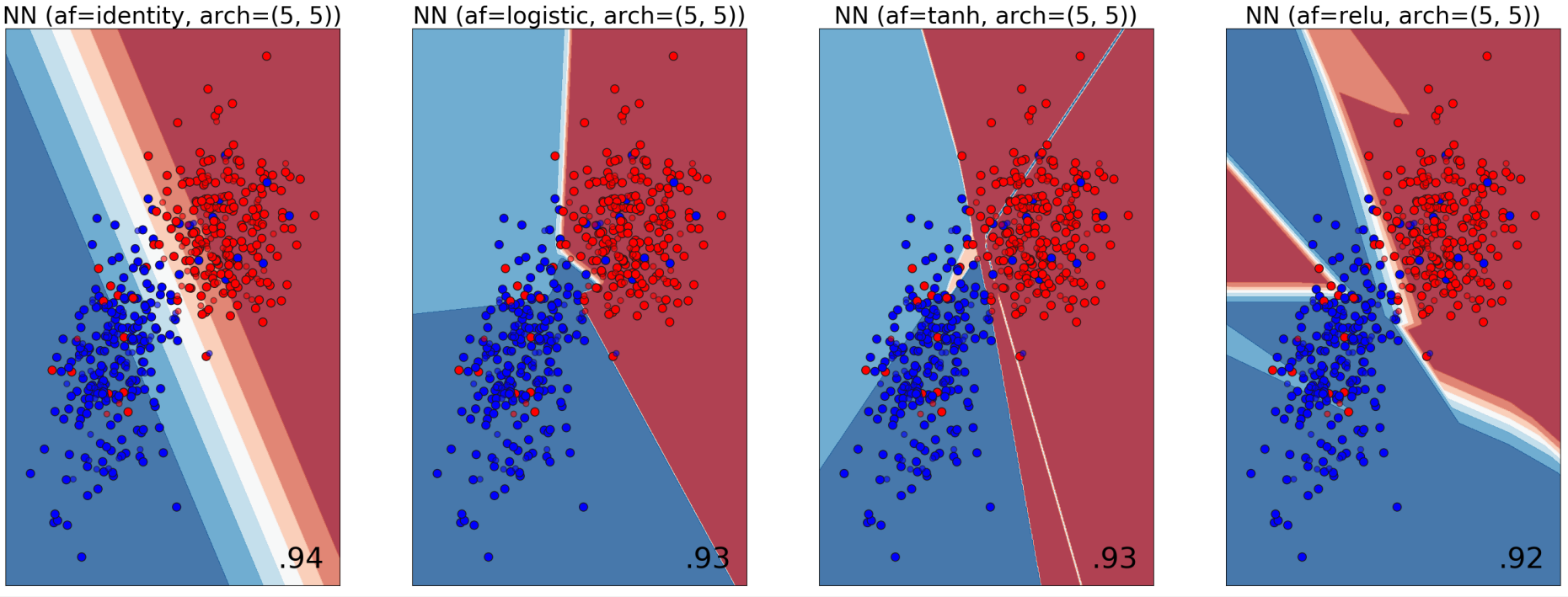
	Discriminative model	Generative model
Goal	Directly estimate $P(y\|x)$	Estimate $P(x\|y)$ to then deduce $P(y\|x)$
What's learned	Decision boundary	Probability distributions of the data
Illustration		
Examples	Regressions, SVMs	GDA, Naive Bayes

