Using Machine Learning like a responsible adult

Ben Sunshine-Hill and Aline Normoyle

Havok

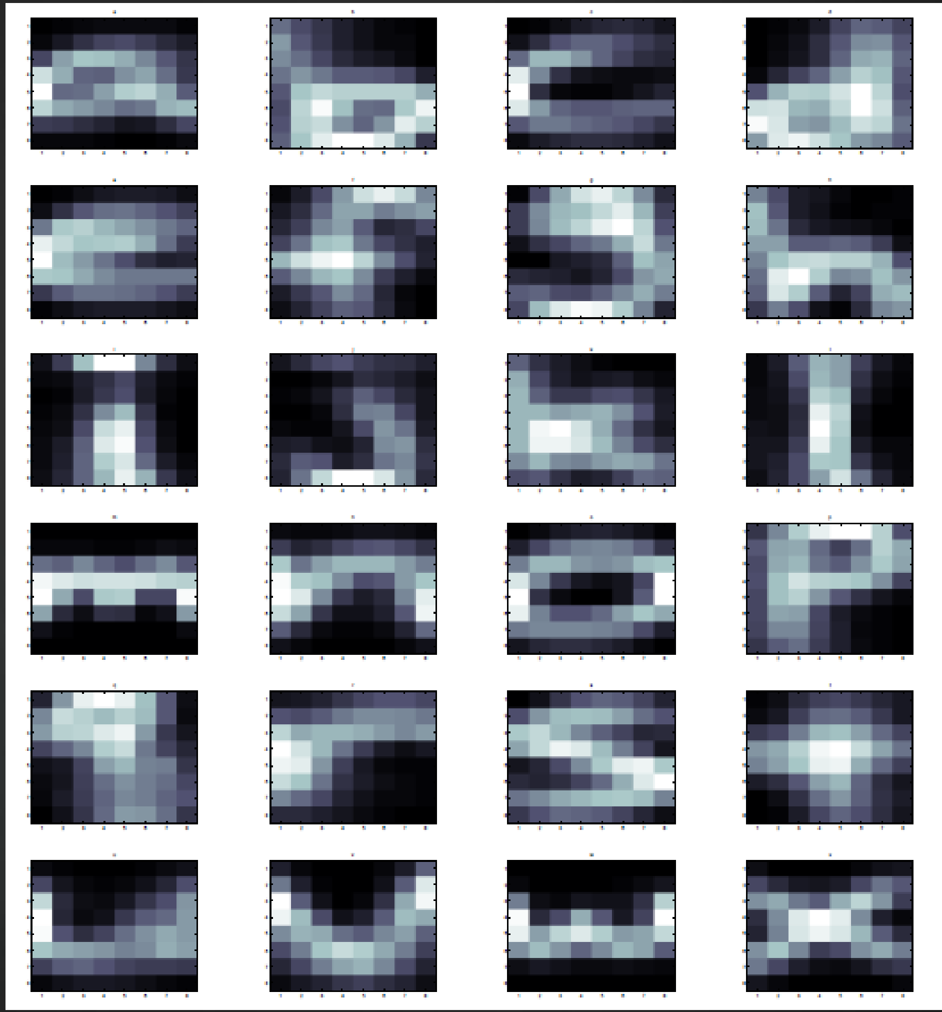
University of Pennsylvania

Why *not* use machine learning?

- Too slow
- Too opaque
- Too unreliable

Slow?

Opaque?



Unreliable?


defibtech
Lifeline AUTO
DEFIBRILLATOR
check pads
do not touch patient
analyzing

Maybe it's you

- Few game AI programmers are skilled enough at ML to effectively evaluate it
  - They teach programmers about Neural Networks and Genetic Algorithms, because they're easy, and cool
  - They teach statisticians all the other stuff
- Effective ML requires stepping outside your comfort zone

ML can be really useful

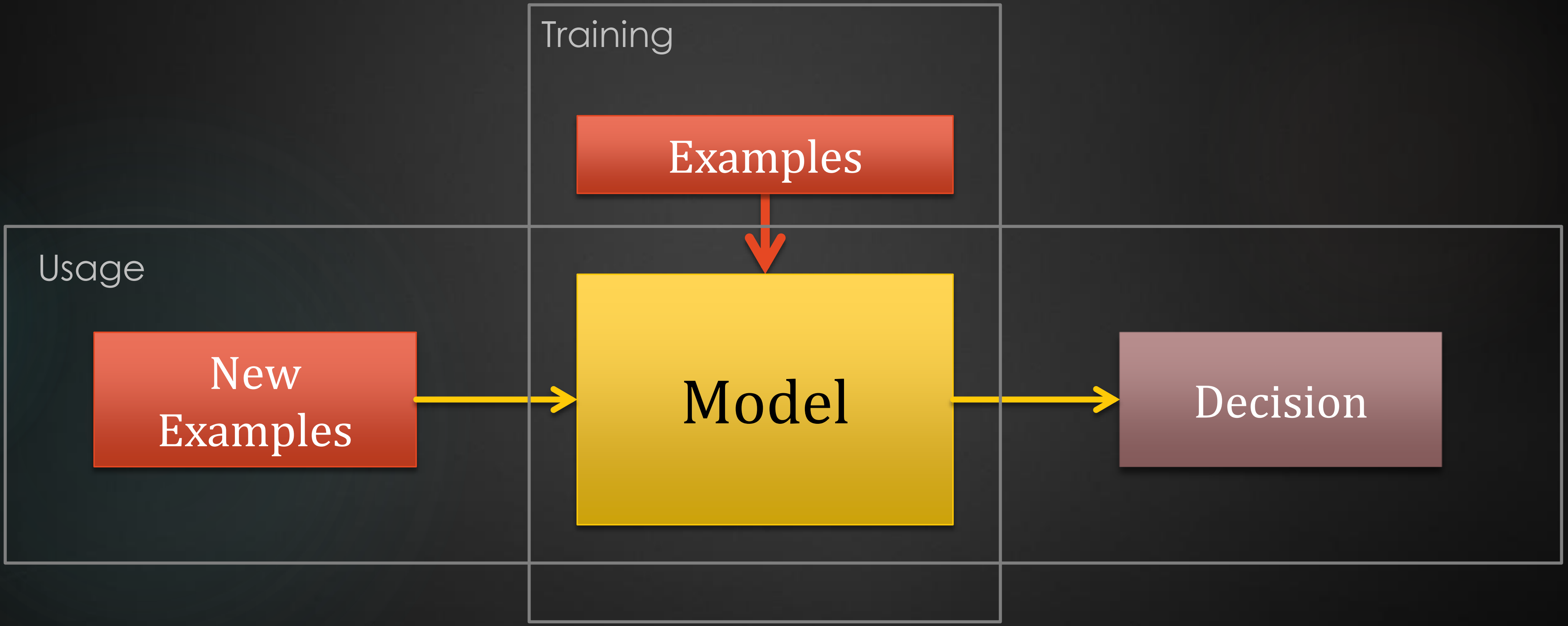
- ML can solve problems which are not easily coded up directly
  - Based on what we've seen, what is the underlying process?

- Replace manual tweaking with automated refinement
- Turn gameplay traces into bots
- Tons of neat stuff

