

NI 기술 문서 #3

XGBoost를 활용한 OpenStack 환경 내 VNF Anomaly Detection 방법

문서 ID	NI-AnomalyDetection-01
문서 버전	v1.3
작성일	2020.09.15
작성자	포항공과대학교 홍지범, 이도영

• 문서 변경 기록

날짜	버전	변경 내역 설명	작성자
2020.09.10	v1.0	초안 작성	홍지범, 이도영
2020.09.13	v1.1	그림 갱신	홍지범, 이도영
2020.09.14	v1.2	오타 수정 및 보완	홍지범
2020.09.15	v1.3	오타 수정 및 보완	홍지범

목 차

1. 서론	1
1.1 개요	1
1.2 가상 네트워크 관리를 위한 Anomaly Detection 방법	1
2. XGBoost 기반 Anomaly Detection 방법	4
2.1 XGBoost 알고리즘	4
2.2 문제 정의 및 모델 학습	5
2.3 성능 검증	7
2.3 모듈 구현	11
3. XGBoost 기반 Anomaly Detection 모듈 사용법	14
3.1 환경 구축	14
3.2 웹 서버 기반 Anomaly Detection 모듈 설치	15
3.3 웹 서버 기반 Anomaly Detection 모듈 사용법	16
3.4 CLI 기반 Anomaly Detection 모듈 실행	20
4. 결론	21
5. 참고문헌	22

요 약 문

현대의 네트워크는 다양하고 급변하는 서비스 요구사항을 만족시키기 위해 보다 유연하고 민첩한 네트워크 구축 및 관리가 요구된다. 네트워크 기능 가상화 (NFV, Network Function Virtualization)는 이러한 요구사항을 실현하기 위한 기술 중 하나로, 기존 전용 하드웨어를 통해 제공되는 네트워크 기능을 소프트웨어 형태로 구현하여 상용 서버에서 가상 네트워크 기능 (VNF, Virtualized Network Function)으로 운영하는 것이다. NFV 개념이 제안된 이후 통신 사업자와 서비스 제공 업체는 NFV 기술을 활용하여 네트워크 구축 및 서비스를 보다 효율적으로 제공하려고 노력하고 있다. NFV 기술은 네트워크 기능 가상화하여 네트워크에 유연하게 적용할 수 있는 장점이 있지만 수 많은 가상 자원을 생성하기 때문에 네트워크가 더욱 복잡해져 네트워크 관리 문제를 발생시킨다. 이와 같이 발생하는 모든 복잡한 네트워크 관리 문제를 네트워크 관리자 (사람)가 직접 해결하는 것은 한계가 존재하기 때문에, 최근에는 사람의 개입 없이 인공지능 (AI, Artificial Intelligence)을 적용하여 네트워크를 관리하려는 연구들이 활발하게 이루어지고 있다.

NFV 환경의 Anomaly Detection은 VNF 라이프 사이클 (Life-cycle) 관리 기능 중 하나로, VNF 인스턴스 (Instance)의 장애 및 비정상적인 동작 상태 (abnormal state)를 탐지하는 것이다. 본 문서에서는 VNF 인스턴스의 과부하, SLA (Service Level Agreement) 위반과 같은 비정상적인 동작 상태를 탐지하는 것을 목표로 정의하고 서술한다. NFV 환경에서 운영되는 대부분의 서비스는 네트워크 기능을 적용한 Service Function Chaining (SFC)를 통해 제공된다. 본 문서에서는 NFV 환경에서 SFC가 구성된 특정 서비스를 동작시키고 이러한 상황에서 VNF 인스턴스들의 비정상적인 동작 상태를 탐지하기 위한 XGBoost 기반 Anomaly Detection 방법에 대해 서술한다.

1.1 개요

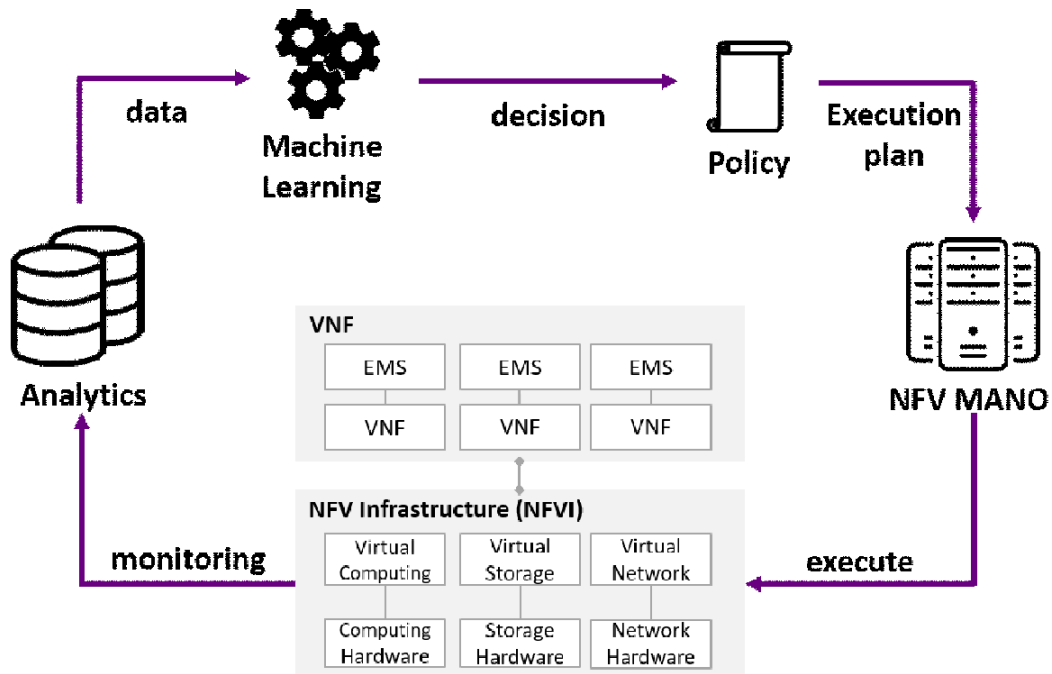
오늘날 소프트웨어 정의 네트워킹 (SDN, Software-Defined Networking) 및 네트워크 기능 가상화 (NFV, Network Function Virtualization) 기술은 네트워크를 설계, 구축 및 운영하는 새로운 패러다임을 제공하고 있다. 이러한 기술을 통해 통신 사업자와 서비스 제공 업체는 기존의 폐쇄된 네트워크 기능을 소프트웨어화한 VNF (Virtual Network Functions)로 대체하여 CAPEX 및 OPEX를 줄일 수 있다 [1]. 이에 따라 가상 네트워크에서 작동하는 VNF 및 서비스의 수가 크게 증가하고 있지만 반대로 이러한 증가는 가상 네트워크를 복잡하게 만들 수 있다. 따라서 이로 인하여 성능 저하 문제, 서비스 오류 또는 시스템 과부하 문제들이 발생한다. 이러한 네트워크 관리 문제를 해결하기 위해 오케스트레이션, 성능, 보안 등과 관련된 몇 가지 요구 사항 [2]이 정의되었으며, 대표적으로 자원 관리 및 장애 관리 등이 있다.

기존 네트워크 관리는 네트워크 관리자가 수동으로 네트워크 관리를 직접 수행했지만 네트워크 및 서비스 요구 사항이 다양해지고 복잡해짐에 따라 서비스를 배포하거나 변경하는데 더 많은 시간이 걸리게 되었다. 또한 세부 네트워크 구성에 대한 전문적인 인력을 필요로 하게 되었다, 그러나 소규모 네트워크가 아닌 대규모 네트워크와 복잡한 종속성을 가지고 있는 네트워크의 경우 네트워크 관리자가 이를 모두 해결하는 것은 상당한 시간과 비용을 소모하기 때문에 기존과는 다른 접근 방식이 필요하게 되었다. 이에 최근 머신러닝 또는 딥러닝을 적용한 가상 네트워크 관리에 대한 연구가 주목을 받고 있으며, 사람의 개입 없이 네트워크의 상태를 이해하고 네트워크를 최적으로 관리할 수 있는 기술을 개발하려는 연구들이 진행되고 있다 [3].