Stanford CS 229

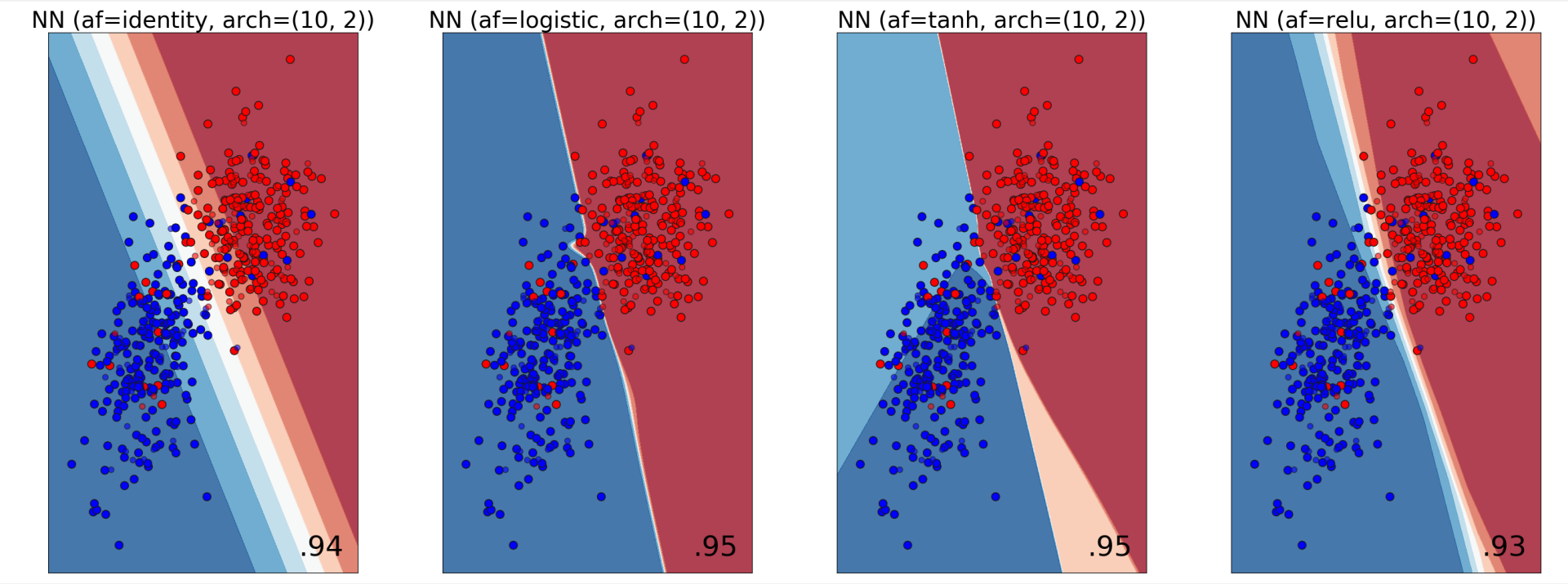
Marketing Dataset Example



Marketing Dataset Example



Marketing Dataset Example



A Simple Example

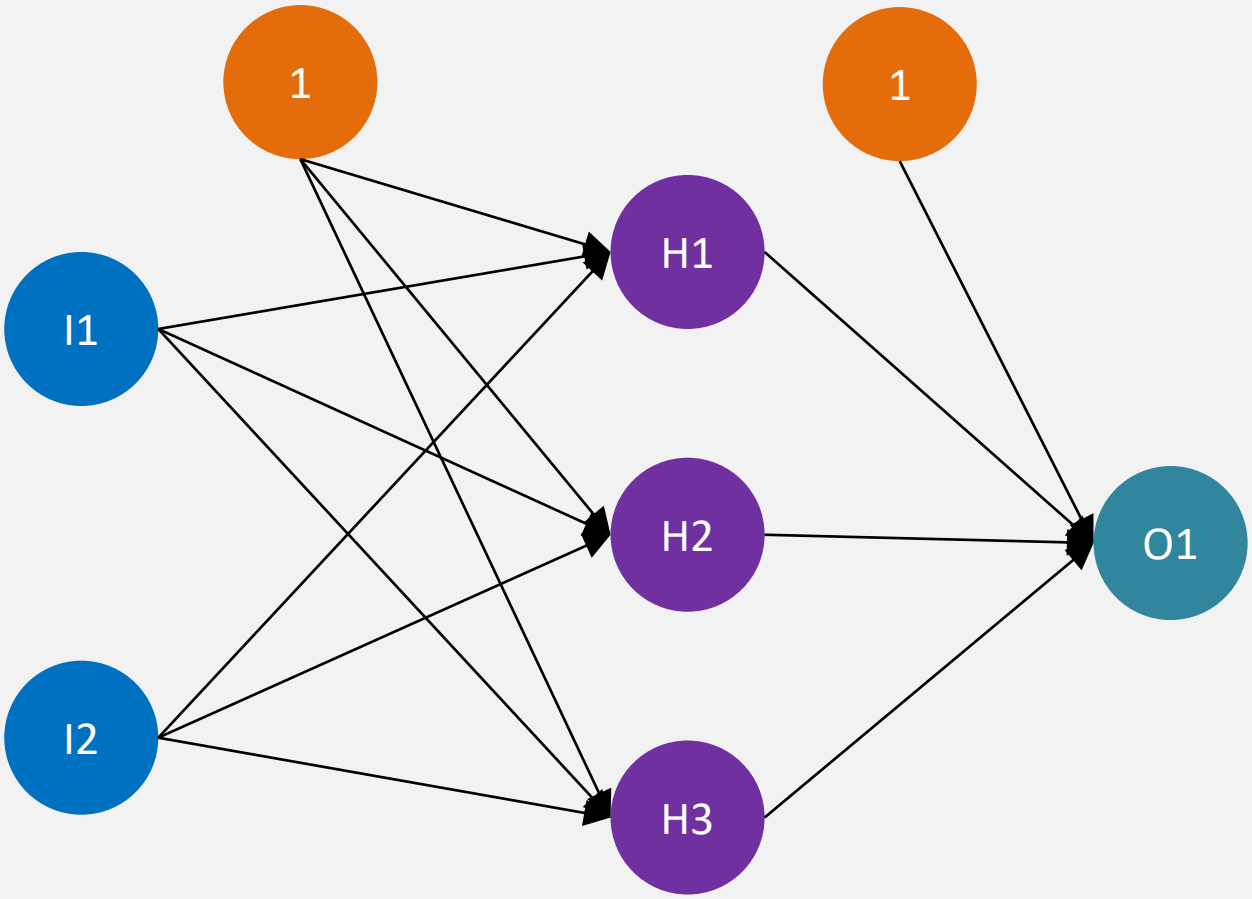
- Training data: two features, one target
- Architecture of neural network (chosen by me)
  - 2 input neurons, 3 hidden neurons, 1 output neuron
  - Activation function for hidden neurons = relu
    - f(x) = 1/(1+exp(-x))
  - Activation function for output neuron = identity
    - f(x) = x
  - Error function = sum squared error: e(x, y) = ½ * (y - x)^2

