



MACHINE LEARNING

한글번역 Full Script

Week 1

No.	Title
Week 1-1	Welcome to Machine Learning

00'00"

기계 학습이란 무엇일까? 우리는 아마 자신도 모르는 사이에 하루에도 수십 번 사용하고 있을 것이다. 구글이나 빙(Bing)에서 웹 검색을 할 때마다 적절한 검색 결과를 보게 되는데, 이것은 각 검색엔진의 기계학습 소프트웨어가 검색 페이지를 분류하는 좋은 알고리즘을 갖고 있기 때문이다. 페이스 북이나 애플의 사진 어플이 사용자의 사진을 보고 친구를 인식하는 것 또한 기계 학습이다. 엄청난 양의 스팸 메일을 걸러주는 필터 또한 컴퓨터가 스팸이 아닌 메일과 스팸 메일을 구분하는 방법을 학습했기 때문이다. 이것 또한 기계 학습이다. 컴퓨터가 프로그램 되지 않고도 학습하는 데에는 과학적인 원리가 있다. 나는 로봇이 집을 청소하도록 만드는 연구 프로젝트를 진행 중이다. 어떻게 하면 될까? 바로 로봇이 당신이 청소하는 모습을 보고 학습하게 만들면 된다. 당신이 줍는 물건이나 놓는 장소를 보고 당신의 모습이 보이지 않더라도 똑같은 일을 할 수 있다. 내가 여기에 관심이 있는 이유 중 하나는 바로 인공지능 때문이다. 정말 지능적인 머신을 만들 수 있다면 여러분과 내가 할 수 있는 일은 무엇이든지 시킬 수 있다. 많은 과학자들이 이 분야에서 진전하기 위해서는 신경망이라 불리는 알고리즘 학습을 이용해야 한다고 말한다. 신경망 알고리즘은 인간의 두뇌 활동을 모방한 것이다. 이것 또한 강의에서 다룰 예정이다. 이 강의에서는 기계 학습과 스스로 이를 실습하는 방법을 가르칠 것이다. 강의 신청을 해서 함께 배우길 바란다.

No.	Title
Week 1-2	Welcome

00'00"

기계 학습에 대한 무료 온라인 강의에 온 여러분을 환영한다. 기계 학습은 최근 가장 흥미로운 과학분야 중 하나이다. 여러분은 이 강의를 통해 기계학습의 최첨단 이론뿐만 아니라 알고리즘을 구현하는 방법을 배우게 될 것이다. 여러분은 아마도 이 학습 알고리즘을 인지하지 못한 채 매일 여러 번 사용하고 있을 것이다. 구글이나 빙을 통해 인터넷에서 무언가를 검색할 때, 검색엔진이 효과적인 이유는 구글이나 마이크로소프트가 설치한, 검색결과를 어떻게 내놓을지를 배우는 학습 알고리즘이 때문이다. 매번 페이스북이나 애플의 사진 앱을 사용할 때마다 여러분 친구의 얼굴을 인식하는데, 이 또한 기계 학습이다. 이메일을 확인할 때 스팸 필터로 여러분이 스팸 메일을 걸러내지 않아도 되는 것 또한 학습 알고리즘의 예다.

00'50''

내가 이렇게 흥미로움을 느끼는 이유 중 하나는 AI로 인간처럼 지능이 높은 기계를 언젠가 만들어 낼 수 있을 것이기 때문이다. 아직 이 목표를 달성하기엔 역부족이긴 하지만, 많은 인공지능 부문 연구진들은 인간의 뇌가 학습하는 것을 모방한 학습 알고리즘이 가장 최선의 방식이라고 한다. 여러분은 이것에 대해서도 배우게 될 것이다. 여러분은 이 과정을 통해 최첨단의 기계 학습 알고리즘을 배울 것이다. 하지만 당신의 해결할 문제를 잘 모르면서 이 알고리즘이나 수학을 아는 것만은 충분하지 않다. 그래서 우리는 여러분이 각각의 알고리즘이나 온 설명과 더불어 이를 활용할 수 있도록 많은 시간에 걸쳐 연습할 것이다. 그렇다면 왜 기계 학습이 최근 들어 더 널리 퍼지고 있을까? 기계 학습 분야가 AI, 혹은 인공지능이라 불리는 분야에서 파생되었기 때문이다. 우리는 기초적인 몇 가지의 프로그래밍을 통하여 기계로 하여금 A에서 B까지 최단 경로를 알아내는 것을 할 수 있게 하였다. 하지만 대부분의 경우 우리는 인공지능 프로그램을 웹 검색이나 사진 태깅, 스팸메일 거르기 등 더 흥미로운 것들을 위해 활용하는 방법은 알지 못했다. 앞서 언급한 것을 하기 위해서는 기계가 스스로 학습하도록 해야 한다는 것을 깨달았다. 따라서 기계 학습은 컴퓨터의 전에 없던 새로운 능력이며, 현재의 많은 영역과 기초 과학분야에서 활용되고 있다. 나는 현재 기계 학습과 관련해서 업무를 하고 있고, 최근 몇 주 동안 헬리콥터 파일럿, 생명과학자, 혹은 여기 스텐포드에서 함께 일하는 동료 컴퓨터 시스템 공학자들과 대화를 나누었고, 평균적으로 1주에 2-3번은 그들이 가진 문제에 학습 알고리즘을 적용하고자 하는 실리콘밸리의 종사자들로부터도 메일을 받고 있다. 이는 기계 학습의 다룰 수 있는 범위에 대한 조짐이라고 볼 수 있다. 자동 로봇공학이나 컴퓨터 생명공학 혹은 그 외 실리콘밸리의 수많은 분야에서 기계 학습은 영향을 끼치고 있다.

02'52''

기계 학습의 또 다른 예가 있는데, 그것은 데이터베이스 마이닝이다. 기계 학습이 활발해진 이유 중 하나는 웹과 자동화의 성장인데, 이는 우리가 그 어느 때 보다 많은 데이터 양을 보유하고 있다는 것을 의미한다. 그래서 요즘 수많은 실리콘 밸리의 회사들은 웹 클릭으로 얻어지는 데이터, 즉 클릭스트림 데이터를 수집하고, 이 데이터를 분석하기 위해 기계 학습 알고리즘을 사용하여 사용자를 더 이해하고 사용자에게 더 나은 서비스를 제공해주고자 한다. 이는 실리콘밸리의 많은 부분을 차지하고 있다. 의료 기록을 살펴보자. 자동화의 출현으로 오늘날 우리는 전자 의료 기록을 보유하고 있는데, 이 의료 기록을 의학 지식으로 만들 수 있다면 질병을 좀 더 잘 이해할 수 있다. 컴퓨터 생명공학의 경우에도 역시, 자동화의 출현을 통해 생물학자들이 유전자나 DNA 염기서열분석 등의 많은 데이터를 모으게 되었고, 기계 학습 알고리즘은 우리가 인간 유전자와 그것이 인간에게 의미하는 바에 대해 더 잘 알 수 있게 한다. 공학의 경우에도 마찬가지로 많은 공학 분야에서 훨씬 더 많은 데이터 세트가 존재하게 되면서 학습 알고리즘의 필요성이 높아지고 있다.

04'01''

두 번째 범위의 기계 응용 프로그램의 프로그래밍은 사람이 직접 할 수 없는 범위다. 그 예를 들자면 나는 수년간 자동화 헬리콥터와 관련해 일해왔다. 나와 연구진은 헬리콥터가 스스로 날수

있게 하는 프로그램의 프로그래밍을 어떻게 해야 할지 알 수 없었다. 여기서 유일하게 가능한 방법은 바로 컴퓨터가 스스로 헬리콥터를 날게 하도록 배우는 것이었다. 필기 인식의 경우, 미국 내에서나 국제로 우편을 발송하는 비용이 저렴해진 이유 중 하나인데, 당신이 봉투에 필기한 것을 학습 알고리즘이 이를 인식하여 자동으로 편지가 발송되야 하는 루트를 찾아낸다. 따라서 수천 마일 이상 떨어진 곳까지 편지를 발송하는 데 얼마 들지 않는 것이다. 그리고 자연어 처리과정이나 컴퓨터 비전 분야의 경우, 언어나 이미지를 이해하기 위한 인공지능 분야라고 할 수 있다. 오늘날 대부분의 자연어 처리나 컴퓨터 비전은 기계 학습이 적용되어 있다. 학습 알고리즘은 또한 자기-맞춤화 프로그램에 광범위하게 사용되고 있다. 여러분이 아마존, 넷플릭스, 혹은 아이튠즈 지니어스에 접속할 때 마다 이들은 영화나 제품 또는 음악을 추천한다. 이것 또한 학습 알고리즘이다. 수백만 명의 유저가 앞서 언급한 회사의 서비스나 프로그램을 사용하는데, 수백만 명을 위한 각자 다른 프로그래밍을 하는 것은 불가능하다. 맞춤형 상품 추천을 가능케 하는 유일한 방법은 여러분의 선호도에 맞춰지도록 학습을 하는 것이다. 마지막으로 학습 알고리즘은 인간의 학습법과 뇌 구조를 이해하기 위해서도 사용되고 있다.

05'55"

여러분은 원대한 인공지능의 꿈을 달성하기 위해 연구들이 어떻게 이를 사용하였는지도 학습할 것이다. 몇 달 전, 한 학생을 통해 12가지의 최고 정보통신 기술에 대한 기사를 접하게 되었다. 이 기술은 IT업계의 고용팀 관리자가 거부할 수 없는 것이었다. 좀 지난 기사이긴 했지만, 리스트에 적힌 12개의 기술 중 가장 위에 기재된 것은 바로 기계 학습이었다. 이 곳 스탠포드에는 기계 학습을 배운 학생을 채용하려고 나에게 연락하는 채용 관리자가 기계 학습을 배워 졸업하는 학생보다 더 많다. 따라서 기계 학습 분야는 공급보다 수요가 현저히 크며, 기계 학습에 대해 배우기 아주 적합한 시기이기에 나는 여러분에게 기계 학습을 강의하고자 한다. 강의를 통해 기계학습 문제나 알고리즘의 주요한 부분들을 다룰 것이다. 여러분은 주요 기계 학습 용어 중 몇 가지를 배우게 될 것이고, 각기 다른 알고리즘에 대해 이해하며, 어떨 때 어떤 것이 사용되는지 알게 될 것이다.

No.	Title
Week 1-3	Supervised Learning

00'00"

이번 강의에서는 가장 일반적인 기계 학습 문제인 지도 학습법에 대해 정의 하고자 한다. 지도 학습의 형식적인 정의에 대해 조금 뒤 다시 설명하겠지만, 어떠한 예가 있는지 알려줌으로써 이해를 도울 것으로 생각한다. 여러분이 집을 사는데 드는 비용을 예측한다고 가정하자. 얼마 전, 한 학생은 오리건 주의 '포틀랜드 협회'로부터 데이터 셋을 수집했다. 여러분이 그 데이터 셋을 그려보면, 이렇게 나타난다고 생각해보자. 여기 수평축에는 각기 다른 집들의 사이즈가 평방 피트로 표시되어 있고, 수직 축에는 1000달러 단위로 다른 집의 가격이 나타나 있다. 주어진 이 데이터에 따라, 750 평방피트의 집을 보유하고 있는 당신의 친구는 이 집을 팔기를 바라고, 이 때 가격을 얼마나 받을 수 있는지를 궁금해 한다고 가정해보자. 이 때 학습 알고리즘을 어떻게 유용하게 이용할 수 있을까?

00'54"

학습 알고리즘은 아마도 데이터에 직선 한 줄을 긋는다거나, 직선을 데이터에 맞출 것이다. 이것을 바탕으로, 아마 이 집은 15만 달러에 팔릴 수 있을 것처럼 보일 것이다. 하지만 이것은 학습 알고리즘을 이용할 수 있는 유일한 방법이 아니고, 더 나은 방법이 있을 수도 있다. 예를 들어, 데이터에 한 줄을 긋는 것보다, 이차함수나 이차 다항방정식을 사용하는 방법도 있을 것이다. 여러분이 그렇게 한다면 이렇게 여기 보이는 것처럼 예상치를 추정하는 것이 가능하다. 그리고 여기 보다시피 20만달러에 집을 팔 수 있을 것이다. 나중에 우리가 배우게 될 것은 선을 그을지 아니면 이차함수를 사용할지 두 가지 선택 중에서 무엇을 어떻게 선택하느냐이다. 이는 여러분의 친구에게 어떤 선택이 더 나은 결과를 가져다 줄 지에 달려 있다. 하지만 두 가지 모두 학습 알고리즘의 좋은 예라고 할 수 있다. 그래서 이것은 지도 학습의 예가 되는 것이다. 그리고 supervised learning이라는 용어는 데이터 셋에 알맞은 답인 '알고리즘'을 제공하는 것을 의미한다. 즉, 이 주택 관련 데이터 셋에서 나오는 각각의 사례에 어떤 것이 옳은 가격이라 실제 가격인지를 말해준다. 그래서 알고리즘은 여러분의 친구가 팔려고 하는 새로운 집에 적합한 답을 도출할 수 있는 것이다. 용어로 설명하자면 회귀 함수 문제가 될 것이며, 회귀 함수로 가격이라고 불리는 값을 예측하기 위해 노력한다. 그래서 실제적으로 가격은 센트 단위로 반올림 될 것이다. 그래서 가격은 이산 값이라고 할 수도 있지만, 보통 우리는 집의 가격이 스칼라 값으로, 즉, 실수라고 생각하기 때문에, 회귀라는 용어는 연속형 값의 속성을 예측하는 것을 의미한다. 또 다른 supervised learning의 예가 있는데, 나와 몇몇 친구들이 실제로 이전에 이 분야에서 일했다. 여러분은 의료 기록을 보고, 유방암이 악성이나 양성이라고 예측한다고 생각해보자.

03'16"

어떤 사람이 유선 종양, 즉 가슴의 어떤 응어리를 발견했을 때, 악성 종양일 경우, 몸에 좋지 않고 위험하며, 양성 종양일 경우에는 위험하지는 않다. 그래서 많은 사람들이 이에 대해 많이

신경을 쓴다. 수집된 데이터 셋을 보고 가정해보자, 여러분의 데이터 셋의 가로축에는 종양의 크기, 세로축에는 0이나 1, 혹은 yes 나 no로 표기할 것이다. 종양이 악성인 경우에는 1, 양성이나 악성이 아닐 경우에는 0으로 표기할 것이다. 우리가 본 종양이 양성으로 판정되었을 때 이렇게 보인다고 가정하자. 하나는 이렇게, 하나는 이렇게 말이다. 슬프게도 우리는 악성 종양을 종종 볼 수 있고, 하나 하나 사이즈를 측정해 나갈 것이다. 5개의 양성 종양의 예가 여기 아래 있고, 악성 종양으로 세로축의 값이 1인 5개의 예가 있다. 우리에게는 불행히도 유선 종양이 있는 친구가 있으며 그 친구의 종양의 크기가 이 정도라고 가정해 보자. 기계 학습의 질문으로는 가능성의 측정할 수 있는지와, 발견된 종양이 악성과 양성일 가능성이 어느 정도 되는지가 나올 수 있다. 용어로 설명하자면 이는 분류 문제의 예로 볼 수 있다.

04'39"

분류라는 용어는 이산 값을 예측하는 것이라고 볼 수 있다. 여기서는 0이나 1 그리고 악성 혹은 양성인 것이다. 분류 문제에서 여러분은 2개의 가능한 결과에 대한 2개 이상의 값을 갖게 될 수도 있다. 좋은 예로는 아마 세 가지 종류의 유방암이 있기 때문에 여러분은 아마 0, 1, 2,나 3의 값을 도출할 수 있을 것인데, 0의 경우는 양성이다. 암이 아닌 양성 종양의 경우이다. 1의 경우에는 1번 타입의 암으로 3은 3번째 타입의 암이라고 볼 수 있다. 하지만 이는 역시 분류 문제가 있을 수 있는데, 왜냐하면 다른 이산 값의 결과물이 암이 아님, 혹은 1번 타입 암, 2번 타입 암, 혹은 3번 타입 암을 칭하기 때문이다. 분류 문제에 있어서는 자료를 다르게 기입하는 방법이 있다. 조금 다른 기호 셋을 사용해 이 데이터를 기입하겠다. 그러니까 종양의 크기가 악성이나 양성 종양을 예측하는데 사용된다면 데이터를 이렇게 기입할 수 있다. 다른 기호를 사용해서 양성과 악성 혹은 음성과 양성 종양을 구분할 것이다. 크로스를 그리는 것 대신, 양성 종양에는 O를 기입할 것이다. 그리고 악성 종양에는 X를 기입할 것이다. 여러분이 좀 더 잘 이해하기를 바란다. 이렇게 자세히 기술해보겠다.

06'25"

다음과 같이 선이 형성된다. 그리고 원이나 십자와 같은 다른 기호를 사용해서 악성과 양성 종양을 구분해서 보여준다. 이제 이 예에서 악성인지 양성인지 예측하기 위해 종양의 크기라는 단 하나의 값만을 사용했다. 하나 이상의 값을 가지고 있는 기계 학습의 경우도 여럿 있다. 여기 그 예가 있는데, 종양의 크기를 아는 것 대신에 우리가 환자의 나이와 종양의 크기를 알고 있다고 하자. 이 경우에는 아마 우리는 환자의 나이와 관련된 데이터 셋과 종양 크기에 대한 데이터 셋을 보유하고 있을 것이며, 이렇게 보일 것이다. 그리고 다른 환자 데이터 셋의 경우 조금 다르게 보일 것인데, 악성 종양이 있는 환자라면 십자 모양으로 표기되어 있을 것이다. 그래서 친구 중 누군가가 불행히 종양이 있으면, 아마 그의 종양의 크기나 나이가 여기 부근 어딘가라고 하자. 이러한 데이터 셋이 있으면 학습 알고리즘은 데이터에 직선을 그어서 악성 종양과 양성 종양을 구분해 낼 것이다. 그러면 학습 알고리즘은 2가지의 종양을 구분해 내기 위해 이러한 직선을 그리게 될 것이다. 그리고 여러분 친구의 종양이 저쪽 선 너머에 위치한다면, 학습 알고리즘은 여러분 친구의 종양을 다행스럽게도 양성으로 판정한다. 그 외의 기계 학습 문제에서는 더 많은 특성이 존재하고, 이것을 연구하는 내 동료들은 유선 종양 덩어리의 두께,

종양 세포 크기의 균일성, 종양 세포 모양의 균일성이나 다른 특성들을 고려한다.

08'32"

이는 우리가 나중에 배우게 될 흥미로운 학습 알고리즘이며, 이 알고리즘은 두 세 개의 특성 이 아니라 무수히 많은 특성을 다룰 수 있다. 이 슬라이드에서 총 5가지의 특성들을 나열해 놓았다. 여기 2개는 이 축에 있고, 3개는 이 위쪽에 있다. 그래서 몇 가지 학습 문제의 경우에는 3-5가지의 특징을 사용하고 싶은 것이 아니라 무한대의 특성을 사용하고 싶을 때가 있다. 그러면 학습 알고리즘은 무수히 많은 특성을 활용해 예측 한다. 무한대의 정보를 컴퓨터에 저장하면 컴퓨터의 저장공간이 부족해 질 것이다. Support Vector Machine이라 불리는 알고리즘에 대해 논할 때, 아주 간결한 수학 공식이 필요한데, 이 공식으로 인해 컴퓨터가 무한의 특성을 다룰 수 있을 것이다. 내가 여기 2가지의 특성과 오른쪽에 3가지의 특성을 기입하지 않고, 대신 무한으로 긴 리스트를 작성하고, 계속 더 많은 특성을 기입한다고 하자. 우리는 이것을 해결하기 위한 알고리즘을 생각해보자. 요약하자면, 오늘 강의에서 우리가 배우게 될 supervised learning이다.

09':47"

Supervised learning의 개념은 모든 데이터 셋에서 어떤 것이 정답인지를 지시 받는 것이다. 앞서 말한 예시처럼 집의 가격이나 종양이 악성 혹은 양성인지에 대한 내용이다. 우리는 회귀 문제에 대해서도 다루었다. 회귀란 연속적인 값을 알아내는 것을 의미한다. 우리는 분류 문제에 대해서도 다루었고, 이는 이산 값을 예측하는 것을 말한다. 마치기 전에, 문제를 보자.

Q1) 여러분이 회사를 경영하고 있고, 두 가지 문제를 해결하기 위해 학습 알고리즘을 개발하려 한다고 가정하자. 첫 번째 문제는 여러분이 동일한 물품에 대한 재고가 아주 많다는 것이다. 따라서 여러분은 수천 개의 동일한 물품의 재고를 가지고 있고, 3달 내에 얼마나 팔 수 있을지를 알고 싶어한다.

Q2) 수많은 이용자가 여러분의 서비스를 이용하고 있고, 각 소비자의 계정을 평가하기 위한 프로그래밍을 하려 한다. 그리고 각각의 계정이 해킹을 당했는지 보안이 위험한지 결정해야 한다. 이러한 문제들이 분류문제인가 아니면 회귀문제인가? 영상이 중단되면 마우스를 사용해서 왼쪽 사지선다 중 맞는 답을 골라라. 정답: 3번

11'20"

아마도 여러분은 이게 답이라고 생각했을 것이다. 1번 문제 같은 경우는 회귀문제이다. 왜냐하면 수천 개의 제품을 갖고 있으면 이것은 실제 값이기 때문에 연속 값이라고 할 수 있다. 내가 팔게 될 상품을 연속 값이라고 생각하면 된다. 2번 문제의 경우에는 분류 문제라고 할 수 있다. 왜냐하면 해킹을 당하지 않은 값을 0으로 표기해서 예측해 낼 것이기 때문이다. 해킹을 당한 경우에는 1로 표기할 것이기 때문이다. 이는 유방암에서 0이 양성이고 1이 악성인 것과 같은 맥락이다. 그래서 해킹여부에 따라 0과 1로 표기될 것이다. 이러한 알고리즘으로 이산 값을 알아낼 수 있을 것이다. 그리고 이산 값은 몇 없기 때문에, 이를 분류 문제로 분류하는 것이다.

이것이 supervised learning이며, 다음 영상에서는 학습 알고리즘의 또 다른 주요 알고리즘인 unsupervised learning에 대해 배울 것이다.

No.	Title
Week 1-4	Unsupervised Learning

00'00"

이번 영상에서는 기계 학습에서 2번째로 중요한 부분인 unsupervised learning에 대해 배울 것이다. 지난번 강의에서는 supervised learning에 대해서 다루었다. 잠깐 돌아가자면, 데이터 셋은 이렇게 생겼고, 각 예들이 양성인지 음성인지에 대해 적혀 있었으며, 양성 종양인지 악성 종양인지에 대해 다루었다. 지시 학습- supervised learning 의 각 예에는 소위 외부에서 말하는 정답이 무엇인지, 그러니까 양성인지 악성인지에 대한 답을 들을 수 있었다. 비지시적 학습 - unsupervised learning에서는 우리는 조금 다르게 보이는 데이터 셋을 받게 될 것이고, 이는 이름의 존재 여부나 아니면 이름이 전부 동일한지도 모르는 데이터 셋일 것이다. 그래서 무엇을 해야 할지, 그리고 데이터 값이 무엇을 의미하는지 지시 받지 않은 채로 프로세스를 시작하게 될 것이다.

00'46"

아무런 정보 없는 데이터 셋이 여기 있다. 이 데이터에서 어떠한 구조를 찾을 수 있겠는가? 이런 데이터 셋이 주어지면, unsupervised learning 알고리즘은 이 데이터가 다른 두 무리에 속해 있다는 것을 결정할 것이다. 그렇다, supervised learning은 이 두 가지 데이터를 다른 두 집단에 분류할 것이다. 이를 군집 알고리즘이라고도 부르며 많은 곳에서 활용된다. 한가지 예로, 구글 뉴스에서 군집 알고리즘이 많이 활용되는데, 이것을 본 적이 없다면 news.google.com에 들어가서 직접 보기 바란다. 구글 뉴스는 매일 셀 수 없이 많은 뉴스를 모으고 그에 알맞은 분야별로 묶어둔다. 예를 들어 여기를 한번 보면 BP 오일 유출 사건과 관련한 다른 뉴스 기사들을 볼 수 있다. 이 중 하나의 링크를 클릭해보자. 그러면 다음의 웹페이지가 뜰 것이다. 이것은 월스트리트지가 BP오일 유출 사건과 관련해 "BP가 마콘도를 죽이다" 라고 언급한 기사이다. 다른 링크를 클릭하면 다른 기사를 볼 수 있을 것이다. 이는 CNN이 게임과 BP 오일 유출에 관한 내용이며, 3번째 링크를 클릭하면 또 다른 기사를 볼 수 있을 것이다. 이것은 영국의 가디언지가 BP오일과 관련하여 쓴 기사이다. 그러니까 구글 뉴스가 하는 일은 수만 개의 뉴스를 수집하여 군집으로 묶는 것이다. 그러면 뉴스 기사는 한 주제 아래에 분류된다. 군집 알고리즘과 unsupervised learning 알고리즘은 많은 분야에서 활용된다. 여기 유전학과 관련한 예로 DNA 미세배열과 관련된 예가 있다.

02'38"

이 개념은 각각 다른 개체를 여러분이 선택해서 어떤 유전자를 가지고 있는지의 여부를 측정하는 것이다. 기술적으로는 어떤 유전자가 얼마만큼 존재하는가를 측정하는 것이다. 그러므로 이 빨강, 초록, 회색 그 외의 색으로 어떠한 유전자를 얼마나 보유하고 있는지의 여부를 나타낼 것이다. 여러분은 군집 알고리즘을 활용해 각기 다른 개체들을 다른 카테고리나 다른 사람으로 묶을 것이다. 이는 unsupervised learning이라고 할 수 있다. 왜냐하면 우리가 알고리즘에 미리 무엇이 1종류, 2종류 혹은 3종류의 사람이라고 지시하지 않고, 그 대신 많은 양의 데이터를 주기만 할 뿐이다. 이 데이터가 무엇인지 알 수 없고 어떤 종류인지도 알 수 없다. 그리고 우리는 어떤 사람인지도 알 수 없다. 그렇다면 여러분은 미리 정보가 주어지지 않았을 때 그 구조를 알아내고 자동적으로 분류할 수 있을까? 우리가 알고리즘에 데이터 셋에 대한 올바른 답의 예를 주지 않기 때문에 unsupervised learning이다. unsupervised learning 혹은 클러스터링은 많은 다른 application에 사용 된다. 이는 또한 대형 컴퓨터 클러스터를 구성할 때 쓰인다. 내 동료 중 몇 명은 대형 컴퓨터 클러스터에서 어떤 기기가 서로 호환을 잘 하는지 알아보고 있다. 호환이 잘 되는 기기들을 조합한다면 여러분의 데이터 센터의 일은 더 수월해 질 것이다. 두 번째 예는 소셜 네트워크 분석이다. 여러분이 어떤 친구에게 메일을 가장 많이 보내는지 알게 되면, 또는 페이스북 친구나 구글+ 서클의 데이터가 주어지면 우리는 자동적으로 여러분이 어떤 친구 집단과 가까이 지내는지, 그리고 어떤 사람들끼리 알고 지내는지 알 수 있을 것이다. 시장 세분화에 대해 이야기해보자. 많은 기업들은 아주 방대한 양의 고객 정보를 보유하고 있다.

04'28"

이 고객 데이터 셋을 보고 시장 세분화와 세분화된 시장에 고객을 분류할 수 있다면 여러분의 제품을 더 효율적으로 판매할 수 있을 것이다. 다시 한번, 이는 unsupervised learning인데, 우리가 고객 정보만을 보유하고 있고 미리 시장 세분화에 대한 정보나 어떤 고객이 세분화된 시장에 존재하는지 모르기 때문이다. 우리는 알고리즘으로 하여금 이 데이터를 모두 분석하여 이러한 결과를 알아내도록 한다. 마지막으로, unsupervised learning은 놀랍게도 천문학 데이터 분석에도 사용되는데, 이 군집 알고리즘은 어떻게 은하계가 형성이 되었는지에 대해 놀랍고도 흥미로운 이론을 제시한다. 이 모든 것들이 unsupervised learning의 예이다. 다른 예를 소개하겠다.

05'26"

칵테일 파티 문제인데, 여러분은 모두 칵테일 파티에 가봤을 것이라 생각한다. 파티가 있고, 방이 사람들로 가득 차 있으며, 모두 둘러앉아있고, 모두 얘기를 나누고 있으며, 많은 사람들이 대화를 나누고 있어서 목소리가 많이 겹치고, 앞사람의 말을 알아듣기도 어려울 정도이다. 두 사람이 하는 칵테일 파티라면 두 사람이 동시에 말하고 있을 테고, 이것은 상대적으로 작은 규모의 파티이다. 우리는 마이크를 사용할 것인데, 마이크는 여기 두 군데에 존재하고, 말하는 두 사람과 다른 거리만큼 떨어져 있고, 각 마이크는 말하는 두 사람의 목소리를 다른 조합으로 녹음하고

있다. 아마 화자 1의 목소리가 마이크 1에서는 조금 더 클 것이며 화자 2의 목소리가 마이크 2에서 조금 더 클 것이다. 왜냐하면 2대의 마이크가 다른 곳에 위치하기 때문이다. 하지만 두 개의 마이크 모두에 겹치는 소리가 있을 것이다. 이는 연구진이 녹음한 내용이다. 1번 마이크의 목소리는 이렇게 들린다. One (uno), two (dos), three (tres), four (cuatro), five (cinco), six (seis), seven (siete), eight (ocho), nine (nueve), ten (y diez). 그다지 재미있는 칵테일 파티는 아닌 듯 싶지만 두 가지의 언어로 1부터 10까지를 세고 있다. 이제는 2번 마이크이다. Uno (one), dos (two), tres (three), cuatro (four), cinco (five), seis (six), siete (seven), ocho (eight), nueve (nine) y diez (ten). 우리는 이 두 녹음을 칵테일 파티 알고리즘이라는 unsupervised learning 알고리즘에 제공하고, 구조를 알아낼 것이다. 알고리즘은 이 오디오 파일을 듣고 두 가지를 결합한 파일을 만들어 낼 것이다. 더불어 칵테일 파티 알고리즘은 두 가지로 분리한 오디오 파일을 만들어 낼 것이다. 여기에 그 결과물이 있다. One, two, three, four, five, six, seven, eight, nine, ten.

07'48"

즉, 한 파일에서 영어 목소리를 분리해냈다. 그리고 이것이 두 번째 파일이다. Uno, dos, tres, quattro, cinco, seis, siete, ocho, nueve y diez. 비슷한 상황의 파일이 또 여기 있다. One, two, three, four, five, six, seven, eight, nine, ten. 이 사람은 귀가해서 라디오에 대고 혼자 얘기하고 있다. 이것은 두 번째 녹음 파일이다. One, two, three, four, five, six, seven, eight, nine, ten. 이 두 가지의 녹음 파일을 알고리즘에 던져준다면, 음원이 두 가지 들린다고 할 것이다. 또 이 앨범이 첫 번째 음원이라고 할 것이다. One, two, three, four, five, six, seven, eight, nine, ten. 완벽하지는 않다. 목소리는 뽑아 냈지만 음악이 섞여 있기 때문이다. 그리고 이것은 2번째 결과물이다. 나쁘지 않다, 여기서는 완벽히 목소리를 제거했다. 음악만을 추출해 낸 것이다. 이제 여러분은 이 unsupervised learning 알고리즘이 얼마나 어려운지를 알 수 있을 것이다. 이런 응용 프로그램을 만들어 내려면 수많은 오디오 프로세싱을 위한 자바 음원 합성 프로그램이나, 음원 분리 프로그램이 필요한 것처럼 보인다. 이 알고리즘은 여러분이 방금 들은 것을 생성해 낼 수 있고, 이렇게 한 줄의 코드로 이런 결과물을 만들어 낼 수 있다. 물론 연구진들이 이 한 줄의 코드를 생각해 내는 데는 오랜 시간이 걸렸다.

09'56"

이것이 쉬운 문제라는 것은 아니다. 하지만 이는 적합한 프로그래밍 환경을 사용한다면 많은 학습 알고리즘이 정말 간단한 프로그램이 될 수 있기도 하다. 그래서 우리는 강의에서 옥타브 프로그래밍 환경을 사용할 것이다. 옥타브는 오픈소스 소프트웨어이며, 옥타브나 매트랩은 많은 학습 알고리즘은 간단히 몇줄의 코드로 실행된다. 향후 이 강의에서는 옥타브를 어떻게 사용하는지 알려줄 것이며, 그리고 당신은 알고리즘 몇 개를 옥타브에서 시행해 볼 것이다. 매트랩을 보유하고 있다면 그것을 사용해도 무방하다. 실리콘 밸리에서 많은 기계 학습 알고리즘과 관련해 옥타브에서 첫 프로토타입을 시행해 보는데, 그 이유는 옥타브가 이런 학습 알고리즘을 빠른 시행을 가능케 하기 때문이다.

10'41"

여기에는 SVD, 즉 특이값 분해와 같은 기능들이 존재한다. 하지만 여기에는 선형 대수학의 규칙이 존재하지만, 이는 옥타브에 탑재되어 있다. 이를 너가 C++나 Java에서 하려고 할 때 여러분은 아마도 다양한 C++ 혹은 자바 라이브러리의 복합구성을 연결할 수많은 코드를 필요로 할 것이다. 그래서 C++나 Java 또는 Python과 같은 언어로 해결하려 하면 해당 언어가 본래 가지고 있는 것보다 더욱 복잡해질 수 있다. 내가 기계 학습을 강의 한지 10년 정도가 되었는데, 여러분이 옥타브를 프로그래밍 환경으로 사용하면 더욱 빨리 배울 수 있다는 것을 알게 되었다. 그리고 옥타브를 학습 툴로, 프로토타이핑 툴로 사용하면 러닝 알고리즘을 프로토타입하고 배우는 시간을 더욱 절약할 수 있다. 그리고 사실 실리콘 밸리의 큰 기업에서 일하는 이들은 옥타브와 같은 툴을 활용해서 학습 알고리즘 프로토타이핑을 한다. 이것이 성공한 경우에 한해서만 C++, Java 혹은 다른 언어들을 사용한다. C++언어에서부터 시작하는 것 보다 이런 알고리즘을 옥타브를 활용하면 더욱 빨리 활용할 수 있다. 그래서 나는 강의자로서, 나를 믿으라고 몇 번 말하겠지만, 옥타브를 사용해 본 적이 없는 사람들에게 한번 믿어보라고, 여러분에게 권하고 싶다. 정말 여러분의 개발 시간을 단축시켜 주기 때문이다. 나는 많은 사람이 이것을 활용하는 것을 직접 보았고, 여러분 역시 기계 학습 연구자로서 혹은 기계 학습 개발자로서, 다른 언어가 아닌 옥타브를 활용한다면 더 생산적으로 일할 수 있을 것이다.

12'21"

마지막으로 이번 강의를 마치기 전에, 복습 문제 하나를 풀어 보자. 우리는 오늘 unsupervised learning에 대해서 배웠고, 이는 당신이 수많은 양의 데이터와 이 안에서 우리를 위한 문제를 해결하기 위해 알고리즘만을 제시 받은 경우의 학습방법을 의미한다.

Q) 4가지의 예 중 어떤 것이 unsupervised learning 알고리즘일까? 4개 중 하나에서 unsupervised learning 알고리즘을 골라 보라. 체크박스에 체크하고, 오른쪽 아래 버튼을 눌러 답을 확인해 보라. 정답: 4번

영상이 멈추면 슬라이드의 문제에 답하라. 아마도 여러분은 스팸 폴더 문제를 기억하고 있을 것이다. 여러분이 데이터에 이름을 표기해 놨다면 이것은 supervised learning일 것이다. 뉴스의 예는 구글 뉴스 예와 동일한 것인데, 이것이 군집 알고리즘으로 기사를 모으는 알고리즘이라고 했으니, 이것이 unsupervised learning이다. 시장 세분화의 예는 내가 앞서 언급한 것인데, 이것 역시 unsupervised learning이다. 왜냐하면 고객을 자동적으로 세분화된 시장에 배분하기 때문이다. 그리고 마지막 예인 당뇨는 지난 강의에 있었던 유방암과 비슷한 내용이다. 양성이나 악성이나 아니라 당뇨인지 아닌지에 대한 예이므로 우리는 이것이 유선 종양 데이터처럼 supervised learning이라는 것을 알 수 있다. 지금까지 unsupervised learning을 배웠다. 다음 강의에서는 이 알고리즘은 어떻게 실행되는지 또 어떻게 구현할지를 배울 것이다.

No.	Title
Week 1-5	Model Representation

00'00"

우리가 처음 학습하게 될 알고리즘은 선형 회귀이다. 이번 강의에서 이 모델이 어떻게 생겼는지 학습하고, 그리고 더 중요한 지도학습의 전반적인 프로세스에 대해 배우게 될 것이다. 먼저 집값을 예상하는 예시로 시작해보자. 오리건주 포틀랜드 시의 주택 가격 데이터 세트를 활용해보고자 한다. 여기에 이렇게 다양한 가격대로 팔린 다양한 크기의 주택 수 데이터 세트로 좌표를 그린다. 여기 데이터 세트가 있고, 여러분의 친구가 집을 팔려고 하는데, 집 사이즈가 1250 스퀘어 피트고, 그 친구에게 집이 얼마쯤에 팔릴 수 있을지 말해주려고 한다. 먼저 모델을 적용시켜 본다. 이 데이터를 따라서 다음과 같이 직선을 그린다. 그렇게 되면 그 친구에게 집을 약 22만 달러에 팔 수 있을 것이라고 말해줄 수 있다.

01'02"

이것이 바로 지도 학습 알고리즘의 예시이다. 이러한 지도 학습은 예시마다 정답이 주어지게 된다. 즉, 우리에게 실제 팔린 집의 가격이 데이터 세트에 주어졌고, 이는 회귀라는 용어가 출력의 실제 값을, 즉 가격을 예측하는 것을 가리키는 회귀 문제의 예시가 된다. 다시 말해, 지도 학습 문제의 가장 흔한 형태는 분류 문제라고 한다. 여기서 우리는 암 종양을 보고 이것이 악성인지 양성인지를 구분하는 것과 같은 출력의 개별 값을 예측한다. 따라서 이는 zero-one 값 개별 출력이다.

01'52"

공식적으로 지도 학습에서는 데이터 셋이 주어지는데, 이 데이터 셋은 학습 집합이라고 불린다. 그래서 주택 가격의 예시를 보면, 각기 다른 주택 가격의 학습 세트가 주어졌고, 이제 이 데이터를 가지고 주택 가격을 예측하는 것을 학습하면 된다. 이제 이번 과정에서 활용하게 될 용어 몇 개를 집고 넘어가자. 꽤 많은 기호가 등장하게 될 것이다. 지금 당장 모든 기호의 뜻을 기억하지 못해도 괜찮다. 하지만 이를 알고 있다면 과정이 진행될수록 유용하고 편리할 것이다. 그러므로 이번 과정에서는 m 을 가지고 학습 예시의 수를 표기하겠다. 따라서 여기 데이터 셋에서 이 테이블에 47행이 있다고 가정하자. 그러면 47 학습 예시가 있으므로, $m=47$ 이 성립한다. 그리고 x 를 입력 변수 또는 특징이라고 하자. 여기에 x 가 있고, 이는 입력 특징이 된다. 또, y 는 내가 예측할 출력 변수 또는 타겟 변수로 정의할 것이고, 이는 여기 제2열에 있다.

02'55"

이를 바탕으로 (x, y) 로 하나의 학습 예시로 표기하겠다. 그러니까 이 테이블에서 한 개의 행은 한 개의 학습 예시에 해당한다. 특정 학습 예시를 나타내기 위해, $(x(i), y(i))$ 로 표기하여 이를 i^{th} 학습 예시라고 하겠다. 여기서 위첨자 i 는 지수(exponentiation)가 아니다. 여기 $(x(i), y(i))$ 에서 위첨자 i 는 여기 학습 세트에 있는 지표(index)이고 이 테이블에서 i^{th} 행을 나타낼 뿐이다.

그러니까 i 는 x 와 y 의 거듭제곱 형태가 아니라 그저 테이블에서 테이블에서 i^{th} (i 번째) 행을 나타내는 것이다. 예를 들면, $x(1)$ 은 첫 번째 학습 예시의 입력 값을 의미하므로 이는 2104가 된다. 여기서 x 는 첫 번째 행에서 x 이다. 또한 $x(2) = 1416$ 이 성립한다. 이게 두 번째 x 값이고, 그리고 $y(1) = 460$ 이다. 첫 번째 학습 예시의 y 값이 바로 (1)이 의미하는 것이다.

04'25"

아까 언급했다시피, 가끔 문제를 내서 여러분이 잘 따라오고 있는지 체크를 할 수 있도록 해놨다. 몇 초 후에 객관식 문제가 나타날 것이다. 그때 마우스로 정답을 선택하면 된다.

Q) 아래와 같은 학습 세트가 있다. $(x(i), y(i))$ 는 이 테이블에서 i^{th} 열을 나타낸다. 그렇다면 $y(3)$ 값은?

(정답 3번)

해당 학습 세트가 의미하는 것은 무엇인가? 이제 지도 학습 알고리즘이 어떻게 작용하는지 알아보자. 주택 가격 학습 세트와 같은 학습 세트를 봤고, 이를 학습 알고리즘에 적용해보자. 학습 알고리즘의 역할이 함수를 출력하는 것이라면 이는 보통 h 로 표기되고 이는 가정을 말한다. 여기서 가정의 역할은 주택 크기, 그러니까 여러분의 친구가 팔려고 하는 새로운 집의 사이즈를 입력하여 x 값으로 나타내고, 이에 대응하는 주택이 y 측정으로 출력되는 함수다. 그러므로 h 는 x 와 y 으로 이루어진 함수다.

04'25"

가끔 사람들이 이 함수가 왜 가정이라고 불리냐고 질문을 한다. 여러분 중에 아는 분도 있겠지만, 가설이라는 용어의 뜻을 사전이든, 과학 등 어느 분야에서든. 기계 학습에서 이 용어는 기계 학습 초기기에 사용된 용어이고 그대로 이어진 것이다. 그래서 주택 크기와 그 예측값을 나타내는 함수의 종류 이름으로 쓰기에는 썩 어울리지는 않지만, 가정이라는 용어는 이 함수에 적용 가능한 최고의 이름은 아닐 수 있다. 하지만 이는 기계 학습에서 사람들이 널리 사용하고 있는 표준 용어이다. 그러므로 왜 그런지 크게 신경 쓰지 않아도 된다.

06'10"

학습 알고리즘을 디자인할 때, 다음으로 이 가정 h 를 어떻게 표현할지에 대해 결정해야 한다. 이번과 이어진 몇 개의 강의에서 사용하기 위해서 이 가설을 표현할 최초의 선택을 할 것이고, 이는 다음과 같다. 우리는 h 를 다음과 같이 표현할 것이다. 이는

$h_{\theta}(x) = \theta_0 + \theta_1 x$ 와 같다. 이를 약칭하면 $h(x)$ 가 된다. 하지만 주로 위와 같이 θ 식을 사용할 것이다. 이제 좌표를 그려볼 건데, 이 좌표에서 y 가 x 의 선형 함수임을 예상할 수 있다. 그러므로 이것이 데이터 셋이다. 여기 함수의 역할은 y 가 x 의 직선 함수라는 것을 예측할 수 있다. 이는 $h_{\theta}(x) = \theta_0 + \theta_1 x$ 와 같다.

07'15"

그렇다면 왜 선형 함수인가? 때로 우리는 더 복잡한 비선형 함수도 적용해 볼 수 있다. 하지만 선형 사례가 간단한 구성요소이기 때문에 이러한 선형 함수로 먼저 예시를 들었다. 이후에 더 복잡한 모델, 더 복잡한 학습 알고리즘을 적용해 볼 것이다. 이 특정 모델 이름을 설명해주겠다. 이 모델은 선형 회귀라고 불리는데 예를 들어 이는 원래 한 개의 변수를 가진 선형 회귀고, 그 변수는 x 다. 이는 하나의 변수 x 로 모든 가격을 예측한다. 그리고 또 다른 이름은 단변량 선형 회귀인데 단변량이란 변수가 하나임을 나타내는 세련된 표현이다. 그러므로 이는 선형 회귀다. 다음 강의에서는 이러한 모델을 어떻게 구현하는지 알아보도록 하겠다.

No.	Title
Week 1-6	Cost Function

00'00"

이 강의에서는 비용 함수를 정의하고자 한다. 비용 함수는 최선의 직선을 알려준다. 선형 함수에서 우리는 트레이닝 셋에 대해 배웠는데, 여러분은 표기법 M 이 트레이닝 예의 수를 나타내고, m 값은 47이라는 것을 알 것이다. 우리가 예측을 하기 위해 만든 가설은 선형 함수였다. 용어에 대해 조금 더 설명하자면 이 θ_0 과 θ_1 은 Maldo의 매개 변수들을 안정화 시킨다. 이번 영상에서는 이 θ_1 , θ_0 변수 값을 어떻게 택할지를 다룰 것이다. θ_0 과 θ_1 변수 중 다르게 선택하면 다른 가정이 될 것이고, 다른 함수를 쓰게 될 것이다.

