4. Detection

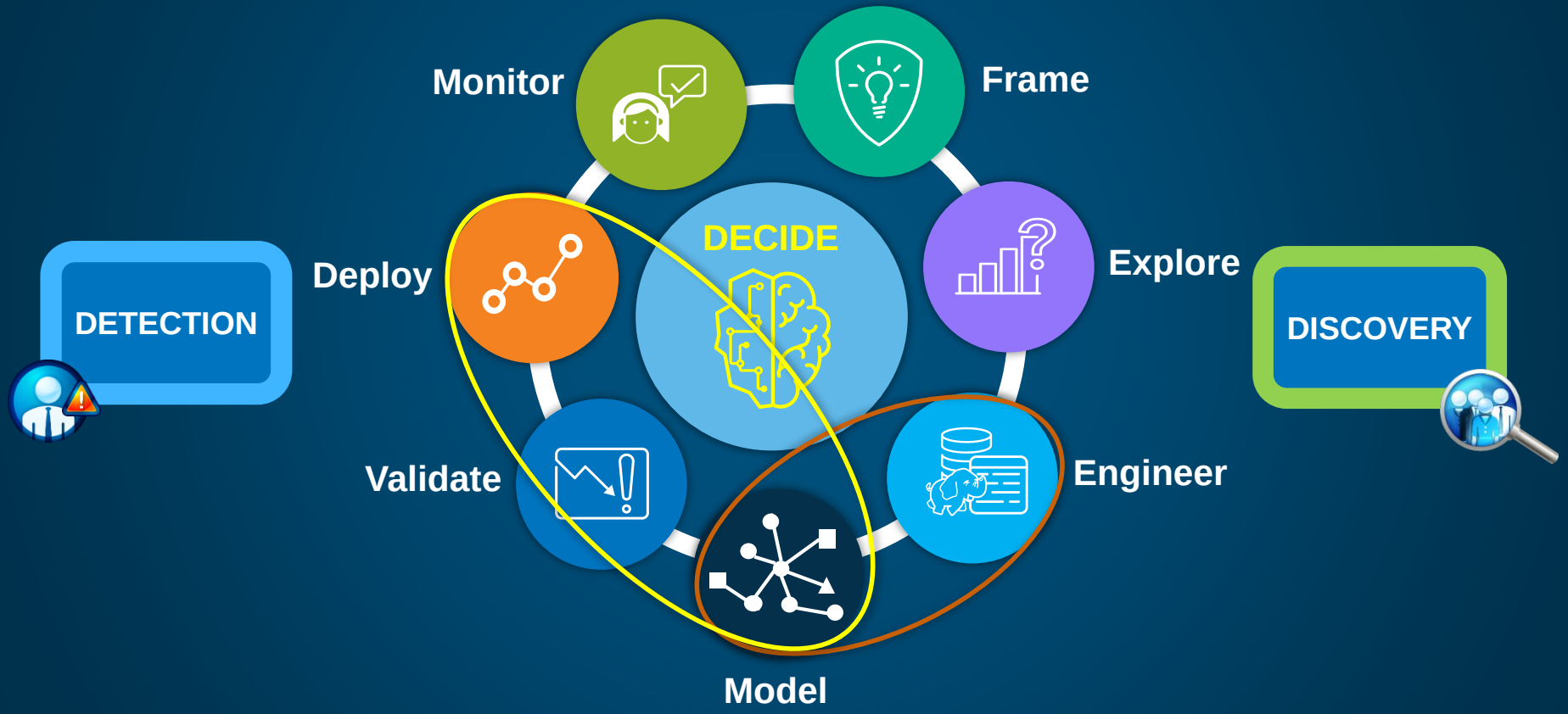
Refined and Predictive Alerts via Supervised Machine Learning

Cybersecurity Data Science (CSDS)

TOPIC
1. FRAME
2. DATA
3. DISCOVER
4. DETECT
5. DEPLOY

Learning Objectives

Cybersecurity Data Science (CSDS) Lifecycle



Objectives for Cybersecurity Detection and Prediction

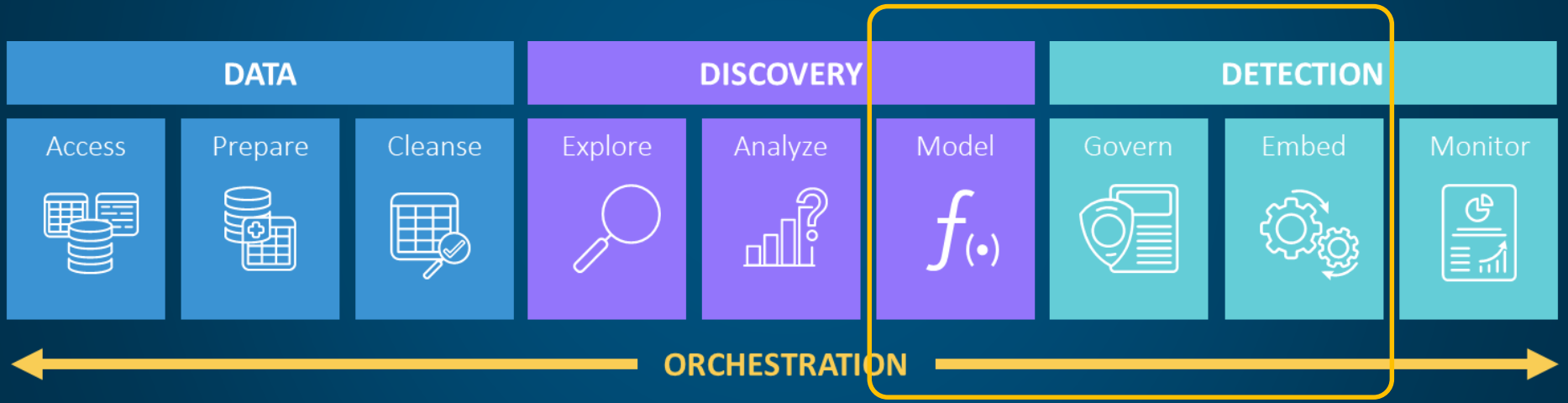
Predictive Detection Using Supervised Machine Learning

Detection and prediction: targeted alerts using supervised machine learning

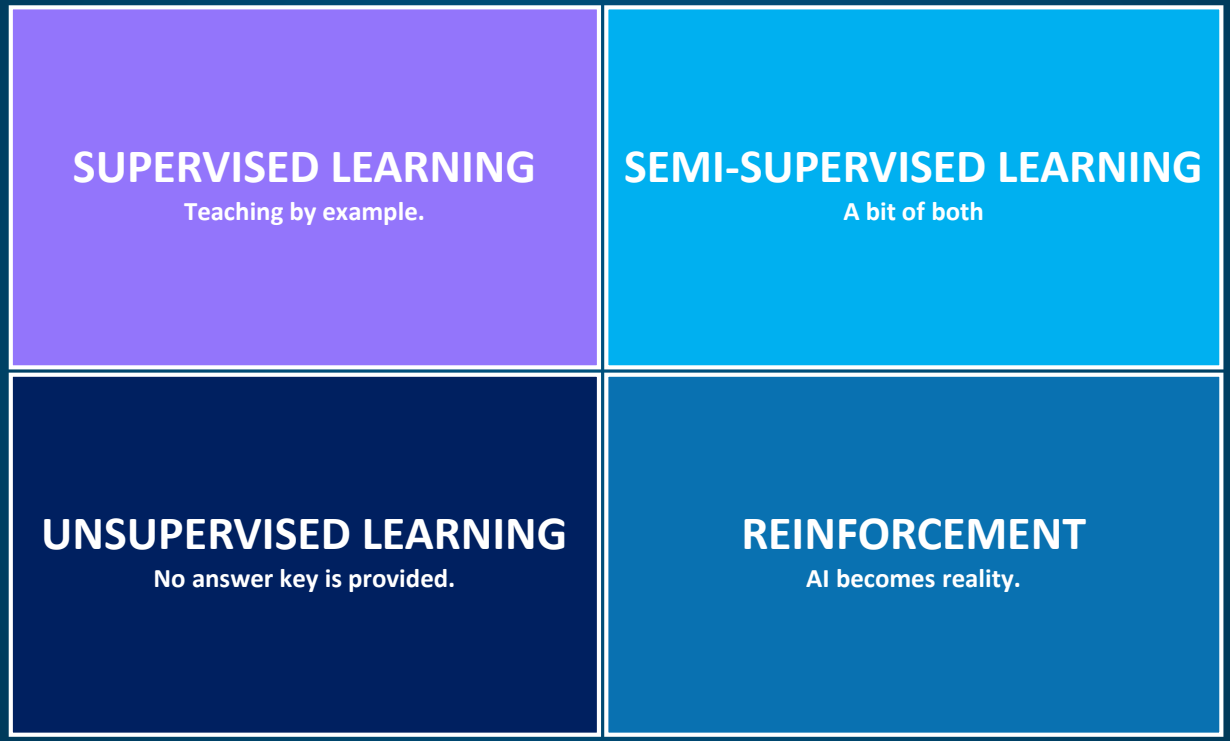
- Overview: the power of labeled data and identified incidents
- Apply supervised analytics machine learning algorithms to predict and to detect targeted anomalies
- *Demo / Exercise: Enterprise Miner for supervised analytics*

CSDS Process

Unified Orchestration



Role of Algorithms



Predictive (Supervised) Machine Learning

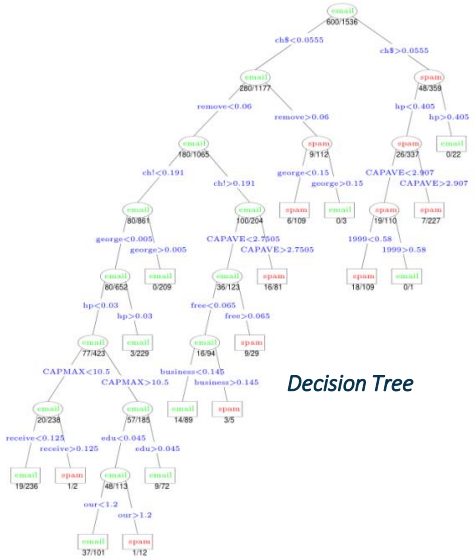
Machine Learning

Descriptive (Unsupervised)

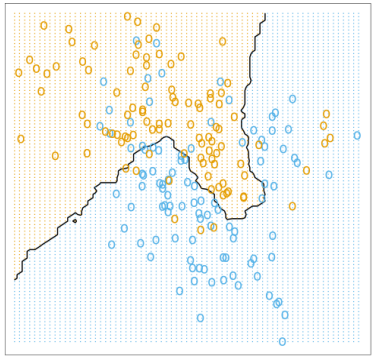
- Cluster analysis
- Factor analysis
- Self-Organizing Maps (SOMs)

Predictive (Supervised)

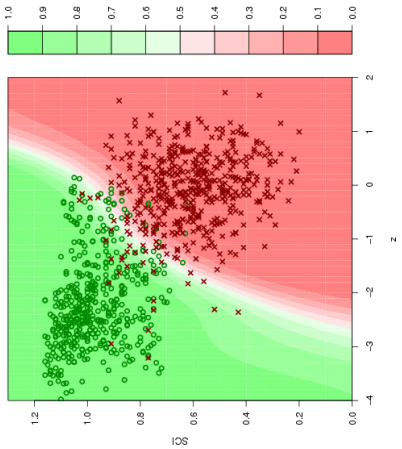
- K-Means
- Decision Trees (DT)
  (random forests, boosted trees)
- Naïve Bayes classifier
- Neural networks
- Support Vector Machine (SVM)
- Ensembles / Ensemble Learning



Decision Tree



k-nearest neighbors



Support Vector Machines

Machine learning tasks

Regression

Bayesian Statistics

Decision trees

Gradient boosting

Random forests

SVM

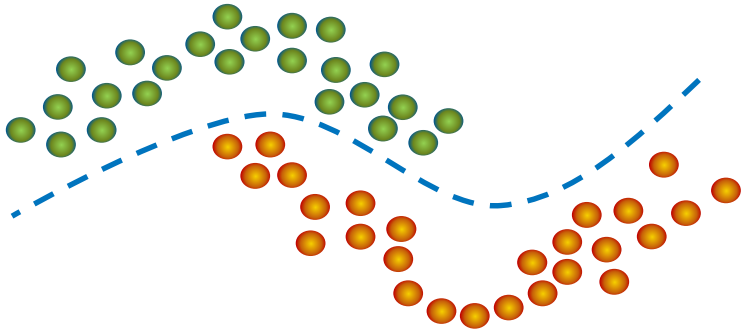
Gaussian processes

Supervised Learning

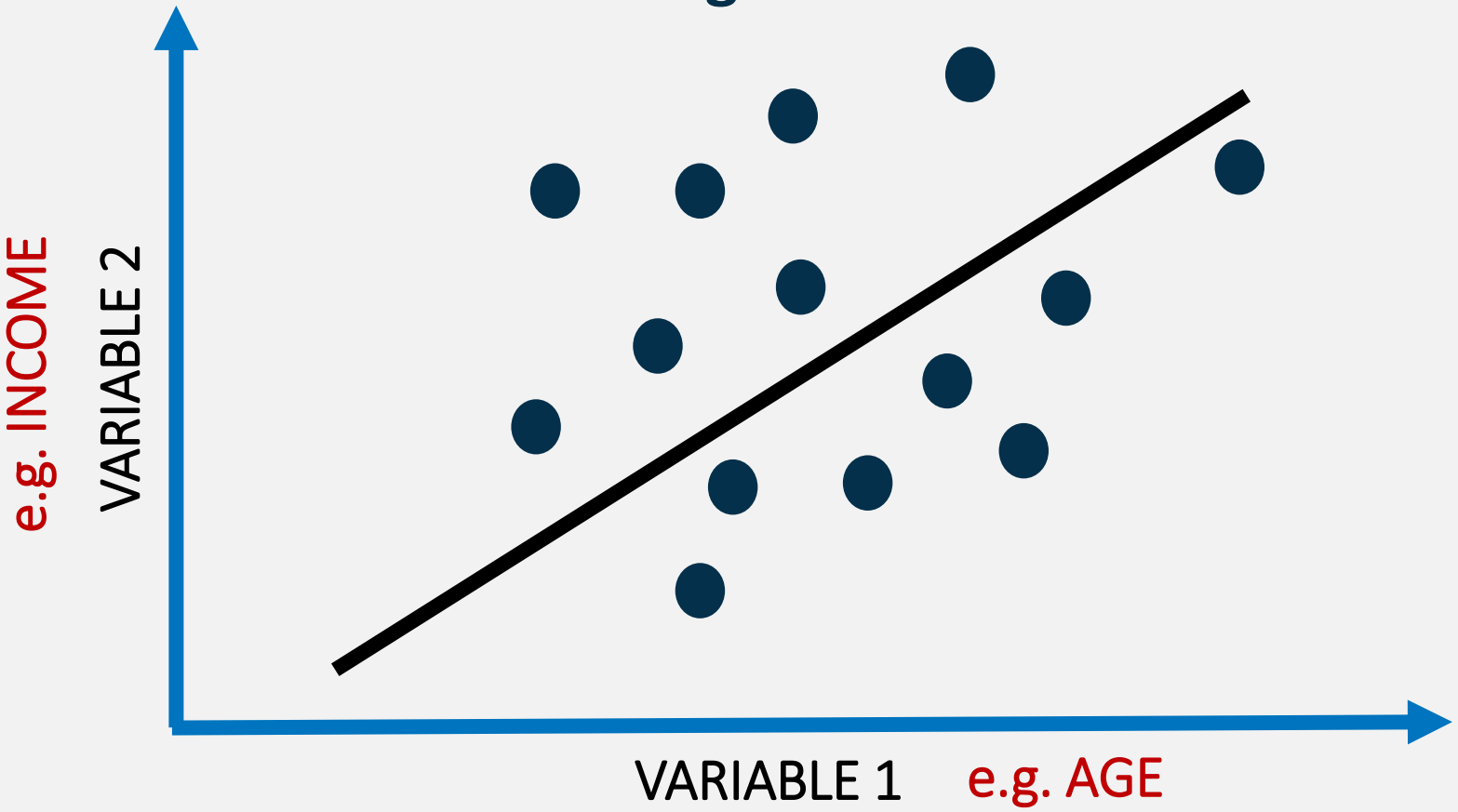
- Trained on labeled examples. We have a target we are predicting.
- Map inputs to desired output.
- Suitable for classification, regression, prediction.

Considerations

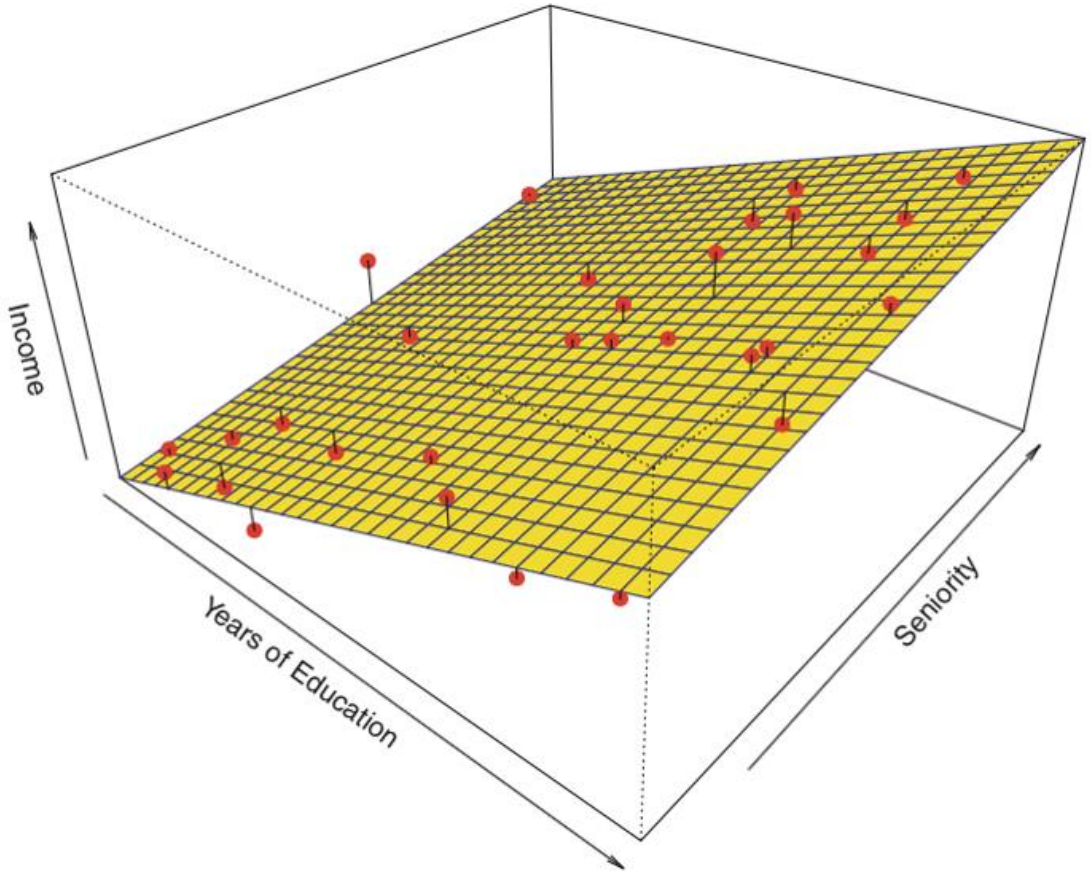
- Getting labeled data for rare events can be a challenge
- Suspicion is not fraud
- Data is skewed 99-1



