AWS

MACHINE LEARNING

CERTIFICATION

DOMAIN #3: MODELING (36% EXAM)

AWS ML CERTIFICATION EXAM DOMAINS

Domain	% of Examination
Domain 1: Data Engineering	20%
Domain 2: Exploratory Data Analysis	24%
Domain 3: Modeling	36%
Domain 4: Machine Learning Implementation and Operations	20%
TOTAL	100%

Source: https://d1.awsstatic.com/training-and-certification/docs-ml/AWS%20Certified%20Machine%20Learning%20-%20Specialty_Exam%20Guide%20(1).pdf

DOMAIN #3 OVERVIEW:

SECTION #8: MACHINE AND DEEP LEARNING BASICS – PART #1

- Artificial Neural Networks Basics: Single Neuron Model
- Activation Functions
- Multi-Layer Perceptron Model
- How do Artificial Neural Networks Train?
- ANN Parameters Tuning – Learning rate and batch size
- Tensorflow playground
- Gradient Descent and Backpropagation
- Overfitting and Under fitting
- How to overcome overfitting?
- Bias Variance Trade-off
- L1 Regularization
- L2 Regularization

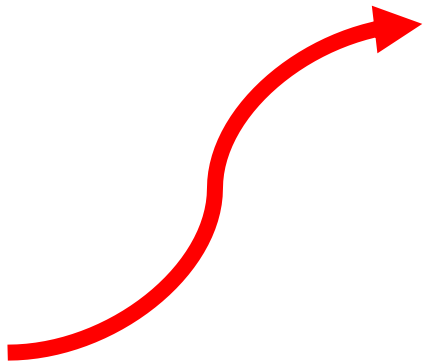
SECTION #9: MACHINE AND DEEP LEARNING BASICS – PART #2

- Artificial Neural Networks Architectures
- Convolutional Neural Networks
- Recurrent Neural Networks
- Vanishing Gradient Problem
- LSTM Networks
- Model Performance Assessment – Confusion Matrix
- Model Performance Assessment – Precision, recall, F1-score
- Model Performance Assessment – ROC, AUC, Heatmap, and RMSE
- K-Fold Cross validation
- Transfer Learning
- Ensemble Learning – Bagging and Boosting

DOMAIN #3 OVERVIEW:

SECTION #10: MACHINE AND DEEP LEARNING IN AWS – BUILT-IN ALGORITHMS PART #1

- AWS SageMaker
- Deep Learning on AWS
- SageMaker Built-in algorithms
- Object Detection
- Image Classification
- Semantic Segmentation
- SageMaker Linear Learner
- Factorization Machines
- XG-Boost
- SageMaker Seq2Seq
- SageMaker DeepAR
- SageMaker Blazing Text



WE ARE HERE!

SECTION #11: MACHINE AND DEEP LEARNING IN AWS – BUILT-IN ALGORITHMS PART #2

- Random Cut Forest
- Neural Topic Model
- LDA
- K-Nearest Neighbours (KNN)
- K Means
- Principal Component Analysis (PCA)
- IP insights
- Reinforcement Learning
- Object2Vec
- Automatic Model Tuning

DOMAIN #3 OVERVIEW:

SECTION #12: MACHINE AND DEEP LEARNING IN AWS – HIGH LEVEL AI/ML PART #3

- ReKognition
- Amazon Comprehend and Comprehend Medical
- Translate
- Transcribe
- Polly
- Forecast
- Lex
- Personalize
- Textract
- AWS DeepLens
- AWS DeepRacer

RECALL OUR MINI CHALLENGE AND PRIZE!

- For those of you who will successfully complete the entire section and watch the videos till the end, they will receive a valuable prize!

https://pixabay.com/illustrations/trophy-win-prize-transparent-1414791/

AWS SAGEMAKER

AWS SAGEMAKER BUILT-IN ALGORITHMS

SAGEMAKER AVAILABLE ALGORITHMS

BlazingText Word2Vec	BlazingText implementation of the Word2Vec algorithm for scaling and accelerating the generation of word embeddings from a large number of documents.
DeepAR	An algorithm that generates accurate forecasts by learning patterns from many related time-series using recurrent neural networks (RNN).
Factorization Machines	A model with the ability to the estimate all of the interactions between features even with a very small amount of data.
Gradient Boosted Trees (XGBoost)	Short for “Extreme Gradient Boosting”, XGBoost is an optimized distributed gradient boosting library.
Image Classification (ResNet)	A popular neural network for developing image classification systems.
IP Insights	An algorithm to detect malicious users or learn to usage patterns of IP addresses.
K-Means Clustering	One of the simplest ML algorithms. It’s used to find groups within unlabeled data.
K-Nearest Neighbor (k-NN)	An index based algorithm to address classification and regression based problems.
Latent Dirichlet Allocation (LDA)	A model that is well suited to automatically discovering the main topics present in a set of text files.
Linear Learner (Classification)	Linear classification uses an object’s characteristics to identify the appropriate group that it belongs to.
Linear Learner (Regression)	Linear regression is used to predict the linear relationship between two variables.
Neural Topic Modelling (NTM)	A neural network based approach for learning topics from text and image datasets.
Object2Vec	A neural-embedding algorithm to compute nearest neighbors and to visualize natural clusters.
Object Detection	Detects, classifies, and places bounding boxes around multiple objects in an image.
Principal Component Analysis (PCA)	Often used in data pre-processing, this algorithm takes a table or matrix of many features and reduces it to a smaller number of representative features.
Random Cut Forest	An unsupervised machine learning algorithm for anomaly detection.
Semantic Segmentation	Partitions an image to identify places of interest by assigning a label to the individual pixels of the image.
Seqence2Sequence	A general-purpose encoder-decoder for text that is often used for machine translation, text summarization, etc.

RECALL WHAT IS AN ALGORITHM? AND A HEURISTIC?

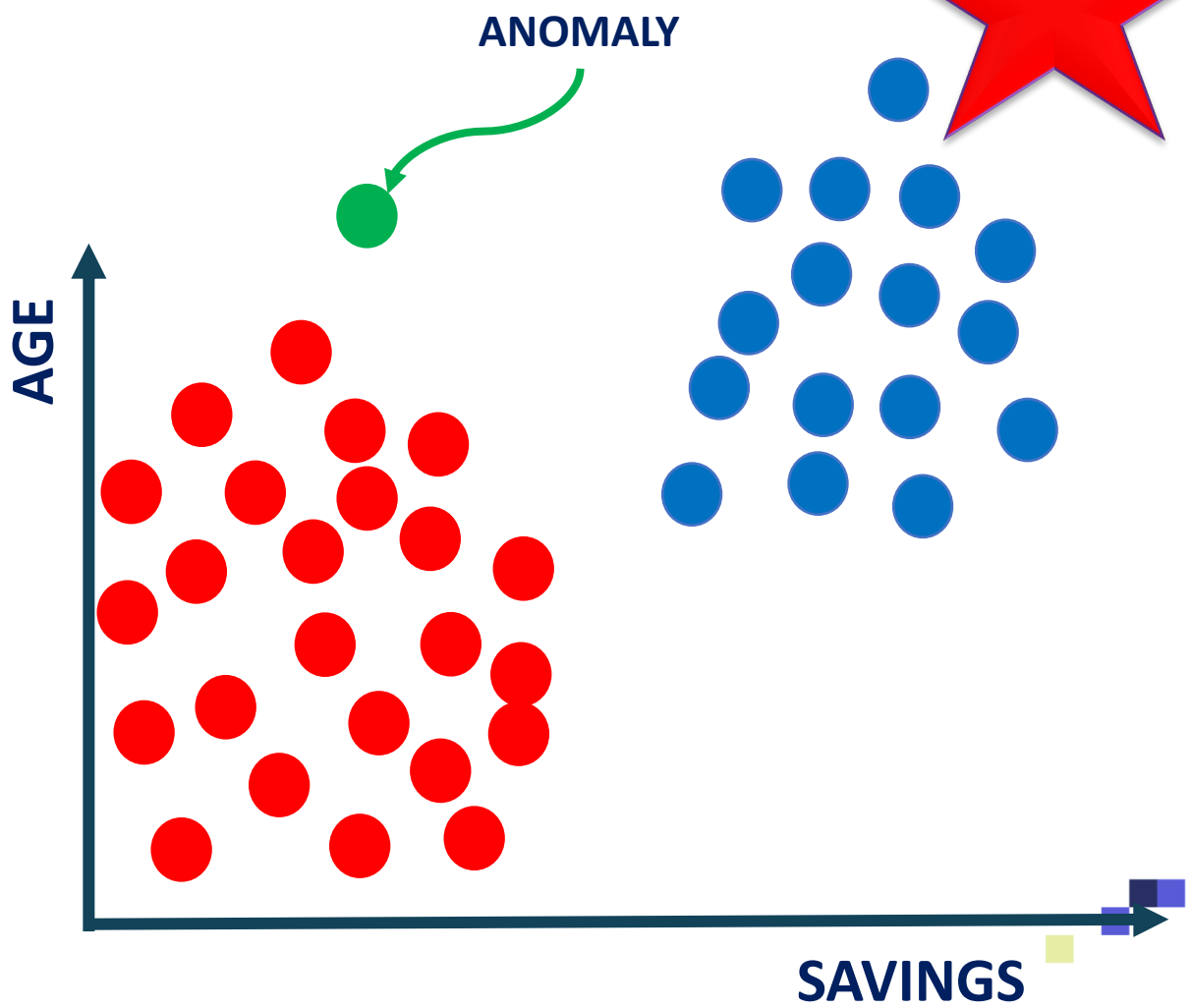
An Algorithm consists of multiple well defined steps to solve a given problem. An important feature of an algorithm is that the outcome should be repeatable across multiple runs.

A heuristic: is an “educated guess”, it represents an acceptable solutions that’s generated much faster by applying “rule of thumb”. Heuristics do not guarantee a consistent outcome.