1.2 가상 네트워크 관리를 위한 Anomaly Detection 방법

NFV 환경에서 머신러닝을 활용하여 가상 네트워크 관리의 자동화는 일반적으로 그림 1과 같은 프로세스로 구성된다. 먼저, VNF들은 가상 리소스를 사용하여 NFVI (NFV Infrastructure) 상에서 동작하고, 데이터 분석부 (Analytics)에서는 VNF들의 기본 정보 및 리소스 사용량을 모니터링한다. 이 때 모니터링을 통해 수집되는 데이터는 일반적으로 CPU 사용량, 메모리 사용량, 네트워크 트래픽로드 등과 같은 데이터로 구성되며 [4], 수집된 데이터는 데이터베이스에 저장된다. 그 후 수집된 데이터를 학습을 위한 데이터셋으로 변환하여 머신러닝 알고리즘을 훈련시키고 해당 알고리즘을 기반으로 특정 목적을 가진 모델을 생성한

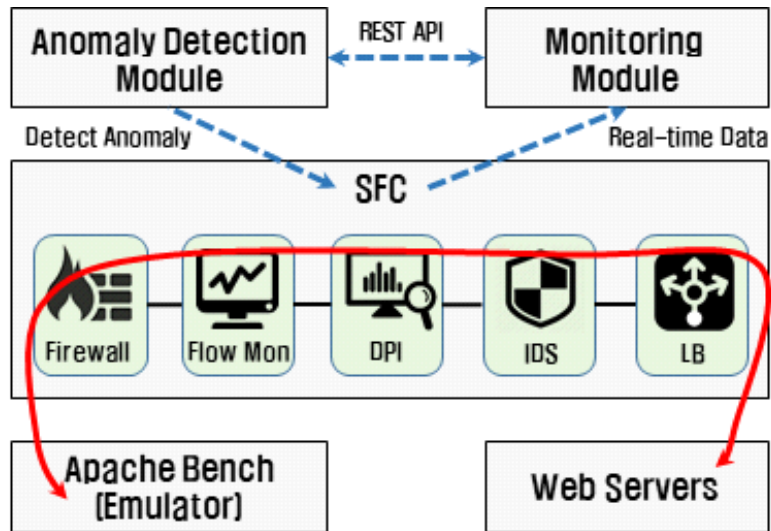
다. 그 후 생성된 모델은 현재 상태의 데이터를 기반으로 VNF 및 가상 네트워크의 운영을 위한 의사결정에 도움을 주거나 직접 네트워크 관리를 위한 정책 (policy)을 생성한다.



(그림 1) NFV 환경에서의 가상 네트워크 관리 프로세스

마지막으로 NFV Management and Orchestrator (MANO)는 생성된 네트워크 관리 정책에 따라 전체 NFV 환경을 운영하고 관리한다.

본 문서는 이러한 네트워크 관리 자동화에 대한 요구 사항 중 하나인 장애 관리 기술에 주목하여 NFV 환경에서의 VNF 이상 상태 탐지 방법에 대해 서술한다. 일반적으로 이상 상태 탐지는 1) 시스템 이상 상태 탐지와 2) 네트워크 공격 탐지의 두 가지 유형으로 구분된다. 시스템 이상 상태 탐지는 물리 노드 및 VNF의 상태를 모니터링하여 CPU 및 메모리와 관련된 메트릭과 같은 리소스 과부하 상태를 탐지한다. 네트워크 공격 탐지의 경우 모델은 네트워크 트래픽의 동작을 학습하고 트래픽의 급격한 증가와 같은 평소와 다른 패턴을 탐지한다. 본 문서에서는 머신러닝 알고리즘을 통해 모니터링 데이터를 학습시켜 시스템 및 네트워크 리소스 사용량을 기반으로 VNF의 이상 상태를 탐지한다. 기존 연구는 대부분 CPU / 메모리 사용량 및 처리량과 같은 시스템 자원 과부하와 관련된 VNF의 이상을 감지하기 위한 이진 분류 (binary classification) 방법을 제안하였다. 하지만 본 문서에서는 시스템 자원 과부하 뿐만 아니라 SLA (Service Level Agreement) 위반 현황을 모두 고려한 머신 러닝 기반 VNF 이상 탐지 및 이상의 원인이 되는 VNF가 무엇인지 (Root-Cause Localization) 보다 상세하게 분석한다.



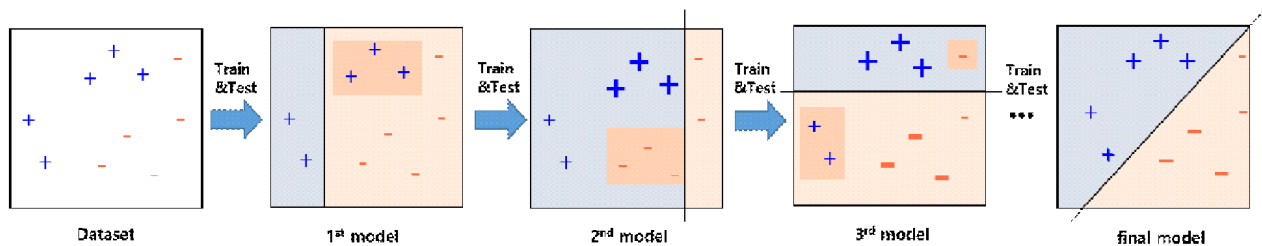
(그림 2) 실험 시나리오 구성

검증을 위한 서비스 시나리오는 그림 2와 같이 OpenStack 테스트베드를 사용하여 실제 토폴로지에서 웹 및 로그인 서비스를 다양한 VNF를 통해 SFC를 구성하여 동작시키고, 테스트 베드에서 수집한 데이터를 다양한 머신러닝 모델을 통해 성능을 분석하여 최적의 머신러닝 모델을 도출한다. 마지막으로 도출된 최적의 모델을 테스트베드에 실제 운영함으로써 실시간으로 VNF의 이상 상태를 탐지한다.

2.1 XGBoost (Extreme Gradient Boost) 알고리즘

머신러닝을 이용한 VNF의 이상 탐지 모델을 학습시키기 위해 본 문서에서는 머신러닝의 3가지 범주 (지도학습, 비지도학습, 강화학습) 중 지도학습 기반의 머신러닝 알고리즘을 활용한다. 지도학습 기반의 머신러닝 알고리즘을 이용한 VNF 이상 탐지 방법은 NFV 환경의 모니터링 시스템에서 수집한 VNF의 시스템 자원 사용량 및 SLA 위반 상태를 기반으로 데이터에 레이블을 지정하여 그 데이터를 기반으로 모델을 학습시킨다. 본 문서에서는 다양한 지도학습 알고리즘 중 XGBoost (Extreme Gradient Boost)라는 알고리즘을 사용하여 이상 탐지 모델을 학습시킨다.

먼저 XGBoost의 기반이 되는 GBM 알고리즘 [5]은 여러 개의 분류기 (classifier)를 생성하고 각 분류기의 결과를 결합함으로써 단일 분류기를 활용하는 모델보다 정확한 결과를 도출하는 기법인 앙상블 러닝 (ensemble learning) 기법으로, 그 중 여러 개의 분류기가 순차적으로 학습을 진행하는 부스팅 (boosting) 기법을 기반으로 하는 알고리즘이다. GBM은 여러 개의 의사 결정 트리 (decision tree)를 사용한다는 점에서 Random Forest (RF) 알고리즘과 비슷하다. 그러나 병렬적으로 각 트리를 학습시킨 후 voting 방식을 통해 최종 분류 결과를 도출하는 RF 알고리즘과는 달리 GBM은 그림 3과 같이 데이터셋을 통해 트리를 학습시킨 후 생성된 모델의 결과를 분석하여 잘못 구별한 예리 데이터에 대해 가중치를 주어 반복적으로 트리를 학습시켜 최적의 모델을 생성한다. 이 때 모델의 최적화는 손실 함수 (loss function)의 기울기 (gradient)를 통해 오류를 최소화하는 방향으로 학습을 진행한다.



(그림 3) GBM 알고리즘의 모델 학습 과정

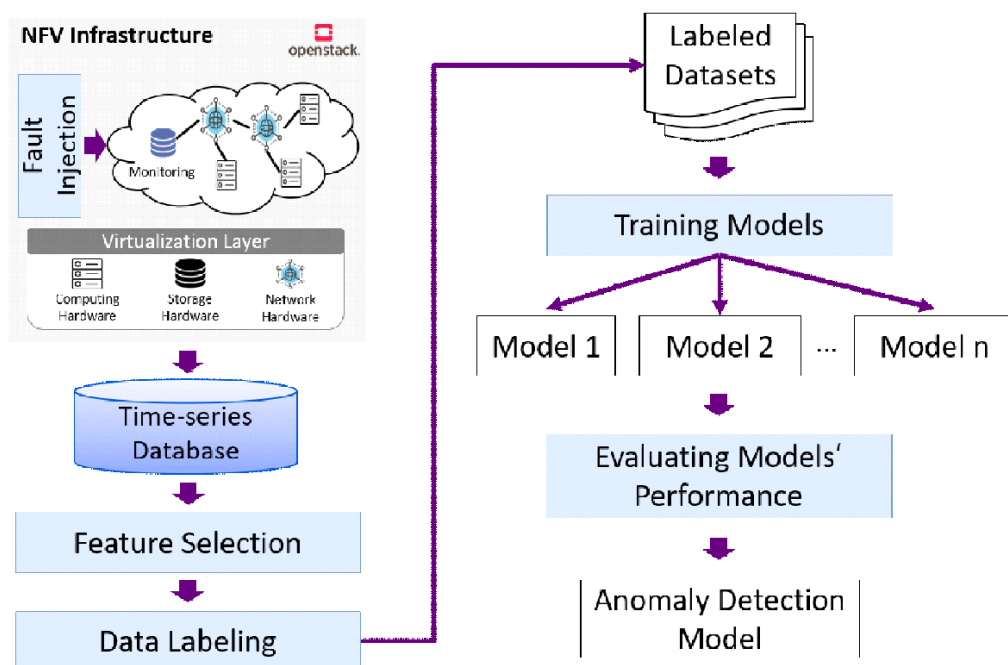
하지만 이러한 GBM 알고리즘은 트리의 학습을 순차적으로 하기 때문에 학습 시간이 오래 걸리고, 과적합 (over-fitting)이 일어날 위험이 크다. XGBoost 알고리즘 [6]은 이러한 GBM 알고리즘의 단점을 개선하여 보다 빠르게 학습을 진행하기 위해 희소 데이터 (sparse data)를

처리하기 위한 프로세스와 트리 학습에서 가중치를 처리할 수 있는 weighted quantile sketch 라는 프로세스를 포함한다. 또한 과적합을 방지하기 위해 정규화 프로세스 및 row/column 샘플링 방법을 사용한다.