Before we get started, some terminology…

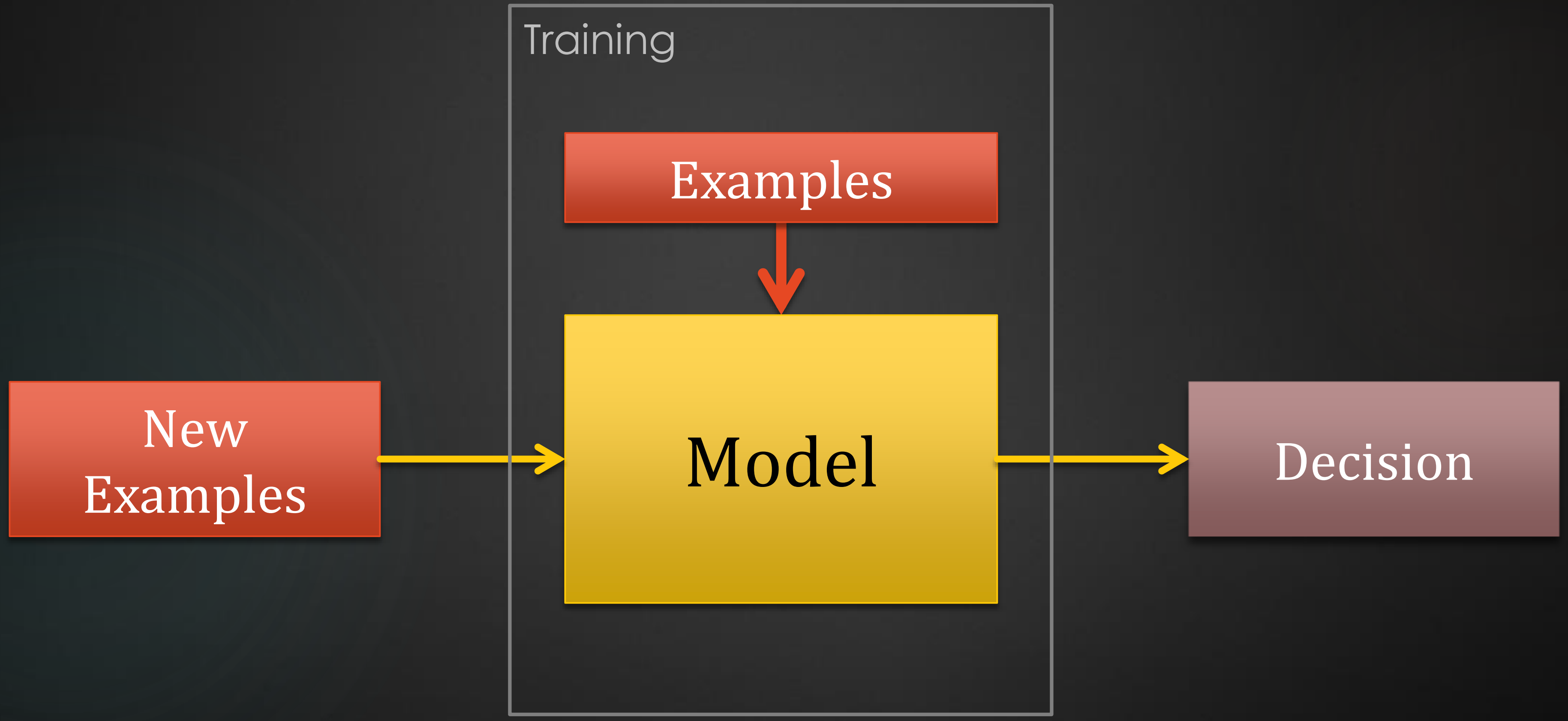
Primary goal is generalizability

Based on examples, how to **learn** a model which allows us to **predict**, **classify**, or **cluster** new examples?



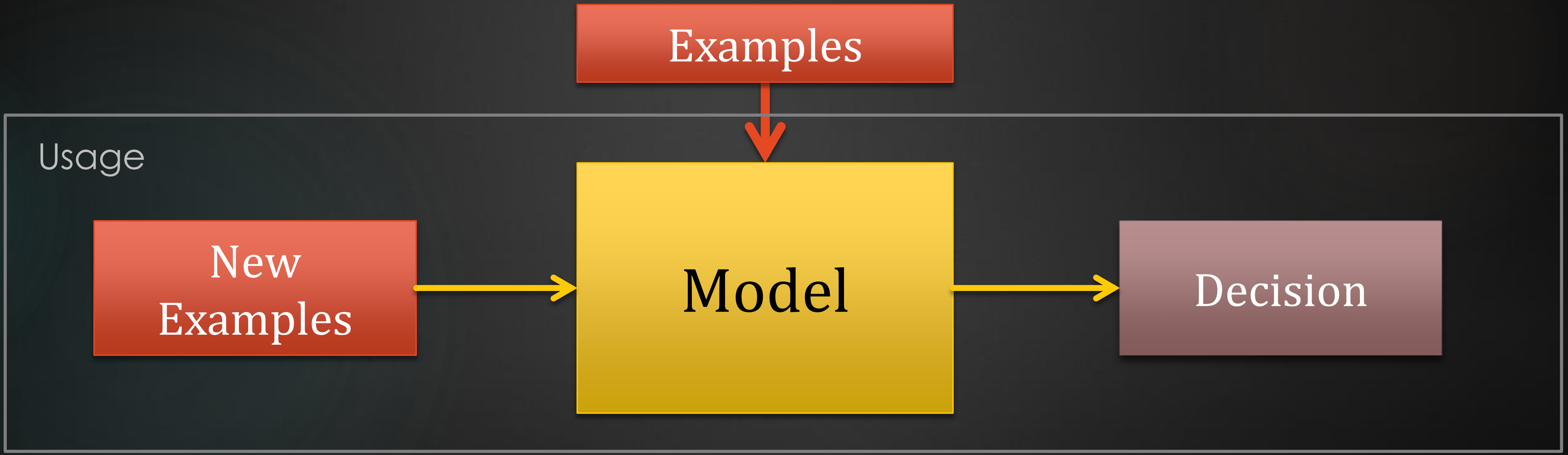
Primary goal is generalizability

First step is to **train** the model using the examples we have already



Primary goal is generalizability

The trained model is then tried on **new examples** it's never seen before

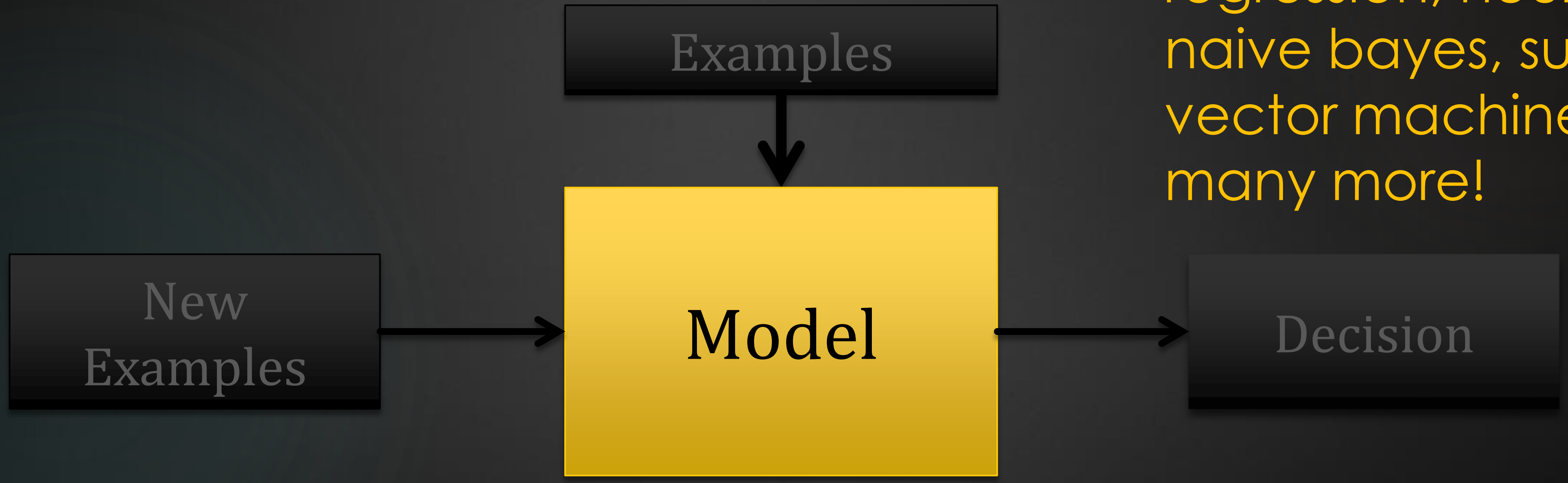


Models

Representation of the underlying process

Encodes how inputs relate to output

Examples: Decision trees, k-NN, linear regression, neural nets, naive bayes, support vector machines, and many more!

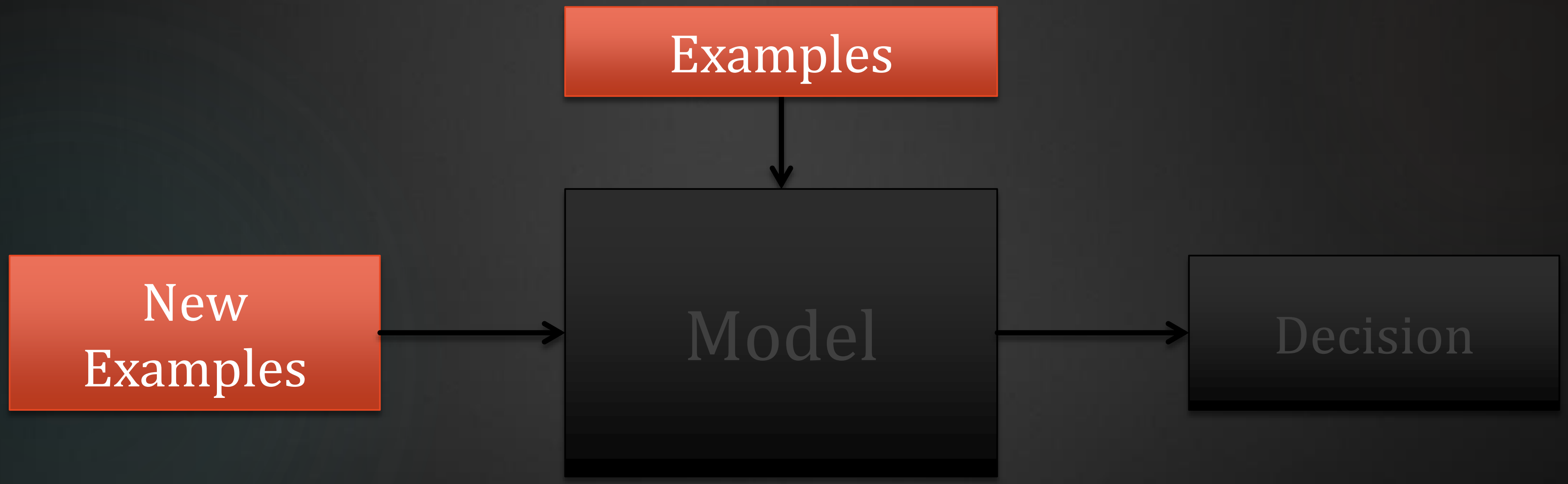


havok

Features

ML inputs are called **features**

Features are typically stored together in big feature vectors

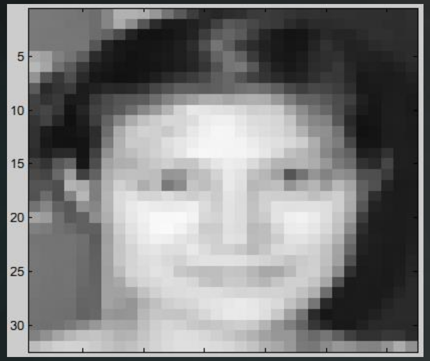


Features

ML inputs are called **features**

Features are typically stored together in big feature vectors

Example: Image features



32x32 pixel image

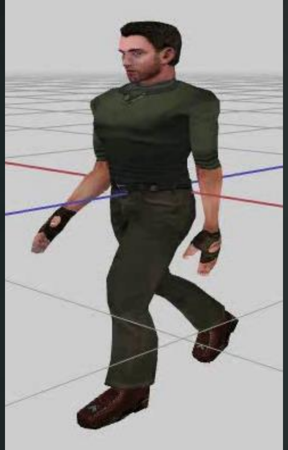
1x1024 feature vector

Features

ML inputs are called **features**

Features are typically stored together in big feature vectors

Example: Motion feature



(Keyframe1, Keyframe2, Keyframe3, Keyframe4, Keyframe5)

where each Keyframe = (Joint1_Rotation, ..., Joint33_Rotation)