00'56"

여러분 중 일부는 오늘 배우게 될 내용이 익숙할텐데, 복습을 위한 몇 가지 예시들을 보자. θ_0 이 1.5이고 θ_1 이 0이라면 가설 함수는 이렇게 생겼을 것이다. 왜냐하면 여러분의 가설 함수가 $h(x)=1.5+0x$ 이고, 함수의 계속 값이 1.5이다. 만약 $\theta_0 = 0$, $\theta_1 = 0.5$ 인 경우에는, 가설 함수는 이렇게 보일 것이고, 이렇게 2를 지날 것이다. 그러면 여러분은 $h(x)$ 함수를 알 수 있게 된다. 혹은 $h_{\theta}(x)$ 이거나. 간결함을 위해 θ 를 빼도록 하겠다. 그러면 $h(x)$ 는 $0.5x$ 가 될 것이며, 이렇게 보일 것이다. 마지막으로, θ_0 이 1이라면 θ_1 은 0.5가 될 것이고, 가설 함수는 이렇게 보일 것이다. 이는 (2,2)를 지날 것이다. 보다시피 새로운 벡터함수, 또는 $h(x)$ 이다. 무엇을 여러분이 기억하든 $h_{\theta}(x)$ 이다. 앞으로는 그냥 $h(x)$ 라고 칭하겠다. 선형 회귀 함수에서 우리는 이미 여기에 표시한 트레이닝 셋이 있다. 이렇게 θ_0 과 1의 값이 있으며, 이렇게 직선을 그려 데이터와 잘 맞아 떨어지게 할 수 있다. 어떻게 데이터와 잘 맞아 떨어지는 θ_0 과 1의 값을 알아낼 수 있을까? 우리는 변수 θ_0 , θ_1 을 선택해야 $h(x)$ 의 우리 트레이닝 셋 (x,y) 의 y 값에 가까워 질 것이다.

03'00"

그러니 우리의 트레이닝 셋에서는 x 값이 모두를 결정하는 많은 예를 보게 될 것이고, 얼마의 값에 팔리는지 이미 알고 있다. 우리는 둘째의 값을 선택하여 적어도 이 트레이닝 셋에서는 x 의 값으로 y 값을 예측할 수 있게끔 해야 한다. 그렇다면 회귀 함수에서 우리가 하게 될 것은 최소화

문제를 해결하는 것이다. θ_0 과 θ_1 의 차이를 최소화를 하게 될 것이다. $H(x)$ 와 y 의 차이가 작도록 하고 싶다. 우리는 가설 함수에서 나온 결과와 실제 집의 가격의 차의 제곱을 최소화할 것이다. 좀 더 자세히 알아보자. 내가 $(x(i), y(i))$ 로 표기하여 i 번째 트레이닝 셋을 나타냈었다. 이 트레이닝 셋을 이와 같이 더할 것이다. i 의 범위는 1에서 m 까지일 것이고, 집 가격에 대한 입력값이 i 일 때의 가정에 대한 예측인 것이다. 거기서 실제 집 가격을 빼고, i 가 1부터 m 일 때 트레이닝 셋의 합을 최소화 하고 싶다. 이 제곱 오차는 주택가격의 예상값과 실제로 그 주택이 팔린 가격의 차이에 대한 제곱이다. 여기서 다시 한 번 이야기하자면 m 은 우리 트레이닝 셋의 크기이다. 그러니 여기 m 은 트레이닝 예시의 갯수다. 여기 #표시는 트레이닝 예시의 갯수에 대한 줄임 기호이다.

04'57"

우리의 수식을 좀 더 이해하기 쉽게 하기 위해 $1/m$ 으로, 아니 더 줄이기 위해 $1/2m$ 으로 만들어 보자. $\frac{1}{2}$ 로 줄이는 것은 반으로 줄이는 것이기에 수식을 좀 더 쉽게 할 수 있을 것이다. 똑같이 θ_0, θ_1 이 같은 과정을 위한 값으로 주어졌을 때, 해당 함수를 최소화할 수 있는 값을 주어야 한다. 좀 더 명확하게, 이 식에는 문제가 없다. 여기에 표현된 $h\theta(x)$ 이 우리가 해왔던 것인데, 이는 이것과 $\theta_1(x_i)$ 의 합과 같다. 그리고 여기 적힌 'minimize $\theta_0 \theta_1$ '의 의미하는 것은 θ_0, θ_1 이 포함된 이 표현을 최소화할 수 있는 $\theta_0 \theta_1$ 를 찾아야 한다는 것이다. 다시 말하면 이 평균이 $1/2m$ 이고, 트레이닝 셋의 예상치와 실제 값의 제곱 오차의 평균값을 더한 값을 최소화하는 문제는 θ_0, θ_1 을 찾는 것이다. 이는 선형 회귀 함수의 목적함수인 것이다. 좀 더 명확히 하면, 지금까지 해온 방식으로 보통 이렇게 보이는 비용함수를 정의한다. 즉, θ_0, θ_1 를 최소화하고자 할 때, $j(\theta_0, \theta_1)$ 라고 쓸 수 있다. 이것이 비용함수이며 이는 제곱 오차 함수라고 하기도 한다. 이것이 제곱오차함수라고 불릴 때에는 왜 제곱을 하는지 알아야 하는데,

07'18"

제곱 오차 함수는 대부분의 회귀 함수 관련 프로그램에서 가장 활용도가 높다. 비용 함수에서도 해당 빈도들이 높기는 하지만, 대부분의 회귀 함수 문제에 제곱 비용 함수가 가장 높은 빈도로 사용된다. 강의 막바지에 그 외의 다른 비용 함수에 대해서도 다룰 것이지만 선형회귀 문제들을 해결하기 위해 가장 적합하기 때문에 이것을 선택하였다고 보면 되겠다. 이것이 비용함수이고, Function J (θ_0, θ_1)로 표현되는 비용함수의 수학적 정의에 대해 다루었다. 이 함수의 경우, 축약되고, 무엇을 하는지 알기 어려워 보인다. 하지만 조금 뒤에 j 함수가 무엇인지, 왜, 그리고 어떻게 사용하는지에 대해 배울 것이다.

No.	Title
Week 1-7	Cost Function – Intuition I

00'00"

앞선 강의에서는 비용함수의 수학적 정의를 배웠다. 이번 강의에서는 예를 들어서 비용 함수에 대한 이해를 둘기로 하겠다. 이것이 지난 시간에 배운 내용이다. 우리는 우리의 데이터에 직선을 긋고, 우리는 θ_0 , θ_1 에 대한 가정으로 이것을 만들었고, 매개변수가 달라지면 직선 또한 달라진다. 이런 데이터를 만들어 내는 것은 비용 함수이고, 우리의 최적화 목표이다. 이 강의에서 비용함수를 더욱 가시화하기 위해 이 오른쪽에 보이는 것과 같은 간소화된 가정 함수를 사용할 것이다. 그리고 $\theta_1 \cdot X$ 라는 간소화된 가정을 사용할 것이다. 우리는 이것을 변수 θ_0 가 0이라고 가정할 수 있다.

01'00"

그러면 우리는 θ_1 이라는 변수 하나만 갖고 있고, 우리의 비용 함수는 $h(x) = \theta_1 * x$ 와 비슷하다. 단 하나의 변수이니 우리의 최적화 목표는 이것을 최소화하는 것이다. 여기서 $\theta_0 = 0$ 이면 (0,0)을 지난다. 이 간소화된 비용 함수로 비용 함수의 개념을 좀 더 확실하게 이해할 수 있을 것이다. 우리는 2가지 중요한 함수를 이해해야 한다. 첫 번째는 가정함수이고 두 번째는 비용함수이다. 가정함수는 $h(x)$ 이고, θ_1 의 페이스 값을 위한 x 함수이다. 가정 함수는 집 x 의 크기가 어느 정도이냐이다.

01' 59"

반대로, 비용함수 J 의 변수 θ_1 은 함수의 직선 기울기에 영향을 미친다. 이 함수들을 여기 써서 더 잘 이해해보도록 하자. 가정 함수부터 시작하면, 왼쪽을 보면, 트레이닝 셋의 (1,1), (2,2), (3,3)이 있다. θ_1 이 1이라고 할 때, 이 가정 함수는 여기 이 직선처럼 보일 것이다. 그리고 이제 가정함수인 x 축을 보면, 가로축이 x 라고 되어 있고, 이는 집의 크기를 나타낸다. 임시적으로 θ_1 을 0이라고 할 때, 이것의 j 값이 무엇인지 알아내고자 한다. 계속해서 어떻게 되는지 살펴보자. 여러분은 1의 가치를 평가절하할 것이다.

02 '57"

늘 그래왔듯 비용함수의 값은 이러하다. 이 중 몇몇은 제곱 오차 용어의 트레이닝 셋이다. 이것은 이것과 같고, 간략하게 하면 보는 것처럼 된다. 0의 제곱의 반복은 0과 같다. 비용 함수를 보면 여기 있는 것들은 0이다. 왜냐하면 우리의 예에서 (1,1), (2,2), (3,3)만을 갖고 있기 때문이다. 그러면 보이는 것처럼 식이 완성된다. 이러한 용어들은 0이 되는 것이고, 그래서 j 함수가 0이 된다고 했던 것이다. 그러니 $j(1)$ 은 0이 되는 것이다. 그러니 가정 함수는 집 x 의 크기가 어느 정도이냐이다. 반대로, 비용함수 J 의 변수 θ_1 은 함수의 직선 기울기에 영향을 미친다. 이 함수들을 기입하고 더 잘 이해해보도록 하자. 가정 함수부터 시작하면, 왼쪽을 보면, 트레이닝 셋의 (1,1), (2,2), (3,3)이 있다. θ_1 이 1이라고 할 때, 이 가정 함수는 여기 이 직선처럼 보일 것이다. 그리고 이제 가정함수인 x 축을 보면, 가로축이 x 라고 되어 있고, 집의 크기를 나타내는 것이다. 임시적으로 θ_1 을 0이라고 할 때, 이것의 j 값이 무엇인지 알아내고자 한다. 계속해서 어떻게 되는지 살펴보자. 여러분은 1의 가치를 평가절하할 것이다.

04': 19"

오른쪽에 비용함수 j 를 기입하겠다. 그리고 보다시피 이 비용함수는 변수 θ_1 에 대한 것이다. 이 비용함수를 그러면 가로축은 θ_1 이라고 할 수 있다. 그래서 j 함수가 0이 되는 것인데, 계속해서 x 까지 이것을 적어보자. 다른 예를 살펴보면 알다시피, θ_1 은 다른 값을 가질 수 있다. 그러니 이것은 음수, 0, 또는 정수일 수 있다. 그러면 내 가정 함수는 이렇게 보일 것이다. 기울기가 0.5인 직선이고 그러면 j 가 0.5일 때 값을 구해보자. 그러면 이는 $\frac{1}{2m}$ 이 될 것이다.

05'20"

비용함수는 이 선, 저 선과 저 선의 높이의 제곱의 합이 될 것이다. 이 수직 거리는 $y^{(i)} - h_{\theta}^{(x^{(i)})}$ 이다. 따라서 처음의 예는 (0.5-1)의 제곱인 것이다. 왜냐하면 가정함수가 0.5였고, 실제 값은 1이었기 때문이다. 두 번째 예는 (1-2)의 제곱이다. 그 이유는 가정이 1이고 실제 값이 2였기 때문이다. 마지막은 (1.5-3)의 제곱인데, 이는 2/3의 제곱과 같다. 왜냐하면 m 에는 3가지 트레이닝 예가 있기 때문이다. 그러면 괄호를 제외하고 이는 3.5이다. 즉, 3.5/6이고, 0.68정도이다. 그러니 $J(0.5)=0.68$ 인 것이다.

06' 40"

자 이제 기입해 보자. 계산 실수가 있었다. 0.58이다.

Q) 아래 그린 것처럼 $m=3$ 인 트레이닝 예를 가지고 있다고 하자. 우리가 세운 가정은 변수 θ_1 을 가진, $h_{\theta}^{(x)} = \theta_1 \cdot x$ 이다. 비용함수 $J(\theta_1)$ 이 다음 식과 같을 때, $J(0)$ 값은? 정답: 4번

하나 더 해보자. θ_1 이 0일 때 $j(0)$ 은 무엇일까? θ_1 이 0이면 $h(x)$ 는 이런 수평선이다. 오차를 계산해보자. $J(0)=1/2M$ 이고, 보는 바와 같다. 또 적어보면, 값이 2.3이고 θ_1 에 대한 다른 값도 모두 찾아낼 수 있다. 변수가 음수일 수도 있기 때문에, 이 경우에는 $h(x)$ 는 $-0.5 \cdot x$ 이고, $\theta_1 = -0.5$ 가 된다. 그러면 기울기가 -0.5인 직선이 된다. 이러한 오차를 계속 계산할 것이다.

08'14"

0.5의 경우, 굉장히 높은 오차이다. 그러면 5.25가 되고, θ_1 의 값이 달라질 경우를 계산할 수 있다. 여러분이 계산한 값은 이렇게 된다. 값의 범위를 계산함으로써 이런 결과 값을 계속 얻어낼 수 있다. 함수 $J(\theta)$ 는 무엇 일까? 정의하자면 θ_1 의 값들이다. θ_1 의 다른 값들은 다른 가정함수를 낳거나, 왼쪽에 보이듯 다른 선을 만들어낸다. θ_1 의 값이 다르면 J 의 값도 다르게 도출된다. 예를 들어 $\theta_1=1$ 이면 이 데이터를 이렇게 통과하는 직선을 도출 가능하다. 이 빨간 색으로 표기된 $\theta_1=0.5$ 는 이 선에 알맞게 되고, 파란 색으로 표시된 점은 이러한 수평선이 된다. 그러니 θ_1 의 각 값은 $J(\theta_1)$ 의 다른 값을 갖게 되고, 이것을 오른쪽에 표기할 것이다. 학습 알고리즘의 최적화 목표는 θ_1 의 값을 구하는 것이다. 이는 $J(\theta_1)$ 의 값을 최소화한다.

10':16"

이것이 회귀함수의 목표함수였다. 이 곡선을 보면, $J(\theta_1)$ 을 최소화 하는 값은 $\theta_1=1$ 일 때이다.

이것을 보면, $\theta_1=1$ 될 때 최적의 직선임을 알 수 있을 것이다. 아주 완벽하게 들어맞았다. 즉, $j(\theta_1)$ 의 값을 줄여서 데이터에 알맞은 직선을 찾아 내는 것이다. 여기까지다. 이 영상에서는 비용 함수를 이해하기 위해 몇 가지를 기록하고, 알고리즘을 간소화했다. 그래서 θ_1 이라는 변수 하나만을 갖게 되었다. 그리고 θ_0 은 그냥 0으로 두었다. 다음 강의에서는 원래의 문제 형성(formulation)으로 돌아가고 θ_1 과 θ_0 을 동시에 시각적으로 보여줄 것이다. 이것은 이번과 다르게 $\theta_0=0$ 인 경우는 아닐 것이다. 이것이 비용함수가 선형 회귀함수의 형성에서 하는 역할을 이해하는데 도움을 주기를 바란다.

No.	Title
Week 1-8	Cost Function – Intuition II

00'00"

이 강의에서는 비용 함수에 대해 더 깊이 알아볼 것이다. 여기서는 여러분이 contour plot과 익숙할 것이라 가정할 것이다. 여러분이 contour plot이나 contour figure을 잘 모른다면, 이 영상의 몇 그림이 아마도 여러분의 이해를 도울 것인데, 이해하기 힘들어서 그냥 안 듣고 넘어갈 수도 있을 것이다. 여러분이 넘어가더라도 다른 부분을 이해하는 데는 그렇게 문제가 없다. 그냥 이 부분만 모를 뿐이다.

00':31"

여기 문제를 만들어 보자. 여기 가설, 매개변수, 비용함수, 그리고 목표가 있다. 이전과는 다르게, 두 변수 θ_0 , θ_1 를 모두 다룰 것이다. 그리고 우리는 비용함수를 시각화 할 것이다. 지난번처럼, 우리는 가정함수 H 와 비용함수 J 를 알고 싶다. 여기 집 가격이 있다. 가정을 해보자. 저것처럼, 이것도 그렇게 좋은 가정은 아니다. 하지만 $\theta_0=50$ 이고 $\theta_1=0.06$ 일 때 여기 아래의 가정을 만들어 낼 것이고, 이런 직선을 그릴 수 있을 것이다. 이런 변수를 가지고 그려보자. 비용함수는 오른쪽에 있다. 지난 시간에 했던 것은, 우리는 θ_1 의 값만을 가지고 있었다. 다른 말로, θ_1 을 이렇게 표현할 수 있다. 하지만 우리는 변수가 두 개 있다. 그러니 선을 그리는 것이 조금 복잡해 질 것이다. 우리가 변수를 하나만을 가지고 있을 때, 이런 궁형의 함수가 그려진다. 우리가 변수가 2개 있기 때문에 비용 함수가 궁형처럼 보일 것이다. 트레이닝 세트를 보면 비용 함수가 이렇게 보일 수 있다.

02' 03"

이것은 3-D 표기법이고, 축은 θ_0 , θ_1 라고 표기되어 있다. 여러분은 θ_0 , θ_1 의 변수를 바꾸면서 비용함수 $J(\theta_0, \theta_1)$ 의 값을 달리할 수 있고, 이 높이는 θ_0 , θ_1 에서 특정 값으로 떨어져 있다. 그러면 이렇게 궁형의 그래프를 볼 수 있을 것이다. 3D로 나타내면, 이렇게 가로축이 θ_1 이고 세로축은 $J(\theta_0, \theta_1)$, 이고 이렇게 돌려볼 수 있다. 여러분이 이 볼 모양으로 보이는 표면의 비용함수 J 의 형태를 이해하기를 바란다. 남은 시간 동안 여러분에게 비용함수 J 를 표현했던 것처럼 3D로 나타내진 않을 것이다. 대신, contour figure라고도 불리는 contour plot을 사용할 것이다. 둘 다 같은 표현이다. 여기 contour figure의 예가 있다. 오른쪽에 보면 축은 θ_0 , θ_1 이다. 그리고 이 타원모양을 보면 $J(\theta_0, \theta_1)$ 와 같은 값을 가지는 부분들이다. 그래서 예를 들어, 내가 빨간 색으로 표기한 이 점, 이 점, 그리고 이 점을 보면, $J(\theta_0, \theta_1)$ 와 같은 값을 가지는 것이다.

03'57"

이것이 θ_0 , θ_1 축이고, 이 세 가지는 $J(\theta_0, \theta_1)$ 와 같은 값을 갖는다. 여러분이 contour plot을 잘 모른다면, 먼저 생각해 보길 바란다. 이는 궁형의 함수이며, 화면에 곧 뜰 것이다. 즉, 궁형 함수의 최소값, 바닥이 여기 이 점이고, 즉, 이 동심원의 중간쯤인 이 값이다. 이 동심원들은 모두 이 화면에서 같은 높이를 갖고 있다. 그리고 활 모양의 최소값은 여기 오른쪽 아래에 있다. 그러니 contour figures는 함수 J 의 시각화가 좀 더 용이한 방법이라 할 수 있다. 예를 몇 가지 더 보면, 여기 이 점이 있는데, 이는 $\theta_0 = 800$ 일 때, $\theta_1 = -1.5$ 이다. 이 점에서는, 그러니 이 빨갛게 표시된 점에서는 θ_0 , θ_1 의 값들과 일치하고, 우리 가정과 일치하는 것이다. θ_0 가 800정도일 때, 그러니 수직 축 800을 지날 때, 이 기울기는 -0.15 정도이다.

05'21"

이 직선은 데이터에 완전히 들어맞지는 않는다. 이 가정함수 $h(x)$ 는 이 θ_0 , θ_1 의 값과 함께 데이터에 들어맞지 않는다. 이것이 비용이다. 여기 나와있는 값이 최소값과 차이가 많이 나고, 더 높은 비용이다. 왜냐하면 이는 데이터에 들어 맞지 않기 때문이다. 예를 좀 더 보자면, 여기 다른 가정이 있고, 이것이 여전히 데이터와 들어맞지는 않더라도 이전 것 보다는 조금 더 낫다. 이 θ_0 , θ_1 의 값은 각각 360이나 0에 가깝다. 그러면 $\theta_0 = 360$ 이고, $\theta_1 = 0$ 이라고 하자. 이 변수는 가정에 들어맞는다. 바로 여기 평평한 선과 일치하는데 이는 $h(x) = 360 + 0 \cdot x$ 이다. 이것이 가정함수이다. 이 가정의 비용은 저 점에서 J 함수의 높이이다.

06' 38"

조금 더 예를 살펴보자. θ_0 가 이 값이고, θ_1 이 이 값일 때, 이런 가정함수 $h(x)$ 를 놓고, 이 또한 데이터와 들어맞지는 않는데, 사실 최소값에서 많이 떨어져 있다. 마지막 이 예시는 최소값은 아니지만 최소값에 근접해 있다. 그러니 데이터의 특정 θ_0 의 값과 큰 차이가 나지 않는 값은 아니라는 말이다. 이 중 하나는 θ_1 값 중 하나이다. 우린 특정 $h(x)$ 를 알아냈다. 그리고 이것은 완전히 최소값은 아니더라도 꽤 가깝긴 한 편이다. 이는 우리 트레이닝 예의 값과 가정의 오차 제곱의 합인 것이다. 이는 완전 최소값은 아니더라도 최소값에 많이 가까운 편이다.

07'34"

이러한 예가 여러분에게 비용함수 J 의 값이 어떻게 다른 가정에 대응하는지, 그리고 어떤 가정이 데이터의 최소값에 접근해서 나온 가정인지를 알아내는데 더 도움이 되는지 이해를 도왔기를 바란다. 우리가 원하는 것은 자동적으로 비용함수 J 의 값을 최소화하는 θ_0, θ_1 의 값을 알아내는 가장 효율적인 알고리즘을 알아내는 것이다. 우리는 소프트웨어 프로그래밍이나, 그래프를 그리는 것과, 숫자를 직접 읽는 것은 원치 않는다. 사실, 여러분은 나중에 더 복잡한 문제가 있으면 우리는 고차원적인 그래프와 변수를 보게 될 것이다. 그 그래프들은 그릴 수도 없는 것들일지도 모른다. 그러니 우리는 함수의 값을 최소화하는 θ_0, θ_1 을 찾아내는 소프트웨어를 갖기를 바란다. 다음 강의에서는 알고리즘이 어떻게 이를 찾아내는지 배울 것이다.

No.	Title
Week 1-9	Gradient Descent

00'00"

앞서 비용함수 J 에 대해서 배웠다. 이번 강의에서 여러분은 비용함수 J 를 최소화시키는 gradient decent라는 알고리즘에 대해 배울 것이다. Gradient descent는 좀 더 일반적인 알고리즘으로, 선형 회귀함수뿐 아니라 기계학습의 전 과정에서 사용된다. 강의 뒷부분에서 Gradient descent를 활용해 선형 회귀 비용함수 J 뿐 아니라 다른 함수의 값을 최소화하는 방법도 알아볼 것이다. 이번 강의에서는 임의함수 J 의 값을 최소화하는데 gradient decent를 활용할 것이다. 그 이후에 이 알고리즘을 우리가 선형 회귀함수라고 정의한 비용함수 J 에 활용할 것이다. 문제는 이렇게 구성될 것이다. 어떤 함수 $J(\theta_0, \theta_1)$ 은 우리가 최소화 시킬 다른 함수일 수도, 선형회귀에서의 비용함수일 수도 있다.

01'38"

그리고 우리는 $J(\theta_0, \theta_1)$ 값을 최소화하는 알고리즘을 생각해내고자 한다. 기울기 하강이 더욱 일반적 함수에 더 잘 적용된다. J 함수가 있고, $J(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$ 이며 이 J 함수를 최소화한다. 기울기 하강은 더 일반적 문제를 해결할 수 있다. 간결하게 하기 위해, 그리고 적기 편하게 하기 위해서 우리는 변수가 2개만 있다고 하자. 먼저 기울기 하강으로 θ_0, θ_1 에 대한 첫 번째 가정을 해 보도록 하겠다. 사실 어떤 값이든 크게 상관이 없기에 일반적으로는 $\theta_0 = 0, \theta_1 = 0$ 으로 놓는다. 따라서 두 값을 0으로 놓자. 우리는 기울기 하강으로 θ_0, θ_1 을 조금씩 줄여 $J(\theta_0, \theta_1)$ 의 값이 최소값이나 국소값이 될 때까지 계속해서 줄여 나갈 것이다.

02'15"

그림에서 기울기 하강이 어떤 역할을 하는지 보라. 여러분은 이 함수를 최소화시키고 싶어 한다. 이 축을 보면, 가로축은 θ_0, θ_1 이고 세로축은 J 이다. 여기서 높이는 J 의 값을 나타낸다. 우리는 이 함수를 최소화하고 싶다. 우리는 θ_0, θ_1 의 어느 값부터 시작할 것이다. θ_0, θ_1 을 함수의 표면 값에서부터 시작하면, 이 값과 상관 없이, 여기 이 값이 나타날 것이다. 나는 (0,0)으로 놓았지만,

다른 값으로 둘 수도 있다. 이제 여러분은 이러한 구덩이를 생각해보길 바란다. 이것이 어떤 녹지 공원의 풍경이라고 생각하고, 두 언덕이 있다고 생각해보자. 여러분이 언덕의 이 빨간 위치에 서 있다고 생각해 보자. 기울기 하강에서 여러분은 여러분 주위를 360도 돌려서 볼 수 있고, 내려가기 위해서 어떤 길로 첫 발을 내딛어야 할지 알고자 한다. 내가 내려가고 싶으면 최대한, 말 그대로, 빨리 내려가면 된다. 여러분이 이 위치에 서 있다면 여러분은 주위를 둘러보고 가장 최선의 길을 택해서 내려갈 것이다.

03'43"

여러분은 이제 다른 곳에 서 있다. 다시 한 번, 여러분은 주변을 둘러보고 어디로 내려갈지, 어느 쪽으로 작은 발걸음을 떼야 할지 생각할 것이다. 여러분이 어디로 갈지 선택했다면 여러분은 첫 발을 내디딜 것이고, 계속 걸어갈 것이다. 이 새로운 시점에서도 어떤 방향이 가장 빨리 내려갈 수 있는 길인지 선택해야 한다. 계속 걸어가면서 여기 국소점에 다다랐다. 기울기 하강은 이렇게 재미있는 속성을 갖고 있다. 우리가 처음 시작했을 때, 여기서 시작했다. 이 지점에서 말이다. 그리고 이제 우리는 기울기 하강을 사용해서 조금 오른쪽으로 옮겼다. 우리가 이것을 오른쪽 위에서 시작했다고 생각해 보자. 이 과정을 반복하면, 그러니 이 점에서 시작하면 가장 가파른 쪽에서부터 내려올 것이고, 다시 둘러보고 그 쪽으로 내려갈 것이다. 그리고 계속 내려간다.

04'43"

오른쪽으로 조금 걸어가면 기울기 하강은 오른쪽에 있는 2번째 국소 최적값(local optimum)으로 인도할 것이다. 여러분은 여기 첫 번째 점에서 시작해서 여기 국소 최적값에서 끝났다. 하지만 조금 다른 곳에서 출발했다면 또 다른 국소 최적값에서 끝날 수도 있었을지도 모른다. 이것이 기울기 하강의 특성이며 조금 뒤에 더 살펴볼 것이다. 이것이 그림에서 알아볼 수 있는 설명이다. 이제 수학적 부분을 살펴보면, 이것이 기울기 하강 알고리즘의 정의이다. 다음 식을 수렴할 때까지 계속 반복할 것이고, 나의 매개변수 θ_j 를 이번에 α 번 만큼 제외하도록 업데이트 할 것이다. 다시 보면 등식에는 여러 세부사항들이 있기 때문에 조금 풀어서 설명하겠다. 여기 $:=$ 를 assignment 표기를 위해 사용하겠다. 이는 assignment operator이다. 그러니 앞으로 $a := b$ 라고 쓰면 이것이 컴퓨터에서 의미하는 바는 a 의 값에 상관 없이 b 의 값을 계속해서 사용하라는 얘기이다. 이것이 의미하는 바는 b 의 값과 계속 같이 맞추라는 얘기이다.

06'10"

만약 $a := a + 1$ 이라고 한다면 a 의 값을 사용해서 1씩 더하라는 말이다. 반대로 $a = b$ 라고 한다면 이것은 참의 주장이다. 그러니 $a = b$ 라고 쓴다면 a 값이 b 와 같다고 주장하는 것이다. 왼쪽은 컴퓨터 처리를 하고, 우리는 새 값을 여기에 지정할 것이다. 오른쪽은 주장 부분이고, a, b 의 값이 동일하다고 주장한다. 그러니 $a := a + 1$ 이라고 쓸 수도 있고, 이는 계속해서 1씩 증가한다는 뜻인데, $a = a + 1$ 이라고 적지 않도록 주의하라. 이것은 절대로 같을 수 없는 값들이다. 이것이 정의의 첫 부분이다. 여기 α 는 학습 속도(learning rate)이라 불리는 값이다. 여기서 α 가 하는 역할은, 기본적으로 언덕을 내려오는 걸음걸이의 폭을 결정하는 것이다. α 가 크면 기울기 하강의 과정이 아주 과감하며, 과감하게 내려온다는 뜻이고, 그 반대는 우리가 아주 작은 걸음으로 내려온다는

얘기이다.

07'27"

α 를 결정하는 등의 얘기는 조금 후에 더 언급할 것이다. 그리고 여기 이 공식은 미분계수이다. 여기서 언급할 내용은 아니기 때문에 후에 좀 더 자세히 설명하도록 하겠다. 여러분 중 몇몇은 미적분학에 대해 잘 알고 있을 것이다. 여러분이 미적분을 잘 모르더라도 걱정하지 않아도 된다. 여기서 필요한 정도만을 가르칠 것이기 때문이다. 그리고 기울기 하강에서 한 가지 더 주의할 점이 있다. 기울기 하강에서는, 잘 알다시피, θ_0, θ_1 을 업데이트 할 것이다. 그리고 이 업데이트에서는 $j=0, j=1$ 을 사용한다. 그리고 여러분은 θ_0, θ_1 을 업데이트 할 것이다. 그리고 기울기 하강을, 여기 이 공식에 대해 실행할 때 주의해야 할 점은 θ_0, θ_1 을 동시에 업데이트하고 싶어질 것이라는 것이다.

08'33"

무슨 뜻이냐면 이 등식에서 우리는 " $\theta_0 := \theta_j -$ " 부분을 업데이트 할 것이고, " $\theta_1 := \theta_1 -$ "으로 나타낼 것이다. 이것은 오른쪽에서 적어야 하는 내용이다. θ_0, θ_1 에 대한 계산이 동시에 이루어져 여기에 대해서도 업데이트가 될 것이다. 이것은 기울기 하강의 simultaneous update를 정확하게 시작하는 방법이다. 그러면 이제 temp0은 저것과 같고, temp1은 저 것과 같도록 하여 기본 계산이 오른쪽에서 이루어지도록 할 것이고, 계산된 temp0, 1의 값을 오른쪽에 저장할 것이다. 이것이 바른 구현이므로 θ_0, θ_1 의 값도 업데이트 할 것이다. 반대로, 이것이 잘못된 구현이라면 동시 업데이트(simultaneous update)가 불가능할 것이다.

09'24"

잘못된 구현일 경우에는 우리는 temp0을 계산한 후 θ_0 를 업데이트하고, temp1을 계산한 후 θ_1 을 업데이트 해야 한다. 왼쪽과 오른쪽 실행의 차이는, 여러분이 여기 이 부분, 즉, 공식의 이 부분에서 보면 여러분은 이미 θ_0 을 업데이트했고, 새로운 θ_0 의 값으로 이 미분 계수를 계산하려 한다. 그러면 이는 여러분에게 왼쪽과 다른 temp1을 제공한다. 왜냐하면 여러분이 θ_0 의 새 값을 등식에 넣었기 때문에, 오른쪽의 값이 기울기 하강의 알맞은 구현이 아니기 때문이다. 나는 여러분이 왜 동시 업데이트(simultaneous update)를 해야 하는지에 대해서는 말하고 싶지 않다. 이것이 보통 기울기 하강을 실행하는 방법이고, 이에 대해서 좀 더 있다 배우게 될 것인데, 많은 경우 동시 업데이트를 실행하는 것이 자연스럽기 때문이다. 따라서 사람들이 기울기 하강 얘기할 때 항상 동시 업데이트를 얘기한다. 그러니 여러분이 동시 업데이트를 하지 않는다 해도 물론 여러분의 알고리즘은 잘 돌아갈 것이지만 맞지는 않을 것이다. 사람들이 말하는 기울기 하강이 아니고 다른 특성을 가진 다른 알고리즘이 될 것이다. 그리고 여러 가지 이유로 좀 이상한 방향으로 작동할 것이다.

10'41"

이것이 바로 여러분이 기울기 하강의 동시 업데이트를 활용해야 하는 이유이다.

Q) $\theta_0 = 1$, $\theta_1 = 2$ 이고 우리가 θ_0 과 θ_1 을 다음 규칙을 사용해서 동시 업데이트 한다고 하자.
그렇다면 θ_0 과 θ_1 의 결과 값은 무엇인가? 정답: 2번

지금까지 기울기 하강 알고리즘을 대략적으로 살펴보았다. 다음 강의에서는 내가 적었지만 정의하지는 않았던 미분계수를 자세히 다룰 것이다. 여러분이 미적분학 강의를 이전에 들었고 부분 미분계수와 미분계수에 친숙하다면 잘 알 테지만 만약 여러분이 미적분을 잘 모른다 해도 걱정하지 말길 바란다. 다음 강의에서 여러분은 미분계수를 계산하기 위한 미적분을 배우게 될 것이고, 미적분을 잘 모르더라도, 부분 미분계수를 들어본 적이 없더라도 크게 상관이 없을 것이다. 다음 강의에서 기울기 하강을 보다 잘 이해해 이를 활용할 수 있기를 바란다.

No.	Title
Week 1-10	Gradient Descent Intuition

00'00"

이전 강의에서 여러분은 기울기 하강의 수학적 정의에 대해 배웠다. 이제 이 알고리즘이 어떻게 작동하며 어떤 과정으로 이루어지는지 다루어보자.

00'16"

여기 지난 강의에서 본 기울기 하강알고리즘이 있고, 잠깐 되돌아보면 여기 이 매개변수 α 는 학습 속도라고 불린다. 그리고 이 변수는 j theory 매개변수를 업데이트 할 때 우리가 얼마나 큰 걸음을 딛는지를 조정했다. 이것은 j 변수 가정을 업데이트 할 때 우리가 얼마나 큰 범위를 움직이는지를 결정한다. 그리고 여기 두 번째 항은 미분 계수 항인데, 여기서 설명하고 싶은 것은 이 두 용어가 무엇을 하는지, 그리고 합쳐졌을 때 어떻게 작용하는지이다.

00'50"

이것을 설명하기 위해서 좀 더 간략한 예를 사용할 것이다. 우리가 비용 함수 J 를 가지고 있고, 이 함수에는 앞에서 그랬던 것처럼 실수인 θ_1 한가지 변수만 존재한다. 그러니 우리는 d plot을 가지고 있고, 보기 좀 더 편할 것이다. 기울기 하강이 이 함수에서 어떻게 작용하는지 살펴보도록 하자.

01'20"

여기 함수 $j(\theta_1)$ 이 있다. 물론 θ_1 은 여기서 실수이다. θ_1 의 기울기 하강을 여기 이 지점에 표시하자. 그러면 함수의 이 지점에서 시작해 보자. 기울기 하강은 다음을 업데이트 할 것이다. θ_1 은 이렇게 보이는 식처럼 업데이트 될 것이다. 부수적으로 오른쪽의 이 미분계수 항은 여러분이 왜 이런 기호로 바꿨는지 의아해 할 것이다. 여러분이 이런 편미분계수를 나타내는 기호와 θ 와의 차이를 모른다고 하더라도 걱정할 필요는 없다. 수학에서 여러분은 편미분계수라 부르고,

J 함수의 매개변수의 수에 따라 미분계수라고 부르기도 한다. 이것은 수학적인 개념일 뿐이다.

02'31"

그러니 이 강의의 목표는 이 편미분계수와 θ_1 에 대해 아는 것이다. 그러나 진짜 차이점은 몰라도 된다. 수학적으로 정확한 표기만을 하겠지만 이 두 표기는 정말 같은 것이다. 그러니 이 등식이 무엇을 할지 보도록 하자. 우리는 이 미분계수를 계산할 것인데, 여러분이 미적분학에서 미분계수를 봤는지는 모르겠지만, 여기서 이것이 의미하는 바는, 저 점에서의 탄젠트, 저 빨간 직선은 함수에 닿아 있다. 그리고 이 빨간 직선의 기울기를 보자. 이것이 미분계수고, 함수의 탄젠트는 직선의 경사이다. 직선의 이 경사는 이 높이인데, 이는 이 가로로 놓여있는 것에 의해 나눠져 있다. 이것이 양의 경사이고, 양의 미분계수다. 그리고 내가 업데이트할 θ 는 θ_1 인데 다음과 같이 업데이트할 것이다. α 는 항상 양수이다.

03'45"

그래서 우리는 θ_1 을 θ_1 마이너스 무언가로 업데이트 할 것이다. 이렇게 θ_1 을 왼쪽으로 이 만큼 옮기도록 하겠다. 그리고 θ_1 의 값을 줄이고 우리는 이 방향으로 가고 있기 때문에 이게 맞다는 것을 알 수 있다. 그러면 저 최소값에 다다르게 될 것이다. 그러니 기울기 하강에 대해 지금까지는 제대로 실행하고 있다고 할 수 있다. 다른 예를 보도록 하자. 같은 함수 J 를 사용해서 $J(\theta_1)$ 를 그린다. 그러면 나는 왼쪽 저편에서 변수를 시작해야 될 것이고, 즉 θ_1 은 여기 있다. 그리고 함수의 이 지점을 정한다. 이제 미분계수 항은 $\frac{d}{d\theta_1} \cdot J(\theta_1)$ 이다. 그리고 이 점에 대해서 이 선의 오른쪽의 경사를 보면 이것이 이 미분계수 항이다. 하지만 이 직선은 계속 내려가기 때문에 이것은 음의 경사이다. 또 다르게 우리는 이것은 음의 미분계수라고 할 수도 있고 이는 음의 경사를 의미한다.

04'48"

이것은 0보다 작기 때문에 내가 θ 를 업데이트 하면, θ 를 $-\alpha$ 번 빼는 것으로 업데이트 할 것이다. 그러니 사실은 θ 의 값을 증가시키는 것을 의미한다. 왜냐하면 음수를 빼는 것은 더하는 것과 같기 때문이다. 따라서 θ 의 값은 여기에 도달하지 않을 때까지 계속 증가하는 것을 의미한다. 즉, 최소값이 되게 하려고 이렇게 증가시키는 것이다. 지금까지 미분계수가 하는 일에 대한 직관적인 이론적 설명이었다.

05'34"

이제 학습 속도 α 에 대해 알아보자. 여기 기울기 하강규칙을 업데이트 했다. 만약 α 가 너무 작거나 너무 크면 어떻게 될까? 첫 번째 예에서 α 가 너무 작은 경우를 살펴보자. 여기 함수 $J(\theta)$ 를 보자. 여기서 시작하자. 만약 α 가 너무 작다면 내가 할 일은 작은 숫자들을 곱해서 업데이트하는 것이다. 이렇게 아주 조금 나아간다. 이것이 1단계다. 여기 새로운 점에서 또 다른 단계로 아주 조금 나아간다. 하지만 학습 속도 α 가 너무 작으면 아주 조금씩 나아가서 최소값에 도달하게 될 것이다. 따라서 최소값에 도달하려면 아주 많이 나아가야 한다. 즉 α 값이 너무 작으면 기울기 하강 또한 느리다. 전역 최소점까지 도달하는데 굉장히 많이 나아가야 하므로

오랜 시간이 걸리기 때문이다.

06'47"

이제 α 값이 너무 큰 경우를 보자. 여기 나의 $J(\theta)$ 함수가 있다. 그리고 α 값이 너무 큰 것으로 드러났다. 그러면 기울기 하강은 최소값을 뛰어 넘어버리거나 변환하거나 전환하는데 실패할 것이다. 우리의 데이터가 최소값에 근접한다고 하자. 그러면 미분계수 점은 오른쪽에 있다. 하지만 α 값이 너무 크면, 나는 아주 크게 움직여야 한다. 이렇게 크게 움직여서 나의 비용함수는 이렇게 될 것이다. 이제 미분계수 점의 왼쪽을 보면 데이터를 줄여야 한다는 것을 알 수 있다. 하지만 α 값이 너무 크면 이 점에서 저 점까지 가게 될 것이다. 그리고 저기 있게 된다. α 값이 너무 크면 다음 번에도 그 다음에도 계속 크게 움직이게 될 것이고, 여러분이 이미 최소치를 지나쳤다는 것을 인지할 때까지 계속 될 것이다. 따라서 α 값이 너무 크다면 수렴 뿐만 아니라 발산하는데도 실패할 수 있다.

07'50"

여기 또 다른 문제가 있다. 이것은 까다로운 문제인데 내가 처음 접했을 때도 알아내는데 오래 걸렸다. 여러분의 변수 θ_1 이 국소 최소값이라면 기울기 하강의 첫 단계는 어떻게 진행될까?

Q) 아래 그림과 같이 θ_1 이 $J(\theta_1)$ 의 국소 최소값에 있다고 가정하자. 다음 항의 기울기 하강의 첫 단계는 어떻게 진행될까? 정답: 1번

여기 국소 최소값으로 시작했다고 가정해 보자. 여기서 θ_1 이 시작했다고 하고, 이미 국소 최적값이나 최소치에 도달했다고 가정하자. 여러분의 최적값은 0이 되었다. 이 탄젠트 포인트에서 직선의 기울기는 0이고, 이 미분계수 항은 0이 된다. 여러분의 기울기 하강 업데이트는 다음과 같이 이루어진다. 이것이 의미하는 바는 여러분이 이미 국소 최적값에 있다면, θ_1 의 값이 원래 값 그대로라는 것이다. 왜냐하면 이 업데이트는 $\theta_1 = \theta_1$ 이기 때문이다. 따라서 여러분의 매개변수가 이미 국소 최소값에 있다면, 기울기 하강을 한 단계 실행하는 것은 아무런 의미가 없는 것이다. 여러분의 매개변수를 바꾸지 못하기 때문이다. 하지만 이것은 우리의 해를 국소 최적값에 머물도록 하기 때문에 여러분이 바라는 바이다.

09'06"

이것은 기울기 하강이 국소 최소값으로 수렴하는 이유 또한 설명해준다. 학습 속도 α 값이 고정되어 있더라 말이다. 이것이 의미하는 바는 다음 예를 보자. 여기 비용함수 $J(\theta_1)$ 이 있고, 우리는 이것을 최소화하려 한다. 기울기 하강 알고리즘을 여기 자주색 점에서 시작해 보자. 여기서 기울기 하강을 한 단계 실행하면 아마 이 점으로 옮겨갈 것이다. 왜냐하면 미분계수가 꽤 가파르기 때문이다. 현재 이 녹색 점에서 또 한번 기울기 하강을 하게 된다면, 경사가 자주색 점에서의 경사보다 덜 가파른 것을 알 수 있다. 왜냐하면 내가 최소값에 다다르고 있기 때문에 이 미분계수는 0에 계속 가까워지고 있다. 한 번 더 시행하면 미분계수는 더 작아질 것이다.

그래서 한 번 더 실행하면 여기 녹색 점에서 나아가는 폭은 자주색 점에서 나갈 때보다 조금 더 작게 된다. 이제 이 새로운 빨간 점에서는 전역 최소값에 더 가까워지기 때문에 여기서의 미분계수는 초록 점에서보다 더 작아질 것이다. 또 한 번 실행하면 이제 이 미분계수는 더욱 작아져서 θ_1 에 대한 업데이트 강도는 더욱 작아지게 되어 이렇게 아주 조금 나아가게 된다. 이렇게 진행됨으로써 여러분은 계속해서 더 작은 움직임을 나타내게 될 것이다. 이렇게 기울기 하강이 실행되면서 여러분은 자동적으로 점점 더 조금씩 이동하게 된다. 그리고 마침내 국소 최소값에 수렴하게 된다.

10'46"

요약하면, 기울기 하강에서, 우리가 국소 최소값에 다가갈 때 기울기 하강은 자동적으로 더욱 작게 나아간다. 이는 우리가 국소 최소값으로 나아갈 때, 여기서 국소 최소값은 미분계수가 0일 때를 말한다, 이 미분계수 항은 자동적으로 계속해서 작아지고, 기울기 하강은 더 작게 진행된다. 바로 이 그래프가 기울기 하강의 모습을 보여준다. 이는 서서히 α 값을 감소시킬 필요가 없다는 뜻이다.

11'22"

이것이 비용함수 J 를 줄이는데 사용할 수 있는 기울기 하강 알고리즘이다. 여기서 이 비용함수는 우리가 앞서 배웠던 선형 회귀에서 정의한 비용함수는 아니다. 다음 강의에서 이 두 가지를 살펴보고 선형 회귀에서 보았던 비용함수로 돌아갈 것이다. 그리고 기울기 하강과 이 비용함수를 합쳐 볼 것이다. 그러면 우리가 처음 배운 알고리즘인 선형 회귀 알고리즘이 나올 것이다.

No.	Title
Week 1-11	Gradient Descent For Linear Regression

00'00"

지난 시간에는 기울기 하강 알고리즘, 선형 회귀 모델과 제곱 오차 비용 함수에 대해서 배웠다. 이번 시간에는 비용함수와 함께 기울기 하강을 다룰 것이고, 선형회귀함수를 위한 알고리즘이나, 우리의 데이터에 직선을 나타낼 수 있을 것이다. 이것이 우리가 앞서 배웠던 내용들이다. 이 기울기 하강 알고리즘에 대해서 여러분은 이제 좀 익숙할 것이다. 이것은 우리의 선형 가정함수와 제곱 오차 비용함수에 대한 선형 회귀 모델이다. 우리가 배울 내용은 제곱 오차 비용 함수를 최소화하기 위해 기울기 하강을 어떻게 사용하는 것이다.

