2018년도 ICT 융합특론 기말고사

1. Neural Network Model의 Hypothesis를 기술
2. Neural Network Model에서 Back Propagation Algorithm은 무엇을 계산하기 위한 알고리즘인지? 그리고 이것을 계산해서 어디에 어떻게 사용할 수 있는지?
3. Neural Network Model의 weight matrix의 초기 값을 모두 0으로 설정하면 어떻게 되는지?
4. Gradient Checking은 무엇이고, 왜 해야 하는지? 그리고 이것은 Neural Network Model에서만 사용하는지?
5. DNN은 Hidden layer가 많은 Neural Network Model을 의미하는데, 과거에는 Vanishing gradient 같은 문제로 어려웠지만, 최근에는 이러한 문제를 잘 극복해서 많은 성과를 내고 있다. 학습하기 어려웠던 DNN 모델을 학습하기 위해 적용한 아이디어 3가지 이상을 기술
6. ~~Bagging 기법에 대해서 설명하고, 이를 적용할 때 얻을 수 있는 장점은 무엇인지 설명~~
7. ~~Boosting 기법에 대해서 설명하고, 이를 적용할 때 얻을 수 있는 장점은 무엇인지 설명~~
8. ~~Random Forest 알고리즘에서는 기존 Bagging 기법의 Tree correlation 문제를 해결하기 위해 Random vector method를 사용했다고 한다. Bagging 기법의 Tree correlation 문제를 설명하고, Random Forest에서는 이를 어떻게 극복했는지 설명~~
9. ~~Random Forest 알고리즘에서 Variable Importance를 측정하는 방법에 대해서 설명~~
10. K-means clustering 알고리즘의 pseudo code 작성
11. Unsupervised learning이 supervised learning에 비해 어려운 이유는?
12. Structured data, semi-structure data, unstructured data에 대해 각각 설명하고, 차이점 기술
13. Unstructured data를 활용하는 것이 중요한 이유는?
14. Text mining에서 TF-IDF에 대해서 자세히 설명
15. Collaborative Filtering에서 Cold start problem이 무엇인지 설명하고, 어떻게 극복할 수 있는지 설명

2019년도 ICT 융합특론 기말고사

1. Regularized Logistic Regression의 Cost Function이 주어졌을 때, Gradient 계산
2. Regularization을 하는 이유는 무엇이고, 이를 하면 학습한 모델이 어떻게 달라지는가?
3. Regularized cost function에서 Lambda 값의 의미는 무엇이고, 어떻게 결정하는가?
4. 지문을 분류하는 Neural Network Model을 만든다고 할 때, Hidden layer를 2단계로 만들고, 각 Hidden layer에 노드를 각각 100개씩 만드는 구조를 개략적으로 그림을 그려보아라. 그 후, Neural Network Model 파라미터 matrix의 Dimension을 설명
5. 이 Neural Network Model을 학습하기 위한 학습 데이터를 10,000개 생성한다고 할 때, 학습 데이터를 어떻게 수집해서 어떤 형태로 변환하여 저장할 것인가
6. 이 Neural Network Model을 학습하는 과정을 설명. Forward, Back propagation은 이미 구현되어 있다고 가정하고, 어떻게 활용해서 학습하는지 설명. (학습을 위한 과정을 순서대로)
7. Neural Network Model에서 Gradient Checking을 하는 이유와 방법을 설명
8. Gradient Checking에서 사용하는 Gradient 근사값을 직접 학습에 사용하지 않는 이유 설명
9. ~~Bagging과 Boosting 알고리즘을 각각 설명하고, 둘의 차이점에 대해서 설명~~
10. ~~Random Forest에서 Tree correlation이 무엇인지 설명하고, 이를 어떻게 해결하는지 설명~~

11. Random Forest를 예측 모델 외에도 활용할 수 있는 방법이 있다면 2가지 이상으로 설명
12. 군집 분석이 활용될 수 있는 구체적인 사례 2가지 이상 설명
13. K-means 알고리즘의 Pseudo code 작성
14. WSS와 BSS의 의미와 계산하는 방법을 설명하고, 군집 분석을 평가할 때 어떻게 사용할 수 있는지 설명
15. Text mining 과정 설명
16. TF-IDF 설명 및 계산 과정

1. 텍스트 마이닝의 절차

   일반적인 텍스트 마이닝의 과정은 데이터 수집 -> 어휘 추출 (형태소 분석, TF/IDF) -> 정보 추출 (필요한 정보 추출) -> 정보 분석 (빈도, 분류, 클러스터링 등)의 과정으로 이뤄진다. 이를 조금 더 자세히 살펴보면, 크롤링 등을 통해 수집된 데이터가 있다고 가정하고, ETL 과정을 통해 Tidy text와 같은 구조화된 텍스트로 만들어야 한다. 이 과정을 위해 가장 먼저는 텍스트에서 meaningful unit 이라고 할 수 있는 token을 만든다. 대부분의 token은 single word 이고, 이렇게 token으로 쪼개는 것을 Tokenization 이라고 한다. (unnest_token) 이 과정에서는 보통 구두점을 제거하고, 소문자로 변경한다. 그 후에는 정규표현식과 anti_join 등을 활용하여 Stop-words를 제거해주도록 한다.

2. 군집 알고리즘 설명 및 비교

   K-means는 거리 기반 알고리즘으로, 가까운 거리에 있는 데이터들을 같은 군집으로 배정하는 알고리즘이다. 또한 사용자로부터 군집의 개수를 입력 받아서 해당 개수만큼 군집을 반환한다. 장점은 데이터에 대한 사전 정보가 없어도 어느 정도의 성능을 낼 수 있지만, 클러스터의 개수 k를 결정하는 것이 힘들고 이 값에 따라 결과 차이가 크다. 알고리즘 작동 방식은 중심점의 개수인 k를 입력 받으면, k개 만큼의 중심점을 Random 하게 만든다. 그리고 각 중심점에 대해 모든 데이터들의 거리를 계산해서 가장 거리가 가까운 중심점에 해당하는 군집을 할당 시킨다. 그리고 같은 군집에 해당하는 데이터의 평균값을 통해 중심점을 이동시켜서 위 과정을 다시 반복한다. 그 후 중심점이 이동하지 않거나, 군집이 변화하지 않는다면 탐색을 중단한다.

   계층적 군집은 고차원 혹은 다차원 데이터를 군집화 하는 알고리즘이다. 이는 가까운 데이터끼리 클러스터를 형성하고, 가까운 클러스터끼리 군집을 병합하는 과정을 반복해서 하나의 클러스터가 되게 하는 방식이다. 일반적으로 사용하기 간단하고, 직관적이다. 또한 Dendogram에서 적절한 포인트로 자르면 되기 때문에 클러스터 개수에 대한 가정이 필요 없다.