RANDOM CUT FOREST

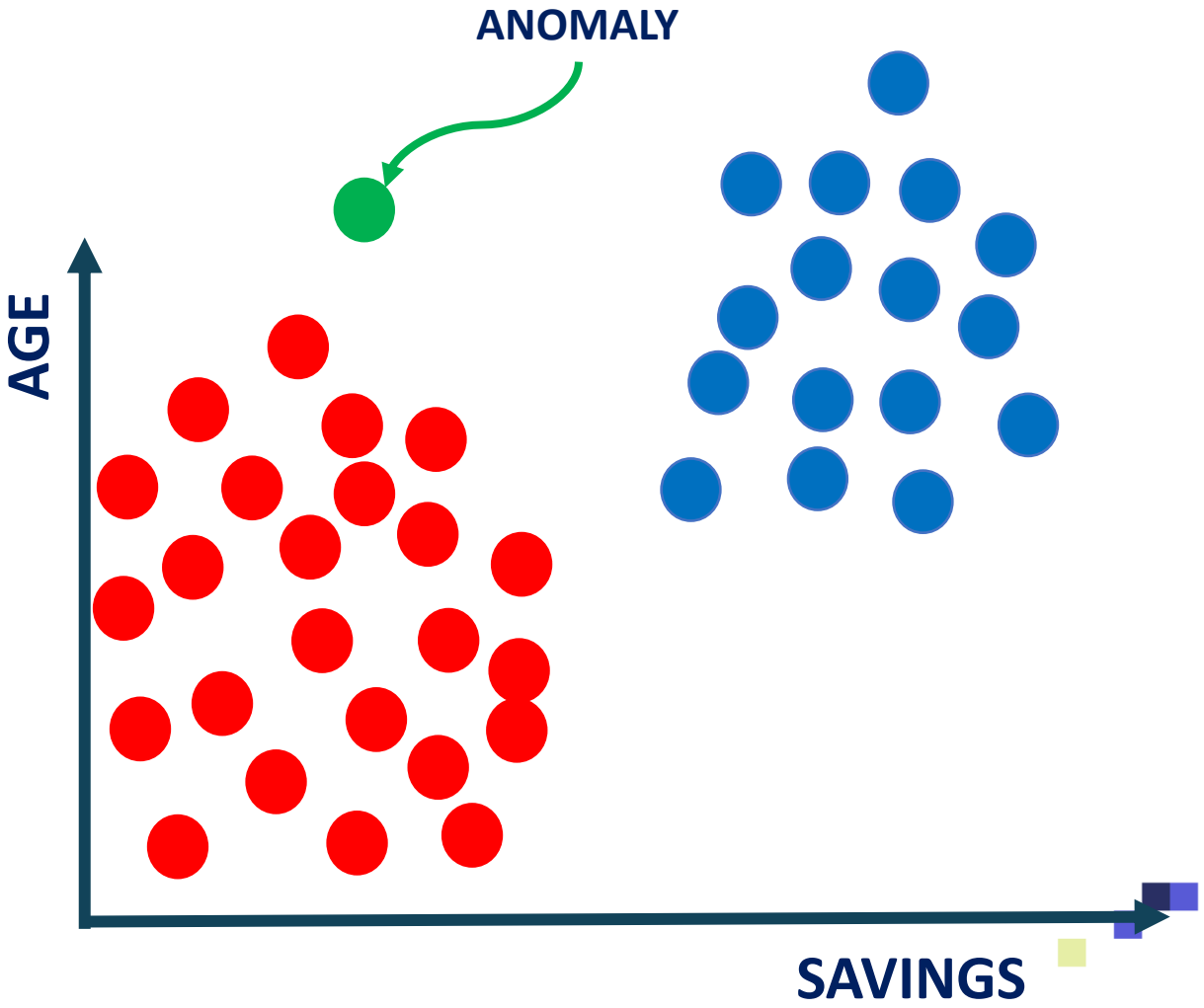
RANDOM CUT FOREST (RCF): OVERVIEW

- Random Cut Forest algorithm will be on the exam!!
- Amazon SageMaker Random Cut Forest (RCF) is an unsupervised algorithm
- It could be used for anomaly detection.
- Anomalies are data points that diverge from the rest of the properly structured data.
- Anomalies could be spikes or breaks in the dataset and could be detected from the regular structured dataset.
- Anomaly detection and removal is crucial in machine learning because adding anomalies will unnecessarily increase the complexity of the model.



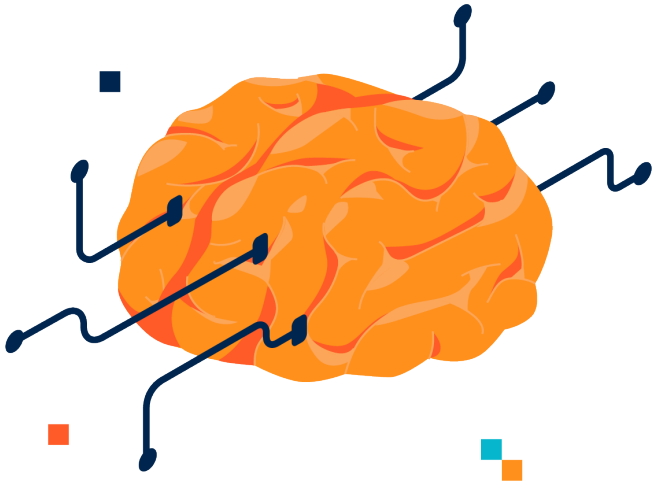
RANDOM CUT FOREST (RCF): OVERVIEW

- RCF algorithm assigns an anomaly score to each and every point in the dataset.
- If the score is more than 3 standard deviations, the anomaly score is considered high.
- The algorithm can work well with one and multi dimensional time series data.
- The algorithm works by sampling data randomly.
- The algorithm works by creating a forest of trees where each tree is a partition of the training data.
- The algorithms then examines the expected change in complexity of the tree after a point has been added to it.



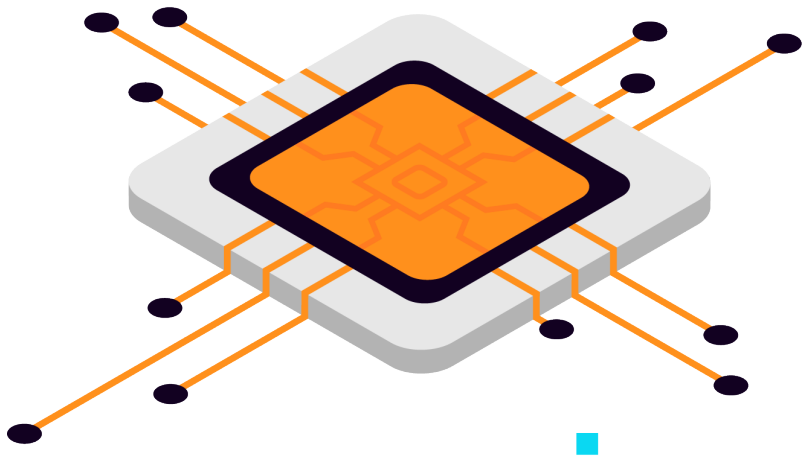
RANDOM CUT FOREST: INPUT/OUTPUT

- Amazon SageMaker Random Cut Forest require training channel and optional testing channels.
- Metrics such as accuracy, precision, recall, and F1-score can be calculated based on the testing dataset provided in the testing channel.
- The algorithm supports both File and Pipe modes
- Train and test data content types are:
  - application/x-recordio-protobuf
  - text/csv formats.
- During inference, RCF supports application/x-recordio-protobuf, text/csv and application/json.



RANDOM CUT FOREST: EC2 INSTANCE

- For RCF algorithm training, the following instances are recommended:
  - For training, ml.m4, ml.c4, and ml.c5 instances.
  - For inference, ml.c5.xl instance which will lead to maximum performance at an optimized cost.
- RCF could run on GPU instance types but it is not necessary.



RANDOM CUT FOREST: HYPERPARAMETERS

- feature_dim: number of features in the data set.
- eval_metrics: A list of metrics used to score a labeled test data set such as accuracy, positive and negative precision, recall, and F1-scores.
- num_samples_per_tree: Number of random samples given to each tree from the training data set.
- num_trees: Number of trees in the forest.

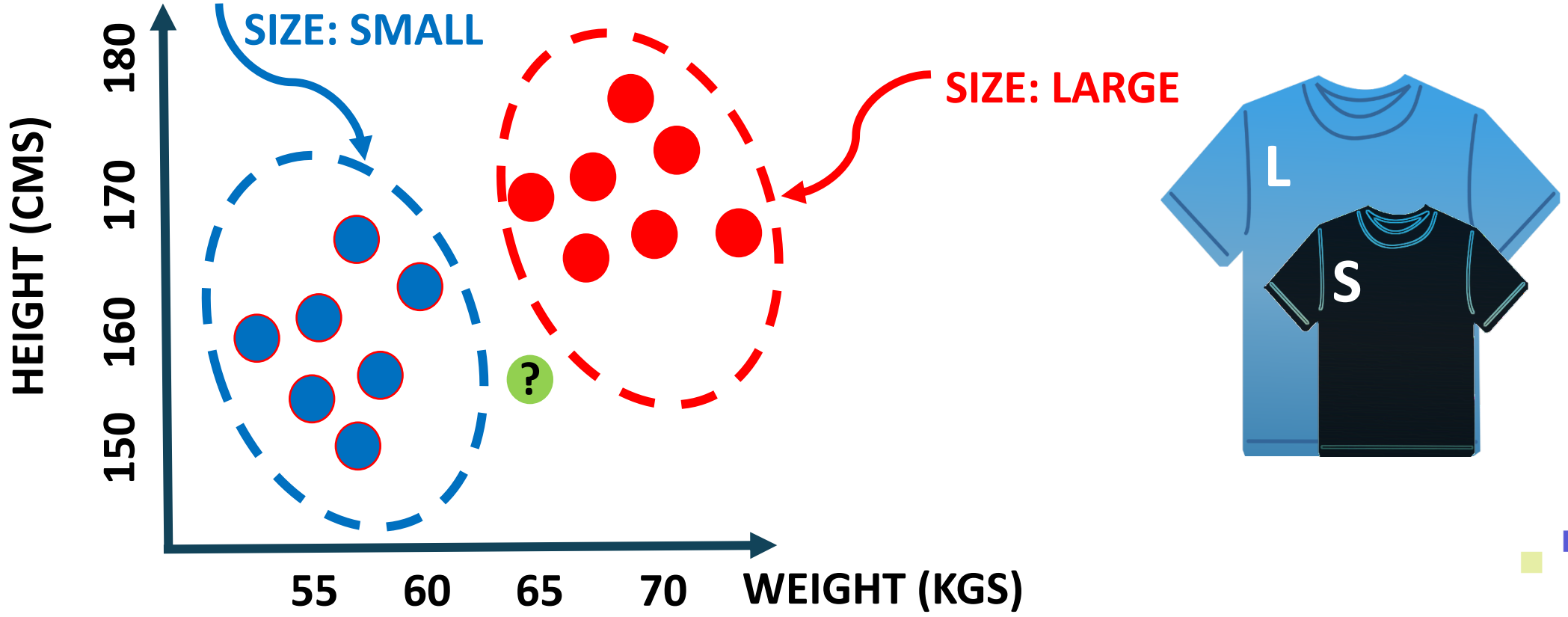
NOTES:

- num_trees and num_samples_per_tree are the most important parameters in the RCF algorithm.
- SageMaker recommends using 100 trees, this will strike a good balance between model complexity and noise.
- As the number of num_trees increases, the noise observed in anomaly scores is reduced because the final outcome is the average of all the output from all the trees.
- num_samples_per_tree should be chosen such that 1/num_samples_per_tree equals to the ratio of anomalous to normal data.
- Example: if in each tree, there is 256 samples then the data will contain 1/256 (0.4%) anomalies.

K NEAREST NEIGHBORS (KNN)

K NEAREST NEIGHBORS (KNN): INTUITION

- k-nearest neighbors algorithm (KNN) works by finding the **most similar** data points in the training data, and attempt to make an **educated guess** based on their classifications.