calories	protein	rating
0.636	0.400	0.367
0.455	0.600	0.486
0.455	0.400	0.286
0.545	0.200	0.200
0.909	0.600	0.252
0.182	0.600	0.666
0.636	0.200	0.157
0.545	0.200	0.278
0.455	0.600	0.361
0.727	0.400	0.164

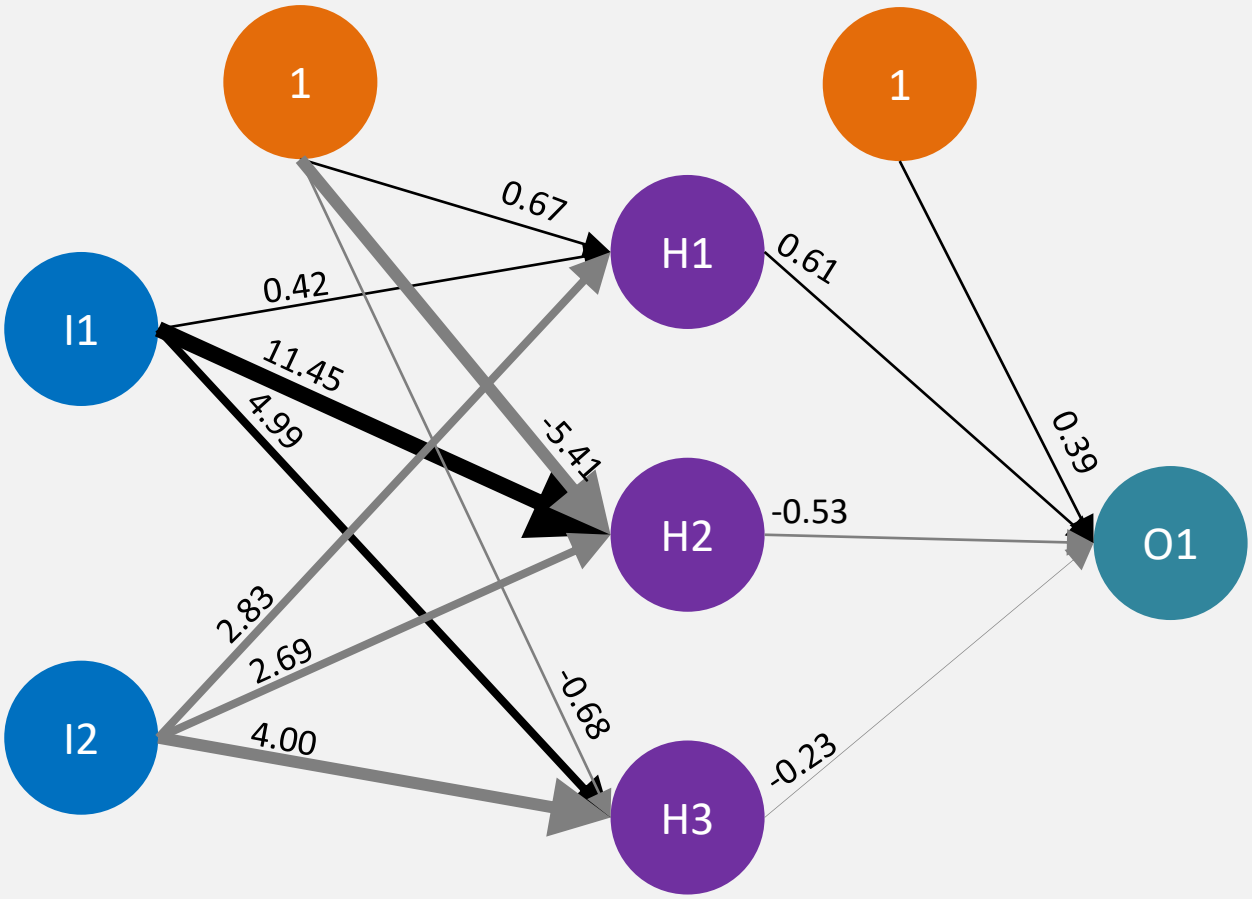
Training data
- Range normalized
- From *cereal* dataset

