공학석사 학위논문

확장 학습 블룸 필터의 효율적인 구현

Efficient Implementation of

Extended Learned Bloom Filter

2021년 8월

서울대학교 대학원

컴퓨터공학부

양 수 현

확장 학습 블룸 필터의 효율적인 구현

지도 교수 김 형 주

이 논문을 공학석사 학위논문으로 제출함
2021년 6월

서울대학교 대학원
컴퓨터공학부
양 수 현

양수현의 공학석사 학위논문을 인준함
2021년 7월

위 원 장 _____ 문 봉 기

부위원장 _____ 김 형 주

위 원 _____ 강 유

초 록

기존의 자료구조는 데이터의 분포와 무관하게 일정한 성능을 가지는 것을 목표로 하고 있다. 하지만 데이터의 분포를 사용한다면 성능을 개선할 수 있다는 연구가 학습 인덱스라는 이름으로 진행되고 있다.

본 연구에서는 학습 인덱스의 종류 중 하나인 학습 블룸 필터를 확장하고 구현하는데 초점을 둔다. 이를 확장 학습 블룸 필터라고 부르며, 이는 기존의 학습 블룸 필터의 구조에 학습 해시 함수를 추가한 자료구조이다. 해당 자료구조는 하이퍼파라미터 α를 통해서 학습 해시 함수와 보조 필터의 비율을 조정할 수 있으며, 기존의 학습 블룸 필터에 비해서 거짓 양성 비율을 개선시킬 수 있음을 실험을 통해 보인다.

추가적으로 확장 학습 블룸 필터를 구현하는 도중에 발생했던 모델의 정밀도 문제를 소개하고, 이를 64비트 부동소수점으로 해결할 수 있음을 보인다. 그 외에도 모델 조정을 통해서 확장 학습 블룸 필터의 성능이 개선될 수 있음을 보이고, 학습 해시 함수가 성능 개선에 기여하는 방법을 이해하고자 한다.

주요어 : 학습 블룸 필터, 학습 해시 함수, 학습 인덱스
학 번 : 2019-25475

목 차

표 목 차

그 림 목 차

제 1 장 서 론

최근에 인공신경망(Neural network)의 비약적인 성능향상으로 컴퓨터 공학 관련해서 하드웨어부터 소프트웨어까지 다양한 분야에서 인공신경망을 접목하는 연구들이 진행되고 있다. 인공신경망이 대표적으로 사용되는 분야는 컴퓨터 시각(Computer vision), 자연어 처리(Natural language processing) 그리고 음성 인식(Speech recognition)으로, 위의 문제를 기계학습(Machine learning)을 통해서 해결하기 위해서는 대용량의 데이터가 필요하다. 대용량의 데이터를 저장하고 분석하기 위해서 하둡(Hadoop)이라는 분산 파일 시스템이 등장하였고, 맵리듀스(Map reduce)방식을 사용하여 인공신경망이 필요로 하는 데이터를 정제해서 전달하는 과정을 가진다. 이와 같이 인공신경망을 동작시키기 위해서는 데이터를 효율적으로 저장하고 질의하는 시스템을 필요로 한다.

데이터를 효율적으로 저장하고 질의하는 방식은 데이터베이스의 요구사항 중 하나로 인덱스(Index)라는 이름으로 불린다. 대표적인 인덱스는 B-트리(B-Tree) 자료구조로 관계형 데이터베이스(Relational database)에서 질의(Query)의 시간을 단축시키는 역할을 수행한다. 그 외에도, LSM Tree(Log-Structured Merge-Tree)에서 저장된 데이터의 존재 여부를 확인하기 위해서 블룸 필터(Bloom filter)를 인덱스로 사용한다.

블룸 필터[1]는 데이터베이스, 네트워크, 컴퓨터 보안 등의 다양한 분야에서 사용되는 자료구조이다. 블룸 필터는 데이터를 비트 형식으로 인코딩하여 적은 양의 공간을 사용함과 동시에 데이터의 존재 유무를 판단할 수 있기 때문에 시스템의 전반적인 속도를 향상시킬 목적으로 사용된다. 대용량의 데이터를 빠르게 처리하기 위해서는 블룸 필터를 사용하거나, 블룸 필터를 대체할 수 있는 새로운 자료구조를 사용해야 한다.

The case for Learned Index Structures[2]에서는 기존에 사용되는 인덱스인 블룸 필터에 모델을 추가적으로 사용할 것을 제안한다. 이를 학습 블룸 필터(Learned bloom filter)라고 부르며, 기존과 동일한 공간을 사용하는 경우 개선된 거짓 양성 비율(False positive rate)을

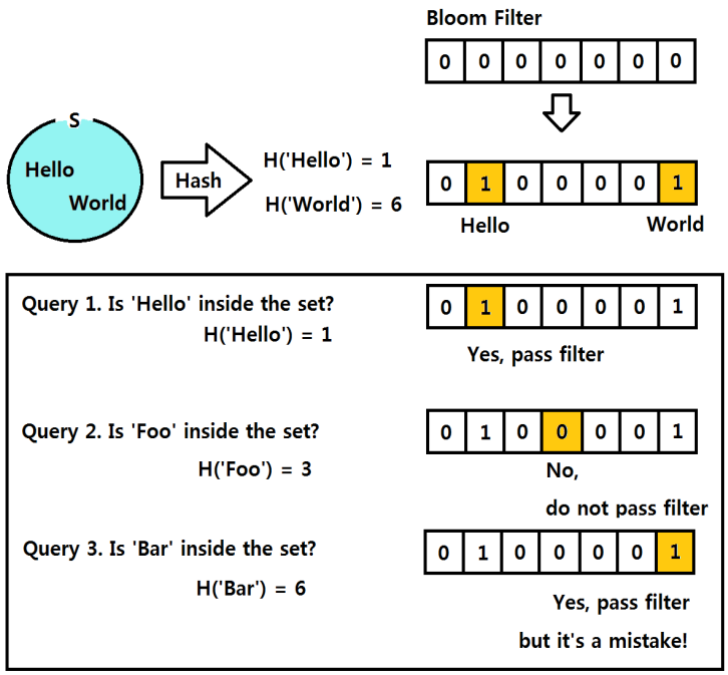
가진다. 대용량의 데이터를 기존의 블룸 필터를 처리하기 위해서는 선형적으로 공간 사용이 늘어나는 단점이 있다. 반면에 학습 블룸 필터를 사용한다면 이보다 효율적인 공간 사용으로 거짓 양성 비율을 개선하는데 기여할 수 있을 것으로 판단된다.

본 연구에서는 학습 블룸 필터의 변형인 확장 학습 블룸 필터(Extended learned bloom filter)를 소개한다. 이는 학습 불룸 필터에 학습 해시 함수(Learned hash function)를 추가적으로 적용한 자료구조이다. 확장 학습 블룸 필터는 학습 블룸 필터에 비해서 개선된 성능을 가지는 것을 실험적으로 보였으며, 확장 학습 블룸 필터를 구현하는데 발생했던 문제점들에 대해서도 살펴보도록 한다.

제 2 장 배경 지식

이 장에서는 확장 학습 블룸 필터를 이해하기 위한 배경 지식인 블룸 필터와 학습 블룸 필터에 대해 설명한다. 1절에서는 블룸 필터에 대한 개념을 설명하고, 2절에서는 블룸 필터가 사용되는 어플리케이션에 대해서 살펴본다. 3절에서는 학습 블룸 필터에 대한 개념을 설명하고, 4절에서는 학습 블룸 필터가 사용되는 어플리케이션에 대해서 살펴보고자 한다.

2.1 블룸 필터의 개념



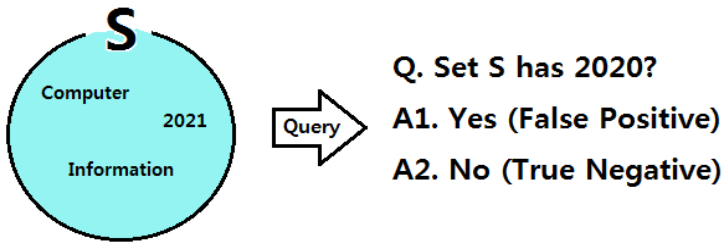
[그림 1] 블룸 필터의 개요

블룸 필터는 집합(Set)을 표현하기 위한 자료구조이다. 예를 들어, 집합에 { “hello”, “world” }라는 값이 있다면 해당 원소에 대해서는 참을 반환하고 그 외의 원소에 대해서는 거짓을 반환하는 것을 목표로 한다. 하지만, 블룸 필터는 공간을 작게 사용하는 이유로 가끔 오류가 발생하게 된다. 해당 오류를 거짓 양성(False positive)이라고 부르며, 참을 반환했으나 해당 정보가 거짓이었다는 것을 의미한다. 즉, 위의

예제를 계속해서 사용한다면, “bar”에 대해서 참이 나온다면 이를 거짓 양성이라고 볼 수 있다. 위와 같이 거짓 양성이라는 치명적인 오류가 있음에도 불구하고 블룸 필터를 사용하는 이유는 거짓 음성(False negative)이 존재하지 않기 때문이다. 즉, “hello”와 “world”에 대해서는 어떠한 경우에도 참을 반환한다는 것이 보장된다는 뜻이다.

블룸 필터는 해시 함수(Hash function), 집합의 크기, 비트 벡터(Bit vector), 거짓 양성 확률(False positive probability)로 총 4가지의 요소로 구성되어 있다.

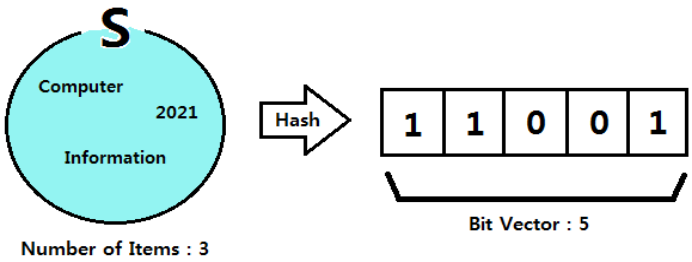
해시 함수는 블룸 필터에서 집합에 포함되는 여부를 표현하기 위해서 사용된다. 해시 함수의 특징으로는 결정성(Deterministic)을 만족한다는 것이다. 따라서, 어떠한 입력값이 들어오면, 해당 입력값에 대해서는 항상 동일한 출력값을 가지게 된다. 예를 들어 “hello”를 해시 함수의 입력으로 준 경우 출력값이 1이였다면, 항상 “hello”에 대해서는 1이라는 출력값이 보장된다는 의미이다. 블룸 필터에 입력되는 원소를 저장하기 위해서는 1개 또는 그 이상의 해시 함수를 사용한다. 1개의 해시 함수로도 충분히 원소의 존재 유무를 판단할 수 있지만, 2개 이상의 해시 함수를 이용하는 것은 거짓 양성 확률을 개선하기 위함이다.



[그림 2] 블룸 필터의 거짓 양성 오류

거짓 양성 확률은 블룸 필터 안에 발생하는 거짓 양성의 비율을 결정하는 요소이다. 즉, 거짓 양성 확률을 0.1으로 음성 데이터에 대해서만 질의를 10,000회 수행되었다고 가정하자. 1,000회의 질의에 대해서는 거짓 양성이 발생하고 나머지 9,000회의 질의에 대해서는 참 음성(True negative)이 발생한다. 거짓 양성 확률은 변수이며 사용자가 결정하거나 해시 함수의 개수와 비트 벡터의 크기를 통해서 결정할 수 있다. 해시 함수의 개수가 늘어날수록 원소를 구분하는데 많은 비트를 읽어서 거짓 양성 횟수를 줄일 수 있다. 추가적으로, 비트 벡터의 크기를 늘린다는 것은 서로 다른 값을 가지고 있는 원소 간에 동일한 해시 함수 출력 값을 가지는 경우를 줄이는 것을 의미한다. 결과적으로

비트 벡터의 크기를 늘리는 것 또한 거짓 양성 횟수를 줄이는데 사용할 수 있다.



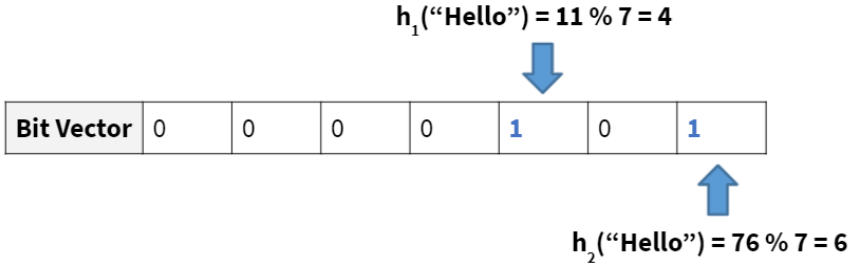
[그림 3] 블룸 필터의 비트 벡터와 집합의 크기

비트 벡터는 블룸 필터에서 데이터를 저장하는 공간이다. 비트는 0과 1로 구성되어 있고, 컴퓨터에서 표현할 수 있는 가장 작은 단위이다. 비트를 연속적으로 나열한 것을 비트 벡터라고 말한다. 즉, 0 또는 1로 구성된 배열이 있다고 생각하면 될 것이다. 블룸 필터에서 특이한 점은 데이터를 저장할 때, 데이터의 존재 여부에 대해서만 저장한다는 점이다. 예를 들어 “hello”라는 원소를 저장할 때, 해시 함수가 “hello”에 대해서 반환한 출력값을 인덱스로 사용한다. 비트 벡터의 인덱스 값을 0에서 1로 갱신하여 “hello”데이터의 존재함을 나타낸다. 단, 비트 벡터의 인덱스 값은 다른 원소가 저장되어 1인 상태일 수도 있다. 이 경우에도 동일하게 1로 해당 비트를 덮어쓴다. 데이터의 원본을 저장하는 것이 아닌 데이터의 존재 여부를 저장하는 특징으로 인하여, 데이터의 집합을 작은 크기로 표현할 수 있게 된다.

집합의 크기는 블룸 필터에 저장하고자 하는 집합의 원소 개수를 말한다. 집합의 크기가 중요한 이유는 거짓 양성 확률과 연관이 되어 있기 때문이다. 즉, 비트 벡터의 크기에 비해서 너무 많은 원소를 저장하게 된다면 모든 질의에 대해서 거짓 양성을 반환하는 문제가 발생할 수 있다.

다음은 블룸 필터의 초기화, 입력 그리고 질의에 대해서 알아보도록 한다. 블룸 필터의 초기화를 위해서는 집합의 크기와 거짓 양성 확률에 대해서 알고 있어야 한다. 집합의 크기는 입력하고자 하는 데이터의 개수를 세면 되고, 거짓 양성 확률은 사용자가 필요로 하는 성능에 알맞게 확률을 설정하도록 한다. 대부분의 어플리케이션에서는 거짓 양성 확률을 0.01로 설정하여 1%의 거짓 양성 비율을 갖도록 설정한다. 블룸 필터의 초기화 예시로는 집합의 크기(n)가 2이고, 거짓 양성 확률(p)이 0.2라고 가정한다. 블룸 필터의 초기화를 위한 비트 벡터의

크기를 구하는 공식은 $\lceil -nlog(p) / (log2)^2 \rceil$ 이고, 해당 공식에서 사용하는 n과 p 변수는 위에서 각각 2와 0.2로 설정하였다. 이를 대입하면 비트 벡터의 크기를 7로 설정했을 때, 집합의 크기 2와 거짓 양성 확률 0.2을 모두 만족한다는 것을 의미한다. 이제는 집합의 크기(n), 거짓 양성 확률(p) 그리고 비트 벡터의 크기(m)을 알고 있으므로, 최적의 해시 함수의 개수(k) 구하는 공식을 사용할 수 있게 된다. 최적의 해시 함수의 개수를 구하는 공식은 $(m/n) * log(p)$ 로, 위에서 구해진 m, n 그리고 p의 값을 각각 7, 2, 0.2로 대입한다. 이를 통해서 해시 함수를 2개 사용하는 것이 목표로 하는 거짓 양성 확률을 만족하는 최적의 해시 함수의 개수임을 알 수 있다. 블룸 필터를 구성하는 4개의 변수가 확정되었다면, 크기 7의 비트 벡터를 모두 0의 값으로 메모리에 할당하고, 사용할 해시 함수 2개도 준비해 놓는다. 해시 함수는 본 논문에서는 Murmurhash3를 사용하였고, 2개의 해시 함수는 Murmurhash3에 2개의 다른 시드(seed)값을 입력하여 사용하였다. Murmurhash3 외에도 xxHash를 해시 함수로 사용할 수도 있으며, 각 해시 함수가 수용하는 범위가 32, 64, 128비트로 별도로 존재하므로 주의해야 한다.



[그림 4] 블룸 필터의 입력

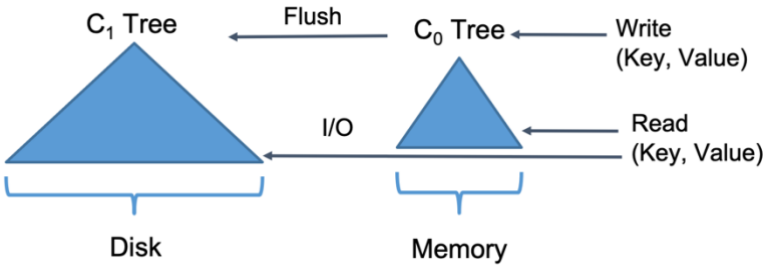
블룸 필터의 입력은 k개의 해시 함수를 사용한다. 입력 시 사용되는 공식은 $h_i(x)$ % m이다. 즉, x라는 원소를 입력하기 위해서는 해시 함수에 x를 입력하여 받은 출력값을 비트 벡터의 인덱스로 매핑한다는 뜻이다. 위의 공식으로 구해진 인덱스의 비트 벡터값을 1로 수정하면 입력이 완료된다. 입력을 통해서 서로 값이 다른 원소 간에 동일한 인덱스로 매핑되는 경우가 존재하며, 이로 인해서 거짓 양성이 발생한다. 예를 들어, "hello"에 대한 입력이 위에서 생성한 크기 7인 블룸 필터에 들어간다고 가정해보자. 만약에 h_1("hello") = 11이라고 가정하면, 11 % 7로 인덱스를 구할 수 있다. 따라서, 첫번째 해시 함수는 4라는 인덱스와 연결된다. 이와 유사하게 h_2("hello") = 76이라면, 76 % 7로 두번째 해시 함수는 6이라는 인덱스와 연결된다. 즉, "hello"를

입력하기 위해서는 비트 벡터의 4번째와 6번째 인덱스를 1로 갱신해주면 된다.