Regression

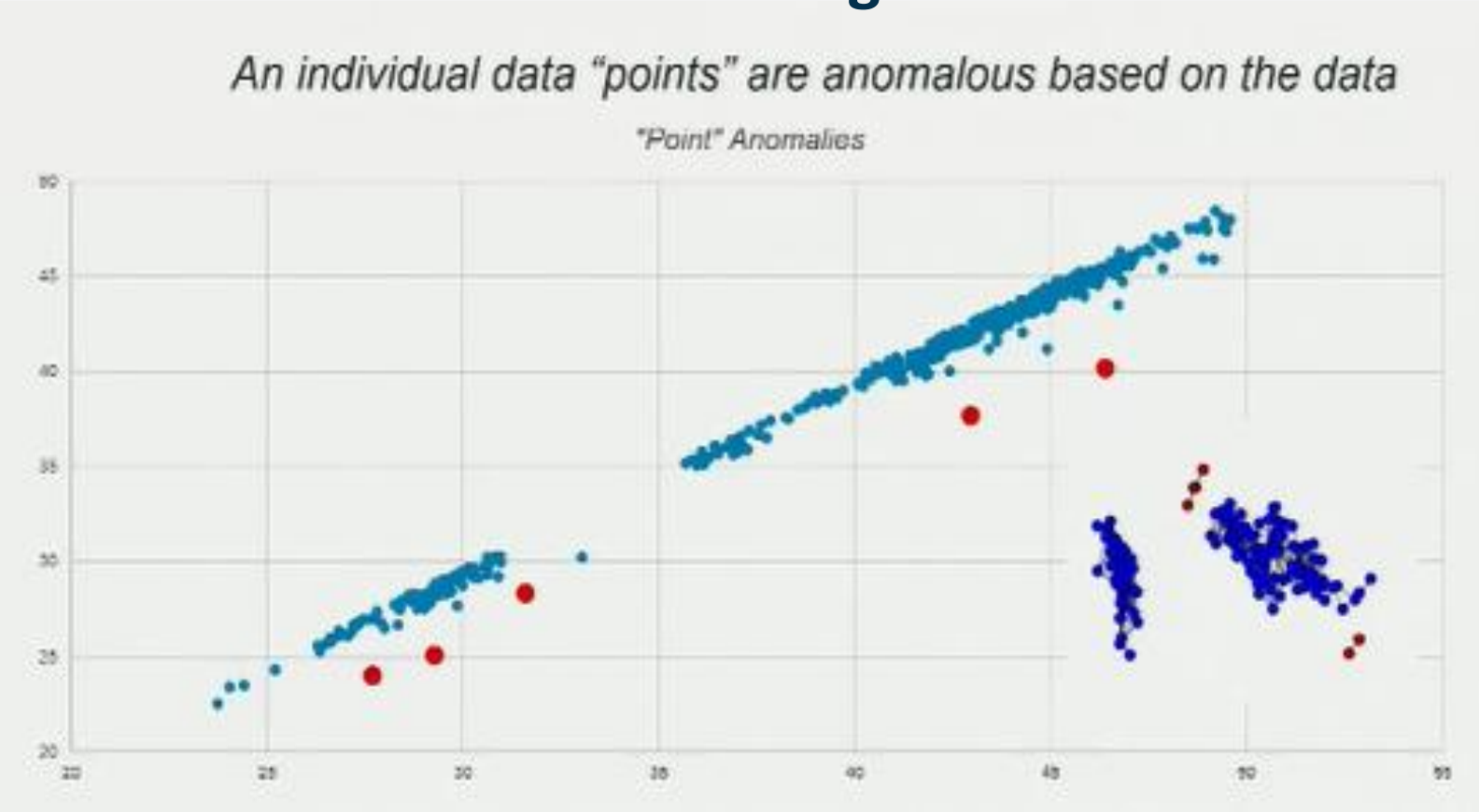


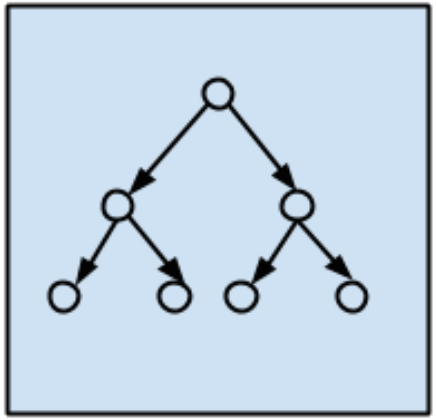
Multilinear Regression



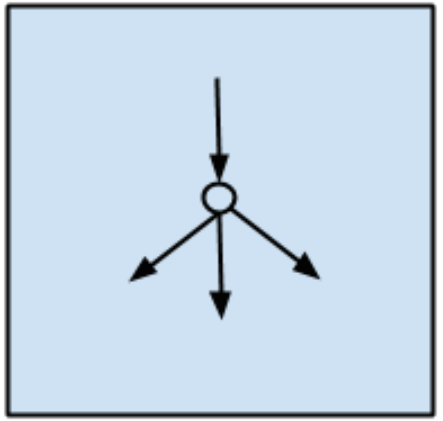
Econometric Forecasting

Econometric Diagnostics



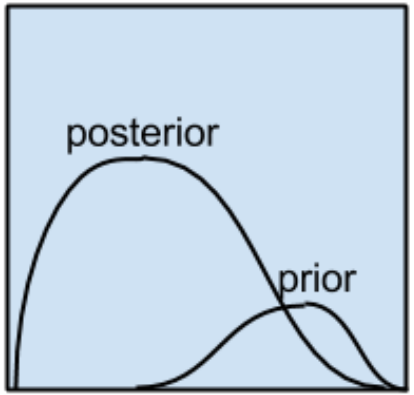


Decision Tree Algorithms

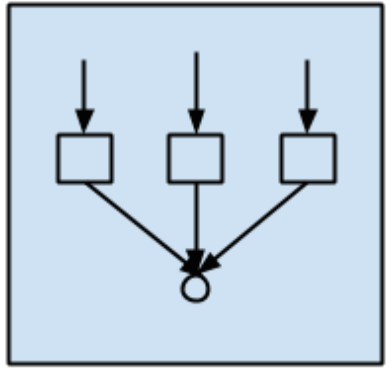


Artificial Neural Network Algorithms

Many Predictive Machine Learning Algorithms…



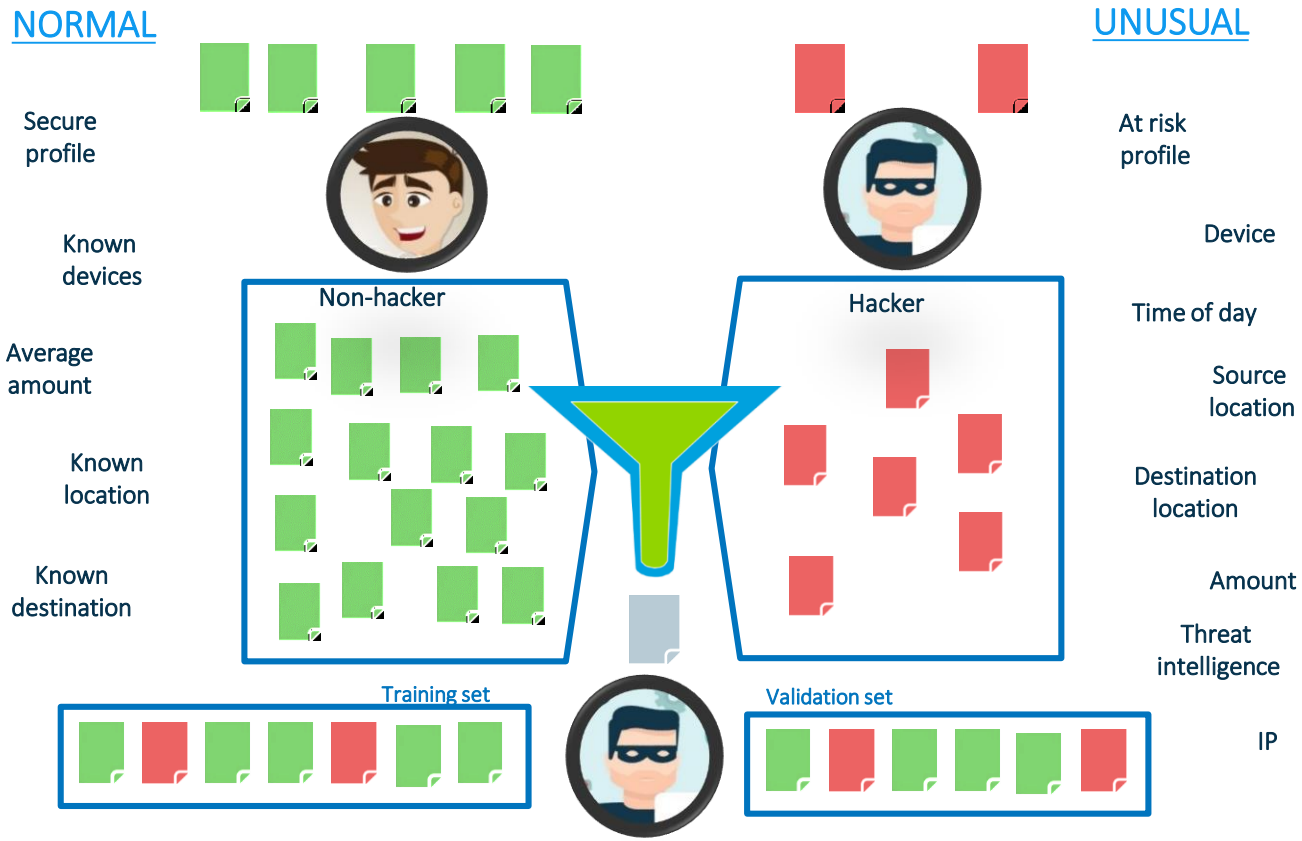
Bayesian Algorithms



Ensemble Algorithms

http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/

Predictive Machine Learning (Supervised)



Predictive ML

Why and How for Cyber?

Predictive Cybersecurity Use Cases

WHAT?

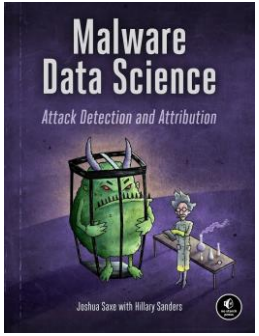
1. Malware detection
2. Device behavior
3. Known incidents

HOW?

1. Recording of behavior (e.g. Wireshark communications)
2. CMDB listing of devices + recording of behavior (e.g. authentication + NetFlow + DNS)
3. E.g. password attack in Linux authentication logs; DDOS attack

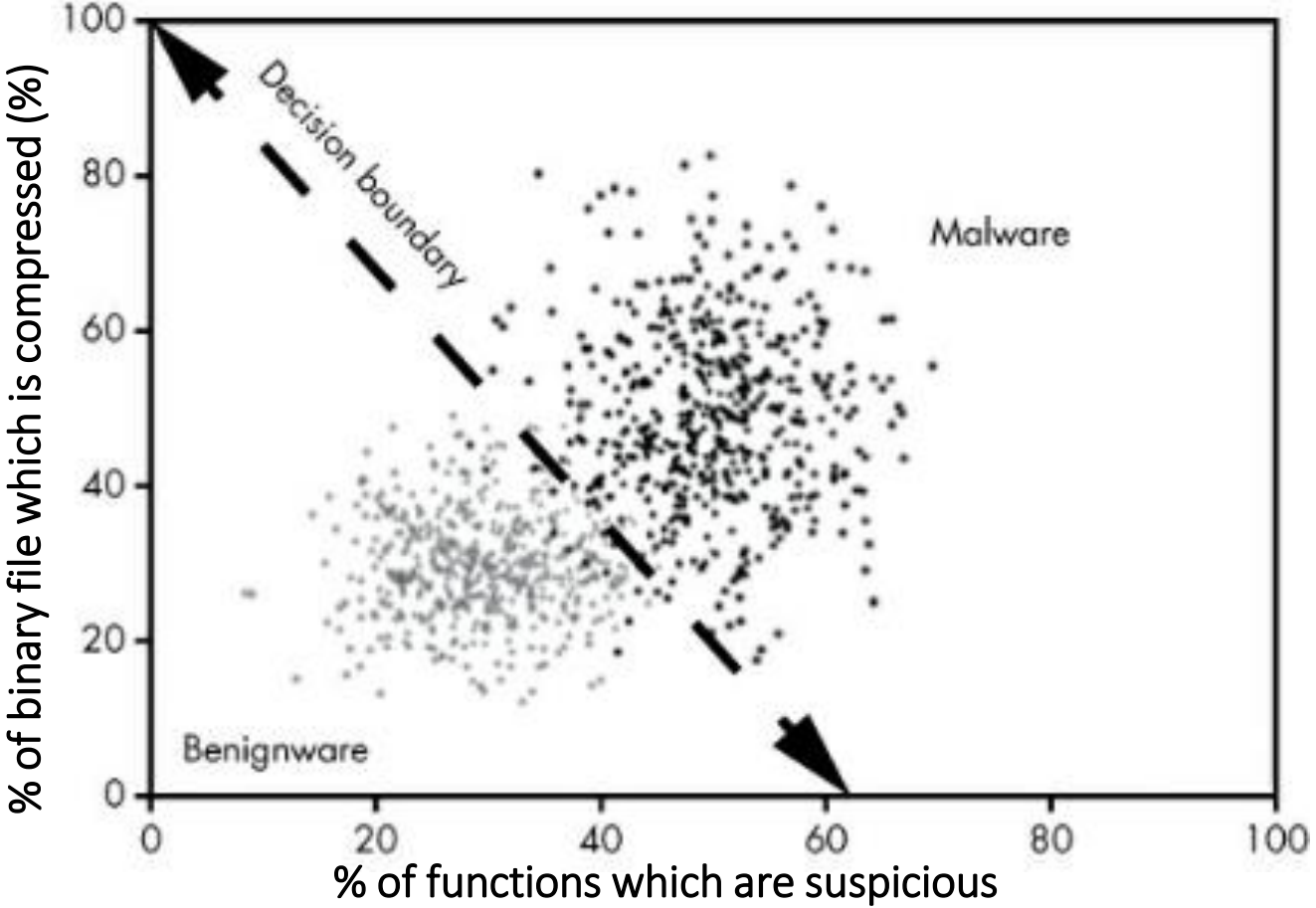
Predictive ML for Malware

1. **Collect** examples of malware and benignware. Use these examples to train ML algorithm to recognize malware.
2. **Extract** features from each training example to represent example as an array of numbers. This step also includes research to design good features that will help your machine learning system make accurate inferences.
3. **Train** the machine learning system to recognize malware using the features we have extracted.
4. **Test** the approach on some data not included in our training examples to see how well our detection system works.



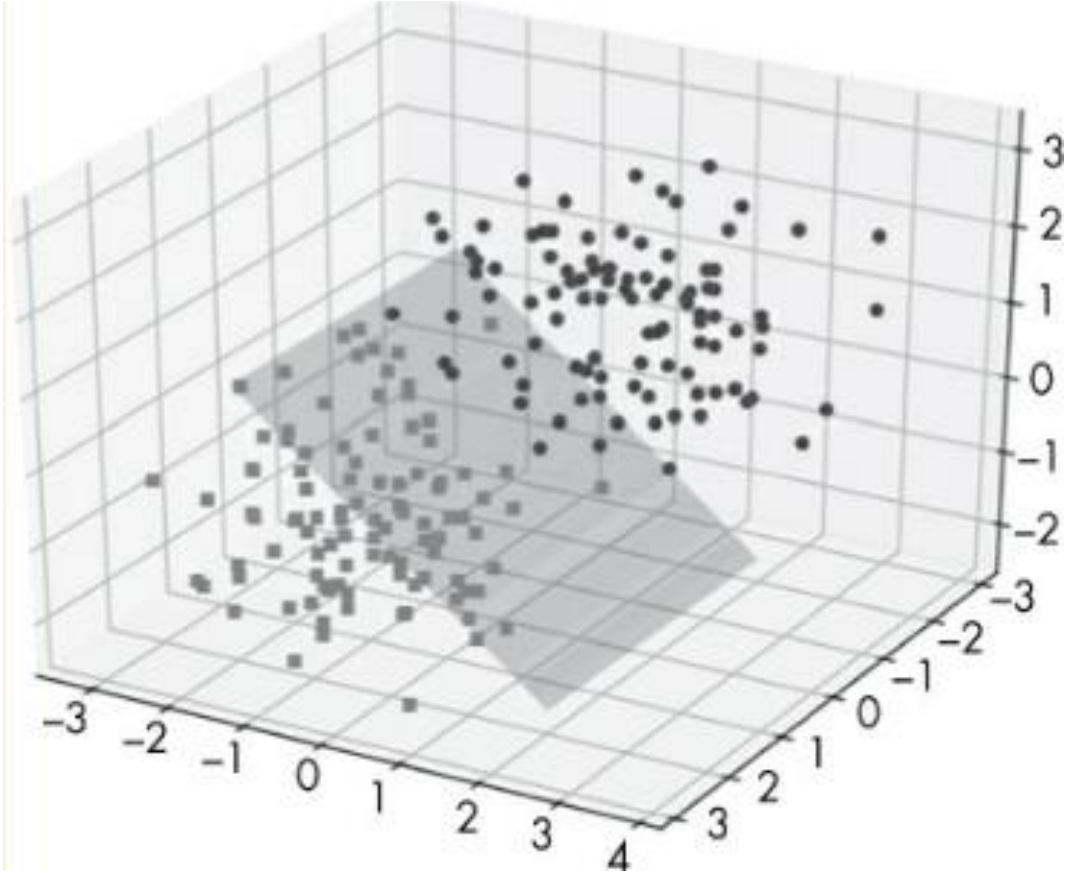
Saxe, Joshua
Malware Data Science

Simple Example: Training Based on a Decision Boundary (2 feature Logistic Regression)



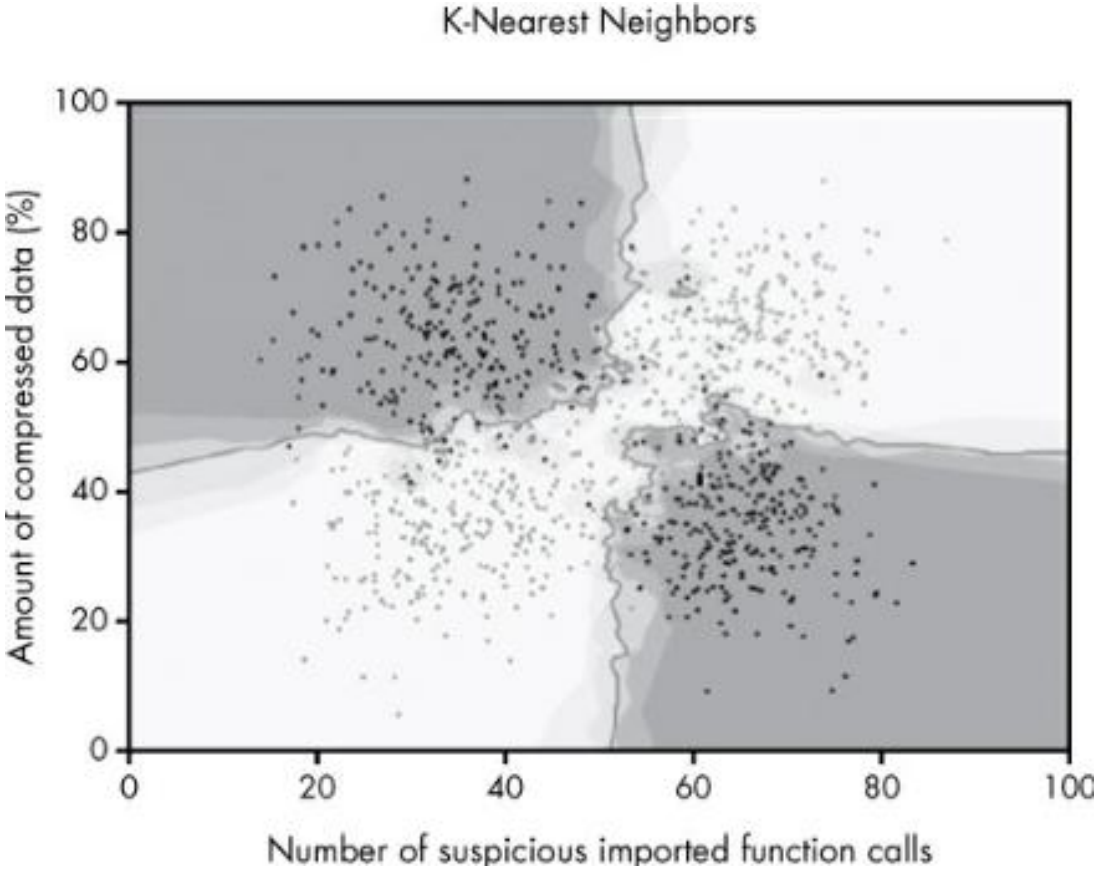
Reference: *Saxe, Joshua. Malware Data Science*

...and 3 dimensional Logistic Regression feature space



***Reference**: Saxe, Joshua. Malware Data Science*

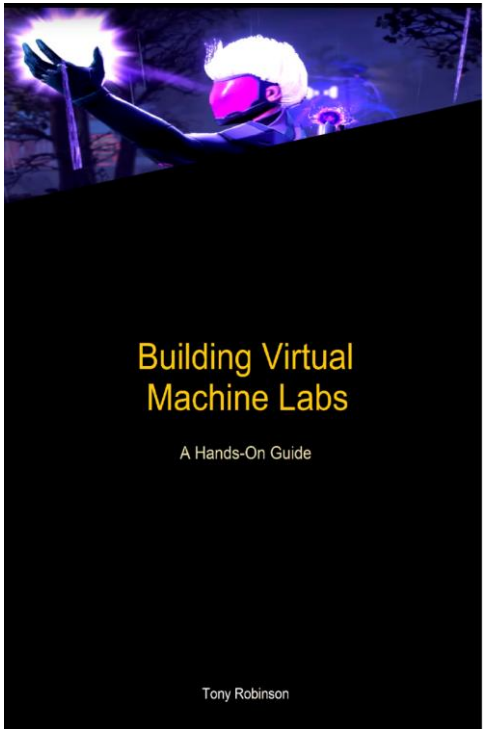
Irregular and Disjointed Decision Boundaries: e.g. K-Nearest Neighbor



***Reference**: Saxe, Joshua. Malware Data Science*

Where do I get malware for analysis?