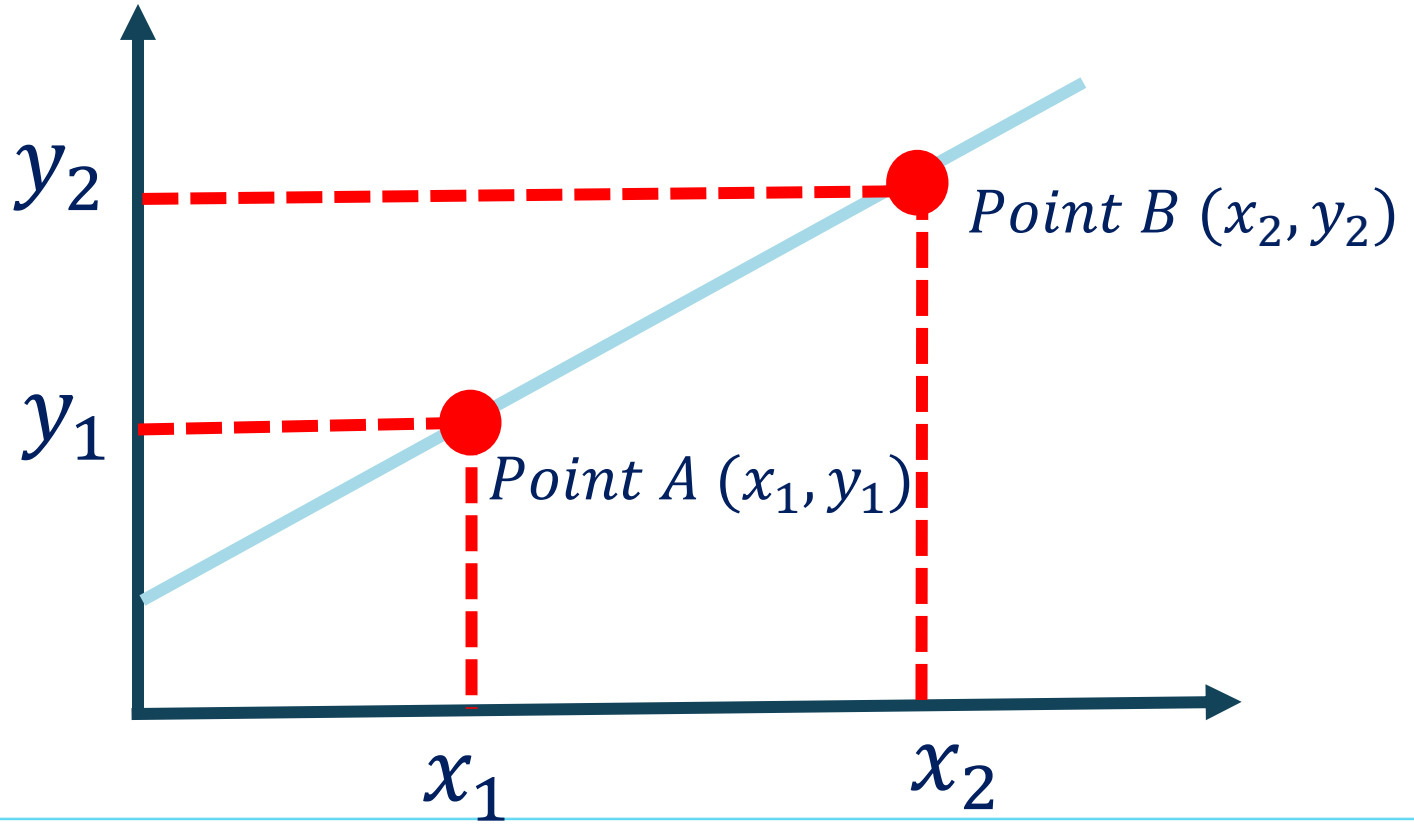


K NEAREST NEIGHBORS (KNN): ALGORITHM STEPS

- Select a value for k (e.g.: 1, 2, 3, 10..)
- Calculate the Euclidian distance between the point to be classified and every other point in the training data-set
- Pick the k closest data points (points with the k smallest distances)
- Run a majority vote among selected data points, the dominating classification is the winner! Point is classified based on the dominant class.
- Repeat!

EUCLIDEAN DISTANCE: INTUITION

$$\boldsymbol{Euclidean\ Distance} = \sqrt{(\boldsymbol{x_2 - x_1})^2 + (\boldsymbol{y_2 - y_1})^2}$$



K NEAREST NEIGHBORS (KNN): EXAMPLE

- KNN will look for the 5 data points that are closest to the new customer data point
- The algorithm will determine which category (class) are these 5 points in.
- Since 4 points had class ‘Small’ and 1 had ‘Large’, then new customer shall be assigned Small size.

Height (in cms)	Weight (in kgs)	T Shirt Size	Eucledian Distance	Vote
158	58	S	4.242640687	
158	59	S	3.605551275	
158	63	S	3.605551275	
160	59	S	2.236067977	3
160	60	S	1.414213562	1
163	60	S	2.236067977	3
163	61	S	2	2
160	64	L	3.16227766	5
163	64	L	3.605551275	
165	61	L	4	
165	62	L	4.123105626	
165	65	L	5.656854249	
168	62	L	7.071067812	
168	63	L	7.280109889	
168	66	L	8.602325267	
170	63	L	9.219544457	
170	64	L	9.486832981	
170	68	L	11.40175425	

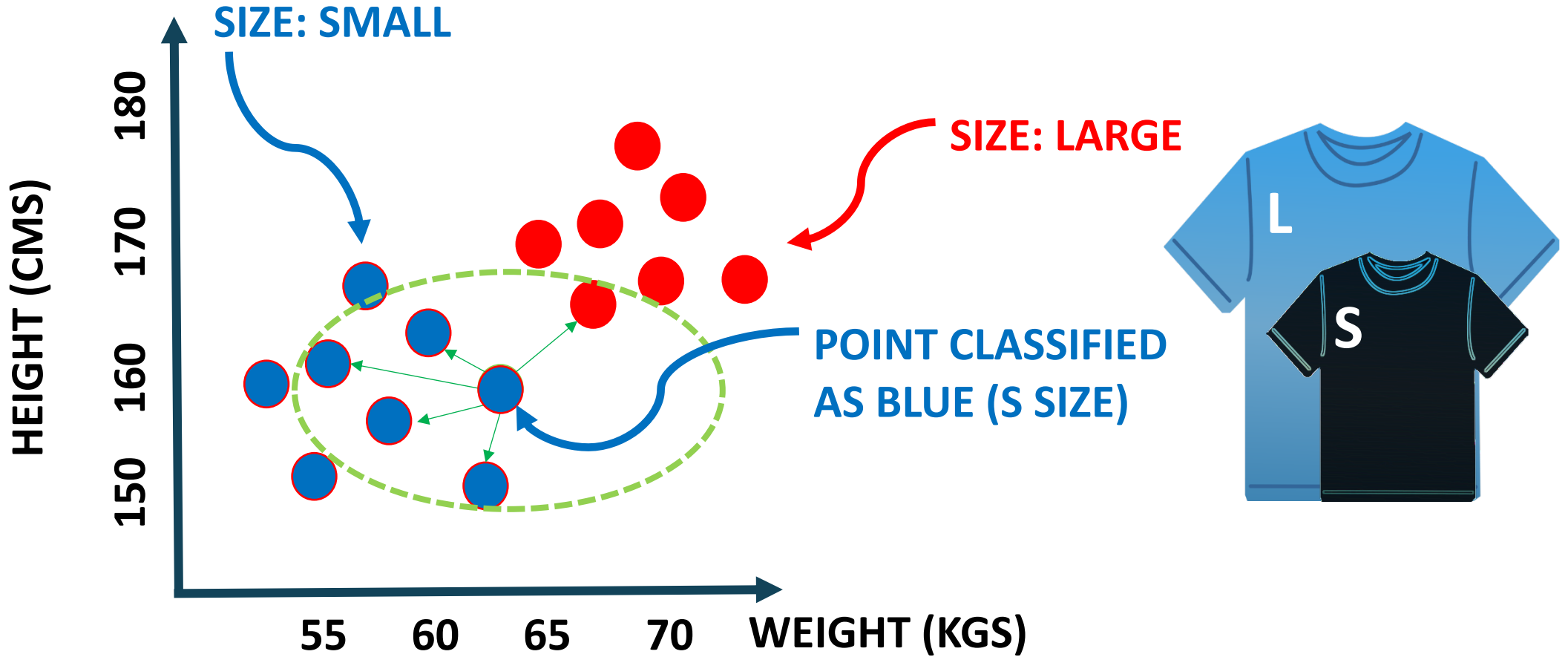
New Customer Information	
161	61

Assume k = 5

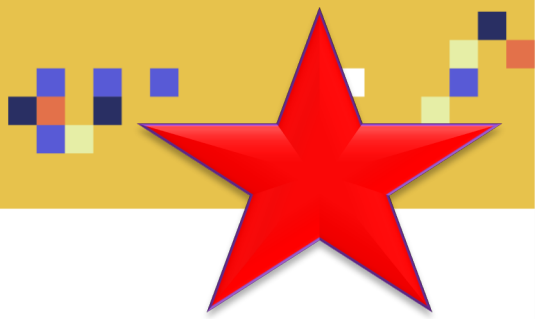
https://www.listendata.com/2017/12/k-nearest-neighbor-step-by-step-tutorial.html

K NEAREST NEIGHBORS (KNN): EXAMPLE

- Let's understand this example visually!



K NEAREST NEIGHBORS (KNN) IN SAGEMAKER



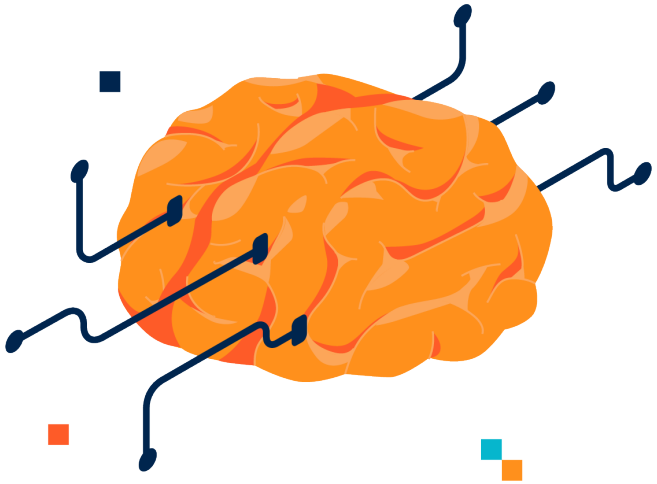
- KNN in SageMaker could be used to perform simple classification or regression
  - Classification: algorithm finds the K-closest points to a given sample point and return the most frequent label
  - Regression: algorithm finds K-closest points to a given sample point and return the average value.
- KNN is a lazy algorithm, it does not try to generalize the model for the entire training dataset but it rely on neighbouring data points.
- KNN is used for grouping some customers based on their credit risk or perform recommendations.
- Training with the KNN algorithm has three steps:
  - Sampling
  - Dimension reduction
  - Index building
- Sampling is used to minimize the size of dataset to optimize memory.
- Dimensionality reduction is performed to:
  - Decrease the feature dimension of the data to reduce the footprint of the k-NN model in memory and inference latency and avoids the “curse of dimensionality”

K NEAREST NEIGHBORS (KNN): HYPERPARAMETERS

- Full set of hyperparameters: https://docs.aws.amazon.com/sagemaker/latest/dg/kNN_hyperparameters.html
- K: The number of nearest neighbors
- Sample_size: The number of data points to be sampled from the training data set.
- feature_dim: The number of features in the input data.
- predictor_type: The type of inference to use on the data labels.
- dimension_reduction_target: The target dimension to reduce to.

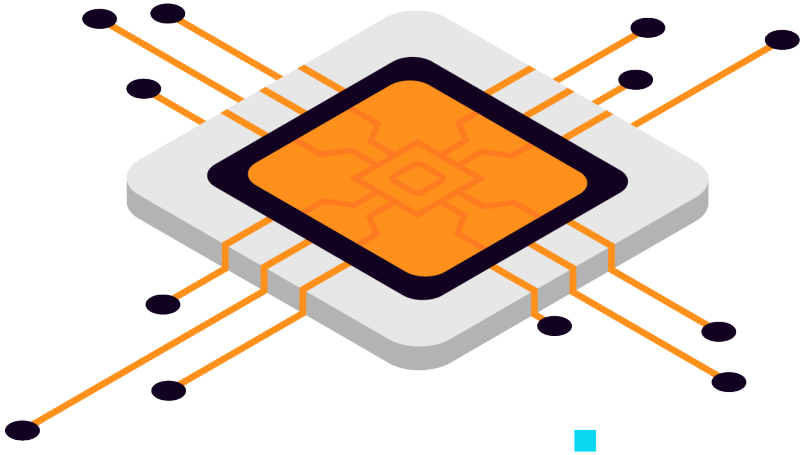
K NEAREST NEIGHBORS (KNN): INPUT/OUTPUT

- KNN supports two channels:
  - Train channel contains training data
  - Test channel to provide test scores such as accuracy for classifier or MSE for regressor
- SageMaker KNN algorithm supports recordIO-protobuf or CSV formats
- KNN can be used in both File or pipe mode



K NEAREST NEIGHBORS (KNN): INSTANCE TYPES

- For KNN Training:
  - CPU such as Ml.m5.2xlarge
  - GPU such as Ml.p2.xlarge
- For Inference:
  - GPU for higher throughput on large batches
  - CPU generally provides lower latency



K MEANS

K MEANS: OVERVIEW

- K-means is an unsupervised learning algorithm (clustering).
- K-means works by grouping some data points together (clustering) in an unsupervised fashion.
- Amazon SageMaker K-means algorithm scales for large amount of data and offers great computational efficiency.
- Amazon SageMaker offers these advantages by streaming mini-batches (small, random subsets) of the training data.
- The k-means algorithm relies on tabular data:
  - observations are put in rows
  - Attributes of the observations are put in columns. So in every row, these attributes demonstrates a single point in *n*-dimensional space.
- The algorithm groups observations with similar attribute values together by measuring the Euclidian distance between points.