5 keyframe motion

Features

ML inputs are called **features**

Features are typically stored together in big feature vectors

Example: Emails

IT TRAINING TUITION SCHOLARSHIPS FOR COLLEGE FACULTY, STUDENTS AND STAFF

National Education Foundation CyberLearning, a non-profit organization dedicated to bridging the Digital Divide since 1994, is offering "No Excuse" tuition-free on-

(word1_count, word2_count, ..., wordM_count)

Features in matrix form

X is our features. This can be either our training set or new examples we've never seen before.

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & \ddots & & \\ & & & \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{bmatrix}$$

X has dimensions N x M
(N examples, M features)

Features in matrix form

Each row is an example

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & \ddots & & \\ \vdots & & & \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{bmatrix}$$

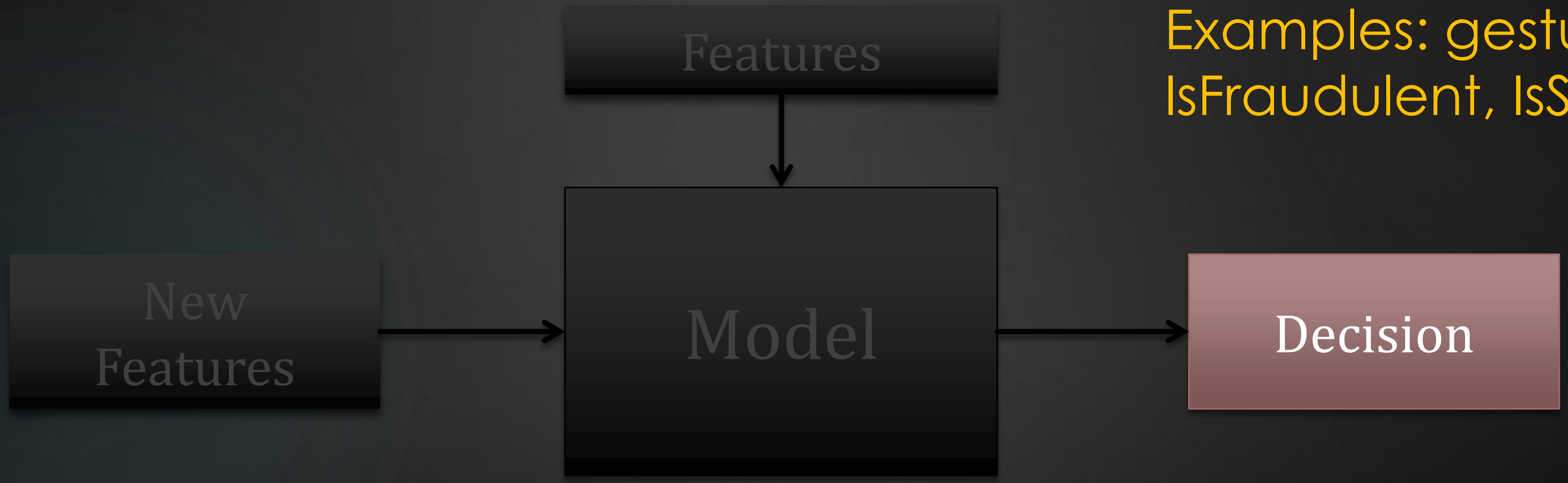
Features in matrix form

Each column is a feature

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1M} \\ x_{21} & \ddots & & \\ \vdots & & & \\ x_{N1} & x_{N2} & \cdots & x_{NM} \end{bmatrix}$$

Labels

ML outputs are often called **labels**, particularly for classification



Examples: gesture type, IsFraudulent, IsSpam

Labels in matrix form

Like features, labels can be collected together in a vector, with each row corresponding to an example.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Useful techniques

Types of learning

- **Supervised:** Given a set of questions and correct answers, can we answer new questions correctly?
  - Observations: features, labels
- **Unsupervised:** Can we find structure in a given dataset?
  - Observations: features
- **Reinforcement learning:** Can we learn to perform a task better over time?
  - Observations: states over time, reward function

Decision trees

Automatic decision tree learning

NPC HP	Hair color	Player stunned?	What to do?
88	Blue	No	**Attack**
23	Blue	No	**Retreat**
60	Red	Yes	**Attack**
40	Green	Yes	**Attack**
15	Red	No	**Retreat**
⋮	⋮	⋮	⋮

Decision trees are white boxes

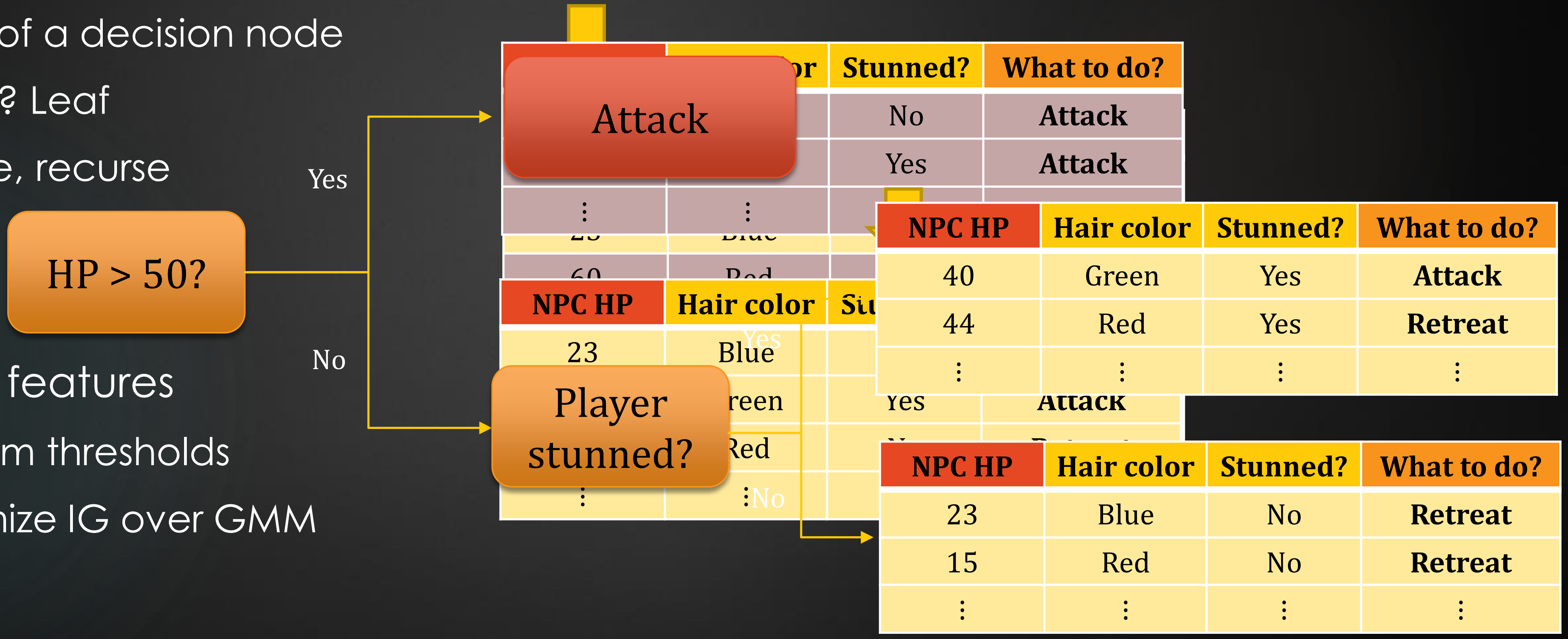
- Tells you what it's thinking
- Debug bad outputs
  - Chain of decisions
  - Relevant training examples
- Tweakable
  - Snip branches as desired

Black-box neural network

Input layer	hidden layer (nodes)					output layer (nodes)	
node/ weight	1st	2nd	3rd	4th	5th	1st	2nd
0	-0.204716	1.533574	1.452831	0.129981	-1.784807	0.854229	-0.883808
1	-1.843673	1.957059	-2.668371	-0.551016	1.505628	-5.294533	5.303048
2	-1.324609	0.258418	-1.280479	-0.476101	0.827188	-7.468771	7.514580
3	-1.281561	1.697443	6.865219	4.212538	-1.953753	-5.082050	5.003566
4	-1.159086	-0.345244	-4.689749	-0.406485	1.027280	4.014138	-4.006929
5	-2.042978	0.182091	2.612433	2.399196	-1.397453	-4.105859	4.105161
6	-4.076656	1.416529	0.979842	-2.589272	0.068466		
7	-0.499705	-1.383732	-2.411544	0.173131	-1.919889		