- Malware Zoos (e.g. theZoo) https://github.com/ytisf/theZoo
- Build your own zoo https://www.sans.org/reading-room/whitepapers/malicious/paper/33543
- Metasploit
  - Metasploit Framework is a complex tool that can surveil targets with scans, select exploits to match vulnerabilities (~600), create payloads (~200), receive connections from backdoors, launch exploits

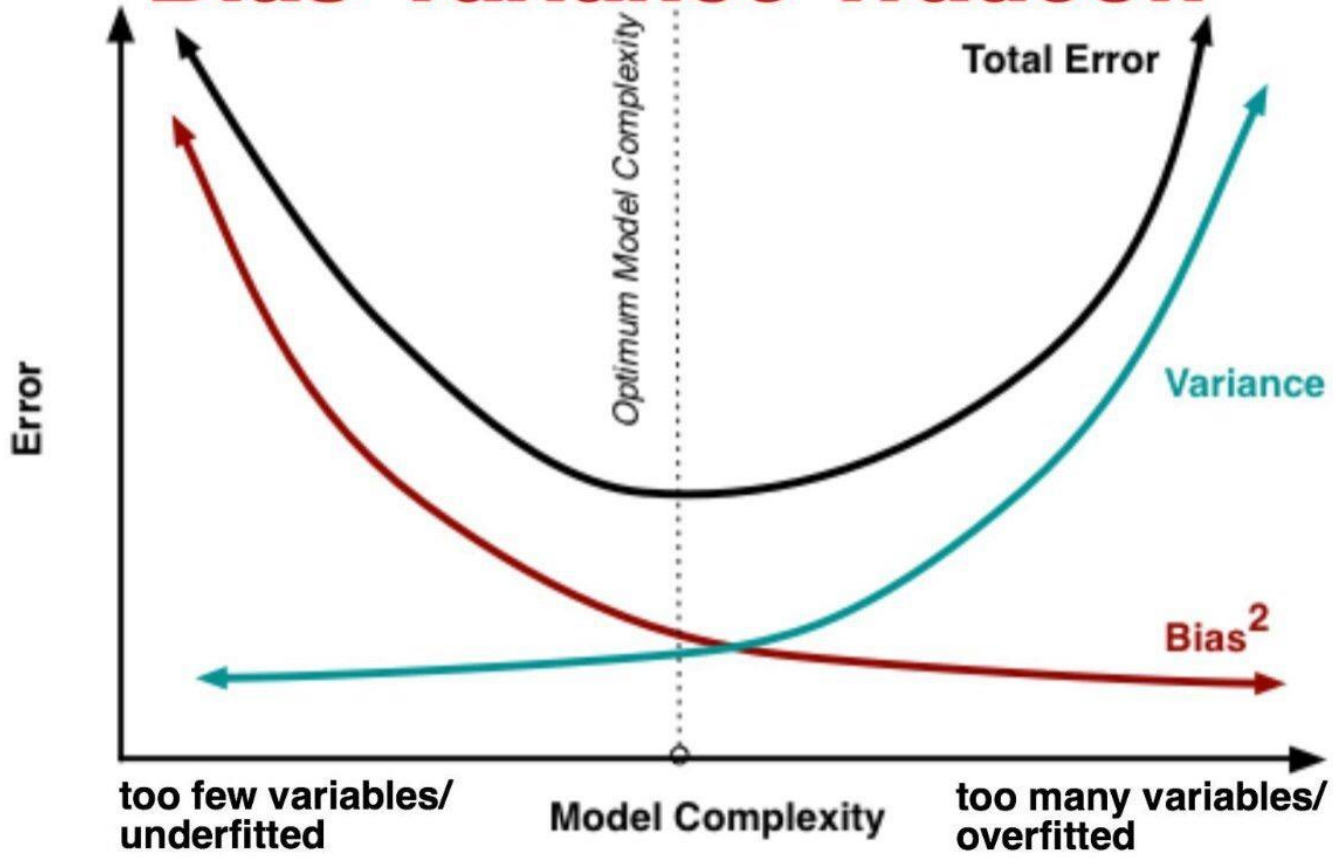


Where can I test and analyze?

- 'Sandbox' of test computers
- Honeypots (e.g. RasPis)
- VM lab

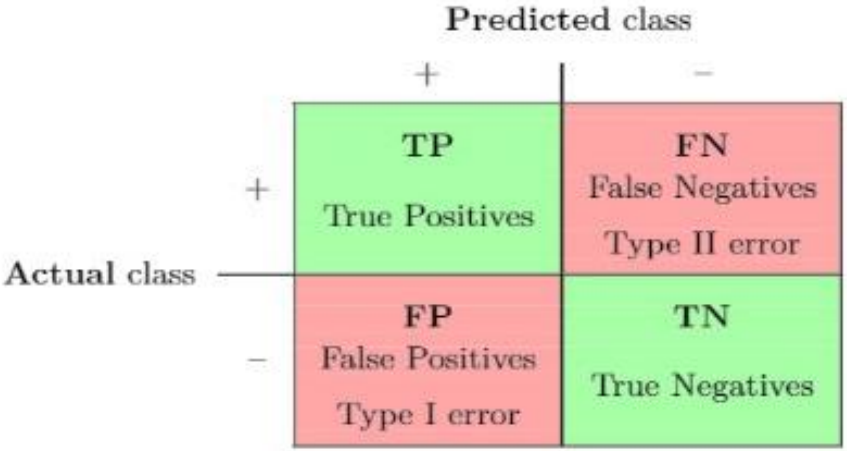
Maar let op! Openbaar Ministerie beschouwt Metasploit en Nmap als malware
https://www.security.nl/posting/567852/Openbaar+Ministerie+beschouwt+Metasploit+en+Nmap+als+malware

Bias-Variance Tradeoff



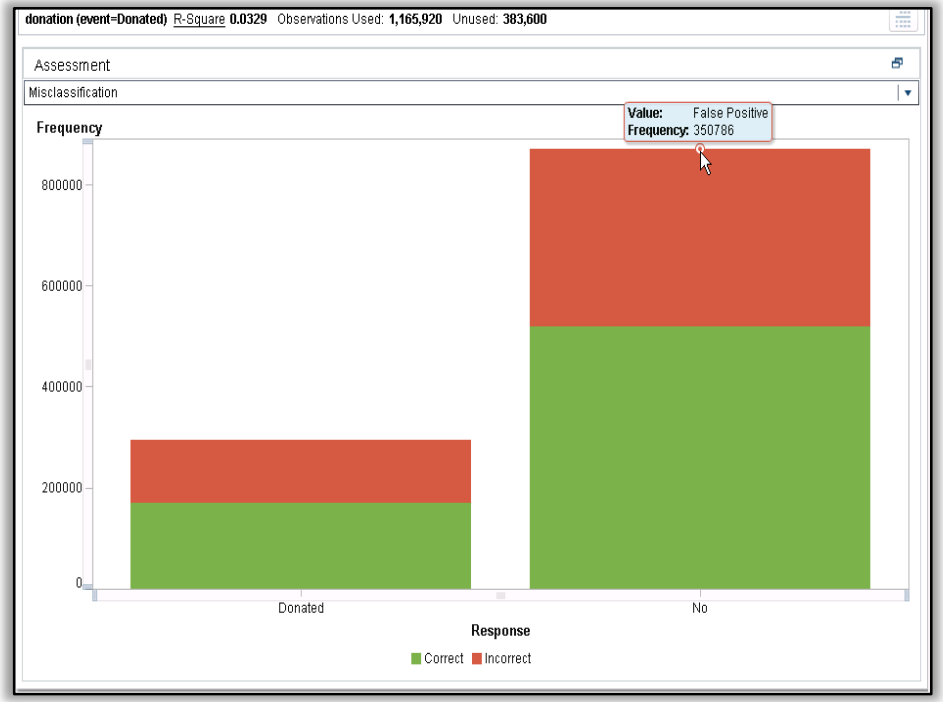
Model Diagnostics

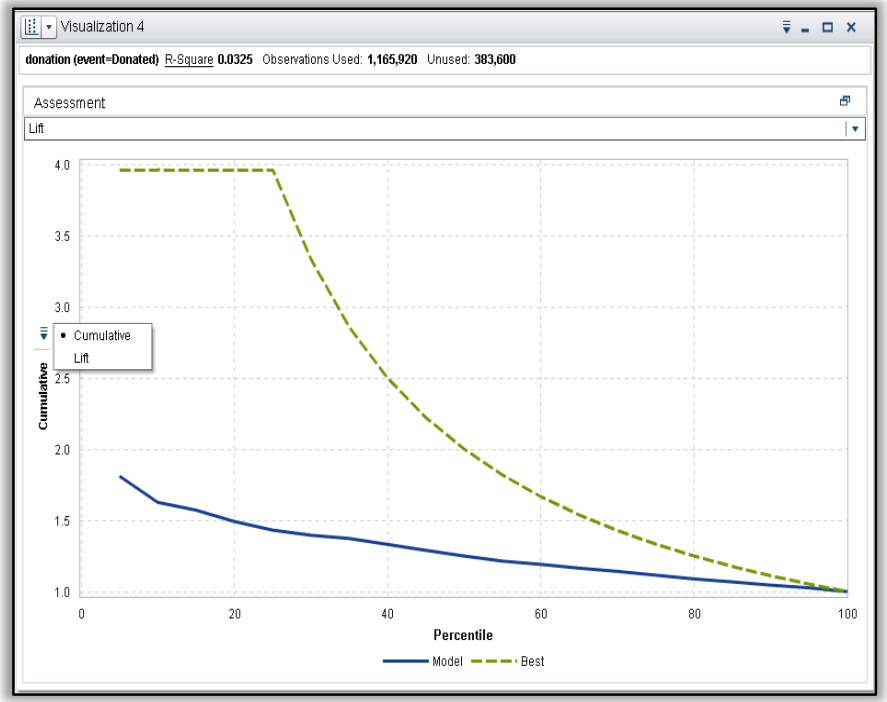
Confusion Matrix



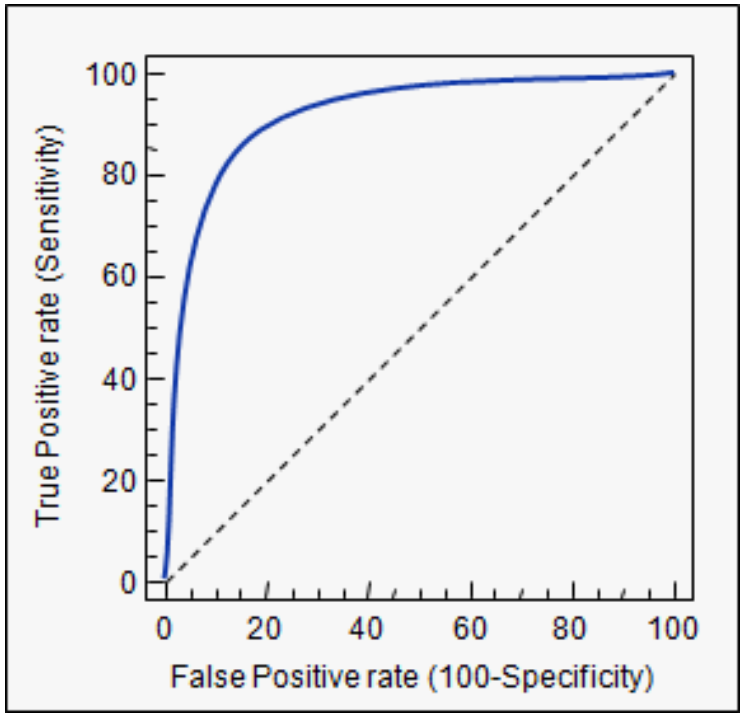
Metric	Formula	Interpretation
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Overall performance of model
Precision	$\frac{TP}{TP + FP}$	How accurate the positive predictions are
Recall Sensitivity	$\frac{TP}{TP + FN}$	Coverage of actual positive sample
Specificity	$\frac{TN}{TN + FP}$	Coverage of actual negative sample
F1 score	$\frac{2TP}{2TP + FP + FN}$	Hybrid metric useful for unbalanced classes

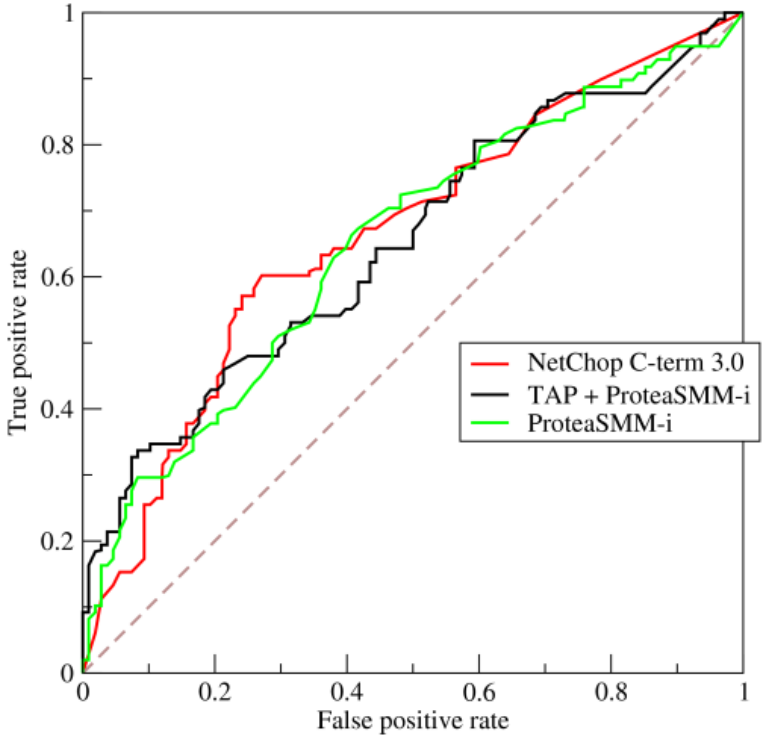
Model Diagnostics: Lift and Misclassification Rate





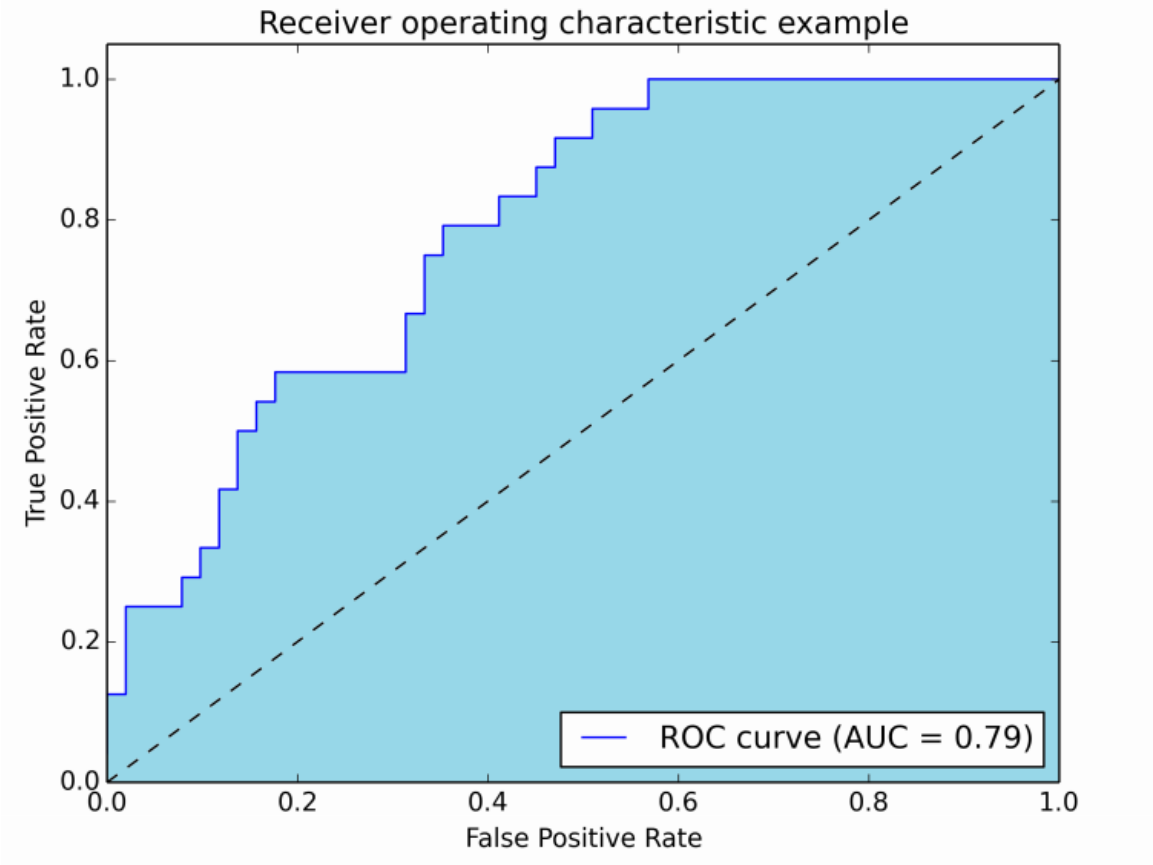
Receiver Operating Characteristic (ROC curves)





False-positive (FP) rate is also known as the fall-out or probability of false alarm and can be **calculated** as (1 − specificity, where specificity = TN / (TN + FP)).
The **ROC curve** is thus the sensitivity as a function of fall-out.

Area under the Curve (AUC)



ML Tools

Example Machine Learning Tools

Open source

- R
- Python
- Weka
- Spark
- …

Commercial

- SAS JMP
- SAS Enterprise Miner
- SAS Viya VDMML
- IBM SPSS
- Oracle Data Mining
- Rapid Miner

Predictive ML

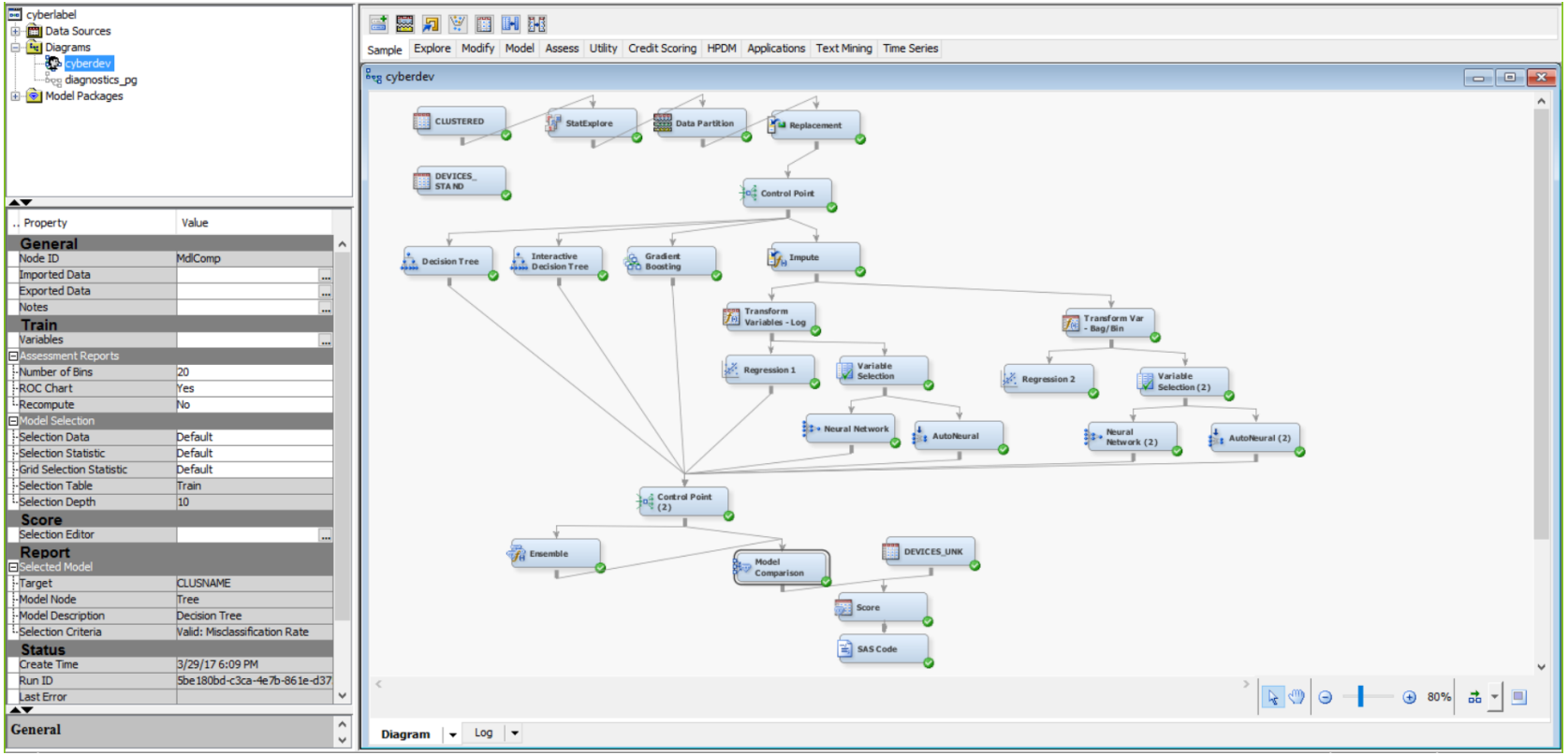
Demonstration and Hands-on Exercise

Predicting Device Types

This demonstration illustrates how to predict unknown device types given a sample of known devices

Predictive Device Labeling – Enterprise Miner

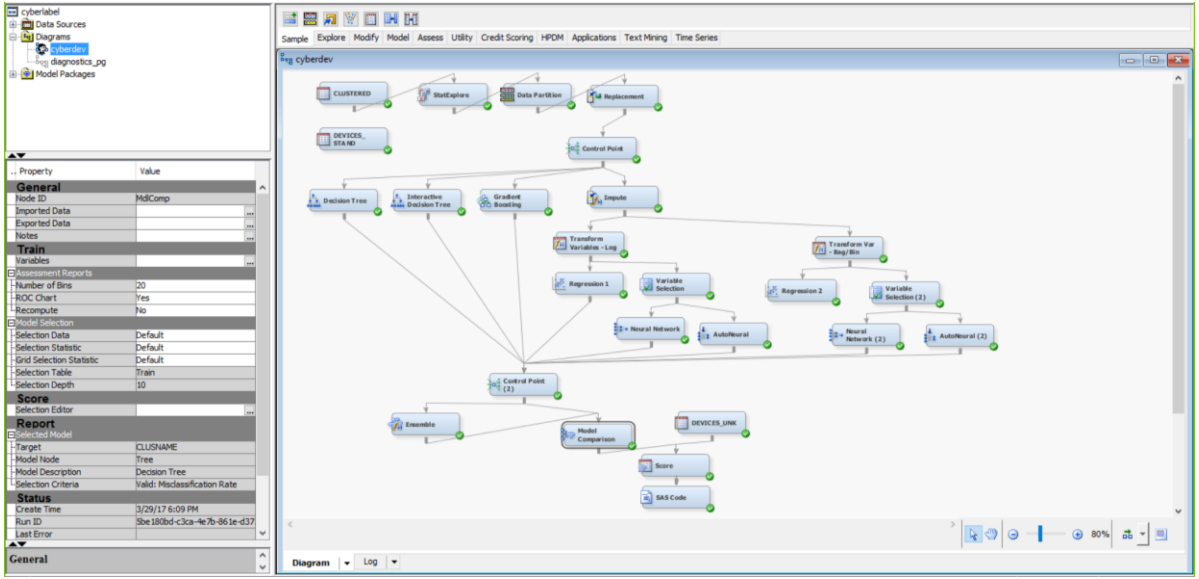
Using Labeled Devices to Predict Unlabeled Devices



