AWS SageMaker - XGBoost

Algorithms Overview

Algorithm	Description
Linear Models	+ Simple and easy to understand + Performs surprisingly well for a variety of problems - Difficulty handling non-linear datasets - Features on similar scale, one-hot encoding, complex features
Decision Tree	+ Can Handle Complex non-linear relationship + Easily handles numeric categorical data, missing data - Prone to overfitting - Poor predictive accuracy
Ensemble Methods	+ Combines multiple simple decision trees + Addresses decision tree overfitting problem + Much better predictive performance - More complex to understand

Must Watch Videos

Gradient Boosting Machine Learning by Trevor Hastie

Learning Decision Tree by Alexander Ihler

Ensembles (Bagging) by Alexander Ihler

Lab: Compare XGBoost and Linear Regression

Compare XGBoost and Linear Regression Algorithm

Using a Simple Dataset

Understand how these algorithms learn from data

Train locally on Notebook instance

Lab: Compare XGBoost and Linear Regression

Using a **non-linear** dataset

Understand how these algorithms learn from data

Train locally on Notebook instance

Lab: Forecast Bike Rental Count

Old Kaggle Competition Problem

Complex dataset

Need to forecast hourly rentals

Lab: Forecast Bike Rental Count - Optimization

Transform Count to *Log (Count)*

A technique used when model needs to predict positive integers

Use inverse transform *Exp (Count)* on predicted value

Smoothen effect of seasonality and trend, brings count to a similar scale

Lab: Train using SageMaker’s XGBoost

Upload Train and Validation files to S3

Specify Algorithm and Hyperparameters

Configure type of server and number of servers to use for Training

Create a real-time Endpoint for interactive use case

Lab: Prediction using SageMaker’s XGBoost

Invoke Endpoint for interactive use cases

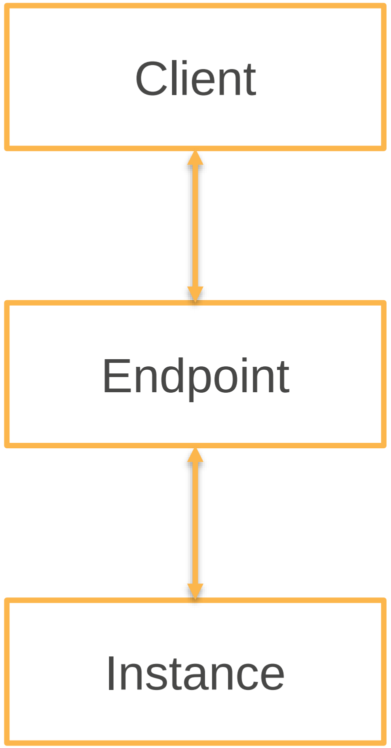
Connect to an existing Endpoint

Endpoint Security

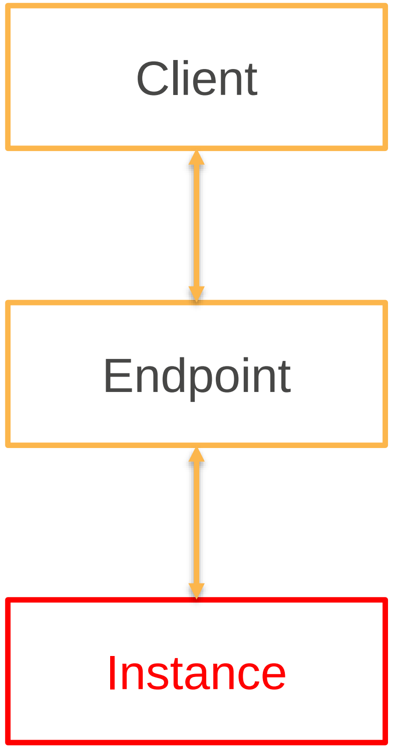
Multiple observations in a single round trip