블룸 필터의 질의는 입력과 유사하게 진행된다. 단, 비트 벡터의 값을 1로 갱신하는 것이 아닌 비트 벡터의 값이 1인지 확인하는 과정을 가진다. 질의 과정에서 가장 중요한 것은 거짓 양성이 발생할 수 있다는 것이다. 즉, 블룸 필터에 입력하지 않은 원소가 입력되었다고 착각하는 경우가 발생한다. 이는 비트 벡터의 크기를 제한적으로 사용해서 발생하는 현상이며, 블룸 필터는 오류를 허용하는 어플리케이션에서 사용해야 한다. 추가적으로, 블룸 필터에서는 거짓 부정은 존재하지 않는다는 특징이 있다. 즉, 블룸 필터에 입력된 모든 원소에 대해서는 원소가 존재함을 반드시 나타낸다는 뜻이다. 예를 들어, 블룸 필터의 입력 예시에서부터 질의를 수행한다고 가정한다. 해시 함수는 결정성을 만족하기 때문에 “hello”라는 값은 동일하게 첫번째 해시 함수는 인덱스 4 그리고 두번째 해시 함수는 인덱스 6을 가리킬 것이다. 따라서, 4번째와 6번째 인덱스를 확인하고 모두 1인 상태를 확인할 수 있다. 단, “bar”라는 블룸 필터에 입력되지 않은 원소가 질의되는 경우도 생각해보자. “bar”의 첫번째 해시 함수가 인덱스 6을 가리키고, 두번째 해시 함수가 인덱스 4를 가리킨다고 가정한다. 위의 가정에 의해서 “bar”는 블룸 필터에 입력되지 않았음에도 불구하고, 존재한다고 판별될 것이다. 이를 거짓 양성이라고 부르고 블룸 필터의 성능 향상은 대부분 거짓 양성 횟수를 줄이는데 초점을 둔다.

2.2 블룸 필터의 어플리케이션

블룸 필터는 암호 감지, 인터넷 캐시 프로토콜, 비트코인의 지갑 동기화, 해시 기반 IP 추적, 바이러스 스캔과 같은 사이버 보안 등 다양한 분야에서 사용되고 있다. 본 논문에서는 로그 구조 병합 트리 [3]와 HTTPS에 대한 블룸 필터의 사용에 대해서 살펴보도록 한다.



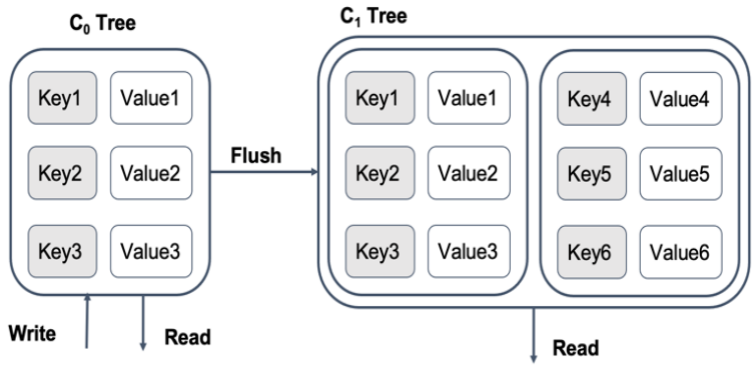
[그림 5] LSM Tree의 구조

LSM Tree는 입력에 대해서 최적화된 자료구조이다. 기존에는 질의에 최적화 되어있는 B-Tree의 사용으로 입력을 위해서는 항상 O(logn)이라는 비용을 부담해야 한다. 반면에 LSM Tree는 로그 형식의 입력 방식으로 위의 불필요한 O(logn) 비용을 제거할 수 있다. 로그 형식의 입력은 데이터를 입력 순서대로 저장하여 O(1)이라는 시간을 소요하며 입력을 수행할 수 있다. 따라서, 입력이 많은 데이터에 대해서는 LSM Tree형식의 데이터베이스를 사용하고 질의가 많은 데이터에 대해서는 B-Tree형식의 데이터베이스를 사용할 수 있을 것이다.

LSM Tree는 키-값(Key-Value)형식의 데이터를 담는데 사용된다. LSM Tree의 구조는 메모리에 존재하는 트리(C_0 Tree)와 디스크에 존재하는 트리(C_1 Tree)로 구성된다. 메모리에 존재하는 트리의 실제 구현은 B-Tree 또는 스킵 리스트(Skiplist)를 사용할 수 있다. B-Tree는 트리의 균형을 맞춰서 O(logn)의 성능을 가지도록 하는 자료구조이고, 스킵 리스트는 연결 리스트(Linked list)를 계층적으로 사용하여 B-Tree와 유사한 성능을 가지는 자료구조이다. 단, 스킵 리스트가 B-Tree와 유사한 성능을 가지는데도 불구하고 사용되는 이유는 구현의 간단함 때문이다. 메모리에 존재하는 트리에 데이터를 입력하게 되면, 데이터는 곧바로 디스크에 전달되지 않는다. 메모리에 있는 데이터가 디스크로 이동하는 시점은 사용자가 설정한 임계치(Threshold)에 따라서 결정된다. 메모리에 존재하는 트리의 크기가 임계치를 넘게 되면 메모리에 있는 트리 전체를 디스크에 있는 트리로 이동시키고, 메모리에 있는 트리를 초기화한다.

디스크에 있는 트리는 반드시 메모리에 있는 트리를 통해서 입력이 수행된다. 디스크에 있는 트리의 실제 구현은 정렬된 문자열 테이블(Sorted String Table)을 사용한다. 정렬된 문자열 테이블은 키-값 형태의 데이터를 저장하는 자료구조로서, 키-값 형태의 데이터가 들어온 경우 키를 기준으로 정렬하면서 입력을 수행하게 된다. 즉, 해당 자료구조는 데이터를 키 기준으로 정렬한다는 특징을 가지고 있다. 이 외에도 불변성(Immutable)이라는 특징을 가지는데, 이는 한번 입력된 키-값의 데이터는 수정되지 않는 것을 의미한다. 불변성으로 인하여 디스크에 있는 트리에는 중복된 키가 존재하게 되는데, 데이터베이스에서 중복되는 데이터는 공간의 낭비를 의미한다. 따라서, LSM Tree에서는 키의 중복을 막고자 병합(Merge)라는 연산을

제공한다.

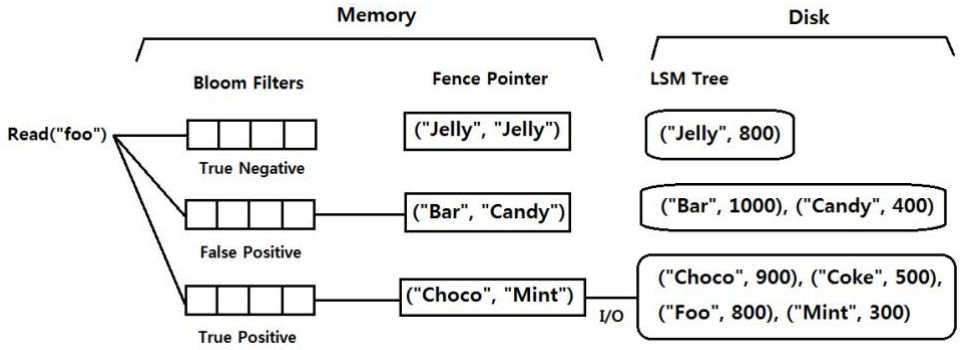


[그림 6] LSM Tree의 읽기와 쓰기

LSM Tree에서는 읽기(Read), 쓰기(Write), 병합에 대한 연산이 존재한다. 읽기를 수행할 때는 특정 키가 메모리에 있는 트리에 존재하는지를 확인한다. 메모리에 있는 트리에 해당 값이 있다면, 이를 사용하면 된다. 이와 반대로 메모리에 있는 트리에 해당 값이 없는 경우, 디스크에 있는 트리 전체를 확인해서 가장 최근의 타임스탬프(Timestamp)를 갖고 있는 키를 반환해야 한다. 쓰기를 수행할 때는 메모리에 있는 트리에 해당 키를 입력한다. 입력 도중에 메모리에 있는 트리의 크기가 사용자가 설정한 임계치 이상의 크기가 되면 메모리에 있는 트리를 디스크에 있는 트리와 함께 합친다. 즉, 메모리에 있는 B-Tree 또는 스킵 리스트를 하나의 정렬된 문자열 테이블로 변환해서 디스크에 저장된다고 볼 수 있다. 마지막으로, 병합은 디스크에 존재하는 여러 개의 정렬된 문자열 테이블을 하나의 정렬된 문자열 테이블로 만드는 행위이다. 병합 연산은 병합 정렬(Merge sort)과 유사하게 진행되는데, 각 정렬된 문자열 테이블을 순회하는 반복자(Iterator)를 사용해서 가장 작은 키 값부터 순차적으로 입력해서 새로운 정렬된 문자열 테이블을 생성한다. 병합 연산으로 인해서 LSM Tree의 중복된 원소를 제거하면, 전체적인 트리의 크기가 감소하여 질의하는 시간이 단축된다.

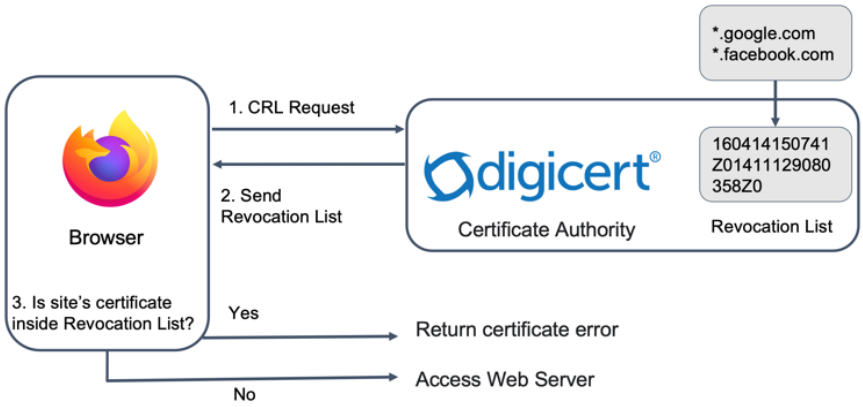
Monkey[4]는 LSM Tree의 변형이다. 해당 어플리케이션에서는 위의 LSM Tree와는 다르게 디스크에 존재하는 트리가 하나 존재하는 것이 아닌 여러 개의 트리로 존재한다. 즉, 하나의 큰 트리를 여러 개의 작은 트리로 만들어서 읽기 성능을 향상시키는 구조를 가진다. 위의 변화와 별도로 읽기 성능을 향상시키기 위해서는 각 트리의 키 존재 여부를 확인해야 하는데, 해당 부분에 블룸 필터를 사용한다. Monkey는

디스크에 존재하는 하나의 트리마다 하나의 블룸 필터를 갖도록 설계되어 있다. 각 블룸 필터는 각 트리의 키를 이용하여 초기화되며, 읽기를 수행하는 경우에는 모든 트리를 조회하는 것이 아닌 블룸 필터가 키가 존재한다고 명시한 트리에 대해서만 읽기를 수행한다. 결과적으로 전체 트리를 읽는 것이 아닌 일부 트리에 대해서만 읽기를 수행하게 되어 어플리케이션의 성능이 향상된다.



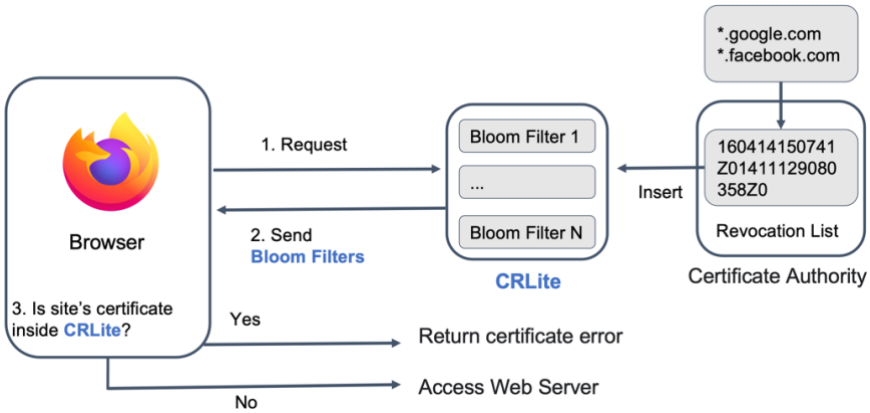
[그림 7] Monkey에서의 블룸 필터 사용

HTTPS는 HTTP의 보안 문제를 해결하기 위해서 사용하는 네트워크 프로토콜이다. 예를 들어, 브라우저에서 HTTP 프로토콜을 통해서 서버와 클라이언트간의 통신을 수행하게 되면 중간자 공격(Man in the middle attack)에 취약할 수 있다. 즉, 네트워크상에 존재하는 제3자에 의해서 비밀번호와 같은 개인 정보 유출이 될 수 있다. 클라이언트에서 HTTPS를 사용하기 위해서는 인증 기관(Certificate authority)을 통해서 접근하고자 하는 도메인의 인증서를 전달 받아야 한다. 인증서에는 도메인 서버의 공개키(Public key)가 있으며, 클라이언트는 해당 공개키를 사용하여 암호화(Encryption)된 요청을 서버에게 보낼 수 있다. 단, 클라이언트에게 인증서를 안전하게 전달하기 위해서는 인증 기관이 신뢰할 수 있는 기관이여야 한다는 조건을 만족해야한다. 이는 인증 기관이 신뢰할 수 없는 인증 기관인 경우, 기관 자체에서 중간자 공격을 할 수 있기 때문이다. 인증서는 인증서의 유효기간이 지난 경우 또는 도메인 서버의 해킹 공격으로 폐기될 수 있다. 인증서를 폐기하는 가장 기본적인 방식은 인증서 폐기 목록(Certificate Revocation List)을 클라이언트가 전달 받아서 인증서 정보를 갱신하는 작업을 수행하면 된다. 단, 해당 과정에서 인증서 폐기 목록의 크기가 크다는 문제점이 존재한다.



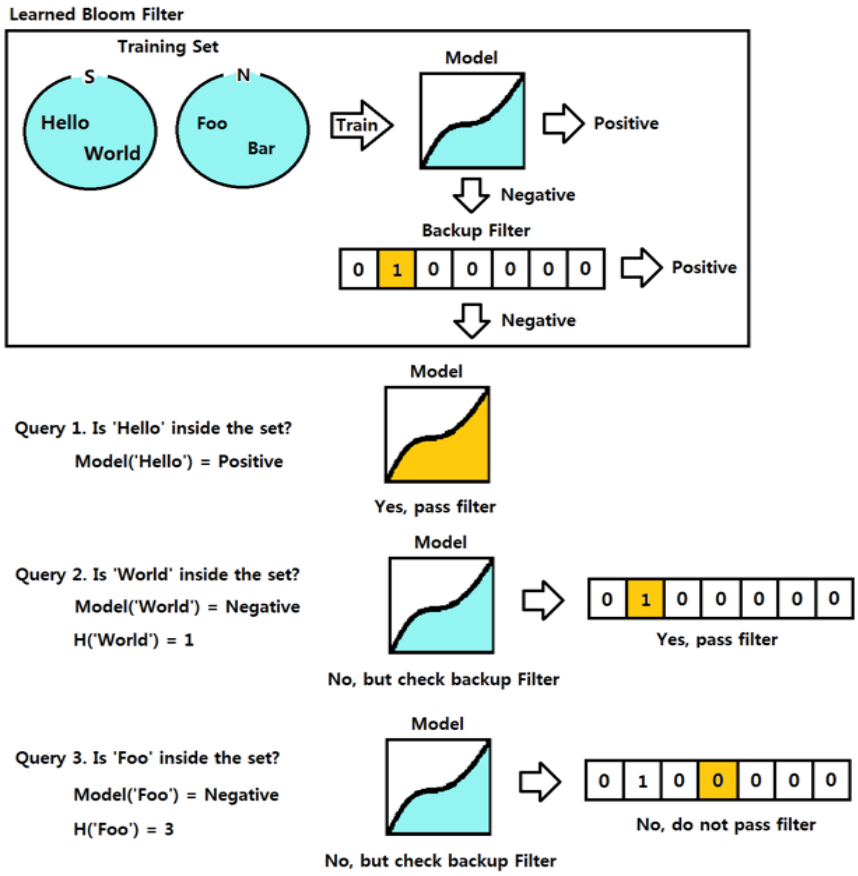
[그림 8] 인증서 폐기 목록의 진행 순서

CRLite[5]는 HTTPS를 사용하는 경우 블룸 필터를 통해서 인증서를 확인할 것을 제안한다. 기존의 인증서 폐기 목록 방식은 도메인 서버의 정보를 암호화해서 클라이언트에 전달하지만, CRLite는 인증서의 폐기 유무를 확인하는 블룸 필터를 클라이언트에게 전달하는 구조를 가진다. 즉, 전체 인증서 정보가 아닌 인증서의 폐기 여부를 전달하기 때문에 클라이언트에게 전달해야 하는 데이터의 양이 기존에 비해서 작아진다는 이점이 있다. 하지만 블룸 필터에는 거짓 양성이라는 오류가 발생할 가능성이 존재하므로, 일반적인 블룸 필터로는 인증서의 폐기 여부를 정확하게 확인할 수 없다. 따라서, CRLite는 여러 개의 블룸 필터를 계층적으로 사용하여 거짓 양성이 존재하지 않을 때까지 블룸 필터를 생성할 것을 제안한다. 또한 CRLite는 여러 개의 블룸 필터를 사용했음에도 불구하고, 기존의 인증서 폐기 목록보다 공간을 적게 사용하여 실용적인 어플리케이션이라 볼 수 있다. CRLite는 현재 파이어폭스(Firefox) 브라우저에 적용되어 HTTPS의 인증서 목록을 확인하는데 사용되고 있다.



[그림 9] CRLite의 진행 순서

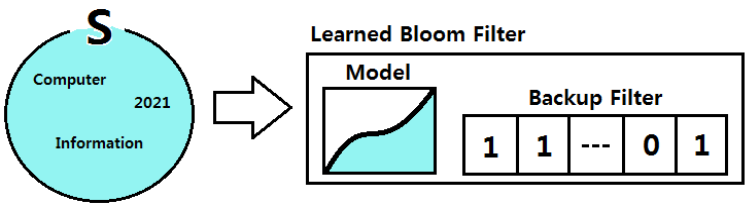
2.3 학습 블룸 필터



[그림 10] 학습 블룸 필터의 개요

학습 블룸 필터는 The Case for Learned Index Structures[2]에 의해서 제안된 자료구조이다. 블룸 필터는 원소가 주어지면, 해당 원소가 집합에 속하는지 여부를 반환하는 자료구조이다. 이를 데이터 과학 관점에서 해석하면, 원소가 집합에 존재하는 여부를 이진 분류기 모델을 통해서도 식별할 수 있음을 알 수 있을 것이다. 단, 블룸 필터를 사용하면 거짓 음성이 존재하지 않는데 반해, 모델을 사용하면 거짓 음성이 발생할 수 있다. 따라서, 모델에서 분류하지 못한 거짓 음성을 처리하기 위해서 블룸 필터를 추가적으로 사용하고 이를 보조 필터라고 부른다. 블룸 필터는 거짓 음성이 나오지 않는 특징이 있기 때문에, 결과적으로 모델과 블룸 필터를 함께 사용한 학습 블룸 필터에서는 거짓 음성이 존재하지 않게 된다. 즉, 학습 블룸 필터는 블룸 필터와 동일한 기능을 수행할 수 있게 된다. 학습 블룸 필터와 블룸 필터가 동일한 기능을 수행함에도 불구하고, 학습 블룸 필터를 사용해야 하는 이유는

거짓 양성 성능을 개선시킬 수 있기 때문이다. 단, 학습 블룸 필터는 블룸 필터에 비해서 시간이 오래 소요되며 학습을 위한 데이터가 충분히 있어야 한다는 전제조건이 있다.