2.2 문제 정의 및 모델 학습

본 장에서는 VNF Anomaly Detection 모델 학습 방법과 이상 상태 정의 방법에 대해 서술한다. 머신러닝 기반 VNF VNF Anomaly Detection 모델의 학습 과정을 나타낸다. 학습 과정은 크게 가상 네트워크 모니터링, 전처리, 모델 학습의 3가지 프로세스로 이루어진다. 상기 프로세스를 통해 생성된 모델들은 그 성능을 비교하여 최적의 모델을 도출한다.

먼저 모니터링 프로세스는 NFV 환경에 구축된 가상 네트워크에서 동작하는 VNF들의 데이터를 모니터링하고 저장한다. 이러한 모니터링 프로세스를 위한 기능은 모니터링 에이전트, 모니터링 모듈 및 대시보드로 구성된다. 모니터링 에이전트는 가상 네트워크에서 동작하는 각 VNF로부터 데이터를 수집하여 데이터를 저장하거나 제공하는 모니터링 모듈로 전달한다. 이 때 이상 탐지를 위해 수집되는 메트릭은 CPU 사용량, 메모리 사용량, 네트워크 트래픽 로드 등과 같은 NFV 환경 관리를 위해 사용되는 메트릭들을 사용한다 [4]. 그 후 모니터링 에이전트는 수집된 데이터를 모니터링 모듈로 보낸다. 이 때 모니터링 모듈은 데이터를 시계열(time-series) 데이터베이스에 저장하거나 실시간으로 대시보드 및 질의(query)를 통해 사용자에게 데이터를 제공한다.



(그림 4) 머신러닝 기반 VNF Anomaly Detection 모델 학습 과정

상기 방법을 통해 Anomaly Detection 모델에 필요한 학습 데이터를 수집하는 과정에서 실제 운용 환경에서는 이상 상태 (anomaly)가 잘 발생하지 않기 때문에 본 문서에서는 Fault Injection 기법을 통해 다양한 소프트웨어 및 하드웨어 결함과 같은 이상 상태를 에뮬레이션한다. 본 문서에서는 이와 같이 VNF의 이상 상태를 에뮬레이션하는 방법으로는 예를 들어 VNF에 직접 과부하를 주는 방법의 경우, CPU 사용률, 메모리 사용량, 디스크 I/O, 네트워크 트래픽 등의 부하를 주는 것과 같이 VNF에 직접 과부하를 주어 해당 VNF에 오류를 발생시키거나, client side에서 트래픽을 과다하게 발생시켜 전체 서비스에 과다한 워크로드를 발생시키는 방법이 있다. 본 문서에서는 오픈소스 툴들을 사용하여 상기 2가지 Fault Injection 기법을 적용하고, 이를 통해 SLA 위반 및 VNF의 과부하를 유도한 후 이 데이터를 학습에 활용한다.

다음으로, 전처리 프로세스에서는 수집된 모니터링 데이터를 모델 학습에 적합한 형태로 변환한다. 전처리는 크게 Feature Selection과 Data Labeling으로 구성된다 (그림4). feature selection은 모니터링을 통해 수집된 메트릭을 통해 이상 상태를 구분하는 기준과 가장 연관이 있는 메트릭을 선정한다. 이 과정에서는 각 메트릭 간의 상관관계가 높은 메트릭을 제거한다. 본 문서의 모델 학습 과정에서 활용한 feature의 목록은 표 1과 같다.

Features	Description	Features	Description
time	Measurement time	mem_used	Memory - used space
instance	VNF instance name	disk_free	Disk - free space
cpu_idle	CPU - idle time	disk_reserved	Disk - reserved space
cpu_interrupt	CPU - interrupt time	disk_used	Disk - used space
cpu_nice	CPU - nice status time	io_read	I/O - read bytes
cpu_softirq	CPU - softirq time	io_write	I/O - write bytes
cpu_steal	CPU - stolen time	io_time	I/O - spent time
cpu_system	CPU - used by kernel mode	network_rx_bytes	Received traffic bandwidth
cpu_user	CPU - used by user mode	network_tx_bytes	Transmitted traffic bandwidth
cpu_wait	CPU - I/O wait time	network_rx_packets	Received traffic bandwidth
mem_free	Memory - free space	network_tx_packets	Transmitted traffic bandwidth
mem_buffered	Memory - buffered space	network_latency	Hop latency between VNFs
mem_cached	Memory - cached space		

(표 1) VNF Anomaly Detection 모델 학습에 활용한 feature 목록

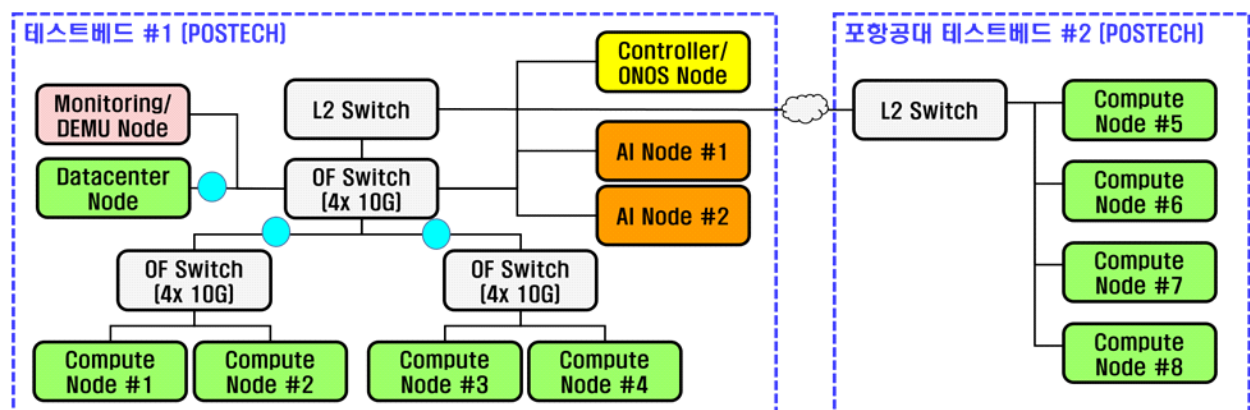
다음으로 Data Labeling은 본 문서에서 서술하는 Anomaly Detection 문제 정의에 맞추어 추출된 feature 데이터를 정상 및 이상 상태로 구분하여 모델 학습 및 성능 검증을 위한 데이터셋을 생성한다. 이 때 Root-Cause Localization을 통해 이상의 원인이 되는 VNF까지 분석

하는 경우는 정상 및 이상의 이진 분류가 아닌 이상 상태가 발생했을 때 VNF의 위치까지 보다 구체적으로 labeling한다. 본 문서의 Anomaly Detection 방법에서 VNF 이상 상태에 대한 정의는 1) VNF의 시스템 자원 사용 과부하 상태, 2) SLA 위반 발생 상태의 두 가지 경우로 이상 상태를 정의한다. 먼저 VNF의 시스템 자원 사용 과부하 상태의 경우, VNF에 Fault Injection을 통해 직접 CPU, 메모리, disk I/O 액세스에 대해 과부하를 준 상황을 labeling 하였고, 다음으로 SLA 위반 발생의 경우에는 본 문서의 실험 환경인 웹 서비스의 SLA 기준을 따른다. 일반적으로 웹 서비스에 대한 SLA 기준은 평균 응답 시간 (average response time) 과 가용률 (availability)을 이용하고 있기 때문에 본 문서에서는 이에 대한 문서 [7]를 참고하여 이를 테스트베드에 맞게 조정하여 labeling을 진행한다.

마지막으로 모델 학습 프로세스는 상기 설명한 과정에서 생성된 데이터셋을 통해 Anomaly Detection 모델을 학습시킨다. 모델의 학습은 본 문서에서 제안하는 XGBoost 알고리즘의 하이퍼파라미터 (e.g., the number of trees, depth of trees, the number of leaves, row/column sample rate 등)를 조정하여 수행한다. 그 후 생성된 모델들의 성능을 분석 및 비교하여 최적의 Anomaly Detection 모델을 도출한다.