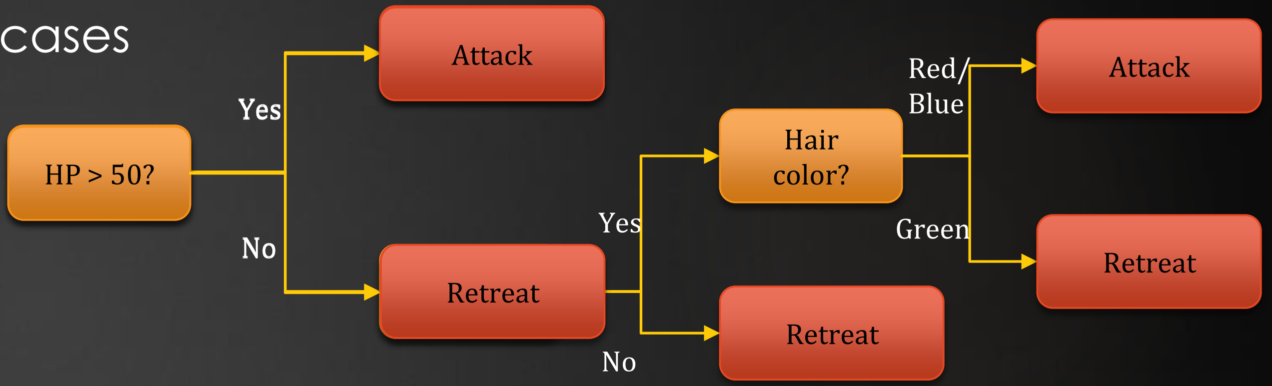
Build decision trees with ID3

- Choose the most important feature
  - Separate the output labels as cleanly as possible
- Divide examples based on that feature
  - Children of a decision node
  - All agree? Leaf
  - Otherwise, recurse
- Continuous features
  - Try random thresholds
  - Or maximize IG over GMM



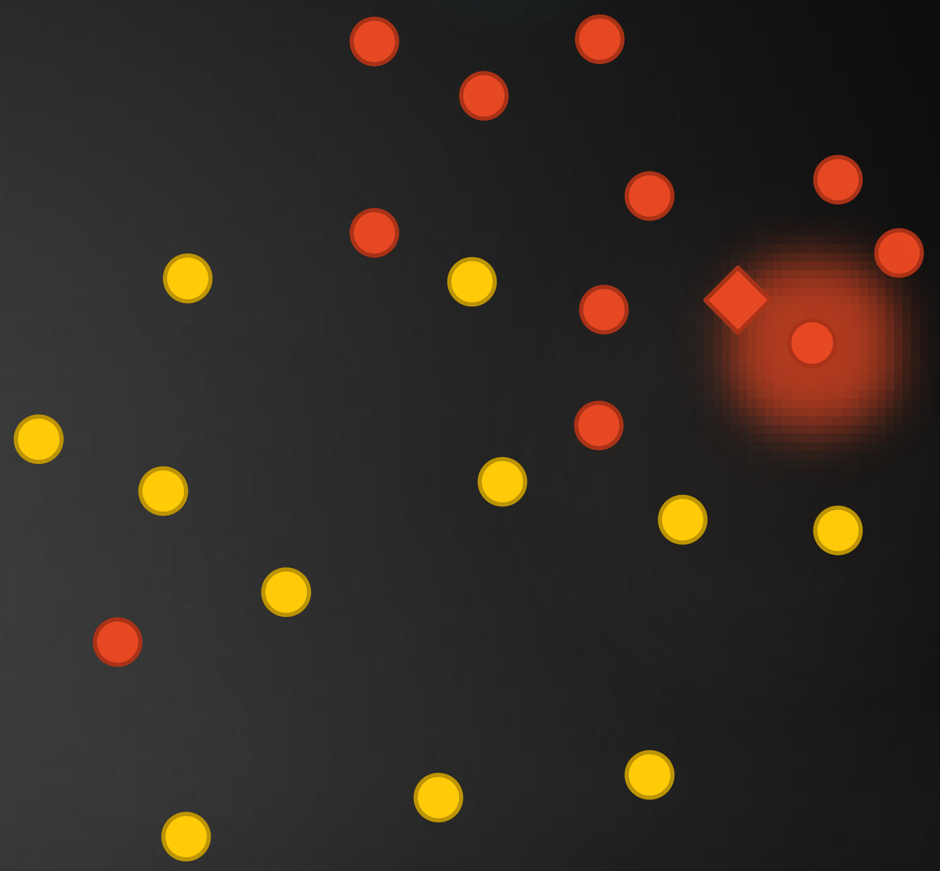
Drawbacks of decision trees

- Difficult to tune complexity
  - Too complicated → fixate on irrelevant features
  - Too simple → fail to consider special cases
- Can't relate continuous features
  - Retreat if HP < ATT
- Still awesome



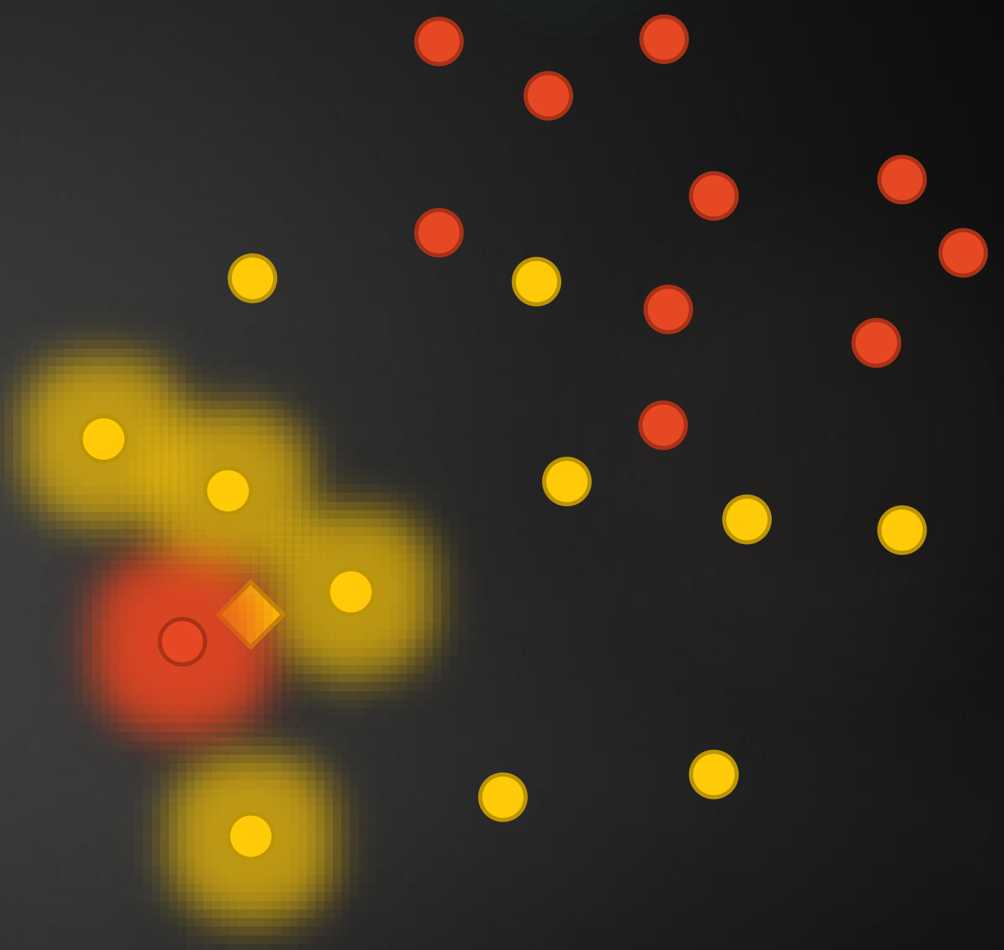
Nearest Neighbor

- No training process
  - The model is the training set
- Procedure
  - Find most similar training example
    - "Closest"
  - Use its label



k-Nearest Neighbors

- Because regular NN sucks
  - Overfitting
- Find closest k examples
  - They vote on what label wins
  - Closer examples get a bigger vote?
- Higher k
  - Paves over weird training examples
  - Doesn’t respect genuine special cases



Problems with kNN

- High dimensionality is a real problem
  - Low dimensional → Use kD trees
  - High dimensional → Brute force
- Distance metric
  - Scaling is important
  - Distance between "orc" and "goblin"?
- Good with low-dimensional sets with clean training data

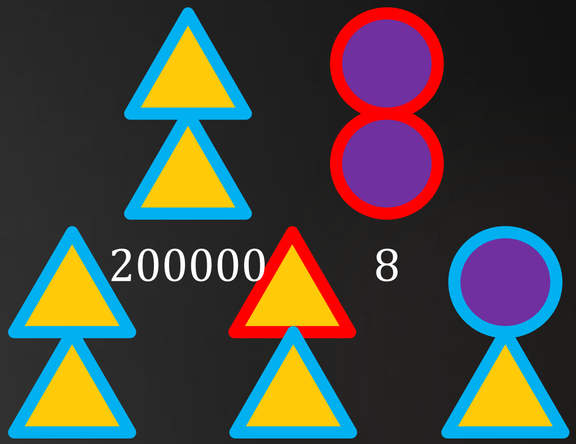
Genetic algorithms

- Stuff where
  - Bunch of potential solutions
  - They do battle with a black box
  - The survivors have sex
  - Their kids mutate a little
  - Keep doing more generations
    - Until optimum reached

- Use it to make your model!

