Structuring Machine Learning Projects

[Metric]

단일 실수 평가 지표가 있다면 진행 속도가 빨라진다.

1) 정밀도(정확도) : 올바르게 판별하는 확률 (T라고 판별한 것 중 실제 T인 것의 비율)
2) 재현율 : 정답(실제 T) 중 몇퍼센트가 분류기를 통해 올바르게 인식(T라고)이 되었는가
3) F1-score : 1) + 2) 의 조화평균

N개의 metrics 중, Optimizing Metric 1개 + Satisficing Metric (N-1)개
ex) 정확도 – 최적화 지표 / 러닝 타임 – 만족 지표

[dev / test sets]

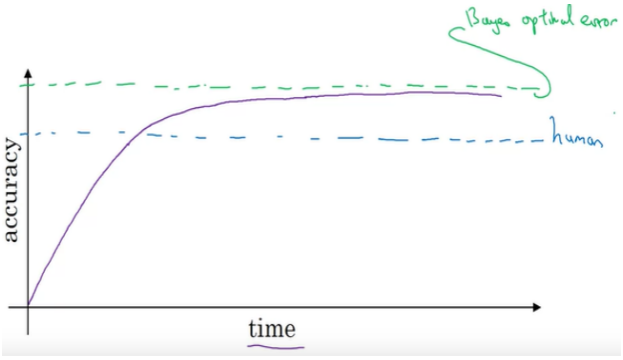
dev set과 test set이 같은 분포에서 만들어 질 것을 권장(개발셋은 어디를 타겟으로 잡아야 할지 알려주며, 이것이 테스스셋에서도 일반화가 되게 해줘야 함)
테스트 셋에 이미지 추가하면 예상되는 데이터 분포를 반영하지 못한다.
설정할 때 미래에 어떤 데이터를 반영시킬 지와 어떤 것을 중요시 여길지를 고려해 적합한 세트를 고르는 것이 중요하다. 또한 같은 분포, 유형의 데이터를 확보해야한다.

좋지 않은 ex) 개발셋 : 중산층의 집코드(우편번호) / 테스트셋 : 저소득층 우편번호

[오류 지표가 만족스럽지 못할 때] – 유지보다는 새로운 지표 사용

오류율이 낮지만, 정확하지 않을 때 개발/테스트셋의 metrics를 변경하려면 error의 가중치를 부여하는 방법이 있다. 정답과 거리가 멀 때 더 큰 가중치를 부여해 오류 값이 높아지도록 한다.

[베이즈 최적 오류]



알고리즘을 계속 훈련시키면 더 많은 데이터를 더 큰 모델에 사용할 수 있겠지만, 성능은 이론적인 한계점에 다가갈 뿐 도달X

최상의 오류로 어떤 x, y함수 매핑이 일정 정확도를 넘는 방식

왜 인간 수준 성능을 넘으면 진행 속도가 더뎌질까 ?
- 많은 작업에서 인간 수준 성능이 베이즈 최적 오류 지점과 멀리 떨어져 있지 않기 때문
  인간 수준 성능을 이미 능가한 시점에서는 더 발전할 수 있는 부분이 제한적
- 성능이 인간 수준에 미치지 못할 경우 여러 가지 도구를 이용해 성능을 개발할 수 있다.
  이런 도구는 인간 수준 성능을 넘은 뒤에는 사용하기 어렵다.

머신러닝 알고리즘이 인간 수준 성능에 미치지 못한다면 ?
- 인간으로부터 표기 데이터를 받아올 수 있다. (레이블된 데이터)
- 수동오류 분석
- 편향과 분산에 대한 분석

[편향과 분산]

ex1) 인간오류 1% / train error 8% / dev error 10%
훈련셋에서 알고리즘과 인간의 차이가 매우 크다는 것은 알고리즘이 훈련셋에 잘 피팅되지 않음
-> 편향(과 분산) 줄이기
=> 더 큰 신경망을 훈련하거나 더 나은 최적화 알고리즘을 사용해 오래 훈련하기, 더 나은 신경망 아키텍처나 하이퍼 파라미터 사용하기

ex2) 인간오류 7.5% / train error 8% / dev error 10%
훈련셋에서는 잘하고 있지만, 인간 수준 성능에 약간 못 미치는 수준
-> train error과 dev error 사이의 오류를 줄이는 데 집중 : 학습 알고리즘의 분산 줄이기
=> 정규화를 시도해 개발 오류를 훈련 오류와 비슷한 수준으로 만들기, 더 많은 데이터 얻기