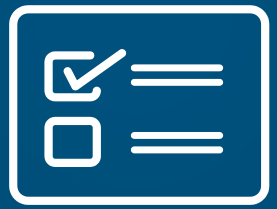
Device Types on the Network

ID	DEVICE TYPE	COUNT
1	Servers	7498
2	Networks (devices)	1810
3	Audio Video Device	856
4	Storage	181
5	Telecommunications	126
6	Application Server	77
7	Facilities	37
8	Critical Workstations	27
9	Printing	17
10	Blade Chassis	8
11	UPS	5
12	Applications	3
13	Clusters	3
14	Service	1
15	Application Code	1
16	Databases	1
17	WebSites	1
		10,652

Unknown devices: 34,615

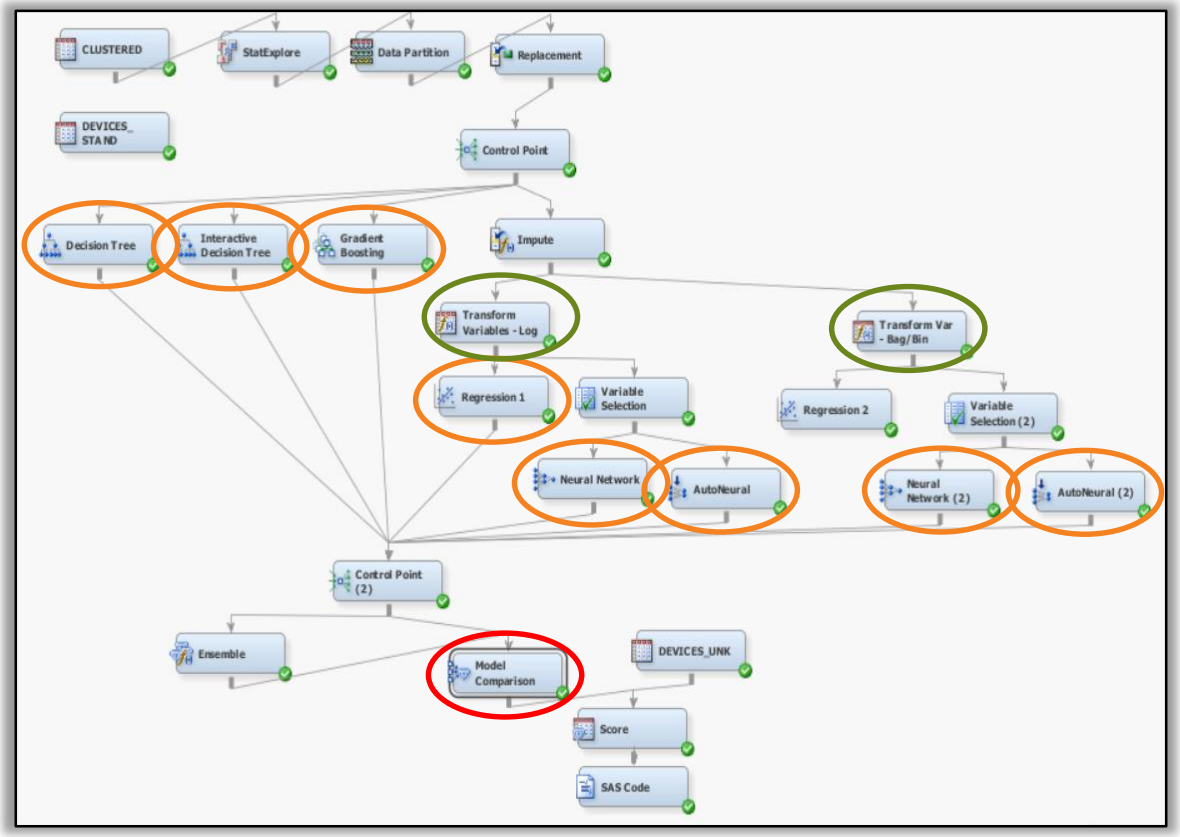


Exercise Review



MODEL MANAGEMENT CHAMPION / CHALLENGER

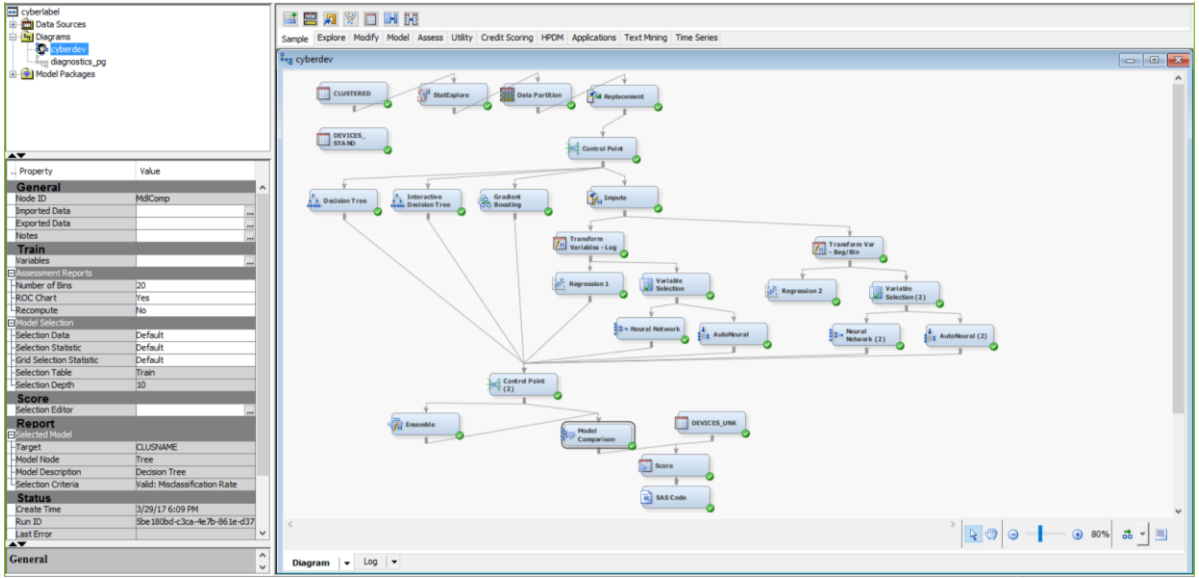
Need to test multiple methods (hint: deep learning is not ALWAYS the best!)



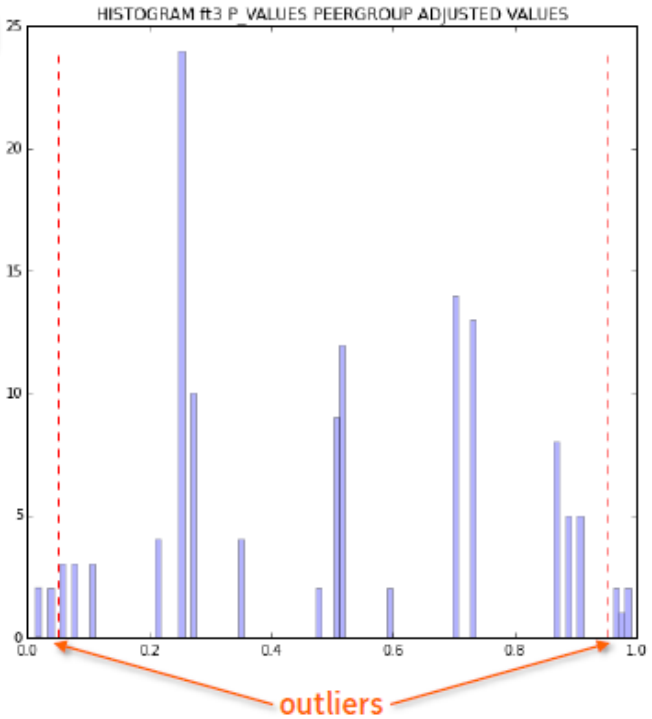
Why were most of predictions 2?

ID	DEVICE TYPE	COUNT
1	Servers	7498
2	Networks (devices)	1810
3	Audio Video Device	856
4	Storage	181
5	Telecommunications	126
6	Application Server	77
7	Facilities	37
8	Critical Workstations	27
9	Printing	17
10	Blade Chassis	8
11	UPS	5
12	Applications	3
13	Clusters	3
14	Service	1
15	Application Code	1
16	Databases	1
17	WebSites	1
		10,652

Unknown devices: 34,615



Do not assume given labels are statistically significant!



Deterministic (organizational groupings)

Nominal Logistic Fit for peerGroupCode

Whole Model Test

Model	-LogLikelihood	DF	ChiSquare	Prob>ChiSq
Difference	3305.967	440	6611.934	<.0001*
Full	24287.706			
Reduced	27593.673			

RSquare (U)	0.1198
AICc	49536.7
BIC	52845
Observations (or Sum Wgts)	10740

Little improvement over random groupings

Measure	Training	Definition
Entropy RSquare	0.1198	1-Loglike(model)/Loglike(0)
Generalized RSquare	0.4624	(1-(L(0)/L(model))^(2/n))/(1-L(0)^(2/...
Mean -Log p	2.2614	∑ -Log(ρ[j])/n
RMSE	0.8581	√ ∑(y[j]-ρ[j
Mean Abs Dev	0.8507	∑ |y[j]-ρ[j]|
Misclassification Rate	0.7438	∑ (P[j]
N	10740	n

~25% correct classification rate

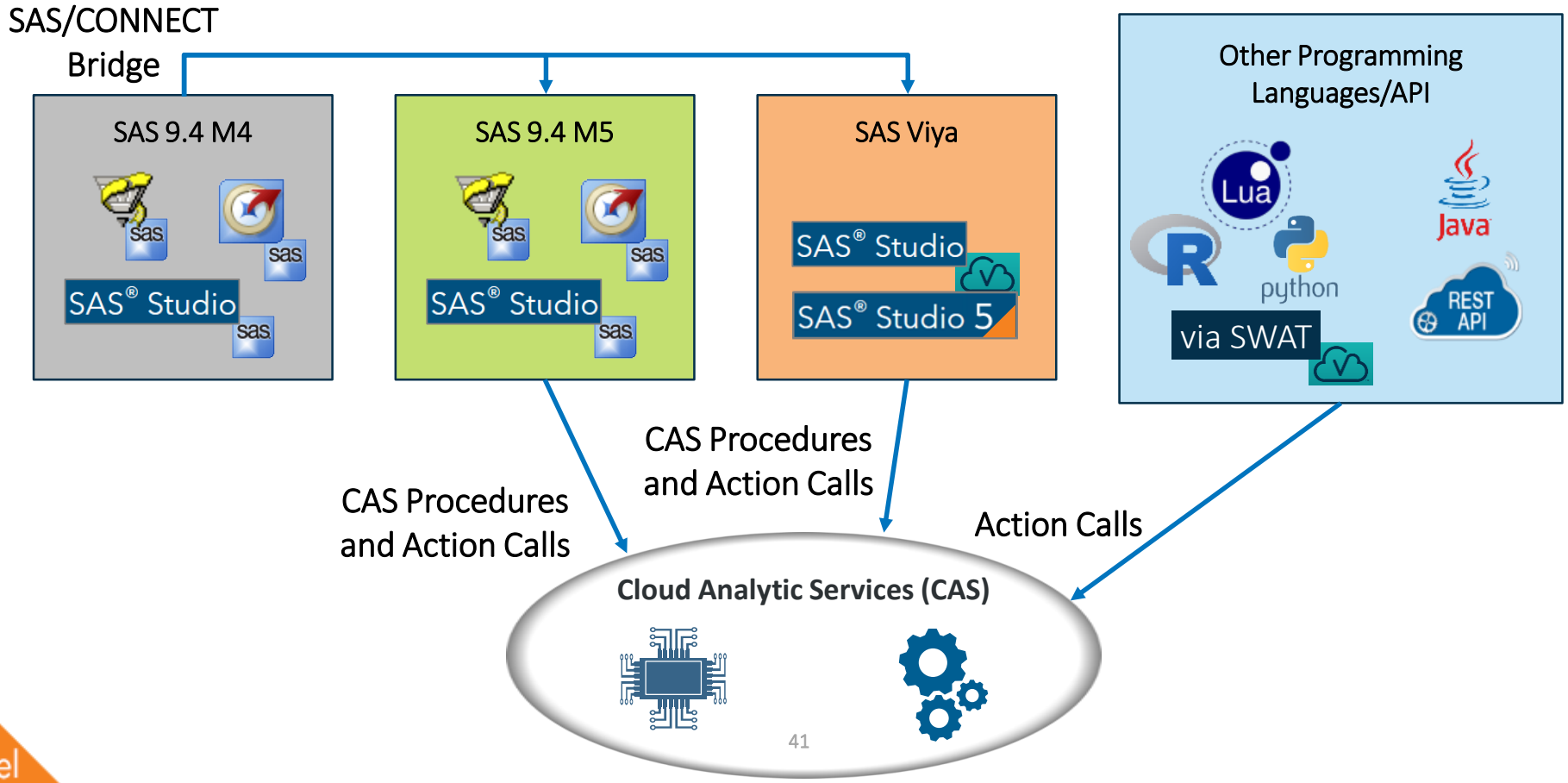
Lack Of Fit

Source	DF	-LogLikelihood	ChiSquare
Lack Of Fit	209740	24210.109	48420.22
Saturated	210180	77.598	Prob>ChiSq
Fitted	440	24287.706	1.0000

Python on Viya CAS

Running Python on CAS via SWAT

How to program against CAS?



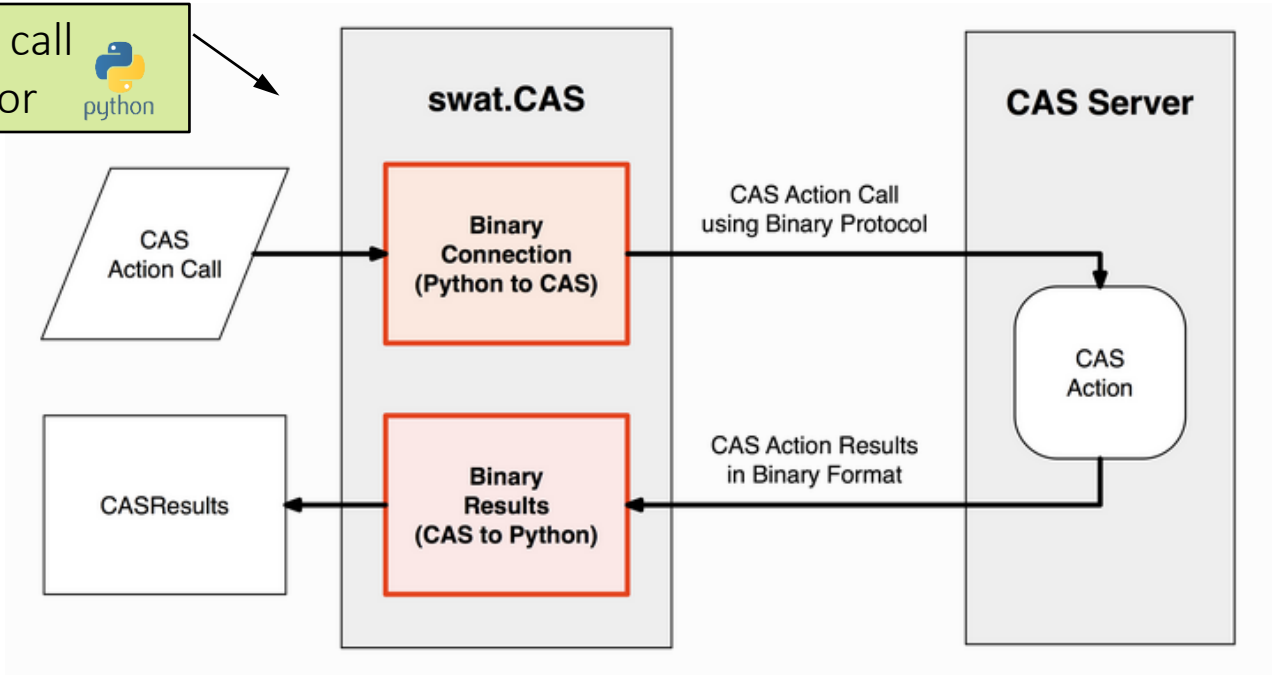
SAS and Open Source Integration

Base SAS	Interactive Matrix Language (IML)	Enterprise Miner	SAS Viya
• Execute open source code using system commands via the DATA step.	• Submit R code within IML from SAS Code Editor. • Pass data between R and SAS.	• Execute R code from the open source integration node. • Execute Python code using a system command.	• Use open source software to take control of analytical tools.

What is SWAT?

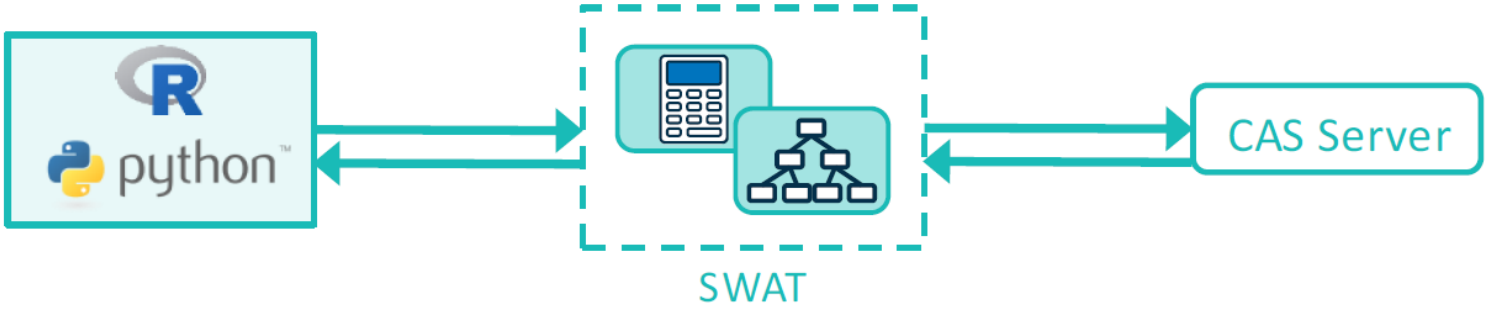
- SAS Scripting Wrapper for Analytics Transfer
- Enables Python, Lua, and R to interface with CAS

Example CAS call using SWAT for python



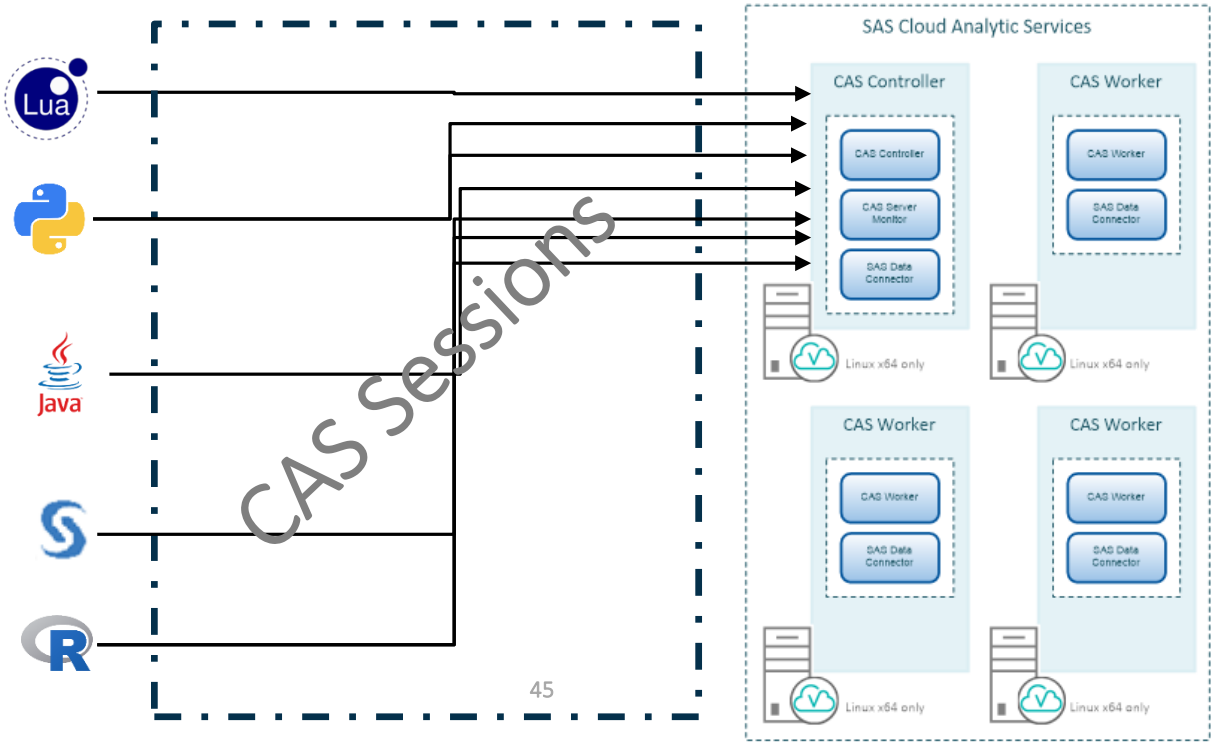
Scripting Wrapper for Analytics Transfer (SWAT)

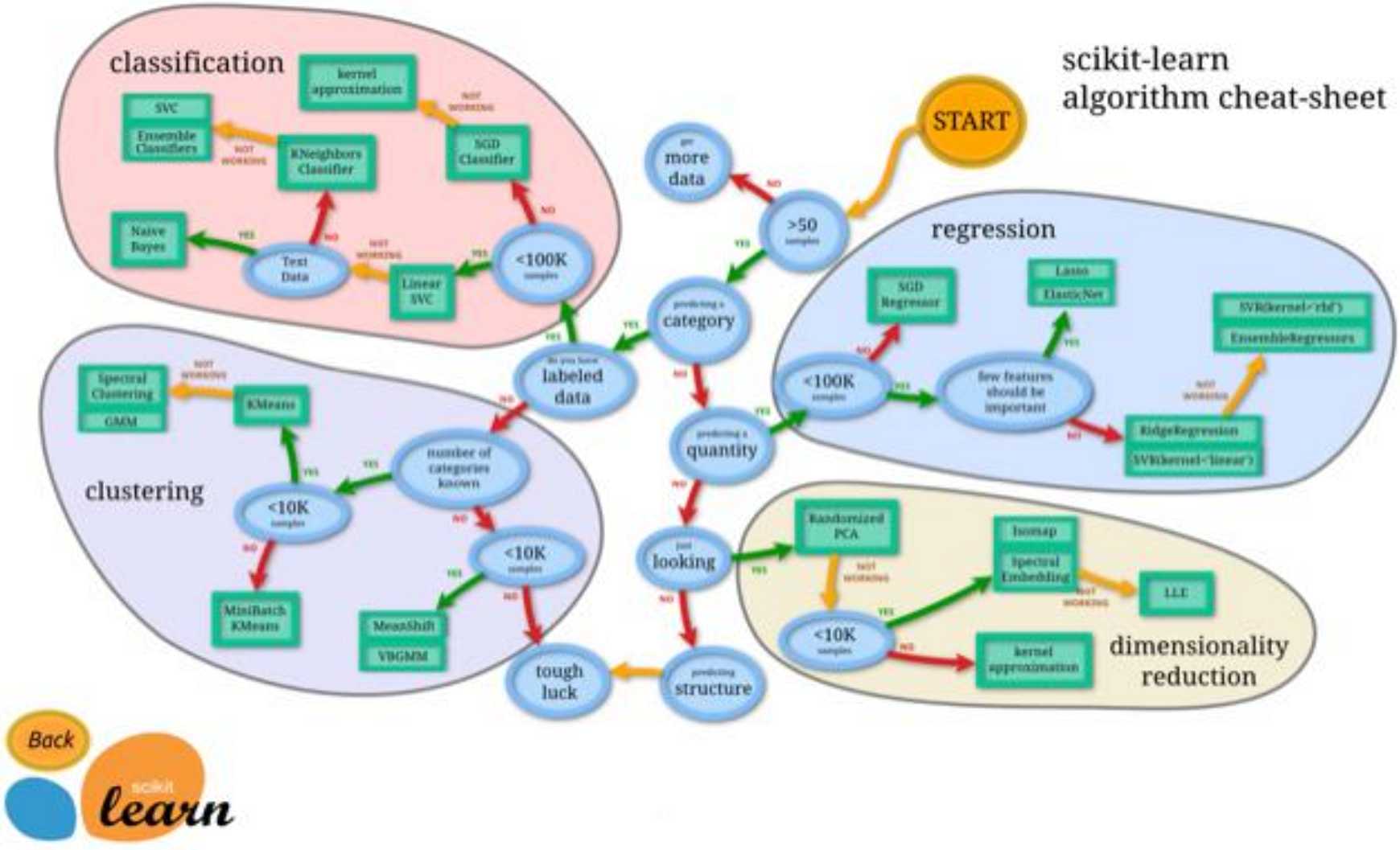
- The SWAT package enables you to interface with CAS from R or Python.
- You can write an R or Python program that connects to a CAS server, load data into CAS, analyze large in-memory data sets quickly and efficiently using CAS actions, and work with results of your analyses using familiar data wrangling techniques in the open source language.