Selecting genes for the next generation

- Roulette wheel selection
  - Randomly, weighted by each solution's fitness score
  - Relies on well-behaved fitness score
- Rank selection
  - Randomly, weighted by each solution's fitness **rank**
  - Avoids "crowding out" in early generations
  - Slower convergence



Pitfalls of GAs

- Slower and less effective than model-specific optimization methods
- Can be difficult to tweak
- A backup plan

Things that go wrong:

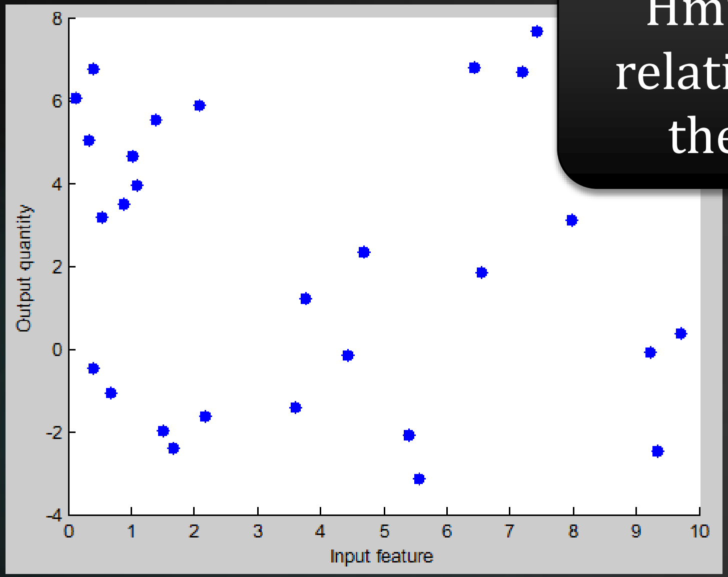
The wrong features

The wrong features

Situation: You've tried a lot of different models, but keep getting disappointing results

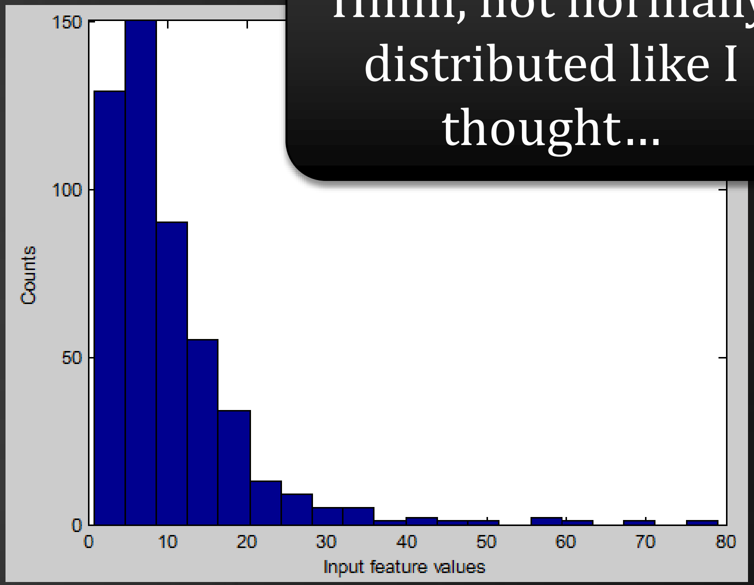
The wrong features

- Solution: Look at your data!
  - Do exploratory data analysis (EDA)



Hmm, no clear relationship with the outcome

Scatter plots to see relationships



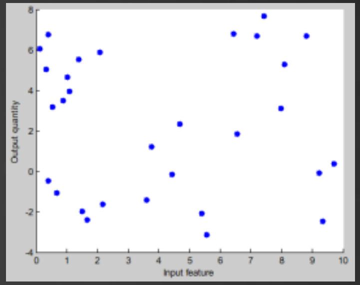
Hmm, not normally distributed like I thought…

Histograms to understand distributions

The wrong features

- Solution: Boil down your data
  - Eliminate irrelevancies

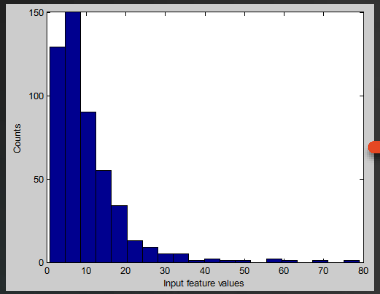
Remove features like these



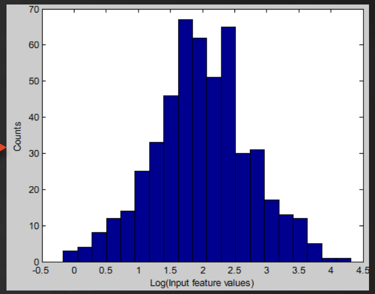
Crop

The wrong features

- Solution: Look at your data!
  - Check whether transformations of your data help

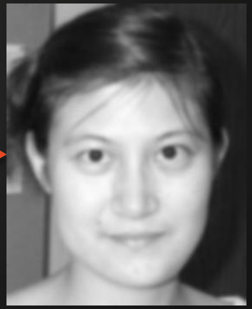


Log transform



Desaturate

Blur



The wrong features

- Solution: Look at your data!
  - Make sure features all have comparable scale

(weapon_power, player_level, gold_amt)

Range: [10-50] Range: [1-100] Range: [0-2,000,000]

Distance metrics will be dominated by gold_amt!
Solution: transform gold_amt to adjust scale

The wrong features

Situation: You're feeding in 50,000 features, and your classifier sucks. It worked better when it was only 100 features.

The wrong features

- What's going on?
  - **Curse of dimensionality!**
    - Everything is far apart
    - As the feature space grows, you need more examples to understand it

The wrong features

- Solution: Reduce the dimensionality
  - Automatic methods such as Principal Component Analysis (PCA) can help

In a stylistic walking motion dataset, PCA reduces motion examples by 94%

Original motion based on 540 features

PCA-transform motion based on 29 features

The wrong features

Situation: You have insanely good accuracy on the test set but the model is terrible in practice

The wrong features

- Possible problem: Contamination
  - Some of your test data snuck into the training set
  - Check and fix your code

The wrong features

- Possible problem: Data Leakage
  - A feature not available for prediction was used for training the model

LOG FILE:

Player_LVL	#Kills	Weapon_power	Score
88	56	100	**10206**
23	24	30	**2413**
20	18	35	**1915**
45	42	60	**7049**
3	5	5	**450**
⋮	⋮	⋮	⋮

If Score = function(#Kills),

Using #Kills to predict score is <u>cheating</u>!

The wrong features

- Possible problem: Sampling bias
  - The training data is not similar enough to real world
    - Decisions of how, what and when you log data can matter

Ex. Behaviors of players who log in everyday are likely different from players who log in once a week

Things that go wrong:

The wrong model

The wrong model

Situation: You've tried a lot of different features, but have disappointing results

The wrong model

- Solution: Try a different model
  - Actually, try lots of models…
    - WEKA to the rescue!!

Data mining software in JAVA
http://www.cs.waikato.ac.nz/ml/weka/

The wrong model

- Solution: Try an ensemble of models
  - boosting, stacking, bagging
  - Weak models working together can outperform a single, more sophisticated learner
  - Large ensemble models were the best performers in the **Netflix Prize**

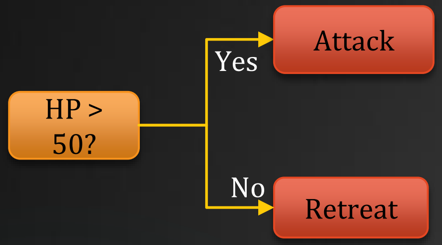
Things that go wrong:

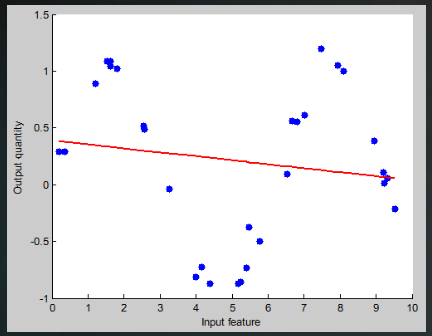
Overfitting

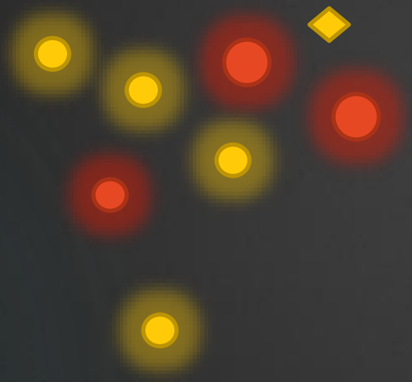
Overfitting

Situation: Your classifier has amazing accuracy with the training set but performs poorly on data it's never seen before

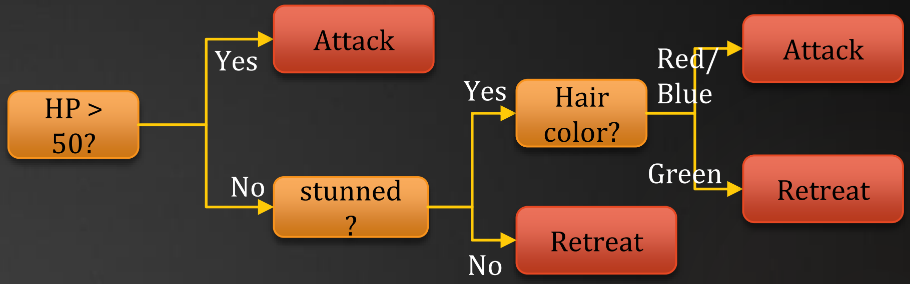
What's happening?