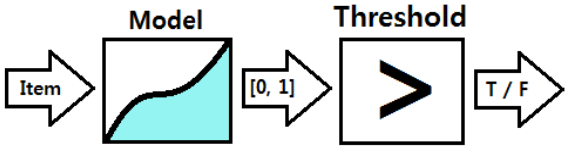


[그림 11] 학습 블룸 필터의 구성 요소

학습 블룸 필터를 구성하는 요소는 모델, 임계치, 데이터 그리고 보조 필터로 총 4가지 요소로 구성되어 있다. 모델은 기계 학습 알고리즘을 통해서 생성된 인스턴스를 이야기한다. 사용할 수 있는 모델의 종류는 다양하지만, 학습 블룸 필터에서는 이진 분류 문제를 해결해야 한다. 따라서, 이진 분류를 해결할 수 있는 모든 모델을 학습 블룸 필터에 적용할 수 있다. 대부분의 경우에는 결정 트리(Decision tree) 또는 인공신경망을 통해서 이진 분류를 수행하였다. 학습 블룸 필터를 제안한 The Case for Learned Index Structures[2]에서는 네트워크 관련 어플리케이션을 통해서 실험을 진행하였다. 해당 네트워크 관련 어플리케이션은 간단하게 블랙리스트에 URL이 존재하는지 여부를 학습 블룸 필터를 이용하여 확인한다. 블랙리스트에 있는 차단할 URL 목록을 인공신경망을 통해서 학습해서 모델로 사용하였다. 해당 논문에서의 인공신경망 구조는 총 3개의 계층을 가지며, 32차원의 문자 임베딩, 16차원의 GRU(Gated Recurrent Unit)[6] 그리고 1차원의 완전 연결 계층을 사용하였다. 또한, 마지막 계층의 경우에 활성화 함수를 시그모이드(Sigmoid) 함수로 사용하기 때문에 인공신경망의 출력값은 [0, 1] 사이의 값이 나오게 된다. 이는 URL이 블랙리스트에 존재할 확률을 표현한다고 볼 수 있다.

임계치는 모델의 출력값과 비교하여, 원소가 집합에 존재하는지 여부를 확인할 때 사용되는 값이다. 원소에 대한 모델의 출력값이 임계치보다 크면, 해당 원소는 집합에 존재한다고 판단한다. 반대로, 원소에 대해서 모델의 출력값이 임계치보다 같거나 작으면, 해당 원소는 집합에 존재하지 않는다고 판단한다. 추가적으로, 임계치의 값에 의해서 모델에서 발생하는 거짓 양성 확률이 변하게 되며, 임계치는 학습 데이터의 음성 데이터를 사용하여 결정한다. 예를 들어 사용자가 모델에서 원하는 거짓 양성 확률이 0.05였다면, 개발 환경의 음성

데이터를 입력으로 넣어서 나온 모델의 출력값을 저장하고 정렬한 뒤, 전체 음성 데이터의 개수 중 상위 5%에 해당하는 출력값을 임계치로 사용한다. 개발환경 데이터를 통해서 임계치를 결정했음에도 불구하고, 테스트 데이터에 대해서도 유사한 성능을 가진다는 것을 A model for learned bloom filters and related structures[7]에서 수학적으로 증명한다.



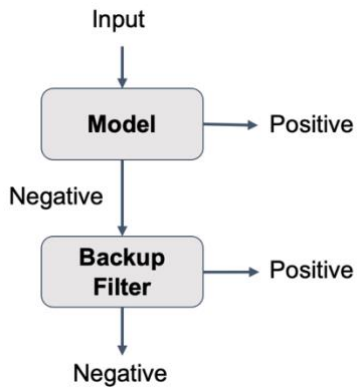
[그림 12] 학습 블룸 필터의 임계치

데이터는 키와 키가 아닌 데이터가 함께 존재해야 한다. 기존의 블룸 필터 환경에서는 키에 대해서만 입력을 수행하지만, 모델을 학습하기 위해서는 키와 키가 아닌 데이터가 필요하다. 따라서, 키만 존재하는 경우에는 키가 아닌 데이터를 추가적으로 찾거나 생성할 필요가 있다. 예를 들어, URL 데이터에 대해서 블랙리스트에 대한 데이터가 있다고 가정해보자. 모델 학습을 위해서 키가 아닌 화이트리스트에 대한 데이터를 추가적으로 구하거나, 블랙리스트에 임의의 랜덤한 노이즈를 사용하여 블랙리스트가 아닌 새로운 데이터를 인위적으로 생성해서 모델을 학습해야 한다. 즉, 학습 블룸 필터를 적용하기 위해서는 키가 아닌 데이터를 추가적으로 필요로 한다. 본 논문에서는 키 데이터를 양성 데이터로 부르고, 키가 아닌 데이터를 음성 데이터로 부른다.

보조 필터는 모델에서 생성된 거짓 음성을 제거하기 위해서 사용되는 요소이다. 보조 필터는 블룸 필터와 이름만 다를 뿐, 기존의 블룸 필터와 동일한 자료구조이다. 단, 학습 블룸 필터에서 주의할 점은 공간 측정하는 경우, 기존에는 블룸 필터의 크기만 측정하면 되지만 학습 블룸 필터에 대해서는 공간을 측정할 때 모델의 크기와 보조 필터의 크기를 함께 합해야 구할 수 있다는 점이다.

다음으로 학습 블룸 필터의 초기화, 입력 그리고 질의에 대해서 살펴보도록 한다. 학습 블룸 필터의 초기화를 위해서는 모델과 보조 필터의 거짓 양성 확률 설정, 모델 학습, 그리고 임계치 설정하는 과정을 거쳐야 한다. 사용자로부터 주어진 거짓 양성 확률이 0.01인 경우, 절반은 모델에서 처리하고 나머지 절반은 보조 필터에서 처리한다. 즉, 모델에서 0.005의 거짓 양성 비율이 발생하고 보조 필터에서

0.005의 거짓 양성 비율을 발생하도록 설정한다. 이후에는 모델을 학습하는 과정을 거치는데, 이 때 기계학습 알고리즘에 학습 데이터를 입력하여 모델을 학습시킨다. 본 논문에서는 학습, 개발, 테스트용 데이터의 비율을 8:1:1 비율로 사용하였다. 학습 데이터는 모델을 학습하는 경우에 사용되며, 개발 데이터는 임계치를 설정하는데 사용되고, 테스트 데이터는 전체적인 학습 블룸 필터의 성능을 측정할 때 사용된다. 마지막으로, 임계치 설정은 개발 환경의 음성 데이터를 학습된 모델에 대입해서 나온 출력값을 사용하여 설정한다. 개발 데이터에 대한 모델의 출력값을 정렬해서, 전체 출력값 중에서 상위 0.5%에 해당하는 출력값을 임계치로 사용하면 된다. 위에서 설정한 임계치는 모델이 정상적으로 학습되었다는 가정하에 0.005의 거짓 양성 비율을 가지게 된다.



[그림 13] 학습 블룸 필터의 구조

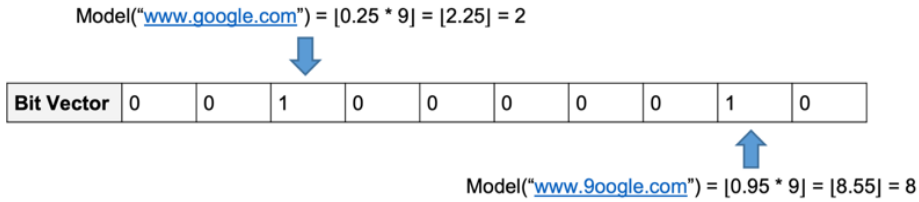
학습 블룸 필터의 입력은 양성 데이터에 대해서만 수행한다. 즉, 블룸 필터를 사용했을 경우 입력되었을 데이터에 대해서만 입력을 수행한다. 위와 같이 강조하는 이유는 학습 블룸 필터는 추가적으로 음성 데이터를 사용하여 모델을 학습하고 임계치를 설정하지만, 해당 데이터는 학습 블룸 필터에 입력하지 않음을 알리기 위함이다. 입력은 2가지의 경우가 존재하는데, 원소가 모델에 의해서 분류된 경우와 모델이 분류하지 못해서 보조 필터에 입력된 경우로 구분된다. 예를 들어, 임계치가 0.9이고 모델에 “9oogle.com”이라는 URL을 입력되었다고 가정해보자. 만약, “9oogle.com”에 대한 모델의 출력값이 0.95였다면, 0.95는 임계치인 0.9보다 크기 때문에 모델에 의해서 분류가 된다. 따라서, 해당 URL에 대해서는 보조 필터에 추가적으로 저장할 필요가 없고 모델을 통해서 해당 값이 저장되었다고 볼 수 있다. 반대로, “google.com”이라는 URL을 입력하였고,

출력값이 0.25였다고 가정해보자. 0.25는 임계치인 0.9보다 작기 때문에 모델에서 분류할 수 없는 값이다. 따라서, 이는 거짓 음성으로 분류된 상태이고 보조 필터에서 해당 URL을 저장하도록 한다. URL을 보조 필터에 저장하는 것은 블룸 필터와 동일한 방식을 사용한다. 따라서, 블룸 필터에 존재하는 k개의 해시 함수에 URL 데이터를 입력하여 저장된다고 볼 수 있다.

[그림 14] 학습 블룸 필터의 입력

학습 블룸 필터의 질의에는 양성 또는 음성 데이터가 모두 사용될 수 있다. 학습 블룸 필터의 질의는 3가지 경우의 수가 존재한다. 첫번째로, 모델에서 의해서 원소가 양성으로 분류되는 경우이다. 즉, 해당 원소가 모델을 통해서 집합에 존재한다고 반환된 경우이다. 원소에 대한 모델의 출력값이 임계치보다 크면 집합에 존재한다고 볼 수 있지만, 해당 원소가 반드시 존재한다고 확신할 수 없다. 이는 학습 블룸 필터가 음성 데이터임에도 불구하고 집합에 존재한다는 거짓 양성을 발생시킬 수도 있기 때문이다. 둘째로, 보조 필터에 의해서 양성을 반환하는 경우이다. 즉, 보조 필터에서 원소가 집합에 존재한다고 반환하는 경우이다. 이 경우는 질의하고자 하는 원소의 모델 출력값이 임계치보다 작은 경우에 발생한다. 임계치보다 작은 원소들은 모두 모델에 의해서 음성으로 분류된 후에 보조 필터에 의해서 양성 또는 음성을 반환하게 된다. 보조 필터를 확인한 후에, 보조 필터에 질의한 원소가 존재한다고 판단된 경우에는 양성을 반환한다. 단, 보조 필터는 블룸 필터와 동일한 자료구조이기 때문에 거짓 양성 확률에 따른 오류가 발생할 수 있다는 점을 유의해야한다. 마지막으로, 보조 필터에서 음성으로 반환된 경우이다. 두번째 경우와 동일하게 모델에서 질의한 원소를 음성으로 분류한 경우에 발생한다. 모델의 분류 작업이 음성으로 반환된 경우 보조 필터에서 추가적인 판별 작업을 수행한다. 보조 필터에서 원소의 존재 유무를 확인했으나 음성이 나오는 경우가 세번째 경우이다. 이 때는 보조 필터에서 반환된 음성이기 때문에 해당 원소는 반드시 집합에

존재하지 않는다고 확신할 수 있다. 즉, 학습 블룸 필터에 의해서 음성으로 분류된 원소들은 학습 블룸 필터에 입력된 적이 없다는 것을 확신할 수 있다.



[그림 15] 학습 해시 함수의 입력

추가적으로 The Case for Learned Index Structures[2]에서는 학습 블룸 필터 외에도 학습 해시 함수라는 개념에 대해서도 제안한다. 학습 해시 함수는 학습 블룸 필터에 존재하는 모델을 재사용하여 해시 함수처럼 사용하는 방식이다. 학습 해시 함수는 모델에서 나온 [0, 1] 사이의 출력값을 비트 벡터의 크기인 [0, m - 1]으로 매핑 시켜주는 작업이다. 따라서, 양성 데이터는 1 근처의 모델 출력값을 가지므로 비트 벡터의 m - 1 근처에 분포한다. 반대로, 음성 데이터는 0 근처의 모델 출력값을 가지므로 비트 벡터의 0 근처에 분포한다. [0, 1]의 출력값을 [0, m - 1]의 비트 벡터의 인덱스로 변환 시키는 공식은 $\lfloor Model(x) * (m-1) \rfloor$ 이다. 블룸 필터에서 일반적으로 사용하는 해시 함수는 균일 분포를 가지고 일정한 성능을 가지도록 설계되어 있다. 반면에, 학습 해시 함수는 균일한 분포가 아닌 학습에 의해서 생성된 분포를 가지고 해시 함수의 성능이 결정된다. 즉, 해시 함수는 균일 분포로 원소간의 충돌을 줄이는데 초점을 뒀다면, 학습 해시 함수는 양성 데이터는 양성 데이터간의 그리고 음성 데이터는 음성 데이터간의 충돌이 최대한 많아지도록 만든 방식이다.

다음은 학습 해시 함수에 대한 입력에 대해서 살펴보도록 한다. 예를 들어, 학습 해시 함수를 사용하기 위해서 크기 10의 비트 벡터가 할당되어 있다고 가정한다. 만약에 모델에 "google.com"이라는 값을 입력했을 때 0.25라는 출력값이 나왔다면, 모델의 출력값을 비트 벡터의 인덱스로 변환하는 공식은 $\lfloor 0.25 * (10-1) \rfloor$ 이다. 즉, 2번째 인덱스에 입력값의 존재 여부를 표현할 것을 알려준다. 따라서, 크기가 10인 비트 벡터의 인덱스가 2인 값을 0에서 1으로 갱신하면 입력이 완료된다. 학습 해시 함수의 질의 또한, 위의 동일한 공식을 사용하여 진행된다. 따라서, 해시 함수는 k개의 해시 함수를 사용하지만, 학습 해시 함수는 1개의 해시 함수를 사용한다. 이는 학습 해시 함수를 사용하기 위한

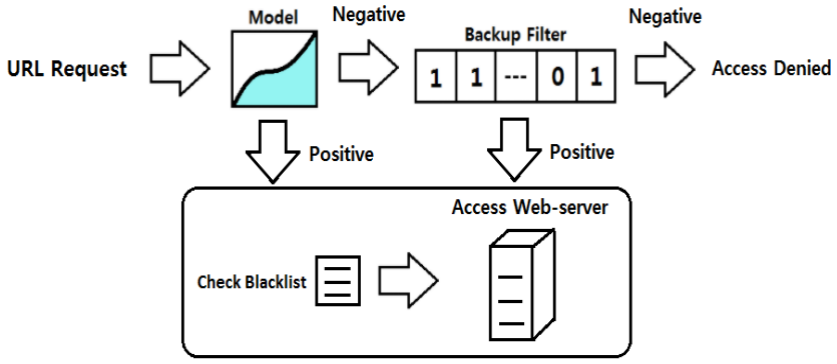
모델이 1개만 존재하기 때문이다.

학습 블룸 필터는 집합의 크기가 커질 수록 블룸 필터에 비해서 개선된 거짓 양성 비율을 가진다. 단, 학습 블룸 필터는 개선된 거짓 양성 비율을 댓가로 블룸 필터에 비해서 시간이 오래 소요되는 큰 단점이 있다. 개선된 거짓 양성 비율과 오래 소요되는 시간은 모두 모델을 적용하므로 나타난 현상이다. Adaptive Learned Bloom Filters under Incremental Workloads[8]에서는 데이터베이스와 유사한 환경인 동적으로 집합의 크기가 커지는 환경에서 학습 블룸 필터를 실험하였다. 결과적으로 모델이 학습하는데 오랜 시간을 소요함에 따라서, 학습 블룸 필터를 동적인 집합에는 적용하기 어렵다는 점을 명시하였다. 위와 같이 학습 블룸 필터는 시간에 대한 약점과 동시에 동적인 집합(Dynamic Set)에 대해서 적용이 어렵다는 단점이 있다.

그럼에도 불구하고 학습 블룸 필터에 대한 확장을 연구 주제로 선택한 이유로는 크기가 결정되어 있는 정적인 집합(Static Set)에 대해서는 충분히 사용할 수 있다는 점과 하드웨어와 관련된 연구들이 진행되고 있다는 이유 때문이다. 2018년 포브스 기사[1]에 의하면 2025년까지 전세계적으로 175제타바이트(Zetabyte)의 규모의 데이터가 있을 것으로 예상하고 있다. 데이터가 커지면 커질수록 제한된 메모리 안에 블룸 필터를 할당하지 못할 가능성이 존재한다. 위와 같이 메모리가 부족한 상황에서는 블룸 필터와 비교하여 더 적은 공간을 사용하는 학습 블룸 필터를 고려해 볼 수 있다. 학습 블룸 필터는 모델의 사용으로 질의의 시간이 블룸 필터에 비해서 느리지만, 시간에 대한 제약조건이 느슨하지만 많은 공간을 필요로 하는 데이터 웨어하우스(Data warehouse) 또는 스토리지(Storage)에서 사용될 수 있을 것이다. 그 외에도 학습 블룸 필터를 어플리케이션에 적용하고자 하는 연구들이 네트워크와 스트리밍 데이터 분야에서 진행되고 있다[9][10]. 추가적으로, 기계학습을 데이터베이스에 적용하는 학습 인덱스 분야의 연구가 활발히 진행되고 있다[11][12]. 학습 블룸 필터는 모델의 사용으로 현재의 하드웨어에는 시간적인 면에서는 실용적이지 않을 수는 있지만, 하드웨어 또는 모델의 개선으로 인해서 미래에 응용 될 수 있는 가능성이 존재한다.