   DBSCAN은 밀도 기반 알고리즘으로, 특정 반경 (Epsilon) 안에 몇 개의 데이터 (MinPts)가 존재하는지에 따라 군집으로 포함시킬지 여부를 결정한다. 이 알고리즘에서 데이터는 Core point, Border point로 나뉘는데, Core point는 Epsilon 범위 내에 있으면서 MinPts 개수를 만족하는 점이고, Border point는 Epsilon 범위 내에 있지만 MinPts 개수를 만족하지 못하면서 Core point의 이웃에 해당하는 점이다. 한편, Noise point는 Core와 Border point가 아닌 점을 의미한다. 알고리즘의 작동 방식은 Epsilon과 MinPts가 설정되면, 현재 군집의 개수를 0으로 설정하여 p 데이터에 대해 core point가 아니라면 Null label을 할당하고, core point라면 새로운 클러스터를 만들고, Epsilon 범위 내에 있는 모든 점을 해당 클러스터로 할당해준다. 그리고 이 과정들을 모든 점에 대해서 반복하면서 Core와 Border point가 아닌 데이터는 Nosie 데이터로 할당해주도록 한다. DBSCAN 알고리즘은 이상치에 대해서는 Noise 데이터로 구분할 수 있으며, 비선형 모양의 군집이 생성될 수 있다는 장점이 있지만, Epsilon과 MinPts 파라미터를 결정하는데 어려움이 있고, 학습에서의 시간이 오래 소요되는 단점이 있다.

3. Recommender System – Collaborative (UBCF, IBCF), Content-based Filtering

추천 시스템의 장점은 Conversion rate를 증가시키거나, 고객의 Loyalty를 증가시킬 수 있다는 장점이 있다. 추천 시스템은 크게 두 가지 종류로 나눌 수 있는데, 먼저 Content-based Filtering 방법은 Attribute나 Features 등을 활용하여 유사한 부분들을 찾아서 추천해준다. 이 방식의 장점은 유저의 선호도에 대한 정보 없이도, 아이템 정보만으로도 추천이 가능하다는 것이고, 단점은 의미 있는 Feature라는 것이 주관적이고, 새로운 종류를 추천 해주기 어렵다는 것이다. 두번째 Collaborative Filtering 방법은 비슷한 취향을 갖는 다른 사람들의 정보를 통해서 선호도를 파악하는 방법이다. 이 방식은 고객들의 선호도가 변하지 않는다는 가정을 가지고 있다. 이 방식의 장점은 아이템에 대한 콘텐츠 정보 없이도 활용할 수 있지만, Cold start 문제나, 유저와 아이템 수가 많아지면 데이터 크기가 기하급수적으로 커진다는 단점이 있다.

Collaborative Filtering 방법은 그 안에서 UBCF(User-based)와 IBCF(Item-based) 방식이 있다. UBCF 모델은 유사한 사용자의 선호를 바탕으로 추천해주는 방식이다. A 사용자가 라면, 삼각김밥, 김치, 콜라를 구입했다면, 라면과 삼각김밥, 김치를 구입하는 B 사용자에게 콜라를 추천해주는 방식이다. 이 모델의 단점은 모든 사용자들과 유사도를 매번 계산해야 되기 때문에 실시간 추천에 한계가 있다.
IBCF 모델은 아이템 간의 유저 목록을 비교하여 추천하는 방법이다. 이 방식은 모델을 위한 matrix가 상대적으로 작아서 UBCF 보다 조금 더 경제적이라는 장점이 있다.

일반적으로 추천 시스템에서는 Cold start 문제가 이슈가 되는데, 이는 새로운 또는 어떤 유저들에 대한 충분한 정보가 수집된 상태가 아니기 때문에 해당 유저들에게 적절한 제품을 추천해주지 못하는 문제를 이야기한다. 일반적으로 신제품이 출시되거나, 새로운 유저들이 가입했을 때 이러한 문제가 주로 발생한다고 할 수 있다. 이를 해결하는 방법은 유저의 초기 프로필을 완성시키거나, 인기 있는 제품들을 먼저 추천하여 데이터를 쌓는 방법 등이 있다.

Text mining

- 데이터의 종류

  일반적으로 데이터는 세 가지 종류로 나눠져 있다. 가장 먼저는 정형 데이터 (Structured data)로 schema-first 성격을 갖으며 잘 구조화 된 데이터라고 할 수 있다. 그래서 관계형 데이터베이스나, DBMS, CSV 파일 등이 이해 해당한다. 두번째로는, 반정형 데이터 (Semi-structured data)로 schema-later 의 성격을 갖으면서 형태만 존재하고, 연산이 불가능한 데이터를 의미한다고 할 수 있다. 이에 해당하는 것은 XML, HTML, JSON 등의 파일이다. 마지막으로는, 비정형 데이터 (Unstructured data)로 schema-never 의 성격을 갖고, 형태도 없고, 연산도 불가능한 데이터이다. 보통 영상이나 이미지, 음성, 텍스트 등의 데이터를 의미한다. 실제 데이터 중 약 80% 이상이 이 데이터이기 때문에 중요하다고 할 수 있다.

- 텍스트 마이닝

  테이블은 한 row 당, 한 개의 token 을 가지고 있는데, 여기서 token 은 word 와 같이 단어의 유의미한 unit 이라고 할 수 있고, 텍스트를 token 으로 나누는 것을 tokenization 이라고 한다. 일반적으로 tokenization 과정은 single word 를 뜻한다고 할 수 있다. 이 작업이 끝난 후에는 구두점이나 stopwords 를 제거해주도록 한다.

- TF/IDF 개념 –> Document-term matrix

  TF/IDF 개념을 이해하기 위해서는 TF 와 IDF 개념을 각각 이해해야 한다.

  TF 는 Term Frequency 로 해당 문서에서 등장한 특정 단어의 빈도를 의미한다. '해당 문서에서의 특정 단어 수 / 해당 문서에서의 전체 단어 수'로 계산된다. DF 는 Document Frequency 로 특정 단어가 각 문서에 얼마나 나왔는지 빈도를 의미한다. 따라서 TF 는 단어의 빈도, DF 는 문서의 빈도를 의미한다고 할 수 있다. IDF 는 Inverse Document Frequency 로 DF 의 역수이고, 이를 통해 큰 값을 작은 값으로 줄여서 많이 등장하는 단어를 패널티를 주도록 한다. 이는 'ln(전체 문서 수 / 특정 단어를 포함하고 있는 문서의 수)'으로 계산된다.