A Simple Example

- The architecture

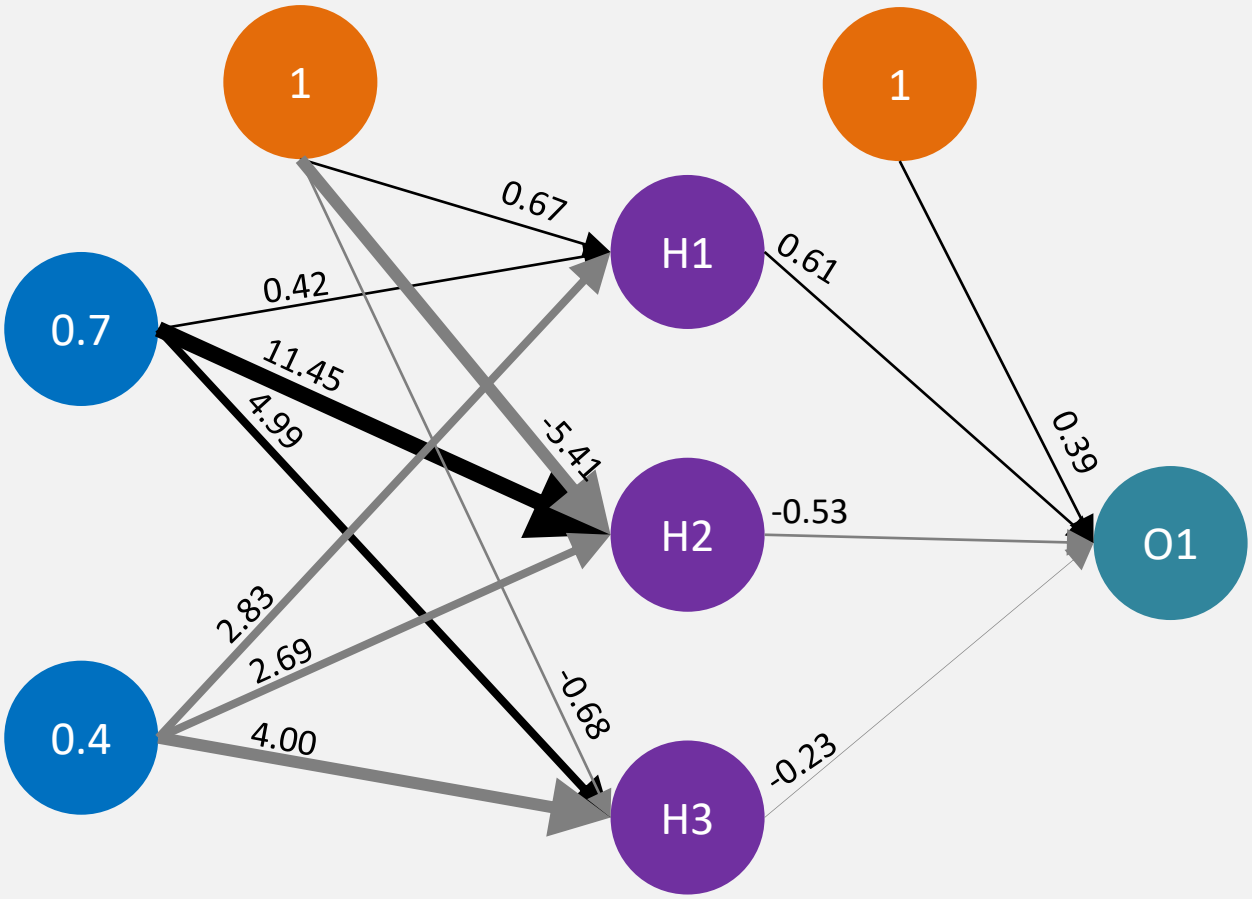


A Simple Example