Model Hosting

Model Hosting



Model Hosting



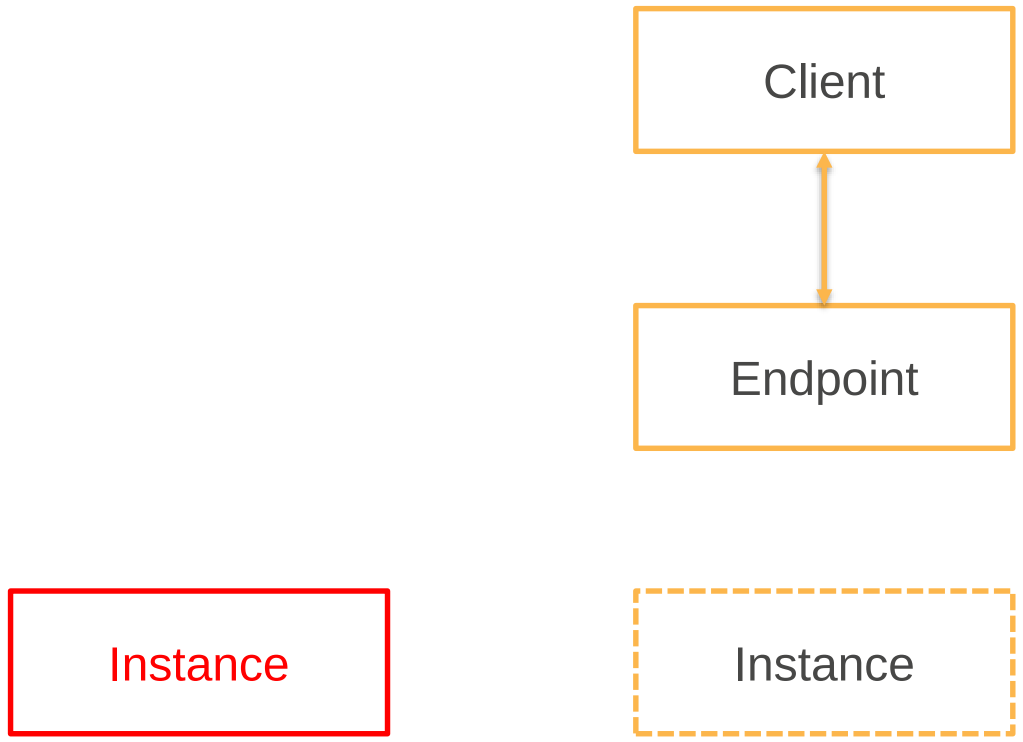
Single Instance Hosting = Single Point of Failure

Monitoring and Scaling

CloudWatch – Monitoring Service

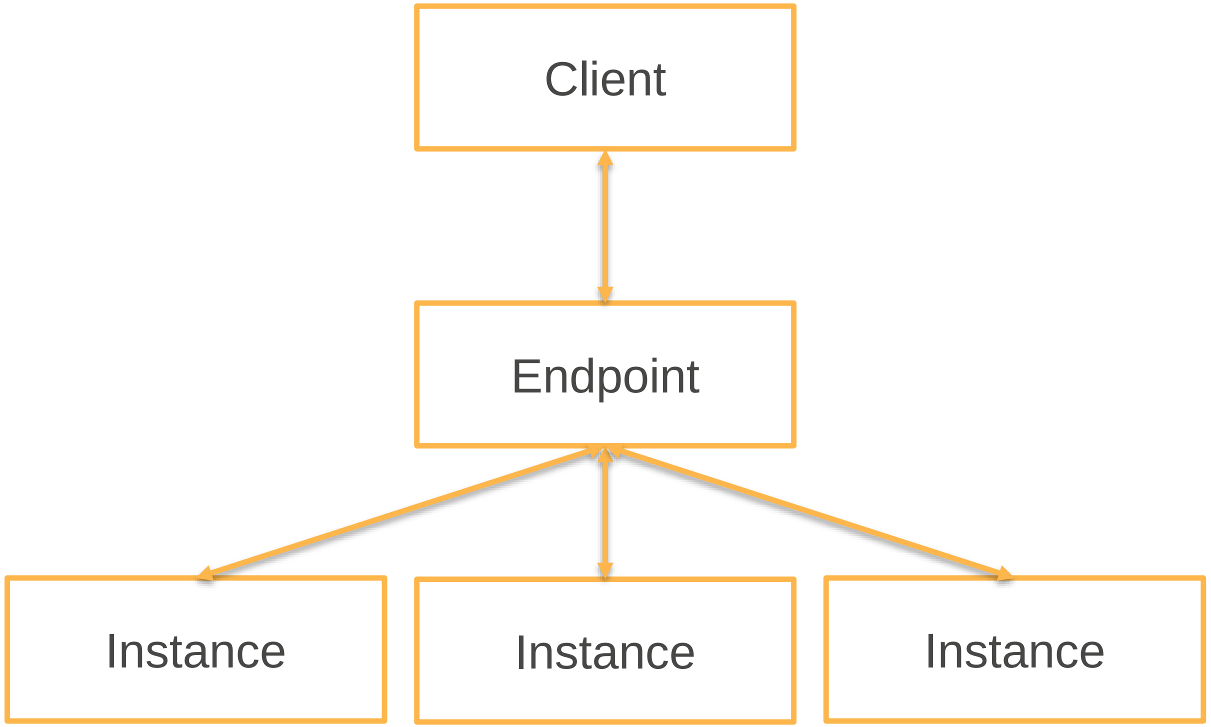
AutoScaling – Take automated scaling actions to maintain capacity