00'39"

기울기 하강을 사용하기 위해서 이런 코드를 써야 할 것이다. 우리에게 필요한 핵심은 미분 계수 개념을 적용하는 것이다. 따라서 우리는 편미분계수를 이해하고 비용함수 j 의 정의에 적용하면 이렇게 된다 내가 여기서 한 일은 여기에 비용함수 J 의 정의를 대입한 것이다. 좀 간소화 해보면 다음과 같다. 시그마 i 는 1에서부터 m 까지, $\theta_0 + \theta_1 x^{(i)} - y^{(i)}$ 의 제곱이다. 내가 세운 가설의

정의를 저기 적용한 것이다.

01'50"

그리고 우리는 J 가 0일 경우와 1일 두 가지 경우의 partial 미분계수를 구해야 한다. 그러니, θ_0, θ_1 일 경우의 다음 편미분계수를 알아내야 하는 것이다. 답을 적어보도록 하겠다. 이 첫 번째 항은 $1/m$ 식으로 간소화 될 것이다. 그리고 두 번째 항은 다음과 같이 도출된다. 이 편미분계수들을 계산하면 다음 공식, 즉 여기 위의 공식을 아래의 두 공식으로 내려가서 계산한다. 이 계산과정은 다변수의 미적분을 필요로 한다. 여러분이 미적분을 알고 있다면 여기에 적용해보고, 이것과 같은 답이 나오는지 비교해보자.

03'01"

하지만 미적분을 잘 모른다 하더라도 크게 상관없다. 왜냐하면 이 공식을 사용하고 그 외의 미적분은 우리가 앞으로 활용하는데 사용되지 않을 것이기 때문이다. 그러니 기울기 하강을 사용해서 다시 해보도록 하자. 우리가 공부한 이러한 정의를 활용해서, 즉 비용 함수 J 의 기울기에 해당하는 미분계수를 사용해서, 이 값을 다시 기울기 하강 알고리즘에 적용해 보자. 이것은 선형 회귀의 기울기 하강이고, 이는 알다시피, 수렴될까지 반복될 것이다. 그리고 θ_0 값과 θ_1 값은 계속 $-\alpha$ 곱하기 미분계수를 뺀 만큼 업데이트 될 것이다. 여기 이 항은 우리의 선형 회귀 알고리즘이다. 여기 첫 항은 θ_0 의 편미분계수로 앞서 공부했었다. 두 번째 항은 θ_1 의 편미분계수이다.

04'12"

그리고 잠깐 상기해보면, 기울기 하강을 시행할 때 여러분은 θ_0 값과 θ_1 값을 동시 업데이트 해야 한다. 이제 기울기 하강이 어떻게 작동하는지 살펴보자. 앞서 배운 것은 기울기 하강은 국소 최적값에 아주 민감하게 반응한다는 것이었다. 이를 설명할 때 먼저 다음의 그래프를 보여주었다. 언덕 표면에서 아래로 내려갈 때, 우리가 어디서 시작하는지에 따라 다른 국지 최적값에 다다르는 것을 알 수 있었다. 여러분은 여기서도 저기서도 시작할 수 있다. 하지만 선형 회귀의 비용함수는 항상 이런 궁형 함수이다. 이것은 Convex 함수라고 불린다.

05'07"

여기서는 Convex 함수의 공식적 정의를 내릴 것은 아니다. 하지만 일반적으로 Convex 함수는 우뚝한 그릇 모양의 함수를 의미하고, 이 함수는 아무런 국지 최적값을 갖지 않는다. 오직 최적 해만을 가질 뿐이다. 이 비용함수의 기울기 하강은 여러분이 선형 회귀함수를 사용할 때 항상 전역 최적값으로 수렴한다. 다른 국소 최적값이 없고 오직 전역 최적값만 존재하기 때문이다. 이제 이 알고리즘을 표현해 보자. 늘 그래왔던 것처럼 가정 함수와 비용함수 j 를 그려보자. 이 지점을 매개변수로 지정했다고 하자. 예를 들어, 여러분은 대체로 매개 변수를 0,0으로 지정한다고 하자. θ_0 는 0인 것이다. 하지만 여기 그림에서는 θ_0 를 900으로 두고 θ_1 은 -0.1으로 두어도 괜찮다. 그리고 이 선은 $h(x) = -900 - 0.1x$ 에 대응한다. 이 선, 여기 비용함수에서 말이다.

06'19"

이제 기울기 하강을 한 단계 실행해보면 우리는 여기 이 점에서 왼쪽 아래로 이동하게 되고 이 두 번째 점으로 오게 된다. 여러분은 이 직선의 변화를 인지했을 것이다. 그리고 또 한번 움직임으로써 왼쪽의 직선에 변화가 있을 것이다. 나의 비용함수의 점도 새로운 곳으로 움직인다. 내가 계속 이 알고리즘을 사용하면 우리는 비용함수 밑으로 내려갈 것이다. 따라서 우리의 변수들은 이런 궤적을 따라 움직인다. 왼쪽을 보면 이것은 우리의 가정에 대응한다. 즉, 우리가 전역 최소값에 도달할 때까지, 그리고 이 최소값이 이 가정에 대응될 때까지 계속 데이터에 맞춰져 갈 것이다. 이것이 바로 기울기 하강이다. 우리는 방금 이 알고리즘을 실행했고 주택 가격에 대한 나의 데이터와 잘 맞아떨어지게 되었다. 여러분은 이제 이를 가지고 예측을 할 수 있는데, 만약 여러분의 친구가 1250 평방피트의 집을 가지고 있다면 이제 그래프에서 값을 읽어낼 수 있고, 잘 모르지만, 예를 들어 25만 달러의 가치가 있다고 할 수 있다.

07'49"

이를 다른 이름으로 부르면, 우리가 방금 학습했던 알고리즘은 때로 "batch" 기울기 하강이라고 불리기도 한다. 그리고 이는 기계 학습에서는, 나는 잘 모르겠지만, 사람들이 알고리즘에 이름을 잘 붙이지 못하는 것 같다. 이 용어 batch 기울기 하강은 기울기 하강의 모든 단계, 그러니까 우리가 트레이닝 예시를 살펴보는 것을 뜻한다. 따라서 기울기 하강에서 미분계수를 계산할 때, 우리는 여기 이 합을 계산하는 것이다. 그러니 이 기울기 하강의 모든 단계에서 다음과 같은 m트레이닝 예의 합을 계산하게 된다. 이는 batch 기울기 하강을 실행하면 모든 트레이닝 예를 볼 수 있다는 것을 의미한다.

08'35"

이는 그다지 좋은 이름은 아닌 것 같지만, 기계 학습을 하는 사람들이 사용하는 이름이다. 그리고 batch 버전이 아닌 다른 버전의 기울기 하강도 존재한다. 하지만 이 기울기 하강들에서는 전체 예시를 보는 것이 아니라 한번에 더 작은 예시의 부분 집합을 보는 것이다. 이 버전들은 나중에 배우게 될 것이다. 지금은 이 batch 알고리즘을 사용해서 어떻게 선형 회귀를 위한 기울기 하강을 시행하는지를 배웠다.

Q) 다음 중 참인 진술을 골라라. 정답: 3,4 번

이것이 기울기 하강을 활용한 선형 회귀이다. 여러분이 진보된 선형대수를 예전에 본 적이 있다면, 여러분은 이에 대한 강의를 들었던 것이다. 여러분은 비용함수 J 를 기울기 하강과 같은 알고리즘을 반복적으로 사용해서 해를 구하지 않아도 수학적으로 비용함수 J 의 최소값을 구하는 방법이 있다는 사실을 알고 있다. 이에 대해서는 강의 후반부에 다루겠다.

09'20"

그 다른 방법은 정규방정식이다. 여러분이 이것에 대해 들어본 경우에는 기울기 하강이 이 방법보다 더 큰 데이터에 대해서 더 잘 조정한다는 사실을 알 것이다. 지금까지 기울기 하강이

무엇이고 서로 다른 많은 경우에 사용될 수 있다는 사실을 배웠으므로 많은 분야에서 활용할 것이다. 첫 기계 학습 알고리즘을 학습한 것을 축하한다. 이것을 어떻게 활용하는지는 연습 문제를 통해서 다지기를 바란다. 그리고 여러분에 알맞게 사용하게 되길 바란다. 다음 강의에서는 기울기 하강 알고리즘을 더욱 강력하게 만들어 줄 일반화에 대해 배울 것이다.

No.	Title
Week 1-12	Matrices and Vectors

00'00"

이번 강의를 시작하기 전에 선형 대수학에 대해 한 번 짚고 넘어가자. 이번 강의에서는 행렬과 벡터에 대해 설명할 것이다. 행렬은 사각형 괄호 안에 쓰인 숫자의 직사각형 배열이다. 예를 들어보자. 먼저 왼쪽에 사각형 괄호를 그리고, 그 안에 숫자를 적어준다. 이 숫자들은 학습 문제의 feature 또는 어디선가 얻게 된 데이터일 수도 있다. 하지만 특정 값은 중요치 않으니 오른쪽에도 괄호를 그려준다. 이게 하나의 행렬이다. 여기 행렬 예시 하나가 더 있다. 1, 2, 3, 4, 5, 6 을 써보자. 행렬은 2D, 2차원 배열이라고 보면 된다.

00'56"

또 알아야 할 내용은 행렬의 크기는 행의 수 x 열의 수로 표현된다. 그러므로 왼쪽 예시는 4 열과 2 행이 있으므로, 행의 개수와 열의 개수를 곱해, 4×2 행렬이다. 4행 2열이다. 오른쪽 행렬은 행이 2개다. 이게 1행이고 이게 2행이다. 그리고 3열이 있다. 이게 1열, 2열, 3열이다. 따라서 두 번째 행렬은 2×3 행렬이다. 따라서 이 행렬의 크기는 2×3 이다. 때로는 이러한 행렬이 좌측 행렬의 경우 $R^{4 \times 2}$ 로 표현되거나, 이 행렬을 보고 세트 $R^{4 \times 2}$ 이라고 말하는 사람도 있다. 이는 그저 모든 행렬 세트의 크기가 4×2 라는 뜻이다. 오른쪽 예시는 비슷하게 $R^{2 \times 3}$ 으로 쓸 수도 있다. 그러므로 앞으로 $R^{4 \times 2}, R^{2 \times 3}$ 와 같은 형태를 본다면 이는 행렬 4×2 , 또는 행렬 2×3 과 같다라는 것을 알아두면 된다.

Q) 다음 중 옳은 것을 모두 고르시오.

(정답 1, 3, 4번)

02'29"

이번에는 행렬의 특정 원소를 가리키는 법을 이야기해보자. 행렬에서 말하는 행렬 원소는 항목을 의미하는데, 바로 행렬 안에 있는 숫자를 의미한다. 표준 표기법은 A행렬에서 A옆에 ij는 ij

항목을 일컫는데, 이는 i번째 행과 j번째 열에 있는 원소이라는 뜻이다. 예를 들면, a11은 첫 번째 행과 첫 번째 열의 원소를 가리키므로, a11은 1402를 말한다. a12는 첫 번째 행의 두 번째 열이므로 191로 나타낼 수 있다. 좀 더 빠르게 몇 개 살펴보면, a32는 세 번째 행, 두 번째 열이므로 1437이 된다. 마지막으로 a41은 4번째 행 1번째 열이므로 147이 된다. 만약 여러분이 a43을 찾거나 말하고자 한다면, 이는 4번째 행, 3번째 열이므로 이 행렬에서는 3번째 열이 없다. 그러므로 이는 오류라고 말할 수 있다. a43 원소는 없으므로 이를 나타낼 수 없다.

04'17"

행렬은 빠르게 다양한 데이터를 정리, 지수화 그리고 접근할 수 있게 해준다. 강의 중에 꽤 많은 개념을 빠르게 설명하는 것 같은데, 이 모든 용어를 다 외울 필요는 없다. 하지만 강의 웹사이트에 강의 노트를 올려놨으니 거기에 나온 정의를 참고하면 된다. 따라서 강의 슬라이드와 주어진 과제를 나중에 다시 찾아볼 수 있다. 그러므로 강의를 듣다가 잘 잊어버린다면 'a41이라고? 무슨 행 무슨 열?' 지금 당장은 모든 것을 외우려고 하지 않아도 된다. 강의 사이트에 올라와 있는 자료를 참고할 수 있다. 지금까지 행렬에 대해 알아봤다. 이제 벡터로 넘어가보자.

04'57"

벡터는 행렬의 특수한 형태라고 볼 수 있다. 벡터는 행렬 중에서도 열이 하나뿐인 $N \times 1$ 형태의 행렬이다. 이건 기억해야 한다. N은 행의 개수이고, 1은 열의 개수인데, 그러므로 열이 한 개인 행렬을 벡터라고 부른다. 여기 예시를 보면, 여기서 N은 원소가 4개이다. 그러므로 이 행렬은 또한 4차원 벡터라고 부를 수 있는데, 이는 원소가 4개인, 숫자 4개가 들어있는 벡터라는 뜻이다. 그리고 아까 배운 행렬에서 R^{2x3} 은 행렬 2x3을 나타내는데, 이것을 R^4 로 나타낼 것이다. R^4 는 4차원 벡터 세트를 의미한다.

05'57"

이제 벡터의 원소를 표시하는 법을 알아보자. $y_i = y^{th}$ 원소를 말한다. 그래서 y 는 벡터를 말하고 i 는 i 번째 원소를 말한다. y_1 은 첫 번째 원소인 460, y_2 는 두 번째 원소 232이다.

Q) 다음과 같이 A행렬이 있다. A32는 다음 행렬의 원소이다. A32 값은?

(정답 4번)

여기 첫 번째, 두 번째가 있고, $y_3 = 315$ 가 된다. y_1 부터 y_4 까지 4차원 벡터로 정의된다. 또한, 벡터에 지수화하는 두 가지 방법을 소개해주겠다. 1 인덱스를 보통 사용하는데, 0 인덱스도 사용되기도 한다. 왼쪽 예시에서는 y_1, y_2, y_3, y_4 원소로 가지고 있다. 오른쪽 예시는 0 인덱스의 예시로 원소를 0부터 시작했다. 그래서 원소가 y_0 부터 y_3 까지 있다. 이는 프로그래밍 언어의 배열과 흡사하다. 그래서 여기서는 1로 시작하여, 배열의 첫 번째 원소는 y_1 이고, 이러한 숫자 배열 표기는 어떤 프로그래밍 언어를 사용하는 가에 따라 때로는 이와 같이 0 인덱스로 시작할 수도 있다. 따라서 대부분의 수학에서 인덱스 1 버전이 기계학습 응용에서 더 흔하게 사용되고, 인덱스 0 벡터는 좀 더 편리한 표기로 활용된다. 그러므로 따로 명시되어있지 않는 한, 인덱스 1

벡터를 사용한다고 생각해야 한다. 사실 선형 대수학 리뷰 강의에서도 인덱스 1벡터를 사용할 것이다. 하지만 기계 학습 응용에서 때로는 필요할 경우 인덱스 0 벡터 또한 사용할 수도 있다.

08'00"

마지막으로 관례상 행렬과 벡터를 쓸 때, 대부분 행렬은 대문자 알파벳을 사용한다. 그래서 A, B, C, X와 같은 대문자는 행렬을 가리키고, a, b, x, y와 같은 소문자는 숫자나 스칼라, 또는 벡터에서 활용된다. 이러한 룰이 항상 적용되는 것은 아니지만 대부분 y는 벡터를 의미하고, 대문자는 행렬의 의미한다. 지금까지 행렬과 벡터에 대해 배워봤다. 다음 시간에는 이를 활용하는 것을 배워보자.

No.	Title
Week 1-13	Addition and Scalar Multiplication

00'00"

이번 강의에서는 행렬의 덧셈과 뺄셈 그리고 스칼라 곱이라고 하는 행렬과 상수 간의 곱셈에 대해 이야기 하겠다. 예제를 하나 보자. 이와 같은 두 개의 행렬을 더하려면 어떻게 해야 할까? 그리고, 행렬 간의 덧셈의 의미가 무엇일까? 두 개의 행렬을 서로 더하려면 단순히 각 행렬 내의 원소들을 더해주면 된다. 따라서, 두 행렬의 합은 다른 행렬의 같은 위치의 있는 것을 더하면 된다. 첫 번째 원소를 구하려면 각각의 행렬에서 1과 4를 가져오고 이를 각각 더하여 5를 얻을 수 있다. 두 번째 원소는 2와 2를 각각 가져올 수 있고 이를 서로 더하여 4가 되고 $3 + 0 = 3$. 이런 식으로 원소들을 서로 더하면 된다. 이제 색깔을 그만 바꾸겠다. 그리고 오른쪽 열은 0.5, 10, 2 이다. 그리고 여러분은 같은 차원인 두 행렬만 더할 수 있다는 것을 알았다. 이 예제는 3행 2열의 행렬 덧셈이다. 왜냐하면 이 행렬은 3개의 행과 2개의 열을 가지고 있다. 그러니까 3×2 이것 역시 3행 2열의 행렬이고 이 두 행렬합의 결과 역시 3행 2열의 행렬이다. 그래서 오직 같은 차원의 행렬들만 더할 수 있고, 그 결과는 여러분이 더한 것처럼 같은 차원을 가진 또 다른 행렬이 된다.

01'28"

반면에, 여러분이 이 두 행렬을 택한다면, 즉 하나는 3×2 행렬, 여기 이건 2×2 행렬. 이 두 행렬은 같은 차원이 아니기 때문에, 이건 잘못됐다. 그래서 이 두 행렬은 더할 수가 없다. 이들의 합은 제대로 정의되어있지 않다.

Q) 다음 행렬을 연산하여라.

(정답 2번)

이것이 바로 행렬의 덧셈이다. 다음으로, 행렬의 스칼라 곱에 대해서 이야기 해보자. 스칼라는

단지, 어쩌면 너무 화려한, 숫자, 또는 실수를 가리키는 용어입니다. 자, 스칼라는 실수라는 뜻이다. 이제 숫자 3을 가지고 이 행렬에 곱해보다. 그렇게 하면, 결과는 여러분이 상상하는 것과 꽤 비슷할 것이다. 행렬의 행렬 요소를 가지고, 한번에 하나씩 3과 곱하면 된다. 그래서, $1 \times 3 = 3$. $2 \times 3 = 6$, $3 \times 3 = 9$, 다시 색깔 바꾸는걸 그만하겠다. $0 \times 3 = 0$. $3 \times 5 = 15$, $3 \times 1 = 3$. 그래서 이 행렬은 왼쪽의 행렬에 3을 곱한 결과다. 그리고, 다시, 이 행렬은 3×2 이고 결과 행렬도 같은 차원의 행렬인 것을 알아차렸을 것이다. 이것은 3행 2열, 두 행렬 모두 3×2 차원인 행렬이다. 그리고, 스칼라 곱은 어느 쪽으로도 쓸 수 있다. 그래서, 3 곱하기 행렬이 있고 이것을 이렇게 쓸 수 도 있다. 0, 2, 5, 3, 1 이 행렬을 그냥 오른쪽에 복사했다. 이 행렬에 3을 곱할 수도 있습니다. 그래서 $3 \times$ 행렬 또는 행렬 $\times 3$ 은 같은 것이고 가운데에 있는 것이 결과 행렬이다.

01'28"

행렬에 숫자로 나누기 또한 할 수 있다. 그래서 이 행렬을 4로 나눈 것이 사실 숫자 4분의 1을 이 행렬에 곱한 것과 같은 것임을 알 수 있다. 4, 0, 6, 3 그리고 여러분은 답을 알 수 있다. 이 곱의 결과는, $1/4 \times 4 = 1$, $1/4 \times 0 = 0$, $1/4 \times 6 = 3/2$, $6/4$ 는 $3/2$ 이고, $1/4 \times 3 = 3/4$ 이다. 그래서 이 행렬을 4로 나눈 것을 연산한 값이 이것이다. 벡터들이 결과를 준다.

Q) 다음을 연산하면?

(정답 1번)

04'02"

마지막으로, 조금 더 복잡한 예제로 여러분은 이 연산들을 조합할 수 있다. 이 연산에서 3 곱하기 벡터 더하기 벡터 빼기 3으로 나눈 또 다른 벡터가 있다. 이것들이 어떤 것인지 확인해보자. 이 곱셈은 이것은 스칼라 곱의 예시다. 3을 곱하고 있기 때문이다. 그리고 이것은 또 다른 스칼라 곱셈이다. 스칼라 나누기에 더 가깝다고 할 수 있다. 이건 그냥 $1/3$ 곱하기 행렬과 같다. 그래서 이 두 연산자를 먼저 연산한다면 이것과 같습니다. 그래서 3 곱하기 이 벡터는 3, 12, 6 더하기 가운데 있는 벡터 0 0 5 빼기 0, 1, 2/3. 다시 연산이 어떻게 이루어지는 이해하기 위해서 살펴보면, 여기 더하기 기호는 행렬 덧셈이다. 이것들은 벡터이기 때문에 벡터는 행렬의 특별한 경우로 기억하고 있다. 이건 역시 벡터 덧셈이라고 부를 수 있다. 여기 빼기 기호는 행렬 뺄셈이다. 그런데 이것은 n 나누기 1이고, 실제로 3 나누기 행렬이다. 이것은 실제로 벡터이고 이 열 또한 벡터이다. 이 행렬을 벡터 뺄셈이라고도 부른다. 마지막으로 마무리하기 위해 이 연산은 벡터를 결과로 준다. 첫 번째 원소는 $3 + 0 - 1$ 이 될 것이고, 즉 3-1이고 이것은 2다. 두 번째 원소는 $12 + 0 - 0$, 즉 12다. 그리고 세 번째 원소는 $6 + 5 - (2/3)$, 즉 $11 - (2/3)$ 다. 10과 $1/3$ 이다. 대괄호를 닫는다. 그래서 이 연산은 3×1 행렬을 준다. 또는 3차원 벡터라고도 불리는, 연산의 결과가 여기 있다. 그래서 이것이 행렬, 벡터를 더하고 빼고 스칼라나 행의 수에 곱하는 방법이다.

Q) 다음을 연산하시오

(정답 3번)

06'41"

지금까지 어떻게 행렬과 벡터를 스칼라, 숫자 열과 곱하는 법에 대해서 이야기했다. 다음 강의에서 우리는 두 행렬을 가지고 두 행렬끼리 곱하는 더 흥미로운 과정에 대해서 이야기해 보겠다

No.	Title
Week 1-14	Matrix Vector Multiplication
개요	
Machine Learning Specialization 1-14 번역본	

00'00"

이번 강의에서는 두 행렬을 곱셈하는 방법에 대해 이야기해보겠다. 먼저 특수한 케이스인 행렬을 벡터와 곱하는 행렬 벡터 곱셈을 다뤄보겠다. 여기 예시가 있다. 여기 행렬과 벡터가 있다. 여기 행렬과 벡터 이 둘을 곱하고 싶다고 한다면, 결과는 어떻게 나올까? 먼저 예시를 연산하는 과정을 보여주고 난 후, 다시 돌아와 설명해주겠다. 이 곱셈 과정의 결과는 벡터 그 자신이 된다. 이를 먼저 연산한 후에, 다시 돌아와서 설명을 해주겠다. 이 벡터에서 첫 번째 원소를 구하기 위해, 먼저 여기 숫자 두 개를 가지고 행렬 첫 번째 행과 곱해서 이 두 숫자를 더한다. 그러면 $1 \times 1 + 3 \times 5$ 이므로, $1 + 15 = 16$ 이 된다. 여기서 16을 쓴다. 두 번째 원소를 구하기 위해 두 번째 행과 이 벡터를 곱해준다. $4 \times 1 + 0 \times 5 = 4$ 가 된다. 여기에 4를 써준다. 마지막으로 $2 \times 1 + 1 \times 5 = 7$ 이므로 7을 써준다. 결과적으로 3×2 행렬과 2×1 행렬의 곱셈이, 이 것도 2차원 벡터인데 이 결과가 여기에서 3×1 행렬이 나와 결국 3×1 행렬이 나왔고 이는 3차원 벡터라고도 할 수 있다. 좀 빠르게 진행되어서 이것을 똑같이 연산할 수 있을지 확신이 서지 않을 수 있지만, 방금 연산 과정을 다시 한번 살펴보면서 행렬과 벡터의 곱셈 과정을 돌이켜보자.

02'26"

여기에 행렬과 벡터의 곱셈 과정이 자세히 나와있다. 여기 행렬 A가 있고, 벡터 x와 곱하려고 한다. 결과는 벡터 y가 될 것이다. 행렬 A는 $m \times n$ 행렬로 m 행과 n 열이 있는데, 이를 $n \times 1$ 행렬과 곱하려고 한다. 여기서 $nx1$ 은 n차원 벡터다. 여기 n이 여기 있는 n과 숫자가 같아야 한다. 다시 말해, 첫 번째 행렬에서 열의 개수가 두 번째 행렬의 행의 개수인 것이다. 여기 열의 개수와 여기 행의 개수가 같아야 한다는 것이다. 이 벡터의 차원 수와 같아야 한다. 연산의 결과는 m차원 벡터 y가 된다. 다시 말해, 여기서 행의 수와 A행렬의 행의 수인 m과 같다는 것이다. 그렇다면 이 벡터 y를 어떻게 연산할 것인가? 벡터 y 값을 얻기 위해서, y_i 값을 구하기 위해서 먼저 A의 i번째 행렬 원소와 벡터 x의 원소를 곱해서 더하면 된다. 다시 말해, 벡터 Y의 첫 번째

원소를 구하려면, 첫 번째 숫자가, 어떤 것이 주어지든, 행렬 A의 첫 번째 행에서 가져와 이를 벡터 x의 원소와 하나씩 차례로 곱하라는 것이다. 그러니까 이 행렬에서 첫 번째 원소와 이 행렬에서 첫 번째 원소를 곱하고, 여기 두 번째 원소와 여기 두 번째 원소를 곱한다. 마찬가지로 세 번째로 이렇게 곱하고, 이를 끝까지 반복한다. 그리고 나온 결과를 모두 더하여 이 값이 벡터 y 의 첫 번째 원소가 된다. 그런 다음, 벡터 y 의 두 번째 원소를 구하기 위해서는 이렇게 표시하고, 행렬 A의 두 번째 행을 가져와 아까와 같이 반복하면 된다. 행렬 A의 두 번째 행을 가져와 이를 x 원소와 곱해서 그 값을 모두 더하면 벡터 y 의 두 번째 원소를 구하게 된다. 이와 같은 과정을 반복해서 행렬 A의 세 번째 행을 가져와서 벡터 x 원소와 곱해서 그 값을 모두 더하면 벡터 y 의 세 번째 원소를 얻게 되고 마지막 값을 구할 때까지 반복해주면 된다. 이게 바로 연산 과정이다. 예시를 하나 더 살펴보자.

05'00"

또 다른 예시가 있다. 먼저 몇 차원인지 살펴보자. 3×4 행렬이 있고, 여기는 4차원 벡터, 즉 4×1 행렬이 있다.

Q) 여기 두 행렬이 있다. 이 두 행렬을 연산하면 몇 차원 행렬이 나오는가?

(정답 1번)

이 연산의 결과는 3차원 벡터가 될 것이다. 이는 3개의 원소를 가진 벡터다. 그럼 이제 연산을 시작해보자. 첫 번째 원소는 여기 숫자 네 개를 가져가 이를 벡터 x와 곱한다. 그래서 $1 \times 1 + 2 \times 3 + 1 \times 2 + 5 \times 1$ 를 하면 이는 $1 + 6 + 2 + 5 = 14$ 가 된다. 두 번째 원소는 이 열을 가져가 여기 벡터와 곱해주면 $0 \times 1 + 3 \times 3 + 0 \times 2 + 4 \times 1$ 는 $9 + 4 = 13$ 이 된다. 마지막 원소는 이 행을 가져가 $-1 \times 1 + (-2) \times 3 + 0 \times 2 + 0 \times 1 = -1 - 6 = -7$ 이 되므로 벡터 7이 된다. 그러므로 최종 결과는 14, 13, -7 이다. 그리고 예상했던 대로, 3×1 행렬이다. 이게 바로 행렬과 벡터의 곱셈이다. 현재 슬라이드에서 다양한 연산이 이루어졌고, 이 숫자가 어떻게 나온 것인지 잘 이해가 되지 않는다면 영상을 잠시 멈춘 후에 우리가 방금 했던 연산을 천천히 살펴보면서 이 슬라이드에서 배운 연산 과정을 통해 14, 13, -7이 나왔다는 것을 잘 이해하고 넘어갈 수 있도록 한다.

Q) 다음을 연산하시오.

정답 3번

07'30"

여기서는 숨겨진 비밀을 공개하려고 한다. 여기에 집 4 채의 집합이 있는데, 각기 다른 사이즈다. 이 집이 가격을 예측하는 가정이 있고, 이를 연산하고자 한다. 그래서 4 채의 집의 $h\theta(x)$ 값을 구하려고 한다. 이 가정을 적용하면 내 모든 집을 한 번에 연산하는 간단한 방법이 있다. 이를 행렬 벡터 곱셈에서도 적용이 가능하다. 이제 그 과정을 보여주겠다. 다음과 같이 행렬을 만들어보겠다. 1, 1, 1, 1 을 적고 그 옆에 주택 사이즈를 적는다. 벡터도 적어볼 텐데, 원소가 -40,

0.25 이렇게 두 개다. 이는 아까 위의 식의 계수다. 데이터 0과 데이터 1이다. 이제 행렬과 벡터를 가지고 곱해서 여기 이건 곱셈 기호다. 그렇다면 결과값은? 이것은 4×2 행렬이고, 이건 2×1 행렬이다. 그러므로 결과는 4×1 벡터가 될 것이다. 그렇다면 이제 4×1 행렬, 4차원 벡터 결과를 얻게 될 것이다. 이제 4가지 원소의 실제 숫자값을 적어보자. 첫 번째 원소의 값은, 여기서 사용한 연산법은, 이것을 가져가서 벡터와 곱해준다. 따라서 $-40 \times 1 + 4.25 \times 2104$ 가 된다. 아까 전 슬라이드에서는 $1 \times -40, + 2104 \times 0.25$ 이런 순서로 썼지만, 순서는 바뀌어도 관계가 없다는 것을 알 것이다. -40×1 와 1×-40 은 같기 때문이다. 여기 첫 번째 원소는 $h\theta(2104)$ 가 되는데 이는 내 첫 번째 주택의 예상가격이다. 두 번째 원소로 넘어가자. 이제 어떻게 하는지 감이 오기를 바란다. 여기 두 숫자를 가져가 벡터와 곱해준다. $-40 \times 1 + 0.25 \times 1416$ 가 되고, $h\theta(1416)$ 이다. 그렇게 세 번째, 네 번째 원소도 구하면 이는 4×1 벡터가 된다. 다시 한번 말하지만, 방금 초록색 박스로 표시한 부분은 실제 숫자이다. 하나의 숫자이고, 여기 자홍색, 자주색 박스로 표시한 부분도 실제 숫자다. 그러니까 오른쪽에 있는 것은 4×1 차원 행렬이자, 4차원 벡터이다.

10'59"

여기에 숨겨진 사실은 이와 같은 과정을 소프트웨어에서 실행할 때는, 집이 네 채가 있고, 가정을 통해 가격을, 이 네 채의 집의 가격 y 를 예측하려고 한다. 이는 무슨 뜻이냐면, 수식 한 줄로 정리할 수 있다. 나중에 옥타브와 프로그램 언어를 다루게 되면, 그때는 이를 한 줄로 정리할 수 있을 것이다. 예상값 = 데이터 행렬 \times 매개 변수 라는 것이다. 그러니까 데이터 행렬은 이것을 말하고, 매개 변수는 이것을 말한다. 여기서 곱셈은 행렬 벡터 곱셈을 의미한다. 이를 연산하면 변수 예상값이 나오는데, 이 수식을 실행하면 행렬 벡터 곱셈을 다룰 줄 안다는 것이다. 이것만 연산하면, 예상값은 여기 오른쪽의 4×1 차원 벡터가 되고, 모든 예상 가격을 알려줄 것이다.

12'05"

행렬 벡터 곱셈의 또 다른 방법은 이렇게 $i = 1 : 4$ 쓸 수 있다. 만약 집이 천 개라면 $i = 1 : 1000$ 으로 쓸 수 있다. 그런 다음 $\text{prediction}(i) : \dots$ 이렇게 많이 적어야 한다. 그러므로 주택의 수가 많아지면, 예상하는 가격 또한 4개가 아니라 천 개의 집을 연산해야 하는데, 이를 컴퓨터에서 실행하면, 이와 같은 것을 실행하면, 어떠한 다양한 컴퓨터 언어에서도, 단지 옥타브나 수프라서버, 자바, 파이썬 또는 다른 고차원 언어에서도 마찬가지다. 왼쪽의 이와 같은 형태로 코드를 쓰면 코드를 단순화 시켜주는 것뿐 아니라, 그 안에 많은 형태가 들어있는 것보다 간단하게 코드 한 줄만 입력하는 것이다. 그런 이유로 나중에도 살펴보겠지만, 집 값의 결과를 예측하는데 왼쪽처럼 하면 오른쪽이나 여러분의 공식을 적는 것보다 훨씬 더 효과적으로 연산할 수 있게 된다.

13'13"

나중에 벡터화 할 때 더 설명하도록 하겠다. 그러니까 이러한 방법으로 예측하는 것은 간단한 코드를 얻는 것 분 아니라, 더 효과적이게 된다. 이것이 바로 행렬 벡터 곱셈이다. 이를 활용하는 과정은 이후에 다른 모델에서 `living` 회귀를 더 알아보면서 진행하도록 하자. 다음 번 강의에서는

이를 바탕으로 행렬과 행렬 곱셈에서 일반화하겠다.

No.	Title
Week 1-15	Matrix Matrix Multiplication

00'00"

이번 강의에서는 행렬과 행렬 간의 곱셈, 그러니까 두 행렬을 곱하는 방법에 대해 다뤄보겠다. 선형 회귀에서 매개 변수 θ_0 과 θ_1 을 기울기 감소와 같은 쌍방향 알고리즘 없이도 한번에 연산하는 방법을 이야기할 때, 그러한 알고리즘에 대해 다룰 때, 행렬과 행렬간 곱셈이 이를 해결하는 중요한 단계이다. 그렇다면 지난 번처럼 예시로 시작해보자. 행렬 두 개가 있고, 이를 곱하려고 한다. 먼저 이를 연산하는 과정을 보여준 후, 자세하게 설명해주겠다. 이 오른쪽 행렬의 첫 번째 열만 가지고 나와 이 왼쪽에 있는 행렬과 곱하려고 한다. 그렇게 되면, 이 벡터에서 11과 9를 얻을 수 있다. 그러니까 지난번 강의에서 본 행렬과 벡터 곱셈과 같은 것이다. 미리 연산을 해놨기 때문에 답이 11, 9가 나온다는 것을 알고 있다.

01'10"

두 번째 단계는 오른쪽 행렬의 두 번째 열을 가지고 이것을 왼쪽의 행렬에다가 곱해주는 것이다. 다시 말해, 지난번 강의에서 본 행렬과 벡터 곱셈과 같은 것이다. 곱셈의 결과는 이 벡터에서 10, 14를 얻을 수 있다. 만약 행렬과 벡터 곱셈을 연습하고 싶다면, 잠시 영상을 멈추고 스스로 복습해보기를 바란다. 그런 다음에 결과값 두 개를 이렇게 적어주면, 바로 답이 나온다. 그러므로 이 결과 행렬은 2×2 행렬이 된다. 이제 행렬 안에 먼저 원소 11, 9를 적어주고, 그 다음에 10, 14를 두 번째 행에 적어준다.

02'07"

이것이 바로 행렬과 행렬 간의 곱셈 과정이었다. 정리하자면 두 번째 행렬의 열을 하나씩 연산해서 답을 모아주면 된다. 잠시 후에 이러한 과정을 좀 더 자세히 살펴보겠다. 여기서 짚고 넘어가야 할 것은 첫 번째 행렬은 2×3 이고, 이를 3×2 행렬과 곱했더니 그 결과는 이렇게 2×2 행렬이 나왔다는 것이다. 잠시 후에 이 과정에 대해 자세히 살펴보자. 지금까지 행렬간 곱셈 과정을 알아봤다. 이제 어떤 과정을 거쳤는지 좀 더 자세히 살펴보자.

02'47"

여기에서 행렬간 곱셈을 상세하게 살펴보자. 행렬 A가 있고 이를 행렬 B와 곱하면 결과로 행렬 C가 나오게 된다. 여기서 보면 차원이 같은 행렬끼리만 곱셈이 가능하다는 것을 알 수 있다. 행렬 A는 $m \times n$ 행렬로, m행 n을 가지고 있다. 이를 $n \times o$ 행렬과 곱해주면, 여기 n이 여기 n과 같은 값이 나온다. 따라서 첫 번째 행렬의 열의 수는 두 번째 행렬의 행의 수와 같아야 한다. 이를 연산하면 행렬 C처럼 $m \times o$ 행렬이 나온다. 지난 번 강의에서 다뤘던 특수한 케이스를 적용해보면, o는 1이 된다. 이는 행렬 B가 벡터라는 것이다. 하지만 여기서는 o의 값이 1보다 큰

경우를 다를 것이다. 이제 두 행렬을 다시 곱해볼 것이다. 먼저 행렬 B에서 첫 번째 열을 벡터의 형태로 가져와 이를 행렬 A와 곱해준다. 그러면 $n \times 1$ 가 나올 것이고, 이를 행렬 C 여기에 써준다. 그 다음으로는 행렬 B에서 두 번째 열을 가져온다. 그러면 또 다른 $n \times 1$ 벡터가 나올 것이고 여기 이 열은 $n \times 1$ 이 되는 것이다. n 차원 벡터가 되는 것이다. 그래서 이 행렬을 여기 $n \times 1$ 벡터와 곱해주면 될 것이고, 이를 여기에 써주고 이를 반복하면 된다. 이제 행렬 B에서 세 번째 열을 가져와 여기 행렬과 곱해준다. 그러면 m 차원 벡터가 나올 것이고, 마지막 열 연산까지 이를 반복하면 된다. 마지막 열과 이 행렬을 곱해주면 행렬 C의 마지막 열이 나올 것이다. 다시 한번 정리하면, 행렬 C의 i번째 열은 행렬 A와 행렬 B의 i번째 열을 곱했을 때 얻어지고, i값은 1, 2에서 시작해 ∞ 까지 적용된다. 지금까지 행렬 C를 연산하는 방법을 알아봤다.

05'12"

예시를 하나 더 살펴보자. 이 행렬 두 개를 곱하려고 한다. 먼저 두 번째 행렬의 첫 번째 열을 꺼낸다. 이는 이전 슬라이드에서 행렬 B와 같은 것이므로, 이 행렬과 벡터를 곱해준다. 그렇다면 얼른 연산을 해보자. 여기 1, 3 을 $0 + 3 \times 3$ 이 된다. 두 번째 원소는 2, 5를 $0 + 5 \times 3$ 이 된다. 결과적으로 9, 15가 나온다. 연두색으로 표시해보겠다. 여기 9, 15가 나왔다. 다음으로 이 행렬의 두 번째 열을 꺼내서 연산해보자. 이 행렬을 이 벡터의 1, 2와 곱해주면, 연산해보면 $1 \times 1 + 3 \times 2$ 가 나오고 이는 여기 첫 번째 행을 연산해줬다. 이 다음을 연산해보면, $2 \times 1 + 5 \times 2$ 가 나온다. 그러면 결과는 7과 12가 나온다. 따라서 여기 나온 것처럼 두 행렬의 곱셈의 결과로 이것은 여기에, 이것은 여기에 써주면 된다. 9, 15 그리고 4, 12 이렇게. 눈치챘겠지만, 2×2 행렬과 또 다른 2×2 행렬을 곱해주면 그 결과는 처음 2와 마지막 2로 2×2 행렬이 나옴을 알 수 있다.

Q1) 이와 같은 방정식에서 a의 값은?

(정답 3번)

Q2) 이와 같은 방정식에서 b의 값은?

(정답 3번)

Q3) 이와 같은 방정식에서 c의 값은?

(정답 3번)

Q4) 이와 같은 방정식에서 d의 값은?

(정답 4번)

07'35"

마지막으로 행렬과 행렬간 곱셈을 하는 숨겨진 팁을 하나 보여주겠다. 예전에 다뤄본 사이즈가 다른 네 채의 주택 가격을 예측하려고 한다. 여기 오른쪽에 세 개의 경쟁 가설이 있다. 이 세가지 가설을 네 채의 주택에 적용하려고 한다면 행렬과 행렬 곱셈을 활용하여 이를 효율적으로 연산할 수 있다. 지난 번 강의에서 봤던 행렬이 왼쪽에 있고, 여기 값은 주택의 가격이다. 그리고 그

왼쪽에 1로 추가해놨다. 그리고 옆에 새로운 행렬을 적어놨는데, 첫 번째 열은 -40, 0.25 그리고 두 번째 열은 200, 0.1 또 이렇게 있다. 이 두 행렬을 곱해주면 결과적으로 여기 첫 번째 열을, 파란색으로 표시하겠다. 이 열을 어떻게 구할 수 있는가? 여기 첫 번째 열을 구하기 위해 행렬과 행렬간의 곱셈 방법은 이 행렬을 가지고 여기 첫 번째 열과 곱해주면 된다. 지난 번 강의에서 살펴봤듯이, 이것이 바로 여기 있는 첫 번째 가설의 예상 주책 가격이 된다. 그렇다면 두 번째 열은? 여기 두 번째 행이 있다. 이 또한 마찬가지로 이 행렬과 여기 두 번째 열을 곱해주면 된다. 그렇게 되면 여기 위에 있는 두 번째 가설을 예측한 값이 나온다. 세 번째 열도 이런 식으로 해주면 된다. 여기서 연산 과정을 하나하나 보여주지 않았지만, 영상을 잠시 멈추고 직접 연산해봐서 연산 과정이 맞는지 확인해보면 좋을 것이다.

10'01"

여기서 우리는 두 개의 행렬을 만들어, 이를 통해 여기 3 개의 가설을 네 개의 주택 사이즈에 모두 빠르게 적용해 12개의 예상 가격을 얻어낼 수 있었다. 그러니까 하나의 행렬 곱셈만으로 12개의 예상값을 구할 수 있었던 것이다. 더 좋은 점은 이러한 행렬 곱셈을 하기 위한 다양한 좋은 선형 대수학 라이브러리가 많이 있다는 것이다. 또한 여러분이 어떠한 프로그래밍 언어를 사용한다 할지라도 상위 10개 인기 프로그래밍 언어는 다양한 선형 대수학 라이브러리를 가지고 있을 것이다. 그리고 행렬간 곱셈을 매우 효과적으로 연산할 수 있도록 최적화된 훌륭한 선형 대수학 라이브러리가 있을 것이다. 여러분의 컴퓨터에서 실행 가능한 어떠한 종류의 병렬 연산에도 활용할 수 있다. 컴퓨터가 멀티플 코어인지 멀티플 프로세서를 가지고 있는지는 관계 없다. 프로세서에서도 SIMD 패러렐리즘이라 불리는 패러렐리즘도 있으니 여러분의 컴퓨터에서 활용 가능하다. 행렬 간 곱셈을 할 때 활용 가능한 무료 라이브러리도 얼마든지 좋은 것들이 있으니, 이러한 가설을 통해 다양한 예상값을 효과적으로 도출해낼 수 있다.

No.	Title
Week 1-16	Matrix Multiplication Properties

00'00"

행렬 곱셈은 하나의 행렬 곱셈 과정에 다양한 연산과정을 담을 수 있다는 점에서 매우 유용하다. 하지만 이를 활용할 때 주의해야 한다. 이번 강의에서는 행렬 곱셈의 법칙에 대해서 몇 가지 살펴보고자 한다. 실수나 스칼라를 가지고 연산할 때, 곱셈은 교환법칙이 성립한다. 즉, $3 \times 5 = 5 \times 3$ 같으므로 어떤 숫자가 먼저 오든지 곱셈할 때 관계가 없다는 것이다. 이를 실수 곱셈의 교환

법칙이라고 한다. 이 법칙은 곱셈을 하는 것의 순서를 서로 바꿀 수 있다는 것이다. 하지만 이는 행렬에서는 적용되지 않는다. 예를 들어 행렬 A와 행렬 B가 있다. 일반적으로 $A \times B = B \times A$ 가 성립하지 않는다. 따라서 이를 주의해야 한다. 따라서 행렬 곱셈을 할 때 임의로 순서를 바꿔서는 안 된다. 그러므로 세련되게 표현하자면 행렬 곱셈은 교환법칙이 성립하지 않는다고 말할 수 있다.

01'08"

예를 들어보자. 여기 두 개의 행렬이 있다. 여기 $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ 과 $\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}$ 이 두 행렬을 곱하면 결과는 오른쪽처럼 나온다. 그럼 이제 두 행렬의 순서를 바꿔보자. 이 두 행렬을 가져와 순서를 바꾸면 된다. 바꾼 두 행렬을 곱해보면, 이와 같이 다른 두 번째 답이 나온다. 여기 보면 이 두 행렬은 서로 같지 않다. 그러므로 일반적으로 $A \times B$ 와 같은 행렬 연산을 해야 할 경우, A가 $m \times n$ 행렬이고 B가 $n \times m$ 행렬일 경우, 예를 들자면, 이는 행렬 $A \times B$ 는 $m \times m$ 행렬의 결과가 나온다. 반면에

$B \times A$ 의 결과는 $n \times n$ 이 나온다. 그러므로 행렬의 차원조차도 다른 것이다. 다시 말해, $A \times B$, $B \times A$ 각각의 결과로 나온 행렬의 크기도 서로 다르다는 것이다. 좌측의 예시는 모두 2×2 행렬을 예로 들었다. 그래서 행렬의 차원이 서로 같았다. 하지만 일반적으로 행렬의 순서를 바꾸게 되면 그 결과로 나오는 행렬의 차원이 서로 다르다. 그래서 행렬 곱셈은 교환법칙이 성립하지 않는다.