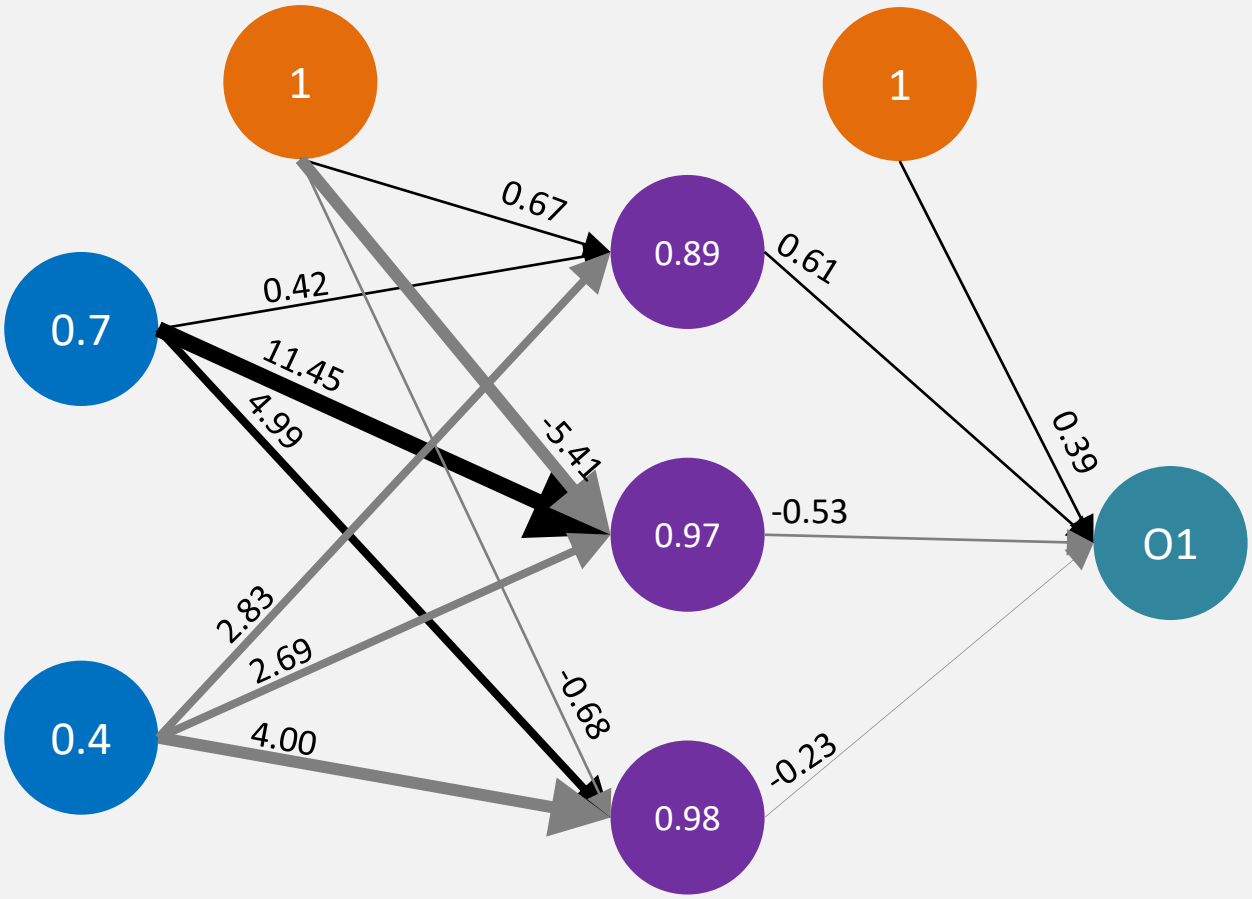
- The architecture
- The learned weights after training (i.e., backpropagation)