Model Hosting

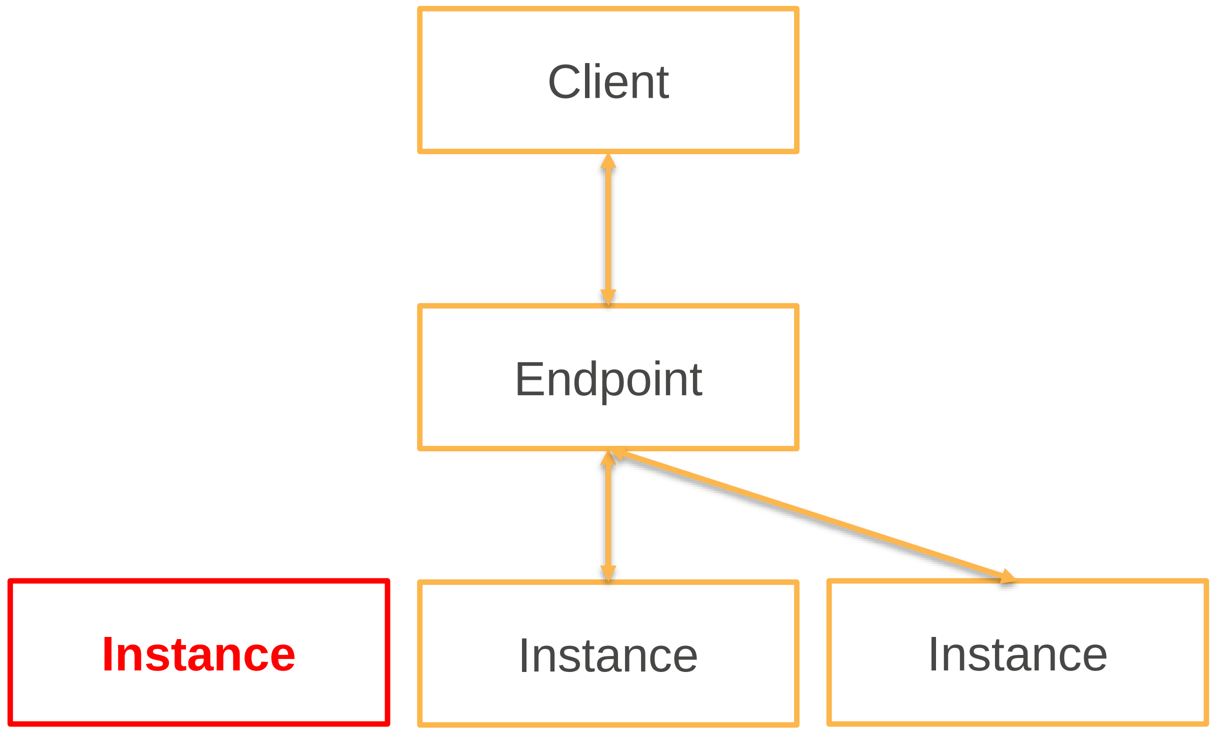


New instance launched to replace a failed instance

Model Hosting – Multiple Instance