  TF/IDF 는 TF 와 IDF 의 곱으로 계산되고, 많은 문서들 안에 있는 토큰을 통해 특정 문서를 대표하는 키워드를 추출하는 방법으로 이해할 수 있다. 이 값이 크다는 것은 다른 문서에는 많지 않고, 해당 문서에만 자주 등장하는 것임을 뜻한다.

  이 개념이 활용되는 사례는 문서의 유사도를 구하거나, 검색 시스템에서 검색 결과의 중요도를 정하는 작업, 문서 내에서 특정 단어의 중요도를 구하는 작업에서 활용된다.

문서1 : 먹고 싶은 사과
문서2 : 먹고 싶은 바나나
문서3 : 길고 노란 바나나 바나나
문서4 : 저는 과일이 좋아요

이를 문서 단어 행렬로 표현하면 다음과 같습니다.

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

- Topic Modeling

  LDA (Latent Dirichlet Allocation)는 확률 분포로부터 토픽을 찾아내는 방법으로, 특정 토픽과 유사한 단어들의 group 을 만드는 것으로 묶인 group 에 대한 해석은 사람이 한다는 특징을 가지고 있다. 즉, 한 문서에 특정 토픽들이 어떤 비율만큼 있는지를 확인하여 최종적으로 문서를 분류해주는 것이다.

  복잡한 LDA 를 근사 시킨 방법이 Gibbs sampling 방법이다. 이는 A, B, C 중에서 B 와 C 를 안다면 A 를 시뮬레이션 해보는 방법으로 설명해볼 수 있다. 구체적인 과정은 한 단어를 고르고, 해당 단어가 포함된 하나의 문서는 어떤 토픽을 어떤 분포로 갖는지 확인한다. 그리고 그 단어들은 모두 각각 어떤 토픽에 해당되는지를 찾고, 앞서 골랐던 단어가 어떤 토픽에 해당되는지를 정한다. 이 두 과정을 통해서 각 단어를 랜덤하게 하나의 토픽을 지정해주는 것이고, 모든 단어에 대해 이 과정을 반복하면서 각 문서 당 단어의 토픽들을 결정하여 분포를 알 수 있게 된다.

Clustering

- 지도학습과 비지도학습
  지도학습은 정답 (Label)이 있는 상태에서의 학습을 의미해서 크게는 연속적인 값을 예측하는 회귀와 범주형 변수를 예측하는 분류 문제가 있다. 비지도학습의 경우에는 결과값이 정해지지 않은 Unlabel 데이터에 대해서 학습하는 모델을 의미하는데, 보통 차원 축소나, 군집 분석이 비지도학습에 해당한다고 할 수 있다. 일반적으로 비지도 학습에서의 군집은 고객들의 특성에 따라 Customer segment 를 나누거나, 뉴스들의 텍스트를 바탕으로 다양한 뉴스들을 군집화 하는 것, 유전자 분석 과정에서 시퀀스가 서로 유사한지 확인하는 것, SNS 에서 친분 있는 관계를 묶는 과정들에서 사용된다고 할 수 있다.

- K-means 군집
  K-means 는 거리 기반 알고리즘으로, 가까운 거리에 있는 데이터를 군집으로 묶는데, 이때 군집의 개수를 사용자가 k 개로 입력하여 결과를 만들어낸다. 따라서 최적의 k 를 결정하는 것도 매우 중요하다고 할 수 있다. 동작하는 과정은 다음과 같다.

  1. 군집의 중심점을 사용자에게 입력 받은 k 개로 결정한다.
  2. 그 중심점으로부터 모든 데이터들 사이에 거리를 계산한다.
  3. 가장 거리가 가까운 군집에 할당한다.
  4. 각 군집의 데이터에 대해 평균을 계산하여 중심점을 옮겨주도록 한다.
  5. 해당 과정을 계속 반복하고, 중심점이 움직이지 않거나, 데이터가 변화하지 않으면 중단한다.

- 군집 분석 평가 (WSS 지표)
  비지도학습은 사람의 주관이 들어가는 한계점이 있는데, 나름의 평가 방식이 존재한다. 먼저는 WSS (Total Within Sum of Squares)로 군집 안에 데이터들이 얼마나 중심점에 잘 모여 있는지를 뜻한다. 따라서 더 가까이 모여 있을수록 좋은 군집이라고 할 수 있다. 계산 방법은 중심점의 거리 제곱의 평균을 계산하여 이를 모두 더한 값이고, 낮을수록 좋다. WSS 의 장점은 두 알고리즘 중에 어떤 것이 더 잘 되었는지 확인할 수 있다. 하지만 같은 알고리즘을 썼다고 하더라도, k 가 클수록 WSS 는 필연적으로 감소하게 된다는 한계점이 있다.

- Calinski-Harabasz index (CH-index)
  따라서 이를 보완하기 위해 CH-index 가 등장했다. TSS (Total Sum of Squares)는 전체 데이터의 중심점으로부터 각 점들 사이의 거리를 뜻하는 것으로 TSS 에서 WSS 를 빼면 BSS 를 계산할 수 있다. TSS 는 전체 데이터에 대한 variance, WSS 는 군집 내의 variance, BSS 는 각 군집 사이의 거리라고 할 수 있다. 따라서 좋은 군집의 조건은 다음과 같다.
  1. WSS 가 작을수록 각 군집이 서로 잘 모여있기 때문에 좋다.
  2. BSS 가 클수록 간 군집이 멀리 떨어져 있어서 명확하게 구분되기 때문에 좋다.
  3. K 가 커질수록 WSS 는 감소, BSS 는 증가. K 의 값에 따라 WSS 와 BSS 가 달라진다.
  4. CH-index 가 클수록 좋다.

- Hierarchical Clustering

  최종적인 결과물로 Hierarchical 트리로 nested clusters 를 만들 수 있다. 이에 대한 장점은 Dendogram 에서 적절한 포인트로 자르면 되기 때문에 클러스터의 개수에 대한 가정이 필요 없다. 또한 Hierarchical clustering 은 생물학 등에서 사용되는 것처럼 의미 있는 분류에 찾을 수 있다. 이에 대한 해석은 3 가지로 나뉜다.

  1. Single Linkage (Minimum Distance): 군집이 커질수록 가장 가까이에 있는 두 점의 거리가 이 가깝게 되기 때문에 클러스터가 클수록 좋다. (군집이 클수록 좋다.)
  2. Complete Linkage (Maximum Distance): 군집이 커질수록 가장 멀리 있는 두 점의 거리가 멀어져서 불리하다고 할 수 있다. (군집이 작을수록 좋다.)
  3. Average Linkage (Average Distance): 군집 내 각 점들의 거리의 평균을 계산하여 계산량이 많은 것이 특징이다.

