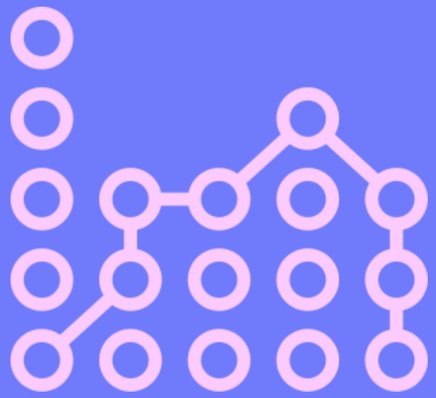
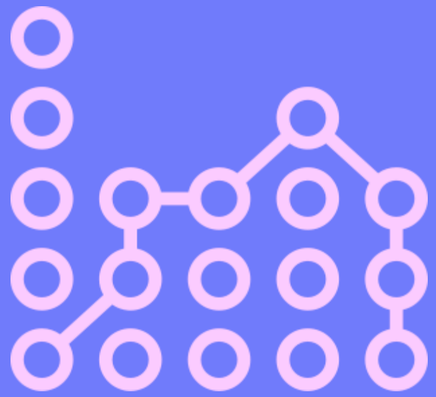
Section with Barbara

Week 4

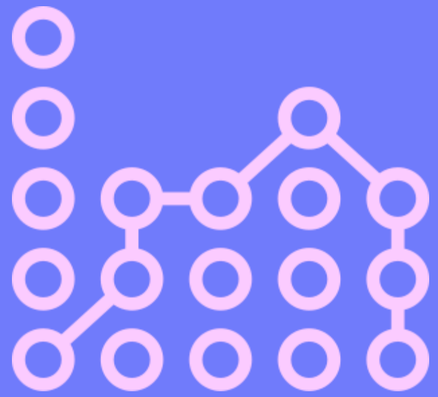
AI Stories



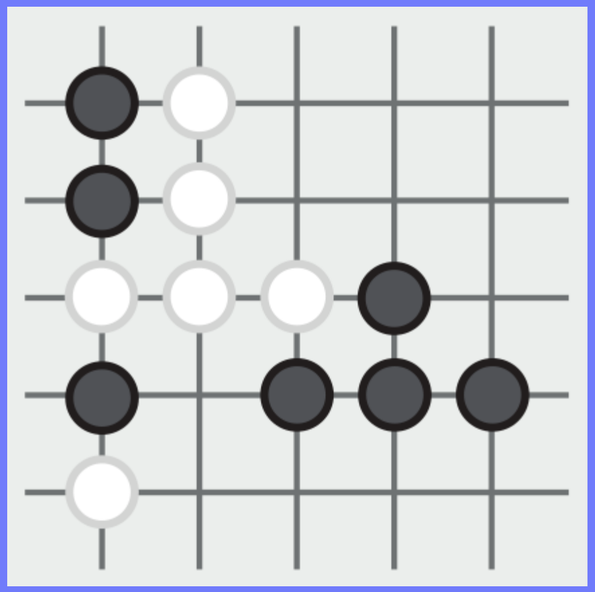
Projects



Learning



AI Stories: AlphaGo



SUPERVISED LEARNING

Make predictions from data

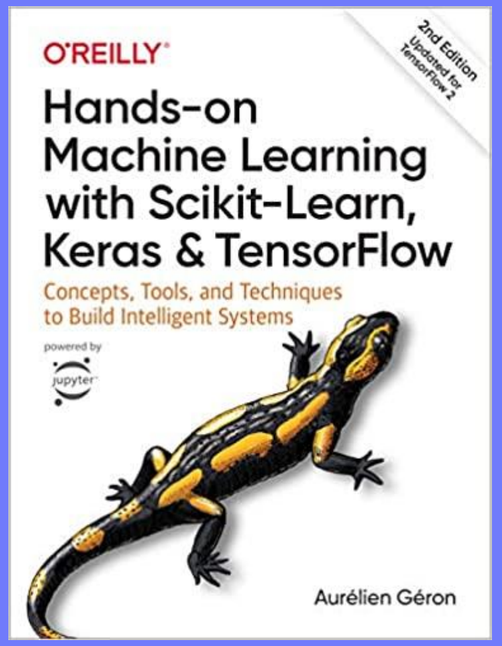
UNSUPERVISED LEARNING

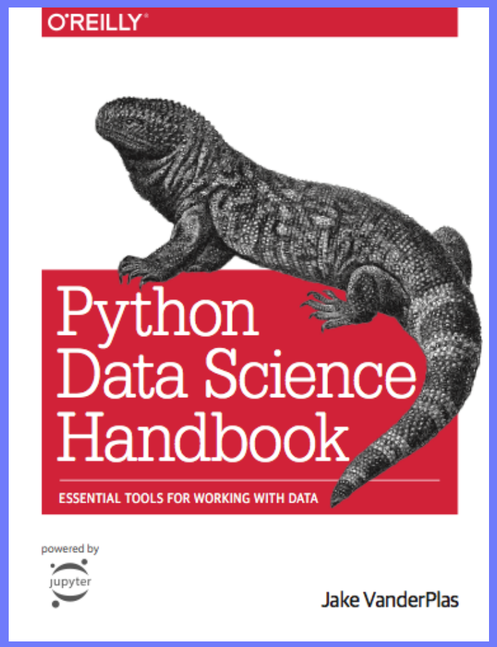
Find patterns in data

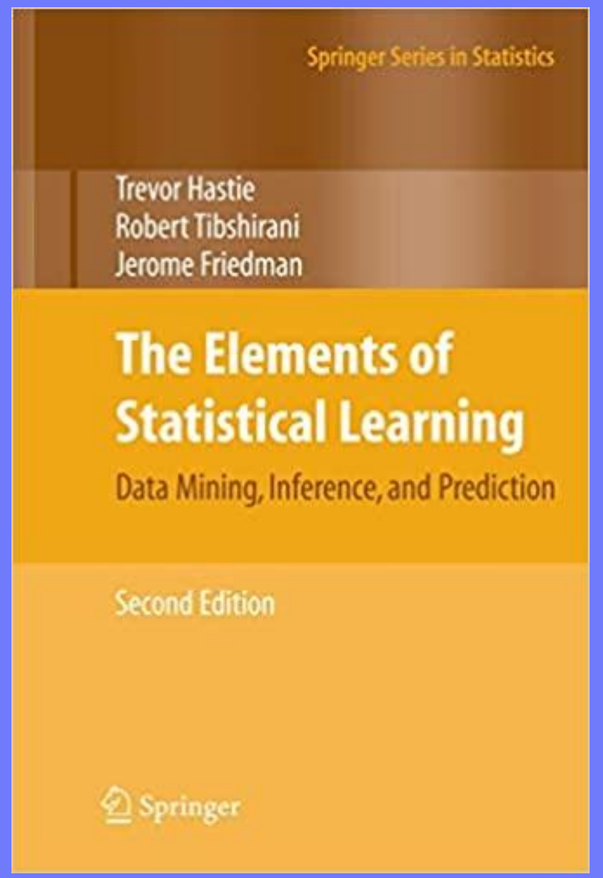
REINFORCEMENT LEARNING

Generates data from an environment


Reinforcement Learning
An Introduction
second edition
Richard S. Sutton and Andrew G. Barto


O'REILLY
2nd Edition
Updated for TensorFlow 2
Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow
Concepts, Tools, and Techniques to Build Intelligent Systems
powered by
jupyter
Aurélien Géron


O'REILLY
Python Data Science Handbook
ESSENTIAL TOOLS FOR WORKING WITH DATA
powered by
jupyter
Jake VanderPlas


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

https://towardsdatascience.com/

PyCon Talks

Andrew Ng’s Machine Learning Course

Machine Learning

PROCESS

Data Collection, Feature Engineering, Server Infrastructure

PAYOFF

Configuration, Evaluation, Verification, Debugging

PREDICTION

Automation, Deployment, Management, Monitoring

Python Tools

PROCESS

SQL, JSON, Pandas, Numpy, APIs, etc

PAYOFF

Sci-Kit Learn, Keras, Tensorflow, Pytorch, etc

PREDICTION

Kubernetes, Spark, Flask, Dask, etc

DATA MIRRORS HUMAN BIAS

https://www.nytimes.com/2019/12/06/business/algorithm-bias-fix.html

DATA AND HUMAN PRIVACY

https://www.nytimes.com/series/new-york-times-privacy-project

REINFORCEMENT LEARNING

ENVIRONMENT

WHAT IS REQUIRED?

- MARKOV CHAIN
- STATES

AGENT

WHAT IS INVOLVED?

- HISTORY
- BELIEFS
- LEARNING MODEL

VALUE FUNCTION

HOW IS IT IMPLEMENTED?

- REWARD
- GOOD STATE
- BAD STATE

POLICY

HOW IS IT REPRESENTED?