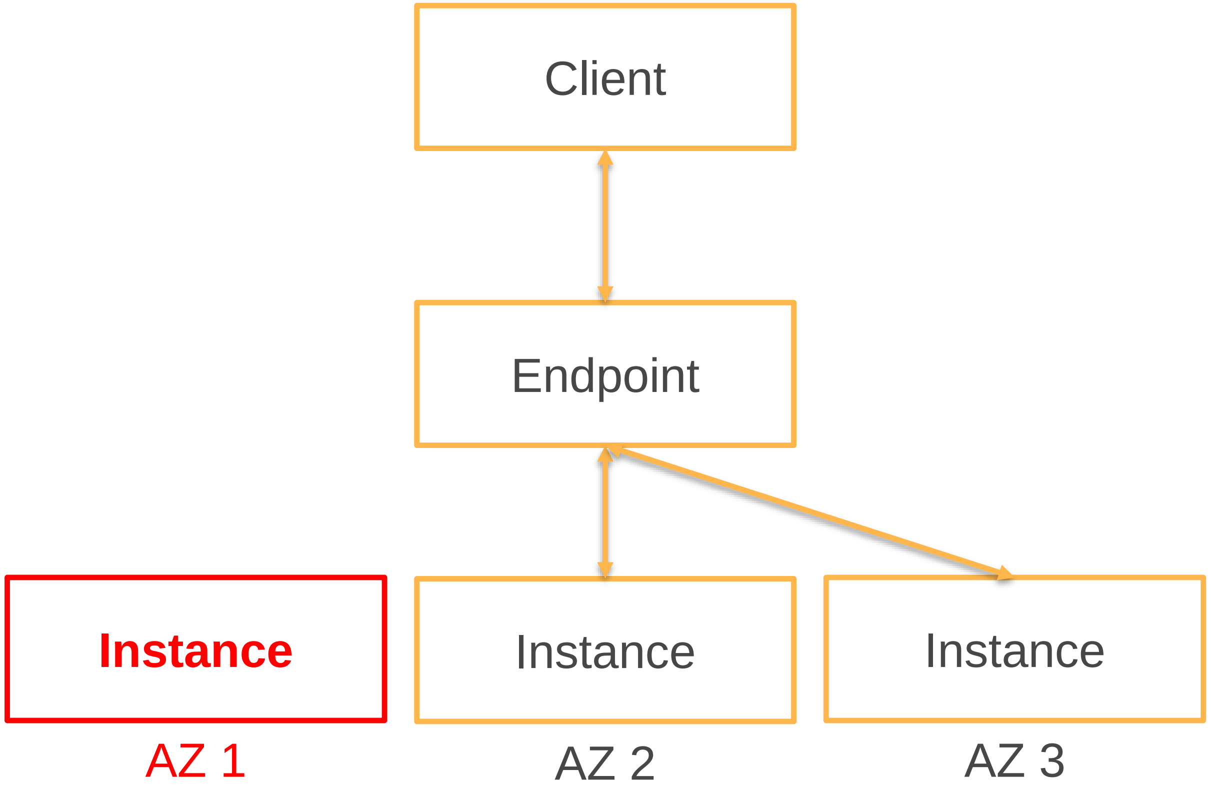


Model Hosting – Multiple Instance



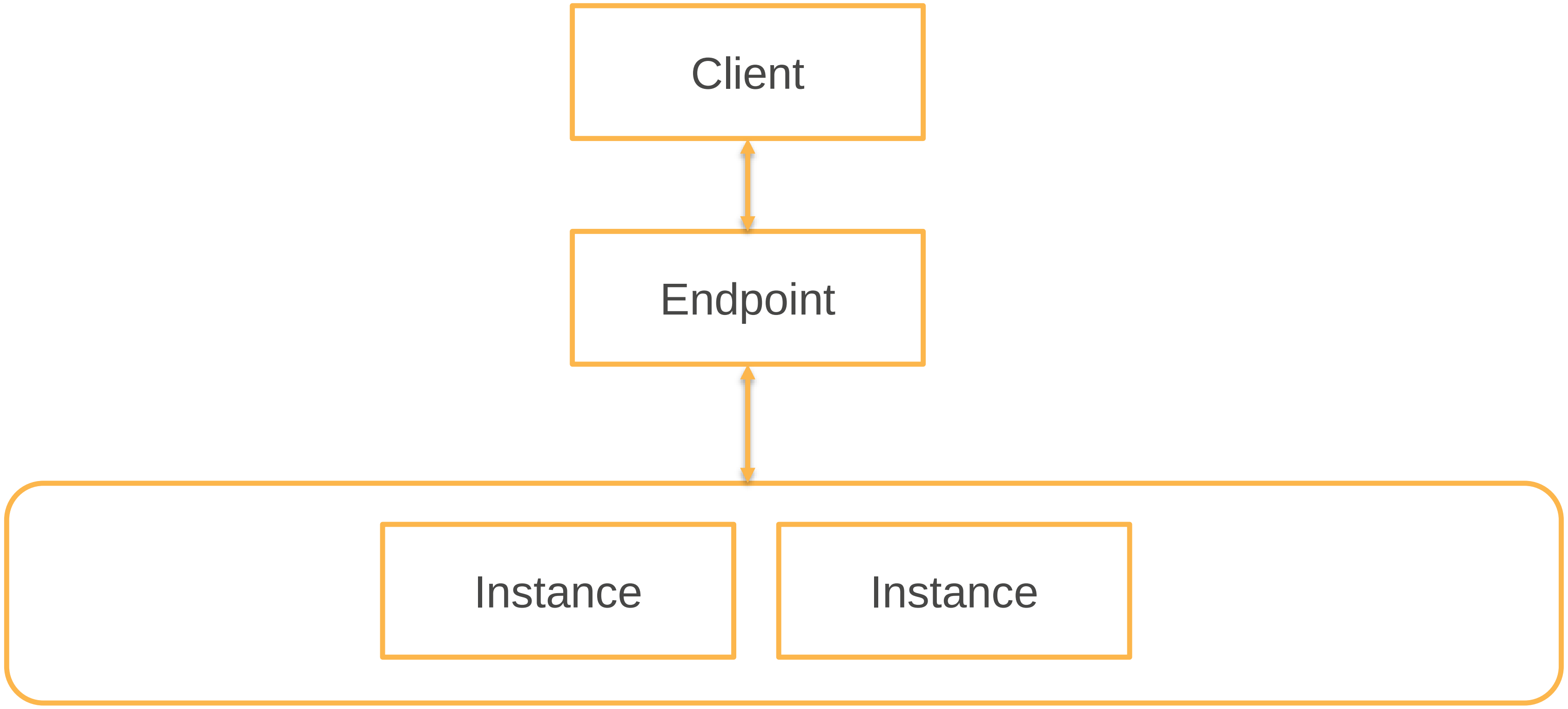
Requests load balanced across healthy instances