- DBSCAN 알고리즘

  DBSCAN 알고리즘은 밀도 기반의 알고리즘으로 특정 반경 안에 몇 개의 데이터가 존재하는지에 따라 군집으로 포함시킬지 여부를 결정하는 알고리즘이다. 여기서 특정 반경은 Epsilon 으로, (Core point 안에) 데이터 포인트의 개수는 MinPts 파라미터로 결정된다. 이 알고리즘의 군집에서는 Core point 와 Border point 로 나뉘는데, Core point 는 Epsilon 범위 내에 있으면서 그 안에 MinPts 개수를 만족하는 점을 의미하고, Border point 는 Epsilon 범위 내에 있지만 MinPts 개수를 만족하지 못하면서 Core point 의 이웃에 해당하는 점이다. 한편, Noise point 는 Core point 도, Border point 도 아닌 점을 의미한다. 동작 과정은 다음과 같다고 할 수 있다.

  1. Epsilon 과 MinPts 를 설정한다.
  2. 현재 군집의 개수가 0 이고, 모든 점을 p 라고 할 때, 만약 p 가 core point 가 아니면 null label 을 할당한다.
  3. 만약에 p 가 core point 라면, 새로운 클러스터를 만들어주고, Epsilon 범위 내에 해당하는 모든 점들을 해당 클러스터로 할당해준다.
  4. 이 과정들을 모든 점들에 대해서 반복해주도록 한다.

- K-means vs DBSCAN

  K-means: 거리 기반 군집 알고리즘, k 의 개수를 Input, Noise 에 민감, 군집의 개수를 따로 정의해줘야 함,

  DBSCAN: 밀도 기반 군집 알고리즘, Epsilon 과 MinPts 를 Input, Noise 처리가 가능, 군집 개수를 정의하지 않아도 됨, 결과가 비선형으로도 나올 수 있음, Epsilon 하이퍼 파라미터를 결정하는데 어려움이 있음

Recommender System

- Recommender System 의 장점

  Conversion rate 를 증가시킬 수 있다. 연관된 상품을 추천해주는 Cross-selling 이 가능하다, 고객의 Loyalty 를 증가시킬 수 있다.

- Recommender System 의 두 가지 종류

  1. Content-based filtering: Product attributes 를 통해서 고객의 선호 파악

     : attribute 나 features 등으로 유사한 부분들을 찾아서 추천해준다. 하지만 의미 있는 Feature 라는 것이 무엇인지 주관적이고, Content 라는게 특정 선택에 있어어 유일한 답이 아닐 수도 있으며, 새로운 종류를 추천해주기는 조금 어렵다.

  2. Collaborative filtering: 비슷한 취향을 갖는 다른 사람들의 정보를 통해서 선호도 파악

     : 다른 많은 사람들의 선호도나 정보들을 수집해서 고객의 선호도를 예측하는 방법. 단, 고객들의 선호도는 변하지 않는다는 가정을 가지고 있다. 일반적으로 평점이나, 등급 등이 활용된다.

- Collaborative Filtering

  1. User-based CF (UBCF) 모델

     : 유사한 사용자의 선호를 바탕으로 추천해주는 것. 모든 사용자들과 유사도를 뜻하는 거리를 매번 계산해야 하므로 실시간 추천에 한계가 있을 수도 있음.

  2. Item-based CF (IBCF) 모델

     : User-item 매트릭스의 item 사이에 관계를 바탕으로 추천을 해줌. Model 을 위한 matrix 가 상대적으로 작아서 UBCF 보다 조금 더 경제적임.

  3. Model-based CF 모델 -> Cluster users, Association rules

- Cold Start 문제

  새로운 유저가 등장해서 충분한 데이터가 없거나, 아직 아무도 rating 을 매기지 않은 신작의 경우에는 추천 시스템을 활용하기 힘들다. 즉, 추천 시스템이 새로운 또는 어떤 유저들에 대한 충분한 정보가 수집된 상태가 아니라서, 해당 유저들에게 적절한 서비스를 제공해주지 못하는 문제를 뜻하는 것이다.

Tree Based Model – Decision Tree, Ensemble (Bagging, Boosting, Random Forest, AdaBoost, XGBoost)
https://tyami.github.io/machine%20learning/decision-tree-1-concept/

- 결정트리 (Decision Tree)
  : 결정트리 알고리즘은 일련의 필터(Partition, Splitting) 과정을 통해서 모델이 구축된다. 다시 말해서, 주어진 변수를 어떻게 사용하고, 필터링 하는가에 따라 다양한 트리 모델을 만들 수 있는데, 좋은 트리 모델을 만들기 위해서는 '불순도(Impurity)'를 고려하게 된다. 이 불순도라는 것은 '변별력이 좋은 질문'으로 이해할 수 있는데, 이는 해당 범주 안에 서로 다른 데이터가 얼마나 섞여 있는지를 뜻하는 것이다. 따라서 최고의 성능을 보이는 트리 모델은 불순도가 낮은 지표를 찾아서 나무를 구성하면 되는 것이다. 일반적으로 불순도는 Entropy와 Gini index를 사용하는데, Entropy 기반의 알고리즘은 ID3, C4.5, C5.0이고, Gini index 기반 알고리즘은 CART이다. 트리 모델은 어떤 의사결정을 의미하는지 직관적으로 이해가 가능하고, 어떤 변수가 더 predictable 한지 쉽게 이해할 수 있다는 장점과 함께 Overfitting이 발생할 가능성이 높다는 한계가 있다. 따라서 이 한계점을 극복하기 위해 가지치기(Pruning)를 수행하는데, 이 종류는 사전 가지치기와 사후 가지치기가 있다. 이는 나무의 최대 깊이(Depth)를 제한하는 것, 자식 노드의 최소 샘플 수를 설정하는 것, 불순도의 최솟값을 설정하는 것, 그리고 불순도 변화값의 최솟값을 설정하는 등의 방법을 활용한다.

- 결정트리의 ID3 알고리즘
  : ID3는 Iterative Dichotomiser의 약자로, 'Dichotomiser' 단어는 프랑스어로 이분하다라는 뜻을 가지는데, 이는 반복적으로 이분하는 알고리즘이라고 할 수 있다. 일반적으로 트리 모델은 불순도 값이 작은 방향으로 학습이 되는데, ID3 알고리즘은 Entropy를 불순도로 사용한다. Entropy는 다음과 같은 공식을 통해 계산할 수 있고, 범주 안에 다른 데이터가 많이 섞여 있는 경우에 혼잡도가 크기 때문에 Entropy가 크다고 할 수 있다. 즉, 특정 사건에 대한 확신도가 강한 경우에는 Entropy가 낮고, 확신도가 약한 경우에는 Entropy가 크다.