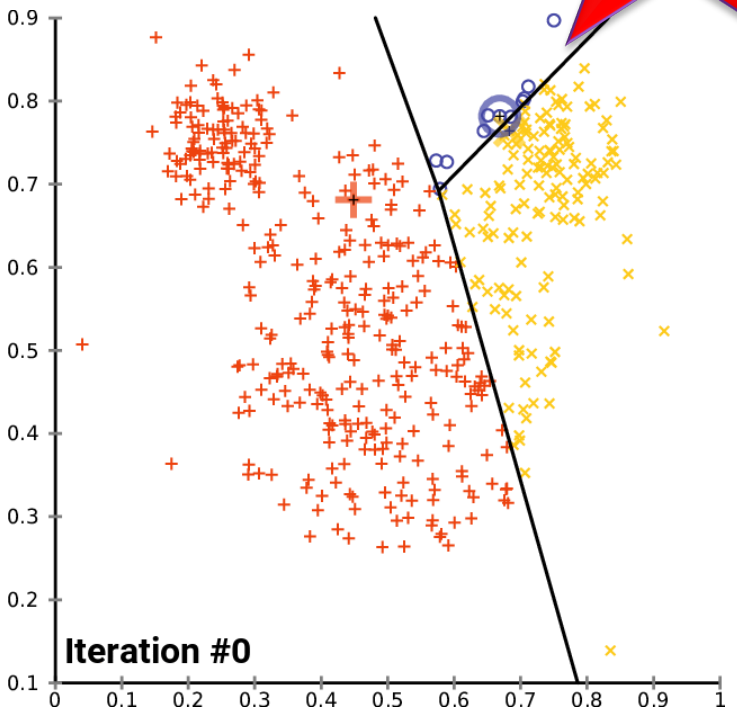


Photo Credit: https://en.wikipedia.org/wiki/K-means_clustering

K MEANS: OVERVIEW

- K-Means algorithm works as follows:
  - 1. K-Means determines the centers for the initial K cluster. This could be achieved randomly or using K-means++ which allows the clusters to be set away from each others.
  - 2. The algorithm iterates over input training data and recalculates cluster centers.
  - 3. The algorithm reduces resulting clusters to k.

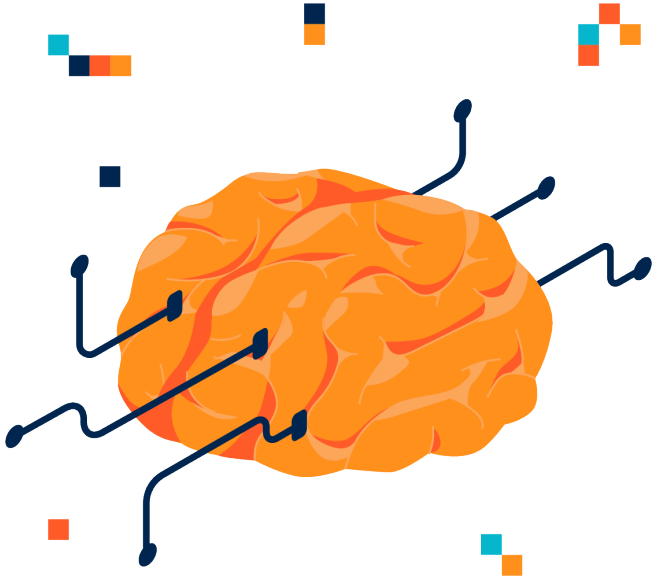
- Choice of K is critical in order to achieve great results. The optimal K is obtained using the "*elbow method*".

K MEANS: HYPERPARAMETERS

- Full set of hyperparameters: https://docs.aws.amazon.com/sagemaker/latest/dg/k-means-api-config.html
- feature_dim: The number of features in the input data.
- K: The number of clusters
- init_method: hyperparameter that determines how the K-mean algorithm will choose the initial cluster centers.
- eval_metrics: metrics to assess the performance of the model
- extra_center_factor

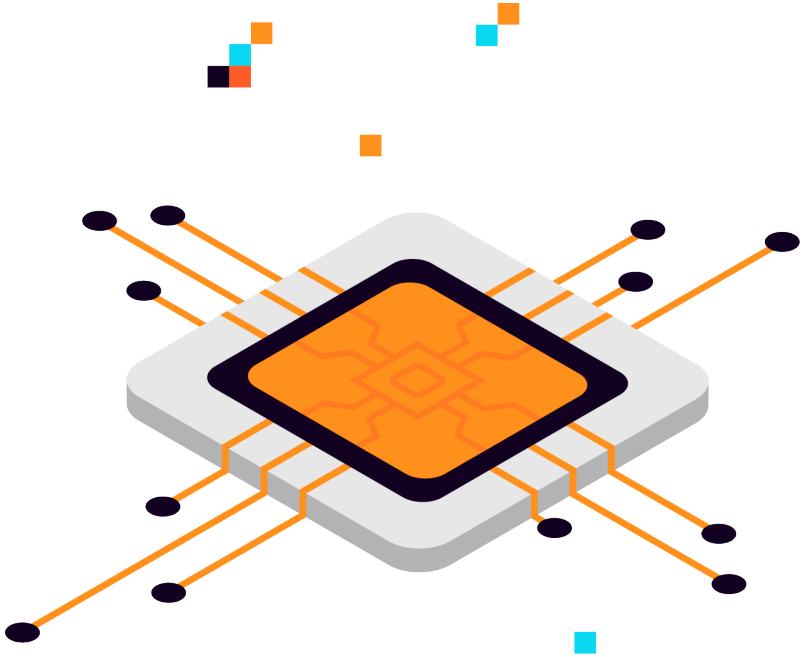
K MEANS: INPUT/OUTPUT

- K-Means supports two channels:
  - Train channel contains training data
  - Optional Test channel
- SageMaker K-Means algorithm supports recordIO-protobuf or CSV formats
- K-Means can be used in both File or pipe mode



K MEANS: EC2 INSTANCE

- For K-Means Training:
  - CPU instance is recommended
  - Technique could work on GPU as well but only one GPU per instance used on GPU
  - p*.xlarge recommended in case of using GPU



PRINCIPAL COMPONENT ANALYSIS (PCA)

PRINCIPAL COMPONENT ANALYSIS: OVERVIEW

- PCA is an unsupervised machine learning algorithm.
- PCA performs dimensionality reductions while attempting at keeping the original information unchanged.
- PCA works by trying to find a new set of features called components.
- Components are composites of the uncorrelated given input features.
- The first component accounts for largest data variability followed by second component and so on.

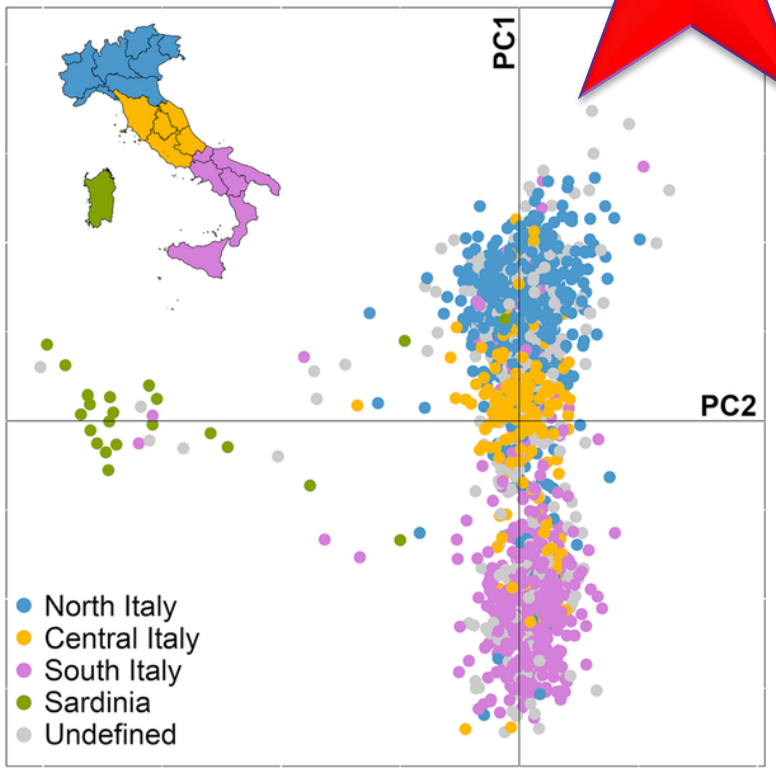


Photo Credit: https://commons.wikimedia.org/wiki/File:Principal_Component_Analysis_of_the_Italian_population.png

PRINCIPAL COMPONENT ANALYSIS: OVERVIEW

- PCA (Principal Component Analysis) performs dimensionality reduction as shown below.
- The algorithm generates the principal components by calculating the covariance matrix first and then doing singular value decomposition.
- In Amazon SageMaker PCA operates in two modes:
  - Regular: works well with sparse data small (manageable) number of observations/features.
  - Randomized: works well with large number of observations/features.

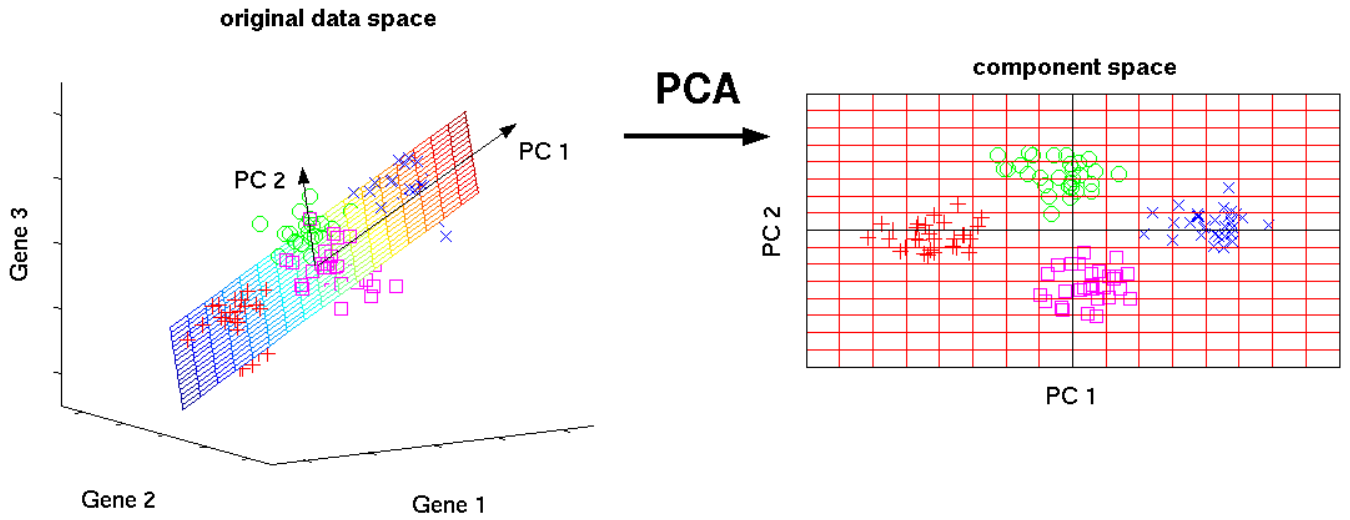


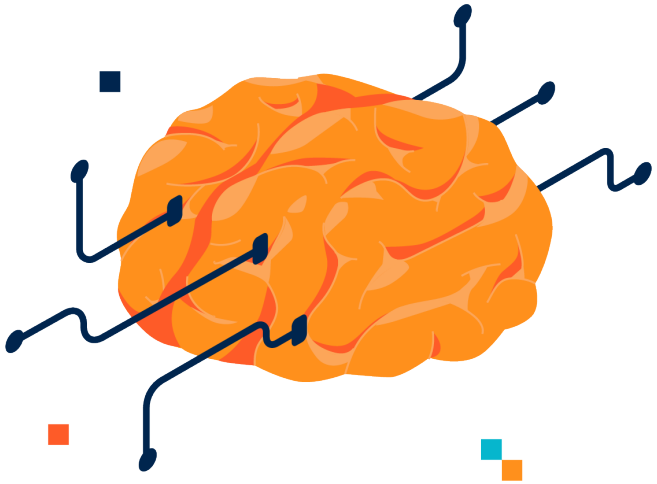
Photo Credit: http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de/