[1] T. Coughlin, '175 Zettabytes By 2025', *175 Zettabytes By 2025*, Forbes, 2018, https://www.forbes.com/sites/tomcoughlin/2018/11/27/175-zettabytes-by-2025 (2021.6.25 접속).

2.4 학습 블룸 필터의 어플리케이션



[그림 16] 학습 블룸 필터의 콘텐츠 제어 어플리케이션

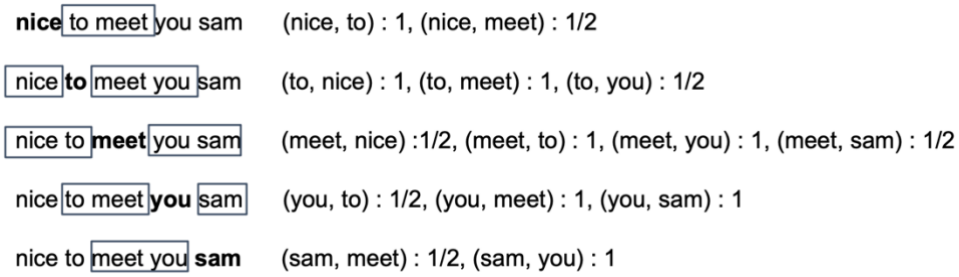
학습 블룸 필터는 바이러스 스캔 또는 악성 URL 검출 등 다양한 분야에서 사용되고 있다. 학습 블룸 필터는 정적인 집합에 대해서만 적용이 가능하기 때문에 블룸 필터에 비해서 제한적인 분야에서만 사용할 수 있다. 본 논문에서는 콘텐츠 제어(Content control)에 대한 학습 블룸 필터의 사용에 대해서 살펴보도록 한다.

콘텐츠 제어 어플리케이션은 유해한 컨텐츠를 차단하기 위해서 사용한다. 예를 들어, 어린이가 19세 관람가의 영화를 접근하지 못하는 것도 일종의 컨텐츠 제어이다. 차단을 하고자 하는 컨텐츠의 URL을 양성 데이터로 설정하고, 허용을 하고자 하는 컨텐츠의 URL은 음성 데이터로 설정한다. 양성과 음성 데이터 전체를 사용하여 학습 블룸 필터의 모델을 학습시킨다. 초기화된 학습 블룸 필터에 차단하고자하는 URL인 양성 데이터를 입력하면 컨텐츠 제어 어플리케이션에서 학습 블룸 필터를 사용할 수 있게 된다. 예를 들어, 사용자가 차단된 URL을 접근을 하려고 하면 해당 URL은 학습 블룸 필터에게 전달된다. 학습 블룸 필터 내부에 있는 모델 또는 보조 필터를 통해서 차단할 URL인지 여부를 구분할 수 있으며, 차단할 URL이였다고 판단되는 경우 실제로 블랙리스트에 존재하는지 확인한다. 블랙리스트에 해당 URL이 존재할 경우 서버로의 접근을 허용하지 않는다. 단, 학습 블룸 필터에서 차단할 URL이라고 구분했지만 블랙리스트에 존재하지 않는 경우가 존재할 수 있으며, 이는 거짓 양성으로 학습 블룸 필터에서 발생한 일종의 오류이다. 따라서, 해당 URL에 대해서는 컨텐츠를 허용해주면 된다. 반대로 학습 블룸 필터에서 차단되지 않은 URL로 구분된 경우에는, 블랙리스트에 URL이 존재하지 않음을 확신할 수 있다. 확신이 가능한

이유는 학습 블룸 필터에는 보조 필터의 사용으로 인해서 거짓 음성이 존재하지 않기 때문이다. 따라서, 이 경우에는 추가적인 블랙리스트 확인 작업 없이 URL에 대한 컨텐츠를 허용해주면 된다.

블룸 필터도 동일하게 컨텐츠 제어를 위해서 사용될 수 있다. 허나, 149만개의 양성 데이터가 존재한다고 가정하였을 때, 0.01의 거짓 양성 확률을 가지는 블룸 필터는 약 1.7메가바이트(Megabyte)인데 반해 학습 블룸 필터는 약 1.2메가바이트로 표현될 수 있다. 즉, 동일한 성능의 거짓 양성 비율을 더 적은 공간을 사용하여 나타낼 수 있게 된다.

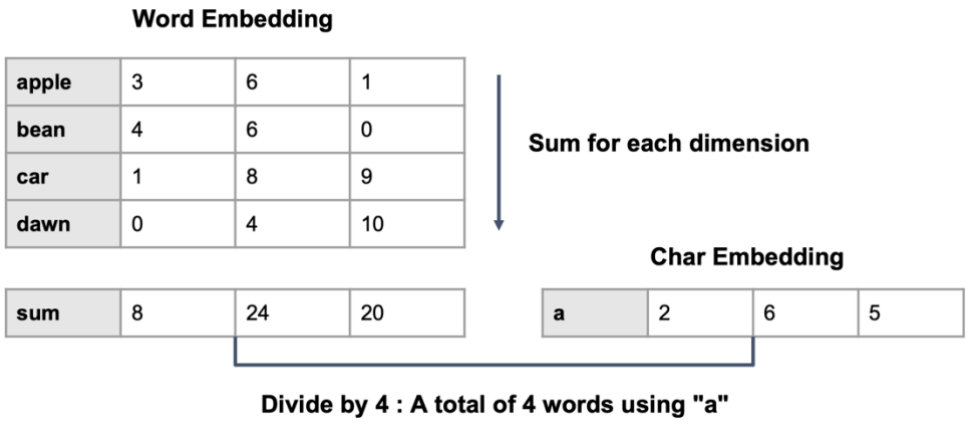
URL을 구분하기 위해서 사용되는 모델은 인공신경망이다. 인공신경망은 The Case for Learned Index Structures[2]에서 제안된 32차원 문자 임베딩, 16차원 GRU 그리고 1차원의 완전 연결 계층으로 구성되어 있다. 32차원 문자 임베딩은 각 URL의 문자를 32차원의 벡터로 변경하기 위해서 사용된다. 즉, 2개의 동일한 문자를 32차원의 벡터로 바꾼다면 해당 벡터들은 동일한 값을 가진다. 32차원의 문자 임베딩 값을 구하기 위해서는 GloVe(Global Vectors for Word Representation)[13]를 사용해야 한다.



[그림 17] GloVe의 동시 발생 행렬

GloVe는 단어를 표현하기 위한 비지도 학습(Unsupervised learning) 방법이다. 비지도 학습은 레이블이 주어지지 않은 상태에서 통계적인 기법을 이용해서 데이터 내부의 특징을 찾는 학습 방법이다. GloVe를 적용하기 위해서는 동시 발생 행렬(Co-occurrence Matrix)을 생성하는 작업이 필요하다. 동시 발생 행렬은 2차원이며 각 차원은 단어 집합(Vocabulary)의 크기에 의해서 결정된다. 동시 발생 행렬의 값을 채우기 위해서는 윈도우(Window)의 크기를 설정해야 하는데, 이는 기준이 되는 단어 주변으로 몇개의 단어를 볼 것인지 결정하는 변수이다. 예를 들어, “nice to meet you sam” 라는 문장이 주어지고 윈도우의 크기가 2라고 가정해보자. “meet” 라는 단어를 기준으로 좌측의 “nice” 와 “to” 단어가 발생하였고, 우측의 “you” 와 “sam” 단어가

발생하였다. 이를 통해서, “meet”라는 단어가 발생한다면 “nice”, “to”, “you”, “sam”과 같은 단어가 발생할 수 있음을 통계적으로 저장하는 것이다. 단, 기준이 되는 단어인 “meet”와 가까이에 존재하는 단어에는 높은 비중의 무게(weight)를 주고, 멀리에 존재하는 단어일수록 비중을 낮춘다. 따라서 위의 예시에서는 기준이 되는 단어와 띄어쓰기 1칸으로 구분된 단어인 “to”와 “you”에는 1의 무게를 주고, 2칸으로 구분된 단어인 “nice”와 “sam”에는 1/2의 무게를 준다. 동시 발생 행렬의 값을 모두 계산한 뒤에는 GloVe에서 제시한 손실 함수를 사용해서 학습을 수행한다. 학습이 끝나면 2차원의 행렬이 생성되며, 첫번째 차원은 단어 집합의 크기이고 두번째 차원은 사용자가 하이퍼파라미터로 설정한 원하는 차원의 수이다. 즉, GloVe는 단어 집합에 존재하는 각 단어 별로 사용자가 지정한 차원의 수를 만들어주는 기계학습 방법이다.



[그림 18] 문자 임베딩

문자 임베딩은 GloVe가 학습한 2차원 행렬을 사용해서 구할 수 있다. GloVe에서 학습한 2차원 행렬은 단어 임베딩이다. 단어 임베딩을 사용하여 더 작은 단위인 문자 임베딩을 구할 수 있다. 예를 들어, “word”라는 단어가 있다면 해당 단어에 사용되는 문자는 “w”, “o”, “r”, “d”이다. 문자 임베딩은 해당 문자가 들어간 모든 단어의 벡터 값을 합해서 평균 내서 사용한다. 즉, “w”에 대한 문자 임베딩을 만들기 위해서는 단어 집합에서 “w”가 포함되는 모든 단어를 찾은 후 해당 벡터 값들에 대해서 평균을 내면 된다. 문자 임베딩을 사용하면, 각 문자는 GloVe에서 설정한 차원의 벡터 매핑으로 매핑된다.

인공신경망의 두번째 계층에 사용된 GRU는 RNN(Recurrent Neural Network) 프레임워크의 일종으로 자연어 처리를 수행하는데

사용된다. GRU는 은닉 계층(Hidden Layer)에 리셋 게이트(Reset Gate)와 업데이트 게이트(Update Gate)를 추가한 구조로 구성되어 있다. 리셋 게이트는 이전에 사용되었던 정보를 제거하는 용도로 사용된다. 즉, 단기(Short-term) 정보를 유지하는데 기여한다. 업데이트 게이트는 이전에 사용되었던 정보를 얼마나 사용할지 결정하는 용도로 사용된다. 즉, 장기(Long-term) 정보를 유지하는데 기여한다. 결과적으로 단기와 장기에 대한 문장 정보를 모두 포함하여 인공신경망이 자연어를 이해할 수 있도록 돕는 역할을 한다. 본 논문에서 GRU는 URL의 정보를 인식하는데 사용된다.

제 3 장 제안 모델

이 장에서는 학습 블룸 필터와 관련된 연구 그리고 본 논문에서 제안하는 확장 학습 블룸 필터에 대해서 살펴보도록 한다. 제1절에서는 학습 블룸 필터와 학습 블룸 필터의 거짓 양성 비율을 개선시키기 위한 연구들에 대해서 설명한다. 제2절에서는 본 논문에서 제안하는 확장 학습 블룸 필터에 대해서 정의한다.

3.1 학습 블룸 필터 관련 연구

The Case for Learned Index Structures[2]는 학습 블룸 필터를 최초로 정의한 논문이다. 해당 논문에서는 블룸 필터에 이진 분류하는 모델을 전처리 과정으로 사용하여 학습 블룸 필터를 만들 수 있음을 보여준다. 이는 블룸 필터를 데이터 과학 관점에서 해석하면 이진 분류로 볼 수 있음을 인지하고 블룸 필터의 구조를 변경한 것이다. 하지만, 블룸 필터는 거짓 음성이 나오지 않는데 반해서 이진 분류 모델은 거짓 음성이 존재한다. 따라서, 보조 필터라는 블룸 필터와 동일한 구조를 사용하여 거짓 음성을 방지한다. 해당 논문은 모델을 전처리 과정 외에도 학습 해시 함수를 통해서 모델을 사용할 수 있음을 보여준다. 학습 해시 함수는 균일 분포를 따르는 일반적인 해시 함수와 다르게, 모델에서 반환한 [0, 1]의 출력값을 비트 벡터의 인덱스와 매핑하는 것을 말한다. 해당 논문은 학습 블룸 필터 외에도 B-Tree를 모델로 변환해서 질의할 수 있음을 말하며, 학습 인덱스 분야를 새롭게 개척한 중요한 논문이다.

Adaptive Learned Bloom Filter(Ada-BF)[14]는 2개의 학습 블룸 필터의 변형된 구조를 제안한다. 처음으로 제안된 구조인 Ada-BF는 모델의 출력값 [0, 1]을 여러 개의 구간으로 나눠서 각 구간별로 해시 함수의 개수를 다르게 적용할 것을 제안한다. 예를 들어, 양성 데이터가 존재할 가능성이 높은 [0.9, 1.0] 구간에서는 해시 함수를 하나만 사용하여, 거짓 양성 비율을 개선할 수 있다. 반대로, 양성 데이터가 존재할 가능성이 낮은 [0.0, 0.1] 구간에서는 해시 함수를 여러 개 사용하여 거짓 양성 비율을 개선할 수 있다. 정리하자면, 거짓 양성

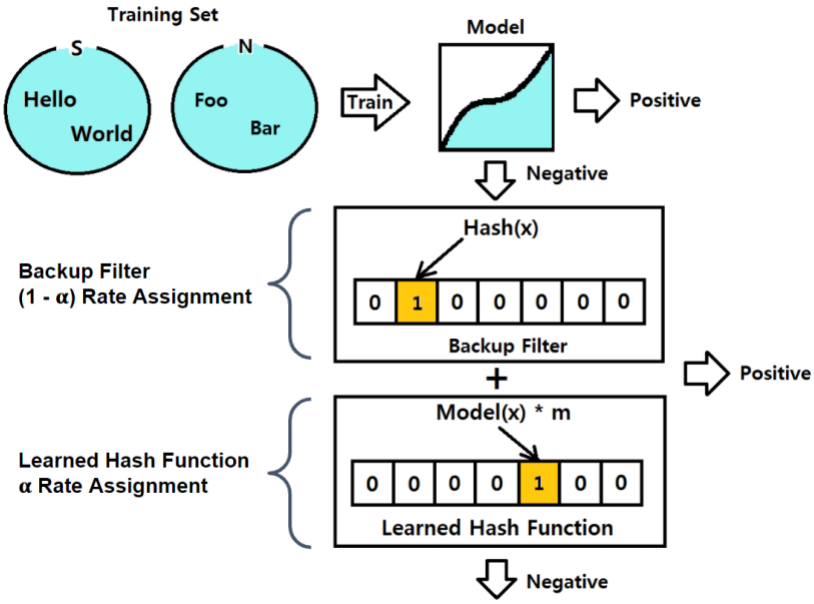
비율이 나올 확률이 낮은 곳에서는 해시 함수를 여러 개 사용하는 것은 낭비이므로 해시 함수의 개수를 줄이는 것이다. 반대로 거짓 양성 비율이 나올 확률이 높은 곳에서는, 여러 개의 해시 함수를 사용하여 거짓 양성 비율을 낮추는 것이다. 따라서, 이는 구간에 따라서 사용되는 해시 함수의 개수를 늘리거나 줄이도록 설계된 학습 블룸 필터이다. 다음으로 제안된 구조인 Disjoint Ada-BF는 모델의 출력값 [0, 1]을 여러 개의 구간을 나눠서, 기존에 하나였던 보조 필터를 여러 개의 보조 필터로 바꿀 것을 제안한다. 예를 들어, 양성 데이터가 존재할 가능성이 높은 [0.9, 1.0] 구간에서는 보조 필터의 크기를 작게 할당하여 공간을 효율적으로 사용할 수 있다. 반대로, 양성 데이터가 존재할 가능성이 낮은 [0.0, 0.1] 구간에서는 보조 필터의 크기를 크게 할당하여 거짓 양성 비율을 줄이려는 것이다. 즉, 하나의 보조 필터를 사용하는 것보다 여러 개의 보조 필터를 사용하여 거짓 양성 비율을 개선시킨 학습 블룸 필터이다. 결과적으로, 사용자가 입력한 범위 내에서의 해시 함수 개수 또는 블룸 필터의 크기를 탐색하며 가장 좋은 거짓 양성 비율을 가지는 Ada-BF 또는 Disjoint Ada-BF를 찾아서 사용하게 된다.

Partitioned Learned Bloom Filter[15]는 Disjoint Ada-BF의 후속 연구이다. Disjoint Ada-BF는 사용자가 설정한 범위 내에서 보조 필터의 크기를 선택하는데, 이는 사용자가 설정한 범위에 따라서 성능 결정되는 단점이 존재한다. 따라서, 위와 같은 휴리스틱(Heuristic)을 사용하지 않고 수학적으로 가장 좋은 보조 필터의 크기를 찾는 공식을 제안한다. 해당 공식을 통해서 양성 데이터와 음성 데이터간의 쿨백-라이블러 발산(Kullback-Leibler Divergence)과 연관되어 보조 필터의 크기가 결정됨을 나타낸다. 여기서 쿨백-라이블러 발산은 주어진 2개의 확률 분포의 차이를 계산하는데 사용된다. 쿨백-라이블러 발산이 양성 데이터와 음성 데이터 분포간에 차이가 적을수록, 보조 필터에 더 많은 공간을 사용해야 한다. 반대로, 쿨백-라이블러 발산이 양성 데이터와 음성 데이터 분포간에 차이가 클수록, 보조 필터에 사용하는 공간이 줄어든다는 것을 보인다. 그 외에도 동적 프로그래밍(Dynamic Programming)을 통해서 모델의 출력값 [0, 1]의 구간을 나눌 때, 최적에 가깝게 나누는 방법을 소개한다. 각 구간의 크기는 구간에 속하는 양성 데이터의 개수와 음성 데이터의 개수에 의해서 결정된다. 결과적으로, 위의 2가지 방법을 사용하여 만들어진 Partitioned Learned Bloom Filter는 Ada-BF보다 개선된 거짓 양성 비율을 가진다.

학습 블룸 필터와 관련된 연구들은 모두 거짓 양성 비율을

개선시키기 위한 방법이라는 공통점을 가진다. 본 논문에서도 확장 학습 블룸 필터를 사용하여 개선된 거짓 양성 비율을 가지는 학습 블룸 필터를 소개하고자 한다.