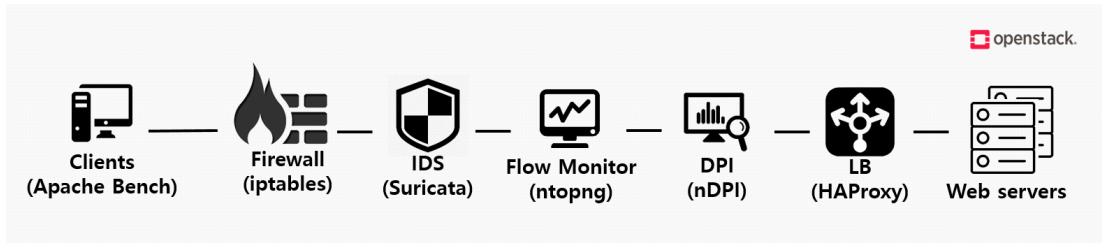
2.3 성능 검증

XGBoost 기반 Anomaly Detection 방법의 성능 검증을 위해 그림 5와 같이 OpenStack 기반으로 테스트베드를 구축하였다. 테스트베드는 OpenStack 환경을 전반적으로 관리하는 컨트롤러 노드 (Controller Node)와 VNF가 설치되어 동작할 수 있는 컴퓨트 노드 (Compute Node)로 구성된다. 상기 테스트베드는 1개의 컨트롤러 노드와 8개의 컴퓨트 노드로 구성되어 있다. OpenStack 테스트베드에서 VNF를 설치하여 동작시킬 때 1개의 VM에 1개의 VNF가 동작하는 것으로 가정한다. 본 문서에서 활용한 OpenStack 환경의 테스트베드 구축을 위해 사용 방법은 Github를 통해 확인할 수 있다 [8].



(그림 5) OpenStack 기반 테스트베드 구성도

성능 검증 시나리오는 그림 6과 같이 웹 호스팅 서비스 시나리오에서 VNF들의 SFC를 구성하였다. SFC는 firewall, IDS (Intrusion Detection System), DPI (Deep Packet Inspection), flow monitor, load balancer (proxy)과 같은 오픈소스 VNF로 구성된다.



(그림 6) 실험 시나리오 구성도: 웹 호스팅 시나리오

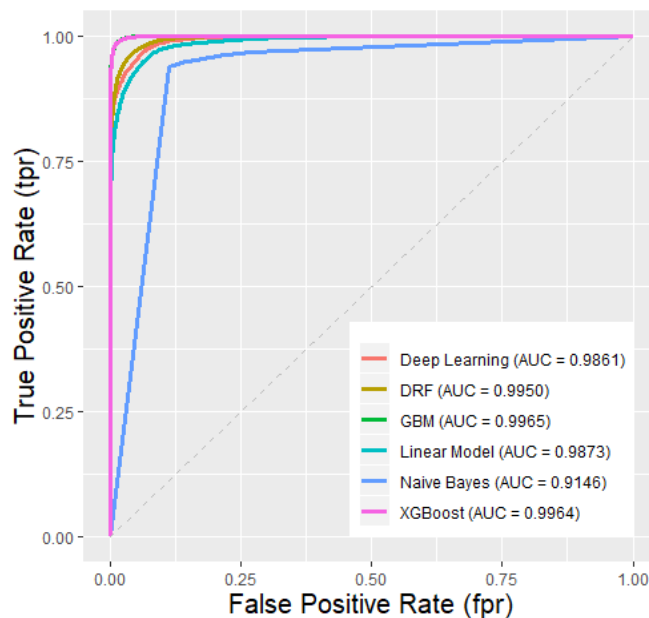
웹 호스팅 서비스 시나리오에서 클라이언트에서 보내지는 트래픽은 웹 서버에 액세스하기 위해 5가지 VNF로 구성된 SFC를 통과하고, 이와 유사하게 로그인 인증 시나리오에서는 클라이언트가 원격에 연결된 로그인 서버로 로그인 요청을 보낸다. 이 때 웹 스트레스 도구 벤치마크 툴인 Apache Bench를 통해 웹 서버에 대한 client side의 HTTP request 트래픽을 생성한다. 실험 방법은 클라이언트에서 테스트베드의 웹 서버에 HTTP request를 보낸다. 해당 HTTP request는 특정 트래픽 패턴을 만들기 위해 서버에 동시에 연결되는 클라이언트의 수를 10 ~ 500 개로, HTTP request 수를 1,000 ~ 50,000 개로 구성하여 보낸다. 이 때 Fault Injection 기법을 활용하여 스트레스 툴 등을 이용해 VNF의 이상 상태를 발생시켰다. 데이터는 상기 서술한 환경에서 모니터링 기능을 통해 데이터를 수집하여 5초 마다 VNF의 상태를 모니터링한다.

실험 시나리오에서 데이터셋은 전체 데이터 중 60%가 정상으로 레이블된 데이터와 40%의 이상으로 레이블된 데이터로 구성된 약 90,000 개의 데이터로 이루어져 있다. 앞서 설명한 바와 같이 SLA 위반 발생에 대한 기준은 [7]에서는 평균 서비스 시간 (응답 시간) 500ms 미만 및 가용성 99.9% 이상으로 정의되어 있지만, 본 문서의 MEC (Multi-Access Edge Computing)을 모사한 테스트베드는 실제 MEC 환경보다 상대적으로 크기가 작기 때문에 보다 강화한 기준을 적용하여 평균 응답 시간 250ms 미만, 가용성 99.95% 이상으로 정의하였다. 이 때 가용성에 대한 부분은 client side에서 생성한 서비스 request 수 대비 돌아오는 정상 응답의 비율을 기준으로 측정하였다. 실험에 사용된 오픈소스 VNF, 모델 학습 환경에 대한 정보는 표 2와 같다. 생성된 모델의 성능 검증은 5-fold cross-validation을 통해 진행하였다.

Name	Version	Purpose
OpenStack	Rocky	Virtualized Infrastructure Manager
Ubuntu	16.04	Base OS of VM
iptables	1.6.0	Firewall VNF
Suricata	5.0.2	IDS VNF
nDPI	3.3.0	DPI VNF
ntopng	2.3.160415	Flow monitor VNF
HAProxy	1.7.0	Load balancer VNF
Apache Bench	2.3.0	HTTP requests generator
Apache2	2.4.0	Web servers
R	3.6.1	Language for data processing/training
H2O	3.31.0	Base machine learning platform

(표 2) 실험 시나리오에 사용된 플랫폼 환경 및 툴 목록

먼저, 각 XGBoost 뿐만 아니라 H2O 플랫폼에서 지원하는 다양한 지도학습 기반 머신러닝 알고리즘 모델들의 성능을 ROC (Receiver Operating Characteristic) curve 를 통해 비교하였다 (그림 7). GBM 및 XGBoost 모델은 0.996 이상의 AUC (Area Under the Curve) 값으로 가장 좋은 성능을 보였다. DRF (Distributed Random Forest) 모델은 0.995로 GBM 및 XGBoost 모델보다 조금 낮은 성능을 보였다. 또한 Deep Learning 및 Linear Model은 상위 3 개 모델보다 정확도가 낮았으며, Naive Bayes 모델은 약 0.914로 가장 낮은 성능을 보였다.



(그림 7) 모델 별 ROC curve 분석

Algorithms	Precision	Recall	F1-Measure	Training Time
Deep Learning	0.9366	0.9265	0.9314	29.47s
DRF	0.9521	0.9537	0.9528	11.46s
GBM	0.9592	0.9647	0.9619	19.96s
XGBoost	0.9581	0.9626	0.9603	5.03s

(표 3) 모델 별 Anomaly Detection 성능 비교표

또한 cross-validation 과정에서 얻은 정밀도 (precision), 재현율 (recall), F1-Measure 및 학습 시간 (training time)을 비교하였다. 표 3은 그림 7에서 높은 성능을 보이는 상위 4 개 알고리즘 기반 VNF Anomaly Detection 모델들의 이진 분류 성능을 나타낸다. 전반적인 모델들의 성능은 F1-Measure 기준 0.9 이상의 정확도를 나타내었다. 그 중 GBM과 XGBoost 기반 모델이 가장 높은 성능을 보여주었으며, DRF와 Deep Learning 기반 모델이 차례로 높은 성능을 보였다. XGBoost 및 GBM 기반 모델의 경우, 0.96 이상의 F1-Measure 값을 보여주었지만 GBM 모델은 XGBoost 모델보다 오랜 학습 시간이 필요한 것으로 나타났다. DRF 기반 모델 또한 높은 정확도를 보여주었지만 Deep Learning 기반 모델은 0.93으로 다른 모델보다 성능이 떨어지는 것을 확인할 수 있다.