02'34"

또 다른 특징을 알아보자. 우리가 실수나 스칼라를 이야기할 때, $3 \times 5 \times 2$ 이런 식이 있다고 가정하자. 그럼 먼저 5×2 연산하고, 3×10 을 연산하든지 아니면, 3×5 를 먼저 연산하고 15×2 를 연산할 수도 있다. 양 쪽 다 같은 답을 준다. 모두 답은 30이다. 그러므로 이런 경우에는 5×2 나 3×5 중에서 어떤 것을 먼저 연산한다고 해도 상관이 없다. 왜냐하면 $3 \times (5 \times 2) = (3 \times 5) \times 2$ 이 성립하기 때문이다. 이는 실수 곱셈의 결합법칙이라고 한다. 행렬 곱셈도 결합 법칙이 성립한다. 예를 들어보자. 이렇게 세 개의 행렬이 있고, 이렇게 $A \times B \times C$ 연산하려고 한다. 이 때 $A \times (B \times C)$ 하거나, $(A \times B) \times C$ 할 수 있다. 둘 다 같은 답이 나올 것이다. 이를 증명해 보이지는 않겠지만, 내 말을 믿어도 된다. 다시 말해서, 여기 방법이 두 가지가 있다. 첫 번째 것을 보자. 첫 번째 방법은

$A \times B \times C$ 를 연산하려고 하면, 먼저 $B \times C$ 를 연산한다. 그리고 $D = B \times C$ 라고 가정하면, $A \times D$ 를 연산하면 되면 결국 $A \times B \times C$ 를 연산하게 되는 것이다. 두 번째 방법은 $A \times B$ 를 먼저 연산하고, $E = A \times B$ 라고 가정한 후에, $E \times C$ 를 연산하면 된다. 그러면 똑같이 $A \times B \times C$ 가 된다. 그리고 장담하건데, 이 두 값은 서로 같다. 따라서 행렬 곱셈은 결합 법칙이 성립한다는 것이다. 결합법칙, 교환법칙과 같은 용어에 대해 걱정하지 마라. 그저 그렇게 부르고 있는 것이고, 이 용어를 다음 강의에서 그렇게 많이 활용하지는 않을 것이다. 그러니 이를 애써 외우지 않아도 된다.

04'51"

마지막으로 특수한 행렬인 항등 행렬에 대해 알아보겠다. 먼저 이를 우리가 잘 알고 있는 실수에

비유해 보겠다. 실수 또는 스칼라에서 숫자 1은 곱셈에서 항등원이다. 즉, 숫자 z 가 있을 때, $1 \times z = z \times 1 = z$ 가 성립하므로 z 에 어떠한 숫자가 들어가도 이는 성립하게 된다. 그래서 1은 항등원이고 이 방정식을 총족한다. 따라서, 행렬의 세계에서, 항등 행렬이 있고, 이는 보통 I 로 표기 되는데, 그 크기를 설명하려고 하면 $I (n \times n)$ 으로 표기하기도 한다. 즉, I 아래함수 $n \times n$ 은 $n \times n$ 항등 행렬이다. n 차원에 따라 항등 행렬이 각기 다르다.

05'52"

몇 가지 예를 들겠다. 여기 2×2 항등 행렬, 3×3 항등 행렬, 4×4 항등 행렬이 있다. 그리고 이러한 항등 행렬은 이렇게 대각선을 따라 숫자 1을 원소로 가지고 있다는 특징이 있다. 또한 그 나머지 원소는 모두 0이다. 따라서 1×1 항등 행렬의 경우는 숫자 1이므로, 1×1 항등 행렬에는 숫자 1하나만 있는 것이다. 따라서 이는 그렇게 흥미로운 항등 행렬은 아니다. 비공식적으로 이 행렬을 대충 빠르게 그리자면, 대부분 이런 식으로 항등 행렬을 요약해서 그린다. 대괄호를 먼저 그리고, 이렇게 숫자 1, 1, 1, ..., 1 을 쓴 다음에 나머지 부분에 0을 막 쓸 것이다. 여기 크게 쓴 0은 이 행렬이 여기 대각선 말고는 나머지는 모두 0으로 이루어져 있다는 것을 표시하기 위함이다. 따라서 이렇게 항등 행렬을 간단하게 표기하는 방법을 알아봤다.

06'51"

따라서 항등 행렬은 자신만의 특징을 지니고 있음을 알 수 있다. 어떠한 행렬 A 가 있을 때, $A \times I = I \times A = A$ 이므로 아까 봤던 방정식과 같은 형태임을 볼 수 있다. 그러니까 여기 $1 \times z = z \times 1 = z$ 처럼 $A \times I = I \times A = A$ 도 성립한다는 것이다. 행렬의 크기가 맞는지 살펴보면, 행렬 A 는 $m \times n$ 이고, 여기 항등 행렬은 $n \times n$ 이 된다. 그렇게 되면, 이 것은 항등 행렬이다. 행렬 곱셈에서 이 공식이 성립하려면, 이 행렬은 $m \times m$ 행렬이 된다. 여기 m 은 이 m 과 같아야 하기 때문이다. 따라서 어떤 경우에는 이 과정으로 나온 결과는 행렬 $A m \times n$ 이 그대로 나온다.

07'45"

항등 행렬 I 를 쓸 때는 행렬의 크기가 표시되어 있지 않을 것이다. 그래서 여기 I 두 개가 있는데 이는 사실은 두 개의 다른 크기의 행렬이다. 하나는 $n \times n$ 이고 다른 하나는 $m \times m$ 이다. 하지만 행렬의 크기를 표시하고자 할 때에는 여기처럼 $I (nxn)$ 이렇게 표시하기도 한다. 하지만 대체로 행렬의 크기는 생략되어 있다. 마지막으로 정리하자면, 아까 말했듯이, 대체로 $AB = BA$ 가 성립하지 않는다. 대부분의 행렬 A 와 B 에서는 이는 성립하지 않는다. 하지만 행렬 B 가 항등 행렬일 경우, $A \times I = I \times A$ 이와 같은 식이 성립하는데, 행렬 B 가 항등 행렬이 아닌 다른 행렬일 경우는 해당되지 않는다.

Q) 다음을 연산하시오.

정답 3번

08'40"

지금까지 행렬 곱셈의 특징과 항등 행렬과 같은 특수한 행렬에 대해서 알아봤다. 다음 번 강의에서는 선형 대수학 리뷰에 대해서 마지막으로 다룰 것이다. 따라서 특수한 행렬 연산에 대해 몇 가지 더 알아본 후에, 이번 강의를 위해 필요한 선형 대수학에 대한 모든 것을 다루도록 하겠다.

No.	Title
Week 1-17	Inverse and Transpose

00'00"

이번 강의에서는 특수한 행렬 두 개를 알아볼 텐데, 바로 역행렬과 전치행렬이다. 먼저 역행렬에 대해 알아보자. 이해를 돋기 위해 실수를 가지고 이를 설명해보겠다. 지난 강의에서 숫자 1은 실수의 세계에서 항등의 역할을 하는데 이는 1 에다가 어떠한 것을 곱해도 바로 그 자신이 나오기 때문이다. 실수는 이러한 특징을 가지고 있는데, 모든 숫자는 역수를 가지고 있다. 예를 들어, 3을 가지고 생각해보면, 여기에 어떤 숫자가 존재하는데, 이는 3의 역수로 이렇게 표현하고 이것을 3과 곱하면 항등 원소 1이 나온다. 여기 3의 역수는 3분의 일과 같다. 또 다른 숫자로 해보면, 12가 있고, 12의 역수인 12^{-1} 즉, $1/12$ 가 있다. 그리고 이 두 숫자를 곱해주면, 1이 나와 항등 원소가 된다.

01'13"

실수의 세계에서 모든 숫자가 역수를 가지고 있는 것은 아니다. 예를 들어 숫자 0은 역수가 없다. 0의 역수는 0^{-1} , $1/0$ 이므로 이는 정의할 수 없는 수다. 이제 이번 슬라이드에서는 역행렬을 연산하는 것에 어떠한 의미를 지니는지 배워볼 것이다. 개념부터 살펴보자. A가 $m \times m$ 행렬이면, 이는 역행렬을 지닌다. 이 부분은 이따가 더 자세히 알아보겠지만, A의 역행렬은 A^{-1} 이 되므로, $A \cdot A^{-1} = A^{-1} \cdot A = I$ 인 항등 행렬과 같다. $m \times m$ 형태의 행렬에서만 m은 역수를 가진다. 그래서 $m \times m$ 행렬은 정방 행렬이라고도 불리는데 이는 행과 열의 수가 같기 때문이다. 그러므로 정방 행렬만 역행렬을 가지고 있는 것이다. 그래서 A는 정방 행렬이자, $m \times m$ 행렬이고 이에 대한 역행렬은 여기 있다.

02'37"

자세한 예시를 살펴보자. 여기 이렇게 $[3, 4 ; 2, 16]$ 행렬이 있다. 이는 2×2 행렬이자, 정방 행렬이므로 역행렬을 가지고 있다. 이의 역행렬은 다음과 같은 $[0.4, -0.1; -0.05, 0.075]$ 2×2 행렬이다. 따라서 이 두 행렬을 곱하면 2×2 의 $[1, 0; 1, 0]$ 항등 행렬이 나온다. 2×2 행렬이다. 여기 보면, 여기 A 행렬이 있고, 이 것은 A의 역행렬이다. 따라서 $A \cdot A^{-1} =$ 항등 행렬이 나온다.

03'37"

그렇다면 여기서 어떻게 역행렬을 연산해냈을까? 가끔 역행렬을 직접 손으로 연산하는 사람들도 있지만, 요즘은 거의 그렇지 않는다. 왜냐하면 매우 성능 좋은 수치연산 소프트웨어가 행렬을 가지고 역행렬을 연산해내기 때문이다. 즉, 오픈 소스 라이브러리가 많기 때문에 어떠한 인기 있는 프로그래밍 언어를 통해 역행렬을 연산할 수 있는 것이다. 간단한 예시를 하나 보여주겠다. 이 역행렬을 연산한 방법은 옵티브라는 소프트웨어를 활용했다. 바로 이것이다. 옵티브에 관한 건 이후에도 많이 등장할 것이다. 여기서는 간단히 예시만 보여줄 것이다. 이렇게 왼쪽에 있는 행렬 A를 $[3 \ 4 ; 2 \ 16]$ 써주면, 이게 바로 행렬 A를 입력한 것이다. 그러면 이렇게 입력이 되고, 여기 왼쪽 아래에 있는 행렬이랑 같은 것이다. 이 소프트웨어는 A의 역행렬을 아주 쉽게 연산하게 해준다. 이렇게 $\text{pinv}(A)$ 를 입력하면 이렇게 $[0.4, -0.1 ; -0.05, 0.075]$ 역행렬이 주어진다. 이는 A의 역행렬이 무엇인지 수치해법을 알려준다. 또한 $\text{inverse}(A) = \text{pinv}(A)$ 라고 입력하고, $A * \text{inverse}(A)$ 를 입력하면 대각선으로 1, 1 그리고 나머지는, 여기 -17, -16이 있는데 수치 정확도가 이 컴퓨터에서 최종 행렬의 찾을 때 반올림 오차 때문에 이 숫자들은 없는 것과 마찬가지고, 따라서 0이라는 것을 확인할 수 있다. 따라서 $A * A^{-1}$ 은 최종적으로 항등 행렬이 되는 것이다. 이를 확인해보기 위해, $\text{inverse}(A) * A$ 를 입력하면 이는 또한 항등 행렬이 나오는데, 여기 대각선은 1이고, 나머지 부분도 반올림 오차를 무시하면 원래는 0이다.

05'45"

여기서 역행렬의 정의는 먼저 강조했다시피 정방 행렬이어야 하고 또한 행렬 A가 역행렬을 가지고 있다면, 정확히 어떠한 행렬이 역행렬을 가지고 있는지는 리뷰를 위한 선형 대수학의 범위를 넘어선다. 그래서 간단하게 설명하자면, 숫자 0은 역수가 없다. 그래서 숫자 0을 원소로 지닌 A행렬 또한 역행렬을 가지고 있지 않은데, 이는 A의 역행렬이 없기 때문이다. 따라서 행렬과 다른 행렬을 곱하면 항등 행렬이 나와야 하는데, 이 행렬은 모두 0으로 이루어져 있다. 이러한 경우 이외에도 이처럼 역행렬을 가지고 있지 않은 행렬이 몇 개 더 있다. 이번 리뷰에서 행렬에서 역행렬을 가지는 것의 의미를 그렇게 깊게 다루고 싶지 않다. 하지만 기계 학습 응용에서는 이것은 크게 문제되지 않을 것이고, 더 정확하게 말하면 학습 알고리즘에서 그러니까 역행렬이 등장하는지 여부를 떠나 해당 학습 알고리즘을 설명할 때가 되면 역수를 가지는 것과 가지지 않는 것에 의미를 설명하고 역수가 없을 때 이를 어떻게 고칠지 설명하도록 하겠다. 간단하게 설명하자면, 역행렬이 없는 행렬은 어찌 보면 0과 매우 가깝기 때문이라고 알아두자. 용어 설명을 마치기 전에, 역행렬을 가지고 있지 않은 행렬은 때로는 특이 행렬 또는 축퇴 행렬이라고 한다. 따라서, 여기 이 행렬은 특이 행렬 또는 축퇴 행렬이 되는 것이다.

07'31"

마지막으로 소개할 특수한 행렬은 행렬을 전치하는 것이다. 여기 행렬 A가 있고, 이 행렬을 전치하면 여기 오른쪽과 같은 행렬을 얻을 수 있다. 이것이 전치되었다는 뜻이고, 기호로는 A^T 쓴다. 이제 행렬을 전치하는 방법을 설명하겠다. A 행렬의 첫 번째 행에서 1부터 0까지를 가져와 A^T 의 첫 번째 열에 놓는다. 그리고 A 행렬의 두 번째 행 3, 5, 9를 가져와 A^T 의 두 번째 열에 놓는다. 컴퓨터가 전치하는 또 다른 방법은 여기에 45도 기울기 축을 그려 미러링하거나 아니면

45도 기울기 축을 따라 돌리는 것이다.

08'22"

이제 좀 더 공식적인 전치 행렬의 정의에 대해 이야기해보자. A 는 $m \times n$ 행렬이고, B 는 이렇게 A 의 전치행렬이라고 하자. 따라서 B 는 반대로 $n \times m$ 행렬이 된다. 이것은 2×3 행렬이므로, 이의 전치 행렬은 3×2 가 된다. 따라서 $B_{ij} = A_{ji}$ 가 된다. 따라서 행렬 B 의 ij 원소는 아까 행렬 A 에서는 ji 원소가 된다. 예를 들면, B_{12} 는 여기 행렬은 보면 B_{12} 는 여기 첫 번째 행, 두 번째 열의 원소 3이다. 이는 여기 원소 3, 그러니까 두 번째 행, 첫 번째 열의 원소 3과 같다. 또 B_{32} 는 원소 9인데 이는 여기 A_{32} 와 같은 9이다. 따라서 전치 행렬이 의미하는 것을 종합해보면 이는 선형 대수학 리뷰의 마지막 단계이다.

Q) 다음 문제를 연산하시오.

정답 4번

09'38"

따라서 지금쯤이면 행렬간 덧셈과 뺄셈, 그리고 곱셈 연산을 할 줄 알며, 역행렬과 전치행렬의 의미가 무엇인지를 이해했을 것이다. 이 과정에서 다를 선형 대수학에서 필요한 주요 내용을 모두 다룬 것이다. 만약 이러한 자료를 처음 본 것이라면, 굉장히 많은 선형 대수학 자료이고, 이를 빠르게 훑어봤으면, 이해해야 할게 많다고 생각할 것이다. 하지만 우리가 배운 정의를 모두 다 암기할 필요는 없다. 이 슬라이드 사본이나 강의 노트를 웹사이트에서 다운받아서 참고 자료로 활용한다면, 정의를 다시 찾아볼 수 있고, 행렬 곱셈, 전치행렬 그리고 그 외의 정의에 대해서도 이해할 수 있을 것이다. 또한 웹사이트에 있는 강의 노트는 선형 대수학에 관한 추가 자료를 제공하고 있으므로 스스로 좀 더 학습할 수 있다.

10'55"

이렇게 배운 행렬을 가지고, 다음 강의에서 선형 회귀에서 더 다양하게 활용하는 방법을 배워서, 더 많은 데이터와 특징, 그리고 학습예시 등을 살펴볼 수 있을 것이다. 이후에 새로운 회귀를 학습한 후에 여기서 배운 선형 대수학을 활용하여 더 학습 알고리즘을 얻어내는 데 활용할 것이다.

Week 2

No.	Title
Week 2-1	Multiple Features

00'00"

이 영상에서는 좀 더 강력한 버전의 새로운 선형 회귀의 대해 살펴보도록 하자. 이 모형에서는 더 많은 변수와 더 많은 값을 적용할 수 있다. 무슨 뜻인지 보자. 우리가 개발한 원래 선형 함수에서, 우리는 x 를 집의 크기라고 하였고 이를 활용해서 집의 가격인 y 값을 예측했다. 하지만 이제는 가격을 예측하기 위해 집의 크기 x 만을 변수로 아는 것이 아니라 다른 요소들, 방의 수나

집의 개수, 그리고 집이 얼마나 오래 되었는가도 변수로 가지게 되었다고 해보자. 이런 모형은 우리가 가격을 예측하는데 더 많은 정보를 제공할 것이다. 이 표기법에 대해서 좀 더 설명을 하자면, 이렇게 x_1 , x_2 등의 독립 변수가 있고, 이 경우에는 4 가지 값이 있다면, 출력값인 집의 가격을 찾기 위해 이것을 Y 라는 종속변수로 표시할 것이다. 추가적으로 우리는 소문자 n 을 사용해서 값의 개수를 표기할 수 있다. 이 경우에는 $n=4$ 이다. 왜냐하면 이렇게 4 가지의 값이 있기 때문이다.

01'38"

이렇게 47 개의 열이 있다면 M 은 이 표의 열의 수 또는 트레이닝 예시의 개수이다. $x^{(i)}$ 는 i 번째 행을 가리키는 입력 변수이다. 구체적인 예로, $x^{(2)}$ 가 두 번째 트레이닝 예시의 벡터 값이라고 하자. 두 번째 트레이닝 예시들은 표에서 두 번째 열을 의미하고 $x^{(2)}$ 는 집의 가격을 예측하기 위한 네 개의 값을 가진다. 여기 $x^{(2)}$ 의 2 는 트레이닝 셋의 인덱스를 의미한다. 이것은 우리 표의 2 번째 줄을 의미한다. X_2 는 이렇게 4 차원 벡터이다. 사실 좀 더 일반적으로 이것은 n 차원 벡터이다.

02'54"

이런 표기법에서 보면 $x^{(2)}$ 는 이제 벡터이다. $x_j^{(i)}$ 를 사용하여 i 번째 트레이닝 예시의 j 의 값을 나타낼 것이다. $x_j^{(i)}$ 는 i 번째 행에서 j 번째 인덱스 (열) 의 값이다. 구체적으로 보면, $x_3^{(2)}$ 은, 이 벡터에서 3번째 행이므로 2를 가리킨다. 따라서 $x_3^{(2)} = 2$ 이다.

Q) 위의 트레이닝 셋에서 $x_1^{(4)}$ 가 가리키는 것은? 정답: 3 번

이제 여러 가지 값을 갖고 있으니 우리 가설이 어떻게 되는지 알아보자. 예전에는 우리의 가설은 이러했고 x 값 하나만 가지고 있었는데, 이제 값이 여러 개가 있기 때문에 이렇게 단순한 식은 사용하지 않을 것이다.

03'44"

그 대신, 선형 회귀의 가설은 $h_{\theta}(x) = \theta_0 + \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \theta_4x_4$ 와 같은 식으로 나타낼 것이다. 우리가 n 개의 값을 갖고 있다면 n 개의 값을 위와 같은 식으로 더하는 것이다. 구체적으로 우리의 매개변수를 예를 들어 세워보면, $h(x)= 80 + 0.1 x_1 + 0.01x_2 + 3x_3 - 2x_4$ 가 된다. 이것은 우리 가정의 예이고, 집의 가격을 예측할 때 원가는 8 만이라고 하고 플러스 이상의 것들을 더하는 것이다. 얼마나 총 수가 많으냐에 따라서 가격은 달라질 것이다. X_2 는 총수를 나타내고, X_3 은 방의 개수를 나타낸다. 여기까지는 가격이 올라가는 변수이다. 하지만 연수에 따라 가격은 줄어들 것이다. 오래된 집일수록 가격이 떨어진다는 것이다.

05'08"

다시 가정을 써보면 이렇다. 그리고 이 등식을 간소화하기 위해 좀 더 표기법을 사용하도록 하겠다. 좀 더 쉽게 하기 위해서 $x_0 = 1$ 이라고 가정하자. 좀 더 자세히 말하면 이것은 i 의 예

마다 x_i 의 값을 갖는데, x_0^i 는 1 인 것이다. 이것을 0 의 값을 추가적으로 발견하는 것이라고 생각해도 될 것이다. 지난번에는 x_1, x_2, \dots, x_n 까지 있기 때문에 n 개의 값을 가졌다. 하지만 값 0 을 포함함으로써 $N+1$ 개의 값을 갖게 된다. 이는 이제 $n+1$ 개 차원의 벡터가 될 것이고, 0 인덱스이다. 앞으로 매개변수를 벡터로 생각할 것이다. 즉, 여기 매개 변수 $\theta_0, \dots, \theta_n$ 이 있다. 이를 매개변수 값으로 쭉 적어 내린다. 이것은 또 다른 0 인덱스 벡터이고 $n+1$ 차원 벡터가 된다. 따라서 이 가정은 이제 $h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \dots + \theta_n x_n$ 으로 적을 수 있다. 이런 등식은 위에 있는 것과 같은데 왜냐하면 $x_0 = 1$ 이기 때문이다. 또한 이는 $\theta^T x$ 로 쓸 수도 있다. $\theta^T = [\theta_1 \dots \theta_n]$ 을 전치한 것으로 나타내어지고 따라서 $N+1$ 곱하기 1 벡터이며 행 벡터라고도 불린다.

07'32"

이것을 벡터 x 즉, x_0, \dots, x_n 과 곱한다. $\theta^T x$ 는 이것과 같아진다. 이것이 가정을 표현하는 좀 더 간단한 방법이고, 변수 벡터 θ 와 θ 벡터 X 를 계산한 결과이다. 표기의 편의를 위하여 이렇게 좀 더 간단하게 사용하도록 하자. 이의 이름을 붙이자면 다변수 선형회귀라고 볼 수 있다. 이 다변수는 변수가 많다는 것을 의미하는데, 우리는 이 값으로 Y 값을 구하려고 하고 있다.

No.	Title
Week 2-2	Gradient Descent for Multiple Variables

00'00"

이전 강의에서는 다변량 선형 회귀함수에 대한 가정에 대해서 다루었다. 이번 강의에서는 매개변수를 어떻게 이 가정에 맞추는지를 다룰 것이다. 특히 다변량 선형 회귀의 gradient descent 를 어떻게 사용하는지에 대해 배울 것이다. 우리의 표기법을 좀 간략히 상기하면 이것이 우리가 $x_0 = 1$ 이라고 정한 다변량 선형 회귀의 가정함수이다. 이 모델에서 매개변수는 $\theta_1 \dots \theta_n$ 이다. 이것을 n 개의 개별 변수로 생각하지 않고 $n+1$ 의 값을 가진 벡터 θ 라고 생각할 것이다.

00'52"

그러니 이 모델의 변수를 벡터로 생각할 것이다. 비용함수 $J(\theta_0)$ 에서 θ_n 은 제곱 오차의 합이라고 볼 수 있다. 하지만 J 를 $n+1$ 숫자들의 함수로 보는 대신 매개변수 벡터 θ 의 J 라고 칭할 것이다.

Q) n 개의 값이 있을 때, 우리는 비용함수를 다음과 같이 정의한다. 선형 회귀에 있어서 다음 중 어느 식이 $J(\theta)$ 를 맞게 정의한 것인가? 정답: 2,3,4 번

그러면 gradient descent 가 이런 모양일 것이다. 우리는 θ_j 를 $\theta_j - \alpha \cdot$ 미분계수에 따라 업데이트 할 것이다. 이를 다음과 같이 표시하겠다. 이 비용함수의 편미분계수는 변수 θ_j 를 반영할 것이다. 여기에 Gradient Descent 를 실행하면 어떻게 되는지 보고, 특히 다음의 편미분계수가 어떻게 되는지 살펴보자.

01'58"

여기 우리가 $N=1$ 의 값을 가질 때의 gradient descent 가 있다. 우리는 매개 변수 θ_0 과 θ_1 에 대해, 업데이트를 해야 하는데, 여러분에게 이 식이 친숙하기를 바란다. 그리고 이 부분은 물론 매개변수 θ_0 를 반영한 비용함수의 편미분계수이다. 유사한 방식으로 변수 θ_1 에 대해서 업데이트를 해야 한다. 조금 다른 점이 있는데, 우리가 이전에는 값을 하나만 가졌고 우리는 그것을 $x^{(i)}$ 값이라고 불렀다. 하지만 이제 우리는 이것을 $x_1^{(i)}$ 이라고 부를 것이다. 이것은 우리가 하나의 값만 가지고 있을 때의 얘기였다. 이제 값이 1 이상인 새로운 알고리즘을 살펴보자. 여기서 n 값은 1 보다 훨씬 클 수 있다. 우리는 이 업데이트 된 gradient descent 를 살펴보고, 미적분을 아는 사람들은 비용함수의 정의와 매개변수 θ_j 를 반영한 비용함수 J 의 편미분계수를 생각해 본다면, 미분계수가 내가 여기 표시한 파란색 상자 부분과 동일하다는 것을 알게 될 것이다.

03'12"

그리고 이것을 실행하면 여러분은 여러 변수를 가진 선형 회귀의 gradient descent 를 실행하게 될 것이다. 마지막으로 이 슬라이드에서 여러분에게 이 오래된 알고리즘과 새 알고리즘이 왜 같은 종류인지, 그리고 왜 비슷한 종류인지, 왜 둘 다 gradient descent 알고리즘인지를 알려주고자 한다. 우리가 2 개의 값이나 혹은 그 이상을 가지고 있는 경우를 생각해 보자. 예를 들어, 우리는 $\theta_1, \theta_2, \theta_3, \dots$ 에 대한 업데이트, 혹은 더 많다면 더 많은 업데이트가 있다고 하자. θ_1 의 업데이트 공식을 보면 $n=1$ 이었을 때의 업데이트 공식과 같다는 것을 알게 될 것이다. 그리고 이 두 공식이 같은 이유는 여기 $x_0^{(i)}$ 를 똑같이 $x_0^{(i)}$ 으로 두었기 때문이다. 따라서 결과물이 자주 색 박스 값과 같다는 것을 알 수 있다. 비슷하게, θ_1 의 업데이트 공식을 보면 이 항이 예전에했던 θ_1 의 항과 같다는 것을 알 수 있다. 우리는 첫 번째 값을 표기하기 위해 $x_1^{(i)}$ 을 만들었고, 다른 변수들에 대해서도 비슷한 업데이트 공식이 적용된다. 이번 슬라이드에서 다른 내용이 많기 때문에 이 영상을 멈추고 이 슬라이드의 공식을 천천히 살펴 본다면 뭐가 뭔지 알 수 있을 것이다. 여기 적혀 있는 알고리즘을 사용한다면 여러 값을 가진 선형 회귀를 사용할 수 있을 것이다.

No.	Title
Week 2-3	Gradient Descent in Practice 1 – Feature Scaling

00'00"

이번 영상과 다음 번 영상에서는 gradient descent 를 잘 활용하기 위한 실용적인 방법들에 관해 논해 볼 것이다. 이번 영상에서는 값 조정(Feature Scaling)에 대해 소개할 것이다.

00'11"

여기 값 조정의 원리가 있다. 여러 가지 값을 가지고 있고, 비슷한 값의 범위에 있는 다른 값들이라면 gradient descent 의 수렴이 더 빨리 이뤄진다. 자세히 설명하면, 집의 크기인 x_1 의 범위는 0~2000 이며, 방의 개수 x_2 의 범위는 1~5 이다. 이 $J(\theta)$ 비용함수의 곡선은 이렇게 나타날 것이다. $J(\theta)$ 는 $\theta_0, \theta_1, \theta_2$ 의 함수이다. 하지만 θ_0 , 값을 무시하고 θ_1, θ_2 만 있다고 가정하고 함수를 그리자. x_1 이 x_2 보다 더 큰 범위의 값을 가지기 때문에 다음과 같이 굉장히 얇은 타원형 함수가 그려질 것이다. 만약 2000-5 의 비율이라면 이보다 더 얇을 것이다. 이는 굉장히 길고 얇은 타원으로 함수 $J(\theta)$ 의 윤곽선일 것이다.

01'29"

이 비용함수에서 gradient descent 를 실행한다면 한다면 굉장히 오래 걸리고 같은 작업을 계속 반복하여야만 전역 극소값에 다다를 수 있다. 실제로 여러분이 그리는 것보다 이 곡선이 더 뾰족할 수 있다. 그리고 gradient descent 는 굉장히 오래 걸린다. 이런 세팅에서는 값을 측정하는 것이 유용하다. 자세히 말하면 x_1 의 사이즈를 범위를 이 정도 두고 조정하고 x_2 를 방의 개수로 측정하는 것보다 비용함수 J 를 제대로 조정하는 것이 좀 더 넓은, 원형의 함수를 그릴 수 있을 것이다. 그리고 비용함수의 gradient descent 가 이렇게 진행된다면 수학적으로, 여러분은 전역 최소값으로 좀더 직선으로 나아갈 수 있을 것이다. 값을 조정함으로써 소비자의 값의 범위가 생겼다. 이 예에서 우리는 x_1, x_2 두개의 값의 범위를 0 에서 1 사이로 놓는다. 여러분은 gradient descent 를 쓰게 될 것이다. 훨씬 수렴을 빨리 해주기 때문이다.

03'17"

일반적으로 우리가 값을 조정할 때, 우리는 종종 대부분의 값을 -1에서 1의 범위에 둔다. 그리고 자세히 말하면 x_0 값은 항상 1이다. 그러니 이것은 이 범위 내에 있지만 아마도 여러분은 다른 값 또한 나눠서 이 범위에 속하게 할 것이다. 여기서 -1이나 1은 그렇게 중요한 것은 아니다. 그러니까 값이 하나 있다면 0-3 까지든 어떻든 크게 문제가 되지 않는다. 여러분이 값을 여러 가지 갖고 있고, -2에서 +0.5 사이의 값이라면 이것은 -1에서 1까지의 범위와 비슷하므로 괜찮다. 만약 x_3 이라고 불리는 다른 값이 있고 -100에서 100 까지의 범위를 갖는다면 -1에서 1 사이의 범위와 너무 차이가 난다. 그렇다면 이 값은 그렇게 좋은 값이 아니다. 또한 -0.0001에서 0.0001 사이의 값이면 너무 작은 범위이다. 따라서 이것 역시 그렇게 좋은 범위의 값은 아니다. 그러니 -1과 1의 범위에서 100이나 0.0001과 같이 크게 차이가 나지 않는 범위를 가지는 것이

좋다. 모두가 다른 규칙을 갖고 있다. -3에서 3이나 $-1/3$ 에서 $1/3$ 은 괜찮다. 0에서 $1/3$, 혹은 $-1/3$ 에서 0까지도 괜찮다. 하지만 x_4 처럼 아주 작은 범위의 값을 갖더라도 걱정할 필요는 없다. 여러분이 기억해야 할 것은 값이 꼭 같은 값의 범위에 있어야 할 필요는 없다는 점이다. 하지만 이 값의 gradient descent에 더욱 가까울수록 여러분의 일이 수월해진다는 것이다. 나누는 것과 더불어 이런 값을 측정할 때 가장 최대치는 평균 정규화된다고 부른다. 이것이 의미하는 바는 X_i 값이 $X_i - \mu_i$ 이더라도 여러분의 값의 평균은 0 정도일 것이라는 것이다.

05'56"

그리고 우리는 특징 x_0 에 이것을 적용하고 싶을 것인데, x_0 는 항상 1이다. 그러니 0의 평균값을 가질 수 없다. 하지만 다른 값들의 경우에는, 예를 들어, 집의 크기가 0에서 2000 사이라고 하고, 여러분이 평균 집 크기가 1000이라고 한다면 여러분은 이 공식을 사용할 수 있을 것이다. 그리고 (사이즈-평균)/2000이 될 것이다. 유사하게, 여러분의 집이 1-5개의 방을 보유하고 있고, 평균이 2라고 한다면 여러분은 이 공식을 사용할 수 있을 것이다. 두 가지 경우에서 여러분은 x_1 , x_2 에 다르게 된다. 이는 -0.5에서 0.5 사이의 범위를 가질 수 있다. x_2 는 0.5보다 크기 때문에 완전히 들어맞는 것은 아니지만 비슷하다. 더 일반적인 방법은 여러분이 x_1 을 $(x_1 - \mu_1)/s_1$ 로 바꾸는 것이다. μ_1 은 x_1 의 평균값이고 s_1 은 값의 값의 범위라고 할 수 있다. 최대치-최소치 혹은 s_1 은 표준편차라고 하자. 최대치-최소치는 괜찮다. 그리고 x_2 는 완전히 맞지는 않지만 $x_2 - \mu_2/s_2$ 라고 할 수 있다. 그리고 이러한 공식은 여러분의 값을 정확하게는 아니지만 어떠한 범위의 값으로 규정할 수 있을 것이다. 그리고 기술적으로 아주 민감한 사람이 있다면 이것은 실제적으로 4라는 것이다. 그러니 최대값이 5이고 최소값이 1이 된다면 4인 것이다. 하지만 이것은 대략적인 값이다. 값의 범위 측정은 정확할 필요는 없다. 그렇게 해야 더 빠른 gradient descent를 갖게 된다. 그러면 적게 반복해도 된다.

Q) 도시에 있는 집 값을 측정하기 위해 학습 알고리즘을 사용한다고 하자. 여러분은 x_i 값을 사용해서 집의 나이 수를 나타내고 싶다. 트레이닝 세트에서 여러분의 집은 모두 지어진 지 30~50년이 되었고 평균은 38년이다. 다음 중 어떤 것을 사용해서 값을 조정하고 평균 정규화를 하겠는가? 정답: 4번

지금까지 값을 조정하는 법을 배웠고 이를 사용해 gradient descent를 실행한다면 속도가 빨라질 것이고 반복을 덜 하고도 수렴할 수 있을 것이다. 다음 강의에서는 gradient descent를 실제로 잘 사용할 방법에 대해 설명할 것이다.

No.	Title
Week 2-4	Gradient Descent in Practice 2 – Learning Rate

00'00"

이 강의에서는 gradient descent 가 더 잘 활용될 수 있는 도움을 주도록 하겠다. 학습속도 알파에 대한 것인데, 여기 descent 의 업데이트 공식이 있다. 디버깅에 대해 다를 것인데 이는 gradient descent 가 수렴하는지 확인하는 작업을 말한다. 두 번째로 알파를 어떻게 골라야 하는지에 대해서 알아볼 것이다. 여기 gradient descent 가 제대로 작동하는지 확인하는 법이 있다. gradient descent 의 역할은 θ 의 값을 찾아내어 비용함수 $J(\theta)$ 를 최소화하는 값을 찾아내는 것이다. gradient descent 를 활용하여 비용함수 $J(\theta)$ 를 그려보자. x 축은 gradient descent 의 반복의 숫자를 의미하고, 이 알고리즘이 진행됨에 따라 여러분은 이런 순서도를 그릴 수 있다. x 축이 반복의 숫자를 의미한다는 것을 기억해야 한다. $J(\theta)$ 의 순서도에는 x 축, 즉 가로축은 벡터 θ 였지만 여기서는 아니다. 중요한 것은 여기서 우리는 gradient descent 를 100 번 반복한다는 것이다. 그리고 100 번 뒤에 얻게 되는 θ 의 값이 무엇이든지 간에, 그러니까 100 번 시행한 뒤에는 어떤 θ 값을 갖게 될 것이다. 그리고 $J(\theta)$ 를 100 번 시행 후 얻게 되는 θ 값으로 계산하겠다. 여기서 이 $J(\theta)$ 의 값은 세로의 높이이다. 즉, 이 θ 의 값을 위해 100 번의 알고리즘 반복을 시행했다. 그리고 이 점은 gradient descent 를 200 번 반복하고 얻은 값이다.

01'56"

따라서 이 그림이 의미하는 바는 gradient descent 의 반복 마다 나오는 비용 함수의 값이라고 할 수 있다. 이 알고리즘이 알맞게 움직이면 $J(\theta)$ 는 매 반복마다 줄어들 것이다. 이것이 알려주는 것은 내가 그린 그림을 보면, 시간이 지날수록, 그러니까 300-400 번 반복하게 되면 $J(\theta)$ 가 훨씬 줄어든 것을 볼 수 있다. 400 번 이상 하게 되면 곡선이 평평해져서 더 많이는 내려가지 않는 것을 볼 수 있다. 비용함수가 더 이상 내려가지 않아서 알고리즘이 더 이상 수렴하지 않는다. 따라서 이 그림은 여러분의 gradient descent 가 수렴하는지 아닌지를 확인할 수 있는 방법이다.

03'01"

하지만 gradient descent 의 반복의 수렴을 위한 실제 활용은 굉장히 다를 수 있다. 한 예를 보면 30 번의 반복 이후에 수렴하지만 다른 예에는 3000 번, 혹은 3 백만번의 반복이 필요하다. 따라서 얼마나 많은 반복이 필요한가를 미리 알려주는 것은 거의 불가능하다. 그리고 일반적으로 반복 수에 증가함에 따라 이런 그래프 모양으로 gradient descent 를 그린다. 그리고 반복할수록 그 함수의 모양처럼 된다. 이 알고리즘이 수렴한다는 것을 알려주는, 자동 수렴 테스트라고 불리는 이런 알고리즘을 사용하는 것도 가능하다.

03'49"

이것은 아마 아주 평범한 자동 수렴 테스트의 예시일 것이다. 이 테스트는 비용함수 $j(\theta)$ 가 어떤 값보다 작게 줄어들 때, 즉, 한 반복당 10^{-3} 정도로 줄어들 때 수렴한다고 하자. 하지만 이러한 기준을 고르는 것은 꽤 어려운 일이다. 따라서 gradient descent 의 수렴을 확인하기 위해서 나는 이러한 왼쪽에 그린 그림을 이용하며, 자동 테스트를 활용하지는 않는다. 이 같은 숫자를 활용하는 것은 여러분에게 여러분의 gradient descent 가 제대로 작동하는지를 미리 알려줄 수 있다. 자세히 말하면 $J(\theta)$ 가 반복의 수의 함수라고 기입해보자. 그러면 함수가 실제로 증가하기에 이렇게 보일 것이다. 그러면 gradient descent 가 제대로 작동하지 않고 있다는 것을 알게 될

것이다. 그렇다면 학습 속도 알파를 사용해야 할 것이다. 학습 속도가 너무 크다면, 즉, 여러분이 여기서 시작한다면 이 알고리즘은 최소치를 넘어 저 쪽으로 가게 될 것이다. 그리고 여러분의 학습속도가 너무 크다면 또 벗어나 저쪽으로 가게 될 것이다. 그러니 여러분은 여기서 시작해서 천천히 내려가야 한다. 하지만 학습속도가 너무 크다면 gradient descent 는 계속 최저치를 크게 벗어나서 점점 나쁜 값을 얻게 된다. 즉, 점점 더 높은 비용함수 $J(\theta)$ 값을 얻게 될 것이다.

05'32"

따라서 이런 모양의 함수를 그리거나 보게 된다면, 알파의 값을 낮게 두는 것이 중요하다. 물론 여러분의 코드에 버그가 없어야 하는 것도 사실이다. 하지만 너무 큰 알파의 값이 가장 흔한 문제이다. 때때로 여러분의 $J(\theta)$ 함수가 이렇게 보이는 경우도 있을 것이다. 내려갔다가 다시 올라가고 내려가고를 반복하는 경우이다. 이런 경우에도 알파 값을 작게 두는 것이 중요하다. 여기서 증명하지는 않겠지만 비용함수 J 에 대한 다른 가정 하에서, 만약 알파 값이 충분히 작다면, $J(\theta)$ 는 반복할 때마다 계속 줄어들 것이다. 즉, 만약 그렇지 않다면 여러분의 알파가 너무 큰 것이고 작게 설정해야 된다. 하지만 여러분은 학습속도를 너무 작게 두어서도 안 된다. 그러면 수렴하는 데 오래 걸리기 때문이다. 알파가 너무 작은 경우에는, 예를 들어, 여러분이 여기서 시작했다면 정말 수렴하는데 엄청난 시간이 걸릴 것이다.

Q) 친구가 gradient descent 를 세 번 실행한다고 하자. 이때 알파 값은 다음과 같다. 그리고 다음의 A, B, C 와 같은 결과를 얻는다. 이 중 어떤 그림이 알파 값에 대응하는가? 정답: 2 번

06'49"

요약하자면, 여러분의 학습속도가 너무 작으면 수렴하는데 오랜 시간이 걸리고, 너무 크면 $J(\theta)$ 가 매 반복마다 줄어들지 않아서 수렴하지 않을 수도 있다. 아주 천천히 수렴할 수도 가장 흔한 문제는 매번 반복마다 $J(\theta)$ 가 줄어들지 않는다는 것이다. 그리고 이 모두를 디버깅하기 위해서, $J(\theta)$ 를 반복의 숫자의 함수로 설정하는 것이 도움이 될 것이다. 구체적으로 내가 gradient descent 를 실행할 때 하는 일은, 값의 범위를 설정하는 것이다. 즉, gradient descent 를 실행할 때, 알파 값의 범위를 0.001에서 0.01 사이로 시작해 보자. 10 개의 다른 알파가 있고, $J(\theta)$ 를 빨리 줄어들게 하는 값을 골라라. 사실 이 10 개 값을 다 해보지는 않을 것이다. 나는 이 범위의 값을 시도해 보는 것이다. 그리고 0.001을 해보고 그리고 0.003을 얻기 위해 알파를 증가시킨다. 그러면 이는 0.003에서 0.01로 3 배 정도 증가한 것이다. 이것은 이전 값에 비해 대략 3 배 더 크게 시도하는 것이다. 그리고 너무 큰 값이나 너무 작은 값을 찾을 때까지 이러한 범위를 조정할 것이다. 그 이후 가장 클 수 있는 값 혹은 유사한 값을 찾아 낼 것이다. 이렇게 함으로써 좋은 학습속도를 찾아낼 수 있다. 그리고 여러분도 이것을 한다면 gradient descent 를 위한 알맞은 학습속도를 찾아낼 수 있을 것이다.

No.	Title
-----	-------

00'00"

여러분은 이제 다변량을 가진 선형 회귀에 대해 알고 있다. 이 영상에서는 값을 선택하는 방법과 어떤 러닝 알고리즘을 사용할지에 대해 배울 것이다. 어떤 러닝 알고리즘을 사용하느냐에 따라서 효과가 다르기 때문이다. 또한 여러분에게 굉장히 복잡한 비선형 함수에도 회귀 방법을 적용할 수 있게 해주는 다항식 회귀에 대해 말하고 싶다.

00'28"

집 가격을 측정하는 예로 시작해보자. 여러분은 두 가지 값을 가지고 있는데, 그것은 집의 정면과 집의 높이이다. 여기 우리가 팔려고 하는 집이 있다. 집의 정면 길이는 이 길이로 정의 되는데, 기본적으로 집의 부지의 폭이나 길이이다. 그리고 높이는 집의 깊이이다. 여기가 정면, 여기가 높이다. 여러분은 정면이 x_1 이고 높이가 x_2 인 선형 회귀를 그릴 수 있을 것이다. 하지만 여러분이 선형 회귀 함수를 적용하면 이런 값을 사용하지 않아도 된다. 여러분이 할 수 있는 것은 새로운 값을 만들어내는 것이다. 그래서 이 집의 가격을 측정하고자 하고, 집의 넓이를 결정짓는 것이 이 구역인지 내가 보유한 구역인지 결정할 것이다.

01'32"

그래서 정면 길이와 높이를 곱한 다른 값 x 를 만든다. 이것은 곱셈 기호이다. 정면 길이와 높이의 곱인데, 왜냐하면 이것이 내가 보유한 땅의 넓이이고 그렇다면 나는 내가 가진 땅을 하나의 값으로만 사용하는 이 같은 새로운 가정을 세울 것이다. 이 사각형의 땅은 길이의 산물이다. 그러니 정면 길이와 높이를 사용해서 계산하기 보다 통찰력을 발휘하면 기준보다 더 진보된 새로운 모델을 만들 수 있다.

02'17"

이것은 여러분의 값을 선택하는 일과 매우 밀접한 관련이 있는데, 이는 다항식 회귀라 불린다. 여러분이 집 가격을 나타내는 다음의 데이터 셋을 가지고 있다고 하자. 그러면 여기에 적합한 각기 다른 모델이 몇 개 있을 것이다. 먼저 다음과 2 차 모델을 생각해 볼 수 있다. 이 값은 직선은 아니지만 데이터에 꽤 잘 들어맞는다. 이런 2 차 모델에 끼워 맞추고 싶다면 여러분은 크기를 생각해야 하고, 이것은 이런 모양으로 데이터에 맞아 떨어질 것이다. 그러나 집 크기가 매우 커질수록 집 가격이 떨어진다고 생각하지는 않기 때문에, 집의 크기가 특정 사이즈 이상으로 커질 때 가격이 낮아지는 2 차 함수 모델은 잘 맞지 않는다고 생각할 것이다. 이 때 우리는 다른 다변수를 가진 선형 회귀 모델을 생각해 낼 것이고, 예를 들어, 3 차 함수를 생각해 낼 것이다. 그러면 이런 모양의 함수를 얻을 수 있다. 그리고 이 녹색 함수는 결국 감소하지 않기 때문에 이 데이터와 더 잘 맞는다고 할 수 있다.