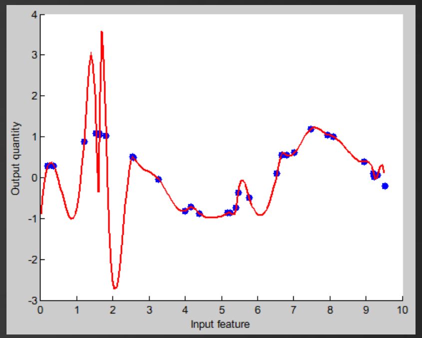


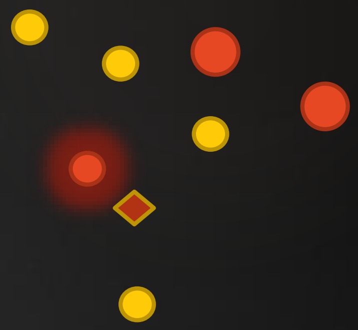




Model **too simple:** data patterns not captured







Model **too complex:** schizo fit with no ability to generalize

Overfitting

- Especially overfitty algorithms
  - k-NN w/ low k
  - ANNs w/ lots of neurons
  - decision trees with arbitrary depth
  - ensemble models

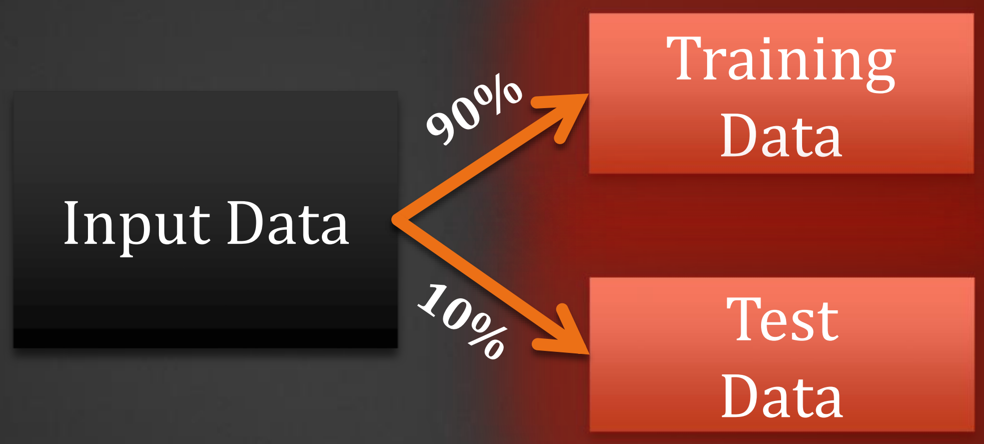
Overfitting

- Solution: Cross-validation
  - Estimate how well your model performs on new data
    - How? Hold-out subsets of your training data to use for testing
  - Try different model parameters to determine balance between simplicity and power