3.2 확장 학습 블룸 필터

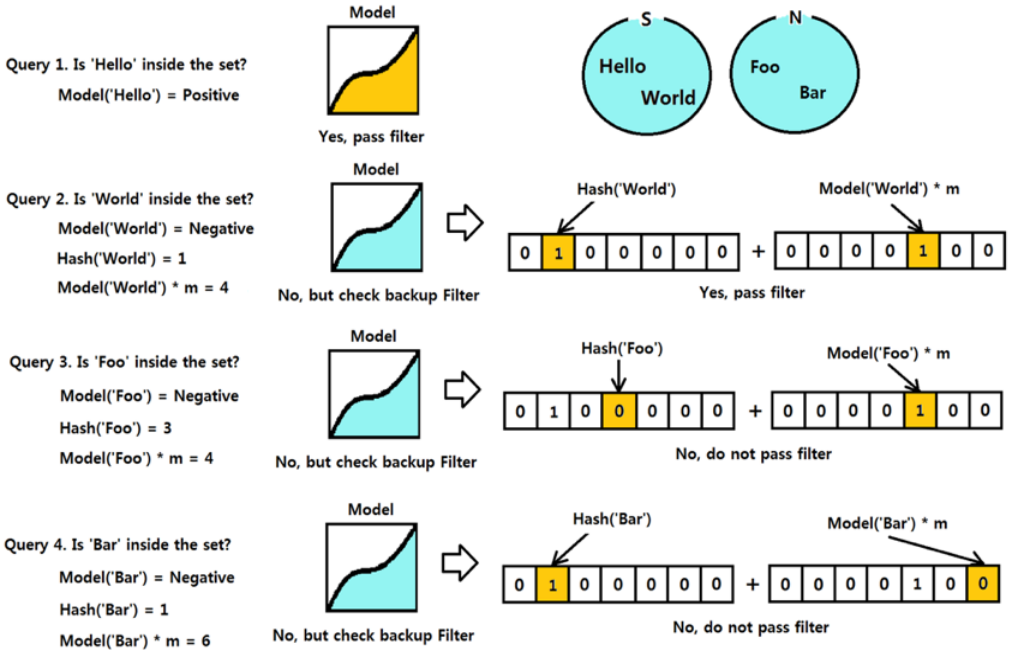


[그림 19] 확장 학습 블룸 필터의 구조

확장 학습 블룸 필터는 학습 블룸 필터의 변형이다. 학습 블룸 필터의 내부에 사용되는 보조 필터를 그대로 사용하지 않고, 보조 필터에 추가적으로 학습 해시 함수를 적용하는 자료구조이다. 위와 같은 구조는 The Case for Learned Index Structures[2]에서 이미 정의되어 있으며, 본 논문에서는 하이퍼파라미터 α를 추가적으로 정의한다. α는 [0, 1]의 범위를 가지는 변수로, 전체 보조 필터의 공간 중에서 학습 해시 함수를 사용할 비율을 나타낸다. 반대로, 1 - α의 공간은 기존의 보조 필터를 사용할 비율을 나타낸다. 학습 해시 함수와 보조 필터를 함께 사용하면 기존의 학습 블룸 필터에 비해서 개선된 거짓 양성 비율을 가지는 구간이 존재하여, 확장 학습 블룸 필터를 소개한다.

확장 학습 블룸 필터는 모델, 임계치, 데이터, 보조 필터, 학습 해시 함수로 총 5가지 요소로 구성된다. 이는 학습 해시 함수의 4가지 요소에 학습 해시 함수를 추가한 것이며, 학습 해시 함수를 사용하기 위한 공간을 보조 필터에서부터 가져온다.

학습 해시 함수와 보조 필터는 동작하는 방식이 다르기 때문에 반드시 공간을 나눠서 사용해야 한다. 보조 필터의 내부에서는 해시 함수가 사용되는데, 이는 입력하는 원소를 비트 벡터에 균일하게 매핑하기 위해서 사용된다. 반대로 학습 해시 함수는 비트 벡터를 균일하지 않게 사용하며, 양성 데이터는 비트 벡터의 마지막 인덱스 근처에 분포하게 된다. 따라서, 해시 함수와 학습 해시 함수를 함께 사용하면 2개의 동작 방식이 혼용되어 성능이 향상되지 않는다.



[그림 20] 확장 학습 블룸 필터의 개요

다음은 확장 학습 블룸 필터의 초기화, 입력 그리고 질의에 대해서 살펴보도록 한다. 확장 학습 블룸 필터의 초기화는 기존 학습 블룸 필터와 다르지 않다. 음성과 양성 데이터를 통해서 모델을 학습하고, 사용자가 설정한 거짓 양성 확률을 모델과 보조 필터에 절반씩 할당한다. 보조 필터에 할당된 거짓 양성 확률을 통해서 보조 필터의 크기(m)를 결정한다. 즉, 블룸 필터의 크기를 결정하는 공식을 사용해서 보조 필터의 크기를 구한다. 이후에는 사용자가 설정한 α 변수에 따라서, $\lfloor \alpha * m \rfloor$ 의 공간은 학습 해시 함수를 수행하는 비트 벡터(B_L)를 생성하고, 나머지 $\lceil (1-\alpha) * m \rceil$ 의 공간은 보조 필터(B_B)를 할당한다.

확장 학습 블룸 필터의 입력은 양성 데이터에 대해서만 수행하고, 2가지의 경우가 존재한다. 입력하고자 하는 원소의 모델 출력값이 임계치보다 큰 경우, 해당 원소는 모델에 의해서 입력되었다고 볼 수

있다. 반대로, 입력하고자 하는 원소의 모델 출력값이 임계치보다 같거나 작은 경우에는 보조 필터에 원소를 입력하는 작업을 수행해야 한다. 확장 학습 블룸 필터의 보조 필터는 학습 해시 함수를 사용하는 공간과 기존의 해시 함수를 사용하는 보조 필터로 나눠진다. 따라서, 입력을 위해서는 기존의 보조 필터를 통해서 입력을 한번 수행하고, 이후에 학습 해시 함수를 이용해서 입력을 한번 더 수행해야 한다. 즉, 보조 필터의 입력은 총 2번 수행된다.

Algorithm 1 Insert of Extended LBF

```
1.  Function insert(x):
2.    if f(x) > threshold:
3.      pass
4.    else:
5.      for i = 1, ... , k:
6.        B_B[h_i(x)] = 1
7.      end
8.      B_L[f(x)] = 1
9.    end
10. end
```

[그림 21] 확장 학습 블룸 필터의 입력 알고리즘

확장 학습 블룸 필터의 질의는 양성 데이터와 음성 데이터에 대해서 모두 수행된다. 질의하는 과정은 입력하는 과정과 유사하며, 질의하고자 하는 원소의 모델 출력값이 임계치보다 높으면 집합에 존재한다고 판단한다. 반대로, 질의하고자 하는 원소의 모델 출력값이 임계치보다 같거나 작으면 보조 필터에서 추가적으로 집합의 존재 유무를 확인한다. 보조 필터에서 확인하는 경우에, 이전과 동일하게 보조 필터에서 한번 질의하고, 이후에 학습 해시 함수를 이용해서 질의를 한번 더 수행해야 한다.

확장 학습 블룸 필터의 거짓 양성 비율은 모델에서 발생하는 거짓 양성 비율(P_M)과 보조 필터에서 발생한 거짓 양성 비율을 더해서 구할 수 있다. 보조 필터에서 발생하는 거짓 양성 비율은 학습 해시 함수(P_L)와 보조 필터(P_B)를 함께 사용하므로, 이는 곱연산으로 표현할 수 있다. 즉, 확장 학습 블룸 필터의 전체 거짓 양성 비율을 수식으로 표현한다면 다음과 같다.

$$P = P_M + (P_B * P_L) \quad (1)$$

Algorithm 2 Query of Extended LBF

```
1.  Function query(x):
2.    if f(x) > threshold:
3.      return True
4.    else:
5.      for i = 1, ... , k:
6.        if B_B[h_i(x)] == 0:
7.          return False
8.        end
9.      end
10.     if B_L[f(x)] == 0:
11.       return False
12.     end
13.   end
14.   return True
15. end
```

[그림 22] 확장 학습 블룸 필터의 질의 알고리즘

모델과 보조 필터에 대한 거짓 양성 확률은 사용자가 설정해줄 수 있으나, 학습 해시 함수는 모델과 비트 벡터의 크기에도 종속적이기 때문에 경험(Empirical)으로 구할 수 밖에 없다. 학습 해시 함수의 거짓 양성 비율을 구하는 공식은 다음과 같다.

$$P_L = \sum_{i=1}^{\lfloor \alpha m \rfloor} \frac{(P \cap N \in i \rightarrow |N_i|) \wedge (\neg P \cap N \in i \rightarrow 0)}{|N_i|} \qquad (2)$$

위 수식에서 i는 학습 해시 함수의 인덱스를 나타내며, $\lfloor \alpha m \rfloor$은 학습 해시 함수가 할당된 공간이다. P는 양성 데이터에 대한 인덱스 집합이며, N은 음성 데이터에 대한 인덱스 집합이다. N_i는 중복집합(Multiset)으로 인덱스 i에 할당된 음성 데이터의 수를 가진다. 즉, 위의 수식은 양성 데이터와 음성 데이터가 공통적인 인덱스를 가지는 경우, 해당 인덱스의 음성 데이터 개수만큼 합을 수행하여 거짓 양성의 발생한 횟수를 센다. 이후에, 전체 음성 데이터의 개수로 거짓 양성이 발생한 횟수를 정규화하여 거짓 양성 비율을 계산할 수 있다.

확장 학습 블룸 필터가 기존의 학습 블룸 필터보다 성능이 좋아지기 위해서는 학습 블룸 필터의 보조 필터보다 작은 공간을 사용하고 동일한 성능을 가져야 한다. 이를 수식으로 표현하면 다음과 같다.

$$m_L + m_B + model \leq m_O + model \tag{3}$$

$$m_L + m_B \leq m_O \tag{4}$$

$$m_L + \frac{-nlnP_B}{(ln2)^2} \leq \frac{-nlnP_O}{(ln2)^2} \tag{5}$$

$$m_L + \frac{-nlnP_B}{(ln2)^2} \leq \frac{-nlnP_BP_L}{(ln2)^2} \tag{6}$$

$$m_L \leq \frac{-nlnP_L}{(ln2)^2} \tag{7}$$

위 수식에서 m_L은 확장 학습 블룸 필터의 학습 해시 함수에 할당된 공간을 의미하며, m_B는 확장 학습 블룸 필터의 보조 필터에 할당된 공간을 의미한다. m_O는 학습 블룸 필터의 보조 필터에 할당된 공간을 말한다. 마지막으로, 다음 수식은 거짓 양성 확률(p)와 집합의 크기(n)이 주어진 경우 최적의 비트 벡터 크기를 구하는 수식이다.

$$\frac{-nlnP}{(ln2)^2} \tag{8}$$

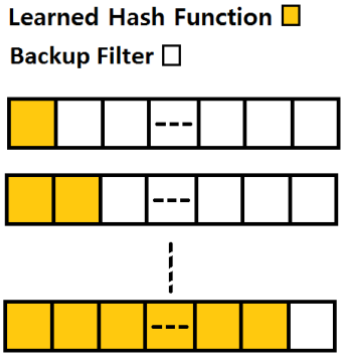
정리된 수식 (7)을 보면, 학습 해시 함수에 대한 거짓 양성 비율(P_L)이 주어진 경우, 해당 거짓 양성 비율로 생성된 블룸 필터에 비해서 공간을 적게 사용해야 한다는 것이다. 즉, 학습 해시 함수의 크기(m_L)가 거짓 양성 비율(P_L)을 갖는 블룸 필터보다 작은 공간을 사용하면서 동일한 거짓 양성 비율을 가진다면, 확장 학습 블룸 필터가 학습 블룸 필터보다 개선된 성능을 가질 것이다.

확장 학습 블룸 필터가 블룸 필터의 대안으로 사용되기 위해서는 거짓 음성이 존재하지 않는다는 특성을 만족해야 한다. 이를 증명하기 위해서 확장 학습 블룸 필터에 거짓 음성이 존재한다고 가정해보자. 거짓 음성이라는 것은 양성 데이터를 음성으로 구분했다는 것이다. 모델과 임계치를 사용하여 분류하는 전처리 과정은 오직 양성만을 반환하기 때문에 음성을 반환하기 위해서는 보조 필터를 거쳐야 한다. 보조 필터에 입력되는 원소는 모두 해시 함수 또는 학습 해시 함수를 통해서 비트 벡터에 저장되기 때문에 한번 저장된 원소에 대해서는 음성을 반환할 수 없다. 이는 기존에 확장 학습 블룸 필터에서 거짓 음성이 존재했다는 가정에 모순된다. 따라서, 확장 학습 블룸 필터에서는 거짓 음성이 존재하지 않는다.

제 4 장 구 현

이 장에서는 확장 학습 블룸 필터의 구현과 구현 과정 중에 발생했던 어려움에 대해서 알아본다. 1절에서는 확장 학습 블룸 필터의 하이퍼파라미터 탐색에 대해서 설명한다. 2절에서는 확장 학습 블룸 필터를 구현하며 생겼던 문제에 대해서 설명한다. 3절에서는 모델 조정을 통해서 변경된 모델에 대해서 소개한다. 4절에서는 학습 해시 함수에 대해서 이해하고자 한다.

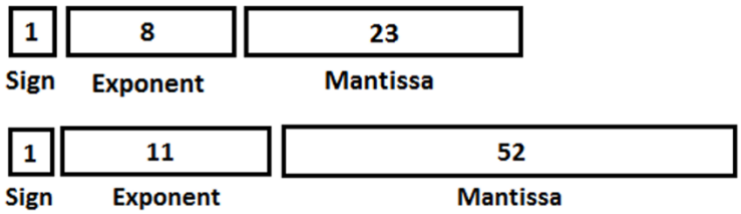
4.1 하이퍼파라미터 탐색



[그림 23] 비트 단위 탐색의 어려움

확장 학습 블룸 필터에서 하이퍼파라미터 α를 넣는 이유는 최적의 모델을 찾기 위함이다. 블룸 필터에서는 이미 비트 벡터의 크기(m)라는 변수가 존재하지만 이를 통해서 최적의 모델을 찾기는 어렵다. 예를 들어, 10,000 비트 벡터의 공간이 주어졌다고 가정해보자. 1비트씩 학습 해시 함수를 늘려가면서 확장 학습 블룸 필터를 초기화하기 위해서는 총 10,000번의 탐색이 필요하다. 하지만, 최적의 확장 학습 블룸 필터를 찾기 위해서 10,000번씩 탐색하는 행위는 많은 시간을 필요로 한다. 실제로 사용되는 비트 벡터는 10,000보다 훨씬 크며, 10단위의 숫자로 나누어 떨어지지 않는 값일 수 있다. 따라서, 본 논문에서는 하이퍼파라미터 α를 제안하고, α값을 0.01부터 0.01씩 증가하면서 모델을 탐색한다면 100번의 탐색 끝에 최적에 근접하는 확장 학습 블룸 필터를 찾을 수 있다.

4.2 모델 정밀도



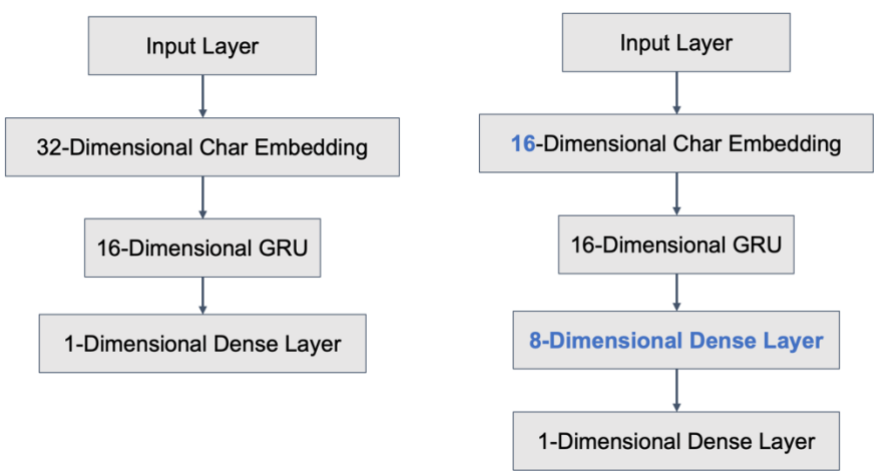
[그림 24] 32비트와 64비트 부동소수점

모델 정밀도 문제는 확장 학습 블룸 필터를 구현 중에 발견한 문제이다. 모델의 정밀도로 인해서 학습 해시 함수로 입력을 했음에도 불구하고 거짓 음성이 나오는 현상이다. 즉, 학습 해시 함수를 통해서 어떠한 원소를 입력했지만, 질의를 하면 해당 원소가 존재하지 않는다고 나타나는 현상이다. 블룸 필터가 실용적인 이유는 거짓 음성이 없기 때문인데, 모델 정밀도 문제로 인해서 거짓 음성이 나오는 것은 치명적인 오류이다. 이는 확장 학습 블룸 필터에 입력을 수행할 때는 인공신경망의 배치(Batch) 크기를 입력되는 양성 데이터의 개수로 수행하고, 질의를 수행하는 경우에는 배치 크기를 1로 설정했기 때문에 발생한 문제이다. 수학적으로는 실수와 실수 사이에는 무한대의 실수가 존재한다. 하지만, 컴퓨터에서는 실수와 실수 사이를 무한대로 표현할 수 없고 이는 컴퓨터의 한정된 컴퓨팅 자원 때문이다. 따라서, 이를 표현하고자 만든 방법이 부동소수점(Floating point)이다. 32비트 부동소수점은 1개의 부호 비트, 8개의 지수 비트 그리고 23개의 가수 비트로 구성되어 있다. 가수는 유효숫자를 저장하는 공간으로 값이 실질적으로 저장되는 공간이라 볼 수 있다. 가수는 2^{23}까지의 범위까지 밖에 표현할 수 없기 때문에 10진수로 표현된 숫자의 경우에 2진수로 저장된 부동소수점과 일치하는 숫자가 없을 수 있다. 이 때, 가장 근접하는 소수를 사용해서 10진수를 표현하게 된다. 부동소수점으로 표현된 숫자는 각 숫자마다 유효한 자릿수 다르나, 32비트의 부동소수점은 최소 6자릿수에 대해서 유효하다. 예를 들어, 배치 크기가 10,000이였을 때 “google.com”에 대한 모델의 출력값이 0.24470171이였다면, 배치 크기가 1이였을 때는 동일한 입력에 대한 모델의 출력값이 0.24470183이다. 위의 작은 차이로 인해서 학습 해시 함수에 의해서 매핑되는 비트 벡터의 인덱스 값이 변하게 된다.

위의 문제를 해결하기 위해서 인공신경망에서 사용하는

부동소수점을 32비트에서 64비트로 변경하여 해결하였다. 64비트의 부동소수점은 1개의 부호 비트, 11개의 지수 비트, 52개의 가수로 구성되어 있다. 가수는 2^{52}까지의 범위를 표현할 수 있으므로, 이전에 32비트 부동소수점을 사용했을 때 6자릿수에 대해서 유효했다면, 64비트 부동소수점을 사용하면 15자릿수까지 유효하다. 따라서, 거짓 음성이 발생되는 버그를 해결할 수 있었다. 단, 32비트에서 64비트로 모델을 변환하는 과정으로 인해서 모델의 크기는 2배 증가하게 된다. 그 외에도, 모델의 학습하는 시간과 질의하는 시간에 추가적인 비용이 소요될 수 있다. 모델의 정밀도 문제로 인해서 학습 해시 함수를 사용하는 비트 벡터의 크기를 늘려도 성능 향상이 안되는 지점이 있을 것으로 판단된다. 64비트 부동소수점을 사용했을 때, 약 100테라바이트(Terabyte) 구간이며, 아직 대부분의 컴퓨터는 메모리를 100테라바이트 단위로 사용하지 않으므로, 64비트 부동소수점의 모델을 적용하면 모델의 정밀도 문제가 발생하지 않는다고 봐도 무방하다.