03'20"

어떻게 이 같은 모델을 우리의 데이터에 맞출 수 있을까? 다변량 선형 회귀를 사용해서, 즉 우리의 알고리즘에 약간 변형을 가해서 맞출 수 있다. 우리가 사용할 가설은 다음과 같다. $h_{\theta}^{(x)} = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2 + \theta_3 \cdot x_3$ 이다. 우리가 이 3 차함수에 맞추고 싶다면, 여기 녹색 상자로 표시한 식을 사용하고 싶다면, 그러니까 이 집의 가격을 측정하고 싶다면 그것은 여기서 보이는 것과 같은 식으로 나타낼 수 있다. 이렇게 2 가지 식을 연결하는 자연스러운 방식은, 바로 x_1 을 집의 크기로 두고 x_2 를 집 크기의 제곱, x_3 를 세제곱으로 두는 것이다. 이렇게 세 가지 값을 선택하고, 기계 선형 회귀함수를 적용해서 나의 데이터에 알맞은 3 차함수를 만들어 낼 수 있다.

04'35"

한가지 더 강조하고 싶은 것은 이런 값들을 고르면 값 조정이 아주 중요해진다는 것이다. 집의 크기가 1에서 1000 까지고, 집 크기의 제곱이 1에서 100 만이고, 3 번째 값인 크기의 세제곱은 1에서 10^9 가 된다고 하자. 따라서 이렇게 서로 다른 값들이 도출되었기 때문에 gradient descent 를 사용해서 이 값들을 비교 가능한 범위에 놓으려면, 이 값들을 조정하는 작업이 필요하다.

05'23"

이제 마지막으로 값 선택의 폭이 얼마나 넓은지를 살펴보자. 이전에 우리는 제곱 모델이 어째서 최선이 아닌지에 대해서 배웠다. 왜냐하면 잘 맞는 것처럼 보였지만 다시 내려가기 때문에 집 가격과 관련하여 우리가 원했던 모델이 아닌 것이다. 하지만 3 차 모델로 가기 전에 다른 선택의 여지가 있을 수 있다. 그리고 선택의 여지는 굉장히 많다. 여기 나와있는 수식을 보라. $h_{\theta}^{(x)} = \theta_0 + \theta_1 \cdot (\text{size}) + \theta_2 \cdot \sqrt{\text{size}}$ 이다. 이러한 함수가 될 것이고, 이 식을 취할 수 있게 하는 x 값이 있을 수 있는데, 그렇게 되면 다음과 같은 함수가 된다. 이 함수는 다음과 같은 모양으로 올라가면서 평평해 지겠지만 내려오지는 않는다. 이 경우처럼, 제곱근 함수가 이 데이터에 잘 들어맞고, 이런 다른 값을 선택함으로써 여러분은 더 나은 모델을 생각해 낼 수 있다.

Q) 여러분이 집 값을 예측할 때 집의 크기를 함수로 둔다고 하자. 여러분이 사용할 모델은 다음과 같다. $h_{\theta}^{(x)} = \theta_0 + \theta_1 \cdot (\text{size}) + \theta_2 \cdot \sqrt{\text{size}}$. 크기의 범위가 1에서 1000 피트라고 하고 $h_{\theta}^{(x)} = \theta_0 + \theta_1 \cdot x_1 + \theta_2 \cdot x_2$ 모델을 실행할 것이다. 마지막으로 평균 정규화 없이 값 조정을 한다고 하자. x_1, x_2 , 값으로 무엇을 사용해야 할까? 정답: 3 번

06'35"

이 영상에서 우리는 다향, 즉 2 차함수나 3 차함수를 여러분의 데이터에 맞추는 법을 알아보았다. 또한 우리는 어떤 값을 사용해야 할지, 예를 들어 정면 길이나 높이 대신 이 둘을 곱해서 집의 부지를 나타내는 새로운 값을 만들어내는 법을 배웠다. 어떤 면에서 이것은 혼란스러워 보일 수 있다. 왜냐하면 많은 선택의 여지가 있기 때문에 어떤 선택을 해야 할지 잘 모를 수 있다. 향후 어떤 값을 사용할지 자동으로 골라주는 알고리즘을 배우게 될 것이다. 여러분이 2 차함수를 원하든 3 차함수를 원하든 말이다. 하지만 우리가 그 알고리즘을 얻기 전까지는 어떤 값을 선택할지 고르는 데 신중을 기해야 할 것이고, 여러분의 데이터에 알맞은 복잡한 함수를 위해

생각해내야 할 것이다. 여러분의 다향 함수를 선택할지 일차함수를 선택할지에 대한 알맞은 통찰력이 여러분의 데이터에 알맞은 모델을 만들어 낼 것이다.

No.	Title
Week 2-6	Normal Equation

00'00"

이번 강의에서는 선형 회귀 문제에 필요한 표준 방정식에 대해 이야기해보겠다. 이는 매개 변수 θ 의 최종값을 구하는 방법을 제시해 줄 것이다. 예시를 먼저 보면, 지금까지 우리가 활용한 선형 회귀 알고리즘은 기울기 감소다. 비용함수 $J(\theta)$ 를 최소값을 구하기 위해 반복 알고리즘을 여러 단계, 즉 여러 번 반복된 기울기 하강을 거쳐 전역 최소값에 모이게 된다. 이와 대조적으로, 표준 방정식은 θ 를 분석적으로 구하는 방법을 제공한다. 그래서 이러한 반복 알고리즘을 구현하는 대신, 한 번에 θ 최적값을 바로 구하게 된다. 그러니까 한 단계만 거치면 여기 이 최적값을 구할 수 있게 되는 것이다.

00'50"

표준 방정식은 장단점이 각각 있는데, 언제 이를 활용하는지 살펴보기 전에, 이것이 과연 어떤 방정식인지에 대해 먼저 살펴보자. 이 방정식을 설명할 예시로 여기 아주 간단한 비용함수 $J(\theta)$ 가 있다. 이는 단순히 실수 θ 함수다. 이제 θ 가 실수 또는 가공되지 않은 값이라고 하자. 이는 벡터라기 보다는 그저 숫자다. 여기에 비용함수가 있는데 이는 실제 값 매개 변수 θ 의 이차함수이고 이는 이렇게 나타낼 수 있다. 그렇다면 이차 함수의 최솟값을 어떻게 구하는가? 미적분에 대해 조금이라도 알고 있다면, 함수의 최소값을 구하는 방법은 미분을 적용해, 도함수를 0으로 놓는 것이다. 그러므로 J 의 도함수를 가지고 이렇게 적어보면, 어떤 공식이 성립될 것이고, 이를 다 적지는 않겠다. 그리고 도함수를 0으로 놓으면, θ 값이 나오고 이는 $J(\theta)$ 의 최소값인 것이다. 이는 데이터가 그저 실수일 경우의 간단한 예시였다.

02'02"

하지만 우리가 관심을 가지고 있는 문제에서는, θ 는 더 이상 실수가 아니라 $(n+1)$ 차원 매개 변수 벡터이고, 비용 함수 J 는 이 벡터값 또는 θ 0부터 θ m 까지의 함수다. 비용 함수는 이렇게 생겼고, 제곱 비용함수가 오른쪽에 있다. 그렇다면 어떻게 비용 함수 J 의 최솟값을 구할 수 있을까? 미적분에서 이를 해결하는 방법 중 하나는 $J(\theta)$ 의 모든 매개변수에 대하여 $J(\theta)$ 에 편미분을 적용한 다음, 이를 모두 0으로 설정하는 것이다. 이렇게 한 후, $\theta_0, \theta_1, \dots, \theta_N$ 까지 값을 구하면, θ 값을 구하여 비용함수 $J(\theta)$ 의 최솟값을 구할 수 있다. 여러분이 미적분을 통해 매개변수 θ_0 부터 θ_N 까지 계산한다면, 결국에는 도함수를 다루게 된 것이다.

03'00"

이제부터는 다음 단계로 넘어갈 텐데, 길고 이미 다룬 도함수를 설명하지는 않을 것이다. 대신에 여러분이 이러한 과정을 구현할 때 알아야 할 내용을 설명하려고 한다. 그래서 θ 값을 계산하여 이는 편미분에 적용되어 이는 0과 같아진다. 아니면 이를 대신하여, θ 값은 비용함수 $J(\theta)$ 의 최솟값이 된다. 내가 전에 설명한 것이 미적분에 대해 알고 있는 사람에게 더 이해하기 쉬웠다는 생각이 들었다. 하지만 미적분에 대해 알지 못한다고 해도, 걱정하지 않아도 된다. 이제 이러한 알고리즘을 구현하고 작동하기 위해 필요한 내용을 설명해주려고 한다.

03'41"

구체적인 예를 들어보자. $m=4$ 학습 예시가 여기 있다. 이와 같은 표준 방정식을 구현하기 위해 다음과 같은 과정을 거칠 것이다. 먼저 데이터 세트를 가지고, 여기 보면 네 개의 학습 예시가 있다. 이번 경우에는 여기 4개의 예시만이 내가 가진 데이터의 전부라고 가정해보자. 이 데이터 세트를 가지고 여기에 x_0 칸을 하나 추가하고 이는 항상 1을 값으로 가진다. 이번에는 X 라는 행렬을 만들어 볼 것이다. 이 행렬은 네 학습 데이터에 모든 feature를 포함한다. 여기에 모든 해당 값이 있고, 이 숫자를 행렬 X 로 가져갈 것이다. 한 열의 데이터를 하나씩 이렇게 옮겨오고, 여기 y 도 똑같이 해줄 것이다. 이렇게 y 값을 가져와 이제 벡터를 만들어, 이를 벡터 y 라고 부르겠다. 그렇다면 X 는 $m \times (n+1)$ 차원 행렬이 되고, y 는 m 차원 벡터가 된다. 여기서 m 은 학습 예시의 숫자이고, n 은 feature의 수이고, 여기 추가로 x_0 이 들어갔기 때문에 $n+1$ 이 된다. 마지막으로 행렬 X 와 벡터 y 를 가지고 이 수식을 연산해보면, 이 값이 바로 비용 함수의 최소값인 θ 값이 되는 것이다. 이번 슬라이드에서 많은 내용을 배웠고, 데이터 세트에서 하나의 특정한 예시를 보여주었다. 아직 이에 대해 확실히 이해하지 못한 이들을 위해 이를 좀 더 일반화한 형태로 보여주고, 이 방정식에 대해 좀 더 설명하고자 한다.

05'59"

아까 배운 예시를 일반화 하면, 여기 M 학습 예시가 있고, X_1, Y_1 부터 X_n, Y_n 그리고 n feature들이 있다. 학습 예시의 하나인 $x(i)$ 는 이렇게 벡터처럼 생겼는데, 이는 $n+1$ 차원 feature 벡터이다. X 행렬을 만드는 방법은 설계 행렬이라고도 불리는 데 다음과 같다. 각각의 학습 예시는 이와 같이 벡터, 그러니까 $n+1$ 벡터의 feature이 된다. 이 설계 행렬을 만들기 위해서 행렬을 이렇게 그리면 된다. 그 다음으로는 첫 번째 학습 예시를 가져와 이것은 벡터니까 이의 전치행렬을 이렇게, 옆으로 길게 내가 그린 행렬 첫 번째 행에 써준다. 또, 두 번째 학습 예시를 가져와 이의 전치 행렬을 새로운 행렬 두 번째 행에 써주고, 마지막 예시까지 적어준다. 이렇게 전치 행렬로 옮겨주면, 이것이 행렬 X 의 마지막 행이다. 그렇게 되면, 행렬 $X, m \times (n+1)$ 행렬이 완성된다.

07'19"

좀 더 자세히 살펴보면, 내가 항상 1인 x_0 외에 feature를 오직 한 개만 가지고 있다고 하자. 만약 x_i 가 이와 같은 행렬이라고 가정하면, 여기 1은 x_0 과 같고, 이것은 주택 크기와 같은 다른 실제 feature이 되고, 설계 행렬 X 는 이것과 같다. 첫 번째 행은 이것을 가지고 와서 전치하면 된다.

1과 $x(1)1$ 을 써주고, 1, $x(1)2$ 그리고 쭉 내려가서 또 1, $x(1)m$ 이렇게 된다. 이렇게 $m \times 2$ 차원 행렬이 만들어졌다. 이것이 바로 행렬 X 를 설계하는 방법이다. 그리고 벡터 y 는, 때로 이렇게 화살표를 표시하는데 이는 벡터라는 의미이다. 하지만 보통은 그냥 y 만 쓴다. y 벡터는 학습 세트에 있는 모든 라벨 그러니까 모든 정확한 주택 가격을 M차원 벡터에 넣으면 얻을 수 있다. 그것이 바로 y 벡터다.

08'33"

이렇게 행렬 X 와 벡터 Y 를 만들어봤다. 이번에는 다음과 같은 식을 통해 연산을 해보겠다.

Q) 다음과 같은 학습 테이블이 있다.

나이 / 신장(cm) / 몸무게 (kg)

함수를 통해 다음 예시의 나이와 신장을 바탕으로 한 아동의 몸무게를 예측하고자 한다.

몸무게 = $\theta_0 + \theta_1$ 나이 + θ_2 신장 일 때,

행렬 X 와 y 은? 정답 4번

이번 설명을 통해 이 방정식을 이해할 수 있게 되어 직접 적용할 수 있기를 바란다. 다시 말하자면, 여기 $X'(1/X)$ 은 무엇인가? $X'(1/X)$ 는 행렬 $X'X$ 의 역행렬이다. 그러니까 세트 A가 $X' \times X$ 와 같다고 하면, X' 은 행렬이고, $X' \times X$ 은 또 다른 행렬을 제공하며, 이는 행렬 A가 된다. 그렇다면 $X'(1/X)$ 는 행렬 A의 역행렬이다. 따라서 이것은 A의 역행렬이 된다. 이렇게 연산하면 된다. $X'X$ 를 먼저 계산하고 그 다음에 이를 역행렬로 연산하면 된다.

09'32"

옥타브에 대해 아직 설명하지 않았다. 이후 강의에서 설명하게 되겠지만, 옥타브 프로그래밍 언어 또는 비슷한 종류에서, 매트랩 프로그래밍 언어도 매우 비슷하다. 이와 같은 수식을 연산하기 위해서는 다음과 같이 써주면 된다. 옥타브에서 X' 은 X^T 를 의미한다. 그러므로 이 빨간 박스 안에 있는 것은 $X^T \times X$ 를 의미한다. `pinv`는 역행렬을 연산하는 함수다. 따라서, 이는 $(X^T \times X)$ 의 역행렬이 되는 것이고, 이를 x' 로 또 곱해주고 y 를 곱해주면 이 공식은 모든 연산 과정이 끝이 나는 것이다. 이를 증명하지는 않았지만, 수학적으로 이를 보여줄 수는 있다. 하지만 여기서 설명하지는 않을 것이다. 이 공식은 θ 의 최적값을 제공한다. 그리고 θ 가 이것과 같다고 할 경우 그리고 θ 값은 새로운 회귀를 위한 비용 함수 $j(\theta)$ 의 최소값이 되는 것이다.

10'42"

지난 번 강의에서 나온 내용을 하나 더 설명하고 넘어가겠다. feature 스케일링과 feature를 각각의 값과 비슷한 범위에 놓는 방법에 대해 이야기했다. 만약 표준 방정식을 활용한다면, feature 스케일링은 꼭 필요하지 않게 되고, 어떤 feature X_1 이 0과 1사이에 있다면, x_2 는 0과 1000 사이에 있고, x_3 는 0부터 10^{-5} 사이에 있다고 하면, 표준 방정식을 활용한다면 이러한 식도 성립하고, feature 스케일링을 적용할 필요가 없다. 하지만 기울기 감소에서는 feature 스케일링이

중요하다.

11'28"

그렇다면 언제 기울기 감소를, 또 언제 표준 방정식을 활용해야 하는 걸까? 여기에 각 방식의 장단점이 있다. 여기에 m 학습 예시와 n 개의 값(feature)이 있다고 가정하자. 기울기 감소에서 단점은 학습률 알파를 선택해야 한다는 것이다. 그리고 이는 각기 다른 학습률을 가지고 이를 여러 번 돌려봐야 하며 가장 적합한 것을 찾아야 한다는 것이다. 그래서 작업할 게 많아진다. 또 다른 단점은 반복이 많다는 점이다. 내용에 따라 다르지만, 속도가 느려진다. 이에 대해서는 잠시 후에 더 자세히 알아보자. 표준 방정식에서는 학습률 알파를 선택할 필요가 없다. 그렇기 때문에 편리하고, 구현하기 간편하다. 이를 돌리면 되는데, 보통 잘 돌아간다. 이를 반복할 필요도 없고, $j(\theta)$ 함수를 그릴 필요도, 수렴을 확인할 필요도 없이 이러한 모든 작업을 거쳐야 할 필요가 없다. 지금까지 보면 표준 방정식이 더 장점이 많아 보이는데 이번에는 바꿔서 표준방정식의 단점과 기울기 하강의 장점을 알아보겠다. 기울기 하강은 꽤 잘 구현되는데, feature 수가 많을 때도 그렇다. feature이 수백만 개가 된다고 해도 기울기 하강은 잘 돌아갈 것이고, 이는 효율적이다. 이는 꽤 합리적이다. 반면에 표준 방정식은 매개 변수 데이터를 풀기 위해서는 이러한 식을 풀어야 한다. 이 부분은 n 개의 feature이 있으면, $n \times n$ 행렬이 된다. 여기서 이 것을 곱해서 이것이 몇 차원인지 알아내면, $X^T X$ 는 $n \times n$ 행렬이 되고, 여기서 n 은 feature의 수가 된다. 그리고 거의 모든 연산을 구현하는데 있어서, 역행렬 비용은 대략 행렬의 차원의 세제곱만큼 오르게 된다. 따라서 역행렬 비용을 연산하기 위해서는 대략적으로 n 의 세제곱이 된다. 그리고 n 세제곱보다는 약간 빠르지만, 이는 우리가 원하는 것을 얻기에는 충분하다.

13'35"

만약 n , feature의 수가 매우 커지면, 이러한 양을 연산하는 것이 느려질 수 있고, 그렇게 되면 표준 방정식은 훨씬 더 느려질 수 있다. 그러니까 n 의 숫자가 크면 기울기 하강을 이용할 것이고, n 의 수가 상대적으로 적을 때는 매개 변수를 푸는 데 표준 방정식이 더 적합할 것이다. 그렇다면 여기서 숫자가 크고 작고는 어느 정도를 말하는 걸까? 만약 n 이 100이라고 하면, 100×100 행렬은 오늘날 연산 표준에는 충분할 것이다. 1000이 되도 아직 표준 방정식 방법을 쓸 것 같다. 1000×1000 행렬은 오늘날 컴퓨터에서는 꽤 빠르게 작동된다. 10000의 경우에는 고민을 하기 시작할 것이다. 10000×10000 행렬부터는 조금씩 느려지기 시작할 것이고, 아마도 기울기 하강 쪽으로 좀 기울어 질 것이다. 하지만 아직 완전히 기운 것은 아니다. $n = 10000$ 이고 10000×10000 행렬도 연산할 수 있을 것이라고 생각한다.

13'31"

하지만 이보다 숫자가 더 커지면, 아마도 기울기 하강 쪽을 택할 것이다. $N=100000$ 이 되면, 1000000×1000000 행렬을 역행렬하는 것은 매우 많은 비용을 필요로 할 것이고, 그렇게 되면 당연히 기울기 하강을 택하게 될 것이다. 따라서 feature 세트가 얼마나 클 때부터 기울기 하강을 선택해야 할지는 정확한 숫자를 말하기는 어렵다. 하지만 나라면 10000정도가 되면 기울기 하강을 선택하게 되거나 아니면 우리가 다루게 될 다른 알고리즘을 선택하겠다.

15'03"

요약하자면, feature의 수가 너무 크지 않다면, 표준 방정식이 매개 변수 θ 값을 구하는데 최적의 방법이 될 것이다. 다시 말해, feature의 수가 1000 미만일 때라면 보통 기울기 하강보다는 표준 방정식을 택한다는 것이다. 다음 시간에 소개할 내용에 대해 언급하자면, 우리가 더 복잡한 학습 알고리즘을 배우게 될 것이다. 예를 들어, 로지스틱 회귀 알고리즘과 같은 분류 알고리즘에서, 곧 이와 같은 알고리즘을 배우게 될 것이다. 표준 방정식 방법은 더 복잡한 학습 알고리즘에서는 작동하지 않는다. 그 때에는 기울기 하강에 의존해야 한다. 그러니까 기울기 하강은 알아두면 매우 유용한 알고리즘이다. 선형 회귀에서는 feature의 수가 많아질 것이고, 이외에도 다른 알고리즘에서도 그럴 것이기 때문에 이 과정에서 보게 될 것이다. 그와 같은 알고리즘에서는 표준 방정식은 적용되지 않고 작동하지 않는다. 하지만 이렇게 선형 회귀의 특정한 모델에서, 표준방정식은 기울기 하강보다 빠른 연산을 제공하는 대안이 될 수 있다. 따라서, 알고리즘의 내용에 따라, 또는 문제의 내용에 따라 그리고 feature의 수에 따라서 두 알고리즘 모두 잘 활용될 수 있을 것이다.

No.	Title
Week 2-7	Normal Equation Noninvertibility

00'00"

이번 강의에서는 표준방정식과 비가역행렬에 대해 알아보겠다. 이는 좀더 고차원적인 개념인데 자주 질문을 받는 내용이다. 그래서 여기서 이를 설명해보고자 한다. 하지만 말했다시피 고차원적인 개념이라 이번 강의는 선택 사항이라고 생각해도 좋다. 나중에 이와 같은 개념을 마주하게 되면, 그 때 이해하기 수월해 질 것이다. 하지만 만약 아직 표준 방정식과 선형 증가를 이해하지 못한 경우에는 이를 먼저 짚고 넘어가야 한다.

00'33"

문제는 이러하다. 여러분 중에는 선형 대수가 더 익숙한 이들이 있을 수도 있는데, 몇몇 학생들이 내게 이렇게 질문을 했다. 이러한 식을 연산할 때, 행렬 X 가 역변환이 불가하면 어떡하냐는 것이다. 선형 대수에 대해 어느 정도 알고 있다면, 몇 안 되는 행렬만이 역변환이 불가하다는 것을, 몇몇 행렬만 역행렬을 가지고 있지 않다는 것을 알고 있을 것이다. 우리는 이를 비가역행렬이라고 부른다. 또는 특이 행렬, 아니면 축퇴 행렬이라고도 한다. $X^T X$ 가 비가역성을 가질 확률은 그리 높지 않다. 또한 옥타브에서 이를 연산 데이터에 입력하면, 이는 사실 연산이 된다. 이렇게 되면 좀 전문용어가 등장하게 되는데 너무 구체적으로는 설명하지 않겠다. 옥타브는 역행렬 함수에서 두 가지 함수를 가진다. 하나는 `pinv` 또 하나는 `inv`이다. 이 둘의 차이점은 조금 전문적이지만, 하나는 의사역행렬이고, 하나는 역행렬이다. 하지만 `pinv` 함수를 사용하기만 한다면

이를 수학적으로 표기할 수 있고, $X^T X$ 가 비가역행렬이라 할지라도 이 θ 값을 연산해 낼 수 있다.

01'50"

이 둘의 차이점은 무엇일까? $Pinv$ 는 뭐고, inv 는 뭘까? 이 또한 어떻게 보면 고차원적인 수연산 개념이라서 이를 설명하고 싶지는 않다. 하지만 이번 강의는 선택 사항이라고 생각하기 때문에, $X^T X$ 가 비가역행렬이라는 것의 의미를 간단하게 설명하고 넘어가겠다. 선형 대수학에 대해 들어본 적이 있으면 아마 더 흥미로운 주제가 될 것이다. 이를 수학적으로 증명해 보이지는 않겠다, 하지만 $X^T X$ 가 비가역행렬이라면, 보통 그 이유는 주로 두 가지로 나뉜다. 첫 번째 문제는 학습 문제에서 중복된 feature를 가지고 있는 것이다. 즉, 주택 가격을 예측하려고 하는데, x_1 이 주택의 크기를 제곱피트로 나타낸 것이고, x_2 는 이를 제곱평방미터로 나타낸 것이다. $1m =$ 대략 3.28 피트이므로, x_1 는 3.28 의 제곱 곱하기 2가 된다. 선형 대수학을 배웠다면, 이를 대수학을 통해 설명하고자 한다면, 이 두 feature이 이와 같은 선형 방정식과 연관이 있다는 것이다. 따라서 $X^T X$ 가 비가역행렬이라는 것이다.

03'08"

$X^T X$ 가 비가역행렬이 되는 두 번째 경우는 많은 feature를 가진 학습 알고리즘을 돌리려고 할 때 발생한다. 즉, m 이 n 보다 작거나 같으면, 예를 들어 $m = 10$, $n = 100$ 이라는 학습 예시가 있다고 하자. 그리고 이 매개 변수를 θ 에 대입하려고 한다. $n + 1$ 이므로 이는 101 차원이다. 그럼 10개의 학습 예시를 가지고 101 매개 변수를 대입하려고 하는 것이다. 이는 결국 연산이 되기는 하겠지만, 항상 좋은 방법은 아니다. 나중에 보게 되겠지만, 10개의 예시만 가지고 100 또는 101개의 매개 변수에 대입하려고 하니 데이터가 충분하지 않은 것이다. 이후의 강의에서 왜 이것이 많은 매개 변수에 대입하기 너무 적은 데이터였는지를 알아보겠다. 하지만 대체로 이럴 경우에 우리는 m 이 n 보다 작은 경우에는 feature 몇 개를 삭제하거나, '조직화'라고 불리는 기술을 활용하기도 하는데, 이는 이후에 다루도록 하겠다. 간단히 말하자면, 이는 상대적으로 작은 학습 세트를 가지고 있어도 많은 매개 변수를 대입하고, 많은 feature를 활용할 수 있게 해준다. 하지만 이는 이번 과정 후반에 등장하게 될 주제다.

04'26"

하지만 정리하자면 이후에 $X^T X$ 가 특이 행렬이거나 또는 비가역행렬인 경우를 발견하게 되면 이와 같이 하면 된다. 먼저 feature를 살펴보고, feature의 수가 x_1 , x_2 처럼 반복되는지 살펴보자. 이는 선형종속이 되거나 서로의 선형 함수가 된다. 이처럼 feature의 수가 반복될 경우에는 그 중에 하나를 삭제하면 된다. feature 두 개나 필요 없기 때문이다. 이중에 하나의 feature를 삭제하면, 비가역행렬 문제를 해결해 줄 것이다. 그러니까 먼저 feature를 생각해보고 이를 확인하면서 혹시 반복되는 게 있다면 반복되는 feature이 나타나지 않을 때까지 계속 삭제해주면 된다. 만약 feature가 반복되지 않는다면, feature의 수가 너무 많은지를 살펴볼 것이다. 만약 그렇다면, feature 몇 개를 지우거나, 더 적은 feature만 활용하려고 할 것이다. 또는 우리가 나중에 배울 조직화를 고려해볼 수도 있다.

05'44"

여기까지가 표준 방정식과 $X^T X$ 가 비가역행렬인 경우를 살펴봤다. 하지만 이와 같은 문제는 거의 마주치게 되지 않을 문제고, 옥타브에서는 p 를 활용해 의사역행렬이라 불리는 p 를 함수를 활용하여 다른 선형 대수학 라이브러리를 활용하여 문제를 해결하는 것이다. 하지만 $X^T X$ 가 비가역행렬인 경우 자체가 그렇게 흔한 일은 아니므로 선형 회귀를 구현하는 데 있어서 문제가 되지는 않을 것이다.

No.	Title
Week 2-8	Basic Operations

00'00"

이제 여러분은 기계 학습에 대해 많은 것을 배웠을 것이다. 이번 강의에서는 프로그래밍 언어 옥타브에 대해 이야기해보고자 한다. 이를 통해 지금까지 다뤄본 학습 알고리즘뿐만 아니라 앞으로 배우게 될 알고리즘 또한 매우 빠르게 구현할 수 있을 것이다. 예전에는 C++, 자바, 파이썬, 넘피 그리고 옥타브와 같은 다양한 프로그래밍 언어를 가지고 기계 학습을 강의했었다. 그랬더니 학생들이 옥타브처럼 상대적으로 고급 프로그래밍 언어를 배웠을 때, 가장 생산적이고 빠르게 배울 수 있으며, 알고리즘의 프로토타입을 가장 빠르게 배울 수 있다는 것을 알게 되었다.

00'40"

실리콘밸리에서도 종종 볼 수 있는데, 만약 학습 알고리즘의 대규모로 배열하려고 하면, 보통 프로토타입을 하고, 프로그래밍 언어는 옥타브를 활용한다. 옥타브는 훌륭한 프로토타이핑 언어이다. 따라서 학습 알고리즘이 빨리 작동되도록 할 수 있다. 대규모 배열이 필요할 때만 활용하면 된다. 그리고 나서 다시 C++, 자바로 알고리즘을 다시 구현하는 데 시간을 활용하면 된다. 왜냐하면 지금까지 우리가 배운 강의에서는 시간을 개발하는 것이다. 바로 여러분의 시간이다. 기계학습에서 시간은 무척 귀중하다. 따라서 옥타브를 활용하여 학습 알고리즘을 더 빠르게 구현할 수 있다면, 전체적으로 시간을 절약할 수 있는데, 먼저 옥타브로 알고리즘을 개발하고, 그 다음에 그 아이디어가 작동할 경우에 C++, 자바 등으로 다시 재 구현하면 된다.

01'28"

기계 학습에서 사람들이 가장 흔히 사용하는 프로토타이핑 언어는 옥타브, 맷랩, 파이썬, 넘피, 그리고 R이 있다. 옥타브는 오픈 소스이기 때문에 좋다. 그리고 맷랩도 잘 구현되지만 비싸다는 단점이 있다. 하지만 맷랩 사본에 접근 가능하면, 이 강의에서 맷랩을 활용할 수 있다. 파이썬, 넘피 또는 R을 알고 있는 사람들도 있을 것이다. 이를 사용하는 사람들을 본 적이 있다. 하지만 대부분 개발이 약간은 느려지는 것을 발견할 수 있었다. 파이썬, 넘피와 같은 언어의 문법은 옥타브의 문법보다 약간 투박하다. 그래서 스타터 코드를 옥타브에서 공개하는 것이다. 그래서 넘피나 R을 가지고 다음 문제를 풀어보지 않기를 강력히 바라며 대신 옥타브를 활용하기를 추천한다.

02'24"

이번 강의에서는 명령어들을 매우 빠르게 짚고 넘어갈 것이다. 따라서 이번 강의의 목표는 옥타브에서 구현할 수 있는 명령어의 범위를 보여주는 것이다. 강의 웹사이트에서 강의에 대한 모든 내용을 자료로 제공하고 있으니 강의를 본 후에 찾고 싶은 명령어가 있을 때 그 자료를 참고하면 될 것이다. 그러니까 먼저 강의 영상을 보고 나서, 옥타브를 컴퓨터에 설치한다. 그리고 나서 강의 웹사이트로 가서 해당하는 날짜의 자료를 다운 받아서 관심 있는 명령어를 직접 입력해서 옥타브에 구현해보기를 추천한다. 그리고 컴퓨터에 구현이 되면 이를 활용할 수 있게 되는 것이다. 그럼 이제 본격적으로 시작해보자.

03'15"

여기에 윈도우 데스크탑이 있다. 옥타브를 먼저 실행시키겠다. 옥타브가 실행되었다. 이게 내 옥타브 프롬프트다. 먼저 옥타브에서 할 수 있는 가장 기본적인 연산을 보여주겠다. $5 + 6$ 를 입력하면 답은 11이 나오고, $3 - 2$, 5×8 , $1/2$, 2^6 는 64이 나온다. 이러한 것은 가장 기초적인 수학 연산이다. 논리 연산 또한 가능하다. 1 은 2와 같다라고 입력하면 이것은 거짓이다. 여기 %는 코멘트를 의미한다. 그래서 $1 == 2$ 는 거짓이라고 평가하고, 이는 0 이라고 나온다. 1은 2가 아니다라고 입력하면, 이는 참이다. 그래서 이는 1이라고 나온다. 같지 않다는 \sim 이렇게 표시한다. !=라고 다른 프로그래밍 언어에서는 표현하기도 하지만 여기서는 아니다. 또 다른 논리 연산 $1 \sim = 0$ 가 있다. 여기서는 & 기호를 두 개 사용했고 이는 AND 를 의미한다. 이는 거짓이다. $1 \parallel 0$ 은 OR 라는 뜻이고 참이라고 나온다. xor(1, 0) 를 입력하면, 이는 1로 평가된다. 여기 왼쪽에 옥타브 3.2.4.exe:11 은 디폴트 옥타브 프롬프트다. 이는 옥타브가 어떤 버전인지 등을 나타낸다. 이러한 프롬프트를 보고 싶지 않으면 조금 난해한 명령 PF 인용 $>>$ 를 활용하여 프롬프트를 바꿔주면 된다. 이 표시는 가운데 있는 문자열이다. $>>$ 를 입력하고 한 칸 띄워준다. 보통 옥타브 프롬프트가 이렇게 보이도록 한다. 이제 엔터키를 입력하면, 이런 실수. 다시 이렇게 PS1을 입력하면. 이제 옥타브 프롬프트가 $>>$ 로 더 보기 좋게 바뀌었다.

05'10"

이번에는 옥타브 변수에 대해 이야기해보자. 변수 $A = 3$ 이라고 하자. 그러면 이제 $A = 3$ 이 된다. 변수를 지정하고 싶지만 그 결과를 출력하고 싶지는 않을 때, 세미콜론을 입력하면, 결과가 출력되지 않는다. 이렇게 하면, 아무것도 입력되지 않는다. 반면에 $A = 3$ 는 이렇게 출력이 되고, $A = 3;$ 는 아무것도 출력이 되지 않는다. 문자열도 다룰 수 있다. $B = hi$ 로 입력하고 B를 입력하면 이렇게 B의 변수를 출력한다. B는 문자열 hi가 되었고, $C = (3 > = 1)$ 이라고 하면 이제 C는 참이 된다. 변수를 출력하거나 표시하고 싶으면, 이 방법을 통하면 된다. $a = pi$ 라고 하자. a를 출력하고 싶으면, 이제 이렇게 a만 입력하면 된다. 좀 더 복잡한 프린팅으로는 DISP 명령어가 있다. 이는 display를 의미한다. disp(a)를 입력하면 이렇게 똑같이 나온다. 문자열도 디스플레이 할 수 있다. disp(sprintf('2 decimals: %0.2f', a))를 입력하면 2 decimals : 3.14가 나온다. 이는 예전 방식 C 문법이다. 예전에 C를 프로그래밍 해본 적 있는 사람이라면, 이것이 바로 프린트 스크린하는데 사용하는 문법이다. Sprintf는 2 decimals : 3.14 문자열을 생성한다. %0.2f 는 a를 대체하는 것으로 소수점 뒤의 두 숫자를 나타낸다. DISP는 문자열 DISP는 Sprintf 명령어를 통해 문자열을

생성한다. 여기 `Sprintf`, `Sprintf` 명령어다. 그리고 `DISP`는 이렇게 문자열을 디스플레이 할 것이다. 또 다른 예를 들어보겠다. `disp(sprintf('6 decimals: %.06f', a))`를 입력하면 파이의 소수점 여섯째 자리까지 출력된다. 마지막으로 `a`를 입력하면 이렇게 보여지는데, 간단하게 `format long`을 입력하면, 문자열이 자동으로 더 많은 소수점 자리까지 보여준다. 또한 `format short`는 소수점 자리를 몇 개만 보여주는 명령어다.

07'42"

지금까지 변수를 활용하는 방법을 배웠다. 이제 벡터와 행렬을 적용해보자. `MAT A`를 `A` 행렬이라고 하자. 여기 예시가 있다. `A = [1 2; 3 4; 5 6]` 를 입력하면 3×2 행렬이 생성되고, 첫 번째 행은 1,2 두 번째는 3,4 세 번째 행은 5,6이 된다. 여기서 세미콜론은 행렬에서 다음 행으로 가라는 명령어이다. 다르게 표현하는 방법도 있다. `A = [1 2 ; 3 4 ; 5 6]` 이렇게 해도 `A`를 3×2 행렬 값으로 설정하는 또 다른 방법이 된다. 비슷한 방법을 벡터에도 적용할 수 있다. 여기서는 `v = [1 2 3]` 이다. 이는 행 벡터 또는 3×1 벡터다. `fat y` 벡터에서, 3×1 벡터가 아니라 1×3 벡터다. 만약 열 벡터를 만들려면, `v = [1; 2; 3]` 이렇게 입력하면 된다. 그러면 3×1 열 벡터가 된다.

08'55"

여기 좀 더 유용한 표기법들을 소개하겠다. `v = 1:0.1:2` 이는 `v`에게 1부터 시작하는 다양한 원소를 제공한다. 이는 2까지 0.1씩 증가한 원소다. 따라서, 이를 입력하면, 행 벡터가 된다. 여기 1×11 행렬이 되는 것이다. 1, 1.1, 1.2, 1.3 그리고 계속 2까지 있다. 이번에는 `v = 1:6` 라고 입력하면 이처럼 1부터 6까지 숫자가 나열된다. 이제 또 다른 행렬을 생성하는 방법을 배워보자. `ones(2,3)`를 입력하면 원소가 모두 1인 2×3 행렬이 생성된다. `C = 2 * ones(2,3)` 라고 하면 원소가 모두 2인 2×3 행렬이 생성된다. 이를 쉽게 입력하려면, `c = [2 2 2 ; 2 2 2]` 를 입력하면 똑같은 행렬이 생성된다. `w = ones(1,3)` 를 입력하면 이는 행 벡터 또는 세 개의 1의 행이 된다. 비슷하게, `w = zeros(1,3)` 를 입력하면, 이와 같은 원소가 모두 0인 행렬 1×3 이 된다. 행렬을 몇 개만 더 생성해보자. `w = rand(1,3)` 를 입력하면 임의의 원소를 지닌 1×3 행렬이 나온다. `w = randn(3,3)` 을 입력하면, 0부터 1까지 사이에서 균일 분포한 임의의 원소를 지닌 3×3 행렬이 생성된다. 이를 계속 입력 할 때마다 0부터 1까지 사이에서 균일 분포한 다른 세트의 임의의 숫자를 얻게 된다.

11'02"

가우스 확률변수에 대해 알고 있거나 표준 확률 변수에 알고 있다면, `w = randn(1,3)` 를 입력하면, 값 세 개를 얻을 수 있는데, 이는 가우스 확률 변수 평균은 0이고, 분산 또는 표준 편차는 1이라는 것이다. 좀 더 복잡한 수식도 적용해볼 수 있는데, 끝에다가 세미 콜론을 입력했는데, 이게 출력되기를 원치 않기 때문이다. 십만 개의, 아니, 만개의 원소를 가진 벡터가 생성될 것이다. 그래도 한번 출력해보자. 그럼 이런 행렬이 생성된다. 만 개의 원소다. 방금 본 게 `w`를 출력한 것이었다. 이번에는 이와 같이 `hist(w)`, `hist` 명령어를 통해 막대그래프를 생성해 보겠다. 옥타브가 `hist` 명령어를 출력했는데, 몇 초 정도 걸린다. 이것은 임의의 변수로 만들어진 막대그래프다. 여기 보면 $-6 + 10x$ 가우스 확률변수 이다. 이제 막대그래프에 더 많은 막대를 추가할 것이다. 50개라고 입력해보자. 이 것은 평균이 -6인 가우스 막대그래프다. 여기에 - 6이

있고, 여기에 $\text{sqrt}(10)$ 을 더하여 ($\text{randn}(1,10000)$) 를 곱해줬기 때문이다. 따라서 이 가우스 확률 변수의 분산은 10이고, 표준편차는 제곱근 10이다. 따라서 3.1이 나온다.

12'42"

마지막으로 행렬을 생성하는 특수한 명령어인 eye 명령어를 하나 더 살펴보겠다. 여기서 eye는 identity를 의미하는 것이라는 말이 있다. eye(4) 는 4×4 항등 행렬을 말한다. 그래서 eye(6)를 입력하면 6×6 항등 행렬이, eye(3)는 3×3 항등 행렬이 나타난다. 마지막으로 이번 강의를 정리하면서, 마지막으로 유용한 명령어를 하나 더 설명하겠다. 바로 help 명령어다. help eye를 입력하면 항등 행렬의 help 함수를 불러온다. 끝내려면 Q를 누르고, help rand를 입력하면, 임의의 숫자 생성 함수 자료를 보여준다. 또한 help help 는 help function에서 help를 보여준다. 지금까지 옥타브에서 기본적인 작동법을 배웠다. 이를 통해 행렬을 생성하고 이를 곱하고 더할 수 있게 되었고, 옥타브에서 기본적인 작동법을 터득하게 되었다. 다음 강의에서는 옥타브에서 좀 더 복잡한 명령어와 데이터를 활용하고 이를 처리하는 것을 배우도록 하겠다.

No.	Title
Week 2-9	Moving Data Around

00'00"

옥타브에 대해 배워보는 이번 두 번째 강의에서는 옥타브에서 데이터를 이동시키는 방법을 배울 것이다. 기계 학습 문제에 관한 데이터를 가지고 있을 때, 이 데이터를 어떻게 옥타브에 로딩하는가? 이를 어떻게 행렬로 나타내는가? 그리고 생성한 행렬을 어떻게 처리할 것인가? 그 결과물을 어떻게 저장할 것인가? 데이터를 어떻게 읽기고 이 데이터를 어떻게 작동시킬 것인가? 여기에 지난 시간에 강의에도 활용한 그대로 옥타브 창이 있다. A를 입력하여 행렬을 생성할 건데, 여기에 $A = [1,2; 3,4; 5,6]$ 입력하면, 3×2 행렬이 완성된다. 옥타브에서 사이즈 명령어는 행렬의 크기를 말해준다. 여기에 $\text{size}(A)$ 를 입력하면 3 2 라고 뜬다. 이 사이즈 명령어는 1×2 행렬로 돌아간다. 그래서 $sz = \text{size}(A)$ 로 하면, 이제 1×2 행렬이 되어, 첫 번째 원소가 3, 두 번째 원소가 2가 된다. 이번에는 $\text{size}(sz)$ 를 입력하면 1×2 행렬이라고 뜨고, 여기 두 원소는 행렬 A에 포함되어 있다.

01'14"

$\text{size}(A,1)$ 을 입력하면 A의 첫 번째 차원의 크기를 말해주며, 이는 행의 수다. $\text{size}(A,2)$ 는 행렬 A에서 열의 개수다. 벡터 v가 있는데 $v = [1,2,3,4]$ 입력한 후에, $\text{length}(v)$ 를 입력한다. 이는 가장 긴 차원의 사이즈를 말해준다. $\text{length}(a)$ 도 입력해보면, 행렬 A는 3×2 행렬이므로 가장 긴 차원은 3이므로 이렇게 3이 출력된다. 하지만 이러한 길이는 보통 벡터에만 적용된다. 그러니까 $\text{length}([1,2,3,4,5])$ 와 같이 입력한다. 행렬의 길이를 구하려면 좀 더 복잡해지기 때문이다. 이번에는 어떻게 데이터를 로딩하고 파일 시스템에 있는 데이터를 검색하는지에 대해 알아보자. 옥타브를

처음 시작할 때 보통 옥타브가 저장된 위치가 어디인지부터 찾아본다. 여기 pwd 명령어는 옥타브가 어디에 설치되어있는지 현재의 디렉토리 또는 현재 경로를 보여준다. 지금은 아마도 검색 범위 밖에 있는 것 같다. cd 명령어는 디렉토리 변경을 의미한다. 그래서 C:\User\Ang\Desktop을 입력하면 이제 데스크탑 안으로 디렉토리가 변경되었다. 이번에는 ls를 입력하면, 이는 유닉스 또는 리눅스 명령어다. 이 명령어는 데스크탑에 있는 디렉토리를 리스트로 보여줘서 데스크탑에 있는 파일들이 이렇게 보인다.

03'14"

사실, 이 데스크탑에는 파일이 두 개가 있다. Feature X와 Price Y가 있는데, 아마도 해결하고자 하는 기계 학습 문제에서 온 듯 하다. 여기 내 데스크탑이 있다. 여기 Feature X가 있는데, 이 Feature X 파일은 두 행의 데이터가 들어 있다. 이는 사실 주택 가격 데이터이다. 이 데이터 세트에는 47행이 있다. 그리고 첫 번째 주택의 크기는 2104 평방피트로 방이 세 개가 있다. 두 번째 주택은 1600 평방피트로 방이 세 개가 있고 이렇게 쭉 나열되어 있다. 그리고 여기 Price Y 파일에는 학습 세트에 있는 가격 데이터가 담겨 있다. 그래서 Feature X와 Price Y는 데이터를 가진 텍스트 파일을 가리킨다. 그렇다면 이 데이터를 어떻게 옥타브에서 로딩시킬 수 있을까? 명령어 load featuresX.dat 이라고 입력하면 Feature X가 로딩되고, 이렇게 입력하면 Price Y도 로딩할 수 있다. 또 다른 방법이 있다. load('featuresX.dat') 이렇게 입력하면, 똑같이 로딩이 될 것이다. 여기 오타가 있었다. 이러면 똑같은 명령어가 된다. 따라서 이렇게 문자열의 파일명만 입력하면 되는 것이다. 그리고 옥타브에서는 문자열을 표현하기 위해서 이렇게 작은따옴표를 사용해서 문자열을 입력하면 그 문자열의 이름을 가진 파일을 로딩할 수 있다.,

04'47"

이번에 볼 WHO 명령어는 내 옥타브 작업영역에서 어떠한 변수를 가지고 있는지 보여준다. 그러니까 who는 옥타브가 가진 변수가 메모리에 현재 있는지 아닌지를 보여준다. Features X 와 Price Y 가 그들 중 하나고, 강의 초반부에 만들어 놓은 것도 변수가 된다. 이제 Features X를 입력하면 Features X의 데이터를 이렇게 보여준다. size(featuresX)를 하면 47 x 2 행렬이라고 나온다. 또한, size(priceY)를 입력하면 47 x 1 벡터라고 보여준다. 이는 47차원 벡터이다. 이는 내 학습 세트에 있는 모든 가격 Y가 들어있는 행 벡터이다. 아까 who 명령어를 입력하면 현재 작업영역에 있는 변수들을 보여주었는데, whos 명령어를 입력하면 이를 더 자세히 보여준다. 여기서 끝에 s가 붙으면서 원래 변수의 목록을 보여주는데, 거기에 그 사이즈도 같이 보여준다는 것이다. A는 3x2 행렬이고, features X 는 47 x 2 행렬이고, Price Y 는 47 x 1 행렬이자 벡터다. 또 이러한 변수의 용량이 얼마인지도 보여준다. 그리고 데이터의 타입도 보여준다. 여기서 double은 더블포지션부동소수점연산으로 이는 실수이자, 부동소수점수라는 것이다.