Overfitting

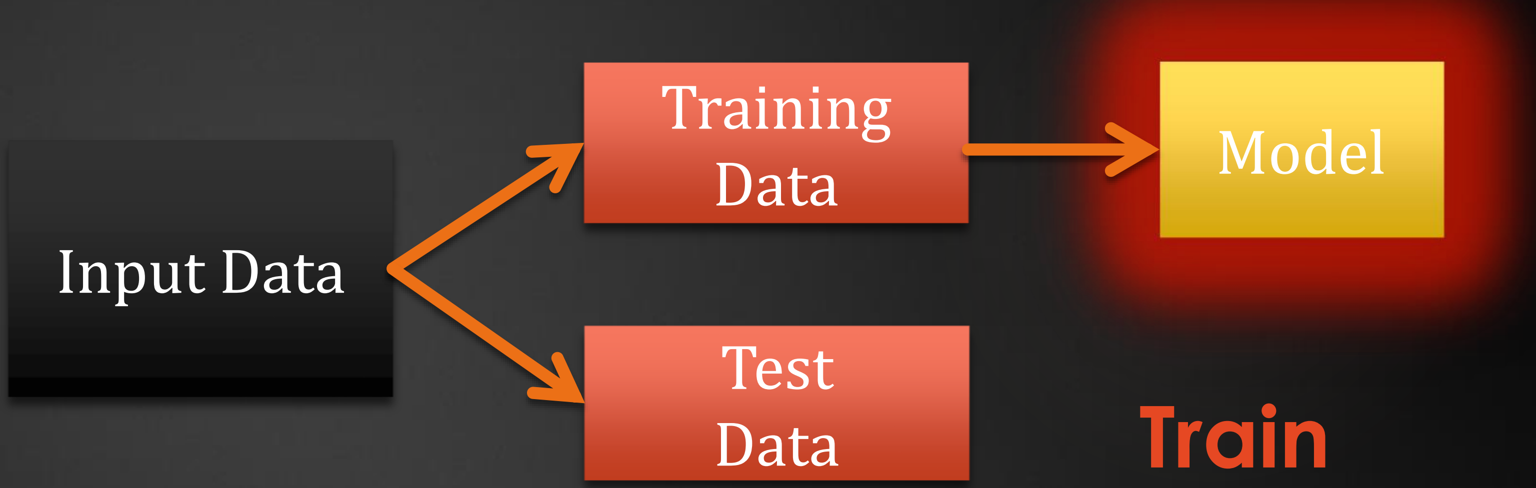
- Solution: Cross-validation
  - Step 1

Split examples randomly into training and test sets



Overfitting

- Solution: Cross-validation
  - Step 2

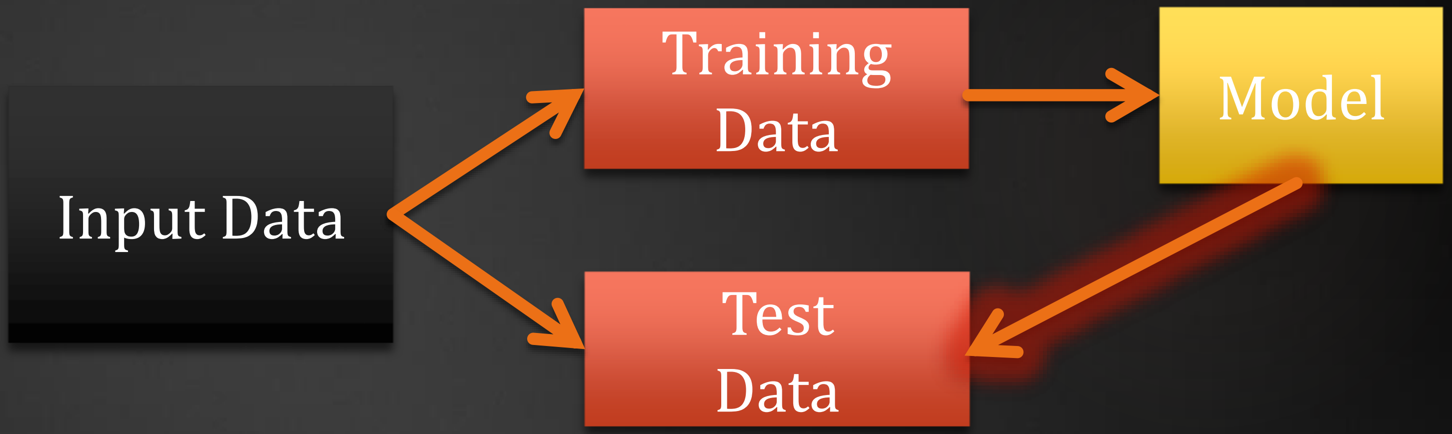


Train Model

Overfitting

- Solution: Cross-validation
  - Step 3

Evaluate Model's performance

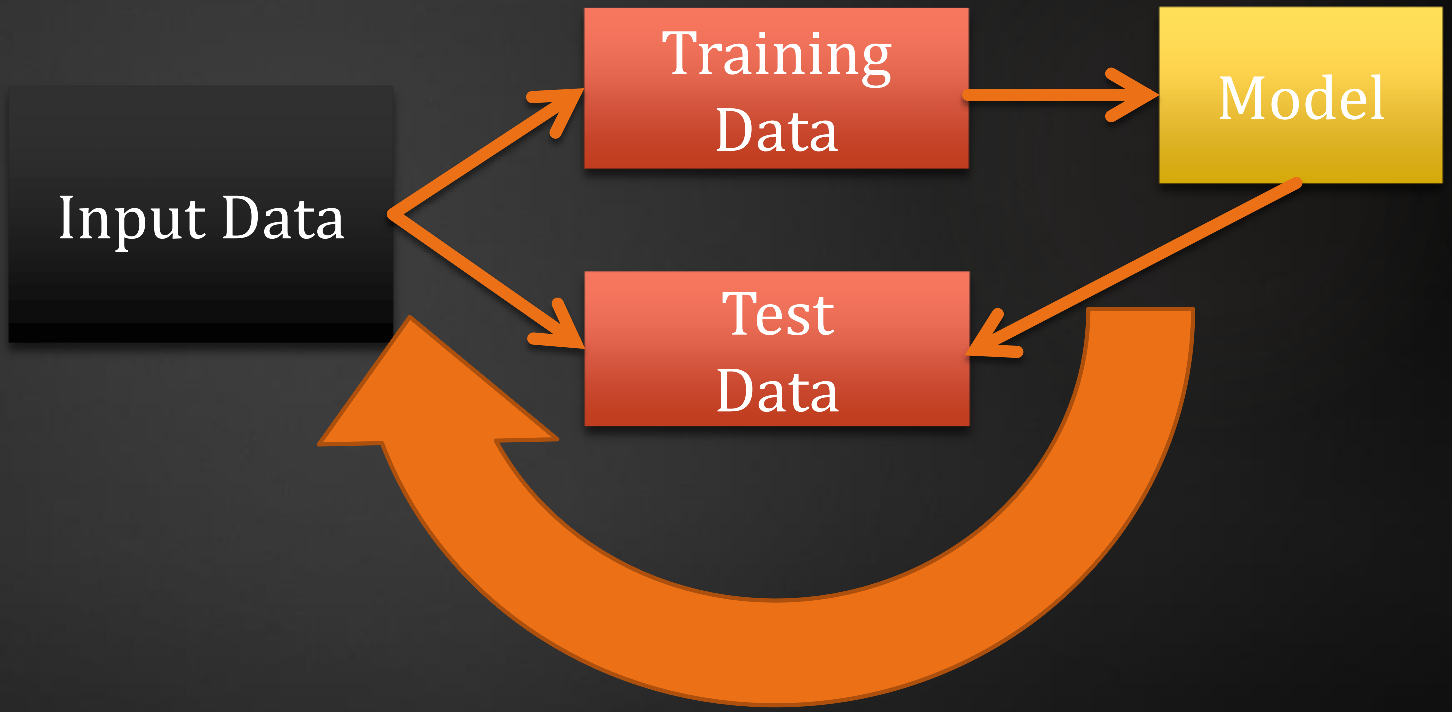


e.g. How much of the test set does it correctly classify?

Overfitting

- Solution: Cross-validation
  - Step 4: Repeat

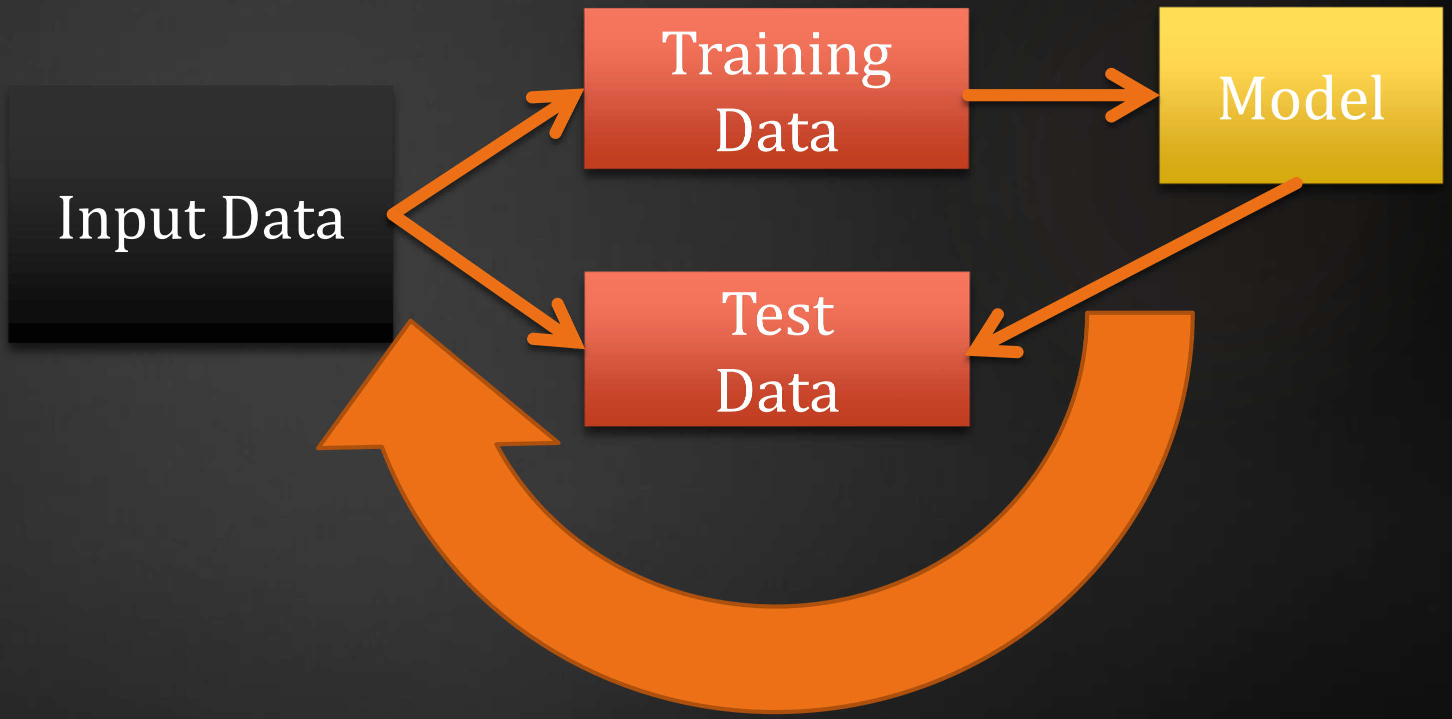
Split examples randomly into <u>new</u> training and test sets and reevaluate



Overfitting

- Solution: Cross-validation
  - Step 4: Repeat

Average over multiple test sets is estimate of performance



tl;dr

ML is powerful and useful

- ML can be real-time, transparent, and reliable
- ML can be the best use of your time
- Effective ML requires stepping outside your comfort zone
- Many straight-forward algorithms besides ANNs and GAs
- Effective ML requires understanding of features and models to work well

Going deeper

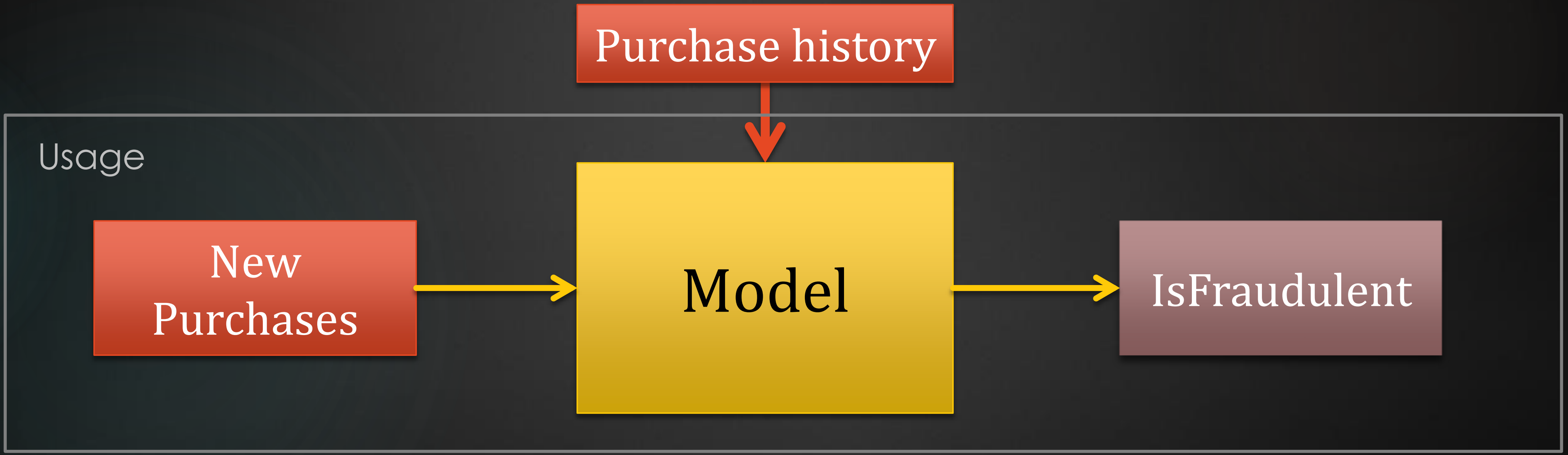
- Stanford's free online Machine Learning course
  - tiny.cc/MLcourse
- *A few useful things to know about machine learning*, Pedro Domingos, 2012
- *Doing Data Science: Straight talk from the frontlines*, Cathy O'Neil, Rachel Schutt