- ACTION
- BEHAVIOR
- MAP

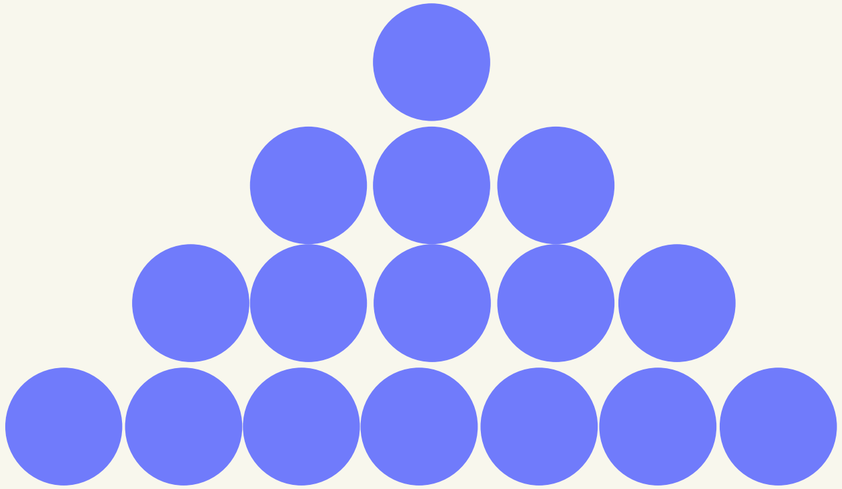
RL Use Cases

GAMING

AUTONOMOUS DRIVING

FINANCIAL TRADING

projects



shopping

```
def load_data(filename):

Follow the instructions in the docstring (the docstring
was too long to include on this slide)
```

shopping

```
def train_model(evidence, labels):
    """
    Given a list of evidence lists and a list of
    labels, return a fitted k-nearest neighbor model
    (k=1) trained on the data.
    """
```

Check out the lecture if you are stuck!

shopping

```
def evaluate(labels, predictions):
    """
    Given a list of actual labels and a list of
    predicted labels, return a tuple (sensitivity,
    specificity). Assume each label is either a 1
    (positive) or 0 (negative). `sensitivity` should be
    a floating-point value from 0 to 1 representing the
    "true positive rate": the proportion of actual
    positive labels that were accurately identified.
    `specificity` should be a floating-point value from
    0 to 1 representing the "true negative rate": the
    proportion of actual negative labels that were
    accurately identified.
    """
```

See next slide for hints!

confusion matrix

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

sensitivity

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

specificity

	Actual Positive	**Actual Negative**
Predicted Positive	True Positive	False Positive
Predicted Negative	False Negative	True Negative

sensitivity = (predicted and labeled positive)
/(labeled positive)

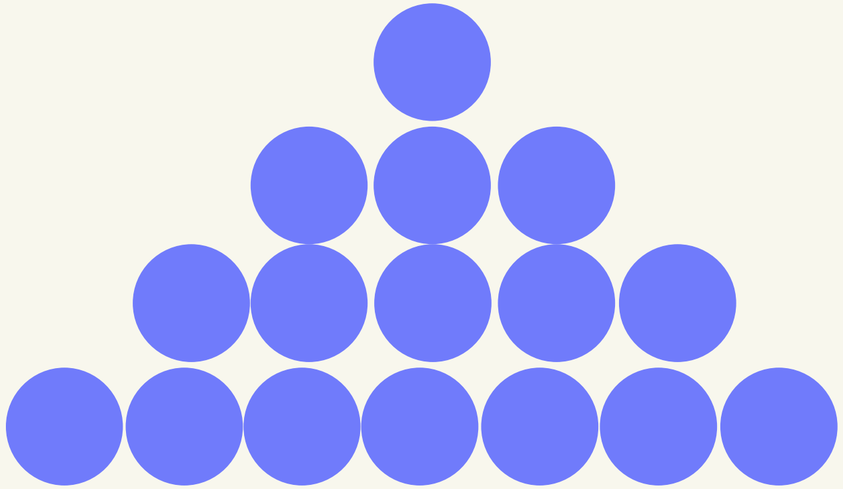
specificity = (predicted and labeled negative)
/(labeled negative)

sensitivity and **specificity** are two of many metrics to evaluate your machine learning model. always choose a metric that makes sense for your model and your business or research objective.

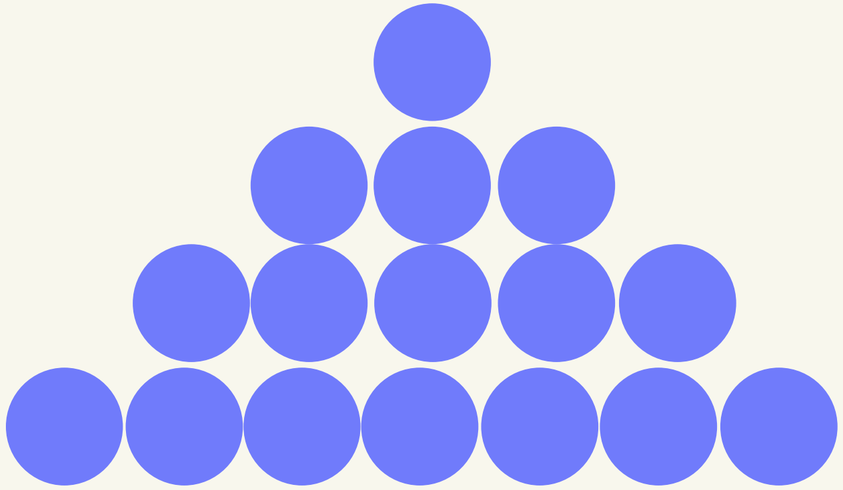
nim

```
def get_q_value(self, state, action):
      """
      Return the Q-value for the state `state` and the
      action `action`. If no Q-value exists yet in
      `self.q`, return 0.
      """
```