인간 오류를 베이즈 오류의 프록시(추정치)로 생각하면, 베이즈 오류의 근사치와 훈련 오류 간의 편차를 회피 가능 편향이라고 한다. 훈련 오류가 베이즈 오류 값으로 내려갈 때까지 계속 개선해야하는데, 베이즈 오류보다 좋으면 안된다. -> 과적합
* 회피 가능 편향 : 편향이나 일부 최소 수준 오류가 있다는 것을 인정 + 베이즈 오류가 7.5%인 경우 이 이하로 내려갈 수 없다.
* Human-level error <--- Avoidable bias ---> Training error <--- Variance ---> Dev error
(proxy for Bayes error)

** dev set과 test set 오류 차이 크면,
- dev set의 과적합을 줄이기 위한 정규화를 늘리고, dev set의 size를 늘린다.

[효과적인 지도 학습 알고리즘]

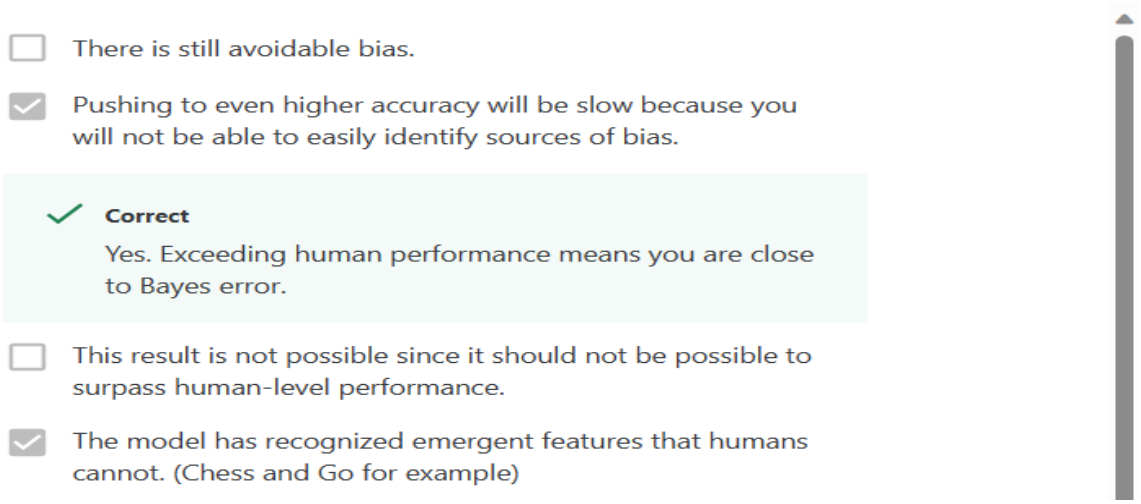
1. 훈련셋을 잘 fitting되어야 하며, 낮은 회피가능 편향을 달성할 수 있는 것과 동일 의미
2. 개발셋이나 테스트셋에서도 잘 fitting 되어야 한다. (작은 분산)

<Quiz>

No. Adding this data to the training set will change the training set distribution. However, it is not a problem to have different training and dev distributions. In contrast, it would be very problematic to have different dev and test set distributions.

훈련데이터 분포와 개발데이터 분포가 달라도 문제가 되지 않지만, 개발데이터 분포와 테스트데이터 분포가 다르면 문제가 된다.

. After working on this project for a year, you finally achieve: Human-level performance, 0.10%, Training set error, 0.05%, Dev set error, 0.05%. Which of the following are likely? (Check all that apply.)



[오류분석] : 개발셋에서 잘못표기된 예시에서 거짓 양성과 거짓 음성 찾음

ex1) 100개의 잘못 표기된 개발 예시를 수동으로 검사

-> 5% (5개)가 실제의 개 이미지 => 100개 중 오직 5개만 정확

-> 오류가 10%에서 9.5%로 내려감 (상대적으로 5% 줄어듬)

-> 효율적인 시간 투자는 아님 => 상한선 설정

ex2) 100개의 잘못 표기된 개발 세트 중 50장이 실제의 개 이미지

=> 50%가 정확

-> 오류가 10%에서 5%로 줄어듬

-> 오류를 반으로 줄이는 것이 노력할 가치가 있다

[훈련셋에서 잘못 라벨링]

입력값 X + 결과값 라벨 Y -> Y의 결과값이 다르다는 것을 확인했을 때 수정할 가치가 있을까 ?

전체 데이터셋과 실제 오류 비율이 너무 크지 않으면 수정하지 않아도 됨

훈련셋 라벨에서 몇 개의 실수가 조금 있음에도 정상적으로 작동하는 머신러닝 알고리즘의 조건
-> 딥러닝 알고리즘이 랜덤한 오류에 강하고, 시스템적 오류는 취약하다.