  Entropy = Information Gain =

  ID3 모델에서는 Entropy 값의 변화량을 나타내기 위해 Information Gain 개념을 사용한다. 즉, IG는 분기 전후로 세 종류의 엔트로피를 계산할 수 있는데, 위와 같은 수식을 통해 분기 전후 엔트로피의 차이값을 계산한다. 이 모델은 분기 후에 Entropy가 작아지는 지표를 선택해야 하기 때문에, IG에서 H(S)는 동일하기 때문에 IG가 가장 큰 지표를 선택해야 한다. 최종적으로 최적의 ID3 알고리즘은 Entropy를 작게 하는 변수와 Information Gain이 큰 변수를 통해 Partition하고, Splitting 하면 된다.

- 결정트리의 CART 알고리즘
  : CART 알고리즘은 ID3와 비슷한 시기에 개발된 알고리즘으로 Classification And Regression Tree의 약자이다. 이름 그대로 분류와 회귀 모두 가능한 알고리즘이라고 할 수 있고, 앞서 이야기했듯 CART 모델에서의 불순도는 Gini index이다. Entropy와 유사한 불순도 개념이므로, 다른 데이터가 많이 섞여 있으면 Gini index가 높고, 섞여 있지 않으면 Gini index가 낮다. CART 알고리즘은 가지를 분기할 때, 여러 개의 자식 노드가 아닌 단 두개의 노드로

분리하는 특징을 가져서, Information Gain을 계산할 때도 모든 클래스의 개수만큼 비교를 해야 한다. 또한 CART 모델은 회귀 모델에도 활용이 가능한데, 이때는 불순도 index를 사용하지 않고, 실제값과 예측값의 오차를 사용하여 분기해서 모델을 구축한다.

- Bias와 Variance
  : 일반적으로 비선형 관계에 있는 데이터를 예측하고자 할 때, 선형 회귀 모델을 사용하면 비선형의 관계를 고려하지 못하기 때문에 어쩔 수 없이 (모델의 한계 때문에) 발생하는 cost가 존재한다. 즉, 비선형의 특징을 가지는 True relationship을 고려하지 못하기 때문에 Bias가 높다고 이야기할 수 있다. 이러한 선형 모델의 한계를 극복하기 위하여 비선형 모델을 활용하는데, 이는 선형 모델보다 더 융통성 있는 모델이라고 할 수 있다. 선형적이지 않은 특징을 고려할 수 있기 때문에 비선형 모델은 Bias가 낮다. 하지만 예측값들이 비선형의 모양을 갖기 때문에 예측값들의 편차가 커서 Variance가 크다고 할 수 있다. 일반적으로 Bias와 Variance는 trade-off 관계로 하나의 값이 커지면, 다른 하나는 줄어드는 관계를 가지고 있다. 따라서 모델의 단순함과 복잡함 사이에서 균형 있게 모델을 구축해야 할 필요가 있고, 가장 많이 활용되는 방식은 Regularization이나 Boosting, Bagging 등의 방법이다.

  - Low Bias & Low Variance: 예측값이 정답 근처에 분포되어 있고, 예측값이 몰려 있음.
  - Low Bias & High Variance: 예측값이 정답 근처에 분포되어 있고, 예측값은 흩어져 있음.
  - High Bias & Low Variacne: 예측값이 정답에서 떨어져 있고, 예측값은 몰려 있음.
  - High Bias & High Variance: 예측값이 정답에서 떨어져 있고, 예측값은 흩어져 있음.

- Bagging – Sample을 복원 추출 방식으로 동시에 여러 개를 만들어서 모델링
  : Bagging 방식은 Bootstrap Aggregating의 약자로, 이름처럼 Bootstrap을 이용한다. 이는 주어진 데이터에서 복원 추출 방식으로 Random Sampling 해서 새로운 데이터를 만들어내는 것을 의미하는데, 이렇게 만들어진 여러 데이터를 통해 weak learner를 훈련하여 결과를 voting 한다. 즉, Bagging 알고리즘이 작동하는 방식은 original 데이터에서 복원 추출 방식으로 random sampling 하여 여러 데이터셋을 만들고, 해당 데이터를 통해 모델을 구축하여 combine 해서 최종적인 1개 모델을 도출하는 것이다. 분류의 문제에서는 다수결의 투표 방식을, 회귀의 문제에서는 평균으로 최종 결과를 도출한다. Bagging은 다양하게 sampling 된 데이터인 Bootstrap을 통해 다양한 모델을 만들기 때문에 모델의 복잡도를 줄여서 variance를 감소시킬 수 있다. 또한 다양하게 만들어진 모델들을 combine 시키기 때문에 bias를 감소시키는 효과가 있다. 결국 Bagging 방식은 Bootstrap을 aggregating 하여 Variance를 감소시키는 모델이라고 할 수 있다. Bagging을 활용한 알고리즘은 Random Forest 등이 있다.

  ➔ 작동 원리는 다음과 같다. 데이터를 복원 추출 방식으로 Bootstrap sample을 만들어서 서로 다른 샘플 데이터를 여러 개 만든 후에, 각각의 샘플에 대해서 다양한 방식으로 모델링을 수행하도록 한다. 그리고 나서는 이 모델들을 모두 합쳐서 최종 모델을 구축하는 방식이다.

- Boosting – 기존 모델에서 틀린 것들을 중점적으로 순차적으로 다시 계속 학습
  : Boosting 방식은 반복적으로 모델을 업데이트 하는 모델로, 이전 iteration 결과에 따라 테이터셋 샘플에 대해 가중치를 부여한다. 결과적으로 반복할 때마다 각 샘플의 중요도에 따라 다른 모델이 만들어지게 되고, 최종적으로는 모든 iteration에서 생성된 모든 결과를 voting하여 모델을 구축한다. 간단한 예로 들면, 식당에서 한번 테스트를 진행하고서 맛 없다고 이야기 했던 손님들만 불러서 다시 평가를 받는 형태로, 틀렸던 것들을 중점적으로 다시 학습하는 모델이라고 이야기할 수 있다. 즉, Boosting 모델은 약한 학습자에게 적합하게 반복하여 앙상블 모델에 반영하도록 해서 다음 모델을 구축할 때, 현재의 모델에서 강점과 약점을 더 잘 고려하여 계속해서 업데이트 해나가는 모델이라고 할 수 있다. 따라서 약한 학습기를 여러 개 연결하여 강한 학습기를 만드는 앙상블 방법이다. Boosting을 활용한 알고리즘은 AdaBoost, GBM, XGBoost, Light GBM 등이 있다.