PRINCIPAL COMPONENT ANALYSIS: HYPERPARAMETERS

- Full set of hyperparameters: https://docs.aws.amazon.com/sagemaker/latest/dg/PCA-reference.html
- feature_dim: number of features in the input data.
- num_components: number of principal components to compute.
- algorithm_mode: Mode for computing the principal components, choose between regular or randomized
- extra_components: As extra componenets go up, more accurate results are achieved at the cost of increased memory/computation consumption.
- subtract_mean: True or false Boolean to dictate whether data should be unbiased or not.

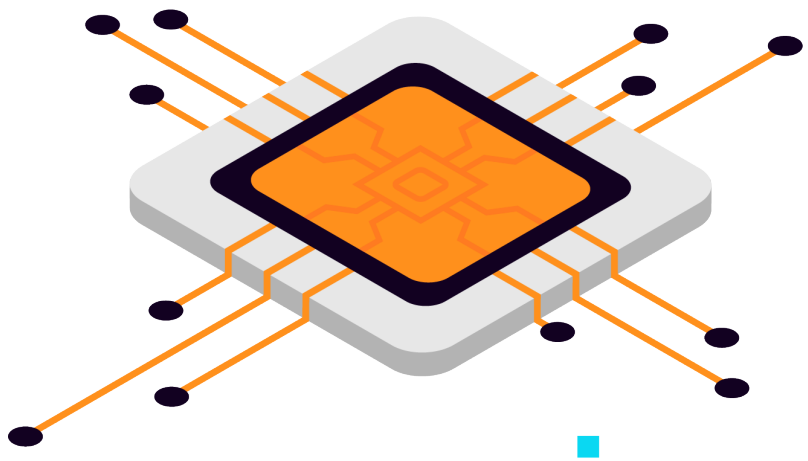
PRINCIPAL COMPONENT ANALYSIS: INPUT/OUTPUT

- SageMaker PCA algorithm supports recordIO-protobuf or CSV formats
- PCA can be used in both File or pipe mode



PRINCIPAL COMPONENT ANALYSIS: INSTANCE TYPES

- For PCA Training:
  - CPU instance or GPU are recommended



IP INSIGHTS

IP INSIGHTS: OVERVIEW

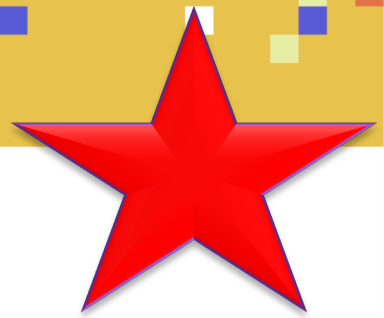
- Amazon SageMaker IP Insights is an unsupervised learning algorithm.
- The algorithm works by learning the usage patterns for various IPv4 addresses.
- The algorithm learns the relationship between various entities and IPv4 addresses.
- IP insights is useful for fraud detection to detect anomalous logins from foreign (unusal) IPv4 addresses.
- Trained IP Insight models can be hosted at an endpoint and works in real-time.

IPv4 address in dotted-decimal notation

172 . 16 . 254 . 1

Photo Credit: https://en.wikipedia.org/wiki/File:IP_address_icon.png
https://commons.wikimedia.org/wiki/File:Ipv4_address.svg

IP INSIGHTS: OVERVIEW



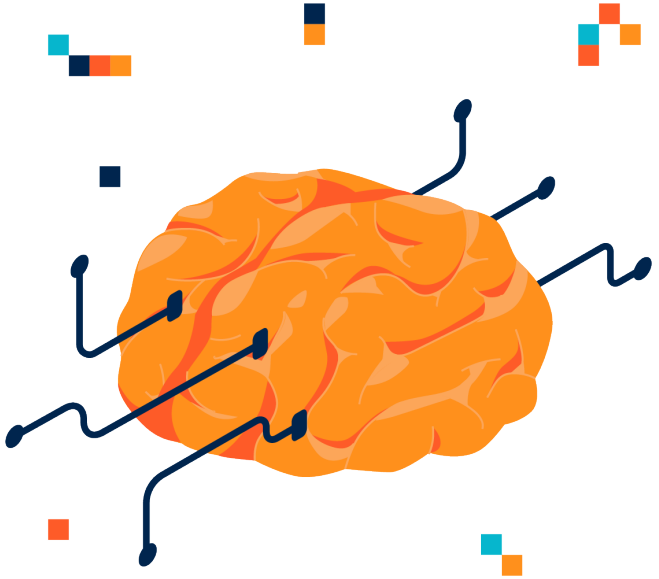
- When you send an (entity, IPv4 Address) event to a trained IP insights model, it provides a score that indicates if the event is ‘anomalous’ or ‘safe’.
- If the event is anomalous, a second factor authentication might be triggered to verify user identity.
- More advanced security systems could be developed by combining IP insight’s generated scores with other features to rank the findings of another security system (Amazon GuardDuty).
- IP Insights is capable of learning embeddings which consists of vector representations of IP addresses.
- Those embeddings can be used as features and being fed into another machine learning model.
- Under the hood, IP insights uses a neural network to learn latent vector representations of both IPv4 address and entities.
- IP insights work by hashing and embedding Entities
- In order to increase datasets during training, the algorithm generates negative samples by randomly pairing entities and IP’s

IP INSIGHTS: HYPERPARAMETERS

- Full set of hyperparameters: https://docs.aws.amazon.com/sagemaker/latest/dg/ip-insights-hyperparameters.html
- num_entity_vectors: indicates the number of entity vector representations to train.
- vector_dim: size of embedding vectors to represent entities and IP addresses.
- learning_rate
- Epochs
- num_ip_encoder_layers: The number of fully connected layers used to encode the IP address embedding.

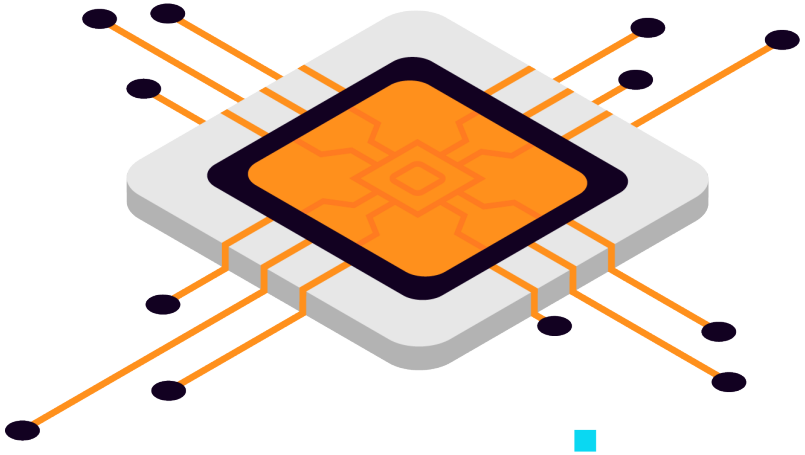
IP INSIGHTS : INPUT/OUTPUT

- IP Insights expects two channels: (1) training and (2) validation (optional).
- Training and validation channels must be in text/csv format.
  - CSV File Column #1: holds a unique identifier for the entity.
  - CSV File Column #2: holds the IPv4 address.
- The validation channel is used to assess the performance of the algorithm and see how good is it in discriminating between positive and negative samples.
- An important metric for performance assessment is the area-under-curve (AUC) score on a predefined negative sampling strategy.
- IP Insights currently supports only File mode.
- For Inference, IP Insights supports text/csv, application/json, and application/jsonlines data content types.



IP INSIGHTS: INSTANCE TYPES

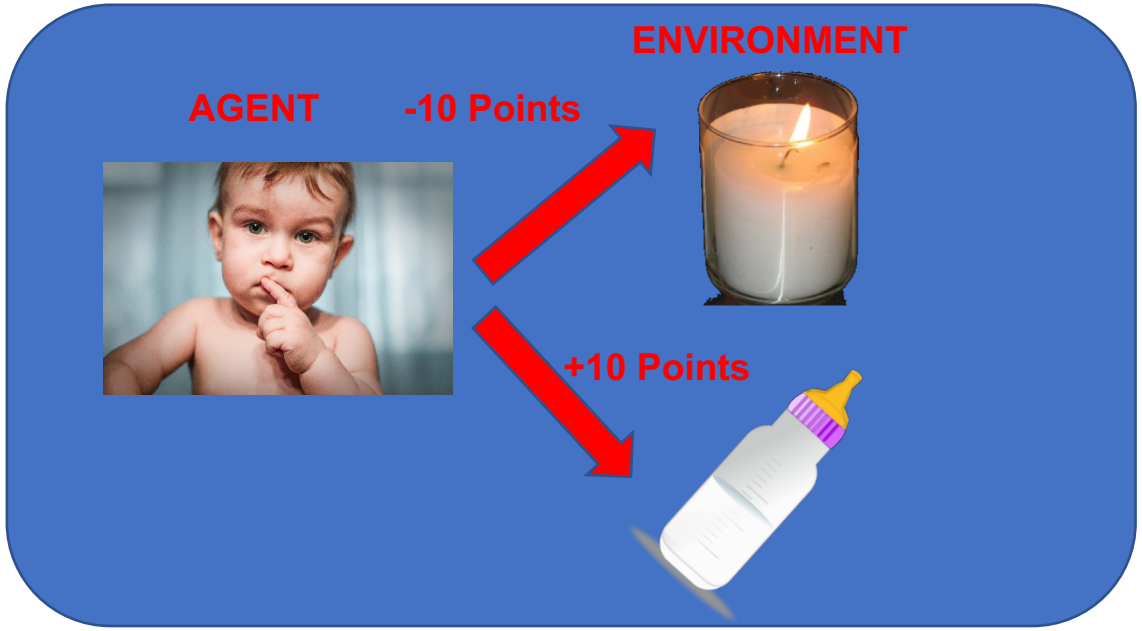
- In order to train the IP insights, a GPU instance is recommended
  - Ml.p3.2xlarge or higher
  - Size of CPU instance depends on vector_dim and num_entity_vectors
  - IP insights can rely on multiple GPUs