A Simple Example



- The architecture
- The learned weights after training (i.e., backpropagation)
- Want to classify new cereal
  - calories=0.7
  - protein=0.4

A Simple Example



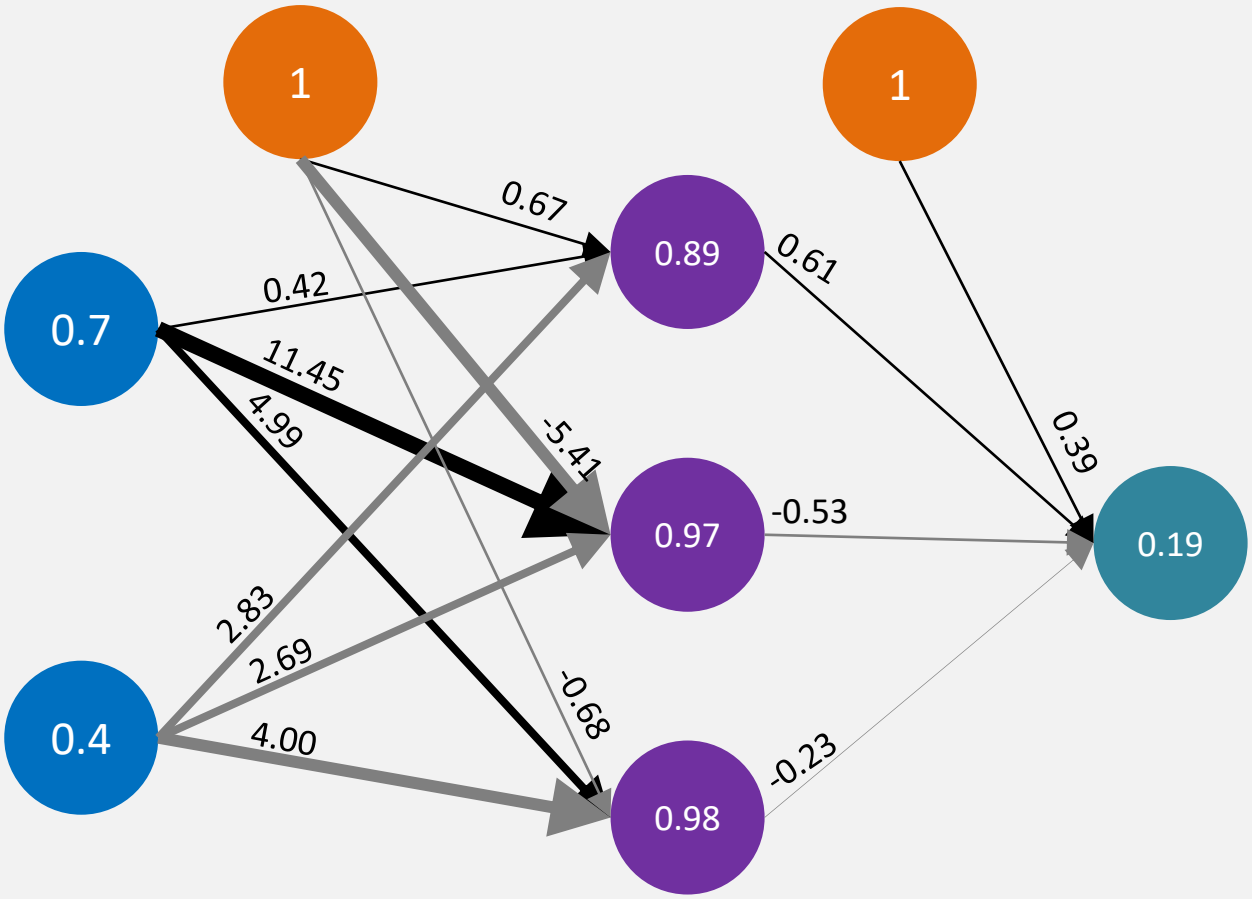
- The architecture
- The learned weights after training (i.e., backpropagation)
- Want to classify new cereal
  - calories=0.7
  - protein=0.4
- Compute hidden neurons