4.3 모델 조정



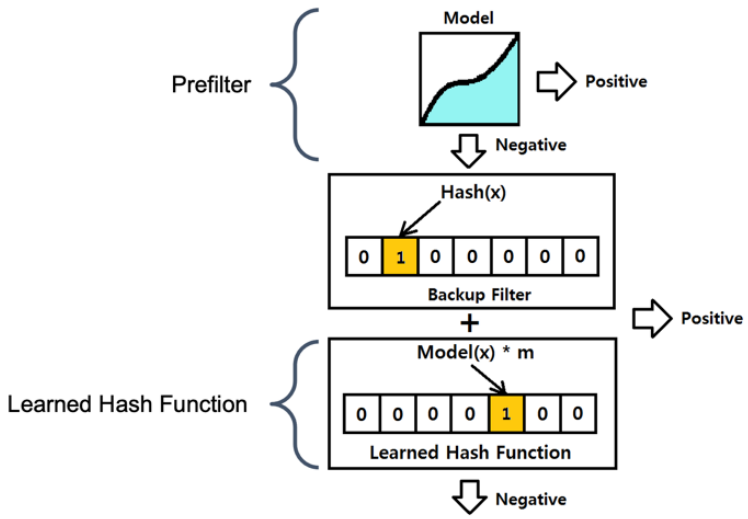
[그림 25] 인공신경망의 구조 비교

인공신경망을 모델로 사용하는 경우 계층에 변화를 주거나, 임베딩 차원을 조정해서 성능을 향상 시킬 수 있다. 모델 관점에서의 성능은 질의에 소요되는 시간과 모델에서 반환되는 음성의 개수를 말한다. 즉, 모델의 성능이 좋다면 전처리 과정에서 많은 양성 데이터가 분류되어서

음성으로 반환되는 데이터 양이 줄어든다. 반면에, 모델의 성능이 좋지 않다면 전처리 과정에서 적은 양의 데이터만을 분류하고 대부분의 데이터는 음성으로 반환되어 보조 필터에서 처리되어야 한다. The Case for Learned Index Structures[2]에서 제안한 모델은 32차원의 문자 임베딩, 16차원의 GRU 그리고 1차원의 완전 연결 계층으로 구성되었으나, 본 논문에서는 이와 다른 구조의 인공신경망을 소개한다.

모델 조정을 통해서 만들어진 새로운 모델은 16차원의 문자 임베딩, 16차원의 GRU, 8차원의 완전 연결 계층 그리고 1차원의 완전 연결 계층으로 구성되어있다. 즉, 문자 임베딩을 32차원에서 16차원으로 줄이고 8차원의 완전 연결 계층을 추가하였다. 위의 모델 조정을 통해서 기존에 4,017개 존재하던 모델의 파라미터 개수를 2,577개로 줄일 수 있었다. 따라서, 모델의 크기는 64비트 부동소수점을 기준으로 40킬로바이트(Kilobyte)에서 20킬로바이트로 감소하였다. 제안된 모델의 구조를 사용하면 확장 학습 블룸 필터는 개선된 거짓 양성 비율을 가지게 되지만, 인공신경망에 1개의 계층이 추가되어 질의하는 시간은 미비하게 증가하였다.

4.4 학습 해시 함수의 이해



[그림 26] 확장 학습 블룸 필터의 모델 이중 적용

확장 학습 블룸 필터에서는 모델을 이중으로 사용한다. 모델을 전처리 과정으로 한번 사용하고, 학습 해시 함수로 한번 더 사용하게 된다. 전처리 과정은 임계치를 기준으로 집합에 존재 여부를 분류하는

과정이고, 학습 해시 함수는 데이터를 저장하는 과정으로 사용되는 방식이다. 따라서, 전처리 과정은 전체 데이터 중에 일부를 분류하고 분류하지 못한 데이터를 보조 필터에 보내는데 반해, 학습 해시 함수는 입력 받은 모든 데이터에 대해서 저장하는 과정을 수행한다. 결과적으로 모델을 사용하는 동작 방식이 다르기 때문에 확장 학습 블룸 필터에서 모델을 이중으로 작용하는 것은 문제가 되지 않는다.

다음은 학습 해시 함수의 동작 방식에 대해서 이해해보도록 한다. 학습 해시 함수는 모델의 값을 사용하므로, 학습 해시 함수가 정상적으로 동작하기 위해서는 모델이 정상적으로 학습됨이 보장되어야 한다. 모델에서 사용하는 손실 함수는 교차 엔트로피(Cross Entropy)이다.

$$H(p,q) = -\sum_{x\in\chi} p(x) log\ q(x) \tag{9}$$

확장 학습 블룸 필터에서는 이진 분류의 모델을 필요로 하므로, 2개의 클래스에 대해서 분류하는 문제를 해결해야 한다. 2개의 클래스에 교차 엔트로피를 적용한 수식은 이진 교차 엔트로피(Binary Cross Entropy)라고 부른다. 손실 함수에 대한 이름은 다르지만, 결국에 교차 엔트로피를 사용하기 때문에 내부적으로는 동일한 의미를 가진다. 교차 엔트로피의 수식을 정리하면 분포 p에 대한 엔트로피에 쿨백-라이블러 발산을 분포 q에 대해서 최소화하는 문제로 해석할 수 있다. 여기서 분포 p는 실제 정답에 대한 분포이고, 분포 q는 모델이 예측한 분포이다. 쿨백-라이블러 발산은 분포의 차이를 나타내는 지표이므로, 이를 최소화한다는 뜻은 정답에 최대한 근접하게 모델의 파라미터의 값을 학습할 것을 의미한다.

$$H(p,q) = H(p) + D_{KL}(p||q) \tag{10}$$

결과적으로, 모델이 정상적으로 학습 된 후에는 양성 데이터의 모델 출력값은 1에 근접할 것이고, 음성 데이터의 모델 출력값은 0에 근접할 것임을 수식을 통해 이해할 수 있다. 학습 해시 함수의 거짓 양성 개수를 세는 수식을 술어(Predicate)으로 표현하는 다음과 같이 표현할 수 있다.

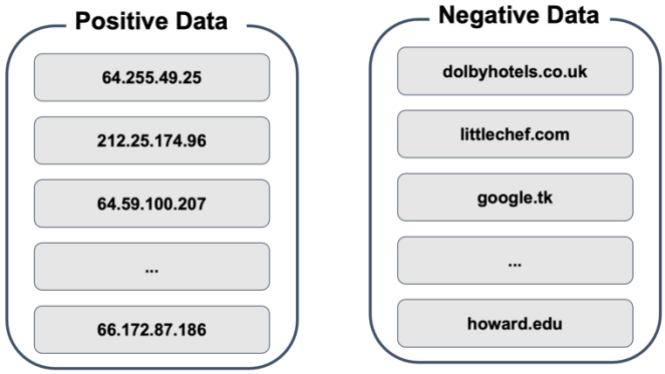
$$\sum_{i=1}^{\lfloor \alpha m \rfloor} (P \cap N \in i \rightarrow |N_i|) \wedge (\neg P \cap N \in i \rightarrow 0) \quad (11)$$

이는 학습 블룸 필터를 통해서 매핑된 양성 데이터와 음성 데이터에 대한 비트 벡터의 인덱스가 동일하다면, 해당 인덱스의 음성 데이터 개수 만큼의 거짓 양성이 발생한다는 의미를 가진다. 이진 교차 엔트로피를 통해서 양성 데이터는 1으로 근접되고, 음성 데이터는 0으로 근접하도록 모델을 학습이 해놓은 상태이다. 따라서 모델이 정상적으로 학습 된 경우에는, 거짓 양성 개수가 증가하기 위한 전제조건을 만족하지 못하게 되어 거짓 양성 비율이 개선될 수 있음을 직관적으로 이해할 수 있다.

제 5 장 실 험

이 장에서는 확장 학습 블룸 필터에 대한 실험 결과를 보이고 각 결과에 대해서 설명한다. 1절에서는 실험 환경에 대해서 설명한다. 2절에서는 하이퍼파라미터 탐색을 통해서 최적의 확장 학습 블룸 필터를 찾는 실험과 그 결과에 대해서 설명한다. 3절에서는 모델 정밀도에 의해서 발생하는 거짓 음성 현상에 대해서 알아본다. 4절에서는 모델 조정을 통한 확장 학습 블룸 필터의 성능 변화에 대해서 실험하고 그 결과에 대해서 설명한다. 5절에서는 모델의 학습으로 인해서 학습 해시 함수의 거짓 양성 비율이 낮아질 수 있음을 실험적으로 보인다.

5.1 실험 환경



[그림 27] Shalla's Blacklist 데이터

확장 학습 블룸 필터를 측정하는데 사용되는 실험 환경은 Intel Core i5 4690 @ 3.50Ghz CPU와 8GB Samsung DDR3 RAM 2개를 사용하였다. 사용한 소프트웨어는 Conda 4.10.1, Python 3.7.4, Tensorflow 2.4.1 그리고 Scikit-learn 0.24.1 버전을 사용하였다.

모델 학습을 위해서 사용된 데이터는 Shalla's Blacklist [2] 를 사용하였으며, 이는 URL 정보를 총 81개의 카테고리로 분류해 놓은 데이터이다. 모델에서는 이진 분류를 수행하므로, 성인용 카테고리와 그

[2] *Shalla Secure Services KG*, https://www.shallalist.de (2021.6.25 접속).

외의 카테고리로 데이터를 정제해서 사용한다. 결과적으로, 양성 데이터는 1,491,178개 그리고 음성 데이터는 1,435,527개를 사용한다.

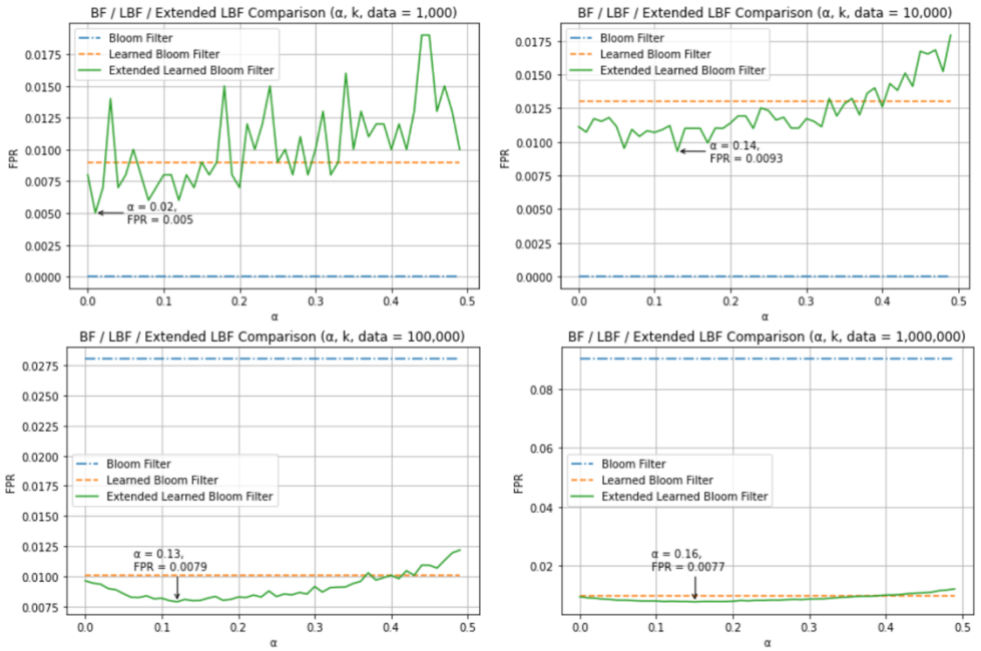
사용되는 인공신경망의 구조는 별도의 언급이 없으면, 본 논문에서 제안된 인공신경망 구조인 16차원의 문자 임베딩, 16차원의 GRU, 8차원의 완전 연결 계층 그리고 1차원의 완전 연결 계층을 사용해서 측정하였다. 문자 임베딩의 값은 glove.6B.50d.txt의 단어 임베딩을 문자 임베딩으로 변환해서 사용한다. 해당 GloVe는 50차원이므로, 16차원의 임베딩을 만들기 위해서는 주성분 분석(Principal Component Analysis)를 통해서 차원 축소를 수행한다. 해당 인공신경망의 손실 함수는 이진 교차 엔트로피를 사용하였으며, 옵티마이저(Optimizer)는 아담(Adam)[16]을 사용하였다. 추가적으로 배치 크기는 1024, 학습률(Learning rate) 0.005, 에폭(Epoch) 40으로 고정해놓고 모든 실험을 진행하였다.

5.2 하이퍼파라미터 탐색 실험

본 논문에서 제안된 확장 학습 블룸 필터에서는 하이퍼파라미터 α값이 존재하며, 해당 값의 변경이 성능에 어떠한 영향을 주는지 살펴본다. 하이퍼파라미터 탐색 실험은 공통적으로 α값을 0.01부터 0.50까지 총 50회씩 수행한다. (1 - α)의 비율로 존재하는 보조 필터는 최적의 해시 함수 개수를 사용하도록 설정한다. 즉, 주어진 공간에 대해서 제일 적은 거짓 양성 비율을 갖는 해시 함수 개수로 설정한다. 마지막으로, 목표로 하는 거짓 양성 비율은 0.01으로 설정해놓은 상태이다. 첫번째 실험은 데이터의 수를 바꿔가면서 블룸 필터, 학습 블룸 필터 그리고 확장 학습 블룸 필터의 거짓 양성 비율 성능을 측정한다. 학습, 개발, 테스트 데이터의 비율은 8:1:1으로 설정하였고, 모든 실험에 동일한 비율을 가지고 실험한다. 예를 들어 데이터를 1,000개 사용한다고 가정해보자. 800개의 양성과 음성 데이터를 모델의 학습을 위해서 사용하고, 100개의 음성 데이터를 모델의 임계치를 설정하는데 사용한다. 마지막으로, 100개의 음성 데이터를 통해서 거짓 양성 비율을 측정하게 된다. 데이터는 1,000개, 10,000개, 100,000개, 1,000,000개로 변경해가면서 실험을 진행하였다.

실험 결과를 보면 1,000개와 10,000개의 데이터가 사용된 경우 확장 학습 블룸 필터와 학습 블룸 필터는 블룸 필터보다 좋지 않은 거짓 양성 비율을 가지는 것을 알 수 있다(그림 28). 이는 확장 학습 블룸

필터에 사용되는 모델의 크기 때문에 발생하는 현상이며, 모델의 크기인 20KB가 1,000개 또는 10,000개의 데이터를 담는데 충분하기 때문이다. 따라서, 블룸 필터의 경우에는 거짓 양성 비율이 0으로 오류가 발생하지 않는다. 확장 학습 블룸 필터와 학습 블룸 필터의 개선이 일어나는 지점은 데이터를 100,000개를 사용했을 때이다. 확장 학습 블룸 필터와 학습 블룸 필터는 모두 블룸 필터 보다 개선된 거짓 양성 비율을 가진다. 또한, $\alpha = 0.13$으로 설정한 경우 목표로 하는 거짓 양성 비율인 0.01보다 21% 개선된 0.0079의 거짓 양성 비율을 가진다.

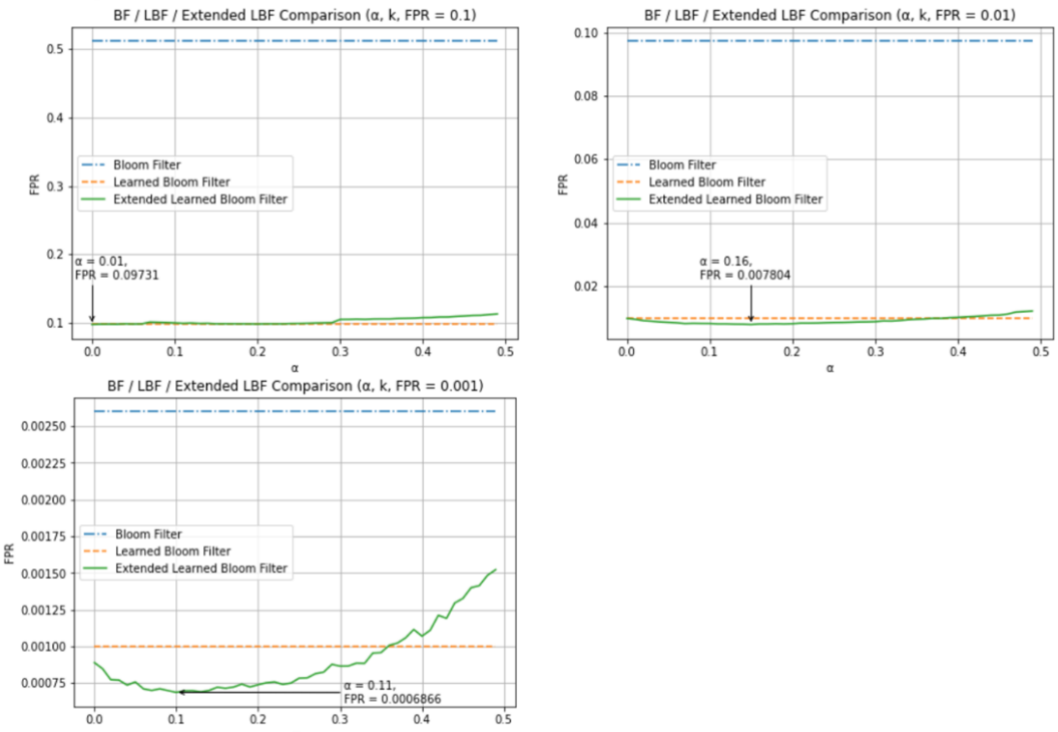


[그림 28] 데이터 개수를 변경한 하이퍼파라미터 탐색

두번째 실험으로는 목표로 하는 거짓 양성 비율을 0.1, 0.01 그리고 0.001으로 변경하는 실험이다. 데이터는 현재 가지고 있는 데이터를 모두 사용한다. 즉, 양성 데이터는 1,491,178개와 음성 데이터는 1,435,527개를 이용한다.