Extras

Primary goal is generalizability

The trained model is then tried on **new examples** it's never seen before

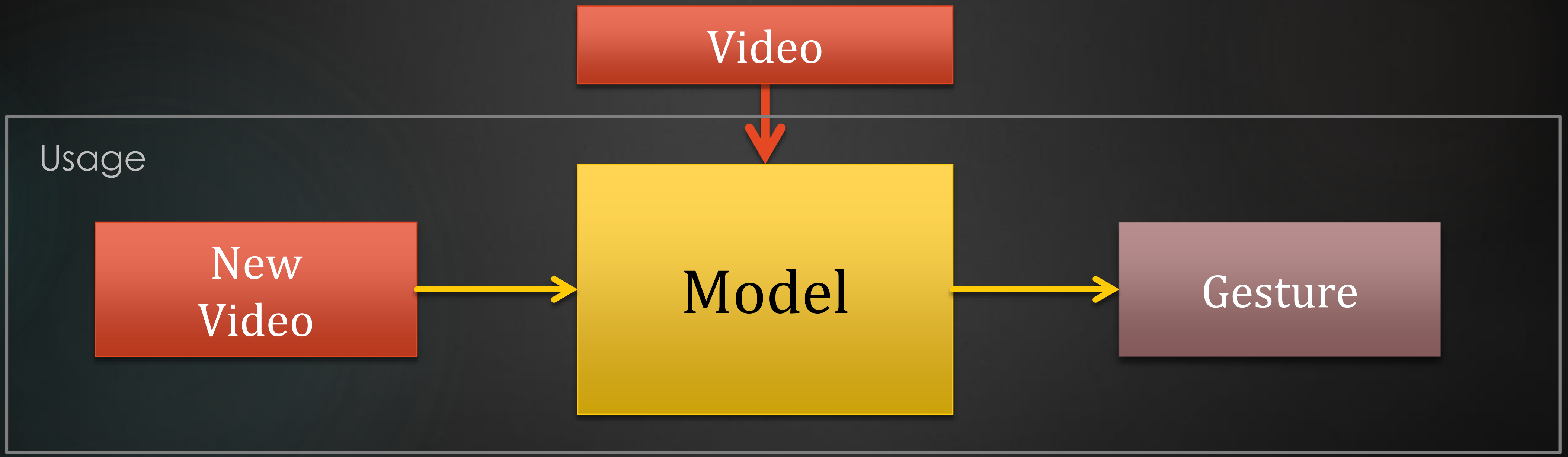
Example: Detect fraudulent purchases



Primary goal is generalizability

The trained model is then tried on **new examples** it's never seen before

Example: Recognize gestures



The wrong model

- Solution: Look at your data!
  - EDA is your friend
    - plot features against each other to gain intuition about what's happening
    - Are your model assumptions appropriate?

Overfitting

- Solution: Biasing, regularization
  - Limit the complexity of your model
    - Limit depth for Decision Trees
    - Specify a minimal value for k
    - Limit the degree polynomial for regression
- "Occam's Razor"
  - Make your model as simple as possible, but no simpler

How to train your algorithm

Observations
(Raw data)

Gather
your data

We want our learner to
understand this!

How to train your algorithm

Observations (Raw data) → Define Features

Format

Clean

Transform

Label

Etc.

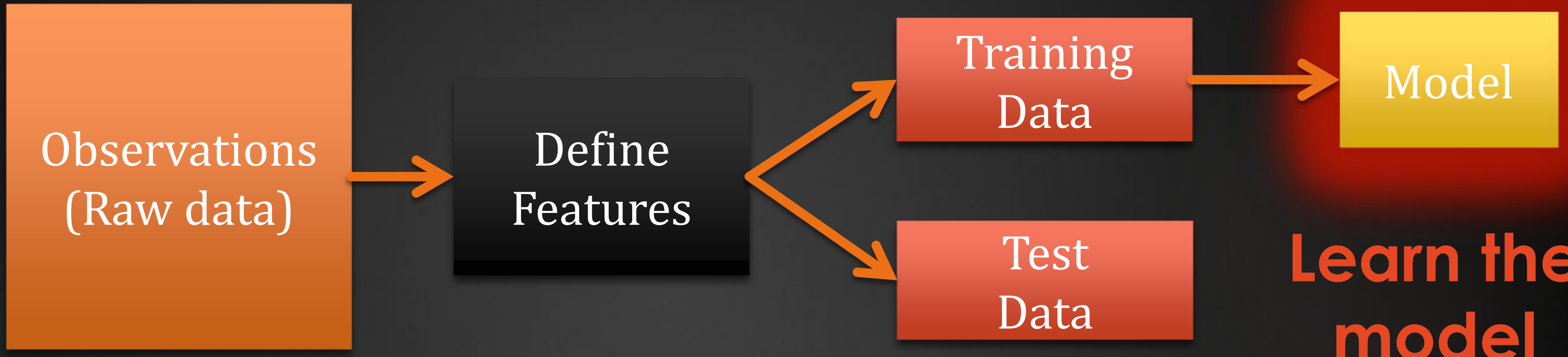
Preprocess your data

How to train your algorithm

Split into training and test sets

Helps us estimate how good the model is on new data

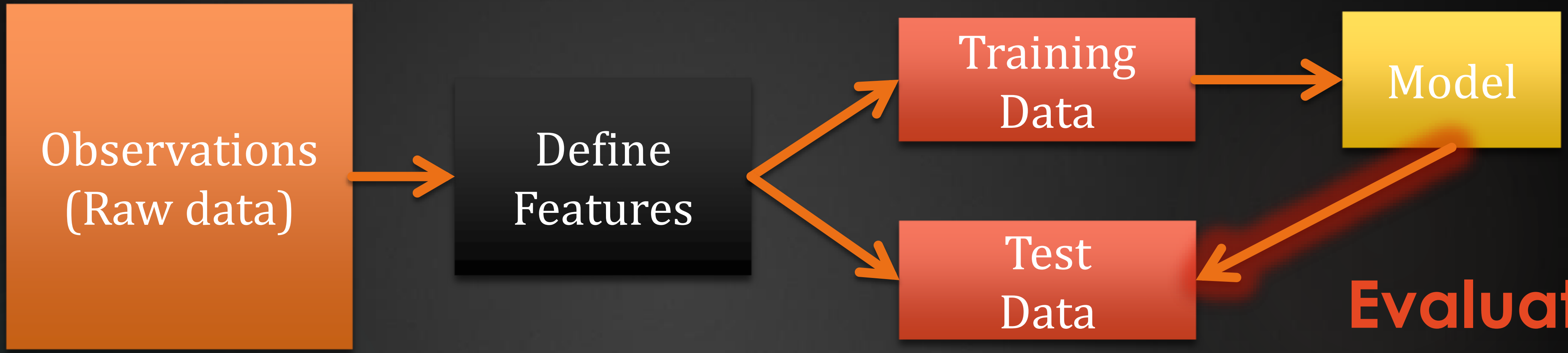
How to train your algorithm



Learn the model

Optimize: What model parameters are most likely, given the training data?

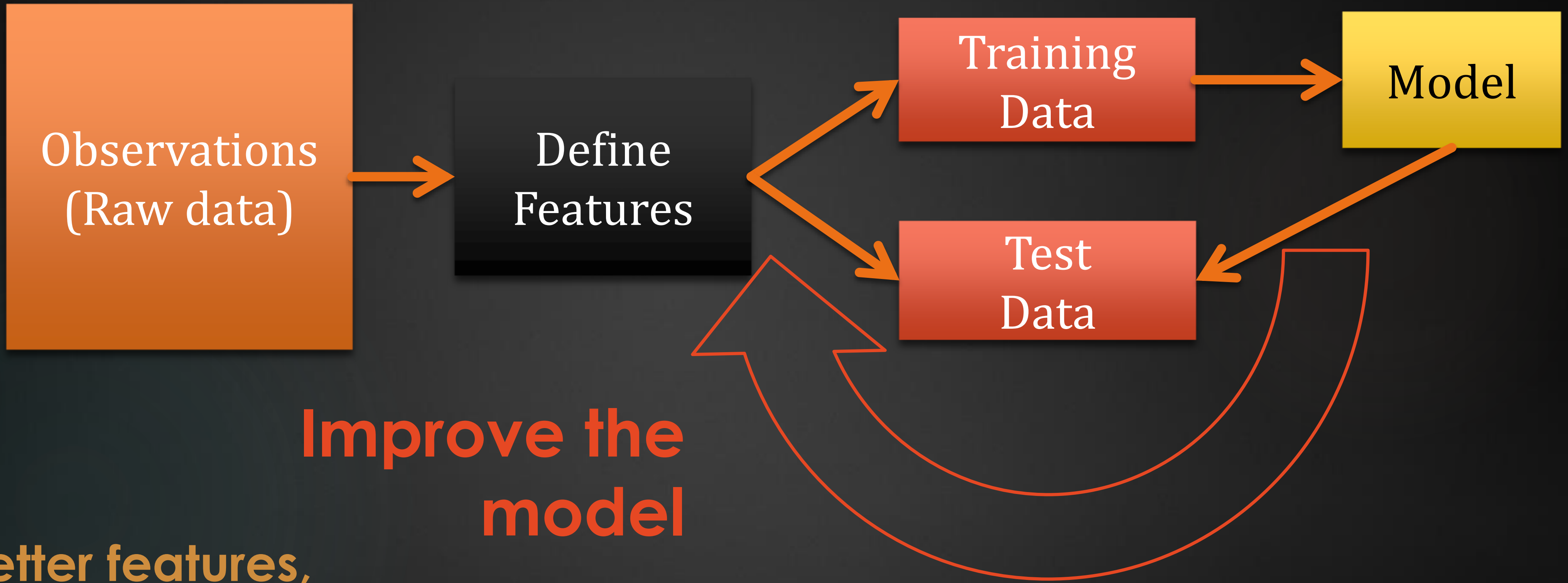
How to train your algorithm



Evaluate model's accuracy

How much of the test set does it correctly classify?

How to train your algorithm



better features,
different models