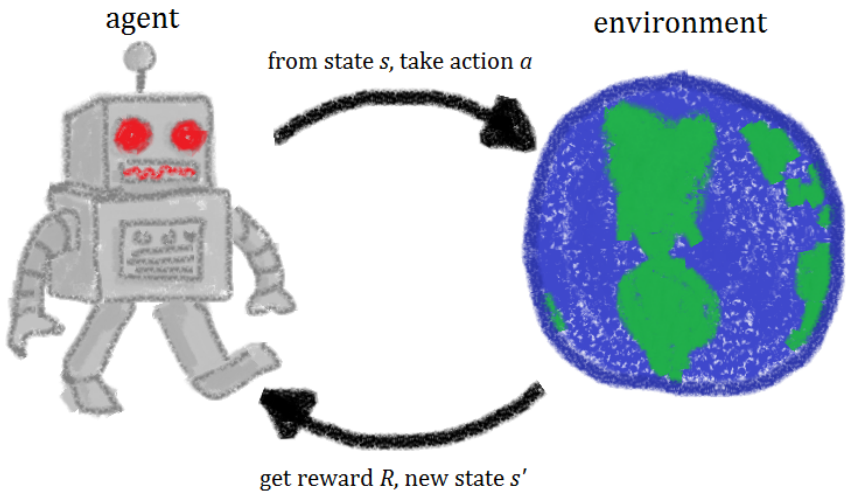


REINFORCEMENT LEARNING

MACHINE LEARNING: REINFORCEMENT LEARNING

- Reinforcement learning allows machines take actions to maximize cumulative reward.
- Reinforcement algorithms learn by trial and error through reward and penalty.
- Two elements: **environment** and **learning agent**.
- The environment rewards the agent for correct actions.
- Based on the reward or penalty, agent improves its environment knowledge to make better decision.





https://commons.wikimedia.org/wiki/File:Rl_agent.png

REINFORCEMENT LEARNING: OVERVIEW

- Reinforcement learning (RL) is a machine learning technique that works by learning a policy or strategy through trial and error.
- In RL, an agent is present in an environment and its objective is to maximize cumulative reward.
- In RL, the following steps are performed by the agent:
  - Takes an action
  - Observes the state of the environment
  - Receives a reward or penalty
- The goal is to maximize the cumulative long-term reward.
- Example:
  - Agent: Robot
  - Environment: maze
  - Objective: Navigate the maze in the shortest time possible
- After the robot is trained, it can make decisions on its own

REINFORCEMENT LEARNING: WHY IS REINFORCEMENT LEARNING IMPORTANT?

- RL works well with complex large scale tasks such as self driving cars, supply chain management, game artificial intelligence, industrial robotics.
- Since RL models learn by trying to maximize the reward and reduce the penalty, it can be trained to make decisions in dynamic environments and under extreme uncertainty.

REINFORCEMENT LEARNING : MARKOV DECISION PROCESS (MDP)

- Reinforcement learning implements Markov Decision Processes (MDPs) model.
- RL training consists of a series of time steps in an MDP starting from an initial state until final state.
- MDP works through a series of time steps that include the following:
  - **Environment:** includes the space in which the RL model operates which could be simulation or real world environments.
  - **State:** indicates the data related to the past actions that the RL model has taken. The state is important in determining the actions that the RL model will take in the future. For example: in case of autonomous cars, the RL model state is the position of the vehicle.
  - **Action:** action taken by the agent such as autonomous vehicle making a right turn.
  - **Reward:** the number that the agent takes after taking the last action. Remember that the agent is trying to maximize the cumulative reward. The RL Model is trying to find the optimal policy or strategy to maximize the reward and avoid penalties.
  - **Observation:** data related to the environment state. It could be visible or partially visible to the agent.
    - Example #1 chess: full state of the board is visible to agent.
    - Example #2: robot in a maze can only view a small portion of the maze.

REINFORCEMENT LEARNING: KEY FEATURES OF AMAZON SAGEMAKER RL

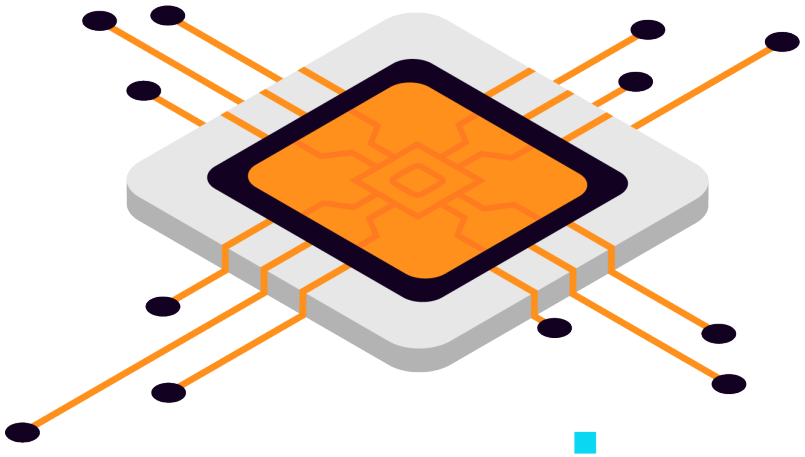
- The following components are available in RL SageMaker:
  - TensorFlow and Apache MXNet deep learning framework frameworks.
  - An RL toolkit:
    - ❖ Includes state of the art RL algorithms and allows for the proper communication between agent and environment.
    - ❖ Amazon SageMaker supports the Intel Coach and Ray RLlib toolkits.
  - RL environment: open source, custom or commercial environments are available.

REINFORCEMENT LEARNING: RL ENVIRONMENTS

- Amazon SageMaker RL offers environments that emulates real world.
- Simulators are crucial in some cases where driving (self driving cars) or flying (drones) in real life is dangerous
- The simulation environment consists of an agent and a simulator.
- Here are a list of options available:
  - Use OpenAI Gym Interface for Environments in Amazon SageMaker RL:
    - ❖Gym is a toolkit for developing and comparing reinforcement learning algorithms. Check out the website here: https://gym.openai.com/docs/
  - Use Open Source Environments: EnergyPlus and RoboSchool, in Amazon SageMaker RL by building your own container.
  - Use Commercial Environments: MATLAB and Simulink.
- Check this out for the full diagram of RL components supported by SageMaker: https://docs.aws.amazon.com/sagemaker/latest/dg/reinforcement-learning.html

REINFORCEMENT LEARNING: EC2 INSTANCES

- GPUs are recommended
- You can run RL on:
  - Multi-core and multi-instance
  - Can distribute training and/or environment



REINFORCEMENT LEARNING: HYPERPARAMETERS

- Hyperparameter tuning job is required in order to optimize hyperparameters for Amazon SageMaker RL.
- Check out this example for abstracting parameters to tune: https://github.com/awslabs/amazon-sagemaker-examples/tree/master/'reinforcement-learning

NEURAL TOPIC MODEL

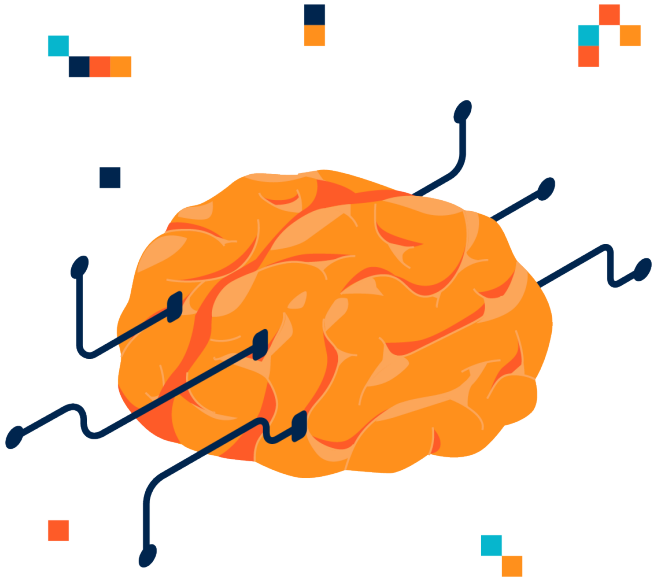
NEURAL TOPIC MODEL (NTM): OVERVIEW

- Neural Topic Model is an unsupervised learning algorithm
- NTM organizes a corpus of documents into topics.
- For example, the words hungry, bananas, chocolate, watermelons share the same topic of “food”
- Topic modeling can be used to classify/summarize documents based on the topics detected.
- Amazon SageMaker offers two algorithms (1) NTM and (2) LDA (to be presented shortly) to perform topic modeling. They could generate totally different results.
- Users of the algorithm need to specify how many topics they’d like to have. Topics are a latent representation based on top ranking words.

Photo Credit: http://www.peakpx.com/465710/vinyl-records-and-file-documents

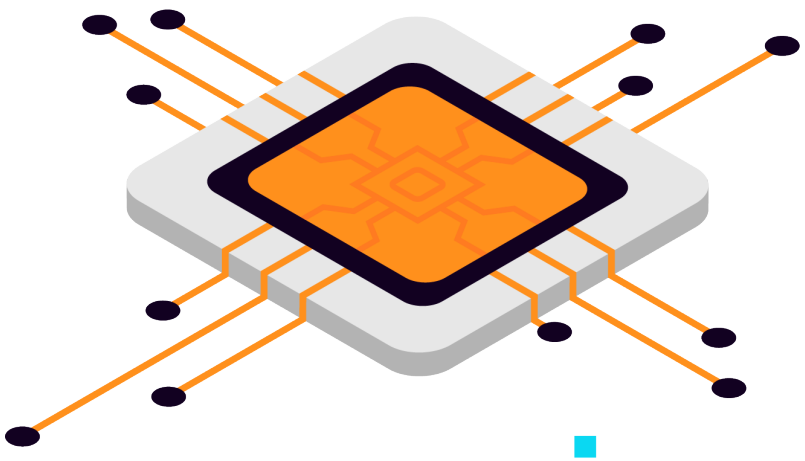
NEURAL TOPIC MODEL: INPUT/OUTPUT

- Amazon SageMaker Neural Topic Model supports four data channels as follows:
  - Train
  - Validation (optional)
  - Test (optional)
  - Auxiliary (optional)
- Supports the following formats:
  - recordIO-wrapped-protobuf (dense and sparse)
  - CSV file formats.
- File mode or Pipe mode could be used to train models on data in recordIO-wrapped-protobuf or CSV formats.



NEURAL TOPIC MODEL: EC2 INSTANCE

- NTM training supports both GPU and CPU instances.
  - GPU instances are recommended
  - For inference, CPU instances are sufficient.



NEURAL TOPIC MODEL: HYPERPARAMETERS

- **For full list of parameters:** https://docs.aws.amazon.com/sagemaker/latest/dg/ntm_hyperparameters.html
- **feature_dim:** vocabulary size of the dataset
- **num_topics:** number of required topics
- **encoder_layers:** number of layers in the encoder and the output size of each layer
- **encoder_layers_activation:** activation function to use in the encoder layers
- **sub_sample:** fraction of training data to sample for training per epoch
- **Epochs**
- **learning_rate**
- **Optimizer**

LDA

LDA: OVERVIEW

- The Amazon SageMaker LDA stands for Latent Dirichlet Allocation.
- The algorithm is unsupervised learning algorithm.
- LDA describes a set of observations as a mixture of distinct categories.
- LDA performs the same function as the Neural Topic algorithm but does not rely on deep learning so it's not computationally expensive (CPU instance would be enough).
- LDA takes in a text corpus and group them into user-specified number of topics

Photo Credit: http://www.peakpx.com/465710/vinyl-records-and-file-documents

LDA: OVERVIEW

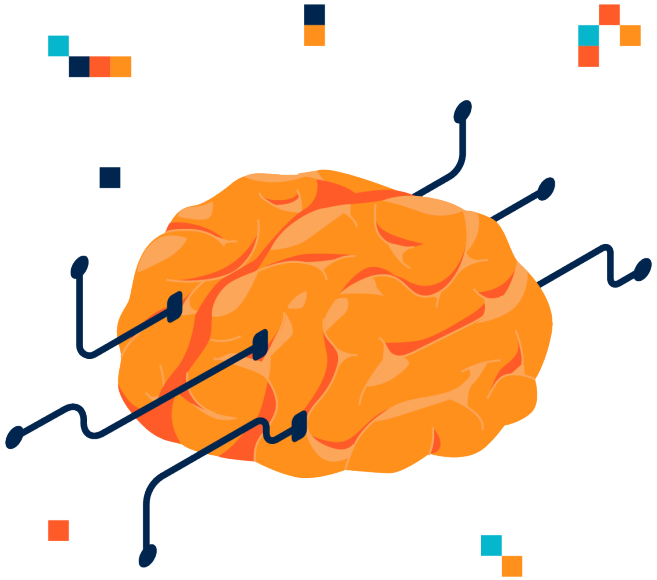
- LDA is an unsupervised algorithm so topics won't necessarily align with what humans could categorize them.
- Example: Let's assume that we have a set of documents and the only words that occur within them are: *eat*, *sleep*, *play*, *meow*, and *bark*
- LDA will generate the following table:

Topic	**eat**	**sleep**	**play**	**meow**	**bark**
Topic 1	0.1	0.3	0.2	0.4	0.0
Topic 2	0.2	0.1	0.4	0.0	0.3

- Topic 1 is probably about cats who meow and sleep
- Topic 2 is probably about dogs play and bark
- These topics can be found even though the words dog and cat never appear in any of the texts.