Model Hosting – Multiple Instance

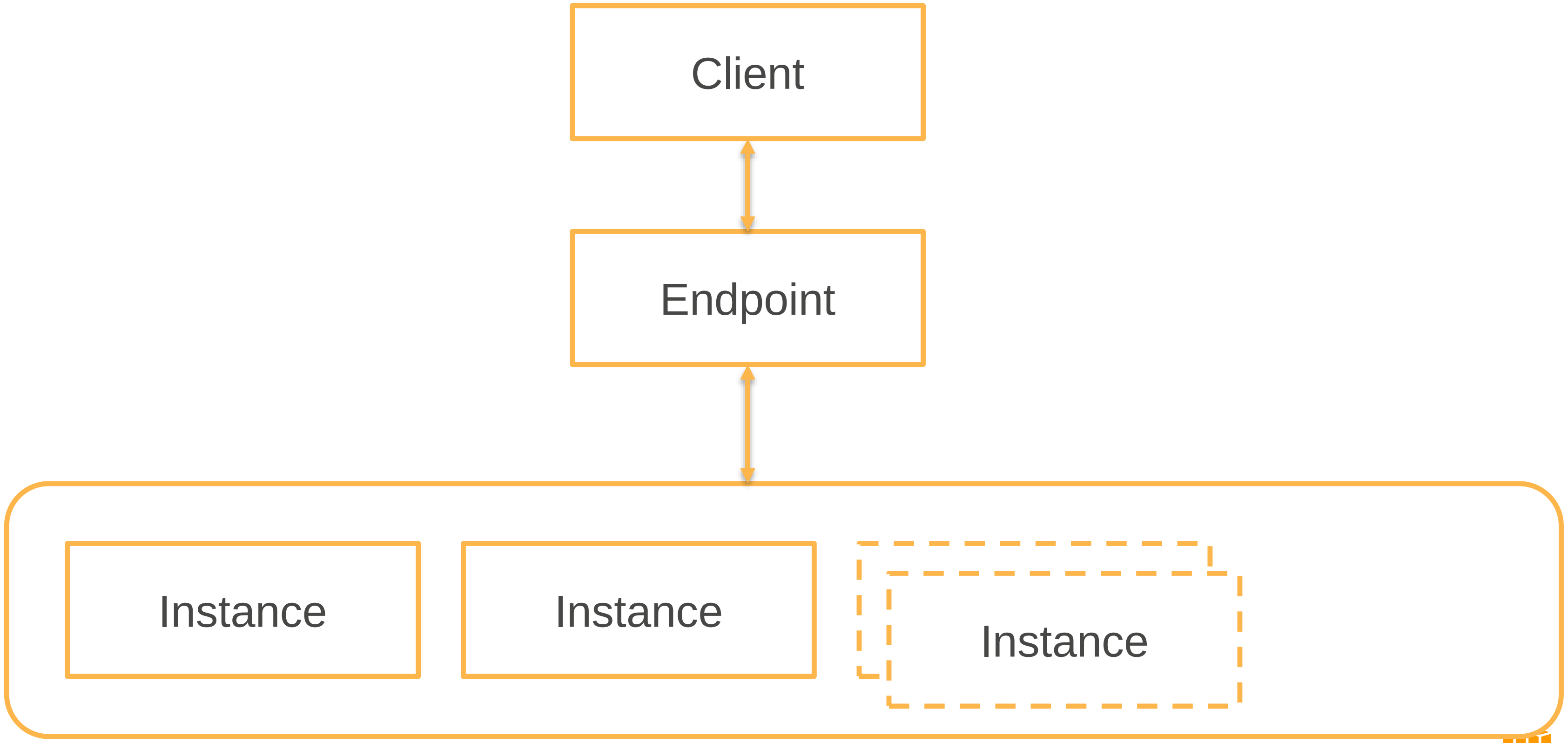


Handles Availability Zone Failures

Model Hosting – Scale on Demand



Model Hosting – Scale on Demand



Scaling based on Invocations

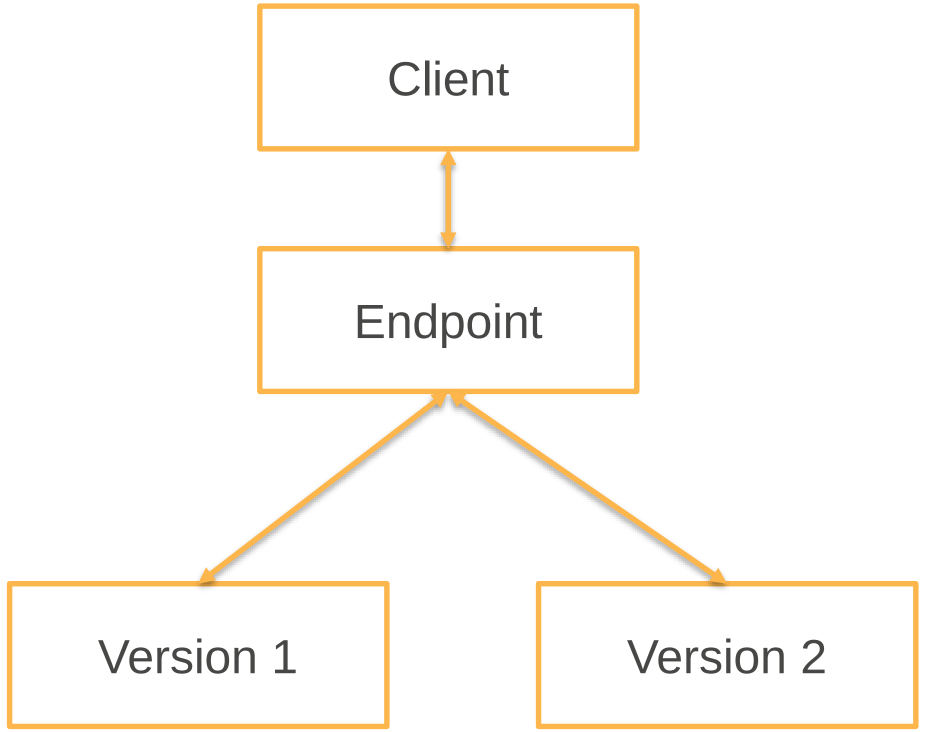
`SageMakerVariantInvocationsPerInstance` = Metric that records average number of requests per minute per instance

```
Use this metric for AutoScaling – For example,
Instance max load (MAX RPS) = 100 requests per second
Safety Factor = 0.5
Target SageMakerVariantInvocationsPerInstance =
           (MAX_RPS * SAFETY_FACTOR) * 60
```

```
SageMakerVariantInvocationsPerInstance = 100 * 0.5 * 60 = 3,000
```

Add additional instance when the metric crosses 3,000

Model Hosting – Variants of Algorithm



SageMaker Hosting

Automatically Replace Unhealthy Instances

Scale number of instances based on workload

Test Multiple Variants of Model

Hyperparameters

Training Objective

objective

- Regression – “reg:linear”
- Binary Classification – “binary:logistic”
- Multiclass Classification – “multi:softmax”

Parameter References:

- SageMaker Documentation
- XGBoost Documentation

Bias and Variance

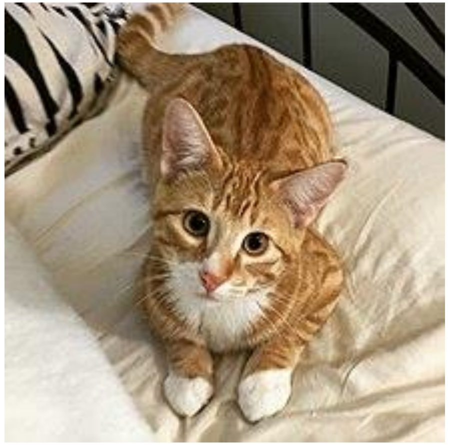


Photo Credit : Ugrashak

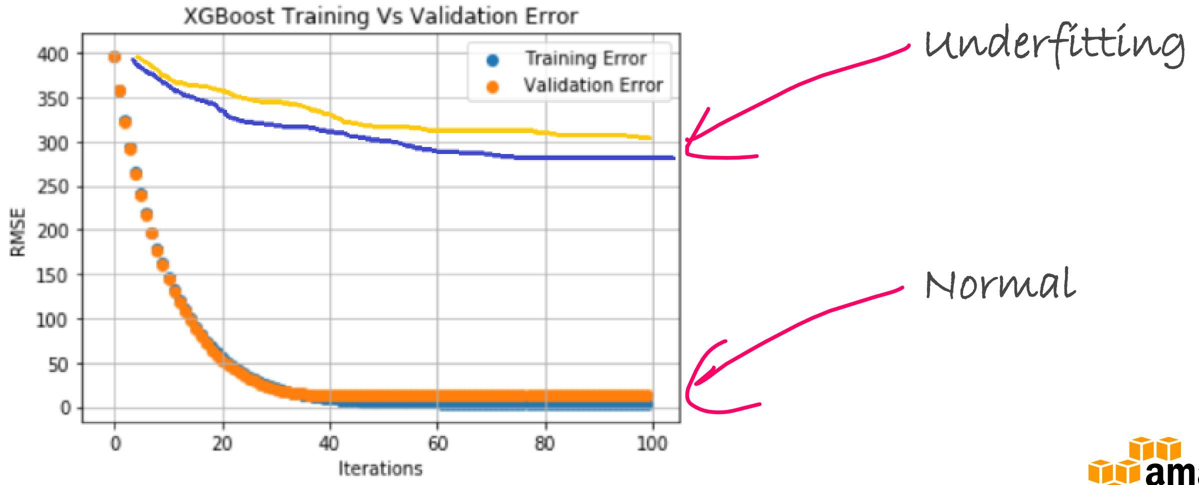
Biased => Does not match reality

High Bias

Model is not learning from data

Translates to large training and validation errors

Underfitting



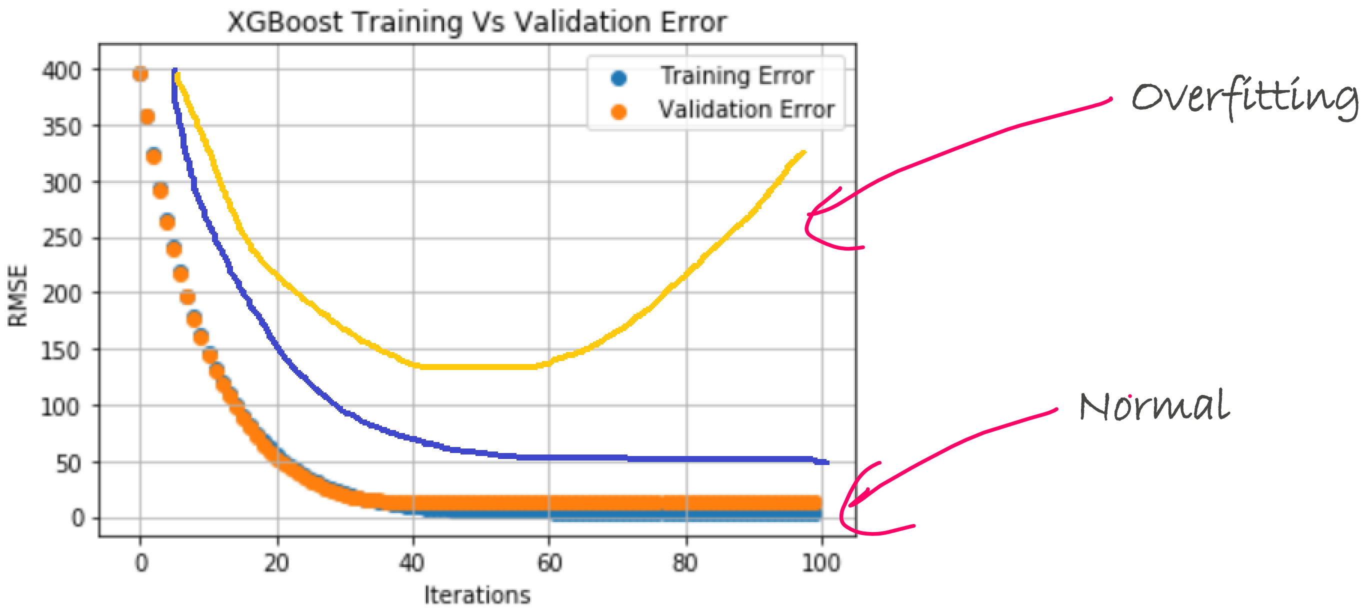
Variance

Measures how well the algorithm generalizes for unseen data

Difference between Validation Error and Training Error

High Variance – validation error is high; but training error is small. Overfitting

High Variance



Strategies to handle High Bias (Underfitting)

Add relevant features

Combine features (Example: area = length * width)

Create higher order features

Train longer (more iterations)

Decrease Regularization

Strategies to handle High Variance – Overfitting

Use fewer features

Use straightforward features (instead of higher order features)

Reduce Training iterations

Increase Regularization

Regularization

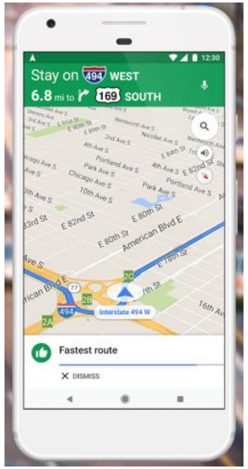
Many features are equally good at predicting outcome

Which combination of features is the model going to use?

Feature selection depends on algorithm and regularization parameters

Regularization

Regularization – Tone down overdependence of specific features



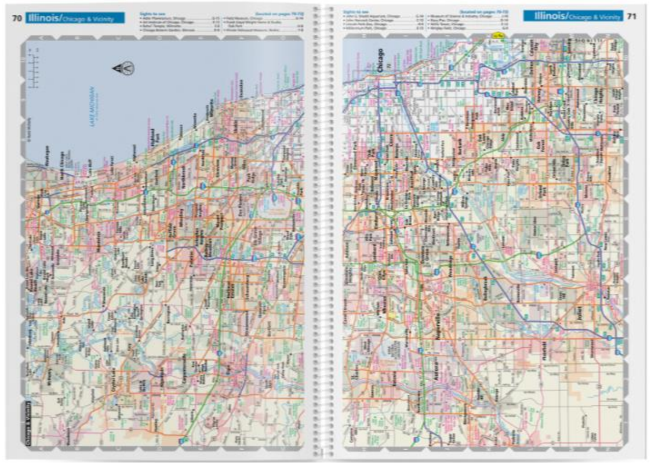

70 Illinois/Chicago & Vicinity
Illinois/Chicago & Vicinity 71

Photo Credit : Google, Garmin, Rand McNally