06'12"

이제 이 변수들을 삭제하고 싶으면 clear 명령어를 쓰면 된다. 이렇게 clear featuresX 와 whos를 입력하면 features X 변수가 삭제되었음을 알 수 있다. 그렇다면 데이터는 어떻게 저장할까? 변수 v를 가지고 와서 v=priceY(1:10) 라고 입력하면 여기 v는 벡터 Y의 첫 번째 10개의 원소가 된다.

이제 who 와 whos 를 입력하자. Y는 47×1 벡터고, V는 이제 10×1 이다. v=priceY(1:10) 를 입력하면 Y 의 첫 번째 10개 원소를 가져오게 된다.

06'54"

이제 이 데이터를 디스크에 저장하려고 한다. save hello.mat v; 를 입력하면, 변수 v를 hello.mat라는 파일명에 저장시켜 줄 것이다. 한번 입력 해보자. 그러면 데스크탑에 Hello.mat라는 파일이 생성되었다. 이 원도우에 맷랩을 설치해놔서, 여기 아이콘이 이렇게 보이는데, 원도우에서 이를 맷랩 파일로 인식하기 때문이다. 그러니까 여러분 것과 아이콘이 다르다고 해서 걱정하지 않아도 된다. 이번에는 모든 변수를 삭제해보겠다. 이렇게 clear 다음에 아무것도 입력하지 않으면, 작업영역에 있는 모든 변수를 삭제해준다. 지금 보면 작업 영역에 아무것도 남아있지 않음을 확인할 수 있다. 여기에 load hello.mat를 입력하면 v 변수를 다시 로딩할 수 있다. 이는 아까 hello.mat 파일에 저장해둔 데이터이다. 이번에는 save hello.mat v를 입력해 데이터를 2진 포맷으로 정했다. 2진 포맷은 좀 더 압축된 포맷이다. 만약 v에 데이터가 많을 경우, 더 압축될 것이다. 여기 이 부분은 지우도록 하자.

08'04"

사람이 읽을 수 있는 포맷으로 데이터를 저장하고 싶은 경우에는 save hello.txt v -ascii 을 입력한다. 그러면 데이터를 텍스트 또는 아스키 포맷의 텍스트로 저장할 것이다. 이렇게 하고 나면, 여기 파일이 생성되었다. Hello.txt 파일이 여기에 있고, 이를 열어보면, 여기 텍스트 파일로 데이터가 저장되어 있다. 이렇게 데이터를 로딩하고 저장하는 것을 배워봤다.

08'35"

이번에는 데이터를 조작하는 것을 이야기해보겠다. $A = [1 \ 2 \ ; \ 3 \ 4 \ ; \ 5 \ 6]$ 이라고 하면 3×2 행렬이 된다. 인덱싱도 마찬가지다. $A(3,2)$ 를 입력하면 이는 행렬 A의 3, 2 원소를 인덱싱하는 것이다. 보통은 $A_{\{3, 2\}}$ 아니면 A_{32} 라고도 쓰는데, 여기서 3, 2 는 A행렬의 세 번째 행과 두 번째 열에 있는 원소 6을 가리키는 것이다. $A(2,:)$ 라고 입력하면, 두 번째 행에 있는 모든 원소를 가져올 수 있다. 여기서 큰따옴표(")는 그 행이나 열에 있는 모든 원소를 가리킨다. 따라서 $A(2, :)$ 은 A행렬의 두 번째 열의 모든 원소를 가져오는 것을 말한다. 비슷한 예로, $A(:, 2)$ 를 입력하면, 이는 행렬 A의 두 번째 열에 있는 모든 원소를 가져오는 것이다. 그러면 2, 4, 6이 있다. 콜론은 모든 것을, 2는 두 번째 열을 의미한다. 이게 행렬 A의 두 번째 열이고, 여기 2, 4, 6이 있다.

09'50"

이번에는 조작하는데 있어 가장 정교한 인덱스에 대해 알아보자. 예시를 먼저 살펴보자. 이 작업을 자주하게 될지는 모르겠지만, $A([1 \ 3],:)$ 을 입력하면, 이는 행렬 A에서 1, 3 행과 모든 열에 있는 모든 원소를 가져오는 것이다. 즉, 행렬 A에서 첫 번째, 세 번째 행과 모든 열에 있는 원소를 가져오는 것이다. 행렬 A에서 $A([1 \ 3],:)$ 첫 번째 행, 두 번째 행, 세 번째 행에서 모든 원소를 가져오라는 것이다. 그리고 여기서 콜론은 첫 번째와 두 번째 열을 말하는데, 그러면 이렇게 1 2 5 6 이 구해진다. 하지만 이와 같은 침자 인덱스 조작 소스는 그리 자주 활용하지는

않을 것이다.

10'39"

예시를 좀더 보여주겠다. 여기 A 행렬이 있고, A(:, 2)는 두 번째 열이 된다. 지정을 할 때도 이를 활용할 수 있다. 행렬 A에서 두 번째 열 대신에 $A(:, 2) = [10; 11; 12]$ 를 지정하면 행렬 A에서 두 번째 열을 가져가고 거기에 열 벡터 10, 11, 12를 지정하는 것이다. 이제 A 행렬이 있고, 1, 3, 5 이 있고 두 번째 열이 10, 11, 12로 대체 된 것을 볼 수 있다. 하나 더 해보겠다. $A = [A, [100, 101, 102]]$ 라고 하면 이는 다른 열 벡터를 오른쪽에 첨부하는 것이다. 이런 여기 실수를 한 것 같다. $A = [A, [100; 101; 102]]$ 세미콜론을 넣었어야 했다. 그러면 행렬 A가 이와 같게 된다. 잘 이해할 수 있었으면 좋겠다. 여기 $[100; 101; 102]$ 는 열 벡터이고 $A = [A, [COLUMN VECTOR]]$ 를 오른쪽에 붙이면, 원래 왼쪽에 원소 6개를 가지고 있던 행렬 A에 붙게 되는 것이다. 즉, 행렬 A의 오른쪽에 또 다른 열 벡터를 덧붙인 것이다. 그래서 지금 행렬 A가 3×3 행렬로 보이는 것이다.

12'15"

이번에는 가끔 사용하는 교묘한 트릭을 하나 알려주겠다. $A(:)$ 이렇게 입력하면, 이는 특별한 케이스 문법이다. 이는 행렬 A의 모든 원소를 하나의 열 벡터로 만들라는 것이다. 그러면, 9×1 벡터가 생성된다. 이렇게 원소가 다 하나로 이어져있다. 예시를 몇 개 더 살펴보자. $A = [1 2 ; 3 4 ; 5 6]$ $B = [11 12 ; 13 14 ; 15 16]$ 이라고 하자. 행렬 A, B를 가지고 새로운 행렬 C를 생성해보자. 여기 행렬 A가 있고, 행렬 B가 있다. 그리고 $C = [A B]$ 라고 하면 두 행렬을 가져다가 연결 시켰다. 왼쪽에는 행렬 A가 오른쪽에는 행렬 B가 있다. 그리고 이 둘을 붙여서 행렬 C를 생성했다. $C = [A ; B]$ 라고 할 수 있는데 여기서 세미콜론은 다음 것을 밑에다가 붙여준다는 의미다. $C = [A ; B]$ 를 입력하면, 행렬 A와 B가 합쳐지지만 위아래로 붙는다는 것이다. 행렬 A가 위에 행렬 B가 아래에 있고, 행렬 C는 이제 6×2 행렬이 되었다. 세미콜론은 보통 다음 줄로 내려가라는 뜻이다. 그러므로 행렬 C는 A행렬과 그 아래에 붙은 B 행렬로 이루어진 것이다. $[A B]$ 는 $[A, B]$ 와 같으므로 어느 것을 입력해도 같은 값이 나온다.

14'09"

지금까지 배운 것을 토대로 행렬을 생성하는 방법을 알게 되었고, 빠르게 행렬을 잊고, 이를 가져와서 더 큰 행렬로 만드는 명령어를 이해했기를 바란다. 옥타브는 짧은 코드 몇 줄로 매우 빠르게 복잡한 행렬을 배치하고 데이터를 이동시킬 수 있어서 매우 편리하다. 오늘의 강의는 여기까지다. 다음 강의에서는 데이터를 가지고 복잡한 연산을 직접 해볼 것이다. 이를 통해 옥타브에서 명령어 몇 개 만으로도 빠르게 데이터를 이동시킬 수 있다는 것을 직접 알 수 있게 될 것이다. 벡터와 행렬을 로딩하고 저장하고, 데이터를 로딩하고 저장하고, 행렬을 둘을 묶어서 더 큰 행렬을 만들고, 인덱싱하고, 행렬에서 특정한 원소를 선택하는 방법 말이다. 이번 강의에서 다양한 명령어들이 나왔는데, 가장 좋은 방법은 강의를 다 듣고 난 후, 강의 자료를 보면서 내가 어떤 명령어를 입력했는지 다시 확인하는 것이다. 강의 사이트를 보면 강의 영상에 관한 자료가 있다. 이를 읽어보면서 옥타브에서 명령어를 직접 입력해보면서 이를 직접 활용해보도록 하자.

분명히 말하자면, 모든 명령어를 외우려고 노력하는 것은 아무 의미가 없다. 대신에 이번 강의를 통해 여러분이 어떤 것을 할 수 있는지에 대한 이해를 하면 좋겠다. 그래서 나중에 학습 알고리즘을 스스로 프로그래밍 하려고 할 때 특정한 명령어를 찾으려고 하면, 옥타브가 이를 할 수 있는데 여기서 여러분이 본 적이 있기 때문이다. 그러니 이번 강의 자료를 참고해서 사용해보고 싶은 명령어를 살펴보자. 여기까지 이번 강의를 마치고, 다음 강의는 데이터의 복잡한 연산을 어떻게 하는지, 데이터를 어떻게 연산하는지, 그리고 학습 알고리즘을 실제로 구현해보겠다.

No.	Title
Week 2-10	Computing on Data

00'00"

여러분은 이제 옥타브에서 데이터를 로딩하고 저장하며, 이를 가지고 행렬로 만들 수 있게 되었다. 이번 강의에서는 데이터를 가지고 연산 작업을 해볼 것이다. 그 다음에는 이러한 연산 작업 소스를 가지고 학습 알고리즘을 구현해 볼 것이다. 그럼 강의를 시작해보겠다. 여기 옥타브 원도우가 있다. 먼저 예시로 사용할 변수들을 빠르게 초기화하도록 하겠다. $A = [1 \ 2; 3 \ 4; 5 \ 6]$ 는 3×2 행렬이고, $B = [11 \ 12; 13 \ 14; 15 \ 16]$ 는 3×2 행렬, 그리고 $C = [1 \ 1; 2 \ 2]$ 는 2×2 행렬이다. 이들 중 두 행렬을 곱해보려고 한다. A 와 C 를 곱해주려면, $A*C$ 를 입력하면 된다. 3×2 행렬 곱하기 2×2 행렬이다. 이는 3×2 행렬을 출력한다. element wise 작업도 할 수 있는데, $A.*B$ 를 하면, 행렬 A 의 모든 원소를 가지고 이에 대응하는 B 행렬의 원소에 곱해줄 것이다. 여기 행렬 A , 행렬 B 가 있다. 그리고 $A.^2$ 가 있다.

01'06"

예를 들어, 첫 번째 원소는 $1 \times 1 = 1$ 이고, 두 번째는 $2 \times 12 = 24$ 가 된다. 이것이 element wise 곱셈이다. 일반적으로 마침표(.)는 옥타브에서 element wise 곱셈을 의미한다. 행렬 A 가 있는데 $A.^2$ 를 입력하면, 행렬 A 의 원소에 각각 제곱하게 된다. 1의 제곱은 1, 2의 제곱은 4 이런 식으로. 이번에는 v 를 벡터라고 하고, $v = [1 \ ; 2 \ ; 3]$ 라고 하면 이는 열 벡터가 된다. 또 $1./v$ 를 입력하면, v 의 모든 원소를 역수로 만든다. 그래서 $1/1, 1/2, 1/3$ 가 나오고, 행렬 A 도 입력하면 이렇게 나온다. $1./$ 를 입력하면 A 행렬의 원소를 모두 역수로 바꿔준다. 즉, 여기 있는 마침표가 이것이 element-wise 작업이라는 힌트를 주는 것이다.

02'07"

$\log(v)$ 를 입력하면, v 의 element-wise 로그이다. $\exp(v)$ 는 원소의 거듭제곱을 의미한다. 이게 E고 이것은 E의 거듭제곱은 EQ 왜냐하면 v 가 이것이기 때문이다. 이번에는 $\text{abs}(v)$ 를 입력하면, v 값의 element-wise 절대값을 구할 수 있다. v 가 양수였기 때문에, -1, 2, -3 를 입력하면, element-wise 절대값으로 이렇게 음이 아닌 정수가 나온다. $-v$ 는 v 값에 -가 붙는다. 이는 $-1 * v$ 와 같지만 보통 $-v$ 만 쓴다. 또 뭐가 있을까? 여기 간단한 트릭을 하나 알려주겠다.

02'57"

먼저 v 의 모든 원소를 1씩 증가시키려고 한다. 한가지 방법은 원소가 모두 1인 3×1 벡터를 만들어서 이를 v 에 추가하는 것이다. 그렇게 하면, v 는 1, 2, 3에서 4, 5, 6으로 증가하게 된다. $\text{length}(v) = 3$ 이므로, $\text{ones}(\text{length}(v), 1)$ 는 원소가 모두 1인 3×1 이고, $\text{ones}(3, 1)$ 가 있고, $v + \text{ones}(3, 1)$ 로 여기 있는 벡터를 v 에 더하는 것이다. 따라서 v 의 원소가 1만큼씩 증가했다. 아니면 그냥 간편하게 $v+1$ 만 입력해도 1만큼 증가하는 것을 볼 수 있다. 여기 v 가 있고 $v+1$ 를 입력하면, v 의 모든 원소에 element wise 1만큼씩 증가함을 의미한다. 또 다른 작업을 알아보자. 여기 행렬 A 가 있고, 이를 전치행렬로 바꾸고 싶으면, A' , 작은 따옴표 기호를 추가하면 된다. 이건 왼쪽 작은 따옴표다. 키보드에 보면 여기에 있다. 왼쪽 작은 따옴표와 오른쪽 작은 따옴표가 있다. 이는 표준 인용 부호다. A' 를 하면 행렬 A 가 전치행렬이 되고, $(A')'$ 는 다시 원래 A 행렬로 돌아온다.

04'24"

좀더 유용한 수식을 알아보자. $a = [1 15 2 0.5]$ 1×4 행렬이다. $\text{val} = \max(a)$ 라고 하면 A 행렬의 최댓값을 보여주는데 여기서는 15다. $[\text{val}, \text{ind}] = \max(a)$ 에서 val 는 a 의 최댓값을, ind 는 인덱스값을 나타낸다. 즉, 원소 2가 최댓값 15이라는 것이다. 주의할게 있는데, A 가 행렬일 때, $\max(A)$ 를 입력하면, 열 방향으로 최댓값을 보여준다. 이에 대해서는 잠시 후에 다시 설명해주겠다. 다시 $a = [1 15 2 0.5]$ 로 돌아와서, $a < 3$ 일 때, element wise 작업 중 하나인 element wise 비교다. 먼저, a 의 원소가 3보다 작아이므로 첫 번째는 참이니 1, 두 번째는 3보다 작지 않아 거짓이므로 0, 3, 4번쩨는 3보다 작으니 1, 1, 값을 가진다. 이것이 바로 $a < 3$ 변수에서 모든 원소 네 개를 element wise 비교한 것이고, 각각 원소가 3보다 작은지 아닌지를 보고 참인지 거짓인지를 판단하여 결과를 보여준다.

05'24"

이번에는 $\text{find}(a < 3)$ 이다. 이 수식은 a 의 원소, 변수 중에서 어떤 것이 3보다 작은지를 묻는데, 여기서는 1, 3, 4번째 원소가 해당한다. 다음 예시는 $A = \text{magic}(3)$ 이다. magic function은, 먼저 help magic 을 입력해보자. 이처럼 magic function은 행렬을 $n \times n$ 행렬로 만들어준다. 모든 수학적 특성인 행과 열, 그리고 대각선이 같은 수로 배열된다. 그래서 이는 기계 학습에서 그다지 유용하지는 않지만, 3×3 행렬을 생성하는 편리한 방법으로 활용하고 있다. 즉, 이 magic squares는 모든 행, 열, 대각선을 같은 수로 만들어주는 특성을 지니고 있다. 수학적 구성이라고 볼 수 있다. 테스트용 프로그램이나 지금처럼 옥타브를 강의할 때 가끔 magic function을 사용하지만, 실제로 유용한 기계 학습 응용에서는 사용하지 않는다.

06'53"

[r,c] = find(A>=7) 라고 입력하면, 7보다 크거나 같은 A에 있는 모든 원소를 찾아준다. 여기서 r,c는 행과 열을 의미한다. (1,1), (3,2), (2,3)이 7보다 크거나 같다는 것을 알 수 있다. (2,3)은 7이기 때문에 당연히 7보다 크거나 같다에 포함된다. 그렇지만, 이 함수가 어떤 기능을 하는지를 다른 함수는 어떤 기능을 하는지 다 외우고 있는 것은 아니다. 이러한 find function를 활용할 때, 가끔 이게 무슨 기능을 하는지 잊어버려서 help find를 입력해 이를 확인한다.

07'37"

이제 두 개만 더 간략하게 살펴보자. 먼저 sum function이다. 여기 a가 있고, sum(a)을 입력하면, a의 모든 원소를 합해준다. 그리고 이를 다 곱하고 싶으면, prod(a)를 입력하면 모든 원소의 곱을 보여준다. 여기 floor(a)는 버림 끝수를 없애서, 0.5는 0이 된다. 그리고 ceil 또는 ceiling(A)는 정수로 올림을 하여 0.5는 1이 된다. 또한, rand(3)를 입력하면 3x3 행렬이 생성되고, max(rand(3),rand(3))는 두 개의 rand(3)의 element-wise 최댓값을 구한다. 여기를 보면, 모든 숫자가 좀 큰 편인데, 이 숫자 모두 임의로 생성된 두 개의 행렬의 element wise 최댓값이기 때문이다. 이제 바로 내 magic number, 3x3이다. 이렇게 max(A,[],1) 입력하면, 이는 열방향 최댓값을 보여준다. 따라서 첫 번째 열에서 최댓값은 8이다. 두 번째 열에서 최댓값은 9이고, 세 번째 열에서 최댓값은 7이다. 여기서 1은 행렬 A에서 가져온 최댓값을 1차원으로 보여주라는 것이다.

09'04"

반면에, max(A, [], 2) 이렇게 입력하면, 이는 행마다 최댓값을 보여준다. 첫 번째 행은 8, 두 번째는 7, 세 번째는 9다. 따라서 이런 수식을 활용하면, 행 또는 열마다의 최댓값을 구할 수 있다. column wise element에 디폴트 방법을 기억해라. 행렬 A 전체에서 최댓값 원소를 구하려면, (max(A)) = 9이다. 아니면 A를 벡터로 바꿔서 max(A(:)) 입력하면 이를 벡터로 인식해서 벡터의 최댓값을 이렇게 찾아준다. 이번에는 A = magic(9) 하면 9x9 행렬이 나오는데, 이러한 magic square는 모든 행과 열 그리고 대각선의 원소의 수가 같다는 특징을 기억해야 한다. 먼저, sum(A, 1)를 입력하면, 열끼리 덧셈하는데, 열마다 원소를 모두 더해주는 데 여기는 9x9 행렬의 모든 열마다 각각의 합이 똑같이 369임을 알 수 있다. 이번에는 행간의 합을 구해보자. sum(A,2)을 입력하면, 행렬 A 모든 행마다 원소를 더하는데, 그 값이 모두 똑같이 369임을 알 수 있다.

10'33"

이번에는 대각선의 원소의 합을 구해서 이 또한 같은 값을 가지는지 알아보자. 이번에는 eye(9)을 입력해 9x9 항등 행렬을 생성한다. 먼저 A 행렬을 만들고, A의 원소에 element wise를 곱해보자. 여기 행렬 A가 있고, A .* eye(9)를 입력하면 이 두 행렬에서 element wise 한 원소를 가지고 와서 A행렬의 이 대각선을 제외하고는 모두 삭제한다. 이제 sum(sum(A .* eye(9)))를 입력하면, 대각선 원소의 합을 알려주고, 이는 또한 369임을 알 수 있다. 반대쪽 대각선도 합계를 구할 수

있다. 그러니까 왼쪽 상단부터 오른쪽 하단까지, 또 반대로 왼쪽 하단부터 오른쪽 상단까지의 합을 구할 수 있다. 이를 위해서 `sum(sum(A .* flipud(eye(9))))`를 입력하면 되는데, 이는 좀 복잡한 편이라 알고 있지 않아도 된다. 혹시 궁금해할 수 도 있어서 보여주는 것이다. 한 번 살펴보자. 여기 `flipud` 는 `flip up down`을 의미한다. 이렇게 입력하면, 반대편 대각선의 합을 보여준다. 그리고 이는 합이 369가 되는 것이다. 이 과정을 보여주겠다. `eye(9)` 행렬이 있을 때, `flipud(eye(9))`을 입력하면, 똑같은 행렬이 수직으로 뒤집힌다. 결과적으로 이렇게 뒤집혀 대각선이 반대가 된 행렬이 나오는 것이다.

12'08"

이제 마지막 명령어만 알아보면 이번 강의는 끝이다. `A = magic(3)` 행렬이고, 이 행렬을 역행렬로 바꾸려면, `pinv(A)`를 입력한다. 이는 의사역행렬이라고 하는데, 그저 행렬 `A`의 역행렬을 생각하면 된다. 그러면, 이렇게 `A`의 역행렬이 나온다. `temp = pinv(A)` 라고 하고, `temp*A`를 해주면 이는 대각선에 1이 있고, 나머지는 모두 0인 단위 행렬이 나오게 된다.

12'44"

지금까지 데이터와 행렬에서 다양한 연산 작업에 대해 알아봤다. 학습 알고리즘을 구현하게 된 후에 가장 유용하게 쓰이는 점은 그 결과를 볼 수 있으므로 이를 도표로 만들어내거나 시각화할 수 있다는 것이다. 다음 번 강의에서는 이와 비슷하게 옥타브에서 한 두줄 정도의 코드를 가지고 설명할 것이다. 데이터를 시각화하고, 도표를 만들고 여러분의 학습 알고리즘이 어떻게 작동하는지에 대해 더 자세히 알아보도록 하자.

No.	Title
Week 2-11	Plotting Data

.00'00"

학습 알고리즘을 개발할 때, 간단한 도표 몇 개가 이 알고리즘의 역할을 잘 이해할 수 있게 도와주고, 또 이 알고리즘이 해야 할 일을 잘 진행하고 있는지 건전성 검증을 할 수 있다. 예를 들어 예전 강의에서 비용 함수(θ)를 도표화하는 것을 이야기한 적 있었는데, 이는 기울기 감소에서 한 점으로 모이도록 할 수 있었다. 종종 데이터나 학습 알고리즘 출력의 그래프가 학습 알고리즘을 어떻게 개선시킬지 아이디어를 주기도 한다. 다행히, 옥타브는 다양한 그래프를 생성할 수 있는 매우 간단한 툴이다. 그리고 직접 학습 알고리즘을 활용하면서 데이터나 학습 알고리즘을 그래프로 만드는 것이 알고리즘을 개선하는 아이디어를 주는데 중요한 역할을 한다는 것을 깨달았다. 이번 강의에서는 이러한 옥타브 툴을 활용하여, 데이터를 그래프로 만들고 시각화 해보겠다.

.00'53"

여기 옥타브 윈도우가 있다. 그래프로 만들 데이터를 먼저 설정해보자. 먼저 $t = [0:0.01:0.98]$; 라고 하면, 여기 0부터 98까지 숫자 세트가 있다. 이번에는 $y1 = \sin(2\pi \cdot 4 \cdot t)$; 를 입력하여, 사인 함수를 그래프화하려고 한다. 매우 간단하다. `plot(t,y1);`를 입력하고 엔터를 누르면, 이러한 도표가 나오는데, 가로축은 T 변수, 그리고 세로축은 y1 인데 우리가 방금 연산한 사인 함수다. 이번에는 $y2 = \cos(2\pi \cdot 4 \cdot t)$; 라고 입력해보자. 그리고 `plot(t,y2,'r');`를 입력하면, 옥타브에서 사인 그래프가 사라지고, 여기 코사인 함수로 대체된다. 코사인의 x 사이즈는 1이다. 이제 사인과 코사인 그래프를 두 개 겹치게 하려면 어떻게 해야 할까? `plot(t,y1);`를 입력하면, 여기 사인 함수가 있고, 이 함수에 `hold on` 기능을 사용할 것이다. 여기서 `hold`는 그래프를 잡아두고 있어서, 새로운 그래프가 위에 겹쳐질 수 있도록 하는 기능이다. 이제 `plot(t,y2,'r');`를 입력해 코사인 함수를 다른 색으로 추가될 수 있도록 하겠다. 이렇게 '`r`'을 추가하면 된다. 그러면 현재 입력한 수치로 대체되는 것이 아니라, 코사인 함수가 이전 그래프에 겹쳐지게 된다. 여기서 `r`은 색깔을 나타낸다.

.02'32"

여기 몇 개 명령어를 더 소개하겠다. `xlabel('time');` 를 입력하면 가로축에 레이블이 입력되고, `ylabel('value');` 은 세로축에 레이블이 입력된다. 또 `legend('sin','cos');` 을 입력해 우측 상단에 사인, 코사인 이름을 두 줄로 입력할 수 있다. 다음으로 `title('my plot');` 은 이 수치의 상단에 제목을 입력할 수 있다. 마지막으로 이를 저장하고 싶으면, `print -dpng 'myPlot.png'`를 입력한다. 여기서 `png`는 그래픽 파일 형식인데 이를 입력하면 해당 형식으로 저장된다. 이를 시행해보면, 디렉토리를 변경해서 이렇게 출력이 될 것이다. 옥타브 환경 설정에 따라 조금 시간이 더 걸릴 수 있다. 저장 디렉토리를 바탕화면으로 해놨기 때문에 지금 저장하는 데 시간이 좀 걸린다. 이제 바탕화면으로 돌아가면, 이것을 숨겨놓고, 여기 옥타브가 저장한 `myplot.png` 파일이 보인다. 이게 바로 `png` 포맷으로 저장된 파일이다.

04'03"

옥타브는 다른 수 천 개의 포맷으로도 저장할 수 있다. 여기에 `help plot` 을 입력하면 `png` 말고도 이러한 수치를 다양한 포맷으로 저장할 수 있다. 마지막으로 그래프를 삭제하고 싶으면, `close` 명령어를 통해 삭제할 수 있다. `Close;` 입력하면 아까 봤던 그래프가 바탕화면에서 사라진 것을 볼 수 있다. 옥타브는 어떠한 형태와 숫자를 구체화할 수 있다. `figure(1); plot(t, y1);` 를 입력하면 이렇게 첫 번째 그래프가 생성된다. 그리고 두 번째로 `figure(2); plot(t, y2);` 를 입력하면 `figure number`를 다르게 해서 이렇게 또 다른 그래프가 생성되었다. 이렇게 두 개의 그래프가 생성되었는데, `figure 1`, `figure 2` 하나는 사인 함수, 다른 하나는 코사인 함수다. 자주 활용하는 간단 명령어인 `subplot` 가 있다. `subplot(1,2,1);`를 입력한다. 이는 것인데, 처음 2 변수를 가지고, 1×2 그리드로 그래프를 세분화한다. 그리고 첫 번째 원소에 접근한다. 이게 바로 첫 번째 변수 1이다. 즉, 수치를 1×2 그리드로 세분화하는 것이다. 이제 첫 번째 그리드에 접근하겠다. 이를 입력하면 이렇게 그래프가 왼쪽에 뜨게 된다. `plot(t,y1);`를 입력하면 첫 번째 그리드에 그래프가 채워졌다. 이번에는 `subplot(1,2,2);` 입력하면, 두 번째 그리드에 접근하여, `plot(t,y2);` 를 입력하면, 오른쪽에 두 번째 그리드, `y2` 그래프가 생성된다.

05'53"

다음으로 알아볼 명령어를 통해 축의 규모를 바꿀 수 있다. 축을 바꾸기 위해 `axis([0.5 1 -1 1])` 를 입력해보자. 그러면 오른쪽에 있는 `figure`의 x 값과 y 값 범위를 설정하여, 오른쪽 그래프의 가로축의 값이 0.5에서 1사이에 있도록 하고, 세로축의 값은 -1 부터 1사이에 오게 한다. 이 명령어들을 모두 외울 필요는 없다. 다른 그래프에 접근하려면 이 접근 명령어가 필요한데, 이는 옥타브 `help axis` 명령어에서 자세한 내용을 알아보면 된다. 마지막으로 몇 개만 더 살펴보겠다. `clf;` 는 수치를 삭제해주는데 여기에 `feature`이 하나 있다.

06'38"

`A = magic(5)` 라고 하자. 이 5×5 행렬을 가지고 이를 시각화 하기 위해 가끔 사용하는 간편한 방법인데 `imagesc(A)`를 입력하면, 5×5 행렬 그래프, 5×5 그리드 색상을 보여준다. 이는 행렬 A에 있는 값에 따라 색상이 달라진다. 또한 컬러바를 이용할 수 있다. 예를 들어, 좀 더 복잡한 명령어를 사용해보겠다. `imagesc(A), colorbar, colormap gray;` 를 입력하면, 이는 명령어 세 개를 한꺼번에 입력하는 것이다. `Imagesc`를 먼저하고, 다음에 `colorbar`, 마지막으로 `colormap gray`가 구현되는 것이다. 그럼 이렇게 회색 칼라 맵이 나오고 오른쪽에는 칼라바가 있다. 여기 칼라바는 다른 색상이 어떤 숫자에 대응하는지를 보여준다. 예를 들어, 행렬 A의 좌측 상단은 17인데, 이는 회색의 중간 색상으로 표현되어 있다. 그 옆에 두 번째 원소 $A(1,2)$ 는 24이고 여기 네모에 대응하며, 거의 흰색을 띠고 있다. 여기 작은 수치를 살펴보면, $A(4, 5) = 3$ 이고 이는 칼라바에서 훨씬 어두운 색 이미지에 대응하고 있다. 예를 하나 더 들어보자. 이번에는 더 큰 행렬을 활용해보자. `A = magic(15)` 이 있고, 15×15 행렬이 있는데, `imagesc(magic(15)), colorbar, colormap gray;` 를 입력하면, 이 행렬의 값을 이미지를 볼 수 있다.

08'33"

마지막으로 강의를 정리하기 전에 방금했던 내용은 콤마 체이닝 함수 호출이다. 방법은 다음과 같다. 여기 이렇게 `a=1,b=2,c=3` 입력하고 엔터를 누르면, 이는 세 가지 명령어를 한꺼번에 처리하는 것이다. 아니면 사실은 하나의 명령어를 먼저 하고 그 뒤에 차례차례 처리한 후, 그 결과를 한꺼번에 보여주는 것이다. 이는 또한 `a=1;b=2;c=3;` 와 비슷하다. 콤마 대신에 세미콜론을 사용하면 아무것도 출력되지 않는다는 점만 다르다. 이게 바로 명령어 콤마 체이닝 또는 함수 호출 콤마 체이닝 이라는 것이다. 이는 옥타브에서 편리하게 다양한 명령어를 아까 `image sc` `color bar`, `colon map` 처럼 같은 선상에 입력하는 방법이다. 이제 마무리하겠다. 여러분은 이제 각기 다른 수치를 옥타브에서 그래프화하는 방법을 알게 되었다. 다음 강의에서는 `if`, `while`, `for` 같은 제어문을 옥타브에서 어떻게 다루는지, 또한 함수의 정의와 활용법을 말해주고자 한다.

No.	Title
Week 2-12	Control Statement: for, while, if statement

00'00"

이번 강의에서는 옥타브 프로그램에서 제어문을 사용하는 방법, 예를 들어 "for", "while" 그리고 "if" 와 같은 제어문이 있고, 이를 어떻게 정의하고 어떻게 활용하는지 알아보자. 여기 옥타브 원도우가 있다. 먼저 'for' 루프를 어떻게 활용하는지 알아보자. $v = zeros(10,1)$ 를 입력하고, 원소가 0으로 이루어진 10×1 벡터가 나온다. `for i=1:10,` 를 입력한다. 그리고 $v(i) = 2^i;$ 입력한 후, `end;`를 입력하면 끝이다. 화이트스페이스는 아무 의미가 없는데, 깔끔하게 보이기 위해서 들여쓰기를 한 것이다. 스페이싱은 아무 영향을 미치지 않는다. 이를 입력하면, v 는 2의 거듭제곱 값을 보여준다. 여기 이 $i = 1:10$ 부분은 i 가 1부터 10까지 반복해서 값을 낼 수 있게 해준다. 또한, 이러한 과정을 다음과 같이도 할 수 있다. `indices = 1:10` 이라고 하면 인덱스가 1부터 10이 된다. 또한, `for i = indices` 라고 해도 $i = 1:10$ 와 같은 말이다. 그래서 이렇게 `disp(i)` 를 입력하면 같은 결과가 나온다.

01'23"

이게 바로 "for" 루프를 활용하는 방법이다. "break" 와 "continue" 명령어에 대해서도 알고 있다면, 옥타브에서 이러한 인사이드 루프를 활용 할 수 있는데, 와일 루프가 작동하는 과정을 보여주겠다. 여기 벡터 v 가 있다. 이제 while 루프를 써보자 `i = 1; while i <= 5,` 그리고 $v(i) = 100;$ $i = i+1;$ 그리고 `end;`를 입력한다. 이는 무엇을 뜻하는가? i 는 1과 같고, 그리고 $v(i)$ 가 100과 같고, 마지막에 i 가 5보다 작거나 같다면 i 에 1을 더해줬다. 그 결과로 아까 v 는 2의 거듭제곱 벡터였다. 이제 여기서 처음 다섯 개 원소를 가져가 여기 100으로 덮어 써웠다. 이게 바로 while 루프 문법이다.

02'23"

또 다른 예시를 살펴보자. `i = 1; while true,` 여기서 break 제어문을 활용하는 법을 보여주겠다. $v(i) = 999;$ 이고 $i = i+1;$ `if i == 6, break;` `end;` `end;` 이는 또한 우리가 if 제어문을 처음 활용한 것이다. 이 로직을 잘 이해하기를 바란다. $i = 1$ 이기 때문에 증가 루프다. $v(i) = 999;$ 이고, $i = i+1;$ `if i == 6,` 일때, `break;` 를 써서 while의 역할을 차단하고, 이에 따른 결과로 여기 벡터 v 의 처음 원소 다섯 개를 999으로 대체하는 것이다. 입력해보면, 이와 같이 처음 원소 다섯 개가 999로 대체된 것을 볼 수 있다. 여기까지 if 와 while 제어문 그리고 end 의 역할을 알아봤다. 여기 end 가 두 개 있다. 여기 있는 end if 제어문을, 두 번째 end는 while 제어문을 끝내는 것이다.

03'33"

이제 if-else 제어문을 활용하기 위한 좀 더 일반적인 문법을 살펴보도록 하자. $V(1) = 999$ 이다. $v(1) = 2;$ 입력해보자. `if v(1)==1, disp('The value is one!');` 을 입력한다. 이제 else 제어문이 등장한다. 여기서는 else if 다. elseif $v(1)==2,$ 여기 예시에서 if는 참이다. `disp('The value is two!');` else는 `disp('The value is not one or two!');` 그렇다. 이것이 if-else 제어문 ends. 물론, 여기서 elseif $v(1)==2,$ 라고 했으니 the value is 2 라고 나온다. 다음으로 이거에 대해 얘기한 적이 없는

것 같지만, 옥타브를 종료하려고 하면, exit 명령어를 입력하고 엔터를 누르면 된다. 그러면 옥타브가 끝나게 된다. Quit의 q 명령어도 이와 같다.

04'42"

이번에는 함수와 이를 정의하고 어떻게 활용하는지 알아보자. 여기 바탕화면이 있고, 파일을 데스크탑에 미리 정의, 저장을 해놨는데 파일명은 "squarethisnumber.m" 이다. 이것이 바로 옥타브에서 함수를 정의하는 방법이다. 먼저 함수 코드와 그 뒤에.m으로 파일을 하나 만든다. 그러면 옥타브가 이 파일을 검색하면 이 파일이 "squarethisnumber.m". 함수의 정의를 물을 때 찾아야 한다는 것을 인식하게 된다. 파일을 열어보자. 내가 이 파일을 열기 위해 마이크로소프트 프로그램인 워드패드를 사용하고 있다는 것을 주목하라. MS 윈도우를 사용하고 있다면, 노트패드 대신 워드패드를 사용하기를 권장한다. 다른 텍스트 에디터가 있다면 사용해도 좋지만, 노트패드는 띄어쓰기가 엉망으로 되어있을 때가 있기 때문이다. 노트패드만 있을 경우에는 사용해도 되긴 하지만, 워드패드나 다른 텍스트 에디터가 있다면 함수를 에디팅하는 데 활용하는 것이 좋겠다.

05'44"

이제 옥타브에서 함수를 정의하는 법을 살펴보겠다. 약간 확대해보겠다. 이 파일에는 세 줄만 적혀 있다. 첫 번째 줄은 function y = squareThisNumber(x) 인데 이는 y 값을 리턴하여 1을 리턴하여, 그 값은 y 변수에 저장될 것이라고 옥타브에게 말해주는 것이다. 또한 이 함수에 argument 하나가 있는데, 이는 argument X이고, 여기서 함수 body가 정의되는 방법은 $y = x^2$; 이다라고 옥타브에게 말해주는 것이다. 그렇다면 이 함수를 "square" 라고 부르기로 하자. Squarethisnumber(5) 라고 입력하면 오류가 나는데 옥타브는 Squarethisnumber가 정의되지 않았다고 말해준다. 왜냐하면 옥타브가 이 파일을 어디에서 찾아야 할 지 모르기 때문이다. 그렇다면, pwd를 입력하면, 디렉토리가 없다고 나오므로, c: \users\wang\desktop를 검색해보자. 오타를 수정하면, 이제 Squarethisnumber(5)를 입력하면 답은 25라고 나온다.

06'47"

이건 좀 어려운 방법이라고 볼 수 있는데, search path 용어를 아는 사람만 참고하면 될 것 같다. 옥타브 search path를 변경하고 싶다면, 이를 고급용 선택 자료 정도로 생각하면 될 것이다. Search path 와 permit language 의 개념이 친숙한 사람이 참고하기를 바란다. 하지만 addpath, safety colon, c: \users\wang\desktop 을 활용해 옥타브 search path 디렉토리에 추가할 수 있다. 그래서 내가 지금 다른 디렉토리에 가려고 해도, 옥타브는 users ANG desktop 디렉토리에 가서 함수를 검색할 것이다. 즉, 내가 다른 디렉토리에 있다고 해도, 옥타브가 이 square this number 함수를 어디에서 찾아야 할지 안다는 것이다. 하지만 이러한 search path 에 대한 개념이 익숙치 않다면, 걱정할 필요 없이 CD 명령어를 활용해서 함수가 저장된 디렉토리에 갈 수 있다면 잘 작동 될 것이다.

07'48"

다른 프로그래밍 언어에는 없는 옥타브가 가진 하나의 특징은 여러분이 함수를 정의할 수 있게 해서 다양한 값 또는 argument 를 얻을 수 있다는 것이다. 여기 예시가 있다. function $[y1, y2] = \text{squareandCubeThisNo}(x)$ 라고 정의했다. 이 말은 이 함수는 $y1$, $y2$ 값 두 개를 돌려준다는 것이다. 여기 이렇게 설정한 것처럼. $y1 = x^2$, $y2 = x^3$ 이를 통해 숫자가 2개 나온다는 것이다. 어떤 프로그래밍 언어를 사용하는지에 따라 C/C++ 자바에서는 함수는 한 개의 값만 돌려준다고 생각한다. 하지만 옥타브에서는 다양한 값을 돌려준다. 다시 옥타브 윈도우로 돌아와서, $[a,b] = \text{squareandCubeThisNo}(x)$ 입력하면, $a= 5$ 의 제곱, 25, $b= 5$ 의 세제곱 125가 나온다. 다양한 값이 필요한 함수를 정의하는 경우에는 옥타브가 편리하다.

08'56"

이번에는 함수의 복잡한 예시 하나를 보여주도록 하겠다. 이와 같은 (1,1) (2,2) (3,3) 데이터 세트가 있다고 하자. 이제 옥타브 함수를 정의하여 비용함수 $J(\theta)$ 값을 연산하여 다른 θ 값을 얻을 것이다. 먼저 데이터를 옥타브에 입력하자. $X = [1,1; 1,2; 1,3]$ 입력하면 이런 행렬이 나오는데, x 는 x_0 로 첫 번째 열이 이에 해당하고, 두 번째 열은 세 개의 학습 예시의 x 값이다. $y=[1;2;3]$ 으로 놓고, 이는 y 축 값이다. $\Theta = [0;1];$ 라고 하자. 여기 바탕화면에 비용함수 j 를 미리 정의해뒀는데, 이를 가져오면 이렇게 된다. function $J = \text{cost function } j(x, y, \theta)$ 이고, 여기에 구체적인 입력을 코멘트 해 놨다. $m = \text{size}(X, 1)$ 는 학습 예시 수이므로, x 의 행의 수이다. Predictions 를 연산하면, $\text{predictions} = x * \theta$ 와 같고, 순환되는 코멘트이므로, 아마도 진행중인 코멘트 라인일 것이다. sqrErrors 에서 y 값은 이렇게 $(\text{predictions} - y)^2$, 마지막으로 이와 같이 비용 함수 J 를 연산하고 있다. 그리고 옥타브는 J 가 내가 원하는 값이라는 것을 알고 있다. 왜냐하면 J 가 여기 함수의 정의에 나타나있기 때문이다.

10'31"

강의 도중에 영상을 일시 정지 하여, 이 함수 정의를 좀 더 살펴보며 이 단계를 확실히 이해하는 것을 추천한다. 이를 옥타브에서 구현하면, $j = \text{cost function } j(X, y, \theta)$ 가 된다. 연산하면, 오타가 있었다. 대문자 X여야 한다. 다시 연산하면, $J = 0$ 으로 나오는데, 이는 데이터 세트가 123 123 이렇게 있었는데, $\theta_0 = 0$ 이고, $\theta_1 = 1$ 이기 때문에 이는 내 데이터 세트가 완벽하게 45도 각도로 그려지기 때문이다. 반대로, 만약 $\theta = [0;0]$ 라고 하면, 이 가정은 모든 것을 0으로 같게 예측할 것이다. $\theta_0 = 0$, $\theta_1 = 0$ 이 되고, 비용함수는 연산하면, 2.333 이 나온다. 그리고 이는 $(1^2 + 2^2 + 3^2) / (2*3)$ 값과 같다. 그래서 이 함수에서 건전성 검증은 올바른 비용함수를 연산했는지를 알아보는 것이다. 이렇게 간단한 학습 예시 몇 가지를 살펴봤다.

건전성 검증을 하면 여기에 정의된, 비용함수 J 를 정확한 비용 함수를 연산 한 것처럼 보인다. 최소한 우리의 간단한 학습 세트에서 학습 예시였던 x, y 값을 구했기 때문이다.

12'25"

이제 우리는 for 루프, while 루프, if 와 같은 제어문을 옥타브에서 어떻게 활용하는지 또한 함수를 정의하고 활용하는 방법에 대해서 배워봤다. 다음 강의에서는 이번 강의에 필요한

로지스틱 작용과 submitting problem 세트 그리고 이 submission 시스템을 어떻게 활용하는지에 대해 간단히 살펴본 후에, 옥타브에 관한 마지막 강의로, 벡터화, 즉, 옥타브 프로그램에 빠르게 돌아갈 수 있게 만들어주는 아이디어에 대해 다룰 예정이다.