How do you submit code?

CAS Session	Used to enable clients to communicate with the server to request actions.

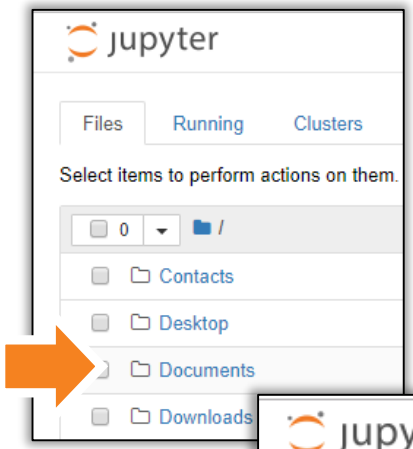


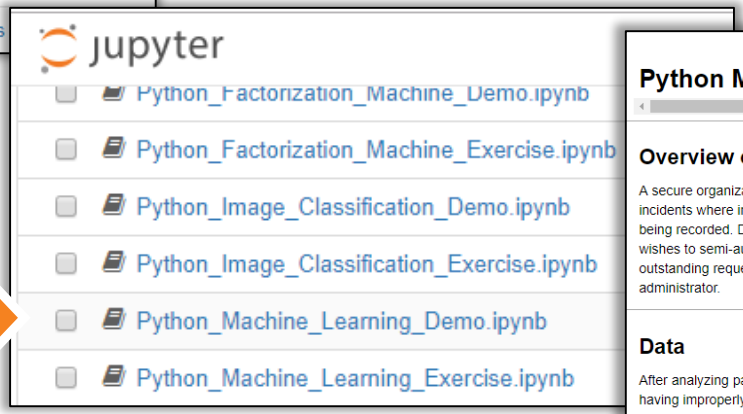

scikit-learn
algorithm cheat-sheet
START
classification
kernel approximation
SVC
Ensemble Classifiers
KNeighbors Classifier
SGD Classifier
Naive Bayes
Text Data
Linear SVC
<100K
NOT WORKING
NO
YES
more data
>50
category
labeled data
regression
SGD Regressor
Lasso
ElasticNet
SVR(kernel='rbf')
EnsembleRegressors
<100K
few features should be important
RidgeRegression
SVR(kernel='linear')
clustering
Spectral Clustering
GMM
KMeans
number of categories known
<10K
MiniBatch KMeans
MeanShift
VBGMM
quantity
just looking
Randomized PCA
Isomap
Spectral Embedding
LLE
<10K
kernel approximation
dimensionality reduction
tough luck
structure
Back
scikit learn

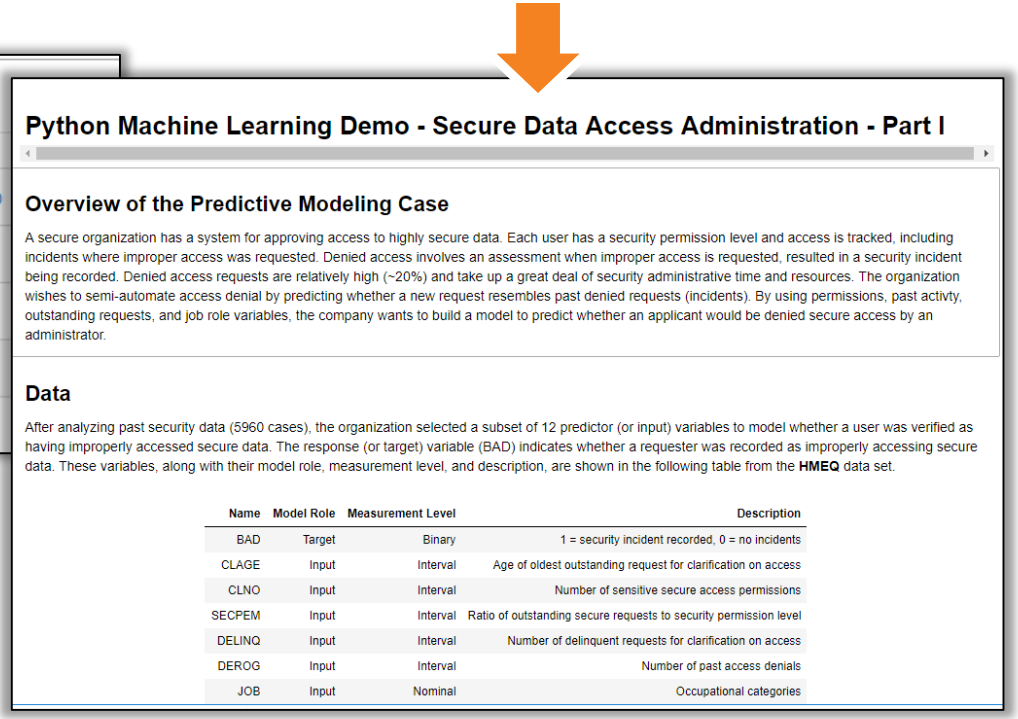
Predicting Security Violations

This demonstration illustrates how to predict when a user requesting secure access should be denied based on past behavior and profile

OPEN JUPYTER NOTEBOOK







Python Machine Learning Demo - Secure Data Access Administration - Part I

Overview of the Predictive Modeling Case

A secure organization has a system for approving access to highly secure data. Each user has a security permission level and access is tracked, including incidents where improper access was requested. Denied access involves an assessment when improper access is requested, resulted in a security incident being recorded. Denied access requests are relatively high (~20%) and take up a great deal of security administrative time and resources. The organization wishes to semi-automate access denial by predicting whether a new request resembles past denied requests (incidents). By using permissions, past activty, outstanding requests, and job role variables, the company wants to build a model to predict whether an applicant would be denied secure access by an administrator.

Data

After analyzing past security data (5960 cases), the organization selected a subset of 12 predictor (or input) variables to model whether a user was verified as having improperly accessed secure data. The response (or target) variable (BAD) indicates whether a requester was recorded as improperly accessing secure data. These variables, along with their model role, measurement level, and description, are shown in the following table from the **HMEQ** data set.

Name	Model Role	Measurement Level	Description
BAD	Target	Binary	1 = security incident recorded, 0 = no incidents
CLAGE	Input	Interval	Age of oldest outstanding request for clarification on access
CLNO	Input	Interval	Number of sensitive secure access permissions
SECPEM	Input	Interval	Ratio of outstanding secure requests to security permission level
DELINQ	Input	Interval	Number of delinquent requests for clarification on access
DEROG	Input	Interval	Number of past access denials
JOB	Input	Nominal	Occupational categories

Python API on SAS Viya CAS

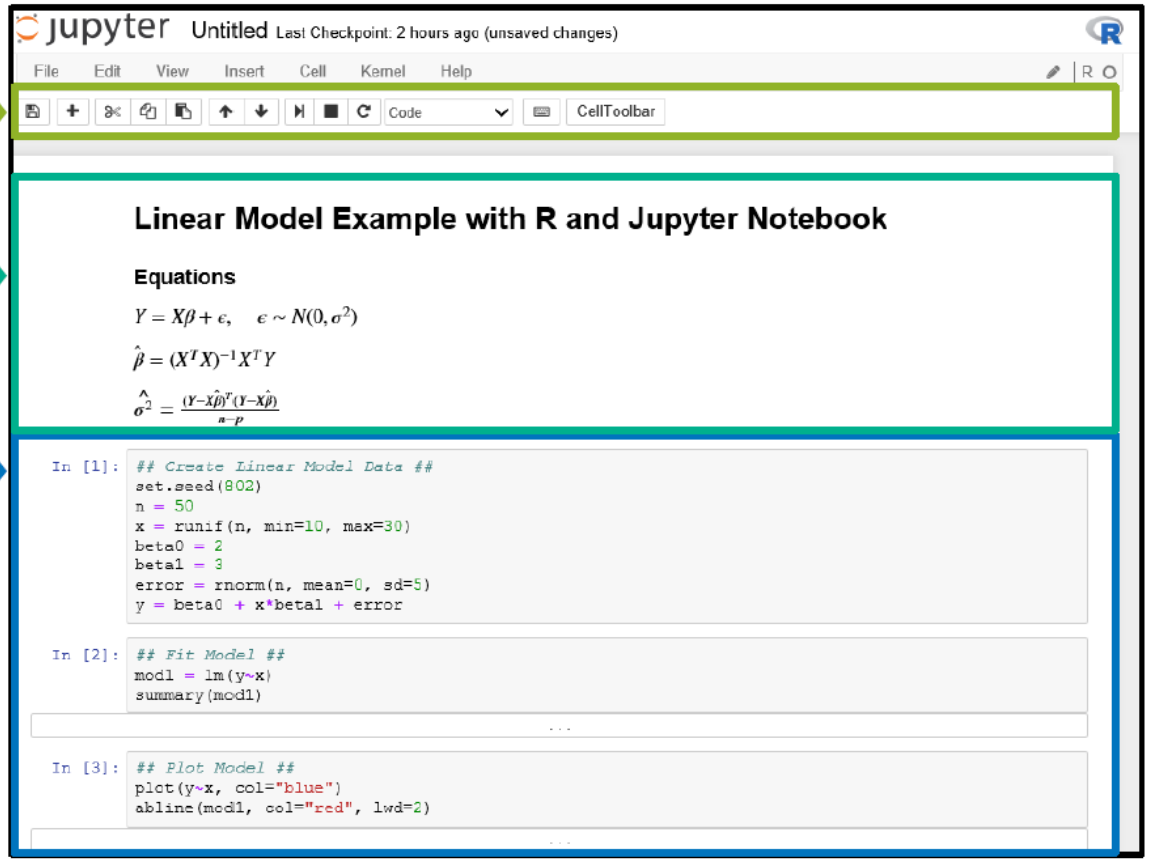
- **Anaconda 3 (distribution of tools)**: jupyter notebook, Spyder, Pscikit-learn, matplot lib, NumPy, pandas
- >> **DOCUMENTS**
  - Python_Machine_Learning_Demo.ipynb
  - Python_Machine_Learning_Exercise.ipynb (completed)
- **Three ways to run code**
  - Run icon (top bar)
  - Ctrl + Enter (selected segment)
  - Shift + Enter (run & move to next cell)

Jupyter Notebook

Toolbar

Markdown Cell

Code Cell



Jupyter Notebook Results

Results

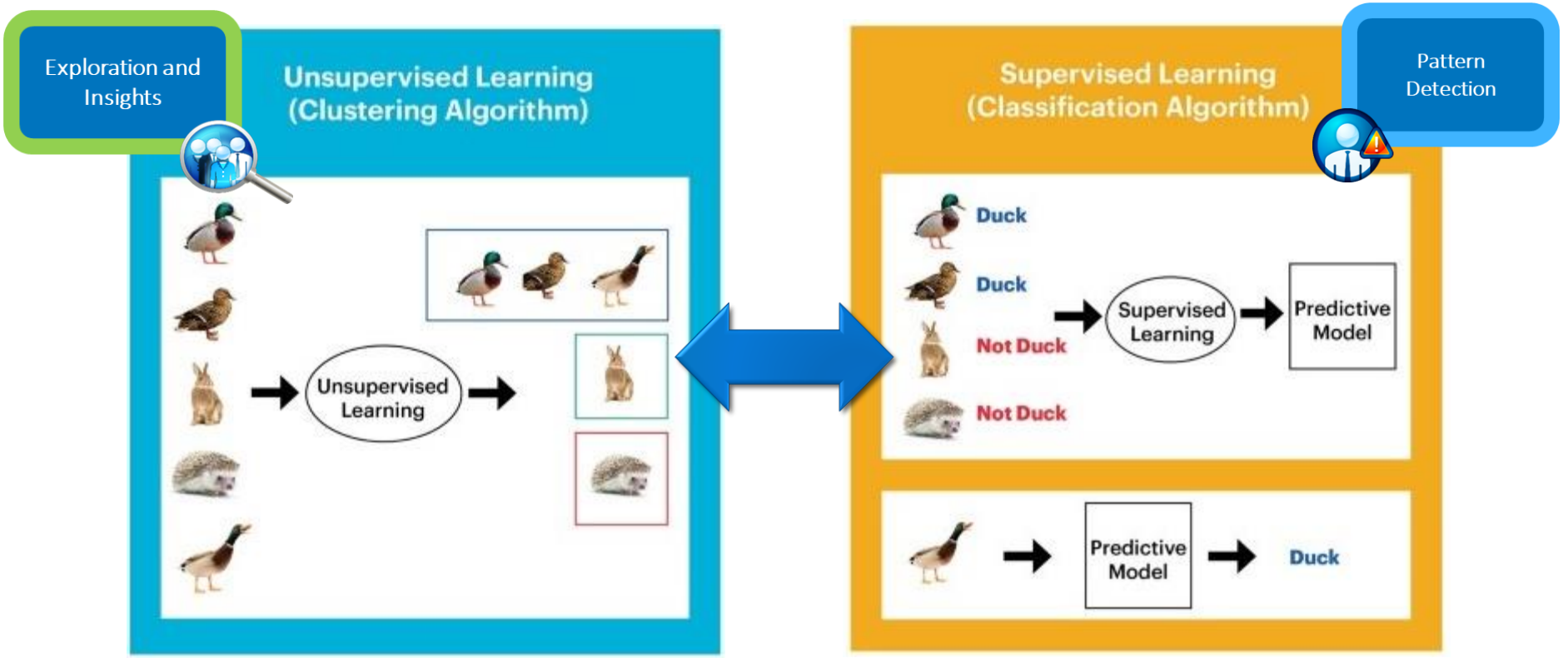
```
In [1]: ## Create Linear Model Data ##
        set.seed(802)
        n = 50
        x = runif(n, min=10, max=30)
        beta0 = 2
        beta1 = 3
        error = rnorm(n, mean=0, sd=5)
        y = beta0 + x*beta1 + error

In [2]: ## Fit Model ##
        mod1 = lm(y~x)
        summary(mod1)

Call:
lm(formula = y ~ x)

Residuals:
    Min      1Q  Median      3Q     Max
-9.3360 -2.7005 -0.2983  2.9151  8.9333

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  -0.6977     2.5214  -0.277    0.783
x             3.1214     0.1183  26.390   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 4.446 on 48 degrees of freedom
Multiple R-squared:  0.9355,    Adjusted R-squared:  0.9342
F-statistic: 696.4 on 1 and 48 DF,  p-value: < 2.2e-16

In [3]: ## Plot Model ##
        plot(y~x, col="blue")
        abline(mod1, col="red", lwd=2)
```

Results

Name	Model Role	Measurement Level	Description
BAD	Target	Binary	1 = security incident recorded, 0 = no incidents
CLAGE	Input	Interval	Age of oldest outstanding request for clarification on access
CLNO	Input	Interval	Number of sensitive secure access permissions
SECPEM	Input	Interval	Ratio of outstanding secure requests to security permission level
DELINQ	Input	Interval	Number of delinquent requests for clarification on access
DEROG	Input	Interval	Number of past access denials
JOB	Input	Nominal	Occupational categories
SECS	Input	Interval	Security sensitivity of requested data access
RECA	Input	Interval	Recent access to secure data
NINQ	Input	Interval	Number of recent access clarification inquiries
REASON	Input	Binary	Reason for access request
SECL	Input	Interval	Security permission level
YOJ	Input	Interval	Years at present job