H1 = f(1*0.67 + 0.7*0.42 + 0.4*2.83) = 0.89

H2 = f(1*-5.41 + 0.7*11.45 + 0.4*2.69) = 0.97

H3 = f(1*-0.68 + 0.7*4.99 + 0.4*4.00) = 0.98

A Simple Example



- The architecture
- The learned weights after training (i.e., backpropagation)
- Want to classify new cereal
  - calories=0.7
  - protein=0.4
- Compute hidden neurons
- Compute output neuron

H1 = f(1*0.67 + 0.7*0.42 + 0.4*2.83) = 0.89

H2 = f(1*-5.41 + 0.7*11.45 + 0.4*2.69) = 0.97

H3 = f(1*-0.68 + 0.7*4.99 + 0.4*4.00) = 0.98

O1 = f(1*0.39 + 0.89*0.61 + 0.97*-0.53 + 0.98*-0.23) = 0.19

Marketing Dataset Example

```
from sklearn.neural_network import MLPClassifier

nn_clf = MLPClassifier(solver='lbfgs', activation='relu', alpha=1e-3,
                       hidden_layer_sizes=(3), random_state=1, verbose=True)
nn_clf.fit(X_train, y_train)

y_pred_nn = nn_clf.predict(X_test)
```

```
w = nn_clf.coefs_ # The ith element in the list represents the weight matrix corresponding to layer i.
w
```

```
[array([[ 0.9756526 ,  1.08832583, -1.04406553],
        [-1.66548065, -1.62569205, -0.61746919]]), array([[ 3.91733101],
        [-2.57062869],
        [ 1.76672827]])]
```

```
b = nn_clf.intercepts_ # The ith element in the list represents the bias vector corresponding to layer i + 1.
b
```

```
[array([-2.08114865, -1.19151499,  1.62527184]), array([-2.59817514])]
```

Marketing Dataset Example

```
print(ConfusionMatrix(y_test, y_pred_nn))
```

```
Predicted  False  True  __all__
Actual
False         51     4       55
True           2    43       45
__all__       53    47      100
```