No.	Title
Week 2-13	Vectorization

00'00"

이번 강의에서는 벡터화의 개념에 대해 알아보고자 한다. 여러분이 옥타브 또는 맷랩, 파이톤, 엔파이어, R, 자바, C++ 등과 같은 다른 프로그래밍 언어에도 내장되어 있거나 다양한 선형 대수학 라이브러리에 쉽게 접근 가능하다. 이들은 매우 잘 쓰였고, 최적화 되어 있는데 이는 보통 수치 계산학으로 박사학위를 가지고 있거나 이 분야에 특출한 사람들이 개발하기 때문이다. 여러분이 기계 학습 알고리즘을 구현할 때, 이러한 선형 대수학 라이브러리를 활용할 수 있고, 직접 코드를 입력하는 것보다 이러한 라이브러리가 대신할 수 있도록 주기적으로 요청을 할 수 있다면 편리할 것이다. 그렇게 되면, 코드를 얻게 되고, 첫 번째로 이를 빠르게 구현할 수 있고, 여러분의 컴퓨터가 어떠한 병렬 하드웨어를 가지고 있을지라도 이 장점을 잘 활용할 수 있다. 둘째, 이는 또한 적은 수의 코드를 적어도 되고, 간단하게 구현할 수 있으므로 비용을 들이지 않아도 된다는 의미이다. 예를 들어, 행렬을 곱하기 위해 직접 코드를 입력하는 것보다는 옥타브에 $a*b$ 를 입력하면 매우 효율적인 방법을 통해 두 행렬을 곱해줄 것이다. 여러분이 벡터화 방법을 구현한다면 더 간편하고 효율적인 코드를 활용할 수 있는 이와 비슷한 예시가 매우 많다. 먼저 예시를 살펴보자.

01'32"

여기에 선형 회귀 가설이 나와 있는데, 여기 $h(x)$ 를 연산하고 싶으면, 오른쪽에 이렇게 합계가 있다. 그러므로 $j = 0$ 부터 $j = n$ 까지 계산해주면 된다. 또 다른 방법은 $h \theta (x) = \theta^T x$ 라고 하면, 두 벡터 사이의 이 값을 연산하면 되는데 $\theta = [\theta_0; \theta_1; \theta_2]$ 이고, feature이 2개이고 $n = 2$ 일 때, $x = [x_0; x_1; x_2]$ 가 된다. 여기 두 식은 서로 다른 구현 방식을 보여줄 것이다. 이는 다음과 같다. 먼저 벡터화하지 않고 $h \theta (x)$ 를 연산 방법이다. Unvectorize 는 벡터화하지 않았다는 것이다. 먼저 $\text{prediction} = 0.0$; 이처럼 초기화하겠다. 이는 결국 $h \theta (x)$ 를 말한다. 그리고 for 루프를 적용해 $\text{for } j = 1:n+1$, $\text{prediction} = \text{prediction} + \theta(j) * x(j)$ 로 증가한다. 여기 이와 같은 표현이 된다. 여기 써놓은 벡터에 0 인덱스를 사용했는데, $\theta_0; \theta_1; \theta_2$ 이렇게 있다. 왜냐면 맷랩에서는 1 인덱스를 사용하여, θ_0 은 θ_1 , 두 번째는 θ_2 그리고 세 번째 원소는 θ_3 이 된다. 우리가 여기서

θ 와 x 를 0부터 시작한다고 써놓더라도 맷랩에서 벡터의 인덱스는 1부터 시작하기 때문이다. 그래서 여기서 for 루프를 사용하는 것이다. 여기 보면 j 가 0부터 n 이 아니라 $j = 1:n+1$ 이다. 하지만 이는 벡터화하지 않은 구현방법이므로 여기서는 for 루프가 원소 n 의 합계를 낼 것이다.

03'34"

이번에는 반대로 벡터화를 통해 구현을 하는 방법을 알아보겠다. 여기서 x 와 θ 를 벡터로 놓고, $\text{prediction} = \theta^T x$ 로 놓고 $\theta^T x$ 를 연산하는 것이다. 이번에는 여기 for 루프 코드를 다 쓸 필요 없이, 딱 한 코드 한 줄만 쓰면 된다. 여기 오른쪽 코드를 가지고 옥타브에 고도로 최적화된 선형 대수학 루틴을 활용해 이 두 벡터, θ 와 x 사이에 있는 내적을 계산하려고 한다. 이렇게 벡터화 구현은 간편할 뿐 아니라, 효율적으로 구현된다.

04'16"

옥타브에서 벡터화 과정을 살펴봤고, 하지만 벡터화는 다른 프로그래밍 언어에서도 활용가능하다. C++ 예시를 살펴보자. 여기 벡터화하지 않은 구현을 먼저 보여주겠다. 먼저 prediction 은 double $\text{prediction} = 0.0;$ 로 초기화하고 for ($\text{int } j = 0, j <= n, j++$), $\text{prediction} += \theta[j] * x(j);$ 인데 아까처럼 이렇게 for 루프를 직접 써줘야 한다. 반면에, C++에서 유용한 선형 대수학 라이브러리를 활용한다면, 이렇게 쓸 수 있다. 아니면, 반면에, C++에서 유용한 선형 대수학 라이브러리를 활용한다면, 이렇게 쓸 수 있다. 따라서 선형 대수학 라이브러리의 내용에 따라서 어떠한 객체를 가지는데, C++ 에서는 벡터 θ 와 벡터 x 가 있다. 그러므로 $\theta.\text{transpose} * x$ 만 적용해주면 되는데, 여기 * 곱셈으로 C++가 과부하 될 수 도 있으니, 이 두 벡터만 곱해주면 된다. 어떠한 선형 대수학 라이브러리를 사용하느냐에 따라, 조금 다른 문법을 사용할 수는 있겠지만, 이러한 내적을 연산하는데 라이브러리를 활용한다면, 더 간편하고 효율적인 코드를 활용할 수 있게 된다.

05'40"

이번에는 좀 더 복잡한 예시를 알아보자. 다시 한번 짚고 넘어가자면, 선형 회귀에서 기울기 하강에 대한 갱신 규칙이 있다. 따라서 우리는 모든 j 의 값 $j = 0, 1, 2$ 등에 이 규칙을 활용하여 θ j 를 갱신할 것이다. 여기 이렇게 $\theta_0; \theta_1; \theta_2$ 방정식이 있고, 우리에게 두 feature이 있다고 하자. 그러면 $n = 2$ 가 된다. 이 방정식들은 우리가 $\theta_0; \theta_1; \theta_2$ 를 연산할 때 쓸 업데이트다. 이에 대해 강의 초반에 언급했었는데, 이는 동시 업데이트다. 이제 여기에 벡터화를 시켜보자.

06'20"

여기에 같은 방정식 세 개가 조금 작은 폰트로 쓰여 있다. 이 세 줄의 코드를 구현하는 방법은 for 루프를 사용하는 것인데, $j = 0, 1, 2$ 를 활용해 θ j 를 갱신할 수 있다. 하지만 이러한 방법 대신에 벡터화를 통해 이 코드 세 줄을 압축하거나 for 루프를 활용해 이 3단계를 한 번에 처리할 수 있는 간편한 방법을 알아보도록 하자. 먼저 이 3단계를 압축해서 하나의 벡터화된 코드로 만들어보자. 방법은 다음과 같다. θ 를 벡터로 놓고, $\theta := \theta - \alpha\delta$ 이고, 여기서 델타(δ)는 이렇게 오른쪽에 적은 것과 같다. 이제 이 과정을 살펴보자. 여기서 나는 θ 를 벡터로 놓고,

여기서 벡터는 $n+1$ 차원 벡터다. 여기 있는 θ 도 벡터, $R(n + 1)$ 로 간신되고, 알파는 실수이고, 델타 또한 벡터다. 따라서 이는 감산 작업이다. 벡터 감산이라고도 한다. 여기 $\alpha\delta$ 는 벡터고, θ 가 여기 $\alpha\delta$ 벡터에서 감산을 하는 것이다.

08'04"

그렇다면, 벡터 델타는 무엇인가? 여기 벡터 델타는 이렇게 생겼는데, 이는 사실 여기 있는 모든 것을 합한 것이다. 즉, 델타는 $n+1$ 차원 벡터이고, 이 벡터의 가장 첫 번째 원소는 바로 이것과 같게 될 것이다. 따라서, $\delta = [\delta_0; \delta_1; \delta_2]$ 가 있다면, δ_0 가 여기 초록색 박스 안의 식과 같아지게 하려고 한다. 그렇게 되면, δ_0 는 다음과 같은 식과 같다는 것을 알게 될 것이다. 우리가 지금 델타가 어떻게 연산이 되는지에 대한 같은 내용을 이해하고 있도록 하자. 델타는 여기 표시된 부분의 합이고, 여기 합은 무엇인가? 여기 표시한 부분은 실수이고, $x(i)$ 이것은 벡터다. 그래서 $x(i) = [x(i)_0; x(i)_1; x(i)_2]$ 이러한 벡터가 된다. 그렇다면 이 합은 무엇인가? 여기서 합은 이 부분이 $(h \text{ of}(x(1)) - y(1)) * x(1) + (h \text{ of}(x(2)) - y(2)) * x(2) + \dots$, 이와 같은 식이라는 것이다. 왜냐하면 이는 i 값의 합이고, i 의 범위는 1부터 m 까지 인데, 이는 여기 이러한 항의 합을 구하는 것이다.

10'14"

그리고 이러한 항의 의미는 이전에 이에 관한 퀴즈와 매우 흡사하다는 것을 떠올릴 수 있을 것이다. 바로 이러한 방정식을 봤을 것이다. 그 때 우리는 이 코드를 벡터화하여, $u = 2v + 5w$ 를 대신 사용하였다. 그러니까 벡터 u 는 2 곱하기 벡터 v 더하기 5 곱하기 벡터 w 와 같다. 이것이 바로 다른 벡터를 더하는 예시였고, 여기 합을 내는 것도 결국 같은 방법이다. 여기 합은 실수를 더하라는 것이다. 여기 숫자 2 아니면 다른 숫자 곱하기 벡터 x_1 이므로, $2v$ 또는 다른 숫자 곱하기 x_1 을 할 수 있다. 그 뒤에 덧셈이 있는데, $5w$ 대신에 다른 실수나 다른 벡터를 더할 수 있고, 그 뒤에도 같은 방법으로 다른 벡터를 계속해서 더해나갈 수 있다. 그러니까, 여기 전체, 그 델타가 어떠한 벡터라는 것이다.

11'22"

다시 말해, 델타의 세 원소는 $n = 2$ 일 때, 델타의 세 원소는 이곳, 두 번째, 세 번째에 대응한다는 것을 알 수 있다. 이는 $\theta := \theta - \alpha\delta$ 에서 θ 를 간신하면, 여기 위에 있는 간신 규칙에 따라서 동시에 간신하게 되는 것이다. 이 슬라이드에서 많은 내용을 살펴봤는데, 내용 이해가 부족하다면 강의를 잠시 멈추고 왜 이 간신과, 여기 델타의 정의 그리고 이것이 왜 이 수식과 같은지 다시 한번 살펴보는 것을 추천한다. 그런데도 이해가 안될 수도 있으니, 여기 이 항들은 벡터 x 와 같다. 그래서 우리가 이 세 가지 연산을 가져가서 한 단계인 벡터 델타로 압축하였고, 이게 바로 선형 회귀 단계에서 이러한 방법으로 벡터화 구현을 한 이유다.

11'37"

이러한 단계를 잘 이해하길 바라면서 영상을 다시 보면서 이해했는지 확인해보자. 만약 이러한 매팽이 잘 이해가 가지 않을 경우, 이 수식을 구현하면, 어쨌든 정답이 나오기는 한다. 그렇기 때문에 이 등가를 잘 이해하지 못했다고 해도, 이러한 방식으로 구현한다면, 선형 회귀를 활용할

수 있을 것이다. 하지만 왜 이 두 단계가 같은 것인지를 이해한다면, 이는 벡터화에 대해 잘 이해할 수 있도록 해줄 것이다.

Q) 다음과 같이 u , v , w 값을 지닌 벡터 세 개가 있다.

여러분은 다음과 같이 코드를 구현할 것이다.

이 코드를 어떻게 벡터화할 것인가?

정답 2번

13'09"

마지막으로 선형 회귀에서 1개나 2개 이상의 feature를 활용하여 구현한다면, 때로는 선형 회귀에서 수십, 수백, 또는 수천 개의 feature를 활용하기도 한다. 하지만 선형 회귀에서 벡터화 구현을 활용한다면, 이전의 00, 01, 02 를 직접 간신했던 for 루프 보다 훨씬 더 빨리 처리할 수 있다는 것을 알 수 있다. 따라서 벡터화를 활용하면, 선형 회귀에서 훨씬 더 효율적으로 작업할 수 있다. 이후 강의에서 등장하게 될 알고리즘을 벡터화하면, 옥타브나 C++, 자바와 같은 다른 프로그래밍 언어에서 활용할 수 있는 간편한 트릭도 있어서, 더 효율적으로 코드를 얻을 수 있을 것이다.

No.	Title
Week 2-14	Working on and Submitting Programming Assignments

00'00"

이번 강의에서는 이번 강좌 과제를 어떻게 작성하고, 기계학습 연습문제에 정답을 바로 확인할 수 있게 해주는 등록 시스템에 대한 로지스틱을 다룰 예정이다. 여기 옥타브 윈도우가 있다. 먼저 바탕화면으로 가자. 첫 번째 연습문제를 바탕화면에 있는 파일에 저장해두었다. 바로 이 디렉토리다. 'ml-class-ex1'. 우리는 다양한 파일을 제공하고 이를 중 몇 개를 수정하라고 요청할 것이다. 첫 번째 파일은 프로그래밍 연습문제에서 pdf 파일로 요구사항을 수행해야 했다. 하지만 우리가 수정하라고 요청한 파일 중 하나가 warmUpExercise.m 파일이 있는데, 이 연습문제는 등록 시스템을 잘 알고 있는가를 알아보기 위한 것이었다. 여러분은 5x5행렬로 돌아가기만 하면 됐다. 그래서 이 연습문제의 정답을 보여주겠다. $A = \text{eye}(5)$ 를 입력하면 된다. 따라서 이는 이 함수를 변경하여, 5x5 항등 행렬을 생성했다. 그러면 여기 warmUpExercise() 함수는 5x5 항등

행렬이 된다. 이제 이를 저장하겠다.

01'13"

이제 연습문제 첫 번째 파트를 마쳤다. 이제 옥타브 원도우로 다시 돌아와서, 디렉토리로 가보자. 'C:\Users\Wang\Desktop\ml-class-ex1'. 내가 이를 구현했다는 것을 확인하고 싶으면, 이렇게 'warmUpExercise()' 를 입력하면 된다. 그러면 이렇게 우리가 생성하려고 입력했던 5x5 행렬이 나타난다. 이제 다음과 같이 코드를 입력해보겠다. 'submit()' 를 이 디렉토리에 입력하고, 이제 파트 1을 제출할 준비가 되었다. Choice 1을 입력하면 이메일 주소를 입력하라고 한다. 그러면 강의 웹사이트에 들어가서, 이것은 내부 테스팅 사이트이기 때문에 여러분의 웹사이트 버전과 조금 다를 수 있다. 이렇게 이메일 주소를 입력하고, 제출 비밀번호를 여기에 입력하면 된다. 이메일을 입력하고, ang@cs.stanford.edu 비밀번호를 9yC75USsGf 입력한다. 이제 엔터키를 누르면 서버에 연결이 되어 등록이 완료된다. 그러면 이러한 안내말이 바로 나온다. "축하합니다. 과제 1 파트 1 제출을 성공적으로 완료하셨습니다." 이는 이 파트를 알맞게 제출했다는 확인을 받은 것이다. 만약 정답을 제출하지 않은 경우, 제대로 제출되지 않았다는 의미의 메시지가 나올 것이다.

02'40"

또한 이 제출 비밀번호를 활용하여 새로운 비밀번호를 생성할 수 있다. 크게 상관없기는 하지만, 다른 웹사이트 로그인 비밀번호를 사용해도 되지만, 여기 보면 모니터에 비밀번호가 그대로 보이기 때문에 여기서 다른 등록 비밀번호를 제공하는 것이다. 여러분의 평소 비밀번호를 원도우에 입력하고 싶지 않을 수 있기 때문이다. 이는 작동 시스템에 따라 옥타브 등록 스크립트에 입력할 때 비밀번호가 텍스트로 보여질 수도 아닐 수도 있다.

03'14"

이것이 바로 과제를 마치고 나서 등록하는 방법이다. 행운을 빈다. 그리고 과제를 할 때 모두 정답을 맞추기를 바란다. 마지막으로 다음 강의에서 이는 마지막 옥타브에 대한 강의가 될 것이다. 벡터화에 대해 이야기하면서 옥타브 코드를 훨씬 더 효율적으로 활용하는 방법을 강의하겠다.

Week 3

No.	Title
Week 3-1	Classification

00'00"

이번 강의와 이후 강의에서는 여러분이 값을 예측하고자 하는 변수 y 분류 문제에 대해서 이야기하려고 한다. 로지스틱 회귀라는 알고리즘을 개발해 볼 것이다. 이는 오늘날 가장 인기 있고 널리 사용되고 있는 학습 알고리즘 중 하나다.

00'19"

여기에 분류 문제 예시가 있다. 전에 이메일 스팸 분류를 분류 문제의 예시라고 말했다. 또 다른 예시에는 온라인 트랜잭션 분류가 있다. 웹사이트에서 물건을 팔고 특정한 온라인 거래가 사기인지 아닌지, 어떤 사람이 훔친 신용카드를 사용하는지, 아니면 다른 사람의 비밀번호를 훔쳐 사용하는 것인지를 알고 싶다. 이런 게 분류 문제다. 좀 전에 종양이 암인지, 악성인지, 양성인지를 분류하는 예시에 대해서도 이야기해봤다.

00'55"

이러한 모든 문제에서 우리가 예측하려고 하는 변수는 바로 변수 y 다. 그래서 두 값 중에 0인지 1인지, 스팸인지 아닌지, 사기인지 아닌지, 악성인지 양성인지 둘 중에 고르는 형식이다. 0을 분류하는 또 다른 이름은 negative 분류이다. 그리고 1을 분류하는 또 다른 이름은 positive 분류이다. 그러니까 0은 양성 종양으로 분류하고, 1은 positive 분류라서 이를 악성 종양으로

분류한다. 스팸메일의 여부와 같이 두 분류를 배정할 때 두 분류를 positive와 negative로 구분짓는 것은 임의적 분류로 보일 수 있다. 보통 negative는 뭔가 '없다'는 의미를 전달한다. 악성 종양이 존재하지 않을 때 negative이다. 반면에 positive 분류 1은 뭔가 존재한다는 것을 의미한다. 그렇지만 어떤 게 negative인지 또는 positive인지는 임의적인 것이라 크게 중요하지는 않다.

02'00"

이제 분류 문제를 0과 1, 두 분류를 통해 이야기해보겠다. 그 후에는 y 가 0,1,2,3 와 같이 4개의 값을 가지는 것과 같은 다중 클래스 문제에 대해 이야기 해볼 것이다. 이는 다중 클래스 분류 문제라고 불린다. 먼저 이후 강의를 위해 2 클래스 또는 2항 분류 문제에 대해서 먼저 다룰 테니, 다중 클래스는 이후에 걱정해도 된다. 그렇다면 분류 알고리즘은 어떻게 개발하는가? 여기 종양이 악성인지 양성인지를 분류하는 분류 문제 학습 세트 예시가 있다. 여기서 악성은 두 값만 가지는데, 여기서 0은 '아니다'이고, 1은 '맞다'이다. 따라서 주어진 학습 세트를 가지고 우리가 알고 있는 알고리즘에 적용해보자.

02'51"

이 데이터 세트에 선형 회귀를 적용하고, 데이터에 직선을 그려보겠다. 이 학습 세트를 가지고 직선을 그려 넣으면, 이렇게 생긴 가정을 얻을 수 있을 것이다. 여기 $h_{\theta}(x) = \theta^T x$ 가 내가 세운 가정이다. 여기에 예측을 할 때 분류의 기준을 0.5로 잡으면, 수직선 값은 0.5가 된다. 그리하여, 가정의 결과값이 0.5 이상이면 $y=1$ 이 되는 것이고, 0.5 보다 작은 경우는 $y = 0$ 이 된다. 이제 어떻게 되는지 살펴보자. 0.5는 한계점이 되고, 이는 선형 회귀를 이렇게 활용한다. 이 곳을 기준으로 오른쪽은 positive 크로스를 예측할 것이다. 왜냐하면 수직선 위에서 출력 값은 0.5보다 크기 때문이다. 그리고 이 점의 왼쪽은 negative 값을 예측할 것이다.

03'55"

이 예시에서, 선형 회귀는 무언가 합리적인 가정으로 보인다. 이 예시가 우리가 관심 있는 분류 토스 임에도 불구하고 말이다. 하지만 이제 이 문제를 조금 바꿔보자. 수평적 접근을 조금 연장해서, 여기 오른쪽에 또 다른 학습 예시를 추가해보자. 여기 보이는 새로 추가된 학습 예시는 사실 아무 것도 바꾸지 않았다. 학습 세트를 보면 이것이 훌륭한 가정임이 분명해 보인다. 이 곳을 기준으로 오른쪽에 있는 것은 이는 positive라고 예측할 수 있다. 그리고 왼쪽은 아마도 negative라고 예측해야 할 것이다. 왜냐하면 여기 학습 세트를 보면, 종양이 이 곳에 있는 특정한 값보다 크다면 악성이고, 이것보다 작은 종양은 악성이 아니라고 나와있다.

04'46"

하지만 여기에 또 다른 예시를 추가하고, 선형 회귀를 구현하면, 데이터에 맞는 직선을 얻게 될 것인데, 아마 이렇게 생겼을 것이다. 한계값 가설이 0.5에 설정되어있고, 여기 이 곳을 중심으로 오른쪽은 positive로 예측되고, 왼쪽에 있는 것은 negative로 예측된다. 이는 선형 회귀가 매우 잘

안된 것처럼 보이는데, positive 예시와 negative 예시가 있기 때문이다. 이 두 분류가 이 곳 즈음에서 나눠져야 하는데, 여기 오른쪽에 하나의 예시가 추가되는 바람에, 이 예시는 아무런 정보도 주지 않고 있다. 즉, 학습 알고리즘에서는 갑자기 나타나는 내용이 있어서는 안 된다. 그러니 여기 추가된 예시는 악성이라고 나온다. 하지만 이 예시가 추가됨에 따라 선형 회귀가 데이터에 맞는 직선을 자홍색 직선에서 파란색 직선으로 바꿨고, 이를 잘못된 가정으로 만들어 버렸다.

05'56"

그러므로 선형 회귀를 분류 문제에 적용하는 것은 보통 그다지 좋은 생각은 아니다. 첫 번째 예시에서는, 새로운 학습 예시를 추가하기 전에는, 이전 선형 회귀에서 특정한 예시에 대한 가정이 들어 맞았기 때문에 운이 좋았다고 볼 수 있다. 하지만 대개 선형 회귀를 데이터 세트에 적용했는데 들어 맞는 경우는 흔치 않다. 그래서 나는 선형 회귀를 분류 문제에서는 활용하지 않는다.

06'29"

여기 선형 회귀를 분류 문제에서 활용했을 때 나타날 수 있는 또 다른 이상한 예시를 보여주겠다. 분류에서 $y = 0$ 또는 1이다. 하지만 가설이 값을 출력할 수 있는 선형 회귀에서는 모든 학습 예시에 $y = 0$ 또는 1이라고 적어 놓을 경우에도 이는 1보다 훨씬 크고, 0보다 작다. 우리가 이미 $y = 0$ 또는 1이라고 적힌 것을 봤는데도 알고리즘이 1보다 훨씬 크거나 0보다 훨씬 작은 값을 출력할 수 있을 때, 이는 앞뒤가 맞지 않는다.

Q) 다음 중 옳은 것을 고르시오.

정답 4번

따라서 이후 강의 영상에서 로지스틱 회귀라 불리는 알고리즘은 개발해 볼 건데, 이는 선형 회귀의 예측 출력값이 항상 0과 1사이에서 나오고, 1보다 커지거나 0보다 작아지지 않는다.

07'26"

로지스틱 회귀를 분류 알고리즘에서 사용하는데, 로지스틱 회귀가 사실 분류 알고리즘이에도 불구하고 회귀라는 용어가 이름에 들어있기 때문에 혼란스러울 수 있다. 하지만 이는 그저 예전부터 사용해온 이름일 뿐이다. 그러니 로지스틱 회귀는 실제로 분류 알고리즘이고 y 는 0 또는 1인 이산값이라고 설정하는 것을 혼동하지 말자. 지금쯤이면 분류 문제에서 왜 선형 회귀를 적용하는 것이 좋지 않은지에 대해 이해하기를 바란다. 다음 강의에서는 로지스틱 회귀 알고리즘에 대해 자세히 알아보겠다.

No.	Title
-----	-------

00'00"

로지스틱 회귀에 대해 이야기해보자. 이번 강의에서는 가설 표현에 대해 이야기해보겠다. 이는 우리가 활용할 함수가 우리가 분류 문제가 생겼을 때, 이 가설을 어떻게 표현할 것인가에 대한 것이다. 이전에 분류기에서 출력한 값은 0과 1사이 값만 내보내려고 한다. 따라서, 이러한 예측이 0에서 1 사이일 거라는 특징을 만족시키는 가설을 생각해 냈다. 우리가 선형 회귀를 사용할 때, 활용할 수 있는 가설의 형태는 $h_{\theta}(x) = \theta^T x$ 이다. 로지스틱 회귀에서는 이를 조금 변형하여 가설을 만들어 보겠다. $h_{\theta}(x) = g(\theta^T x)$ 로 만들어서 함수 g 를 다음과 같이 정의하려고 한다. $g(z)$ 는 여기서 z 는 실수이다. $g(z) = \frac{1}{1+e^{-z}}$

00'58"

이 함수는 시그모이드 함수 또는 로지스틱 함수라고 불리는데, 로지스틱 함수(logistic function)라는 용어에서 로지스틱 회귀(logistic regression)라는 이름이 생겨났다. 시그모이드 함수와 로지스틱 함수는 말하자면 같은 것을 의미하는 유의어다. 따라서 두 용어는 기본적으로 서로 바꿔 쓸 수 있고, 두 용어 중 어떠한 것도 g 함수를 대신해 쓸 수 있다. 만약 우리가 이 두 방정식을 가지고 하나로 합쳐보면, 이는 내 가정을 다르게 쓰는 방법이 된다. 따라서, $h_{\theta}(x) = \frac{1}{1+e^{-\theta^T x}}$ 가 된다. 다음에는 변수 z 를 가지고 여기서 z 는 실수를 의미한다. 여기에 z 대신에 $\theta^T x$ 를 대입한다. 이렇게 z 대신에 $\theta^T x$ 를 대입해봤다.

01'55"

다음으로 시그모이드 함수가 어떻게 생겼는지 보여주겠다. 여기 있는 수치를 그래프화 하려고 한다. 여기 시그모이드 함수 $g(z)$ 는 또한 로지스틱 함수라고도 부르고, 이렇게 그려진다. 0 근처에서 시작해 증가 곡선을 그리다가 원점 0.5를 지나서 다시 이렇게 평행을 그리게 된다. 이게 바로 시그모이드 함수 그래프다. 시그모이드 함수가 1에 점근하고, 0에 점근하는 동안, 여기서 가로축은 z 다. z 가 마이너스 무한대로 갈수록, $g(z)$ 는 0에 가까워진다. 그리고 $g(z)$ 가 무한대로 커질수록, $g(z)$ 는 1에 가까워진다. $g(z)$ 의 세로값이 0과 1사이이기 때문에, $h_{\theta}(x)$ 값 또한 0에서 1사이인 것이다.

02'47"

이번에는 이와 같은 가설 표현을 가지고 우리가 해야 할 것은 먼저 매개 변수 세타를 데이터에 대입하는 것이다. 따라서 주어진 학습 세트를 가지고 매개 변수 세타값을 고르고, 이 가설은 우리가 예측을 할 수 있게 해준다. 매개 변수 세타에 대입하는 학습 알고리즘에 대해서는 나중에 자세히 살펴보겠다. 먼저 이 모델을 해석하는 것에 대해 이야기해보자.

03'18"

이제 $h_{\theta}(x)$ 가설에 대한 출력을 해석해 보겠다. 이 가설이 어떠한 숫자를 출력하면, 이 숫자를

예시 x 라는 새로운 입력에 대한 $y = 1$ 라는 추측 확률이라고 간주할 것이다. 예를 들어 다시 설명해주겠다. 종양 분류 예시를 활용한다고 할 때, 벡터 x 특징을 가지고, 여기서 x_0 는 항상 1과 같다. 그리고 특징 하나는 종양의 크기다. 환자가 들어왔는데 종양 크기가 있고, 이 가정에 벡터 x 특징을 대입해보겠다. 그리고 가설이 출력한 숫자가 0.7이라고 가정한다면, 가설을 다음과 같이 해석할 것이다.

04'08"

이 가설은 내게 특징 x 를 보여주는 환자의 경우, $y = 1$ 일 확률은 0.7이다. 즉, 환자에게, 슬프지만, 종양이 70퍼센트의 확률로, 또는 0.7의 비율로 악성일 가능성이 있다고 말할 것이다. 이를 수학적으로 표현한다면, 이 가설의 출력은, $h_{\theta}(x) = p(y=1 | x; \theta)$ 이 된다. 확률에 익숙한 사람이라면, 이러한 방정식을 이해할 수 있을 것이다. 만약 확률을 잘 모른다고 한다면, 이러한 수식을 이렇게 이해할 수 있다. 이건 $y = 1$ 일 확률이다. 주어진 x 는 환자가 특징 x 를 가지고 있다는, 그러니까 환자가 특징 x 로 표현되는 종양 크기를 가지고 있다는 것이다. 그리고 이러한 확률은 세타로 매개 변수화 된다.

05'07"

이제 가설을 가지고 $y = 1$ 이라는 확률을 예측하는 계산을 해보겠다. 이것이 분류 문제 이기 때문에, 우리는 $y = 0$ 또는 1 여야 한다는 것을 알고 있다. 이 두 값만이 학습 세트나 미래에 내 사무실이나 병원에 올 새로운 환자에게서 y 가 취할 수 있는 값이다. 따라서 여기 $h_{\theta}(x)$ 를 가지고 $y = 0$ 일 확률 또한 연산할 수 있는데, y 는 무조건 0 또는 1이기 때문이다.

Q) 데이터 x 를 가지고 종양이 악성 ($y=1$)인지 양성 ($y=0$)인지 예측하려고 한다. 로지스틱 회귀 분류기는 특정 종양에 대해 다음과 같이 $h_{\theta}(x) = p(y=1 | x; \theta) = 0.7$ 출력한다. 이는 종양이 악성일 확률이 70%가 된다는 것이다. 그렇다면 $h_{\theta}(x) = p(y=0 | x; \theta)$, 즉 종양이 양성일 확률은 얼마인가?

(정답 1번)

우리는 $y = 0$ 일 확률 더하기 $y = 1$ 일 확률이 무조건 1이어야 한다는 것을 알고 있다. 이 첫 번째 방정식은 좀 복잡해 보인다. 이는 말하자면, 매개변수 세타가 주어졌을 때, 특징 x 를 지닌 특정 환자가 $y=0$ 일 확률을 말해준다.

06'00"

여기에 매개변수 세타가 주어졌을 때, 같은 환자가 $y=1$ 일 확률을 더해주면 반드시 1이 나와야 한다. 만약 이 방정식이 복잡해 보인다면, 머릿속으로 x 와 θ 를 지워보라. 그렇게 되면, $y=0$ 일 확률과 $y = 1$ 일 확률을 더하면 1이 된다는 것이다. 우리는 이것이 참이라는 것을 알고 있는데, 이는 y 는 0이나 1이어야 하기 때문이다. 따라서 $y = 0$ 일 확률과, $y=1$ 일 확률은 더하면 1이 되는 것이다. 이 둘을 합치면 반드시 1이 되어야 한다. 이 부분을 가져다가 오른쪽으로 옮기면, 이러한

방정식이 된다. 이는 $y = 0$ 일 확률은 $1 - y = 1$ 일 확률과 같다라는 이 부분과 같아지는 것이다. 따라서 x 의 가설 특징이 여기 이 내용이라면, $y = 0$ 이라는 측정 확률 또한 간단하게 연산 가능하다.

06'55"

이제 여러분은 로지스틱 회귀에서 가설 표현이 무엇인지, 수학적 공식이 무엇인지, 로지스틱 회귀에서 가설의 정의가 무엇인지에 대해 알아봤다. 다음 강의에서는 가설 함수가 어떤 것인가에 대해 좀 더 자세히 살펴보고자 한다. 그리고 결정경계라 불리는 용어에 대해 알아보겠다. 또한, 로지스틱 회귀에서 가설 함수가 어떻게 생겼는지 시각화 하여 그 개념에 대해 더 자세히 알아보도록 하겠다.

No.	Title
Week 3-3	Decision Boundary

00'00"

지난 강의에서는 로지스틱 회귀에서 가설 표현에 대해 알아보았다. 이번에는 결정 경계라는 것에 대해 이야기하고자 한다. 이는 로지스틱 회귀 가설 함수가 어떠한 것을 연산하는지에 대해 더 자세히 알 수 있도록 해준다. 다시 정리해보면, 이는 지난 시간에 배웠던 것으로, 가설은 $h_{\theta}(x) = g(\theta^T x)$, 나타낼 수 있고, 함수 g 는 $g(z) = 1/(1+e^{-z})$ 이렇게 된다. 그래프 상에는 0부터 시작해서 1까지 점근하는 이런 식으로 서서히 증가하는 그래프다.

00'38"

이제부터는 이 가설에서 $y = 1$ 일 때, 언제 예측을 하게 되는지 또는 $y = 0$ 인 경우에는 언제 예측을 하는지 더 자세히 알아보겠다. 그리고 특징이 2개 이상일 때 가설 함수가 어떻게 그려지는지도 알아보자. 즉, 이 가설은 확률의 예상치를 출력하는데, $p(y=1 | x; \theta)$ 이렇게 나타낼 수 있다. 만약 $y = 1$ 또는 $y = 0$ 이라고 예측하고 싶다면, 다음과 같이 하면 된다. 가설이 $h_{\theta}(x)$ 가 0.5보다 크거나 같으면 $y = 1$ 라고 출력하면, 이는 $y = 0$ 일 확률보다는 $y = 1$ 일 확률이 높다는 것이다. 그렇다면 $y = 1$ 을 예측해보자. 반대로 만약 y 가 1보다 클 예상 확률이 0.5 보다 작으면 이는 $y = 0$ 을 예측할 것이다. 따라서 여기서는 크거나 같다고 했고, 여기서는 작다고 했다. 만약 $h_{\theta}(x)$ 가 정확히 0.5가 되면, 이는 positive 또는 negative라고 예측할 수 있는데, 하지만 여기에 0.5와 같다는 허점을 만들어놨기 때문에, 만약 $h_{\theta}(x) = 0.5$ 라면, 이를 positive라고 예측하는 디폴트하게 되는데, 이러한 내용은 그게 중요하지는 않다.

01'56"

$h_{\theta}(x)$ 가 0.5보다 크거나 같은 경우가 정확히 언제인지 이해해서, $y = 1$ 이라고 예측할 수 있게 되는 것이다. 여기 시그모이드 y 함수 그래프를 보면, z 가 보다 크거나 같다면, $g(z)$ 는 0.5보다 크거나 같다는 것을 알 수 있다. 따라서 이 영역에서는 g 는 0.5와 같거나 더 큰 값을 가진다. 여기는 0.5라고 표시되어있다. 그래서 z 가 positive일 때, $g(z)$, 즉 시그모이드 함수는 0.5보다 크거나 같게 된다.

02'42"

로지스틱 회귀 가설인 $h_{\theta}(x) = g(\theta^T x)$ 는 따라서 $\theta^T x$ 가 0보다 크거나 같을 경우, 0.5보다 크거나 같게 된다. 여기 보면 $\theta^T x$ 가 z 의 역할을 대신하고 있음을 알 수 있다. 이를 통해 $\theta^T x$ 가 -보다 크거나 같을 경우, 가설은 $y = 1$ 를 예측하는 것을 알 수 있다. 이번에는 가설이 $y = 0$ 을 예측하는 다른 예시를 살펴보자. 비슷한 argument 가 있는데, $g(z)$ 가 0.5보다 작을 때, $h_{\theta}(x)$ 는 0.5 보다 작다. 이는 z 값의 범위에서 $g(z)$ 의 값은 0.5보다 작기 때문이다. 이는 z 가 negative 일 때다. 따라서 $g(z)$ 가 0.5 보다 작은 경우, 가설은 $y = 0$ 을 예측할 것이다. 그리고 아까 봤던 비슷한 argument 에서 $h_{\theta}(x) = g(\theta^T x)$ 이므로, $\theta^T x$ 값이 0보다 작은 경우, $y = 0$ 임을 예측할 수 있다.

04'04"

지금까지 살펴본 것을 요약해보면, 만약 우리가 $y = 1$ 또는 $y = 0$ 임을 예측하기로 결정하는 것은 예측 확률이 0.5보다 크거나 같은지 혹은 0.5보다 작은지에 달려있다. 따라서 $\theta^T x$ 가 0보다 크거나 같을 때, $y=1$ 이라고 예측하는 것과 같다. 또한 $\theta^T x$ 가 0보다 작을 경우 $y = 0$ 이라고 예측하게 될 것이다. 로지스틱 회귀 가설이 이러한 예측을 어떻게 만들어내는지 더 자세히 알아보자. 이 슬라이드에 보이는 것과 같은 학습 세트가 있다고 하자. 가설은 다음과 같다. $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$.

04'52"

매개 변수를 이 모델이 어떻게 대입하는 지에 대해서는 아직 이야기 하지 않았다. 이에 대해서는 다음 번 강의에서 설명하도록 하겠다. 명시된 과정을 통해서 우리는 다음의 값을 매개 변수로 선택하게 된다. 여기서 $\theta_0 = -3, \theta_1 = 1, \theta_2 = 1$ 로 정하자. 이는 매개 변수 벡터가 $\theta = [-3 ; 1 ; 1]$ 이렇게 된다. 따라서 가설 매개 변수가 이렇게 주어졌을 때, 가설 어느 부분에서 $y=1$ 이라고 예측하는지, 그리고 $y = 0$ 이라고 예측하는지 알아보자.

05'39"

아까 슬라이드에서 배운 공식들을 다시 보면, $y = 1$ 이라는 확률은 $\theta^T x$ 가 0보다 클 때, 0.5보다 크거나 같다. 그리고 여기 밑줄 친 공식은 $-3 + x_1 + x_2$, 그러니까 $\theta^T x$ 는 우리가 선택한 여기 이 매개 변수 값과 같다. 따라서 여기 x_1, x_2 는 이 방정식 $-3 + x_1 + x_2 \geq 0$ 을 만족시키는 어떠한 예시에서 우리가 내린 가설은 $y = 1$ 이라고 예측할 것이다.

06'32"

또한 -3을 오른쪽으로 가져와, $x_1 + x_2 \geq 3$ 이라고 쓸 수도 있다. 따라서 이 가설은 $x_1 + x_2 \geq 3$ 일 경우에 $y=1$ 이라고 예측할 것이다. 수치를 통해 이게 어떤 뜻인지 알아보자. $x_1 + x_2 = 3$ 이라고 할 때, 이 방정식은 그래프 상에 이런 식으로 직선으로 나타날 것이고, x_1 축과 x_2 축의 3을 지나게 된다.

07'16"

인포스페이스 부분은, 즉 x_1, x_2 면 부분은 $x_1 + x_2 \geq 3$ 와 대응하게 되고, 이는 여기 직선 기준 오른쪽 절반을 의미한다. 여기 그려진 자홍색 선을 기준으로 오른쪽은 전부 여기에 속하게 된다. 따라서 가설이 $y = 1$ 이라고 예측하는 부분은 이 부분이고, 여기 오른쪽 위에 큰 공간을 의미한다. 적어보겠다. 이 곳을 $y = 1$ 공간이라고 칭하겠다. 반면에 $x_1 + x_2 \leq 3$ 인 공간에서는 $y = 0$ 이라고 예측할 것이다. 그리고 이는 이 공간을 의미한다. 여기 반쪽 면, 여기 원쪽 지역에서 가설은 $y = 0$ 이라고 예측할 것이다. 여기 자홍색 선에다가 이름을 붙여주려고 한다. 이 직선이 바로 결정 경계인 것이다.

08'24"

즉, 여기 $x_1 + x_2 = 3$ 직선은 좌표 세트와 대응하게 되는데, 이는 $h_{\theta}(x) = 0.5$ 가 되는 지역과 대응하게 되는 것이다. 또한 결정 경계인 이 직선은 가설이 $y = 1$ 이라는 이 공간과, $y = 0$ 이라는 이 공간을 구분하는 직선이다. 한번 더 말하자면, 결정 경계는 이 가설의 특징인데 여기에는 매개 변수 $\theta_0, \theta_1, \theta_2$ 도 포함된다. 학습 세트에 그린 수치는, 시각화를 돋기 위해 데이터 세트를 그려봤는데, 이렇게 데이터 세트를 지운다고 해도 결정 경계와 $y = 1, y = 0$ 이라고 예측할 수 있는 공간은 가설의 특징이자, 가설의 매개변수이지, 데이터 세트의 특징은 아니라는 것이다.

09'22"

다음 번에 이러한 매개 변수를 어떻게 대입하는지에 대해 알아보면서 매개 변수 값을 결정하기 위해서 이 데이터 학습 세트를 활용하도록 하겠다. 하지만 이와 같이 매개 변수 $\theta_0, \theta_1, \theta_2$ 에 특정한 값을 얻게 되면, 이는 결정 경계를 정의하게 되고, 결정 경계를 그래프화 하기 위해서 학습 세트를 대입하지 않아도 된다.

Q) x_1, x_2 두 특징을 가진 로지스틱 회귀를 생각해보자. $\theta_0 = 5, \theta_1 = -1, \theta_2 = 0$ 일 때, $h_{\theta}(x) = g(5 - x_1)$ 이 성립한다. $h_{\theta}(x)$ 의 결정 경계를 바르게 나타낸 그래프는?

정답 1번

이번에는 조금 더 복잡한 예시를 살펴보도록 하자. 여기 x 로 positive 예시를, ○로 negative 예시를 표시해두었다. 이러한 학습 세트가 주어졌을 때, 이러한 데이터에 대입할 로지스틱 회귀를 어떻게 구할 수 있을까?

10'05"

이전 강의에서 다항 회귀에 대해 이야기할 때, 아니면 선형 회귀에 대해서 언급했을 때, 특징에 추가적으로 고차다항식을 더할 수 있는지에 대해 이야기했었다. 로지스틱 회귀에서 이를 똑같이 할 수 있다. 예를 들어, 가설이 이와 같다고 하자. x_1^2, x_2^2 두 개의 항을 더 추가했다. 이제 매개 변수가 θ_0 부터 θ_4 까지 다섯 개가 되었다. 우리가 매개변수 θ_0 부터 θ_4 까지 값을 어떻게 자동으로 선택하느냐에 대해 다음 강의에서 배우기로 했었다. 그러한 다양한 절차가 특정되어 있다면, $\theta_0 = -1, \theta_1 = 0, \theta_2 = 0, \theta_3 = 1, \theta_4 = 1$ 으로 선택해보겠다.

10'59"

이 특정 매개 변수를 선택하는 것의 의미는 매개변수 벡터 $\theta = [-1; 0; 0; 1; 1]$ 이와 같다는 것이다.

아까 살펴본 바와 같이, 가설은 ' $y=1'$, $-1 + x_1^2 + x_2^2 \geq 0$ 이라고 예측할 것이다. 여기 θ^T 곱하기 x 값이 0보다 크거나 같다는 것이다. 여기 -1을 오른쪽으로 가져가면, 가설은 ' $y=1'$, $x_1^2 + x_2^2 \geq 1$ 이 된다. 그렇다면 이 결정 경계는 어떻게 생겼을까? $x_1^2 + x_2^2 = 1$ 곡선 함수를 그려보면, 이는 원점을 중심으로 반지름이 1인 원의 둘레는 구하는 방정식이다. 이것이 바로 여기서 결정 경계가 되는 것이다.

12'10"

원 밖은 모두 $y=1$ 이라고 예측된다. 여기도 $y=1$, 여기도 $y=1$. 그리고 원 안의 공간은 $y=0$ 으로 예측된다. 이렇게 더 복잡한 고차다항식을 특징에 추가함으로써, 더 복잡한 결정 경계를 얻을 수 있는데, positive와 negative 예시를 단순히 직선으로 가르지 않는다는 것을 알 수 있다. 여기 예시에서 결정 경계는 원이었다.

12'44"

다시 말하지만 결정 경계는 매개 변수 가설의 특성이지 학습 세트의 특성이 아니다. 매개 변수 벡터 세타가 주어진다면, 결정 경계의 정의는 원이 될 것이다. 하지만 학습 세트는 결정 경계를 정의하기 위해 활용하는 것이 아니다. 학습 세트는 매개 변수 세타에 대입하기 위해 필요한 것이다. 이와 관해서는 다음 번에 설명하도록 하겠다. 매개 변수 세타가 있을 때 이것이 바로 결정 경계를 정의하는 것이라는 것 명심하도록 하자.

13'14"

학습 세트는 시각화를 할 때 다시 한번 이야기 해보자. 이번에는 아까보다 좀 더 복잡한 예시를 살펴보자. 그렇다면 우리가 이것보다 더 복잡한 결정 경계를 만들어낼 수 있을까? $x_1^2, x_1^2 x_2, x_1^2$ 같은 것보다 더 고차다항식이 주어진다면 아까보다 더 복잡한 결정 경계를 얻을 수 있고, 회귀도 이러한 결정 경계를 찾는데 활용될 것이다. 예를 들어, 이런 모양을 가질 수도 있고, 매개 변수가 조금 달라지게 되면, 조금 이상한 이러한 모양의 결정 경계를 얻게 될 수도 있다.