- Bagging과 Boosting – 각각의 장단점
  : Bagging 알고리즘은 병렬 앙상블 모델로, 복원 추출의 Random sampling 하여 각 모델은 구축하고, 이는 서로 독립적인 특징이 있다. 하지만 Boosting 알고리즘은 연속적인 앙상블 모델로, 이전 모델의 오류를 고려하여 가중치를 부여해서 계속 연속적으로 학습한다는 특징이 있다. 또한 Bagging 알고리즘은 Variance를 감소시키는 것을 목적으로 하며, Boosting 모델은 Bias를 감소시킨다. 각 모델이 적합한 상황은 Bagging은 High variance, low bias의 복잡한 모델에서, Boosting은 Low variance, high bias 모델에서 적합하다. 대표 알고리즘은 Bagging에서는 Random Forest, Boosting에서는 Gradient Boosting, AdaBoost 등이 있다.

- Random Forest – Tree correlation, Variable Importance,
  : Bagging 방식을 통한 트리 기반의 앙상블 모델은 Random Forest가 있다. 이는 많은 Decision Tree가 있고, 이 Decision Tree들은 서로 다른 가지의 개수와 모양, 형태 등을 가지고 있다고 할 수 있다. Bagging 방식이기 때문에 처음에 복원 추출의 Bootstrap sampling을 하여 각 데이터셋 별로 Decision Tree 모델을 만들고, 이를 앙상블 하여 최종적인 모델을 도출한다. Bagging 방식은 다양한 데이터로부터 만든 모델을 combine 하여 Variance를 낮추는 특징을 갖는데, 만약에 예측력이 높은 특정 변수가 모든 모델에 포함된다면 모든 모델들이 유사한 관계를 가지면서 획일화된 모델이 될 수도 있다. 이를 Tree correlation 이라고 하는데, 이러한 한계를 극복하기 위해 Random Forest는 변수를 Random 하게 선택하여 tree 모델을 만든다. 다시 말해서, Random Forest는 각 노드를 만들 때 모든 attribute를 사용하는 것이 아닌, 그 중에 일부만을 사용하면서 빠르게 모델을 만들고, 동시에 각 tree 사이에 correlation을 낮춘다고 할 수 있다. Random Forest의 장점으로는 분류와 회귀 모두에 사용가능하며, 결측치를 다루기 쉽고, 대용량 데이터 처리에 효과적이며, 모델의 과적합을 피해서 정확도를 향상시키고, 상대적으로 중요한 변수를 선정하고, Ranking을 할 수 있다는 특징이 있다.

  ➔ Random Forest가 예측력이 높은 특정 변수가 포함되어 각 트리 모델이 모두 유사하게 만들어져서 획일화 되는 Tree correlation 문제를 극복한 2가지 방식은 다음과 같다. 첫째, Bagging 방식으로 각 tree는 복원 추출로 Random Sampling 된 Bootstrap 데이터를 활용했다. 둘째, Random Vector 방식으로 각 노드를 만들 때 모든 attribute를 사용하지

않고, 일부만 사용하여 모델을 빠르게 만들면서 각 tree 사이에 correlation을 낮춘다.

- Random Forest의 Feature Importance

  : Bagging 방식의 Random Forest는 복원 추출의 Random sampling 하는 Bootstrap 데이터를 활용한다. 그런데 이때 각 iteration을 돌면서, Bootstrap 데이터에 포함되지 않는 데이터가 있다면, 이를 모아서 Out-of-bag 데이터로 만들어준다. 그리고 이 OOB 데이터를 마치 test 나 valid 데이터처럼 활용하며 모델의 예측력을 계산하도록 한다. 또한 모델에 사용된 특정 변수를 섞은 후 (Permutation) 기존 모델이 가지고 있던 정확도와 얼마나 차이가 나는지 확인하여 모델에 있어서 특정 변수가 가지고 있는 중요도를 확인하도록 한다. 이를 통해 Random Forest는 특정 변수가 모델에 있어서 얼마나 중요한지 'Feature Importance(변수 중요도)'를 확인할 수 있다. 또한 파이썬의 사이킷런에서는 어떤 특성을 사용한 노드가 평균적으로 불순도를 얼마나 감소시키는지 확인하여 중요도를 측정하기도 한다.

- AdaBoost

  : Boosting 방식의 AdaBoost는 Adaptive boosting의 줄임말로 트리를 만들지 않고, 하나의 Stump만 만들기 때문에 단일 변수 모델과 같은 단순한 트리라고 할 수 있다. Random Forest의 경우에는 최종 모델을 구축하는데 있어서 각각의 트리들이 모두 동일하게 중요도를 갖는데, AdaBoost는 몇몇의 Stump가 다른 Stump 보다 더 중요할 수도 있기 때문에 가중치를 다르게 부여한다. 또한 모든 트리가 독립적이었던 Random Forest와 다르게, AdaBoost는 모든 트리들이 sequential 하다는 특징이 있다. AdaBoost가 학습하는 방식은 이전의 모델을 보완하는 새로운 모델을 만드는데, 이때 이전 모델이 과소적합 했던 훈련 샘플의 가중치를 더 높여서 새롭게 만들어지는 모델은 학습하기 어려웠던 샘플에 더 맞춰서 학습하도록 한다.

- XGBoost

  : XGBoost는 Extreme Gradient Boosting의 약자로, 일반적으로 Gradient Boosting은 느리고, 과적합의 이슈가 있다는 한계가 있었는데, 이를 보완하기 위해 XGBoost 알고리즘이 등장했다. GBM 보다 빠르고, 과적합을 방지할 수 있는 규제가 포함되어 있으며, CART 모델을 기반으로 하여 분류와 회귀 모두가 가능하다. 또한 Early stopping을 제공하고 있다.

Linear Regression

- (단변량) Linear Regression의 Hypothesis, Cost function, Gradient Descent
  : Linear Regression의 학습을 위해서는 Hypothesis와 Cost function, 그리고 Gradient Descent가 정의가 되어야 한다. 그리고 각각을 다음과 같이 정의할 수 있다.