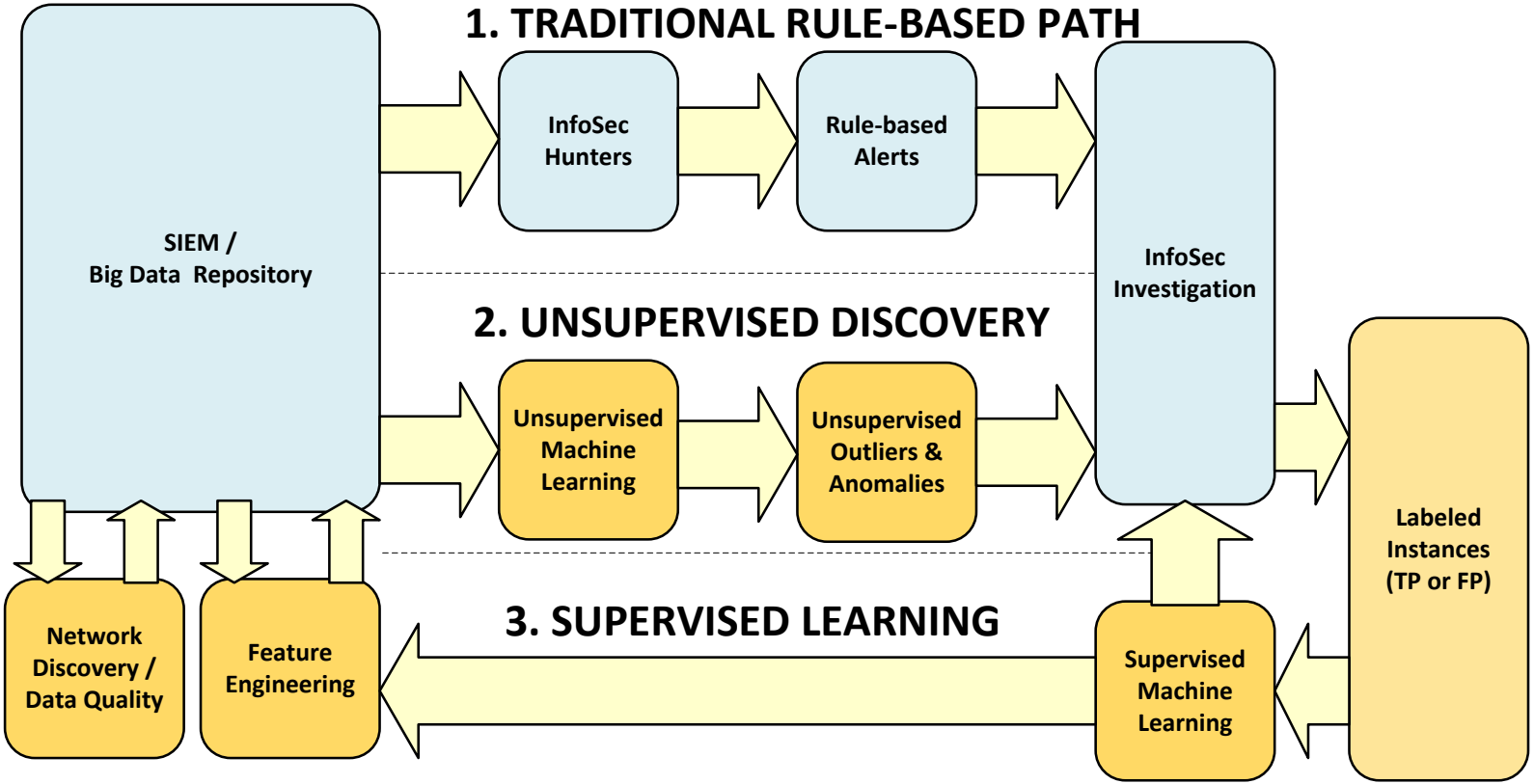
Semi-Supervised Learning

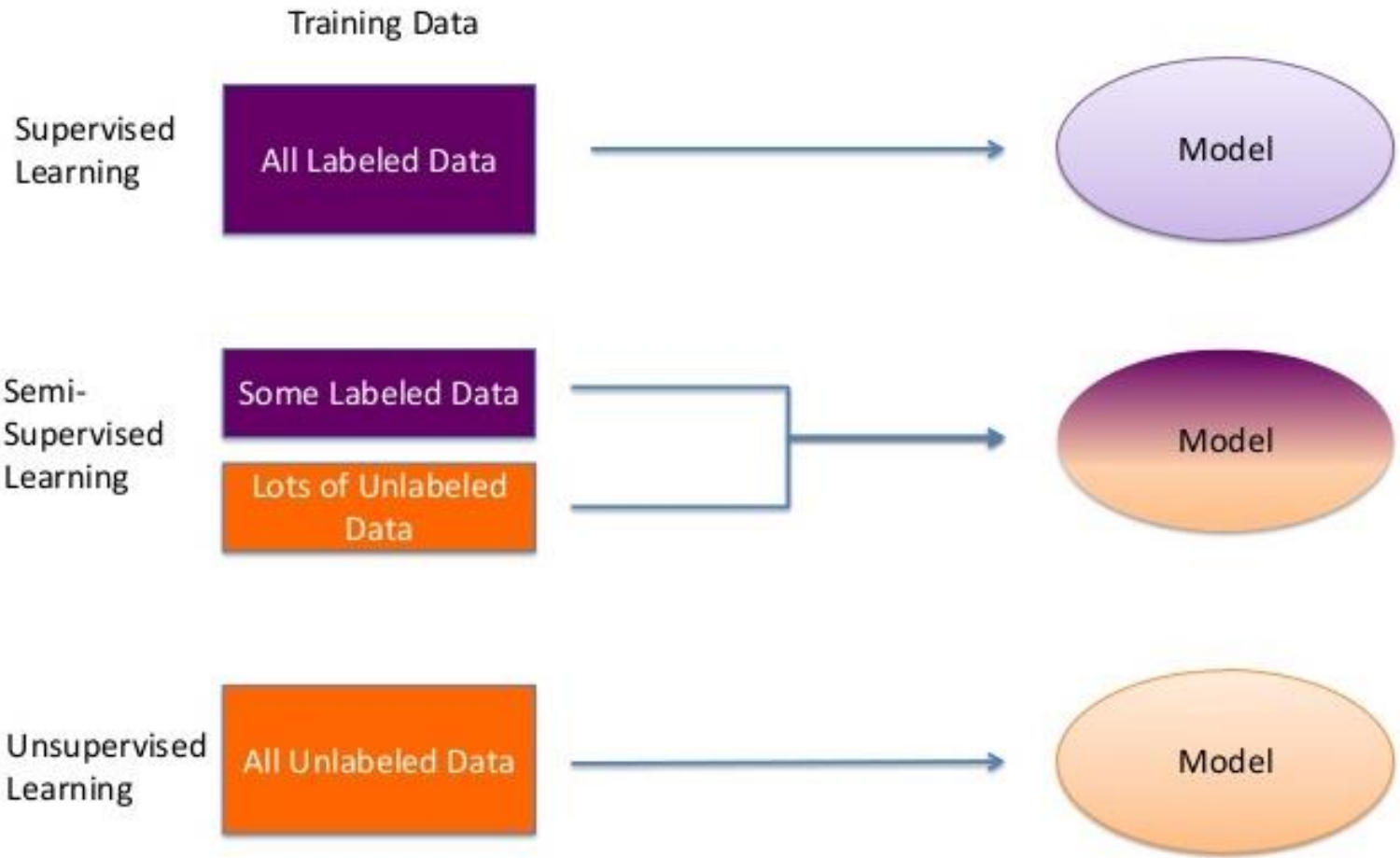
Machine Learning Segmentation and Classification



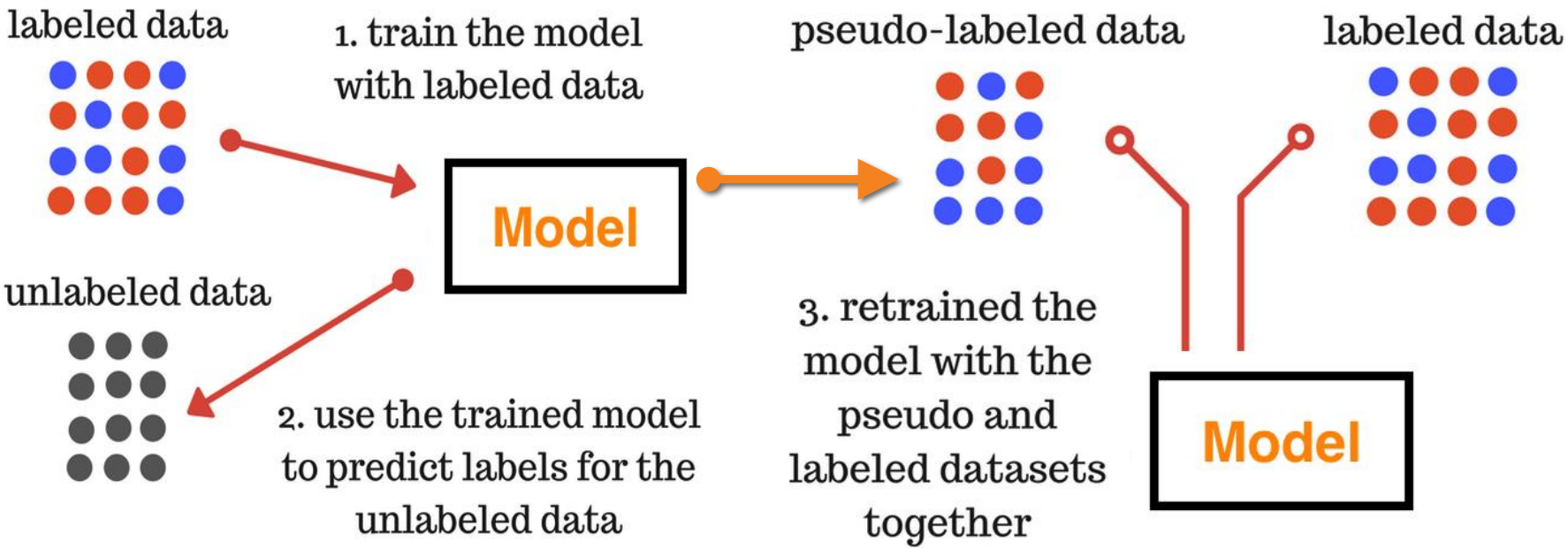
https://medium.com/datadriveninvestor/differences-between-ai-and-machine-learning-and-why-it-matters-1255b182fc6

Human-in-the-Loop: Self-Improving Cybersecurity Analytics Cycle




Training Data
Supervised Learning
All Labeled Data
Model
Semi-Supervised Learning
Some Labeled Data
Lots of Unlabeled Data
Model
Unsupervised Learning
All Unlabeled Data
Model

Semi-Supervised Approach



SOURCE: Analytics Vidhya. Introduction to Pseudo-Labelling : A Semi-Supervised learning technique
https://www.analyticsvidhya.com/blog/2017/09/pseudo-labelling-semi-supervised-learning-technique/

Machine learning tasks

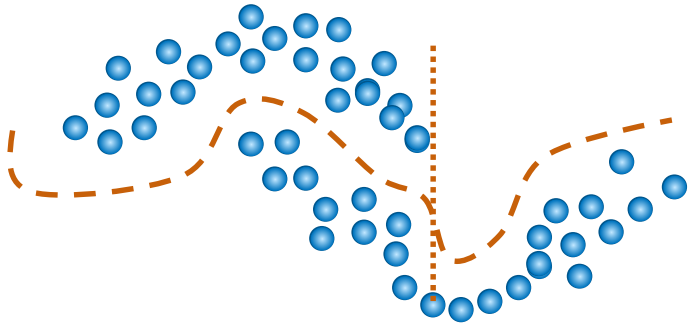
Regression
Decision trees
Gradient boosting
Random forests
Autoencoders
Text Processing
Image Processing

Semi-supervised Learning

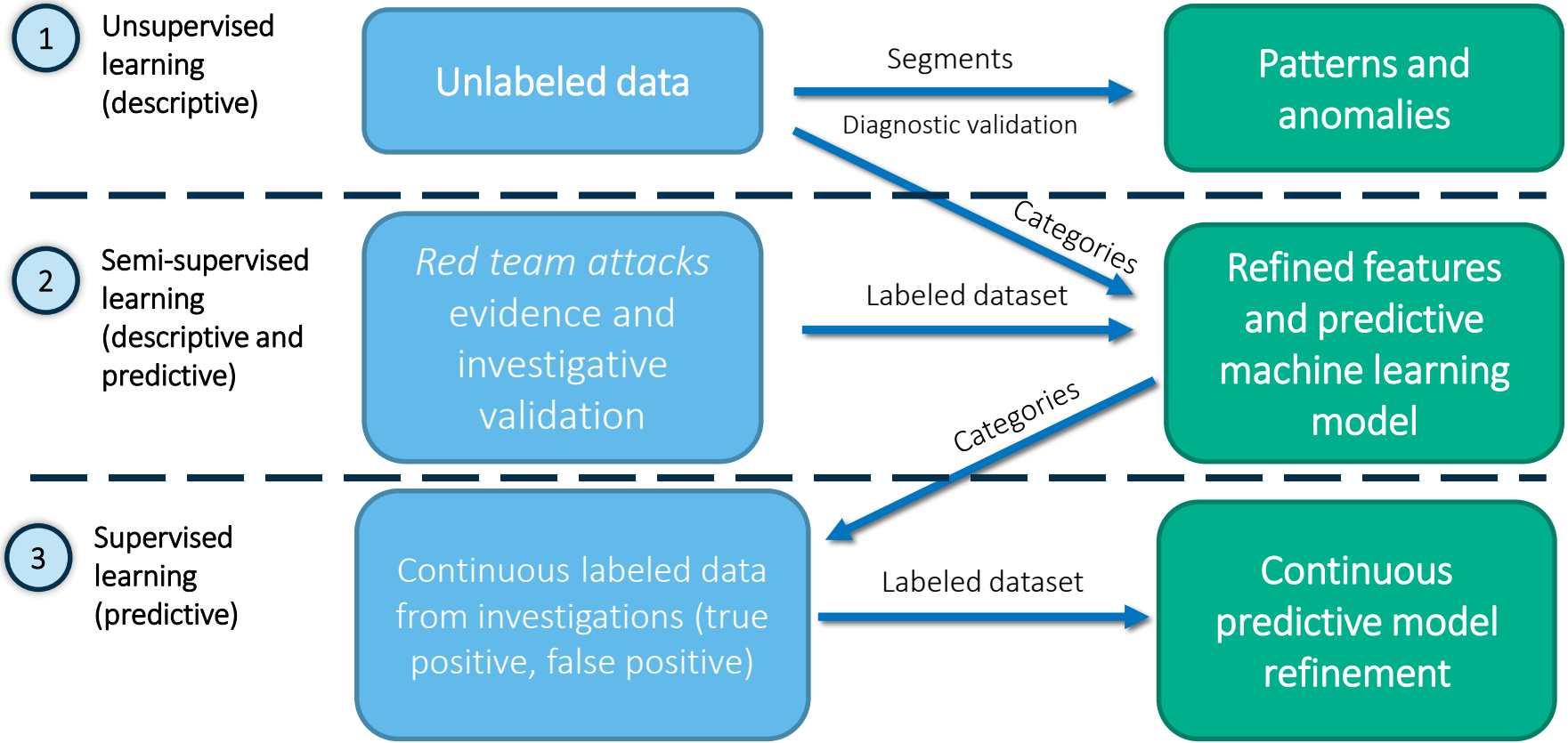
- Helpful when volume or variety of data is too high to allow labelling
- Utilized labeled and unlabeled examples
- Classification, regression, prediction

Considerations

- Best of both worlds?
- More automated



Moving from Anomalies to Focused Incident Detection



Reinforcement Learning

EXAMPLES

Monte Carlo

Neural Networks

Deep Forward Neural Networks

Convolutional Neural Networks

Recurrent Neural Networks

Reinforcement Learning

- Like teaching someone a game
- The machine takes actions and learns from results
- Maximize an expected future reward

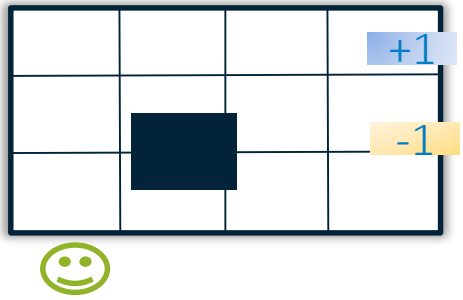
Considerations

- Data intensive – Data drives insight
- Hard to understand what was learned

Random Walk

- The agent lives in a grid
- Walls block the agent's path
- Agent's action has randomness
- Big reward comes at the end

Follow a good policy

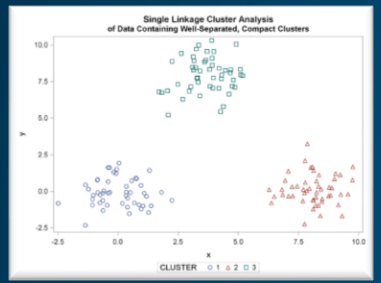


Wrap-Up

Section Review

Review: Machine Learning

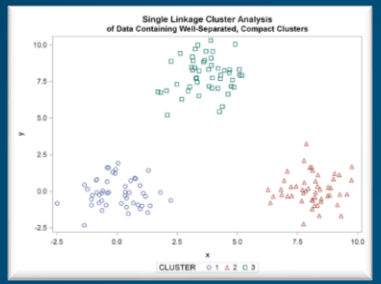
Unsupervised



- No targets required
- Finds similar 'groups' within the data

- Not a direct indicator of incident
- Will generally patterns in all data (need to have right data)

Supervised



- Easy to build
- Easy to understand

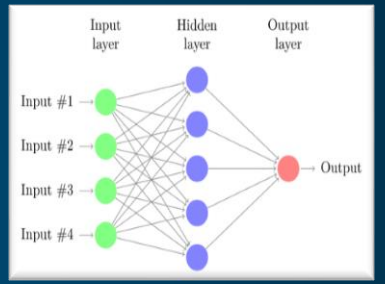
- Only known cases!
- Data skew

Semi-Supervised

- Useful with rare events
- Utilized both labeled & unlabeled examples

- Requires careful selection of cases
- Requires careful 'tuning'

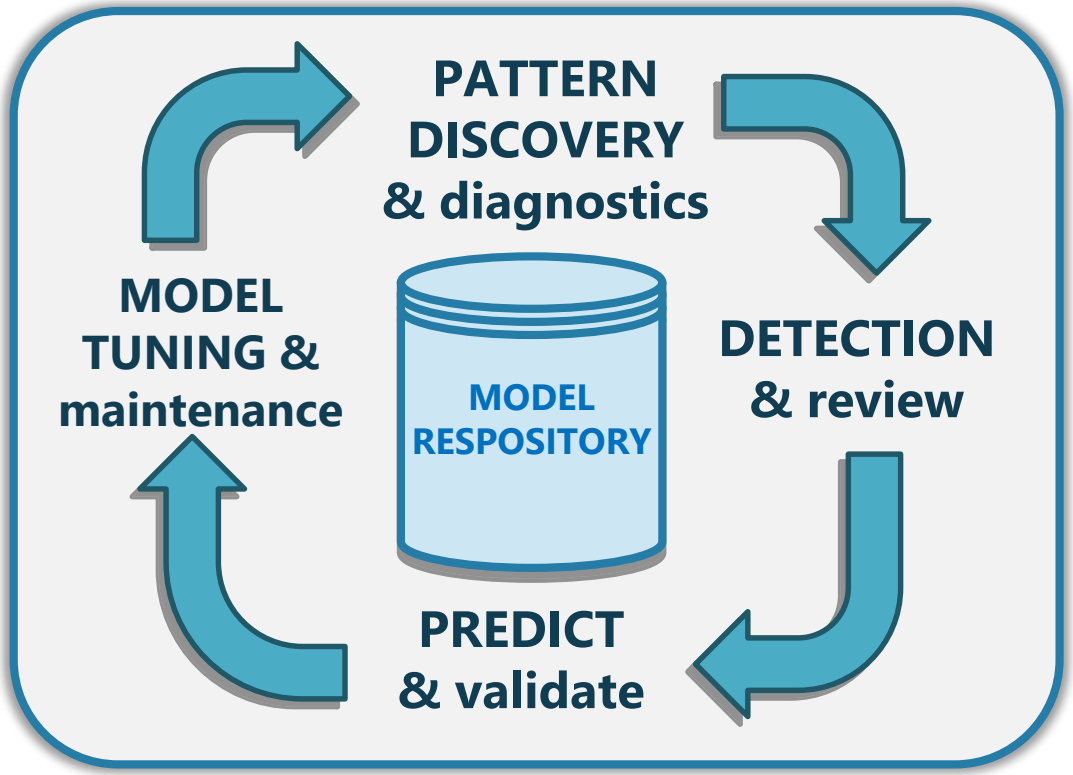
Reinforcement



- Self learning
- Improves over time
- Pattern analysis

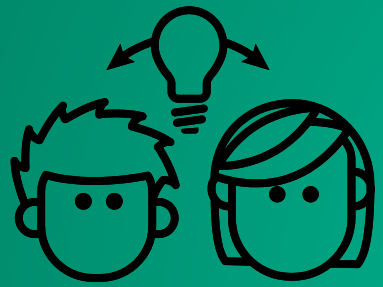
- Black box
- Large training set required
- Overfitting a problem

Machine Learning as a Process



SOURCES

http://www.oreilly.com/data/free/archive.html



Idea Exchange

What do we do when there are very few examples of known incidents?

Cybersecurity Analytics Maturity

Anomaly Detection	Data-aware Investigations	Predictive Detection	Risk Awareness / Resource Optimization
• Big data management • Flags, rules, and alerts --- • Multivariate statistics, inference & unsupervised machine learning • Segments extracted as baselines	**Understanding** • Feature engineering • Labeling • Diagnostics • *Unsupervised ML*	?	?

Cybersecurity Analytics Maturity

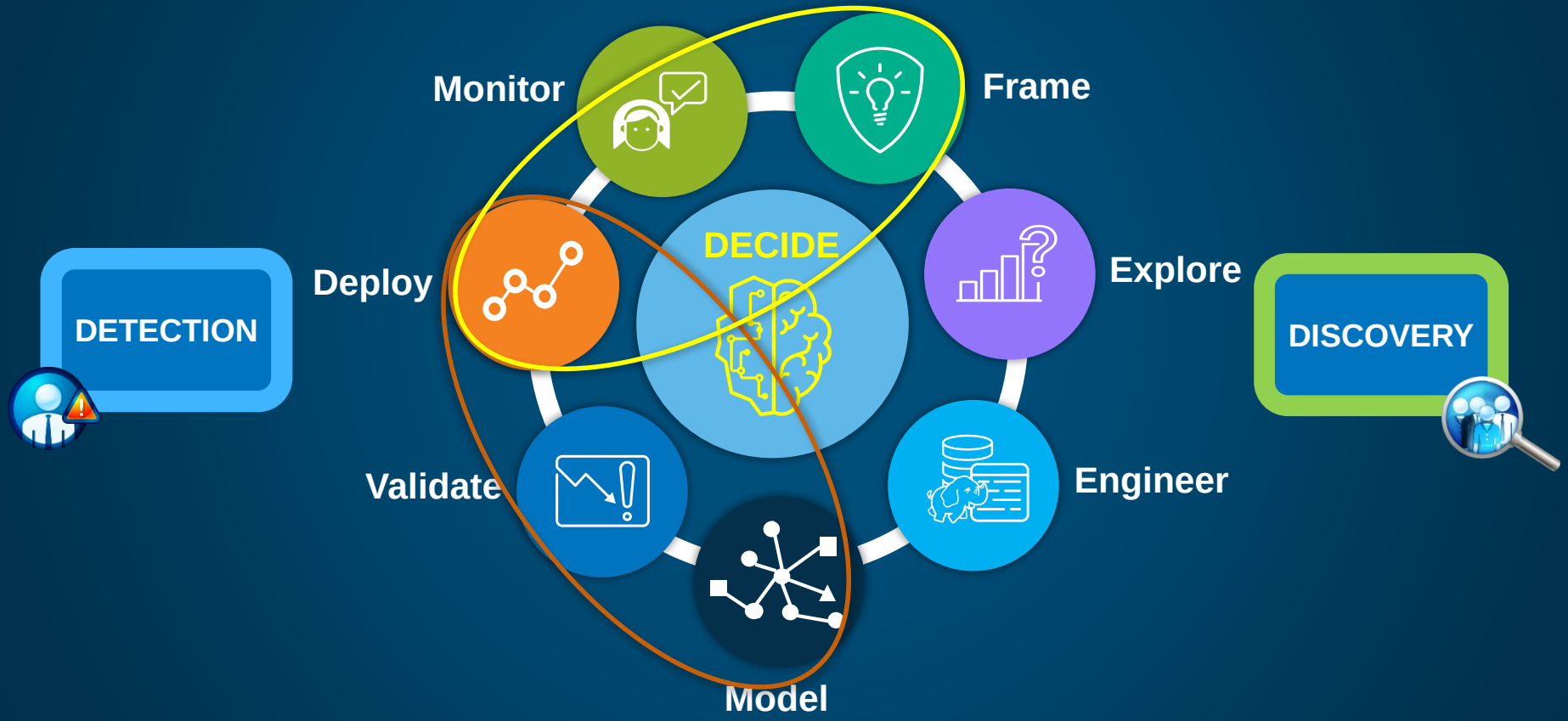
Anomaly Detection	Data-aware Investigations	Predictive Detection	Risk Awareness / Resource Optimization
• Big data management • Flags, rules, and alerts • Multivariate statistics, inference & unsupervised machine learning • Segments extracted as baselines	**Understanding** • Feature engineering • Labeling • Diagnostics • *Unsupervised ML*	**Learning** • Human-in-the-loop reviews • *Combined supervised and unsupervised machine learning*	







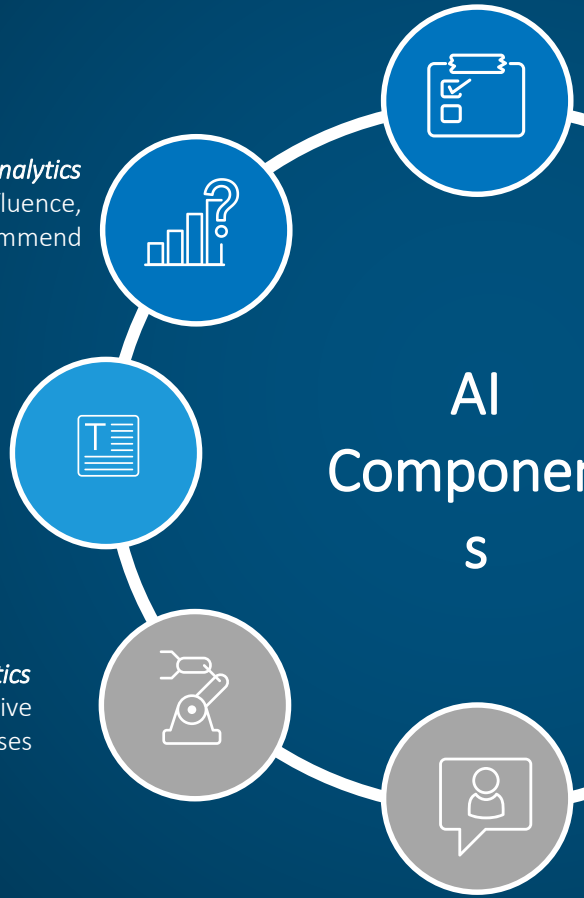
Cybersecurity Data Science (CSDS) Lifecycle



APPENDIX

Deep Learning

What about AI?