Datasets	Measures	Caused in FW	Caused in IDS	Caused in FM	Caused in DPI	Caused in LB	Caused by traffic load
With hop-by-hop Latency	Precision	0.9541	0.9519	0.9711	0.9799	0.9637	0.9708
	Recall	0.9584	0.9375	0.9693	0.9882	0.9744	0.9479
	F1-Measure	0.9562	0.9446	0.9702	0.9841	0.9690	0.9592
Without hop-by-hop Latency	Precision	0.9008	0.9252	0.9102	0.9511	0.9611	0.9306
	Recall	0.9713	0.9487	0.9833	0.9928	0.9860	0.9626
	F1-Measure	0.9347	0.9368	0.9454	0.9715	0.9734	0.9464

(표 4) XGBoost 알고리즘의 Root-Cause Localization 실험 결과

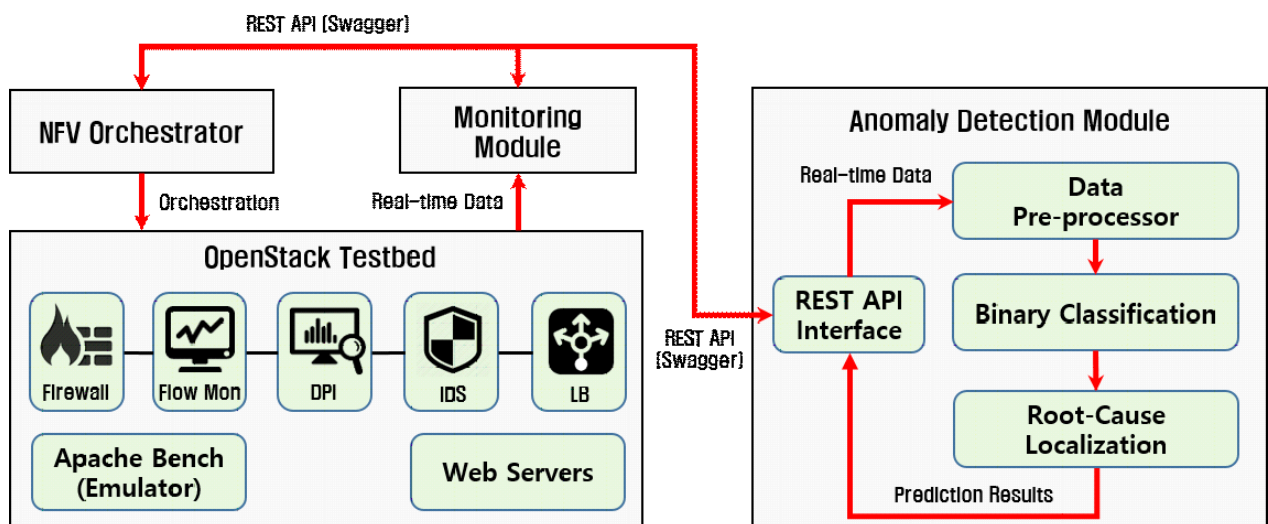
표 3의 결과를 바탕으로 이상을 탐지했을 때, 원인 VNF를 파악하는 Root-Cause Localization 분류 성능을 분석한 결과는 표 4와 같다. 모델은 학습 시간과 성능을 모두 고려하여 가장 좋은 성능을 보이는 XGBoost 기반 모델을 사용하였다. 이진 분류에서 이상 데이터로 분류된 데이터는 그 원인을 SLA 위반을 유발하는 VNF 종류 및 과도한 트래픽 로드로 분류된다. 본 문서에서는 표 4에서 볼 수 있듯이 2가지 종류의 데이터셋을 사용하였다. 데이터셋은 표 1의 feature들 중 variable importance가 가장

높은 network latency (hop-by-hop latency)를 포함하여 학습을 진행한 모델의 성능과 포함하지 않고 학습을 진행한 모델의 성능을 비교하였다. Network latency feature가 포함된 데이터셋은 해당 feature가 포함되지 않은 데이터셋보다 조금 더 높은 성능을 보였다. 그 중 IDS VNF의 이상 예측 결과는 가장 낮은 정확도를 보였고 (0.94), DPI VNF의 경우 가장 높은 정확도 (0.98)를 나타냈다. 그리고 과도한 트래픽로드로 인한 SLA 위반 발생은 약 0.95의 정확도를 보였다.

하지만 해당 모델의 성능을 보다 분석한 결과, 기존 학습시킨 트래픽 패턴과 같거나 유사한 경우 생성된 모델이 이상 상태를 잘 분류하지만 전혀 다르거나 새로운 패턴의 트래픽을 기반으로 모델의 성능을 검증했을 때 그 분류 정확도가 떨어지는 것으로 나타났다. 따라서 향후 후속 연구에서는 이러한 부분을 보완하기 위해 다양한 트래픽 패턴을 학습시키거나 지도학습이 아닌 비지도학습 기반의 알고리즘을 통한 Anomaly Detection 모델에 대해 연구할 예정이다.

2.4 모듈 구현

상기 과정을 통해 생성된 VNF Anomaly Detection 모델을 바탕으로, 본 문서에서는 실제 테스트베드에서 사용가능한 모듈을 만들어 구현하였다. 그림 8은 구현된 VNF Anomaly Detection 모듈의 구조도를 나타낸다.



(그림 8) VNF Anomaly Detection 모듈 구조도

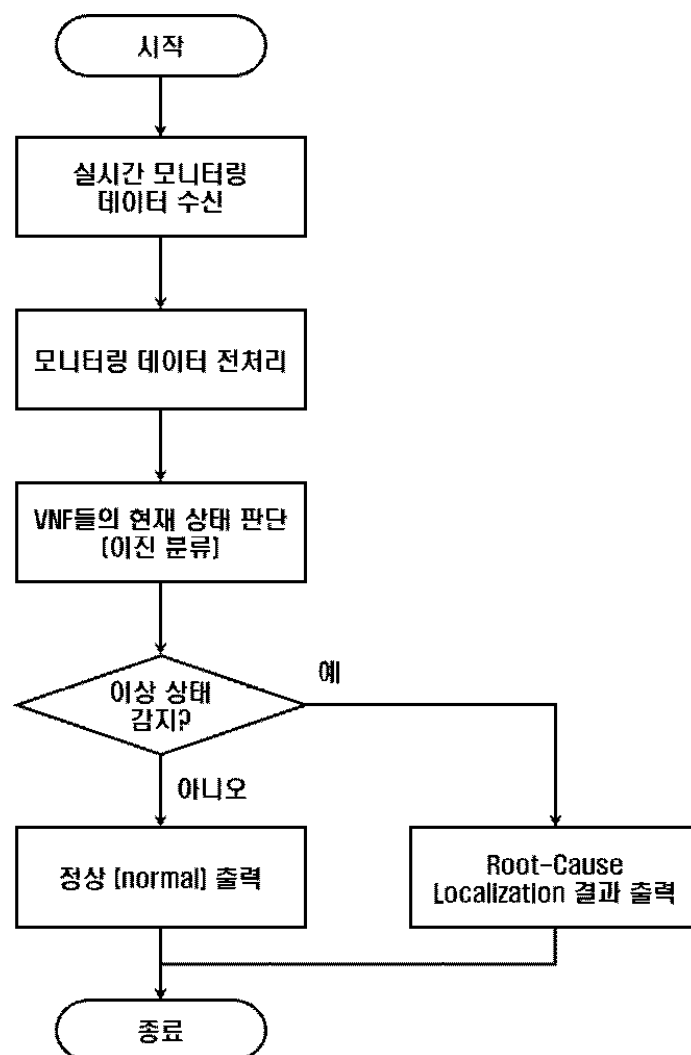
먼저 테스트베드 환경은 상기 설명한 바와 같이 OpenStack을 이용하여 가상 네트워크 환경을 구축하고, 웹 서비스를 운용하기 위한 웹 서버 및 VNF들을 동작시킨다. 가상 네트워크 상에 동작하는 서버 및 VNF들의 정보는 컴퓨터 노드에 설치된 Collectd [9] 모니터링 에이전트를 통해 모니터링 모듈로 전달된다. 모니터링 데이터는 시계열 데이터베이스인 InfluxDB [10]로 전달되며, 본 테스트베드의 모니터링 모듈을 통해 대시보드를 통한 시각화 및 실시간 데이터 전달에 사용된다. 해당 모니터링 모듈 이용에 대한 자세한 정보는 Github 페이지 [11]에서 확인할 수 있다. 모니터링 모듈을 통해 들어온 정보를 통해 NFV Orchestrator는 테스트베드의 상태를 판단하고 그에 따른 orchestration 명령을 내린다.

이 때 본 문서에서 구현한 Anomaly Detection 모듈은 모니터링 모듈로부터 실시간 모니터링 데이터를 Swagger를 활용한 REST API를 통해 데이터를 가져온다. 데이터는 전처리기(Data Pre-processor)를 통해 모델의 입력값으로 활용될 수 있도록 변환되고, 이후 Anomaly Detection 모델은 이 입력값을 기준으로 VNF들의 현재 상태를 먼저 이진 분류를 통해 판별을 한 후, 이상 상태를 탐지했을 경우 Root-Cause Localization 분석을 통해 원인 VNF를 파악하여 그 결과를 출력한다. 이 때 사용자 및 네트워크 관리자는 해당 출력값을 REST API를 통해 그 결과를 전달받을 수 있다. 해당 VNF Anomaly Detection 모듈 구현에 사용된 환경 및 툴들은 표 5와 같다.