Be mindful of the data type `state` and `action` should be



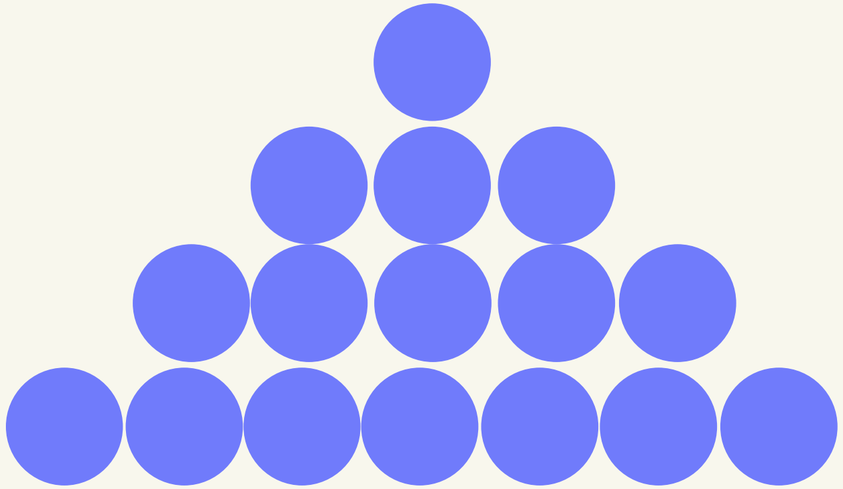
nim



```
def update_q_value(self, state, action, old_q, reward,
future_rewards):
    """
    Update the Q-value for the state `state` and the
    action `action` given the previous Q-value `old_q`,
    a current reward `reward`, and an estiamte of
    future rewards `future_rewards`. Use the formula:
    Q(s, a) <- old value estimate + alpha * (new value
    estimate - old value estimate) where `old value
    estimate` is the previous Q-value, `alpha` is the
    learning rate, and `new value estimate` is the sum
    of the current reward and estimated future rewards.
    """
```

All that is expected is for you to implement the formula!

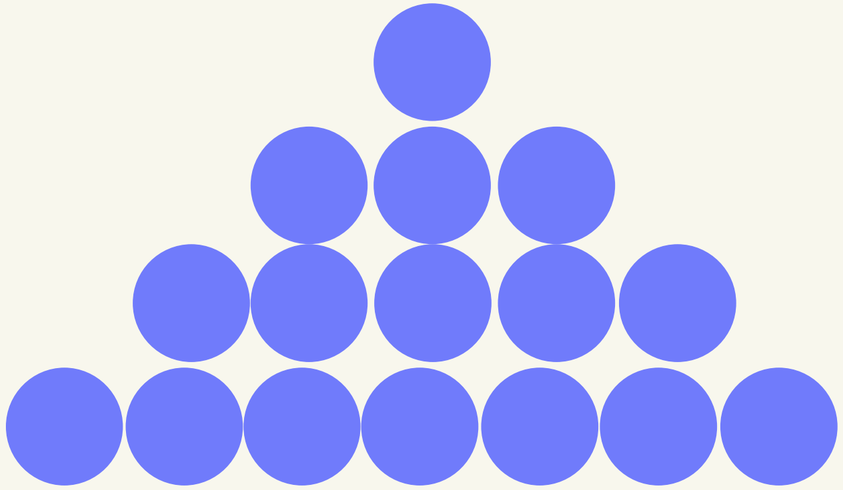
nim



```
def best_future_reward(self, state):
    """
    Given a state `state`, consider all possible
    `(state, action)` pairs available in that state and
    return the maximum of all their Q-values. Use 0 as
    the Q-value if a `(state, action)` pair has no Q-
    value in `self.q`. If there are no available
    actions in `state`, return 0.
    """
```

Hint: Q-values can be negative

nim



```
def choose_action(self, state, epsilon=True):
    """
    Given a state `state`, return an action `(i, j)` to
    take. If `epsilon` is `False`, then return the best
    action available in the state (the one with the
    highest Q-value, using 0 for pairs that have no Q-
    values). If `epsilon` is `True`, then with
    probability `self.epsilon` choose a random
    available action, otherwise choose the best action
    available. If multiple actions have the same Q-
    value, any of those options is an acceptable return
    value.
    """
```

The python module `random` will be helpful here! How do you determine if something is within a probability?