  Hypothesis = Cost Function =

  Linear Regression의 Hypothesis에 따라, 해당 데이터를 가장 잘 설명할 수 있는 직선을 그리도록 한다. 이 과정에서 이 직선을 설명할 수 있는 theta 0과 theta 1을 찾아야 하는데, 이때 필요한 것이 Cost Function이다. 예측 값과 실제 값의 차이에 대한 제곱 합을 통해 Cost 값이 가장 최소가 될 수 있도록 theta 값을 계속해서 변화시켜 가면서 최적의 theta를 찾아주도록 한다. 즉, Hypothesis에 따라 y 값에 대해 선형식으로 표현하는데, 이를 위해 최적의 theta를 찾기 위해 Cost Function을 사용하면서 이 Cost 값을 최소로 만드는 theta를 찾아주도록 하는 것이다. 이때 활용되는 것은 Gradient Descent 알고리즘인데, Random한 theta 값에서부터 시작해서 Cost 함수가 작아지도록 계속 theta 값을 수정하면서 더 이상 값이 감소하지 않을 때까지 이 과정을 반복한다. 이 Gradient Descent 알고리즘은 Linear, Logistic Regression 뿐 아니라, Deep learning에서도 사용된다. 이는 최솟값의 위치를 찾기 위해 비용 함수의 Gradient 반대 방향으로 정의한 step size를 가지고 조금씩 움직이면서 최적의 파라미터를 찾는 방법이다. 즉, Gradient는 파라미터에 대해 편미분한 벡터이고, 이를 반복적으로 조금씩 움직이는 방식이다.

```
##################### Univariate Linear Regression #####################
#####################   Exercise - MY ALGORITHM    #####################
costJ <- function(df, feature_col, actual_col, t0, t1){
  1/(2*nrow(df)) * sum(((t0 + t1*df[, feature_col]) - df[, actual_col]) ** 2)
}

Linear_regression <- function(df, feature_col, actual_col, t0, t1, alpha, iter_num){
  output_df <- data.frame(iter = 0, t0 = t0, t1 = t1, cost = costJ(df, feature_col, actual_col, t0, t1))

  for (i in 1:iter_num){
    derivative0 <- alpha * mean((t0 + t1*df[, feature_col]) - df[, actual_col])
    derivative1 <- alpha * mean(((t0 + t1*df[, feature_col]) - df[, actual_col]) * df[, feature_col])
    t0 <- t0 - derivative0
    t1 <- t1 - derivative1

    output_df <- rbind(output_df, c(i, t0, t1, costJ(df, feature_col, actual_col, t0, t1)))

    if (output_df$cost[nrow(output_df) - 1] - output_df$cost[nrow(output_df)] <= 10e-4){
      break
    }
  }
  return(output_df)
}

costDF <- Linear_regression(exData, 'pop', 'profit', 0, 1, 0.01, 1500)
costDF %>% ggplot(aes(x = iter, y = cost)) + geom_point(alpha = 0.5)

costDF <- Linear_regression(exData, 'pop', 'profit', 0, 1, 0.001, 1500)
costDF %>% ggplot(aes(x = iter, y = cost)) + geom_point(alpha = 0.5)

costDF <- Linear_regression(exData, 'pop', 'profit', 0, 1, 0.0001, 1500)
costDF %>% ggplot(aes(x = iter, y = cost)) + geom_point(alpha = 0.5)

costDF <- Linear_regression(exData, 'pop', 'profit', 0, 1, 0.00001, 1500)
costDF %>% ggplot(aes(x = iter, y = cost)) + geom_point(alpha = 0.5)
```

- (다변량) Linear Regression의 Hypothesis, Cost function, Gradient Descent
  : 단변량 Linear Regression의 Hypothesis와 Cost function, Gradient Descent와 거의 유사하고, 다만 변수의 개수만큼 theta의 개수가 추가된다. 전반적인 작업 과정은 위와 동일하다.

  Hypothesis = Cost Function =

```r
regression <- function(feature_df, label, n, num_iter, alpha){
  # (1) Hypothesis 설정
  h <- function(x, theta_vector){ # x is feature vector
    theta_vector %*% x            # inner product of two vectors
  }

  # (2) Cost 함수 설정
  cost_J <- function(theta_vector){
    v <- as.matrix(feature_df) %*% theta_vector - label
    (t(v) %*% v / (2*nrow(feature_df)))[1, 1]
    # 1 / 2 * mean(v ** 2)
  }

  # (3) feature_df의 열의 개수만큼 theta_vector를 정의해주고, Cost 함수 결과로 output_df 틀 만들기
  theta_vector <- rep(0, ncol(feature_df))

  theta_df <- as.data.frame(t(as.data.frame(theta_vector)))
  rownames(theta_df) <- 1
  colnames(theta_df) <- str_replace(colnames(theta_df), 'V', 't')

  output_df <- data.frame(iter = 0, cost = cost_J(theta_vector), alpha = alpha)
  output_df <- cbind(output_df, theta_df)

  # (4) iteration 횟수만큼 학습하면서 output_df에 결과 누적시키기
  for (i in 1:num_iter){
    v <- feature_df %*% theta_vector - label
    theta_update <- alpha / n * t(feature_df) %*% v
    theta_vector <- theta_vector - theta_update
    output_df <- rbind(output_df, c(i, cost_J(theta_vector), alpha, theta_vector))

  # (5) Cost 함수의 감소 폭이 10e-4 보다 작으면 iteration 횟수와 상관 없이 학습 중단
  if (output_df$cost[nrow(output_df) - 1] - output_df$cost[nrow(output_df)] <= 10e-4){break}
  }

  return (output_df)
}
```

Logistic Regression

- Logistic Linear Regression의 Hypothesis, Cost function, Gradient Descent

  Hypothesis = Cost Function =

```r
Logistic_Regression <- function(feature_df, label, num_iter, alpha){
  # (1) Hypothesis와 Cost 함수 정의
  sigmoid <- function(x){1 / (1 + exp(-x))}
  h <- function(x, theta_vector){sigmoid(theta_vector %*% x)}

  costFunction <- function(theta_vector){
    hx <- apply(feature_df, 1, function(x){h(x, theta_vector)})
    costV <- label * log(hx) + (1 - label) * log(1 - hx)
    -mean(costV, na.rm = T)
  }

  # (2) feature_df의 열 개수만큼 theta_vector를 정의해주고, Cost 함수 결과로 output_df 만들기
  theta_vector <- rep(0, ncol(feature_df))

  theta_df <- as.data.frame(t(as.data.frame(theta_vector)))
  rownames(theta_df) <- 1
  colnames(theta_df) <- str_replace(colnames(theta_df), 'V', 't')

  output_df <- data.frame(iter = 0, cost = costFunction(theta_vector), alpha = alpha, theta_df)

  # (3) Iteration 횟수만큼 학습하면서 output_df에 결과 누적시키기
  m <- nrow(feature_df)
  for (i in 1:num_iter){
    theta_update <- (t(as.matrix(feature_df)) %*%
                       (apply(feature_df, 1, function(x){h(x, theta_vector)}) - label)) / m * alpha
    theta_vector <- theta_vector - theta_update[, 1]
    output_df <- rbind(output_df, c(i, costFunction(theta_vector), alpha, theta_vector))

    # (4) Cost 함수의 감소 폭이 10e-4 보다 작으면 iteration 횟수와 상관 없이 학습 중단
    #if (output_df$cost[nrow(output_df) - 1] - output_df$cost[nrow(output_df)] <= 10e-4){break}
  }
  return (output_df)
}
```