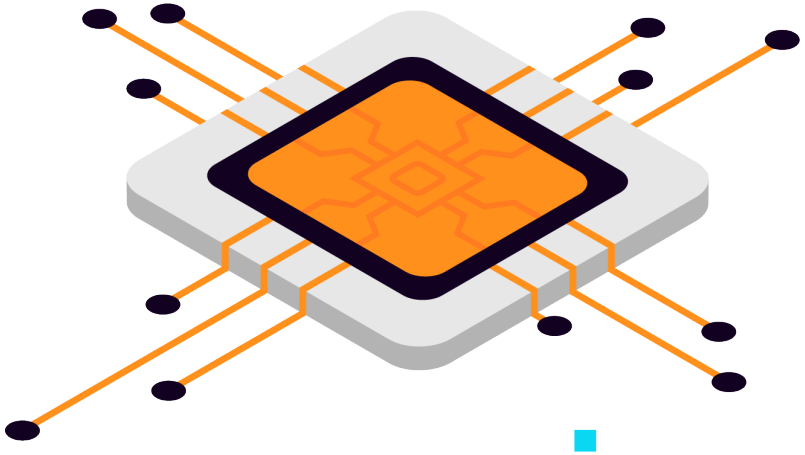
LDA: INPUT/OUTPUT

- Amazon SageMaker LDA supports:
  - Train channel
  - Test channel (optional)
- Supports the following formats:
  - recordIO-protobuf
  - CSV file formats.
- Pipe mode could be used to train models on data in recordIO format.



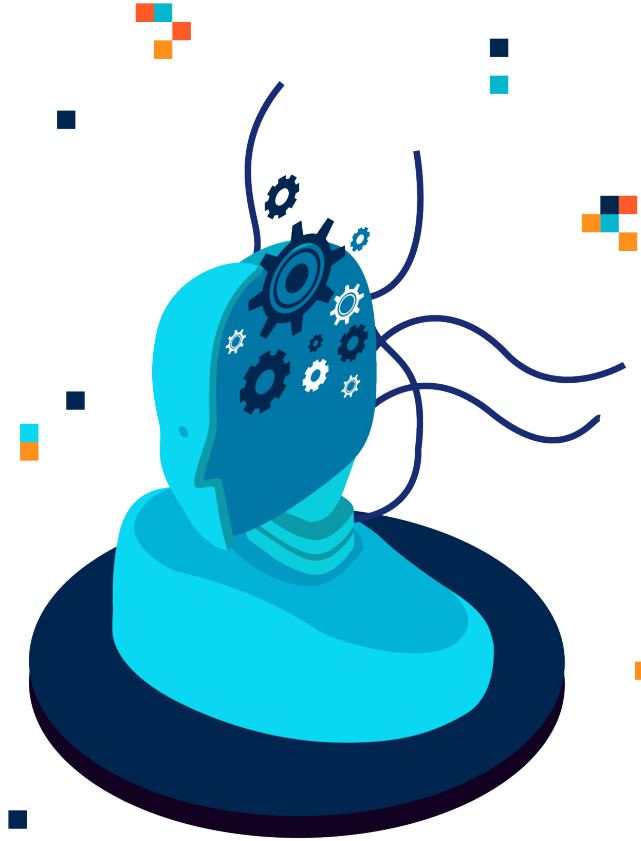
LDA: EC2 INSTANCE

- One CPU instances is sufficient for training and inference.



LDA: HYPERPARAMETERS

- **For full list of parameters:** https://docs.aws.amazon.com/sagemaker/latest/dg/lda_hyperparameters.html
- **feature_dim:** vocabulary size of the dataset
- **num_topics:** number of required topics
- **Alpha0:** Initial guess for the concentration parameter, smaller value generates sparse topic mixtures while Large values (>1.0) produce more uniform mixtures
- **Max_iterations**
- **Max_restarts**



OBJECT2VEC

OBJECT2VEC: OVERVIEW

- SageMaker Object2Vec is an unsupervised algorithm that generalizes the Word2Vec embedding technique introduced before (blazingtext algorithm).
- It is a general-purpose customizable neural embedding algorithm which can be used to embed sentence, movies and products.
- Object2vec can learn low-dimensional dense embeddings of high-dimensional objects.
- The embeddings keeps the relationship between pairs of objects.
- Object2vec can be used to: (1) compute nearest neighbors of objects, (2) visualize clusters of related objects.
- Using these embedding, you can feed this to perform classification or regression.
- Example: you can apply object2vec to learn the embeddings of customers. This can be achieved by creating training data from a sequence of transactions/purchases paired with the ID of the customer.

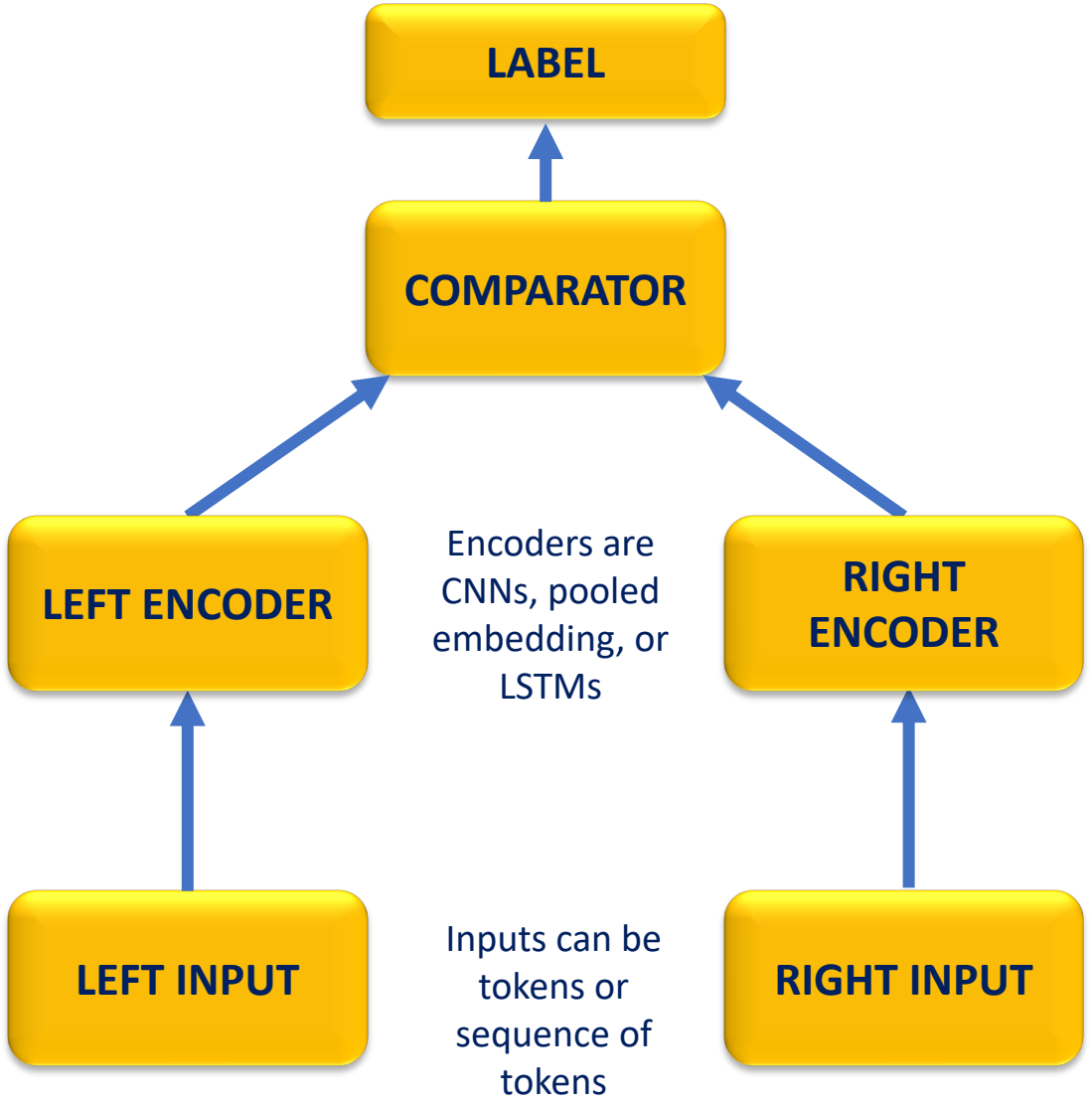
Photo Credit: https://pixabay.com/photos/groceries-fruit-vegan-soy-food-1343141/

OBJECT2VEC: OVERVIEW

- Input Data is tokenized into integers
- Object2Vec natively supports two types of input:
  - A discrete token: represented as a list of a single integer-id such as [5].
  - A sequences of discrete tokens: represented as a list of integer-ids such as [0,12,10,13].
- Object2Vec can be used on many input data types, such as:
  - Item review and user pair: user ID along with all the items he/she bought such as bananas, orange,..etc.
  - Customer to customer pair: customer ID of Jane and ID of Jackie
  - Product-product pair: product ID of banana and product ID of orange

OBJECT2VEC: OVERVIEW

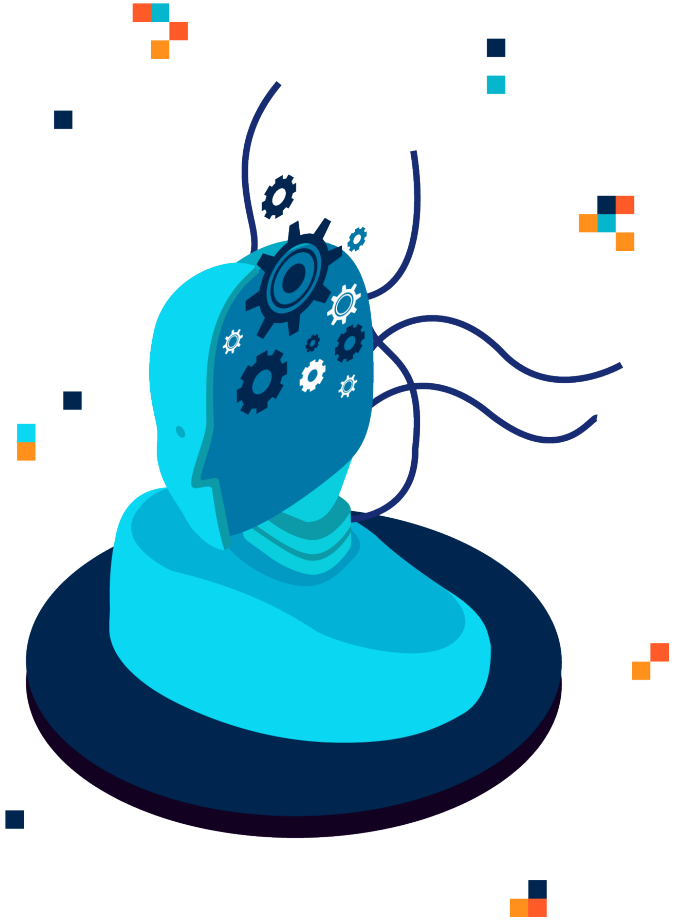
- For object2vec, pairs can as follows:
  - (token, sequence)
  - (token, token)
  - (sequence, sequence)
- Choose the proper encoder:
  - Average-pooled embeddings
  - convolutional neural networks (CNNs)
  - Multi-layered bidirectional long short-term memory (BiLSTMs)



https://aws.amazon.com/blogs/machine-learning/introduction-to-amazon-sagemaker-object2vec/