[개발셋, 테스트셋에서 잘못 라벨링]

잘못 라벨링된 것을 줄일지 여부를 결정할 때 수치 참고(3)
- 전체적인 개발셋 오류 확인
- 오류개수나 잘못될 라벨링으로 인한 오류 비율
- 다른 원인으로 인한 오류 확인 : ex. 컴퓨팅 실수
* 개발 세트의 목적 : 분류기 A와 B 사이에서 어떤 것을 고를지 선택할 수 있는 도움을 주는 것

개발셋의 라벨을 수동으로 검사하고 직접 고치려 할 경우
- 어떤 방식의 프로세스를 적용하던 이를 개발세트와 시험세트에 동시에 적용시키면 좋다.
- 알고리즘이 맞췄던 것과 틀렸던 것의 예시를 잘 살펴보는 것이 좋다. (알고리즘의 판단은 쉽고, 사람이 제대로 맞추지 못한 것들을 수정해야 할 수 있는데, 알고리즘의 오류만 고치면 알고리즘에 더 많은 편향 추정값과 오류가 남는다. -> 틀린 것만 확인)
- 라벨들을 개발/시험셋에서 고칠 경우, 동일 절차를 훈련셋에 적용할 수도, 하지 않을 수도 있다. (훈련셋에서 라벨을 수정하는 것은 덜 중요하다. 또한 개발/시험셋의 라벨이 훈련셋 대비 적은 경우가 많으므로 훨씬 더 큰 훈련셋에서 라벨 수정에 추가적으로 힘들이지 않을 수 있다. 훈련셋이 개발/시험셋과 다른 분포를 갖는 것도 비교적 합리적이다.)

[훈련셋과 테스트셋 분포가 서로 다를 때]

ex) 모바일 앱 데이터(촬영이미지라 희미) 만개 vs 웹에서 다운받은 이미지(전문적, 고해상도) 20만

옵션1) 이미지 21만개를 랜덤으로 섞어, 개발/테스트셋엔 각각 2500개, 훈련셋엔 20만 5천개
장점 : 훈련/개발/테스트셋 모두 같은 분포에서 오기에 관리가 쉬움
단점 : 2500개의 개발셋을 보면 그 중 상당수가 웹페이지 이미지 분포에서 온다. 실제로 중요하게 생각하는 모바일 앱 이미지가 아님.
=> 실제로 원하는 것과 다른 데이터 분포에 맞춰 개발셋을 최적화하는 것이기에 추천하지 않음

옵션2) 훈련셋은 웹에서 온 20만개 (원하면 앱 이미지 5천개 추가), 개발/시험셋은 모두 앱이미지
단점 : 훈련분포와 개발/테스트셋 분포가 다름 -> 장기적으로는 좋음

<편향과 분산>

훈련셋과 개발셋이 동일한 분포일 때 훈련 오류 1%, 개발 오류 10%이면 큰 분산 문제
-> 알고리즘이 훈련셋을 잘 일반화시키지 못한 것, 개발셋에서는 잘 했는데 문제 생김

하지만
훈련셋과 개발셋이 다른 분포일 때(옵션2) 훈련셋은 고해상도이기에 아주 쉽고, 개발셋은 훨씬 어려울 수 있다. -> 분산문제 없을 수 있고, 개발셋에 정확히 분류하기 어려운 이미지들이 더 많이 포함되어 있음을 반영하는 것

여기서 문제, 훈련오류에서 개발오류로 넘어갈 때 한번에 두가지가 바뀜
- 알고리즘이 훈련셋에서는 데이터를 봤지만, 개발셋에서는 아님 (->분산에서 오는 문제)
- 개발셋의 데이터 분포가 다름

⇨ 훈련-개발 세트라는 새로운 데이터를 정의(새로운 데이터의 부분집합)
: 훈련셋에서 일부 추출해서 빼고(훈련셋과 같은 분포), 이부분에서는 따로 훈련하지 않음

- 훈련 오류 1%, 훈련-개발 오류 9%, 개발 오류 10%
  -> 신경망이 훈련셋에서 잘 작동하더라도 같은 분포에서 온 훈련-개발셋의 데이터에 대한 일반화가 잘 되지 않았다(차이 큼)
  -> 같은 분포를 가졌더라도, 처음 보는 데이터(훈련-개발)에 대한 일반화는 잘 되지 않음 => 분산 문제

- 훈련 오류 1%, 훈련-개발 오류 1.5%, 개발 오류 10%
  -> 낮은 분산 문제 => 전형적인 데이터 불일치 문제
  -> 훈련-개발셋과 개발셋은 다른 분포에서 왔기에

- 베이즈(인간) 오류 0%, 훈련 오류 10%, 훈련-개발 오류 11%, 개발 오류 20%
  - 회피가능 편향이 높다. 훈련셋에서도 잘하고 있지 않음, 분산은 작음
  - 데이터 불일치

*인간 ← Avoidable bias → Training ← Variance → Training-Dev ← Data Mismatch → Dev
If 개발셋 성능과 테스트셋 성능에 큰 차이가 있다면, 개발셋에 대한 과적합 조정이 들어간 것
-> 더 큰 개발셋 필요 (테스트셋보다 개발셋에서 더 좋은 성과를 내며 큰 격차를 만들 수 있는 방법은 어떻게든 개발셋을 과적합하는 것)