Rules-based systems
Simplest form of automation, the execution of rules

Machine Learning
Advanced analytic algorithms create insight with more automation

Deep Learning
Self learning algorithms that deliver even more insight and automation

Computer Vision
The addition of images and video

Natural Language Understanding
Both natural language ingestion and generation.

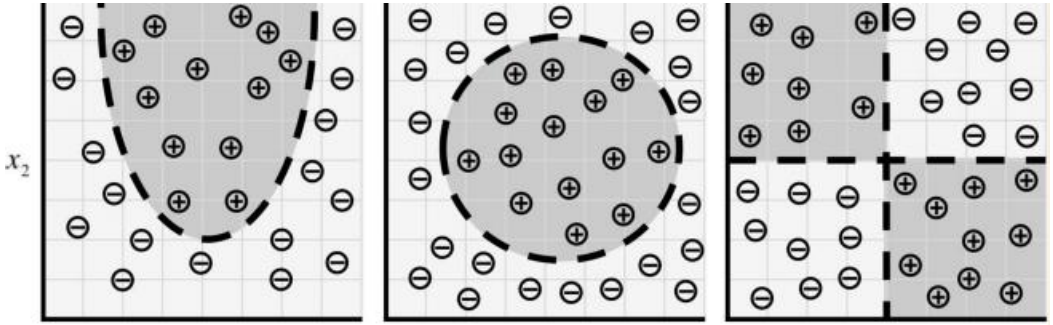
Robotics
Automate repetitive functions and processes

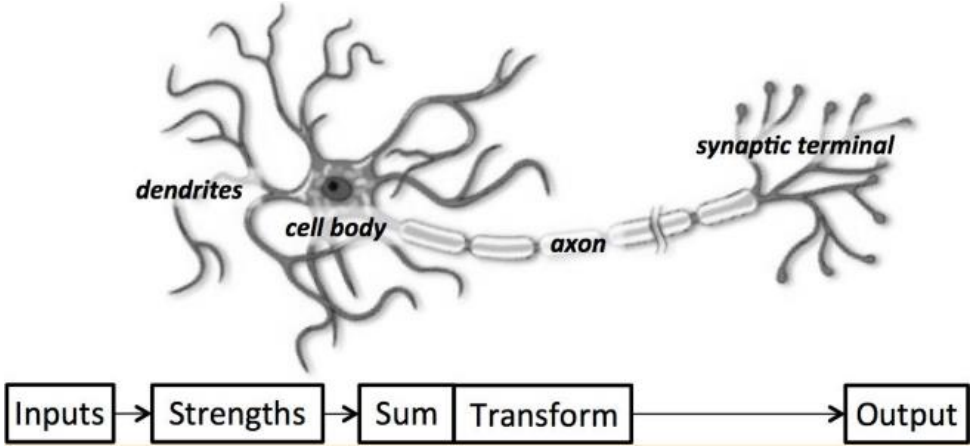
Text Processing
The addition of unstructured text

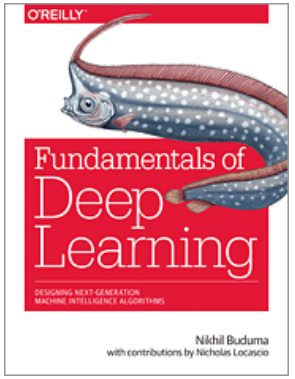
Predictive Analytics
Predict, advise, influence, recommend

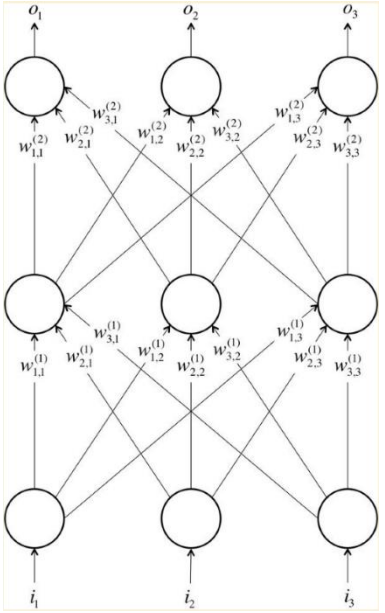
Deep Learning

Nested Neural Networks: train models for complex domains with big data



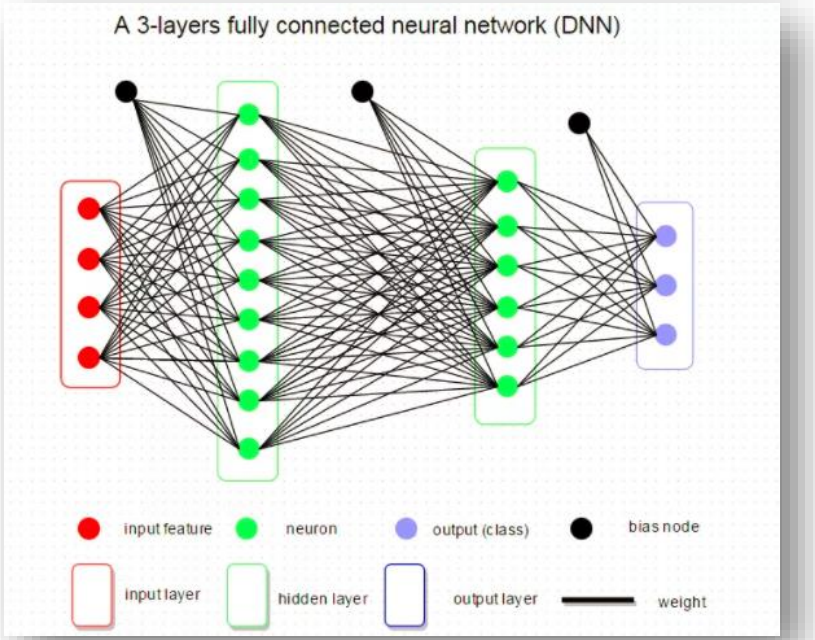






Deep Learning

A specialization of machine learning

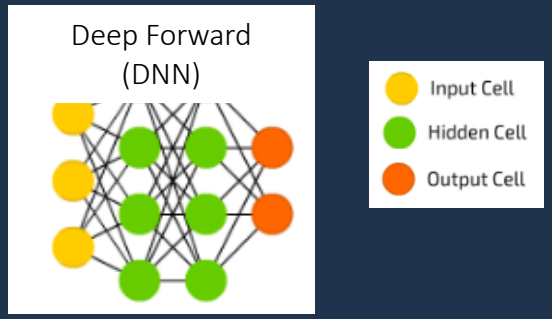


- Neural networks with many layers and different types of …
  - Activation functions
  - Network architectures
  - Sophisticated optimization routines

  Each layer represents an optimally weighted, non-linear combination of the inputs
- Automatic feature generation
- Extremely accurate results if well-trained; use for classification, prediction, or pattern recognition, especially in unstructured data

Image from: https://www.gabormelli.com/RKB/Multi_Hidden-Layer_Neural_Network

Algorithm



Practical Applications:

- Fraud (& cyber)
- Marketing analytics
- Text pre-processing

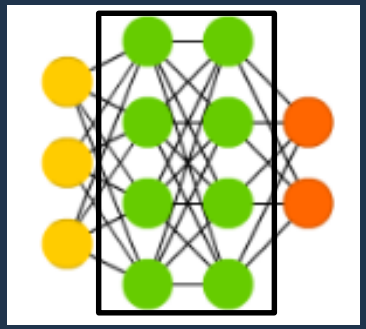
http://www.asimovinstitute.org/neural-network-zoo

Deep Forward Neural Networks

- Simplest type of Deep Learning algorithm, where information is only fed forward, from input to output
- While often combined with other networks to from new networks, small or shallow DFNNs have been shown to be effective with certain tasks, making them more desirable for resource constrained environments (such as cell phones) – versus other, more resource-hungry algorithms
- Can also be applied to NLP tasks including: language identification, part-of-speech tagging, word segmentation, and preordering for statistical machine translation

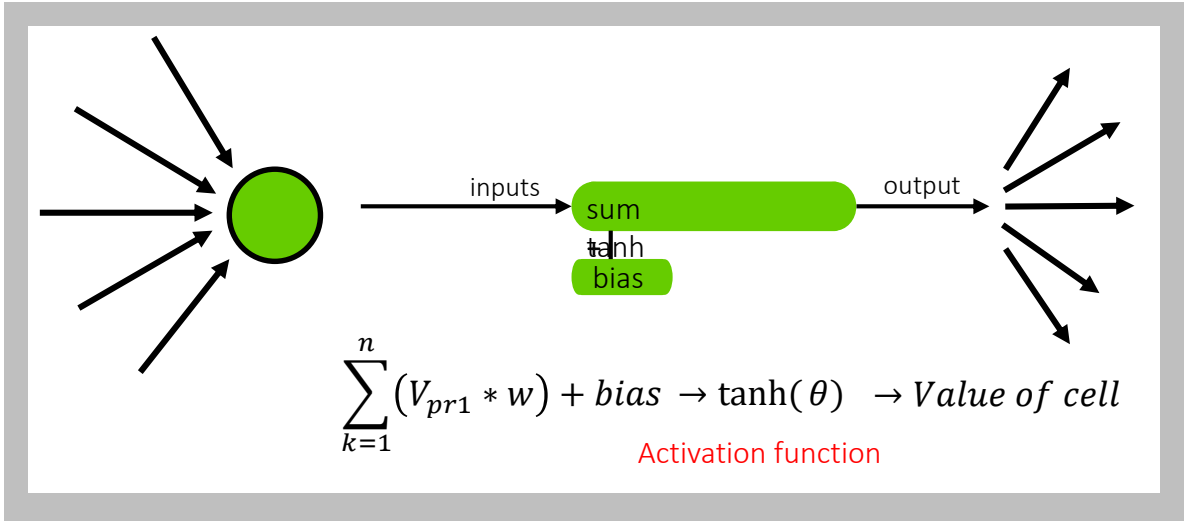
Components

basic feed-forward cell ("neuron")



Fully-connected layers

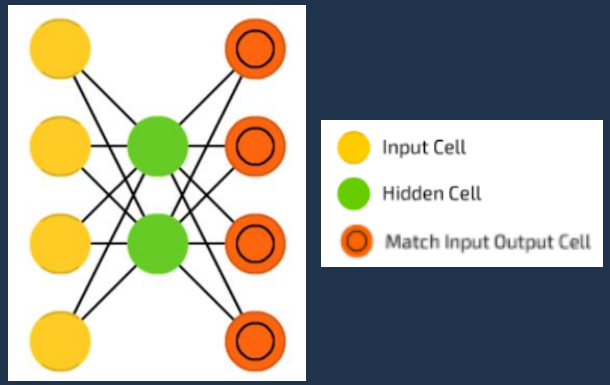
Basic Neurons and Fully-Connected Layers



In a Deep Forward Neural Net:

- Each neuron is connected to all other neurons (**fully-connected layers**)
- All connected previous values, multiplied by their weights, are added to bias and then fed through an activation function

Algorithm



Practical Applications:

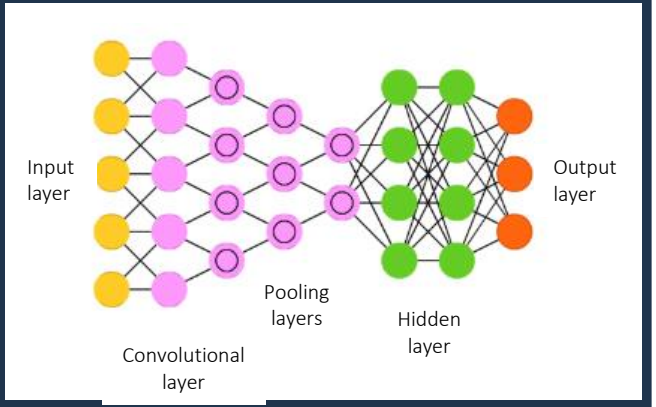
- Anomaly detection
- Feature extraction
- Dimension reduction (VAE)

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

Auto Encoders

- Unsupervised learning models used to encode information via a fully-connected, hour-glass shaped architecture which is always symmetrical around the middle layer(s), where the info is most compressed
- In its simplest form, the output layer is a copy of the input layer and has the task of reconstructing its own inputs (rather than predicting target values y given inputs x)
- The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction and/or noise reduction. Recently, the autoencoder concept has become more widely used for learning generative models of data

Algorithm



Practical Applications:

- Image/object recognition
- Video analysis
- Text Classification

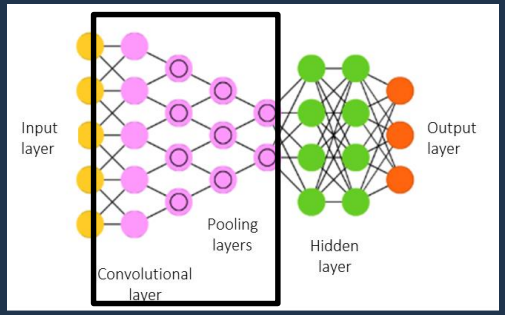
Convolutional Neural Networks

- Versatile, though best-known for their prowess at image analysis tasks. CNNs can detect components in images such as edges and curves, and learn how these features are combined in order to detect larger structures such as faces, numbers, etc.
- Because CNNs preserve spatial structure, they are well-suited to array data where nearby values are correlated (images, sound, video, speech, etc.).
- CNNs are relatively fast because convolutions are critical in computer graphics and can be implemented on GPUs
- For text analysis, word-level or character-level CNNs can be applied for tasks such as sentiment analysis, categorization, or spam detection. Note: because CNNs require fixed input and output sizes, padding is often required for inputs.

Components

Convolutional Layers

- Each neuron is only connected to cells within a certain proximity and NOT connected to all other neurons in the previous layer
- Also called “filters”

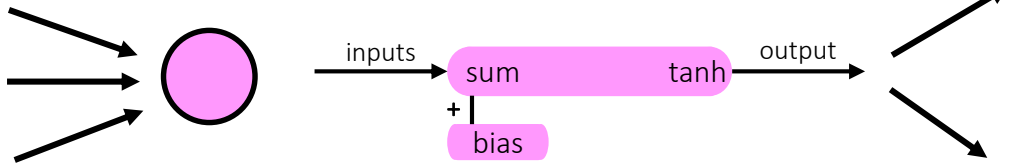


Pooling Layers

- Used for down-sampling, with maxpooling being most popular
- Once a feature is known to be present, its exact location is not as important as its relative location to the other features

Convolutional and Pooling Layers

Convolutional Layer

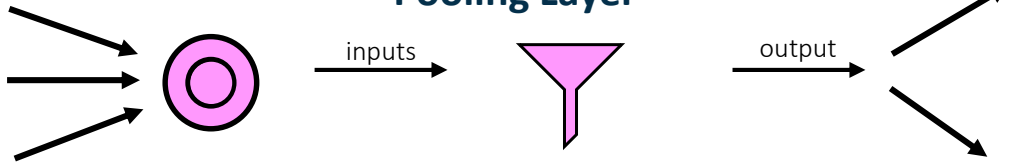


cell is only connected to neighbors within a certain proximity

$$\sum_{k=1}^{x}(V_{pr1} * w) + bias \rightarrow \tanh(\theta) \rightarrow Value\ of\ cell$$

Activation function

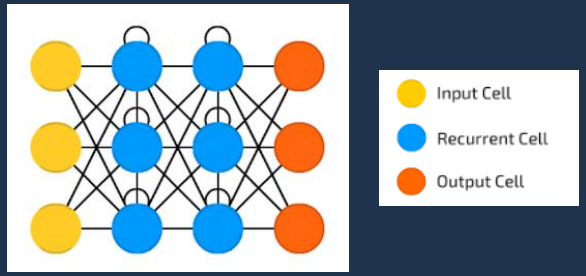
Pooling Layer



Filter options:

- Max (most common)
- Average
- L2-norm

Algorithm



Practical Applications:

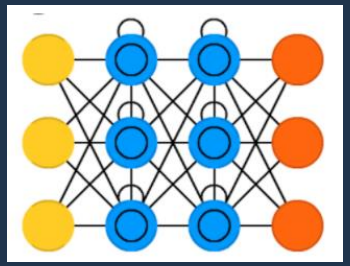
- Language modeling (e.g., statistical machine translation, word prediction)
- Time-series forecasting
- Image/video captioning

Recurrent Neural Networks

- Versatile, but especially well suited for sequence data like time series or text because an RNN will recognize patterns across time or the order of words in a given language.
- Unlike CNNs, RNNs can handle variable-length inputs and produce variable-length outputs, which is useful for applications like NLG and machine translation.
- Stacked RNNs can form a net capable of more complex output (e.g., used for autonomous vehicles)
- Note: even if your data is not sequential, you can still process it sequentially and reap the benefits of RNNs

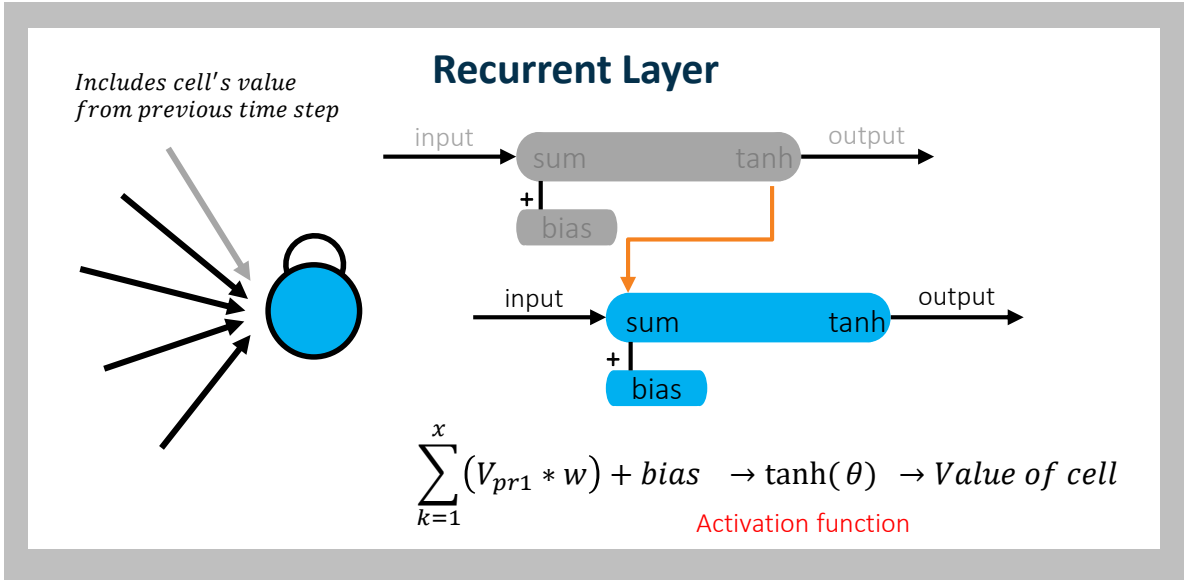
Components

Recurrent cell



Time-Delayed Layers

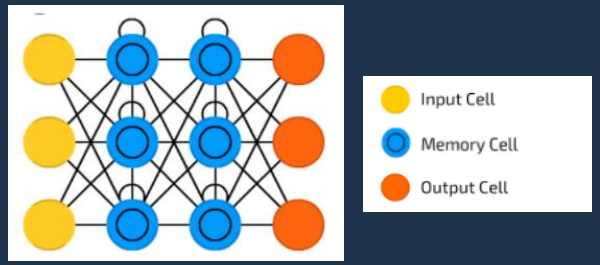
Recurrent (Time-Delayed) Layers



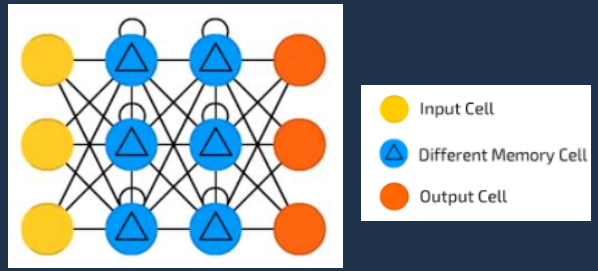
Time-Delayed Layers:

- They are connected to all neurons in the previous layer and updated just like basic cells, but with extra weights: connected to the previous values of the cells and most of the time also to all the cells in the same layer. These weights between the current value and the stored previous value work much like a volatile memory (like RAM).