실험 결과를 보면 확장 학습 블룸 필터와 학습 블룸 필터가 블룸 필터에 비해서 좋은 거짓 양성 비율을 갖는 것을 확인할 수 있다(그림 29). 이는 실험에 많은 데이터가 사용되었기 때문에 나타나는 결과이다. 목표로 하는 거짓 양성 확률이 0.1과 같이 높은 경우에는 거짓 양성 비율에 개선이 없다고 봐도 무방한 0.9731이라는 값이 나왔다. 해당 현상은 보조 블룸 필터로 할당된 공간이 작기 때문에 발생한 현상이다. 거짓 양성 확률을 0.01으로 수행한 경우, 보조 블룸 필터로 할당되는

공간이 늘어나서 α = 0.16 지점에서 목표로 하는 거짓 양성 비율 0.01보다 22% 개선된 0.0078의 거짓 양성 비율을 가지는 것을 볼 수 있다. 즉, 모델에서 분류할 수 없는 데이터가 많아질수록 보조 블룸 필터에 할당되는 공간이 커지며, 해당 공간이 커질수록 확장 학습 블룸 필터가 효율적인 거짓 양성 비율을 가진다. 마지막으로, 거짓 양성 확률이 0.001으로 수행된 경우 목표로 하는 거짓 양성 비율인 0.001보다 31% 개선된 0.0006866의 거짓 양성 비율을 가진다.



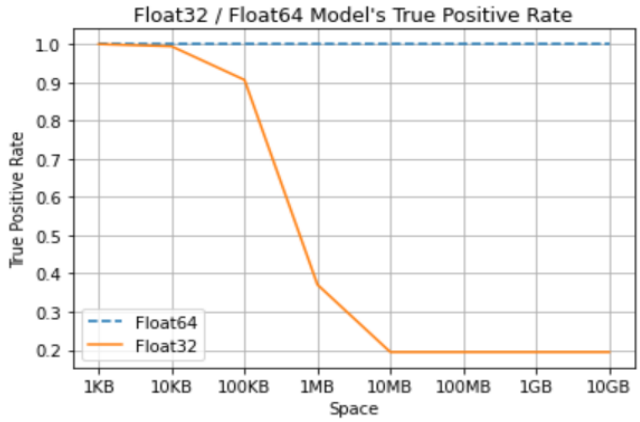
[그림 29] 거짓 양성 확률을 변경한 하이퍼파라미터 탐색

5.3 모델 정밀도 실험

모델의 정밀도 문제는 확장 학습 블룸 필터에서 거짓 음성을 발생시킨다. 즉, 입력을 했지만 질의 시 입력이 되지 않았다는 치명적인 오류가 발생한다. 다음은 보조 블룸 필터의 크기를 강제적으로 1KB, 10KB, … , 10GB으로 변경해가면서 보조 블룸 필터에서 발생한 거짓 음성 비율에 대해서 살펴본다. 단, 실험을 위해서 사용한 데이터는 100,000개이며 학습 해시 함수의 오류를 자세히 살펴보기 위해서 α = 0.99으로 설정하였다.

실험 결과를 보면 모델이 사용하는 부동소수점이 32비트였던

경우에는 10KB부터 거짓 음성 오류가 발생하기 시작하였다(그림 30). 양성 데이터는 참 양성 또는 거짓 음성으로 분류되기 때문에, y축인 참 양성 비율이 낮아지는 것은 거짓 음성이 발생하는 것으로 해석할 수 있다. 부동소수점의 유효 숫자는 6자리지만, 10KB는 비트로 변환하면 81,920비트로 5자릿수이다. 이는 반올림 오차(Rounding error)에 의해서 발생한다고 볼 수 있다. 10MB부터는 참 양성 오류가 0.1944로 수렴하고 더 이상 증가하지 않는다. 이는 모델 정밀도로 인해서 문제가 발생한 원소가 약 81%이고 나머지 약 19%는 정상적을 동작함을 의미한다. 반대로, 모델이 사용하는 부동 소수점이 64비트였던 경우에는 참 양성 비율이 항상 1.0으로, 거짓 음성 오류가 발생하지 않는 것을 알 수 있다.



[그림 30] 모델 정밀도 실험 결과

보조 필터 크기	Float64	Float32
1KB	1.0	1.0
10KB	1.0	0.9943
100KB	1.0	0.9064
1MB	1.0	0.3708
10MB	1.0	0.1944
100MB	1.0	0.1944
1G	1.0	0.1944
10G	1.0	0.1944

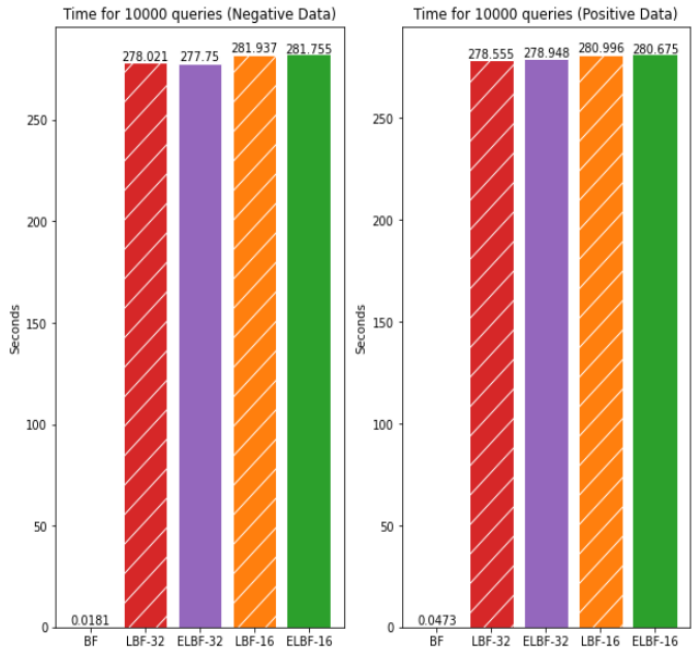
[표 1] 모델 정밀도 실험 결과

5.4 모델 조정 실험

다음은 확장 학습 블룸 필터에서 모델의 구조 변화로 성능 개선이 이루어질 수 있음을 보인다. 본 실험에서는 The Case for Learned Index Structures[2]에서 정의한 모델과 본 논문에서 제안한 모델을 비교해서 실험하도록 한다. 전자의 모델은 32차원의 문자 임베딩, 16차원의 GRU 그리고 1차원의 완전 연결 계층으로 구성되어 있으며, 후자의 모델은 16차원의 문자 임베딩, 16차원의 GRU, 8차원의 완전 연결 계층 그리고 1차원의 완전 연결 계층으로 구성되어 있다.

두 모델간의 가장 큰 차이점은 전자의 모델은 32차원의 임베딩을 사용하고, 후자의 모델은 16차원을 사용한다는 점이다. 따라서, 학습 블룸 필터를 전자의 모델로 생성한 경우 LBF-32의 명칭을 사용한다. 이와 반대로, 학습 블룸 필터를 후자의 모델로 생성한 경우 LBF-16의 명칭을 사용한다. 확장 학습 블룸 필터도 각 생성된 모델에 따라서 ELBF-32와 ELBF-16의 명칭을 사용한다.

본 실험은 블룸 필터, 학습 블룸 필터 그리고 확장 학습 블룸 필터의 성능지표를 CPU 소요 시간과 거짓 양성 비율을 기준으로 진행한다. 블룸 필터에 소요되는 시간을 확인할 때는, 질의되는 데이터 전체가 음성 데이터인 경우와 양성 데이터인 경우로 구분해서 실험을 진행한다. 이는 음성 데이터로 질의하는 경우에는 해시 함수의 개수가 k개인데도 불구하고, 일부의 해시 함수만 확인하고 측정된 시간이기 때문이다. 양성 데이터로 질의를 수행하는 경우에는 모든 해시 함수를 사용해서 집합에 존재함을 구분하기 때문에 블룸 필터가 질의하는데 가장 오래 걸리는 경우라고 볼 수 있다. 데이터를 총 100,000개 사용하고 10,000개의 음성 또는 양성 데이터를 질의를 수행했을 때 소요되는 CPU 시간을 살펴보도록 한다.

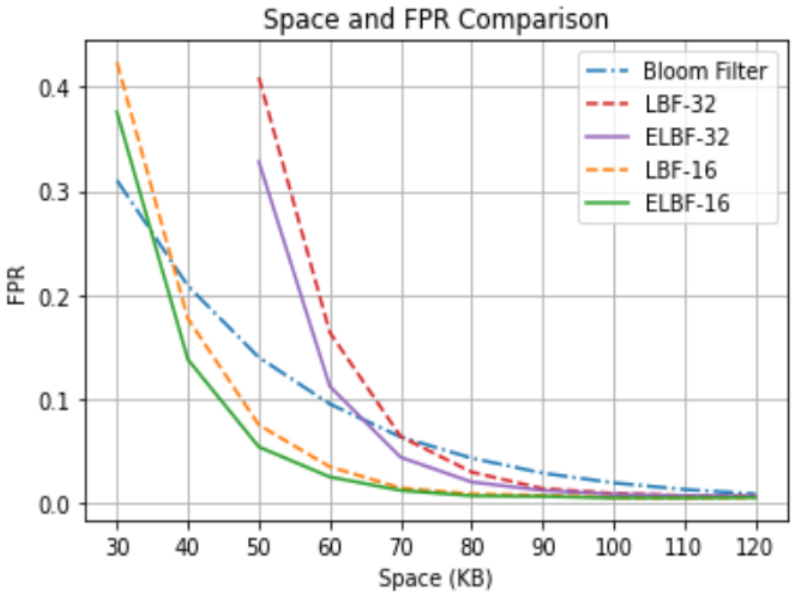


[그림 31] 모델 조정의 CPU 소요 시간 비교

실험 결과를 보면 블룸 필터가 음성 데이터와 양성 데이터 모두에 대해서 압도적으로 빠른 성능을 갖는 것을 볼 수 있다(그림 31). 블룸 필터와 학습 블룸 필터의 성능을 비교하면 블룸 필터가 음성 데이터를 기준으로 약 15,000배 그리고 양성 데이터를 기준으로 약 5800배 빠른 성능을 가진다. 시간은 학습 블룸 필터와 확장 학습 블룸 필터가 갖는 약점 중 하나이다. 본 논문에서 제안된 모델을 사용하는 LBF-16, ELBF-16은 LBF-32와 ELBF-32에 비해서 양성과 음성 데이터 모두에 대해서 2에서 3초 정도 느린 성능을 보였다. 시간이 중요한 어플리케이션은 블룸 필터를 사용할 것이기 때문에 1.5% 정도의 CPU 소요 시간 증가는 큰 문제가 아니라고 판단하였다.

다음 실험에서는 거짓 양성 비율에 대해서 실험을 수행하였다. The Case for Learned Index Structures[2]에서 제안된 모델은 40KB의 공간을 사용하고, 본 논문에서 제안된 모델은 20KB의 공간을 사용한다. 실험은 보조 블룸 필터의 크기를 10KB씩 증가하면서 실험을 진행한다. 즉, 모델에서 반환되는 원소의 개수와 무관하게 보조 블룸 필터의 크기를 강제로 10KB씩 증가하며 실험을 진행하였다. 데이터는 100,000개를 사용하고, 하이퍼파라미터 탐색 실험에서 구한 $\alpha = 0.13$을 사용해서 실험한다. 공간 측정시 주의할 점은 학습 블룸 필터와 확장 학습 블룸 필터의 크기는 모델의 크기와 보조 블룸 필터의 크기를 합한

값이라는 것이다.



[그림 32] 모델 조정의 거짓 양성 비율 비교

실험 결과를 보면 30KB 구간에서는 LBF-16과 ELBF-16 모두 블룸 필터에 비해서 거짓 양성 비율이 높다(그림 32). 즉, 성능이 좋지 않다고 볼 수 있는데, 이는 모델에서 사용되는 공간이 보조 블룸 필터에 비해서 크기 때문에 일어나는 현상이라고 볼 수 있다. 40KB 구간에서부터는 LBF-16과 ELBF-16은 블룸 필터보다 개선된 거짓 양성 비율을 갖는 것을 확인할 수 있다. 추가적으로, 확장 블룸 필터인 ELBF-16은 항상 LBF-16보다 개선된 거짓 양성 비율을 갖는 것 또한 확인할 수 있다. LBF-32와 ELBF-32는 50KB 구간에서부터 측정이 시작되는데, 이는 모델로 사용되는 공간이 40KB이기 때문이다. ELBF-32와 ELBF-16은 전체적으로 유사한 모양을 가지고 있으나, 20KB라는 모델의 크기 차이로 인해서 거짓 양성 비율 관점에서는 ELBF-16에 비해서 뒤처지는 거짓 양성 비율을 가진다. ELBF-16은 ELBF-32의 모델에 사용되는 20KB의 추가적인 공간을 보조 필터로 전환시켜서 거짓 양성 비율이 개선되었다고 볼 수 있다. 따라서, 모델의 성능 향상이 반드시 학습 블룸 필터와 확장 학습 블룸 필터의 거짓 양성 비율 개선에 도움이 된다고 할 수 없다. 모델에서 사용되는 공간과 보조 필터에서 사용되는 공간이 조화를 이루는 최적의 모델이 존재할 것으로 보인다.

공간	BF	LBF-32	ELBF-32	LBF-16	ELBF-16
30KB	0.3100			0.4233	0.3753
40KB	0.2087			0.1772	0.1379
50KB	0.1400	0.4080	0.3280	0.0751	0.0541
60KB	0.0953	0.1641	0.1123	0.0350	0.0252
70KB	0.0636	0.0640	0.0441	0.0146	0.0123
80KB	0.0433	0.0299	0.0204	0.0088	0.0071
90KB	0.0289	0.0141	0.0123	0.0075	0.0065
100KB	0.0196	0.0094	0.0086	0.0055	0.0050
110KB	0.0132	0.0074	0.0073	0.0051	0.0049
120KB	0.0088	0.0071	0.0069	0.0055	0.0054

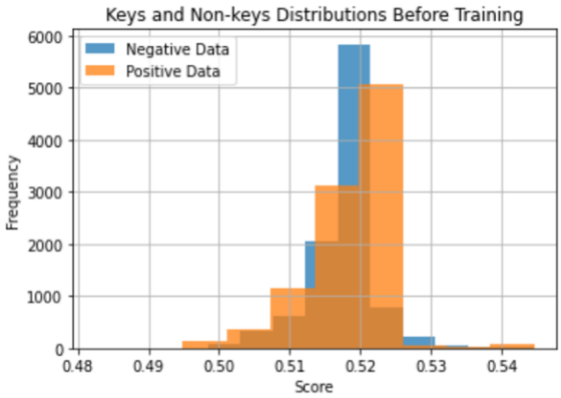
[표 2] 모델 조정의 거짓 양성 비율 비교

5.5 학습 해시 함수의 이해 실험

모델 조정 실험에서는 확장 학습 블룸 필터의 사용으로 거짓 양성 비율이 개선되었음을 보였다. 이번 실험에서는 학습 해시 함수가 어떠한 방식으로 거짓 양성 비율을 개선하는지에 대해서 살펴본다. 학습 해시 함수에서 거짓 양성 비율이 증가하는 경우는 양성 데이터가 음성 데이터의 인덱스가 충돌이 발생한 경우이다. 즉, 양성 데이터와 음성 데이터가 동일한 인덱스로 매핑되는 경우에 거짓 양성이 발생한다. 모델로 사용되는 인공신경망은 양성 데이터를 1로 학습시키고, 음성 데이터를 0으로 학습시킨다. 학습이 정상적으로 수행되면, 대부분의 양성 데이터는 1에 근접하는 값을 가질 것이다. 반대로, 음성 데이터는 0에 근접하는 값을 가질 것이다. 본 실험에서는 학습을 통해서 양성 데이터와 음성 데이터의 분포가 변하는지에 대해서 살펴보도록 한다. 학습 데이터는 100,000개를 사용하고 질의하는 데이터는 양성과 음성 데이터를 각각 10,000개씩 사용한다.

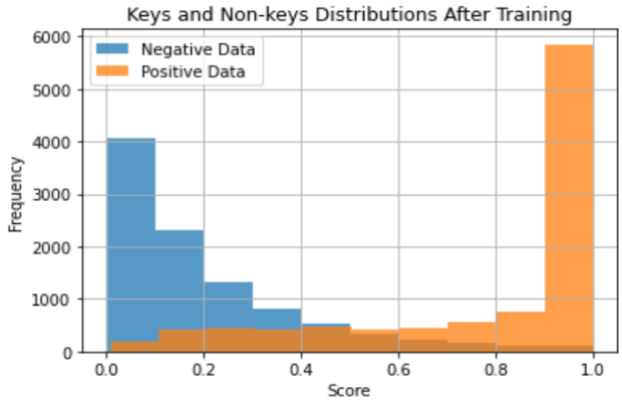
실험 결과를 보면 인공신경망을 초기화하고 학습이 안된 상태에서의 인공신경망의 출력값 분포를 나타낸다(그림 33). 양성 데이터와 음성 데이터 모두 [0.48, 0.54] 사이의 존재한다. 이는 데이터에 무관하게 유사한 출력값을 가진다는 것을 의미한다. 또한, 히스토그램을 통해서 양성과 음성 데이터가 유사한 분포를 가지는 것을 확인할 수 있다. 즉,

학습 전의 모델을 사용해서는 거짓 양성 비율이 개선되지 않을 것이라는 것을 추측할 수 있다.



[그림 33] 학습 이전의 양성과 음성 데이터 분포

실험 결과를 보면 학습 후에는 이진 교차 엔트로피에 의해서 양성 데이터는 1에 근접하고, 음성 데이터는 0에 근접하는 것을 볼 수 있다(그림 34). 이전과 다르게 모델의 출력값이 [0.0, 1.0] 사이에 값이 존재하여, 양성과 음성 데이터 간에 동일한 인덱스로 매핑되는 경우는 줄어들 것이다. 현재 히스토그램은 학습이 잘 되었음을 보기 위해서 구간을 10개로 나눠 놓은 상태이다. 실제로 확장 학습 블룸 필터에서 사용되는 학습 해시 함수의 공간은 80,000비트 이상의 크기를 사용하게 되는데, 이는 구간을 80,000개 이상으로 나누는 것으로 볼 수 있다. 따라서, 양성 데이터와 음성 데이터 간에 일치하게 되는 인덱스의 개수가 학습 후에 적어진다는 것을 직관적으로 이해할 수 있다.



[그림 34] 학습 이후의 양성과 음성 데이터 분포

제 6 장 결론 및 향후 연구

본 연구에서는 확장 학습 블룸 필터를 제안하였다. 이는 학습 해시 함수와 보조 필터의 공간을 비율로 표현한 하이퍼파라미터 α를 제안하여, 최적의 확장 학습 블룸 필터를 찾는데 도움을 준다.