Name	Version	Purpose
OpenStack	Rocky	Virtualized Infrastructure Manager
Ubuntu	16.04	Base OS of VM
Python	3.6.9	Module environment
NI-MANO client	1.0.0	MANO client for testbed
NI-Mon client	1.0.0	Monitoring Module client
iptables	1.6.0	Firewall VNF
Suricata	5.0.2	IDS VNF
nDPI	3.3.0	DPI VNF
ntopng	2.3.160415	Flow monitor VNF
HAProxy	1.7.0	Load balancer VNF
Apache Bench	2.3.0	HTTP requests generator
Apache2	2.4.0	Web servers
Swagger	1.0.0	Rest API interface

(표 5) VNF Anomaly Detection 모듈 구현에 사용된 환경 및 툴 목록

이를 바탕으로 구현된 VNF Anomaly Detection 모듈은 현재 VNF의 리소스 과부하 및 SLA 위반 상태 (response time 200 ms 이상, availability 99.99% 미만)를 탐지하여 제공하고 있으며, 추후 모니터링 모듈에 보다 상세한 메트릭들에 대한 수집 기능이 추가되면 현재 모듈을 개선 및 확장할 예정이다. 상기 구현한 모듈의 동작 순서도는 그림 9와 같다. 먼저 모니터링 모듈로부터 실시간 모니터링 데이터를 수신하고, 데이터를 전처리하여 모델의 입력값으로 변환한다. 그 후 Anomaly Detection 모델은 VNF의 상태 판단을 위한 이진 분류를 통해 정상 및 이상 상태를 판단한다. 분류 결과 정상으로 판별이 되면 모니터링 모듈은 동작을 종료하게 되고, 이상으로 판별을 할 경우 Root-Cause Localization 분석을 통해 이상의 원인이 되는 VNF를 파악하여 그 결과를 출력한다.



(그림 9) VNF Anomaly Detection 모듈의 동작 순서도

3.1 환경구축

본 문서에서 다루는 XGBoost 기반 Anomaly Detection 방법은 OpenStack 환경에서 동작할 수 있는 형태로 구현되어 있다. 상기 서술한 바와 같이 구현된 Anomaly Detection 모듈은 OpenStack 기반 테스트베드 내 가상 네트워크에서 존재하는 각 VNF 인스턴스들의 데이터를 활용한다. 이를 위해 OpenStack 기반 테스트베드 환경과 함께 해당 테스트베드 환경과 상호 연동할 수 있도록 NFVO 모듈과 모니터링 모듈이 필요하다. OpenStack 환경 구축을 위한 스크립트 파일 및 설명 [8]과 NFVO 및 모니터링 모듈 설치를 위한 설명 [12]은 Github를 통해 공개되어 있기 때문에 본 문서에서 해당 요소들의 설치 과정은 생략하였다. OpenStack 환경이 구축되고, NFVO와 모니터링 모듈이 설치한 후에는 Anomaly Detection 모듈을 설치해서 실행할 수 있다. 본 문서에서 모듈 설치와 실행은 Ubuntu 16.04에서 수행하였다. Anomaly Detection 모듈은 Python으로 작성되었으며 모듈 안의 Anomaly Detection 모델은 Java로 작성되어 있다. 실행 환경은 Java 11 이상, Python v3.5.2 이상에서 실행하는 것을 권장한다. Anomaly Detection 모듈을 설치하고 실행시키기 위해서는 관련 패키지들을 미리 포함하고 설치해야 한다. Python 환경에서는 패키지의 버전 문제에 따른 호환 문제가 쉽게 발생할 수 있으므로, 다른 기능들과 충돌을 막기 위해, 가상 환경에서 Anomaly Detection 모듈을 설치한다. 가상 환경 생성은 virtualenv를 사용한다. 아래 명령어들을 통해 Python3 패키지 설치 도구인 pip3과 virtualenv를 설치할 수 있다. virtualenv를 설치한 후에는 임의의 가상 환경명을 갖는 독립된 환경을 구축할 수 있다. 가상 환경에서 포함되고 설치되는 Python 관련 패키지들은 다른 환경과는 서로 간섭이 되지 않아 Anomaly Detection 모듈을 위한 패키지들을 안정적으로 불러오고 설치할 수 있다.

```
# Python 패키지 설치 도구 pip3 설치 및 가상환경 구성을 위한 virtualenv 설치
```

```
sudo apt-get update && sudo apt-get -y upgrade
sudo apt-get install python3-pip
sudo pip3 install --upgrade pip
sudo apt install python3-venv
```

```
# virtualenv를 통한 가상 환경 생성 후 활성화 예시
```

```
python3 -m venv .env
source .env/bin/activate
```

```
# 가상 환경 비활성화
```

```
deactivate
```

virtualenv 명령어를 통해 가상 환경을 생성한 후에 활성화하면 CLI 계정 명 왼쪽에 현재 활성화된 가상 환경 명이 표시된다. 그림 10은 .env라는 가상 환경을 생성하고 활성화 한 후에, 비활성화까지 진행한 화면을 보이고 있다.

```
ubuntu@jib-test:~$  
ubuntu@jib-test:~$ python3 -m venv .env  
ubuntu@jib-test:~$ source .env/bin/activate  
(.env) ubuntu@jib-test:~$  
(.env) ubuntu@jib-test:~$  
(.env) ubuntu@jib-test:~$  
(.env) ubuntu@jib-test:~$ deactivate  
ubuntu@jib-test:~$  
ubuntu@jib-test:~$
```

(그림 10) virtualenv를 통한 가상 환경 활성화 및 비활성화 예시

3.2 웹 서버 기반 Anomaly Detection 모듈 설치

상기 과정을 모두 완료한 후, 가상 환경까지 활성화한 후에 Anomaly Detection 모듈 설치를 진행한다. 전체적인 과정은 SFC 및 Auto-Scaling 과 같은 다른 모듈들과 유사하며, Github [13]에 공개 되어있는 Anomaly Detection 모듈을 내려 받고, 실행에 필요한 패키지를 설치한다. 본 모듈을 위한 패키지는 requirements.txt 파일에 정리되어 있으며, 이를 사용하여 패키지를 설치한다. 패키지를 설치 한 후에는 Python 명령어를 통해 본 모듈을 실행한다. 아래 명령어를 순서대로 입력하면 Anomaly Detection 모듈 다운로드 및 필수 패키지 설치와 더불어 모듈 실행까지 수행할 수 있다.

```
# Anomaly Detection 모듈 다운로드 및 필수 패키지 설치  
git clone https://github.com/dpnm-ni/anomaly-detection.git  
cd anomaly-detection  
sudo pip3 install -r requirements.txt  
  
# Anomaly Detection 모듈 실행  
python3 -m swagger_server
```

3.3 웹 서버 기반 Anomaly Detection 모듈 사용법

Anomaly Detection 모듈 실행 명령어를 입력하면 Python-Flask 기반으로 Swagger 웹 서버가 동작한다. Anomaly Detection 모듈은 탐지 요청 리퀘스트를 받으면 리퀘스트가 들어온 시간의 SLA 위반 또는 리소스 과부하 상태를 실시간으로 탐지한다. 또한, SFC를 구성하고 있는 VNF 인스턴스들의 정보와 실시간 리소스 사용량 또한 제공한다. Anomaly Detection 모듈에서 설정되어 있는 기본 웹 서버 포트는 8005이며, 포트 변경을 원할 경우에는 내려 받은 모듈 폴더의 swagger_server/__main__.py 파일에서 그림 11과 같이 port라고 명시된 부분을 수정한다.

```
#!/usr/bin/env python

import connexion

from swagger_server import encoder

def main():
    app = connexion.App(__name__, specification_dir='./swagger/')
    app.app.json_encoder = encoder.JSONEncoder
    app.add_api('swagger.yaml', arguments={'title': 'NI Project Anomaly Detection Module'})
    app.run(port=8005)

if __name__ == '__main__':
    main()
```

(그림 11) Anomaly Detection 모듈의 Port 설정 부분 - swagger_server/__main__.py

또한 본 모듈은 VNF 인스턴스 정보 및 리소스 사용량 등을 수집하기 위해 NFVO 모듈 및 모니터링 모듈과 상호작용한다. 이를 위해 설정 파일 (config/config.yaml) 파일에 각 모듈이 동작하는 host IP 주소와 포트 번호를 설정해야 한다. 해당 코드는 그림 12와 같으며, NFVO 모듈은 ni_nfvo에, 모니터링 모듈은 ni_mon에 각각 정보를 입력한다.

```
ni_mon:
  host: http://<ni_mon_ip>:<ni_mon_port>
ni_nfvo:
  host: http://<ni_nfvo_ip>:<ni_nfvo_port>
```

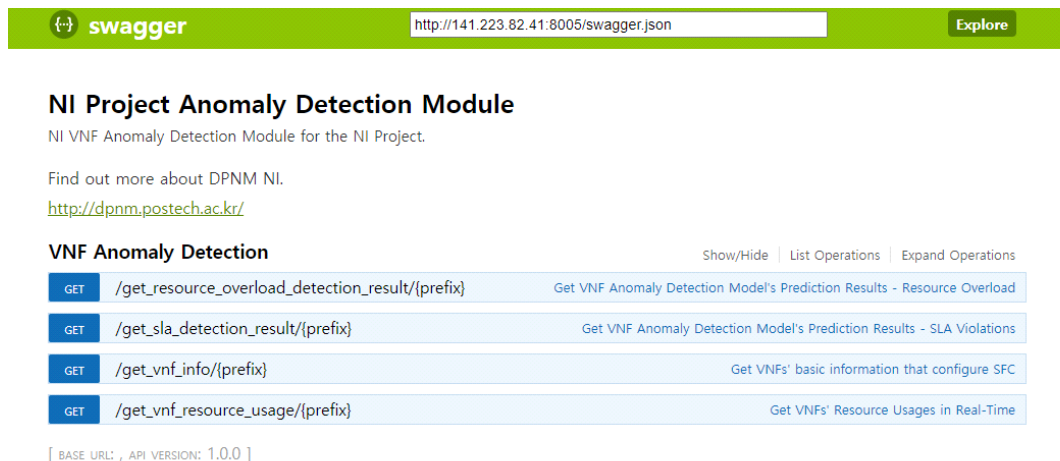
(그림 12) config/config.yaml 설정

설정이 완료되면 실행 명령어 (python3 -m swagger_server)를 입력하여 웹 서버 기반 모듈을 실행한다. 실행된 후의 콘솔 창 화면은 그림 13과 같다.