L1 Regularization

Algorithm aggressively eliminates features that are not important

Example:

Phone GPS = Substantial weight

Standalone GPS = Zero weight

Paper Map = Zero Weight

Useful in large dimension dataset – reduce the number of features

L2 Regularization

Algorithm simply reduces weight of features

Allows other features to influence outcome

L2 Regularization is a good starting point

Example:

- Phone GPS = Larger weight
- Standalone GPS = Medium weight
- Paper Map = Smaller weight

XGBoost Regularization

alpha – L1 Regularization. Default 0

lambda – L2 Regularization. Default 1

Hyper Parameter Tuning

XGBoost Parameter Tuning

SageMaker XGBoost Hyper Parameter Documentation

SKLearn - Automatic Tuning

GridSearch – Exhaustive search using specified lower and upper bound of parameter values

RandomSearch – Random Search of parameters from specified lower and upper bound

SageMaker - Automatic Tuning

Bayesian Search – Smart Search. Treats hyperparameter tuning as a machine learning problem. Often converges faster

Random Search – Random Search of parameters from specified lower and upper bound

Hyper Parameter Tuning

n_estimators (in XGBRegressor) is same as **num_round** (in XGBoost and SageMaker documentation)

This parameter controls number of rounds of boosting i.e. total number of trees.

Make sure you use correct parameter depending on the library. SKLearn *XGBRegressor silently ignores parameters it does not understand* ☹