<데이터 불일치> -> 오류분석

테스트셋 과적합 방지를 위해선 개발셋을 봐야함 -> 개발셋과 비슷해보이는 훈련셋 얻기

[전이 학습] : 한가지 업무에서 학습한 지식을 다른 업무에 적용시킬 수 있음(A -> B 순차적)

=> 출발 지점에 문제 관련 데이터 많고, 도착 지점에 관련 데이터가 적은 경우 전이 학습 필요

X : 이미지, y : 개체
전이학습을 하기 위해 신경망 네트워크의 마지막 결과값 레이어를 삭제, 주어진 가중치도 삭제
-> 마지막 레이어에 대해서만 가중치 한세트를 무작위로 초기화해 생성 (W[l], b[l])
-> 새 데이터셋에서 신경망 다시 훈련시킴
: 데이터셋 작으면, 마지막 레이어의 가중치(W[l], b[l])만 다시 훈련, 나머지 매개 변수는 고정
데이터셋 크면, 나머지 신경망에 대한 모든 레이어 재훈련
이미지 인식기능(원본 데이터셋)에 대한 첫번째 단계(X, y)의 사전훈련(pre-training)
새 데이터셋에서 가중치를 나중에 업데이트하는 미세 조정(fine-tuning)

X : 오디오, y : 해석
음성을 글로 옮겨 출력하기 위해 음성인식 시스템 훈련시킴
전이학습을 하기 위해 신경망 네트워크의 마지막 결과값 레이어를 삭제, 주어진 가중치도 삭제
한 가지 결과값이 아니라 신경망에서 여러 개의 새로운 레이어 생성가능
-> wakeword 탐지 문제를 위한 레이블 y를 붙이기 위해서

[다중 작업 학습] : 신경망이 여러가지 일을 동시에 하게 만들고, 각각의 작업이 다른 작업들 도움

각각의 이미지가 복수 라벨 가질 수 있음(여러 이미지 요소 ex. 한 사진안에 신호등, 사람 등)

개별 신경망 여러 개를 훈련하는 방법도 있는데, 이는 하나의 네트워크가 여러 개를 실행하도록 하는 방법이다. 하지만 신경망의 초반 특성 일부가 다양한 물체들 간에 공유될 수 있는 경우, 한개의 신경망을 훈련시켜 여러 개의 일을 할 수 있도록 하는 것이 여러 개의 신경망을 완전히 따로 훈련시키는 것보다 성능면에서 낫다. <다중 작업 학습의 힘>

어떤 이미지에는 일부 라벨밖에 없고, 나머지 라벨은 알 수 없거나 (?) 신경쓰지 않는 경우에도 알고리즘 훈련할 수 있다. -> 손실함수에서 여러 개의 라벨의 총합을 구하는 것이 아닌 라벨 작업이 된 것만 더함

<다중 작업 학습 효과적일 때>

- 공유된 저수준 특성으로 도움을 받을 수 있는 작업 훈련시킬 때
  ; 자율주행차의 신호등, 보행자, 차량 인식 특성이 정지 신호 인식에 도움을 주는 것과 비슷 – 모두 도로 특성 가지고 있음
- 대게 각 작업이 보유한 데이터의 양이 비슷하다. 다른 작업들이 한 가지 작업보다 훨씬 더 많은 데이터를 가지고 있어야 한다. (A1 작업 100개 물체, --- A100 작업 100개 물체)
  ; 9900(A1 ~ A99)개 -> 100개 <전이학습과 비슷>
- 모든 작업을 잘 수행할 수 있을 정도로 큰 신경망을 훈련시킬 수 있을 때

- 전이학습 VS 다중 작업 학습

전이학습 – 데이터량이 적은 경우

다중 작업 학습 – 작업의 양이 크고, 모든 작업을 한번에 훈련시킬 수 있을 때 / 한 개의 신경망이 여러 작업을 처리할 수 있도록 / 컴퓨터 비전 객체 탐지에 주로 사용

[End-to-end Deep Learning]

여러 단계를 하나의 신경망으로 변환 / 최종 훈련 세트를 가지고 x와 y의 함수 매핑을 직접 학습하면서 많은 중간 단계를 우회

<장점>

- 데이터를 순수하게 반영할 수 있게 해줌
- 필요한 구성요소를 직접 설계하는 일이 적고, 설계 작업 흐름을 단순화할 수 있다.

<단점>

- 데이터가 많이 필요하다. X-Y 매핑을 직접 학습하려면 많은 X, Y가 필요하다.
- 잠재적으로 유용할 수 있는 설계된 수제 요소를 배제한다.