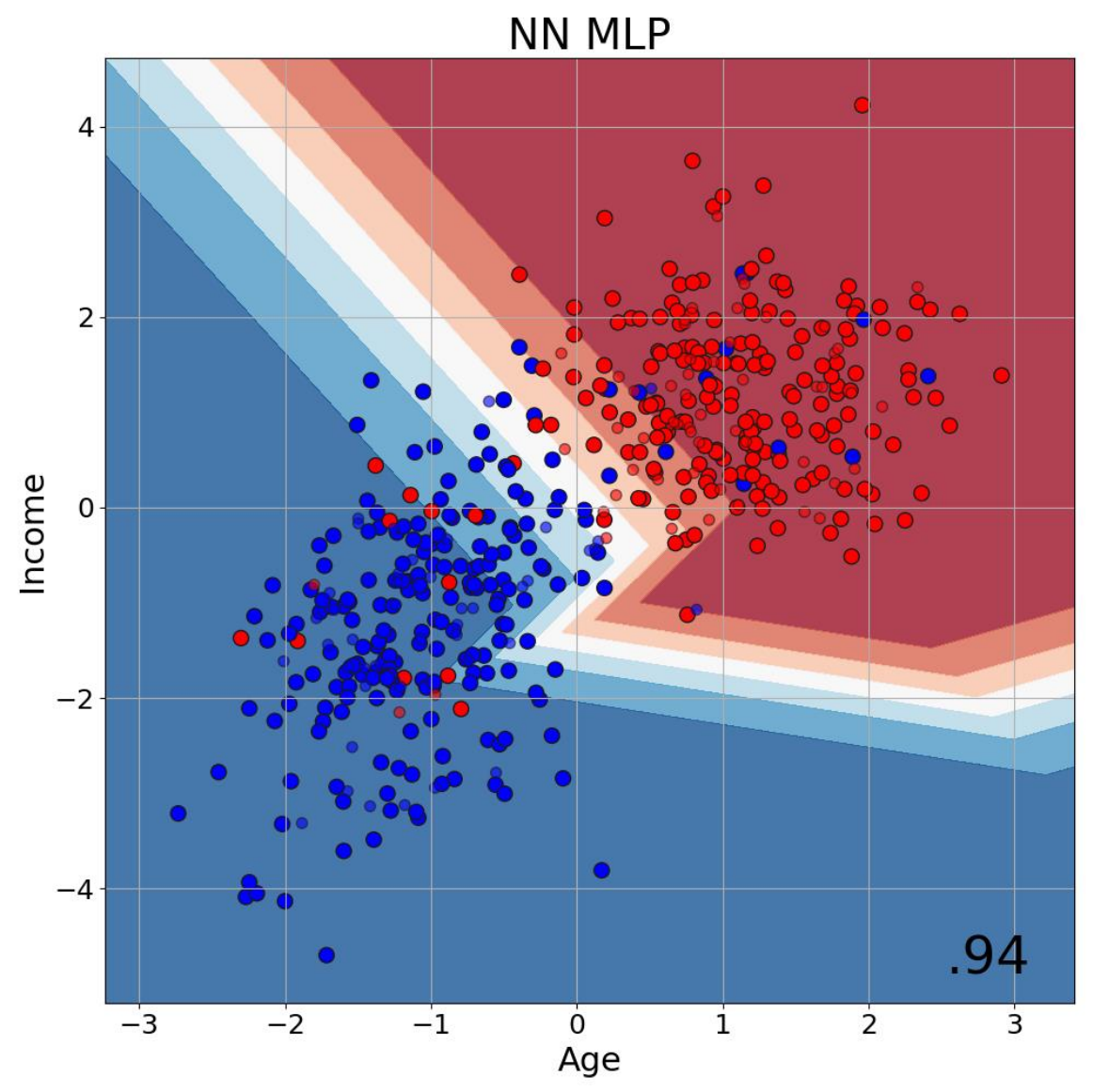
```
print(classification_report(y_test, y_pred_nn, target_names=class_names))
```

```
             precision    recall  f1-score   support

          0       0.96      0.93      0.94        55
          1       0.91      0.96      0.93        45

avg / total       0.94      0.94      0.94       100
```

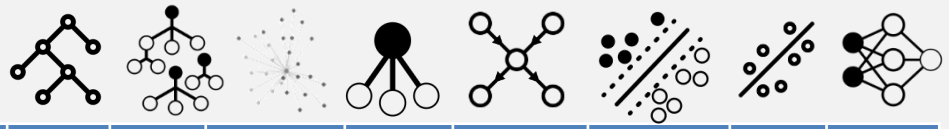
```
print("Accuracy = {:.2f}".format(accuracy_score(y_test, y_pred_nn)))
print("Kappa = {:.2f}".format(cohen_kappa_score(y_test, y_pred_nn)))
print("F1 Score = {:.2f}".format(f1_score(y_test, y_pred_nn)))
print("Log Loss = {:.2f}".format(log_loss(y_test, y_pred_nn)))
```

```
Accuracy = 0.94
Kappa = 0.88
F1 Score = 0.93
Log Loss = 2.07
```

Marketing Dataset Example



Awesome R Packages



Package	Description	DT	RF	KNN	NB	BBN	SVM	LR	NN
caret	Classification and Regression Training. 233 different algorithms. Handy functions for evaluation, etc.	✓	✓	✓	✓		✓	✓	✓
rpart	Recursive partitioning for classification, regression, and survival trees	✓							
e1071	Various classification and clustering algorithms				✓		✓		
PARTY	All about decision trees	✓	✓						
randomForest	All about random forests		✓						
nnet									✓
sparklyr	Has several machine learning functions, implemented in Spark	✓	✓		✓			✓	✓
bnlearn	Bayesian network structure learning, parameter learning and inference					✓			
keras	Interface to keras								✓
tensorflow	Interface to TensorFlow								✓

Overfitting Example