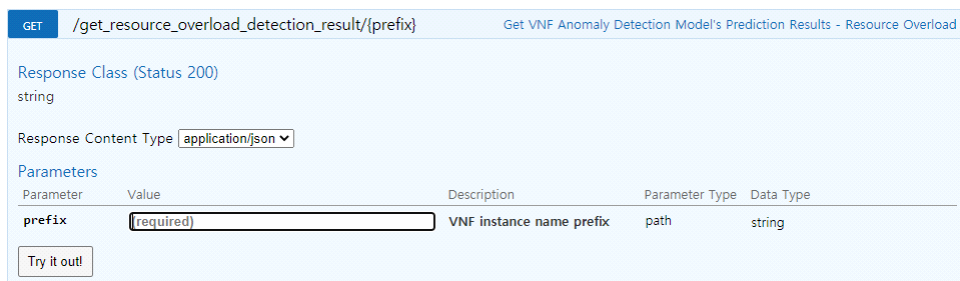
```
(.env) ubuntu@jib-test:~/anomaly-detection$
(.env) ubuntu@jib-test:~/anomaly-detection$ sudo python3 -m swagger_server
* Serving Flask app "__main__" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://0.0.0.0:8005/ (Press CTRL+C to quit)
```

(그림 13) 웹 기반 실시간 Anomaly Detection 모듈 실행

웹 서버 기반 Anomaly Detection 모듈의 실행화면 UI는 그림 14와 같으며, 4개의 기능을 이용할 수 있으며, 각 기능들은 그림 15와 같이 VNF 인스턴스 이름의 prefix를 파라미터로 입력함으로써 동작한다. 현재 제공하는 각 기능은 SFC를 구성하는 VNF들의 인스턴스 기본 정보 (get_vnf_info), VNF들의 실시간 리소스 사용량 (get_vnf_resource_usage), 실시간 VNF들의 과부하 상태 (get_resource_overload_detection_result) 및 SLA 위반 여부 (get_sla_detection_result)로 구성되어 있다.



(그림 14) 웹 기반 Anomaly Detection 모듈의 UI 화면



(그림 15) 웹 기반 Anomaly Detection 모듈의 파라미터 입력 부분

Response Body

```
[
  {
    "flavor_id": "04a63b0b-f59e-46b9-ab76-bd1eb0bb44a4",
    "id": "1893dfe3-f5cd-4b17-80af-24888773dc06",
    "name": "jb_firewall",
    "node_id": 1,
    "ports": [
      {
        "ip_addresses": [
          "10.10.20.111"
        ],
        "network_id": "43bccef5-e205-46ab-b7f4-08bf4c94e1ae",
        "port_id": "7995a1f1-76da-4e57-9c7b-c46cb770d457",
        "port_name": "tap7995a1f1-76"
      },
      {
        "ip_addresses": [
          "10.10.10.196"
        ],
        "network_id": "245d0e8f-c340-4713-a9ee-a3d752482cb5",

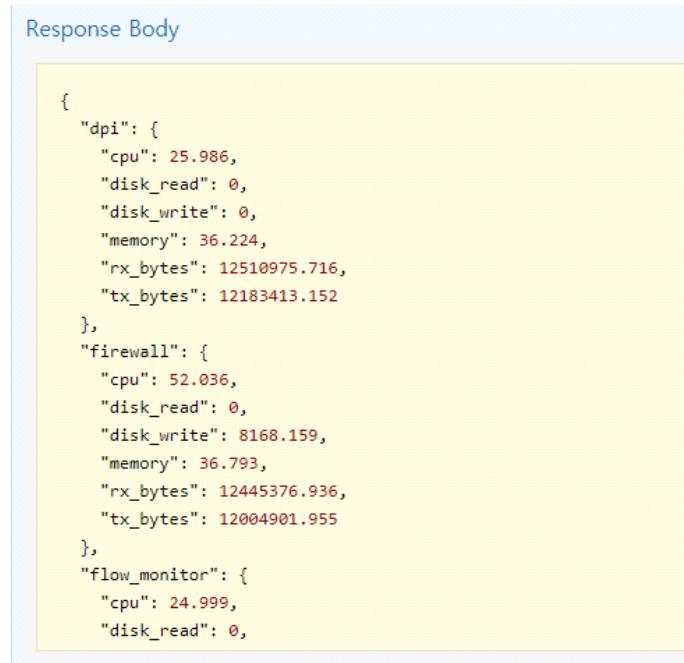
```

```
VNFInstance:
  type: "object"
  properties:
    id:
      type: "string"
    name:
      type: "string"
    status:
      type: "string"
      description: "state of VNF VM. (ACTIVE, SHUTOFF, ERROR, etc.)"
    flavor_id:
      type: "string"
    node_id:
      type: "string"
    ports:
      type: "array"
      items:
        $ref: "#/definitions/NetworkPort"
  example:
    flavor_id: "flavor_id"
    name: "name"
    id: "id"
    ports:
      - port_name: "port_name"
        network_id: "network_id"
        ip_addresses:
          - "ip_addresses"
          - "ip_addresses"
        port_id: "port_id"
      - port_name: "port_name"
        network_id: "network_id"
        ip_addresses:
          - "ip_addresses"
          - "ip_addresses"
        port_id: "port_id"
    status: "status"
    node_id: "node_id"
```

(그림 16) get_vnf_info 기능의 실행 결과 (좌) 및 응답 포맷 (우)

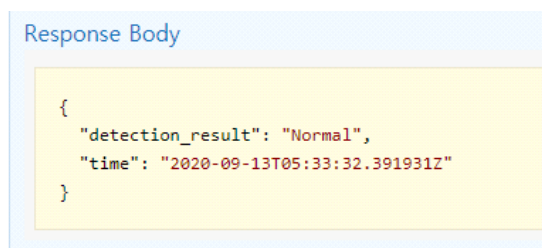
먼저 SFC를 구성하는 VNF들의 기본 정보를 얻는 get_vnf_info는 상기 설명한 바와 같이 SFC를 구성하는 VNF 인스턴스 이름의 prefix를 입력하고 리퀘스트를 보낸다. 이 때 웹 서버는 이에 대한 응답으로 리퀘스트가 들어온 시점에 대한 실시간 탐지 결과를 리턴한다. 응답은 JSON 형식으로 보내지며, 값은 VNFInstance라는 Object로 이루어져 있다. 해당 Object는 VNF의 id, name, flavor_id, port, ip_address 등의 값으로 구성되어 있고, 자세한 정보는 그림 16의 우측 그림에서 확인할 수 있다.

다음으로, VNF들의 실시간 리소스 사용량을 받아오는 get_vnf_resource_usage는 get_vnf_info와 마찬가지로 FC를 구성하는 VNF 인스턴스 이름의 prefix를 입력하고 리퀘스트를 보낸다. 응답 형식은 역시 JSON 형식이고, 값은 그림 17과 같이 VNF 이름, CPU 및 메모리 사용량, disk I/O (read/write), 네트워크 트래픽 양 (rx/tx bytes)로 구성되어 있다. 현재 SFC를 구성하는 VNF의 종류는 고정되어 있는 것으로 간주되어 있지만 향후 다양한 SFC 종류를 지원할 수 있도록 확장 및 개선할 예정이다.



(그림 17) get_vnf_resource_usage 기능의 실행 결과

마지막으로 리소스 과부하 및 SLA 위반 여부를 탐지하는 기능의 경우, 상기 설명한 바와 같이 SFC를 구성하는 VNF 인스턴스 이름의 prefix를 입력하고 리퀘스트를 보낸다. 이 때 웹 서버는 이에 대한 응답으로 리퀘스트가 들어온 시점에 대한 실시간 탐지 결과를 리턴한다. 응답은 JSON 형식으로 보내지며, 값은 detection_result, time 으로 이루어져 있다. 정상 상태의 경우, 그림 18의 좌측과 같이 “Normal” 이라는 응답을 내고, 이상 상태를 감지했을 때는 우측과 같이 “Abnormal” 이라는 응답과 함께 Abnormal의 원인이 되는 VNF를 알려준다.