Multi-class Classification, Regularization

- Logistic model을 활용하여 Multi-class를 분류할 수 있는 방법
  : 총 10개의 class가 있다면, 총 10개의 Logistic model을 만들어서 각 Label 인지 아닌지를 분류할 수 있는 모델들을 만들고, 최종적으로 가장 확률값이 높은 Label을 매기도록 한다.

- Regularization을 하는 이유는 무엇이고, 이를 하면 무엇이 달라지는지?

: 일반적으로 모델을 학습하는 과정에 있어서 변수의 개수가 너무 많이 활용되어 모델이 구축되면, 차원의 수가 그만큼 크기 때문에 모델이 복잡해지는 경우가 있다. 다변량 회귀에서의 Hypothesis를 보면, n의 값이 1000이라고 할 때, 이 모델은 1000차원을 갖
;는 모델이 된다. 이렇게 되면, 학습 데이터에서는 성능이 좋을지 몰라도, 학습의 궁극적인 목표인 모델의 일반화는 과적합 등으로 인해 불가능하다는 한계가 있다. 따라서 Regularization은 이렇게 복잡한 모델을 조금 더 단순화 시켜줄 수 있도록 규제항을 추가해주는 것이라고 할 수 있다. 규제항에 대하여 theta의 값이 커질수록 전체 값이 커지고, theta가 0에 가까워질수록 값이 작아진다고 할 수 있다.

이것을 하면 모델은 어떻게 달라지는가?
규제항에서의 Lambda 값의 의미는 무엇이고, 어떻게 결정되는가?

Neural Network Model

- Neural Network Model의 Hypothesis

- 역전파 알고리즘
  순전파가 Input layer에서 Output layer로 향한 다면, 역전파는 Output layer에서 Input layer 방향으로 계산하면서 가중치를 업데이트 하는 방식이다. 이는 Output layer에서 시작해서 직전 layer로 이동하면서 cost를 구하는 최적화 알고리즘이다. 결국 뉴럴 네트워크 모델에서 중요한 것은 가중치를 올바르게 구했는지에 대한 부분인데, 적은 횟수의 순전파 학습으로는 출력값이 정확하지 않는다는 한계가 있다. 또한 순전파 방식을 통해 적절한 가중치를 찾는 과정에서 연산량이 매우 많아져서 비효율적이라는 문제가 있다. 따라서 역전파 방식을 통해서 Input layer 방향으로 오차를 다시 보내며 가중치를 업데이트 하는 방식이라고 할 수 있다. 이 과정에서는 정답과 출력값의 차이인 오차를 delta라고 하고, 모든 입력값으로부터 얻어지는 출력과 정답 간의 오차 제곱 합을 최소화하도록 가중치를 조정한다. 따라서 Learning from mistakes, Perceptron learning rule이라고도 한다.

- Random Initialize
  인공지능의 학습 과정에서 가중치 설정은 매우 중요한데, 이에 따라 기울기 소실 문제가 발생하는 등의 여러 문제가 생길 수 있다. 또한 과대 및 과소 적합이 발생할 수도 있다. 만약 초기 값을 모두 0으로 설정하게 되면, 모든 뉴런들이 같은 값을 나타내어 역전파 과정에서 각 가중치의 값들이 모두 동일하게 업데이트 된다. 이는 매번 학습이 진행되면서 계속 발생하게 될 것이고, 최종적인 모델의 성능이 개선되지 않을 것이다.

- Gradient Checking
  Gradient checking은 순전파, 역전파 이후에 역전파로 구해진 값이 정확한지 점검하는 과정이라고 할 수 있다. 이 과정은 역전파 알고리즘을 직접 수동으로 검증하는 방법이다. 즉, 기울기에 대한 근사치를 구해서 비교하여 검증하는 방법이다. Theta 값에 대해 아주 작은 값인 e를 빼고, 더해서 두 점 사이에 기울기를 계산한다. 이 과정은 속도가 굉장히 느려서 훈련 할 때는 사용하지 않고, 디버깅 과정에서 사용한다. 이를 하는 이유는 역전파의 과정이 상당히 민감하기 때문에 작은 버그에도 최종적인 값이 크게 차이가 날 수 있다. 실제로 Cost는 줄어드는 것처럼 보여도, 실제로 충분히 감소가 되지 않는 등의 이슈가 발생할 수도 있다. 따라서 이를 확인하는 방법이다.

- Gradient Vanishing

  뉴럴 네트워크를 학습하다보면, 역전파 과정에서 기울기가 점차적으로 작아지는 현상이 발생 할 수도 있는데, Input layer와 가까운 층들에서 가중치가 제대로 업데이트 되지 않으면 최적 모델을 찾을 수가 없다. 이를 기울기 소실 (Gradient Vanishing) 이라고 한다. 반대로 기울기가 점차 커지더니, 가중치가 비정상적으로 큰 값이 되면서 발산하는 경우도 있는데 이는 기울기 폭주 (Gradient Exploding) 이라고 한다. RNN에서 자주 발생한다.

  Sigmoid를 사용하면, 입력의 절댓값이 클 경우에 함수의 출력값이 0이나 1에 수렴하면서 기울기가 0에 가까워진다. 그래서 역전파 과정에서 기울기가 점차 사라져서 Input layer로 갈수록 역전파가 되지 않는 기울기 소실 문제가 발생할 수도 있다. 이를 해결하기 위해 은닉층의 활성화 함수로 Sigmoid 대신에 ReLU를 사용하는 방법이 있다. 또한 배치 정규화를 통해 신경망의 각 층에 들어가는 입력을 정규화 하여 학습을 효율적으로 하는 방법도 기울기 소실 문제를 해결할 수 있는 방법이다. 마지막으로, 세이비어 초기화나 HE 초기화를 통해 가중치를 초기화 시키는 방법을 통해 기울기 소실을 막을 수도 있다.