Algorithm Variants



Long Short Term Memory



Gated Recurrent Unit

Long Short Term Memory (LSTM) RNNs and Gated Recurrent Unit (GRU) RNNs

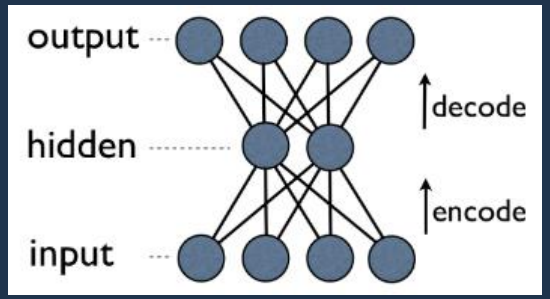
- RNN variants that help with the problem of the “vanishing” or “exploding” gradient by introducing gates and an explicitly defined memory cell
- Can learn complex sequences (e.g., write like Shakespeare, compose primitive music) but typically require more resources to run. In cases where extra expressiveness is not needed, GRUs can outperform LSTMs.
- Each neuron has a memory cell and 2-3 gates which safeguard the information by stopping or allowing the flow of it.

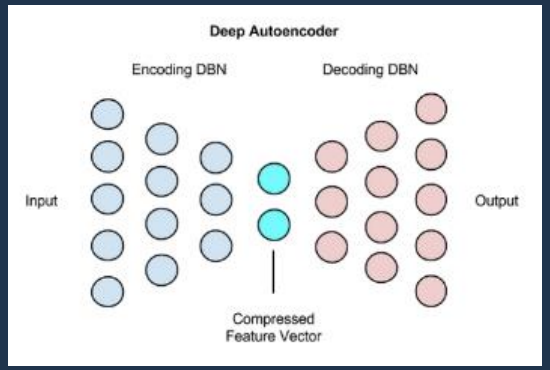
LSTM RNN

- Input gate
- Output gate
- Forget gate

GRU RNN

- Update gate
- Reset gate





What are they?

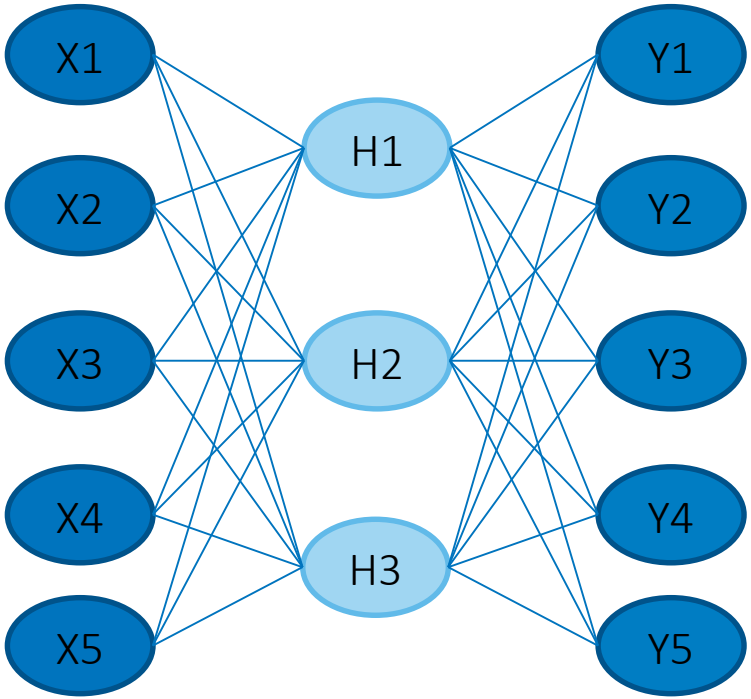
The basic idea behind autoencoders is to encode information automatically.

It requires at least 3 layers with perfect symmetry on both sides.

Since the output layer is a copy of the input layer there is no target.

The output layer is trying to represent the input layer so anything that stands out from the input layer is anomalous.

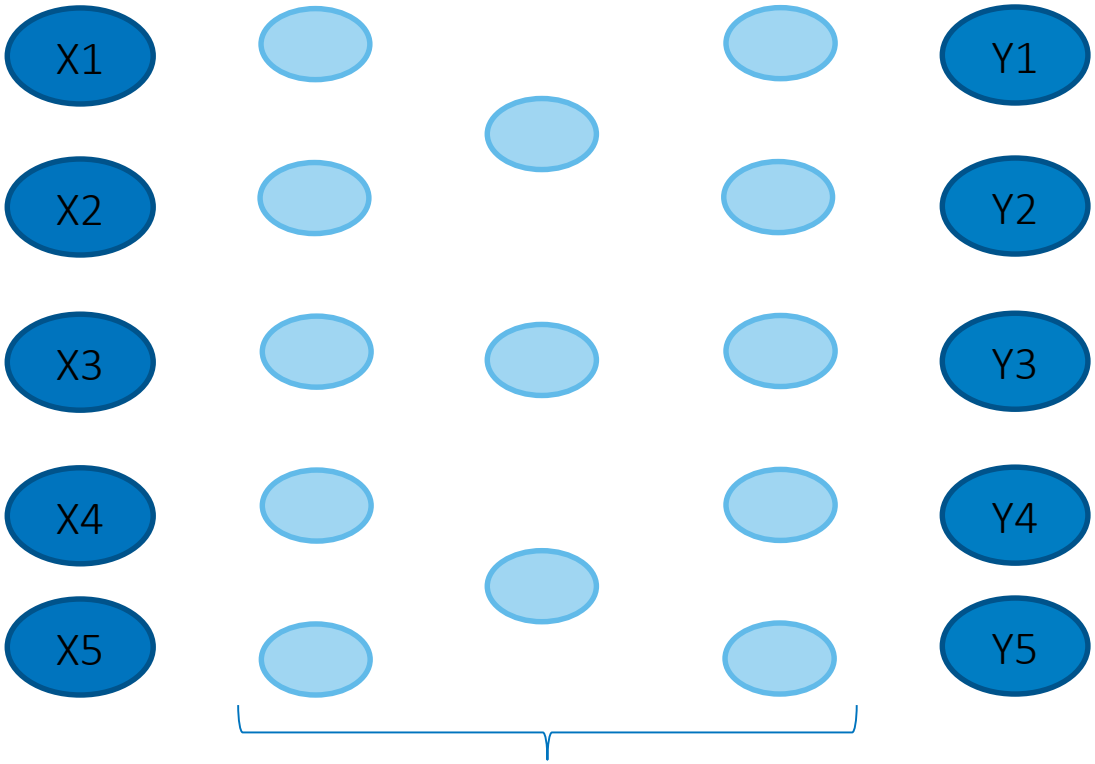
Autoencoder neural network



Output layer is copy of input layer, so no target

Fully-connected, symmetric

Autoencoder neural network



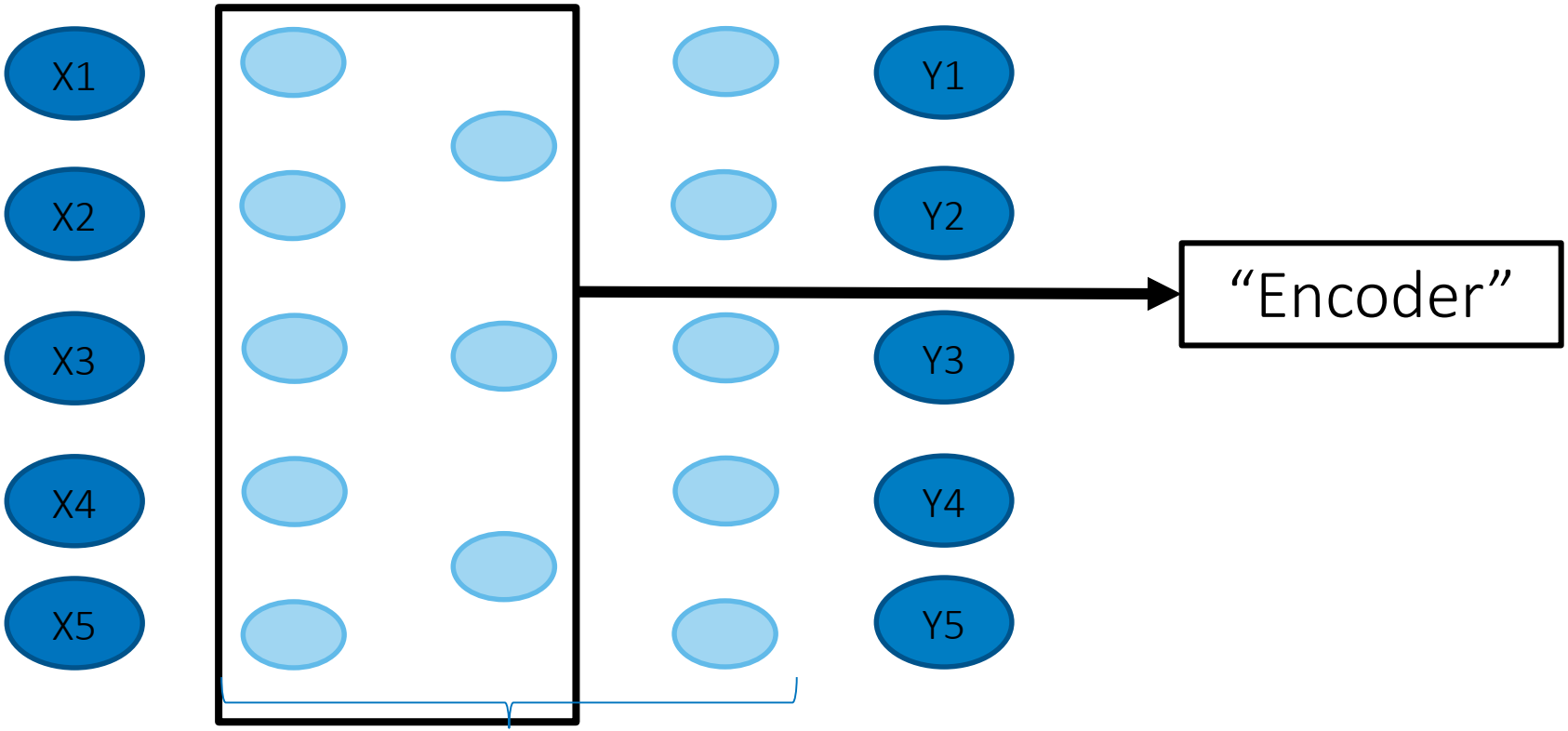
3 symmetric, hidden layers: 5 units, 3 units, 5 units

Odd number of hidden layers

Hidden layers symmetric in number of units, typically decreasing to middle hidden layer then increasing

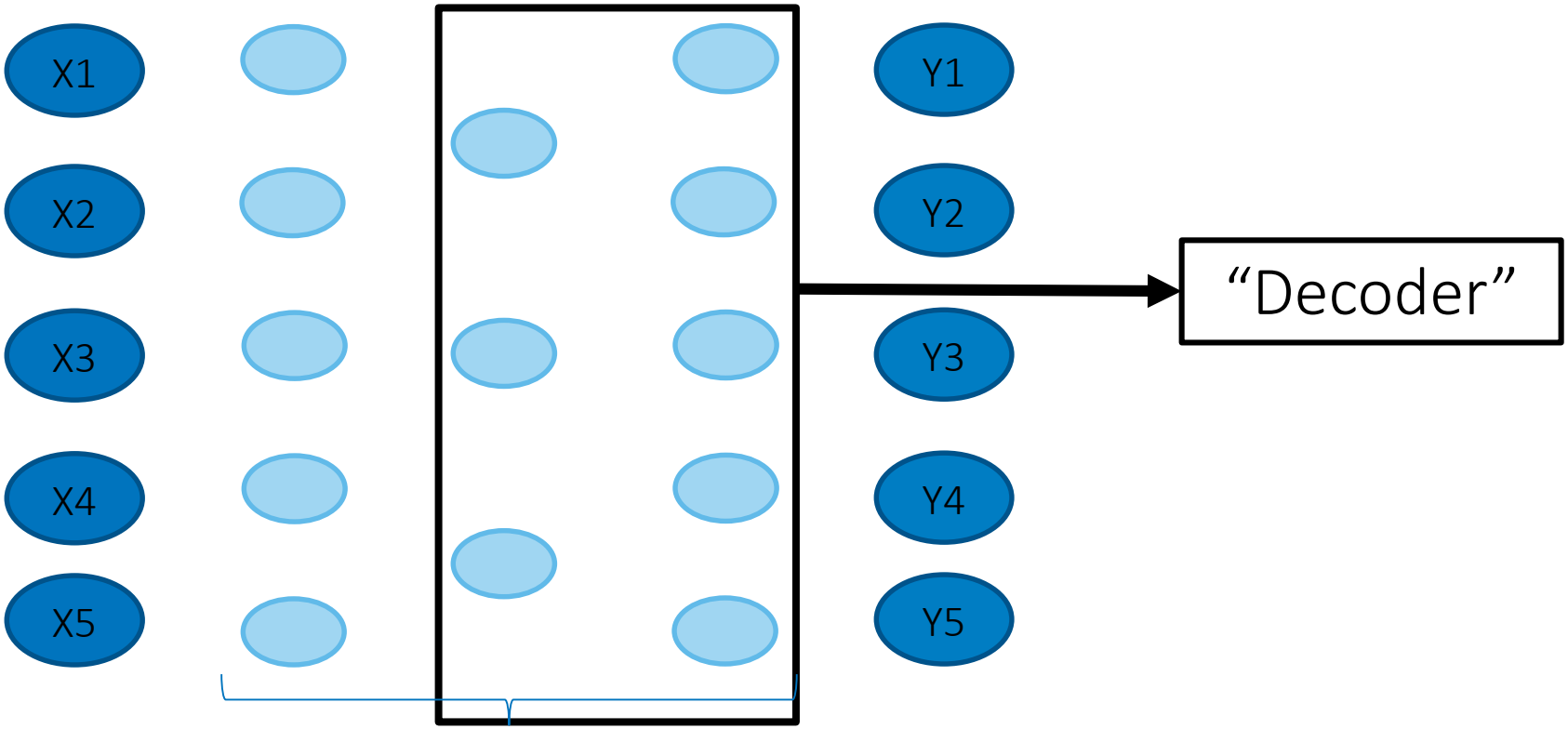
Middle hidden layer (“code” layer) used as new features

Autoencoder neural network



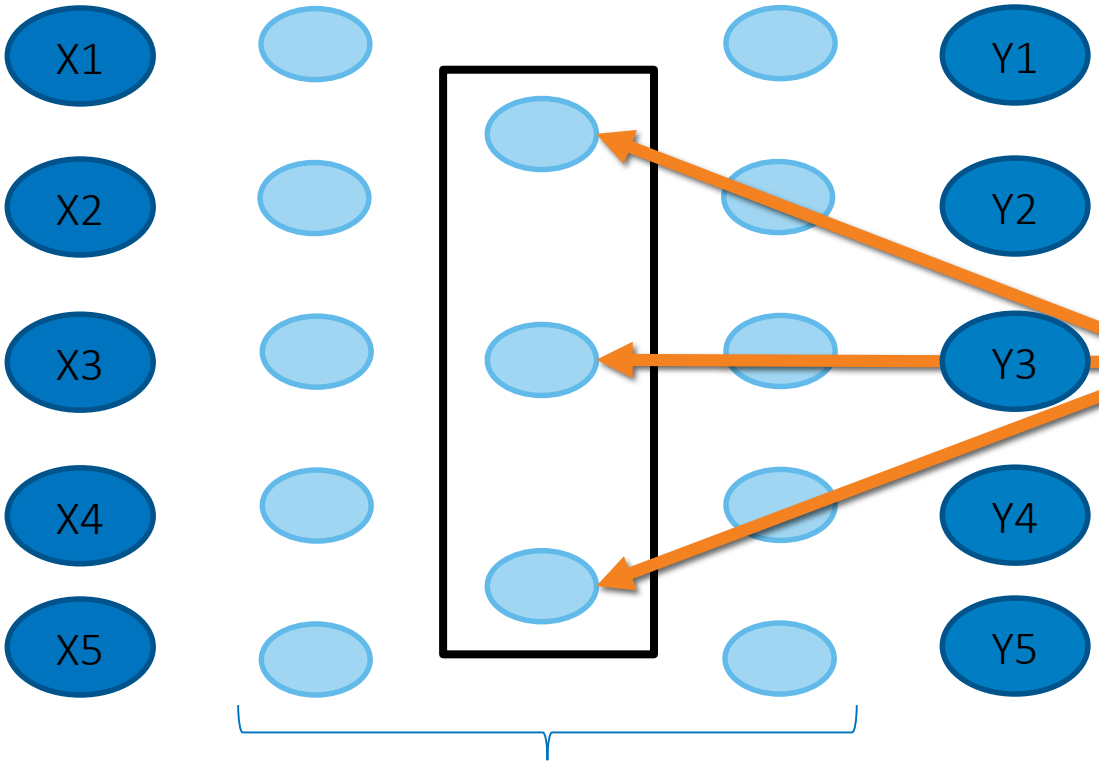
3 symmetric, hidden layers: 5 units, 3 units, 5 units

Autoencoder neural network



3 symmetric, hidden layers: 5 units, 3 units, 5 units

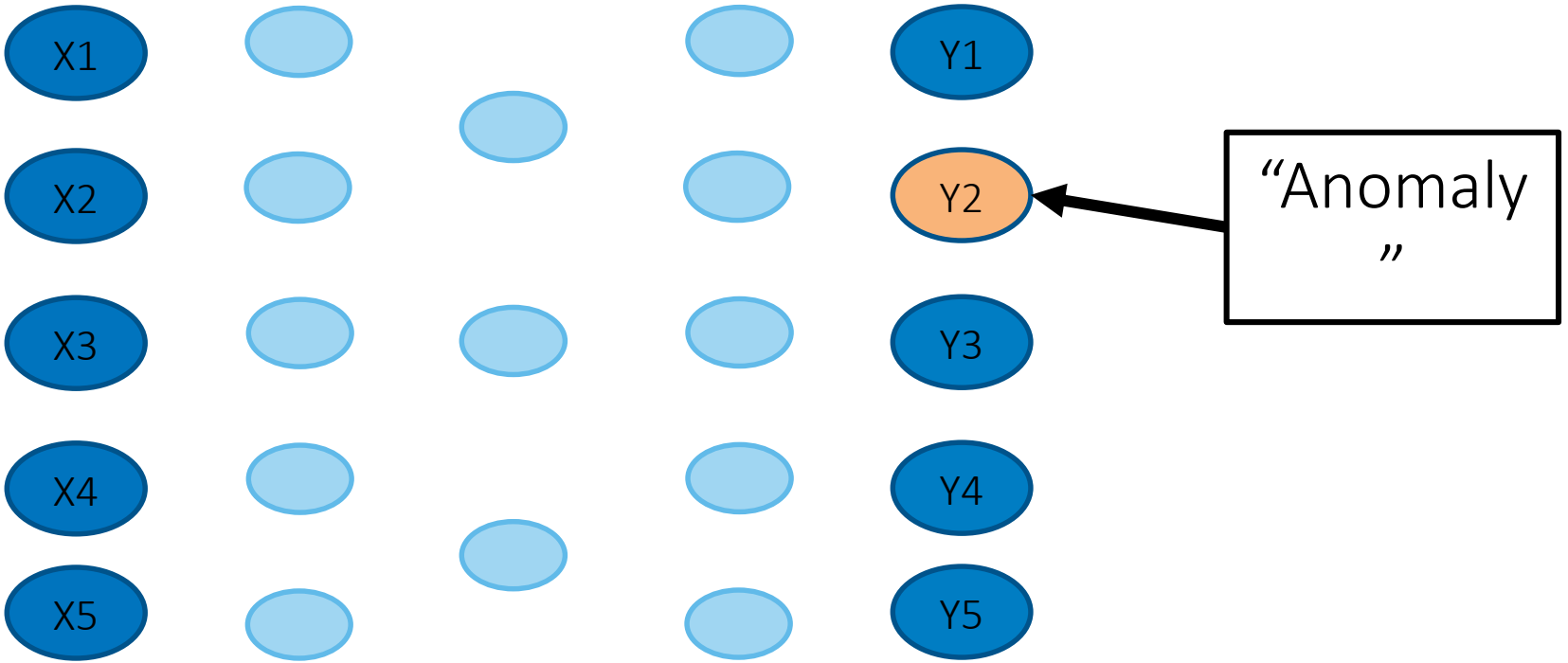
Autoencoder neural network



Units from the middle hidden layer ("code" layer) are our new features, like PCs

3 symmetric, hidden layers: 5 units, 3 units, 5 units

Autoencoder neural network



How to use NNET for autoencoders

Programmatic Approach using

SAS Studio / Code Node

Model Information	
Model	Neural Net
Number of Observations Used	1987
Number of Observations Read	1987
Number of Nodes	76
Number of Input Nodes	25
Number of Output Nodes	25
Number of Hidden Nodes	26
Number of Hidden Layers	3
Number of Weight Parameters	648
Number of Bias Parameters	51
Architecture	MLP (AUTOENCODER)
Seed for Initial Weight	802977670
Optimization Technique	LBFGS
Number of Neural Nets	1
Objective Value	0.7453511545

PROC NNET

Example:

```
proc nnet data=casuser.train;
    input cycle X1 - X24 / level=interval;
    hidden 12;
    hidden 2;
    hidden 12;
    train outmodel=casuser.enginennet;
    optimization regL2=0.1;
    score out=casuser.autoencoder;
    code file="/casuser/nnetscore.sas";
run;
```

Best Practices:

- Input layers = Output layers
- Use *odd* number of hidden layers (ex. 3, 5, 7, etc…)
- Center hidden layer should have *2 nodes*
- Create a bowtie / hourglass between input & output layers:
  - #nodes in layers between input and center should be decreasing
  - # nodes in layers between center and output should be increasing and match input to center pattern
- Deeper the network → more time is needed to train