(그림 18) 리소스 과부하 및 SLA 위반 탐지 기능 실행 결과

리퀘스트가 문제없이 처리되었을 경우, 정상적인 응답 메시지에 기입되는 응답 코드(Response code) 200과 함께, 앞선 그림들과 같이 응답 메시지의 Response Body에 생성된 관련 정보를 받을 수 있다. 만약, 응답 메시지로 오류를 받았을 경우에는 NFVO 및 모니터링 모듈에 문제가 발생해서 정상적으로 VNF 관련 정보를 받아올 수 없거나, 리퀘스트 메시지에 잘못된 형태로 파라미터를 입력한 경우가 대부분이니 각 부분을 확인해야 한다.

3.4 CLI 기반 Anomaly Detection 모듈 실행

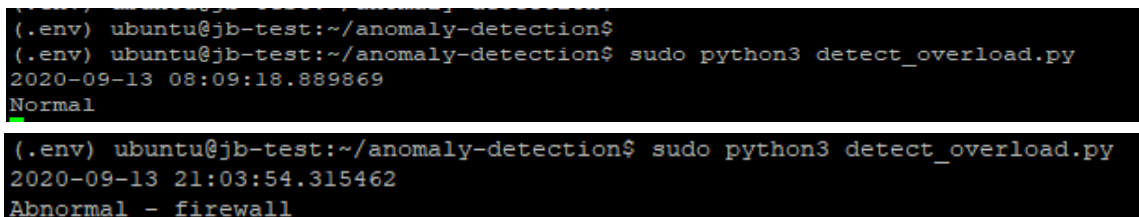
마지막으로 CLI 기반의 환경에서 Anomaly Detection 모듈을 실행하는 방법에 대해 서술한다. 상기 설명한 UI를 이용한 방법 외에 CLI 환경에서는 curl 등을 실행 중인 웹 서버에 직접 request를 보내는 방법이 있다. 해당 방법은 그림 15에 있는 각 기능들의 주소를 직접 입력하여 리퀘스트를 보낼 수 있으므로 자세한 설명은 생략한다. 웹 서버를 실행시키지 않고 Python 프로그램을 이용하여 VNF의 리소스 과부하 상태 및 SLA 위반 여부를 탐지할 수 있다. 실행 명령어는 아래와 같다.

```
# Anomaly Detection 모듈 다운로드 및 필수 패키지 설치
git clone https://github.com/dpnm-ni/anomaly-detection.git
cd anomaly-detection
sudo pip3 install -r requirements.txt

# CLI 기반 Anomaly Detection 모듈 실행 (VNF resource overload)
python3 detect_sla.py

# CLI 기반 Anomaly Detection 모듈 실행 (SLA violations)
python3 detect_overload.py
```

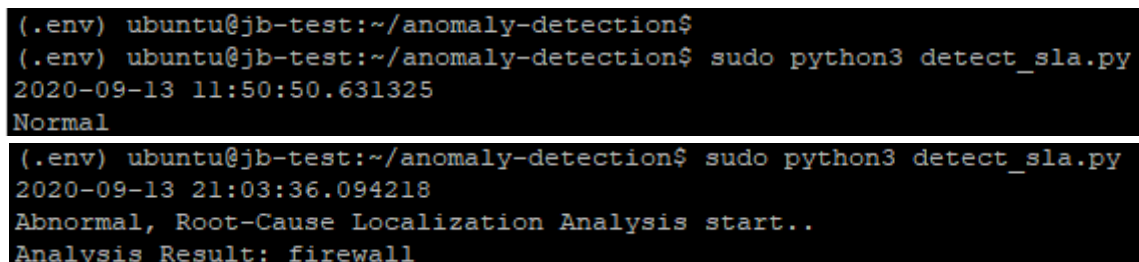
Python 프로그램은 실행 시 1초마다 각 VNF의 리소스 과부하 상태를 탐지하거나 (detect_overload.py) SLA 위반 여부를 탐지한다 (detect_sla.py). 실행을 했을 때의 결과는 그림 19 및 그림 20과 같이 현재 시간 및 탐지 결과로 이루어져 있다.



```
(.env) ubuntu@jib-test:~/anomaly-detection$
(.env) ubuntu@jib-test:~/anomaly-detection$ sudo python3 detect_overload.py
2020-09-13 08:09:18.889869
Normal

(.env) ubuntu@jib-test:~/anomaly-detection$ sudo python3 detect_overload.py
2020-09-13 21:03:54.315462
Abnormal - firewall
```

(그림 19) detect_overload.py 실행 결과: 정상 (상), 이상 탐지 (하)



```
(.env) ubuntu@jib-test:~/anomaly-detection$
(.env) ubuntu@jib-test:~/anomaly-detection$ sudo python3 detect_sla.py
2020-09-13 11:50:50.631325
Normal

(.env) ubuntu@jib-test:~/anomaly-detection$ sudo python3 detect_sla.py
2020-09-13 21:03:36.094218
Abnormal, Root-Cause Localization Analysis start..
Analysis Result: firewall
```

(그림 20) detect_sla.py 실행 결과: 정상 (좌), 이상 탐지 (우)

본 문서에서는 NFV 환경의 SFC에 적용할 수 있는 XGBoost 기반 Anomaly Detection 방법에 대해 서술하였다. 본 문서에서 제안하는 XGBoost 기반 Anomaly Detection 방법은 SFC를 구성하는 각 VNF들의 시스템 자원 사용률, 네트워크 트래픽 양을 실시간으로 모니터링하여 VNF의 현재 상태를 판단하고, 이상이 있을 경우에 이상의 원인이 되는 VNF의 위치를 분석하여 알려준다, 본 문서에서 생성한 Anomaly Detection 모델은 실제 테스트베드인 OpenStack 환경에서 실험을 통해 성능 검증을 수행하여 그 결과 정상과 이상 상태를 탐지하는 이진 분류 (binary classification)에서는 F1-Measure 기준 약 95%의 정확도를 나타내었으며, 원인 VNF를 분석하고 파악하는 Root-Cause Localization (multi-class classification)의 경우 최하 86%의 정확도를 나타내었다. 그 후 상기 모델을 바탕으로 실제 테스트베드에서 동작할 수 있는 모듈을 구현하였다.

향후 연구로는 본 문서에서 서술한 이상 탐지 모델을 확장하여 vIMS 환경 또는 기타 애플리케이션 서비스와 같이 보다 다양한 환경에서의 적용을 검토할 예정이다. 그리고 성능 검증 단계에서 학습된 트래픽 패턴이 아닌 다른 트래픽 패턴을 발생시킬 경우 이상 상태에 대한 분류 정확도가 떨어지는 것으로 분석이 되었기 때문에, 이상 상태를 보다 세밀하게 정의하고 감지하기 위해 지도학습 기반의 알고리즘 뿐만 아니라 비지도학습 기반의 알고리즘의 적용도 검토할 예정이다. 마지막으로 가상 네트워크에서 동작하는 VNF 뿐만이 아니라 물리 노드 및 네트워크 토폴로지의 이상 상태를 탐지하는 모델로 확장할 예정이다.

- [1] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "Open-stack: toward an open-source solution for cloud computing," in *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38-42, Oct. 2012.
- [2] IRTF Network Function Virtualization Research Group, "Network Functions Virtualisation – Update White Paper," [Online]. Available at https://portal.etsi.org/NFV/NFV_White_Paper2.pdf.
- [3] R. Boutaba et al., "A comprehensive survey on machine learning for networking: evolution applications and research opportunities", in *Journal of Internet Services and Applications*, vol. 9, issue 1, 2018.
- [4] ETSI GS NFV-IFA 027, "Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Performance Measurements Specification," 2018.
- [5] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," in *Frontiers in neurorobotics*, vol. 7, pp. 1-21, Dec. 2013.
- [6] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785-794, 2016.
- [7] Grid Resource Allocation Agreement Protocol Working Group, "Web Services Agreement Specification (WS-Agreement)," [Online]. Available at <https://www.ogf.org/documents/GFD.192.pdf>.
- [8] DPNM, "DPNM NI Project Github Repository - ni-testbed," [Online]. Available at <https://github.com/dpnm-ni/ni-testbed-public>
- [9] Collectd, "collectd - the system statistics collection daemon," [Online]. Available at <https://collectd.org/>.
- [10] S. N. Z. Naqvi, S. Yfantidou, E. Zimanyi, "Time series database and influxdb," *Studienarbeit, Universit e Libre de Bruxelles*, 2017.
- [11] DPNM, "DPNM NI Project Github Repository - ni-mon," [Online]. Available at https://github.com/dpnm-ni/ni_mon_client/.
- [12] DPNM, "DPNM NI Project Github Repository - ni-mano", [Online]. Available at <https://github.com/dpnm-ni/ni-mano/>.
- [13] DPNM, "DPNM NI Project Github Repository - anomaly_detection", [Online]. Available at <https://github.com/dpnm-ni/anomaly-detection/>.