OBJECT2VEC: HYPERPARAMETERS

- For the full list of parameters: https://docs.aws.amazon.com/sagemaker/latest/dg/object2vec-hyperparameters.html
  - enc0_max_seq_len: The maximum sequence length for the enc0 encoder.
  - enc0_vocab_size: The vocabulary size of enc0 tokens.
  - comparator_list: A list used to customize the way in which two embeddings are compared.
  - Dropout
  - early_stopping_patience: The number of consecutive epochs without improvement allowed before early stopping is applied.
  - enc_dim: The dimension of the output of the embedding layer.
  - enc0_cnn_filter_width: The filter width of the convolutional neural network (CNN) enc0 encoder.
  - enc0_token_embedding_dim: The output dimension of the enc0 token embedding layer.



https://aws.amazon.com/blogs/machine-learning/introduction-to-amazon-sagemaker-object2vec/

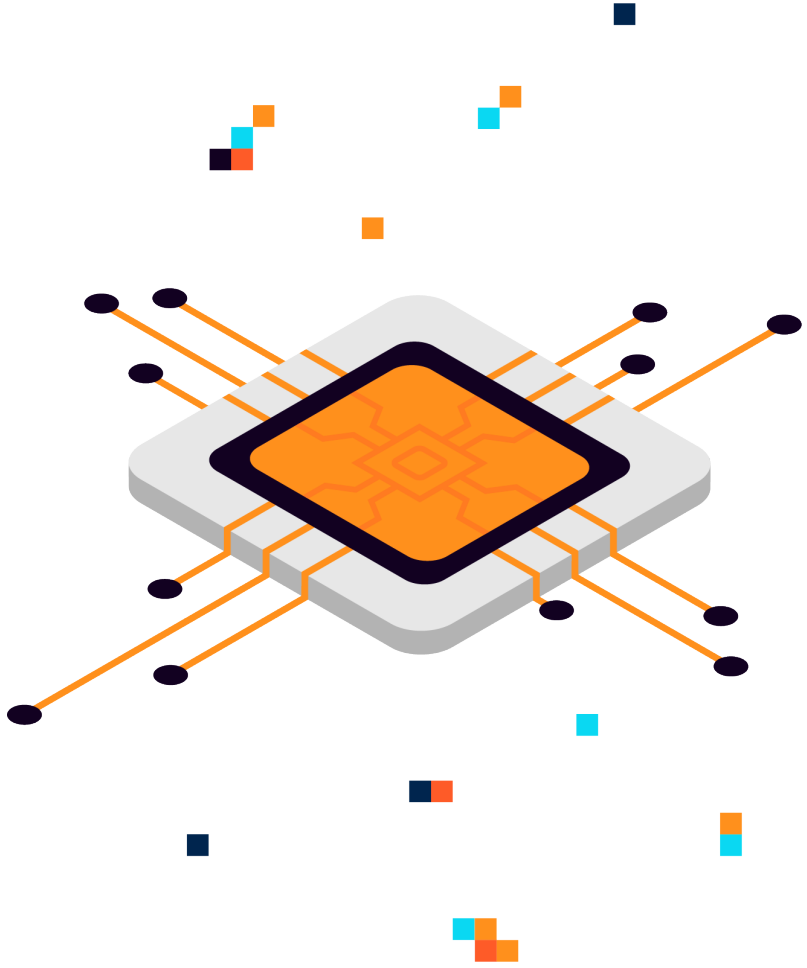
OBJECT2VEC: EC2 INSTANCES

Training:

- You can train Object2vec on CPU and GPU
- For CPU, start with an ml.m5.2xlarge instance.
- For GPU, start with an ml.p2.xlarge instance.
- Larger instance is recommended such as an ml.m5.4xlarge or an ml.m5.12xlarge instance if training takes longer time.
- Object2Vec algorithm can train only on a single machine but it does offer support for multiple GPUs.

Inference:

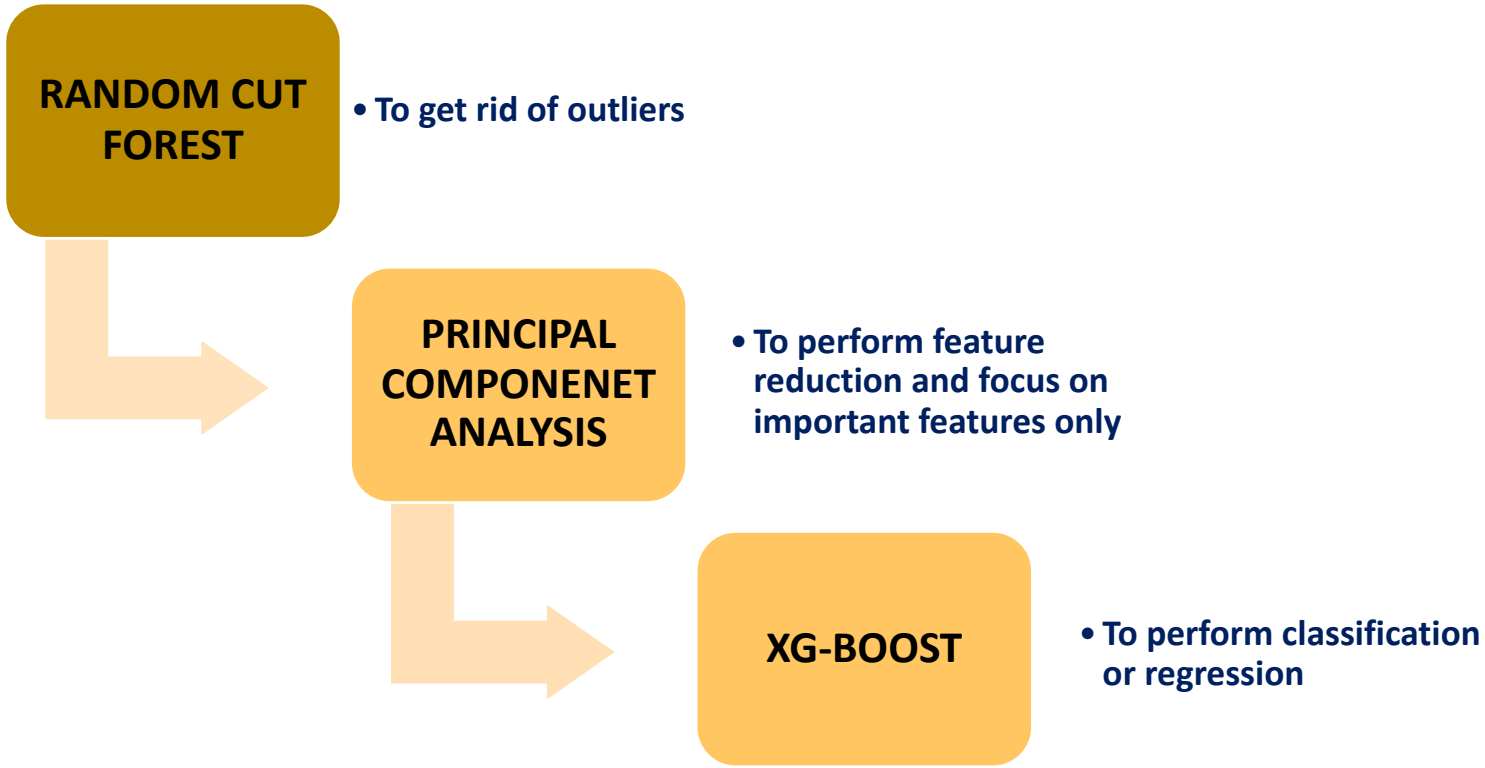
- ml.p3.2xlarge GPU instance is recommended of deep network is utilized.
- You can use INFERENCE_PREFERRED_MODE environment variable due to GPU memory scarcity. GPU optimization can be selected: (Classification or Regression) or GPU optimization (Encoder Embeddings).



MANY LAYERED MODELS TOGETHER

LAYERED ALGORITHMS

- Sometimes we might need to apply many algorithms in a sequential fashion as follows:



AWS SAGEMAKER AUTOMATIC MODEL TUNING

AWS SAGEMAKER AUTOMATIC MODEL TUNING

- Let's first cover the difference between parameter and hyper parameter:
  - **Hyper parameter:** values set prior to the training process such as number of neurons, layers, learning rate..etc
  - **Parameter:** values that are obtained by the training process such as network weights.

- Hyperparameter optimization is an extremely challenging task.
- Luckily, Amazon SageMaker automatic model tuning could perform hyperparameter optimization in an easy way.
- Automatic model tuning works by searching for the best version of a model.
- The user will select (1) the algorithm and (2) a range of hyperparameters (upper and lower bound) and Automatic model tuning will conduct numerous training jobs to find the optimal set of parameters.
- Automatic model tuning will then choose the final set of hyperparameter values that result in the best model.

AWS SAGEMAKER AUTOMATIC MODEL TUNING

- Example: A user can select the following:
  - **(1) Metric:** Users can select the metric to optimize for such as accuracy, precision, recall and AUC.
  - **(2) Training Algorithm:** User can select Xgboost algorithm to perform classification.
  - **(3) Range of hyperparameters:** User set the hyperparameters range such as eta, gamma, and lambda.
- Automatic model tuning will launch multiple training jobs using the range of hyperparameter values specific by the user and return the best set of parameters that optimizes for accuracy, recall or AUC (whatever metric you chose).
- The model is adaptive and learns from experience so it doesn't have to try every possible hyperparameters combination.

AWS SAGEMAKER AUTOMATIC MODEL TUNING

RANDOM SEARCH

- Random search hyper parameter optimization works by selecting random numbers from the range selected by the user.
- Parameter selection does not learn from experience (past runs).

BAYESIAN SEARCH

- Bayesian-based hyper parameter optimization works by learning from its past mistakes!
- The algorithm guesses the next combination to choose based on the previous performance.
- It treats the tuning process as a regression problem.
- Hyper parameter tuning uses an Amazon SageMaker implementation of Bayesian optimization.
- The algorithm works by applying exploration/exploitation.

AWS SAGEMAKER AUTOMATIC MODEL TUNING: BEST PRACTICES

- In order to get the best results when you run the Automatic model tuning, you need to do the following:

USE A SMALL RANGE FOR EACH OF THE HYPER PARAMETER

- For example: we know from experience that battery resistance is between 0 - 0.5 ohms, then use this range and avoid adding negative values.

AVOID RUNNING MULTIPLE TRAINING JOBS IN PARALLEL, THE AUTOMATIC MODEL TUNING LEARNS FROM EXPERIENCE

- So it's recommended that you run one tuning job first and then run another one and so on…the algorithm learns from experience.

ONLY EXPERIMENT WITH A SMALL NUMBER OF HYPER PARAMETERS

- Avoid optimizing many of them at once.
- The maximum per optimization job is capped at 20.

USE LOGARITHMIC SCALES

- Amazon SageMaker automatically tries to know if the hyperparameters are log-scaled or linear-scaled. If you know beforehand that the parameters are log-scaled, it's better to set that initially.