확장 학습 블룸 필터의 구현과 관련해서는 모델의 정밀도 문제가 발생함을 보였으며, 이는 부동소수점으로 인해서 발생하는 문제이다. 해당 문제 발생한 경우, 확장 학습 블룸 필터에서 거짓 음성이 반환되는 버그가 생성된다. 이는 집합에 존재하는 원소가 저장은 되었으나, 부동소수점의 오차 범위로 인해서 다른 인덱스를 조회하게 되는 문제이다. 따라서, 1.2킬로바이트 이상의 학습 해시 함수를 할당한 경우에는 모델에서 사용하는 부동소수점을 32비트에서 64비트로 변경할 것을 제안한다. 추가적으로, 모델 조정을 통해서 The Case for Learned Index Structures[2]에서 정의한 모델보다 거짓 양성 비율을 개선하는 인공신경망의 구조를 소개하였다. 제안된 인공신경망의 구조로 모델의 성능이 향상이 반드시 확장 학습 블룸 필터의 성능 향상으로 이어진다고 볼 수 없음을 실험적으로 보였다.

향후 연구로는 확장 학습 블룸 필터인 최대 단점인 시간을 해결할 수 있는 방법을 찾을 수 있으면 좋을 것으로 기대된다. 또한, 학습 해시 함수는 현재 공간을 균일하게 사용하지만 거짓 양성 오류가 높은 공간에 대해서는 더 많은 공간을 할당해서 오류를 개선할 수 있는 가능성이 존재한다.

참고 문헌

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” Communications of the ACM 13, 7, 422–426, 1970.

[2] T. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, “The case for learned index structures,” In International Conference on Management of Data (ACM SIGMOD), 2018.

[3] P. O’Neil, E. Cheng, D. Gawlick, and E. O’Neil, “The log-structured merge-tree (LSM-tree),” Acta Inf., vol. 33, no. 4, pp. 351–385, 1996.

[4] N. Dayan, M. Athanassoulis, and S. Idreos. Monkey: Optimal Navigable Key-Value Store. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 79–94, 2017.

[5] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, “CRLite: A Scalable System for Pushing All TLS Revocations to All Browsers”, In IEEE Symposium on Security and Privacy, 2017.

[6] K. Cho, B. Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, In Proceedings of the Conference on Empirical Methods in Natural Language Processing(EMNLP), pages 1724–1734, 2014.

[7] M. Mitzenmacher. “A model for learned bloom filters and related structures,” CoRR, abs/1802.00884, 2018.

[8] A. Bhattacharya, B. Srikanta and B. Amitabha, “Adaptive Learned Bloom Filters under Incremental Workloads,” In Proc. of the 7th ACM IKDD CoDS and 25th COMAD. 2020.

[9] Q. Wang, Q. Wu, M. Zhang, R. Zheng, “Learned Bloom-Filter for an Efficient Name Lookup in Information-Centric Networking”,

2019 IEEE Wireless Communications and Networking Conference (WCNC), 2019.

[10] Q. Liu, L. Zheng, Y. Shen and L. Chen, “Stable learned bloom filters for data streams,” Proceedings of the VLDB Endowment, 2020.

[11] J. Ding, U. F. Minhas, H. Zhang, Y. Li, C. Wang, B. Chandramouli, J. Gehrke, D. Kossmann, and D. Lomet. “ALEX: An Updatable Adaptive Learned Index.” In SIGMOD Conference, pages 969–984. ACM, 2020.

[12] T. Kraska, M. Alizadeh, A. Beutel, E. Chi, A. Kristo, G. Leclerc, S. Madden, H. Mao, and V. Nathan, “SageDB: A Learned Database System.” In 9th Biennial Conference on Innovative Data Systems Research, CIDR, 2019.

[13] J. Pennington, R. Socher, and C. D. Manning. “Glove: Global vectors for word representation.” In Empirical Methods in Natural Language Processing (EMNLP), pages 1532– 1543, 2014.

[14] Z. Dai and A. Shrivastava. “Adaptive learned bloom filter (Ada−BF): Efficient utilization of the classifier,” CoRR, abs/1910.09131, 2019.

[15] K. Vaidya, E. Knorr and T. Kraska, “Partitioned Learned Bloom Filter,” arXiv preprint arXiv:2006.03176, 2020.

[16] P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv preprint arXiv:1412.6980, 2014.

[17] J. Bruck, J. Gao, and A. Jiang, “Weighted Bloom filter,” in 2006 IEEE International Symposium on Information Theory (ISIT’06), 2006.

[18] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, “Cuckoo filter: Practically better than bloom.” In CoNEXT, pages 75–88, 2014.

[19] R. Pagh and F. F. Rodler. “Cuckoo hashing,” In Journal of Algorithms, pages 122–144, 2004.

[20] T. M. Graf and D. Lemire, “Xor Filters: Faster and Smaller Than Bloom and Cuckoo Filters,” Journal of Experimental Algorithmics (JEA) Vol. 25, No. 1, pages 1–16, 2020.

[21] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, “Optimizing Bloom filter: Challenges, solutions, and comparisons,” IEEE Communications Surveys & Tutorials, 2019.

[22] M. Mitzenmacher. “A model for learned bloom filters and optimizing by sandwiching,” In Advances in Neural Information Processing Systems, 2018.

[23] J. W. Rae, S. Bartunov, and T. P. Lillicrap, “Meta−learning Neural Bloom Filters,” ICML, 2019.

Appendix

A 블룸 필터의 변형

해당 부분에서는 본 논문에서 다루지 못한 블룸 필터의 변형에 대해서 살펴보도록 한다.

A.1 가중 블룸 필터

가중 블룸 필터(Weighted Bloom Filter)[17]는 질의의 빈도수에 따라서 해시 함수의 개수를 다르게 설정하는 블룸 필터이다. 따라서, 질의의 빈도가 높은 경우에는 해시 함수의 개수를 많이 사용하여 거짓 양성이 발생하지 않도록 방지한다. 반대로, 질의의 빈도가 낮은 경우에는 해시 함수의 개수를 적게 할당하여 비트 벡터에 입력되는 비트의 수를 줄인다. 이는 비트 벡터의 입력이 많을수록, 원소간의 충돌이 일어나서 거짓 양성 비율이 늘어날 수 있기 때문이다.

결국에 해당 자료구조는 질의가 많은 데이터(Hot data)와 질의가 적은 데이터(Cold data)로 이진 분류하고, 분류된 카테고리에 따라서 해시 함수의 개수를 조정한다. 질의가 많은 데이터의 경우에는 약 10 - 23 개의 해시 함수를 사용하고, 질의가 적은 데이터의 경우에는 약 1 - 10 개의 해시 함수를 사용하면 최적의 성능이 나오는 것을 실험적으로 보인다. 위 자료구조의 성능은 질의가 많은 데이터와 적은 데이터의 비율에 따라서 성능이 변할 수 있다.

A.2 쿠쿠 필터

쿠쿠 필터(Cuckoo Filter)[18]는 쿠쿠 해싱(Cuckoo Hashing)[19]을 기반으로 생성된 자료구조이다. 기존의 해시 함수는 충돌이 일어나면, 이를 오류로 간주하고 별도의 조치를 취하지 않는다. 쿠쿠 해싱은 해시 테이블 안에서 값이 다른 원소 간에 동일한 인덱스를 가지게 되는 경우, 새로운 인덱스를 생성하여 충돌을 방지한다. 충돌을 방지하기 위해서는 2 개 이상의 해시 테이블을 사용해야 하지만, 구현의

단순함 때문에 대부분 2 개의 해시 테이블을 사용해서 쿠쿠 해싱을 구현한다.

예를 들어, 첫번째 원소가 첫번째 해시 테이블의 해시 함수를 사용하여 인덱스 5 에 입력 되었다고 가정하자. 입력하고자 하는 두번째 원소 또한 첫번째 해시 테이블의 해시 함수를 사용해서 인덱스 5 로 입력되면, 두번째 원소를 그대로 첫번째 해시 테이블의 인덱스 5 로 저장하고 자리를 빼앗긴 첫번째 원소를 두번째 해시 테이블의 해시 함수를 사용하여 새로운 인덱스를 찾아준다. 위와 같이 인덱스를 빼앗긴 첫번째 원소를 다시 접근할 수 있는 이유는 자료구조가 해시 테이블이기 때문이다. 즉, 블룸 필터처럼 원소의 존재 여부를 비트 벡터에 저장하는 것이 아닌 해시 테이블은 원소 자체를 저장하기 때문에 입력 후에도 원소를 접근할 수 있다. 단, 입력 도중에 해시 테이블의 인덱스 간에 순환(Loop)이 생길 수 있는데, 해당 경우에는 해시 테이블을 새롭게 생성해서 입력을 처음부터 다시 입력해야하는 경우가 생길 수 있다.

쿠쿠 필터는 해시 테이블에 사용되던 쿠쿠 해싱에 변형을 주어 블룸 필터와 유사하게 동작하도록 바꾼다. 쿠쿠 필터는 지문(Fingerprint)을 이용해서 데이터를 저장할 수 있다. 지문은 해시 함수의 출력값을 사용자가 지정한 자릿수만큼 이용하는 것을 말한다. 예를 들어, 지문을 3 비트로 설정했다면 모든 원소에 대해서 반환된 해시값의 첫 3 자리의 비트가 지문이다. 따라서 블룸 필터에서 하나의 원소 표현하기 위해서 1 개의 비트를 사용했다면, 쿠쿠 필터에서는 사용자가 정의한 지문의 크기 만큼의 비트를 할당해서 하나의 원소를 표현한다. 쿠쿠 필터는 지문의 크기로 할당된 배열을 사용하고, 해당 배열에서 수용할 수 있는 지문의 개수를 버킷(Bucket)의 크기라고 부른다. 또한, 버킷의 크기는 목표로 하는 거짓 양성 확률에 따라서 결정된다.

쿠쿠 필터의 입력은 부분-키 쿠쿠 해싱(Partial-key Cuckoo Hashing)을 통해서 이루어진다. 기존의 쿠쿠 해싱은 충돌이 발생한 경우 해시 테이블의 원소를 접근하여 새로운 인덱스를 생성할 수 있다. 하지만, 지문을 입력하는 쿠쿠 필터의 특성상 입력된 원소가 존재하지 않기 때문에 새로운 인덱스를 생성할 수 없다는 문제가 발생한다. 부분-키 쿠쿠 해싱은 원소 대신에 지문을 접근하여, 새로운 인덱스를 만들 수 있게 해주는 해싱 기법이다. 부분-키 쿠쿠 해싱은 쿠쿠 해싱과 동일하게 충돌이 발생한 인덱스에 대해서 새로운 인덱스를 할당하여 공간을 블룸 필터에 비해서 효율적으로 사용할 수 있게 된다.

결과적으로, 거짓 양성 비율 0.03 이하에서는 블룸 필터보다 개선된 성능을 보인다.

A.3 XOR 필터

XOR 필터[20]는 3 개의 해시 함수와 지문을 이용해서 블룸 필터 또는 쿠쿠 필터보다 개선된 성능을 갖는 것을 목표로 한다. XOR 필터가 3 개의 해시 함수만 사용하였지만, 개선된 성능을 가질 수 있는 이유는 비순환 삼분 그래프로 구성된 랜덤 하이퍼그래프(Acyclic 3-Partite Random Hypergraph)를 사용하였기 때문이다.

삼분 그래프의 각 노드(Node)는 3 개의 색으로 이루어져 있고, 입력되는 원소를 노드로 할당해주는 역할을 해시 함수가 수행한다. 각 해시 함수는 하나의 색을 담당하게 되어 총 3 개의 해시 함수가 사용된다. 단, 위 그래프는 동일한 색의 노드 2 개로 만들어진 엣지(Edge)를 허용하지 않는다. 임의의 해시 함수 3 개를 골라서 입력하고자 하는 집합에 대해서 비순환 삼분 하이퍼그래프가 생성되면 XOR 필터가 생성된 것이고, 비순환 삼분 하이퍼그래프가 생성되지 않으면 다시 3 개의 랜덤한 해시 함수를 선택해서 그래프 생성을 시도한다.

정리하자면, 위의 과정은 입력하고자 하는 원소에 대한 해시 함수의 출력값 3 개를 XOR 연산하였을 때, 참이 나오도록 인위적으로 그래프로 만드는 것이다. 결과적으로, XOR 필터는 블룸 필터와 쿠쿠 필터에 비해서 개선된 거짓 양성 비율과 질의 시간을 보인다.

A.4 블룸 필터 최적화

블룸 필터 최적화(Optimizing bloom filter)[21]는 여러 종류의 블룸 필터 개선 방향에 대해서 정리된 논문이다. 블룸 필터의 연구방향은 대부분 응용 또는 성능적인 측면으로 진행되었다. 성능적인 측면은 2 가지로 나눠서 거짓 양성 비율 또는 구현 비용으로 볼 수 있다.

블룸 필터는 블룸 필터의 크기, 입력 집합, 질의 집합, 해시 함수의 개수, 해시 함수의 구현, 거짓 양성 오류로 구성되어 있다. 블룸 필터와 관련된 연구는 거짓 양성 오류를 개선하는 방향의 연구 또는 메모리와

연산을 최소화하는 연구가 있을 수 있다. 이와 별개로 집합의 크기가 동적으로 변한다면, 블룸 필터의 크기 또한 유연하게 조정되도록 하는 연구도 존재한다. 마지막으로, 입력 또는 질의 외의 실용적인 연산자를 추가하는 방향의 연구가 있을 수 있다.

예를 들어, 거짓 양성 오류는 사전 지식(Prior knowledge)의 사용, 비트 벡터의 값 재설정, 특정 해시 함수를 이용하거나 문제의 표현을 바꿔서 개선할 수 있을 것이다. 위에서 언급한 가중 블룸 필터는 사전 지식을 사용한 방식으로 볼 수 있으며, 부분-키 쿠쿠 해싱은 비트 벡터의 값 재설정으로 볼 수 있다. 또한, XOR 필터는 문제의 표현을 하이퍼그래프로 바꿔서 거짓 양성 오류를 개선한 방식이라고 볼 수 있다.

B 학습 블룸 필터의 변형

해당 부분에서는 본 논문에서 다루지 못한 학습 블룸 필터의 변형에 대해서 살펴보도록 한다.

B.1 샌드위치 학습 블룸 필터

샌드위치 학습 블룸 필터(Sandwiched Learned Bloom Filter)[22]는 Partitioned Learned Bloom Filter 이전에 수행되었던 연구다. 학습 블룸 필터가 모델과 보조 블룸 필터로 2 개의 계층으로 구성되어 있다면, 샌드위치 학습 블룸 필터는 보조 필터, 모델 그리고 보조 필터로 총 3 개의 계층으로 구성된다. 모델 앞에 보조 필터를 추가적으로 넣은 이유로는 모델에서 거짓 음성이 반환되는 것을 방지하기 위함이다.

위 논문에서는 최적의 공간 할당을 위해서 1 계층에서 사용되는 보조 필터에 대부분의 공간이 할당되어야 함을 수식으로 보인다. 또한, 2 계층에서 사용되는 모델의 거짓 양성 오류가 늘어날수록 거짓 양성 비율이 최적화에 가까워짐을 수식으로 보인다. 즉, 샌드위치 학습 블룸 필터에 성능이 약한 모델이 사용되어도 된다는 의미이다. 결과적으로 샌드위치 학습 블룸 필터는 추가적인 보조 필터 사용으로 학습 블룸 필터보다 개선된 거짓 양성 비율을 갖는다.

추가적으로, 학습 블룸 필터 또는 블룸 필터는 질의 데이터에 의해서 거짓 양성 비율이 달라진다는 점을 지적한다. 즉, 질의에 사용되는 원소가 10,000 개이지만 동일한 원소로만 존재하는 경우에는 하나의 원소에 종속적으로 성능이 측정된다는 것이다. 따라서, 거짓 양성 비율 측정을 위해서는 임의로 선택한 데이터를 질의 데이터로 사용하여 측정하며, 블룸 필터 또는 학습 블룸 필터에 대한 적대적인(Adversarial) 질의 데이터가 존재할 수 있음을 인지해야 한다.

B.2 메타-학습 뉴럴 블룸 필터

메타-학습 뉴럴 블룸 필터(Meta-Learning Neural Bloom Filters)[23]는 기존의 학습 블룸 필터와 구조가 동일하나 모델을 메타-학습을 사용하는 인공신경망 모델을 사용할 것을 제안한다. 기존의 순환 신경망인 LSTM(Long Short Term Memory) 또는 DNC(Differentiable Neural Computer)는 BPTT(Backpropagation Through Time)를 통해서 학습이 수행되기 때문에, 입력하고자 하는 원소가 많아질수록 학습 또한 어려워진다는 문제가 존재한다. 따라서, 해당 논문에서는 순환 신경망 기반의 네트워크 대신에 슬롯 기반의 메모리 네트워크를 사용할 것을 제안한다. 슬롯 기반의 메모리 네트워크는 BPTT 를 사용하지 않고, 병렬적으로 경사도(Gradient)를 측정할 수 있다는 장점이 있다.

메타-학습 뉴럴 블룸 필터는 블룸 필터와 유사한 동작을 하도록 메모리 기반의 인공신경망 구조에 변화를 줬으며, 원샷 메타-학습(One-Shot Meta-Learning)을 통해서 학습이 수행된다. 결과적으로 5,000 개의 문자열을 0.01 의 거짓 양성 비율로 저장하기 위해서 쿠쿠 필터는 45.3 킬로바이트, 블룸 필터는 47.9 킬로바이트를 사용하는데 반해, 뉴럴 블룸 필터는 1.5 킬로바이트로 대폭 감소된 공간 사용량을 보인다. 하지만, 여전히 시간적으로는 블룸 필터에 비해서 400 배 정도 느린 성능을 가진다. 뉴럴 블룸 필터는 하드웨어로 GPU 가 제공되며, GPU 의 사용으로 대용량의 배치성 쿼리가 수행되는 환경에서 블룸 필터에 비해서 이점이 있을 수 있다.

Abstract

Efficient Implementation of Extended Learned Bloom Filter

Soohyun Yang
Department of Computer Science and Engineering
The Graduate School
Seoul National University

Existing data structures aim to have constant performance regardless of data distribution. However, a study is being conducted under the name of learned index, that the performance can be improved by the use of data distribution.

In this study, we focus on extending and implementing learned bloom filter, which is one of the types of learned index. This is called as an extended learned bloom filter, and it is a data structure which adds a learned hash function into the structure of learned bloom filter. Experiments show that false positive rate can be improved through changing the ratio of learned hash function and the auxiliary filter using the hyperparameter α.

Additionally, we introduce the model precision problem that occurred during the implementation of the extended learned bloom filter, which can be solved by using 64-bit floating point. In addition, we show that the performance of the extended learned bloom filter can be improved by tuning the model, and we try to understand how the learned hash function contributes to performance improvement.

Keywords : Learned Bloom Filter, Learned Hash Function, Learned Index
Student Number : 2019-25475