14'03"

또는 훨씬 더 복잡한 예시를 통해 더 복잡한 모양의 결정 경계를 얻게 될 수도 있다. 이 안은

모두 $y = 1$ 이고, 이 바깥은 모두 $y = 0$ 이 된다. 따라서 이러한 고차다항식을 통해 복잡한 결정 경계를 얻을 수 있게 된다. 이러한 시각화를 통해, 가설 함수의 범위에 대해 이해할 수 있게 되었고, 우리가 로지스틱 회귀에 필요한 표현을 이를 통해 할 수 있다는 것을 배웠기를 바란다. 이제 $h_{\theta}(x)$ 가 표현하는 것이 무엇인지 알아봤으니, 다음 강의에서는 어떻게 주어진 학습 세트에서 매개 변수 세타를 자동으로 선택하여, 이 매개 변수를 데이터에 자동으로 대입하는지에 대해 알아보도록 하겠다.

No.	Title
Week 3-4	Cost Function

00'00"

본 강의에서 로지스틱 회귀를 위한 θ 변수를 어떻게 맞추는가에 대해서 논할 것이다.

Q) 비용 함수 $J(\theta)$ 를 최소화한다고 하자. 다음 중 어떤 함수가 convex인가? 정답: 2번

특별히, 최적화 목표나 변수에 들어맞는 비용함수를 정의하고자 한다. 여기 supervised learning의 로지스틱 회귀 모델을 맞추는 문제가 있다. 우리는 m 트레이닝 셋이 있고, 이 예시들은 $n+1$ 차원이다. 그리고 보통처럼, $x_0 = 1$ 이다. 첫 번째 값은 항상 1이다. 그리고 계산적인 문제 때문에 y 값을 항상 0 아니면 1로 둔다. 이것이 가정이고 매개변수는 여기에 있는 θ 이다. 문제는 이런 트레이닝 셋이 주어졌을 때 어떻게 매개변수 θ 를 고르거나 맞추느냐이다. 예전에 우리가 선형 회귀 함수 모델을 만들 때, 우리는 다음과 같은 비용함수를 따랐다. 이게 $1/2m$ 이었던 것에서 조금의 차이는 있다. 지금은 $1/2$ 을 여기 수식에 대입했다.

01'16"

이제 나는 또 다른 방법으로 이 비용함수를 적어보겠다. 이 제곱식을 적는 것 대신 이런 비용 함수 $h(x)$ 는 이렇게 둔다. $Cost(h_{\theta}^{(x^{(i)})}, y)$ 이다. 이것은 제곱 오차의 $1/2$ 과 같은 수치이다. 이제 비용함수가 이 트레이닝 셋의 합이라는 것을 알 수 있을 것이다. $\frac{1}{m}$ 곱하기 나의 트레이닝 셋의 합이 여기 이 cost항이 된다. 이 등식을 조금 더 간결히 하자면, 이 첨자를 없앤다.

02'03"

그러면 이 비용 함수를 해석해보면, 이 가격은 출력값이 다음과 같을 때 나의 학습 알고리즘이 산출하기를 원하는 가격으로, $h(x)$ 이고 원래는 y 값이라고 할 수 있다. 여기 첨자를 다 생략하고, 선형 회귀 함수가 오차 제곱, 즉 우리가 계산한 것과 실제 값의 제곱의 $1/2$ 이라고 하면 y 의 값은 0이다. 이 비용 함수는 선형 회귀에 적합하다. 하지만 우리는 로지스틱 회귀에 대해 논할 것이다. 우리가 J 에 적용된 비용 함수를 최소화할 수 있다면 잘 구동될 것이다. 하지만 우리가 이 특정 비용 함수를 사용한다면 이것은 매개변수 데이터의 비볼록 함수가 될 것이다. 비볼록 함수라 함은 이것이다. 여기 로지스틱 회귀 $j(\theta)$ 함수와 비선형 함수인 h 함수가 있다. h 함수는 $1/(1+e^{-\theta^T})$ 의 제곱이다. 이 함수를 여기 삽입하고 이 비용함수를 저 곳에 삽입하고, 이제 $j(\theta)$ 가 어떤 모양인지 살펴보자. $j(\theta)$ 함수는 다음과 같은 모양일 수 있다. 이 함수는 국소 최적값을 많이 갖고 있다. 이것은 공식적인 용어로 비선형함수라고 불린다.

03'40"

여러분이 gradient descent를 이런 함수에 사용하면 반드시 전역 최소값에 수렴한다고 말할 수는 없다. 하지만 반대로 $j(\theta)$ 의 볼록한 함수의 경우, 이 궁형 함수의 경우에는, gradient descent를 실행하면, 전역 최소값으로 반드시 수렴한다고 말할 수 있다. 이런 비용 함수를 사용하는 데 있어서의 문제는 이 비선형 변수 값 때문에 $j(\theta)$ 함수가 이렇게 볼록하지 않은 함수로 나타날 수 있다는 것이다. 즉, 이 비선형 변수값을 여기 비용 제곱 함수로 계산하면 말이다. 그래서 다른 볼록형의 비용 함수를 생각하는 것보다 gradient descent 같은 알고리즘을 사용하면 전역 최소값에 수렴할 수 있다.

04'47"

여기 로지스틱 회귀에 사용할 비용함수가 있다. 이제 알고리즘이 내야 하는 비용에 대해서 논하자면, 그러니까 여기 슬라이드 위쪽의 $h_{\theta}^{(x)}$ 의 값이 0.7정도이면 그것은 예측 가능하다. 그리고 실제 가격 label은 y 이다. 가격은 $y = 1$ 이면 $-\log(h(x))$ 와 $y = 0$ 면 $-\log(1 - h(x))$ 가 된다. 이는 상당히 복잡한 함수처럼 보인다. 하지만 무슨 일이 일어나고 있는지 알기 위해 이 함수를 써보자. 먼저 $y=1$ 인 경우부터 시작하자. $y=1$ 일 경우 비용 함수는 $-\log(h(x))$ 이다. 이를 써보면, 즉, 가로축이 $h_{\theta}^{(x)}$ 라면, 우리는 여기 가정의 결과 값이 0~1사이이라는 것을 알고 있다. 즉, 이렇게 0~1사이가 되는 것이다. 이 비용함수가 어떤 모양인지 그러면 이런 모양이 될 것이다. 이렇게 그려지는 이유는 로그 z 에서, z 를 이렇게 가로 축으로 표기하니까, 이렇게 보이는 것이다. 이것은 마이너스 무한대로 다가간다. 로그 함수는 이런 모양이다. 이 점이 0이고, 이 점이 1이다. 여기 z 는 당연히 $h_{\theta}^{(x)}$ 의 역할을 한다. 따라서 마이너스 로그 z 는 이렇게 보일 것이다. 이렇게 표시하도록 하겠다. 그리고 0-1사이의 함수의 값에만 관심이 있으니 이 부분은 지워버리도록 하자. 우리는 이렇게 곡선의 일부분만 보기로 하겠다. 이 부분이 왼쪽의 함수 모양이다.

06' 23"

이제 이 비용 함수는 몇 가지 재미있는 특징을 가진다. 여러분이 보는 것처럼 y 가 1이고 $h_{\theta}^{(x)}$ 가 1일때, 다시 말해 가정이 이렇게 예상한 것과 같을 때 비용은 0이다. 이는 여기 곡선에 대응되는 데, 이 곡선은 완전히 평평해 지지는 않는다. 곡선은 여전히 뻗어나간다. 만약 $h_{\theta}^{(x)}$ 가 1이고, 이 가정이 $y=1$ 이라 했을 때 정말 $y=1$ 이라면 비용은 0이 된다. 이는 여기 이 곡선의 이 점에 대응된다. 하지만 $h_{\theta}^{(x)}$ 가 1이고, 여기처럼 $y=1$ 인 경우만 생각할 때, $h_{\theta}^{(x)}$ 가 1이라면 여기 이 비용 값은 0이 된다. 그리고 이 점은 우리가 원하던 점이다. y 값을 제대로 두면 예측한 출력값 0을 얻을 수 있기 때문이다.

07'20"

하지만 $h_{\theta}^{(x)}$ 가 0에 다가간다는 사실에 주목하자. 따라서 가정의 출력 값도 0에 가까워져서 이렇게 비용은 무한대가 된다. 이것이 주는 의미는 가정이 0이라면, 즉 $h_{\theta}^{(x)}=0$ 이라면, $y=1$ 일 확률이 0과

같다 것이다. 이는 우리가 환자에게 여러분이 악성 종양을 가지고 있을 가능성이 $y=1$ 이라면, 0과 같다고 하는 것과 같다. 그러니 여러분이 악성 종양을 가질 확률이 0이라는 얘기이다. 이렇게 말하고 난 뒤에 환자의 종양이 정말 악성이어서 $y=1$ 이 된다고 해도 종양이 악성일 확률이 0이 된다. 하지만 이 정도의 확신을 가지고 말했는데도 틀렸다면, 우리는 굉장히 큰 비용을 지불하게 될 것이다. $y=1$ 일때 $h_{\theta}^{(x)}$ 가 0으로 수렴한다면 정말 무한대가 되는 것이다. 이 슬라이드는 $y=1$ 인 경우를 정말 잘 고려하고 있다. $y=0$ 일때의 비용 함수에 대해 살펴보자. $y=0$ 이면 이렇게 보일 것이고, 이걸 기입하자면 $-\log(1-z)$ 이 된다. 여러분의 비용 함수가 이렇다는 것이다. 0에서 1의 값을 가지고, $y=0$ 일때의 비용함수는 이런 모양이 된다.

09'00"

이 곡선은 $h(x)$ 가 1로 가기 때문에 이것은 계속 올라서 무한대로 갈 것이다. 앞서 말했던 것처럼 y 는 0이 되는 것이다. 하지만 우리는 $y=1$ 이라고 확신했기에, 이것이 우리의 비용함수가 되는 것이다. 여러분이 이것을 보고 $-\log(1-z)$ 을 그리고, 여러분이 이것이 어떻게 생겼는지 확신했을 때, 0에서 가로축의 z 로 가고 있다는 것을 알게 된다. 이 비용함수를 $y=0$ 인 경우로 생각하라. 이것을 보면 비용함수는 $h(x)$ 가 1로 가기 때문에 양의 무한대로 가고 있다. 그리고 이것은 $h(x)=1$ 이라고 확신했기 때문에 $y=1$ 이 되는 것이다. 그리고 $y=0$ 이라고 답이 나왔으니 이것은 올바른 것이다. 이것은 굉장히 큰 비용이다. 하지만 반대로 $h(x)=0$ 이면 이 비용함수는 0이 될 것이다.

Q) 로지스틱 회귀에서 우리는 $h_{\theta}^{(x)}$ 의 출력값을 예측할 때, 비용 함수를 다음과 같이 두었다. 다음 중 참인 것은? 정답: 1,2,4번

10'41"

이 영상에서 우리는 비용함수를 다루었다. 볼록성(Convexity) 분석은 우리가 다루는 범위를 넘어선 것이지만, 특정 비용함수의 선택에서는 이를 다룰 것이다. Convex 최적화 문제 때문이다. 이는 전체적 비용함수 $j(\theta)$ 가 볼록하고 국소 최적값이 없는 상태를 뜻한다. 다음 강의에서 우리는 비용함수에 대해서 좀 더 깊이 다룰 것이고, 모든 트레이닝 셋을 사용해 비용함수를 정의할 것이다. 또한, 지금까지 사용했던 것 보다 좀 더 간결한 방식을 사용할 것이고, 우리가 grading descent를 바탕으로 로지스틱 회귀 알고리즘을 다룰 것이다.

No.	Title
Week 3-5	Simplified Cost Function and Gradient Descent

00'00"

본 영상에서는 여태껏 해 왔던 방법보다 비용함수를 좀 더 간결하게 적는 법을 배울 것이다. 더불어 gradient descent를 어떻게 로지스틱 회귀의 변수에 맞출지 배울 것이다. 이 영상이 끝날 때쯤이면 여러분은 로지스틱 회귀를 잘 다룰 수 있을 것이다. 이것은 로지스틱 회귀를 위한 비용함수이다. 우리의 비용함수는 다른 y_i 의 예를 사용한 차이를 측정한 모든 값을 더한 값의 $1/m$ 이다. 그리고 우리는 앞서 한 예의 비용에 대해 배웠다. 우리 트레이닝 셋의 분류 예를 다시 상기시키자면 트레이닝 셋 y 는 항상 0이거나 1이다. 이것은 y 에 대한 수학적 정의이다. y 가 0이나 1이기 때문에 우리는 비용함수를 좀 더 수월하게 적을 수 있다. 그리고 특히나 다른 두 가지의 케이스를 또 다르게 두 번 적는 것보다, $y=0$ 이거나 1이기 때문에, 하나의 등식으로 합쳐서 적도록 하겠다. 이것은 비용함수를 적기 좀 더 간편하고 gradient descent를 다루기에도 편하다. 좀 더 명확히 말하자면 우리는 이런 식으로 비용함수를 적어보자. $h(x)$, y . 그리고 $-y$ 곱하기 $h(x) - (1-y)$ 곱하기 $\log(1-h(x))$ 라고 하자. 그리고 두 번째 등식은 이렇다. 이것은 여기 위에 적어 둔 비용함수의 정의를 쉽게 하는 방법이다. 한번 살펴보자.

02'04"

우리는 두 가지 가능한 방법이 있다는 것을 알고 있다. y 는 0혹은 1이다. $y=1$ 이라고 생각하면, 이 등식이 의미하는 것은 비용은, 그러니 이것 역시 1이 되는 것이다. 그리고 $1-y$ 가 0이 되어 간다. 그러니 $y=1$ 이면 $1-y$ 는 $1-1$ 이니까 0이 된다. 두 번째 것은 0을 곱하는 것이니 없어진다. 우리는 첫 번째 것만 보면, $y=1$ 이니까 이것은 $-\log h(x)$ 가 된다. 그리고 이 등식은 우리가 $y=1$ 일 때 썼던 것과 일치한다. 다른 경우는 $y=0$ 이면 우리의 비용함수는 이것이 0이 된다는 의미인데, $1-y$ 의 경우에는 y 가 0이니까 1이 된다는 것이다. 이렇게 비용함수가 간결해진다. 처음 것이 0과 곱해지니까 사라지고, 두 번째 것만 남는데 이것은 $-\log(1-h(x))$ 이다. 이것이 y 가 0, 1일 때 두 가지의 경우만 확인하면 된다. $y=0, 1$ 두 가지 경우에 모두 간결해지는 방법이다. 그러니 이렇게 1개의 선으로 완성될 수 있는 것이다. 우리는 이렇게 로지스틱 회귀를 위한 비용함수를 그릴 수 있게 된다. 이것은 이 비용함수의 합의 $1/m$ 한 값이다. 이것을 정의하자면, 이런 식으로 되는 것이다. 마이너스 표시를 밖으로 꼬집어 내자. 고를 수 있었던 다른 비용함수처럼 보이는 특별한 함수를 고른 이유는 무엇일까.

04'18"

비록 이를 자세하게 설명할 수는 없지만 이 비용함수는 최대공산추정원리를 사용한 통계로 얻어낼 수 있다. 이것은 다른 모델의 변수 데이터를 어떻게 효율적으로 찾아내는가 하는

문제이다. 이것은 볼록성이라는 특성도 가지고 있다. 그러니 이 비용함수는 로지스틱 회귀 모델과 맞게 하려는 모든 사람들이 쓰는 것이라고 할 수 있다. 이것이 무엇을 의미하는지 모르더라도 걱정할 필요가 없다. 이것은 비용함수에 대한 배경이라 할 수 있다. 이런 비용 함수가 주어진다면, 우리는 j 세터 함수를 최소화하는 변수 θ 를 찾아내야 한다. 우리가 이것을 최소화하려 한다면 우리는 θ 변수들의 집합을 구할 수 있을 것이다. 그리고 특성 x 세트를 보면 우리의 예상의 결과 값이나 트레이닝 셋에 알맞은 θ 를 찾아낼 수 있다. 내 가정의 결과 값은 $y=1$ 일 경우이다. x 라는 입력 값을 주고, 이것을 θ 매개변수로 나타내자. 하지만 여러분은 $y=1$ 이라는 가정이라는 것을 알아야 한다. 그러니 $J(\theta)$ 함수를 최소화하는 방법을 알아낸 후에야 우리 트레이닝 셋의 변수와 맞출 수 있는 것이다. 이것을 θ 함수로 최소화하려 하고, 이것은 우리가 사용했던 gradient descent로 학습 속도를 이 미분계수에 알파 번 곱했다. 여러분이 미적분을 안다면 미적분으로 간결화하더라도 같은 값을 얻을 수 있을 것이다. 하지만 모르더라도 걱정하지 않아도 된다.

06'30"

이것을 실제로 계산하면, 여러분은 이 등식을 얻게 된다. 써보자. 이것은 i 가 0부터 m 일 경우에 x_{ij} 를 곱한 값의 합이다. 여러분이 이 편미분계수를 빼서 여기 붙여 놓는다면 우리는 gradient descent 알고리즘을 이렇게 적을 수 있다.

07'49"

이는 로지스틱 회귀를 분석함으로써 풀 수 있다. 이 정의가 바뀐 것을 볼 수 있다. 선형회귀와는 다르게 $h(x)$ 는 $\theta^T x$ 였는데 이제는 이 $h(x)$ 의 정의가 바뀐 것이다. 지금은 $1/(1+e^{-\theta^T x})$ 이다. 꾸며 놓은 것은 같아 보이는데 그것은 가정의 정의가 바뀌었기 때문이다. 이것은 선형회귀를 위한 gradient descent와는 다른 내용이다. 이 때는 이것이 어떻게 수렴하는지 만을 보았다. 이것을 로지스틱 회귀에도 같이 적용하곤 한다. 그리고 여러분이 이 기술을 어떻게 로지스틱 회귀에 활용하는지 알아냈기를 바란다. 우리는 gradient descent 로지스틱 회귀 시행을 위해 이 기술을 활용하기 위해서, 완전히 다른 변수 값을 가지고 있는데, 이것은 θ 0에서 n 까지이며, 이것을 사용해서 업데이트 할 것이다.

Q) 여러분이 로지스틱 회귀 모델에 맞도록 gradient descent 알고리즘을 $\theta \in R^{n+1}$ 매개 변수와 함께 실행한다고 하자. 다음 중 학습 속도 α 를 어떻게 설정하면 gradient descent가 제대로 작동하는지를 확실히 할 수 있을까? 정답: 2번

08'56"

우리는 루프를 만들 것이다. 그러니 i 가 0부터 n 일 때나 1부터 $n+1$ 사이일 때, 차례로 이 변수 값을 업데이트 하는 것이다. 하지만 물론 우리는 루프를 사용하는 것 대신에 vector rise를 시행하면 한번에 이 $m+1$ 변수들을 업데이트 할 수 있다. 이 알고리즘의 vector rise 시행을 하면

여러분은 로지스틱 회귀를 위한 gradient descent를 할 수 있는 것이다. 우리가 앞서 말한 것으로 선형 회귀는 변수 값 조정이었는데, 이것이 gradient descent가 빠른 선형 회귀를 어떻게 만들 수 있는지 살펴보았다. 이러한 변수 값 조정은 gradient descent의 로지스틱 회귀 속도를 높여줄 것이다.

Q) gradient descent의 한 반복이 다음과 같은 업데이트를 동시에 수행한다. 우리는 $\theta := \theta - \alpha \cdot \delta$ 를 실행하려고 한다. 다음 중 벡터화된 실행은 무엇인가? 정답: 2번

이제 여러분은 로지스틱 회귀를 시행하는 방법을 알고, 이것이 아주 널리 사용되는 분류 알고리즘이라는 것을 안다. 이제 여러분은 이것을 혼자서도 잘 실행할 수 있을 것이다.

No.	Title
Week 3-6	Advanced Optimization

00'00"

지난 영상에서 우리는 로지스틱 회귀의 비용함수 $J(\theta)$ 의 최소화를 위한 gradient descent에 대해 배웠다. 이번에는 고급 최적화 알고리즘과 그 개념에 대해 배울 것이다. 이런 아이디어를 활용해서 gradient descent보다 더 빠르게 로지스틱 회귀를 얻을 수 있을 것이다. 이는 특징이 더 많은 등의 더 방대한 머신 학습 문제에 적용될 것이다. 여기는 gradient descent를 대신해서 할 수 있는 관점이다. 여기 비용함수 J 가 있고 우리는 이것을 최소화하고자 한다.

00'39"

우리가 해야 하는 것은 우리는 θ 변수들의 입력 값을 위한 코드를 써야 하고, 이것은 2가지를 계산할 수 있다. $J(\theta)$ 와 편미분계수인데, 여러분이 아는 것처럼 $J=0, 1$ 에서 n 까지이다. 이런 두 가지를 할 수 있는 코드가 주어지면 gradient descent는 이런 업데이트를 계속해서 시행한다. 그러니 이들을 계산하기 위해 쓴 코드가 주어지면 gradient descent는 여기 쓰여지고 우리 θ 변수를 업데이트한다. 이것을 다르게 생각하는 방법은 우리가 $J(\theta)$ 와 이 미분계수를 계산하기 위해 코드를 제공해야 하고, 그 후 gradient descent에 입력하면 이 함수를 최소화한다는 것이다. gradient descent는, 사실 여러분이 비용함수 $J(\theta)$ 를 계산하기 위해 코드를 입력해야 할 필요는 없다. 여러분이 이것을 하는 이유는 미분계수를 계산하기 위해서이다. 하지만 코드가 무엇이 수렴하는 것을 감시한다면, 다시 한번 코드를 제공하는 것을 생각해 볼 수 있을 것이다. 이런 두 가지를 계산하기 위해 코드를 쓴다면, 하나는 gradient descent일 것이다. 하지만 이것이 우리가 사용하는 유일한 알고리즘은 아니다. 더 진보된, 더 복잡한 알고리즘도 존재한다. 하지만 우리가 이 두 가지를 계산하기 위해 한 가지 방법만을 제공한다면, 이것들은 최적화를 위한 다른 방법들일 것이다. 예전 기울기법 BFGS와 L-BFGS는 $J(\theta)$ 함수와 미분계수 계산의 최적화를 위한

다른 더 복잡한 알고리즘이다. 그리고 비용함수 최소화를 위해 gradient descent보다 더 복잡한 방식들을 사용할 수 있다.

03'40"

이 알고리즘은 이 강의에서 다루는 범위를 벗어난다. 여러분은 사실 이런 알고리즘에 많은 시간을 쓰곤 한다. 이것은 여러분이 고급 수학을 배울 때의 얘기이다. 하지만 몇 가지 특징에 대해서 말해보자. 이것들은 많은 이점을 갖고 있는데, 첫 번째로는 이 대부분이 학습 속도 알파를 수동으로 골라야 할 필요가 없다는 것이다. 그리고 이 알고리즘은 미분계수와 비용함수를 계산하기 위한 것이다. 이것이 반복을 갖는 알고리즘이라고 생각할 수 있다. 그리고 실제로 라인 서치 알고리즘이라고 불리는 똑똑한 반복 알고리즘이다. 이는 학습속도 알파를 자동으로 찾아주고 심지어 매 반복마다 찾아주기도 한다. 여러분의 작업을 대신 수행해준다. 이 알고리즘들은 적합한 학습 속도 찾는 이상의 일을 하고 gradient descent보다 더 빠르게 수렴해준다. 그렇지만 좀 더 자세한 내용은 이 강의의 범위에서 벗어난다. 사실 이런 알고리즘을 굉장히 오랜 시간, 10년 이상 꽤 자주 사용되어 왔다. 하지만 아주 최근에서야 이 컬레기울기법이 무엇인지에 대한 자세한 사항을 알 수 있었다. 이 알고리즘을 정확히 사용하고 이해하지 못해도 이를 문제 해결에 적용하는 것이 가능하다. 이 알고리즘에 부족한 점이 있다면, 가장 부족한 점은 gradient descent보다 많이 복잡하다는 것이다. 특히나 다음을 잘 시행하지 못할 것이다. 잘 시행하기 위해서는 여러분이 수에 있어 아주 뛰어나야 한다. 제곱근을 계산을 여러분이 직접 한다거나 역이나 행렬 계산을 직접 하지 말라고 하는 것처럼 이를 직접 사용하기보다는 소프트웨어 라이브러리를 쓰는 것을 권한다. 알다시피 제곱근을 구하기 위해 함수를 사용하는데 이것은 누군가가 이미 써 두었다. 그리고 다행히도 우리가 사용하게 될 옥타브나 MATLAB이 있다. 옥타브는 실제로 매우 훌륭하고, 매우 훌륭한 고급 최적화 알고리즘을 시행하는 라이브러리를 보유하고 있다. 그러니 여러분이 빌트인 라이브러리를 활용한다면 여러분은 좋은 결과를 얻을 것이다. 그리고 이 알고리즘의 좋거나 좋지 않는 시행에 있어서 분명 차이점은 있을 것이다.

03'23"

그리고 여러분이 머신 학습 응용프로그램에 다른 언어를 사용하고 있다면, 여러분이 C, C++, 자바를 이용하고 있다면 다른 라이브러리를 사용해서 여러분이 이 알고리즘 실행을 위한 좋은 라이브러리를 발견했다고 확신하고자 할 것이다. 왜냐하면 분명 LPFGS, 즉 contour gradient의 좋은 시행과 나쁜 시행에는 차이점이 있기 때문이다. 그러니 어떻게 이 알고리즘을 사용하는지 설명할 것이다. 몇 가지 예로 설명할 것인데, 우리가 2개의 변수 θ_0 과 θ_1 있는 문제가 있다고 하자. 그리고 비용함수 $J(\theta)$ 는 θ_1 의 -5제곱 더하기 θ_2 -5제곱이라고 하자. 이것이 비용함수이다. 여러분은 θ_1 , θ_2 의 값을 알고 있다. 여러분이 이 함수를 최소화하고자 한다면 이 가치는 θ_1 은 5이고 θ_2 가 5가 된다. 다시 여러분 중 누군가는 미적분학을 잘 알고 있을 것이지만,

이 미분계수는 이렇게 두 가지로 표현된다. 나는 미적분을 이용했다.

06'20"

여러분이 비용함수 J 를 최소화하기 위해 고급 최적화 알고리즘을 사용하고자 한다면, 그리고 우리는 최소값이 5인 것을 몰랐지만, 5가 최소인 gradient descent는 아니지만 같은 것을 사용하고자 한다면, 여러분은 이와 같은 옥타브 함수를 사용해야 한다, 그러면 비용 함수 델타는 이렇게 될 것이고, 2가지 논점을 가져온다. 첫 번째는 어떻게 비용함수 J 를 계산하느냐이다. J -val이 이 식과 같다고 하고, 이것은 여기 있는 비용함수를 그냥 계산한 것이다. 두 번째 논점은 이 함수의 return이 gradient라는 것이다. 그러니 (2,1) 벡터가 될 것이고, 이 gradient vector는 여기 두 가지의 편미분계수항과 일치할 것이다. 이 비용함수를 시행함에 있어 여러분은 fminunc이라고 불리는 고급 최적화 함수를 사용할 수 있다. 이것은 옥타브에서 자유로운 최소화이다. 여기 옵션을 정할 것이다. 이것은 여러분이 넣고자 하는 옵션을 저장하는 데이터 구조이다. 여기에 이것을 적용시켜보자. 이것은 여러분이 이 알고리즘에 gradient를 제공한다는 의미이다. 이 반복에 100이라는 최대치를 지정할 것이다. 그리고 0에 대한 추측으로 (2,1)을 사용할 것이다 그리고 이것을 fminunc라고 부른다. 이것은 우리가 위에 적어두었던 비용함수를 의미한다. 그리고 우리는 또 다른 진보된 최적화 알고리즘을 사용할 것이다.

08'20"

여러분이 이것을 gradient descent처럼 여기고자 하지만 이것이 알파를 스스로 찾기 때문에 여러분이 할 필요는 없다. 그러나 이것은 더 고급 최적화 알고리즘을 사용할 것이다. gradient descent가 진보된 버전이라고 보면 되겠다. 여러분에게 최적값을 찾아주기 위한 방법을 보자. 옥타브에서 어떻게 보이는지 살펴보면 지난 번 직선에 그렸던 것과 일치한다. 이는 J -val, 즉 비용함수를 계산한다. 또한 편미분계수와 이 비용함수도 계산한다. 이 두 변수 θ_1 과 θ_2 의 비용함수 얘기이다. 옥타브 윈도우로 바꿔보자. 내가 했던 명령어들을 입력할 것이다. 그리고 옵션은 최적화 셋(optim set)과 동일하다. 이것이 내 옵션의 변수를 세팅하는 기법이다. 100개의 반복이 맥시멈이라고 두었기 때문에 이것 또한 내 알고리즘에 적용할 것이다. 그리고 처음의 θ 는 Zeros(2,1)이다. 이것이 처음의 예상이었다. 그리고 이와 같이 기입할 것이다. 이것이 비용함수이고 이것은 나의 첫 번째 예상이다. 이것은 옵션이다. 엔터를 누르면 이 알고리즘이 시행될 것이다. 결과물은 빨리 돌아온다. 여기에 코드가 보이고, 내 명령어 라인은 이렇게 되어 있다. 이것이 의미하는 것은 여러분이 알다시피 gradient descent의 발전한 버전이고, θ 의 최적값은 $\theta_1=0.5$, $\theta_2=-0.5$ 일때, 즉 우리가 원하는 그대로이다. 그러니 최적값은 10에서 -30까지이다. 이것은 0이고, 이것이 우리가 원하던 것이다.

10'25"

그리고 이것은 이것의 통합 상태가 무엇인가를 보여준다. 여러분이 fminunc이 exit flag를 어떻게 해석할지에 대한 문서를 읽기를 돋길 원할 때, 하지만 xexit flag가 통합이 되는지 안 되는지를 알아보게 해 준다. 그러니 어떻게 이것을 옥타브에서 사용하는지 알아보자. 옥타브에서의 실행은, 이 θ 의 값이 d의 rd에서 2보다 크거나 같아야 한다. θ 는 실수이다. 이것이 이차 벡터나 그 이상이라면 fminunc는 잘 시행되지 않을 것이고, 일차함수를 가진 경우에 여러분은 더 세부사항을 얻기 위한 옥타브 문서를 들여다 볼 수 있을 것이다. 하지만 이것을 알아본 것은 큰 발전이라고 생각하자. 로지스틱 회귀에서 우리는 벡터 세터 변수를 갖고 있고, 옥타브 표기와 수학 표기를 섞어 설명할 것이다. 하지만 이 설명이 명확했기를 바라고, 우리 변수 벡터 세터는 이 변수 θ_0 에서 n 까지를 구성하고 있다. 왜냐하면 옥타브 인덱스는 벡터가 지수1에서 0까지이기 때문이다. 우리가 해야 하는 것은 로지스틱 회귀를 위한 비용함수를 찾아내는 것이다. 정확히 말하면 비용함수는 J-val의 return, 즉 j세터나 gradient를 계산하기 위한 코드가 필요하다. 그래서 gradient1은 θ_0 을 고려한 partial 미분계수 계산, 그리고 θ_1 , 그 이상을 위한 계산을 위해 필요한 코드일 것이다.

12'35"

다시 말하자면, 이것은 gradient 1,2 그리고 등등이다. 0부터 시작하지 않는 이유는 옥타브 인덱스는 변수가 0이 아니라 1부터 시작하기 때문이다. 하지만 이 슬라이드에서 가르치고자 하는 것은 비용함수가 돌아오고 gradient가 돌아오는 함수를 쓰는 것이 필요하다는 것이다. 이것을 로지스틱 회귀나 선형회귀에 적용하고자 할 때, 여러분이 선형 회귀를 위한 이 최적화 알고리즘을 사용하고자 한다면 여러분은 계산하기 위해 적절한 코드를 입력해야 한다. 그러면 여러분은 이제 이것들을 사용할 줄 안다. 왜냐하면, 여러분이 이런 복잡한 최적화 라이브러리를 사용하기 때문에 디버그하기 힘들고, 불투명하다.

Q) 여러분이 매개변수 θ_0, θ_1 을 가지고 로지스틱 회귀를 위한 비용 함수를 최소화하기 위해 진보된 최적화 알고리즘을 사용하고 싶다고 하자. 여러분이 사용하는 코드는 아래와 같다.
CODE#1과 CODE#2가 무엇을 계산해야 하는가? 정답: 3번

이 알고리즘은 gradient descent보다 빠르게 실행되고, 특히나 더 방대한 머신 학습 문제일 때, 이것을 사용할 것이다. 그리고 이것이 논리적으로 이해되었기를 바라며, 선형 회귀가 더 많은 문제에 적용되기를 바란다. 이것이 고급 최적화의 개념이다. 그리고 다음 강의에서 여러분에게 로지스틱 회귀 알고리즘에 대한 내용과 다계층의 분류 문제를 다룰 것이다.

No.	Title
-----	-------

00'00"

본 강의에서는 멀티클래스 분류 문제에 적용 가능한 로지스틱 회귀를 배울 것이다. 특별히 one-versus-all 분류라고 불리는 알고리즘에 대해서 얘기할 것이다. 다계층 분류 문제란 무엇인가? 예를 들어, 여러분이 이메일을 다른 폴더로 분류하거나 태그 하는 학습 알고리즘을 원한다고 하자. 그렇다면 여러분의 이메일은 다른 폴더에 분류되어 있거나 업무메일, 친구에게서 온 메일, 가족, 그리고 취미활동과 관련한 이메일로 자동적으로 태그가 달려 있을 것이다. 여기에 y 가 1, 2, 3, 4일 때로 분류할 4가지 종류를 가진 분류 문제가 있다. 다른 예로는, 의학 진단의 경우인데, 환자가 코가 막혀서 여러분을 찾아온다고 하자. 그리고 그들이 아프지 않은 경우가 있을 수도 있다. 감기인 경우는 $y=1$, 독감의 경우 2로 둔다. 그리고 세 번째 예는 여러분이 머신 학습으로 날씨를 분류할 때, 여러분은 아마도 맑음, 구름 낀 날씨, 비, 혹은 눈, 혹은 눈이 올 것이라고 할 수 있을 것인데, 이러한 예에서 y 는 작은 수 즉 1-3이나 1-4까지 정도의 값을 가질 수 있다. 그리고 이것들이 멀티클래스 분류 문제이다.

01'21"

하지만 지수가 0,1,2,3이든 1,2,3,4이든 그렇게 큰 상관은 없다. 나는 지수를 0보다 1에서 시작하는 것을 선호하지만, 이것이 그렇게 큰 영향을 미치지는 않는다. 여태껏 분류가 2진법이었다면 우리의 데이터셋은 이러하다. 멀티클래스 분류 문제는 이렇게 3가지 기호를 사용해서 3가지 종류를 구분한다. 이러한 데이터셋이 주어진다면, 그리고 이것이 한 종류의 예라면, 이것은 다른 종류의 예이고, 이것은 3번째의 예일 것이다. 이 세팅을 위해 어떤 학습 알고리즘을 얻을 수 있을까? 우리는 회귀를 사용해서 2진법의 분류 방법은 배웠다. 우리는 양이나 음의 클래스의 들어맞는 직선을 그릴 줄 알고 있다. 이것이 one-vs-all classification이다. 여러분은 이것을 멀티클래스 분류에 활용할 수 있다. 이렇게 움직인다. 그리고 이는 one-vs-rest라고도 불린다. 왼쪽에 있는 트레이닝셋처럼 우리는 삼각형인 $y=1$, 정사각형인 $y=2$, 그리고 십자인 $y=3$ 일 때, 이렇게 세 가지 종류가 있다. 이 세 가지 분류 문제를 2가지 분류로 바꿔놓으면, 삼각형부터 해보겠다. 그리고 2와 3이 음의 클래스로 배정된 곳에 가짜 트레이닝셋을 만들어낼 것이다. 그리고 클래스 1은 양의 클래스로 배정되었다. 여러분은 오른쪽에 보이는 것과 같은 트레이닝셋을 만들어내고자 한다. 그리고 $h_{\theta}x$ 라고 불리는 classifier에 맞도록 배정할 것이다. 1의 값으로 배정된 삼각형과 0의 값으로 배정된 원들을 생각해보라. 우리는 standard logistic regression classifier에 대해서 배울 것이고, 이것은 우리에게 이렇게 보이는 경계선을 제공할 것이다. 여기 위첨자 1은 클래스1을 위한 것이고 클래스1 삼각형을 칭하는 것이다. 우리는 클래스 2에 대해서도 똑같은 것을 할 것이다. 이 정사각형을 떼서 이렇게 양의 클래스로 보내고, 이렇게 삼각형이나 십자모양을 음의 클래스로 배정한다. 그러면 우리는 두 번째의 logistic regression classifier에 들어맞고, 이것은 h_2x 라고 불린다. 여기서 위첨자 2는 정사각형 클래스를 양의

클래스로 취급한다는 것을 의미한다. 우리는 이렇게 배정하였다.

04'08"

그리고 마지막으로 클래스 3에 있어서도 h_3x 에 맞게 하기 위해 동일한 것을 수행할 것이다. 이렇게 줄을 그어 양과 음의 예를 나눌 수 있을 것이다. 요약하자면 우리가 한 것은 이렇게 세 가지의 classifier에 들어맞게 만들었다. 그러니 $i=1,2,3$ 의 경우, classifier는 $h_\theta^i(x)$ 일 것이다. 그러니 x 가 θ 변수일 때, y 가 클래스 i 일 경우에 대해서 예측하고자 한다. 그러니 이 위쪽에 있는 첫 번째 것의 예는 classifier는 삼각형을 인지하기 위해 배우고 있다. 삼각형이 양의 조항이라면, x 위첨자 1은 y 가 1일 가능성을 측정하고자 하는 것이다. 유사하게, 이것은 정사각형이 양의 클래스고 $y=2$ 일 가능성을 측정하는 것 등과 같다. 우리는 이제 3개의 classifier가 있고, 각각 다른 분류 중 1개를 인지하기 위해 훈련되었다. 요약하자면 우리가 한 것은 우리는 로지스틱 회귀 classifier h 와 첨자 i x 를 구한 것이고, 이것은 $y=i$ 일 때의 가능성을 예측하기 위함이다. 그리고 예측을 위해서 우리는 새로운 입력 값을 이렇게 두었고, 이렇게 예측을 하려 한다. 우리는 x 에 이 3가지 classifier를 작동시켜 3가지를 최대화하는 클래스 i 를 골랐다. 그러니 classifier를 고른 것이다. 3가지 다 신뢰성이 높고 다 참의 조항이라고 할 수 있다. i 의 값이 무엇이든 우리에게 y 가 특정 값이라는 것을 예측할 수 있는 높은 가능성을 제공한다.

Q) k classes인 멀티클래스 구분 문제를 가지고 있다고 하자. one-vs-all방법을 사용해서 얼마나 많은 로지스틱 회귀 classifier를 갖게 되겠는가? 정답: 2번

지금까지 멀티 클래스 분류와 one-vs-all을 살펴보았다. 또한 로지스틱 회귀 classifier를 알아내어 멀티클래스 분류 문제에 활용할 수 있게 되었다.

No.	Title
-----	-------

00'00"

지금까지 여러분은 다른 학습 알고리즘, 선형 회귀와 로지스틱 회귀를 여럿 살펴보았다. 이것들로 많은 문제들을 해결할 수 있지만, 여러분이 이것을 특정 머신 러닝 프로그램에 적용할 때 과적합(overfitting)이라고 불리는 문제에 봉착할 수 있다. 이 강의에서 설명할 내용은 과적합 문제이고, 다음 몇 강의에서 정규화에 대해서 배울 것이다. 이것은 이 과적합 문제를 개선하거나 줄이고 학습 알고리즘이 더 잘 작동하도록 해결한다. 이 과적합은 무엇인가? 선형회귀의 주택가격 측정 예를 계속해서 보면 우리는 집 크기의 함수로 집 가격을 예측하려고 한다. 우리가 할 것은 이 선형 함수를 데이터에 맞추는 것인데, 이게 맞다면 데이터에 알맞은 직선을 찾을 수 있다. 하지만 이것은 좋은 모델은 아니다. 데이터를 보면 집의 크기가 늘어날수록 안정상태를 유지하고, 그렇게 되면 끝이 완만해져서 이 알고리즘은 트레이닝 셋과 맞지 않아진다. 이것은 과소적합 문제이고, 이 알고리즘은 high-bias가 있다는 것이다. 이 두 가지가 의미하는 것은 트레이닝 데이터와 맞지 않는다는 것이고, 그렇다면 이것은 알고리즘은 예상치가 강하거나 데이터와는 다르게 집 가격은 선형으로 가변 한다는 것을 의미한다.

01'49"

반대로 이 예상이 편향되었다는 증거가 있지만 이것은 직선에 맞아 떨어지고 데이터에 그다지 맞지 않게 된다. 이제 중간을 보면 이렇게 이차함수에 맞춰질 수 있고, 이 데이터 셋이 있으면 이차함수에 맞춰질 수 있으니, 아마도 우리는 이런 곡선을 그리게 될 것이고 이것은 잘 작동할 것이다. 다른 극단적 상황에서 보면, 이렇게 다향식의 데이터가 있다. 우리는 5개의 변수가 있고, θ_0 부터 θ_4 까지이다. 이것으로 우리는 이 다섯 가지의 예를 통과하는 곡선을 만들 수 있다. 여러분은 이런 곡선을 그릴 것이다. 한편으로는 이것이 굉장히 잘 맞아 보인다. 하지만 굉장히 구불구불하다. 이렇게 계속 아래위로 움직이기에 좋은 예상 함수라고 할 수 없는 것이다. 이것을 과적합이라 부르고 이는 많은 변화가 보인다. 다변화라는 말은 전통적이거나 기술적인 용어이다. 하지만 중요한 점은 이러한 다향식에 맞아 떨어진다면 어떤 가정을 사용하고, 그것이 얼마나 큰가에 상관 없이 다 맞아 떨어지는 함수라는 것이다. 우리가 가진 데이터가 한정되지 않는다면 좋은 예측이라고 할 수 있지만, 이것은 과적합이다. 그리고 이것은 정확한 이름은 없지만 그냥 이름 붙인 것이다. 이 다향 이차 함수는 이 데이터에 알맞아 보인다. 과적합 문제를 정의하자면, 우리가 너무 많은 특징을 가지고 있을 때 트레이닝 예에 잘 맞아 떨어질 것이다. 여러분의 비용함수는 아마도 0과 비슷하거나 0일 수도 있다. 하지만 이런 곡선을 갖게 되면 이 트레이닝 셋에 맞추기 위해 너무 노력하기 때문에 일반화하기 어렵고 새로운 것을 예측하기도 어렵다. 여기 일반화라는 말은 새로운 예시에 적용 가능한 정도를 의미한다.

04'00"

이것은 집에 대한 여기 트레이닝 셋에 있지 않은 데이터를 의미한다. 이 슬라이드에서는 우리는 선형 회귀의 과적합 문제에 대해 살펴보았다. 비슷한 예가 로지스틱 회귀에도 활용될 수 있다. 특징 X_1, X_2 가 있는 로지스틱 회귀 예가 있다. 우리가 처음으로 할 것은 이런 간단한 가정을 로지스틱 회귀에 맞추는 것이다. G 는 이런 s 자 모양의 함수이다. 그러면 양과 음을 나누는 직선을 사용하게 될 것이다. 이것은 가정과 맞지 않아 보이기에 이것은 과소적합이나, high bias의 예측이라고 할 수 있다. 반대로 이런 제곱항들을 더한다면 여러분은 이렇게 보이는 결정선을 가질 수 있을 것이다. 이것은 데이터에 잘 들어맞는다. 아마도 이것이 최선일 것이다. 마지막의 예를 보면, 여러분이 이런 고차원의 다항식에 맞추려고 한다면, 로지스틱 회귀는 이렇게 뒤틀릴 것이다. 그리고 결정선을 찾기에 매우 난해할 것이고, 데이터에 맞추기 위해 매우 뒤틀릴 것이다. 여기 특징 x_1, x_2 가 유선 종양이 양성인지 악성인지를 구분하는 것이다. 이것은 예측을 위한 좋은 가정으로 보이지는 않는다. 그리고 이것은 과적합 문제이며, 새로운 예를 일반화하는 것이 쉽지 않아 보인다.

Q) 종양이 악성인지 양성이지 구별하는 의학 진단을 생각해보라. $h_{\theta}^{(x)}$ 가 트레이닝 셋에 과적합된다면 이것이 의미하는 바는 무엇인가? 정답: 3번

이 과정의 뒷부분에서 디버깅과 학습 알고리즘의 진단에 대해서 다룰 텐데, 과적합인지 혹은 과소적합인지의 여부를 알아보기 위한 툴을 제공할 것이다.

06'17"

하지만 지금 과적합이 일어나고 있다면 어떻게 해결할 것인가? 앞선 예에서 보듯이 우리는 1-2 차원의 데이터를 보유하고 있고 이 가정을 이렇게 기입하고 어떻게 되는지, 그리고 어떻게 알맞은 정도의 다항을 가지는지를 볼 것이다. 앞서 집 가격 문제를 보면 우리는 이렇게 가정을 적을 수 있고, 아는 것처럼 이런 식으로 데이터에 구불구불하게 맞춰지는 것을 알 것이다. 우리는 이렇게 숫자를 사용해서 어느 정도의 다항식이 적합한지를 고를 것이다. 하지만 이것이 항상 이뤄지는 것은 아니다. 그리고 여러 특징을 가진 학습 문제들이 종종 발생한다. 그리고 이런 것들은 다항의 정도를 고르는 문제가 아니다. 그리고 사실 우리는 특징을 많이 갖고 있기에 이것을 가시화 하는 것도, 이것을 쓰는 것도 힘들다. 그리고 어떤 특징을 보유할지 없앨지를 결정하는 것도 힘들다. 구체적으로 말해 우리가 집의 가격을 예측하려고 할 때 우리는 엄청 많은 값들을 가지게 될 수도 있다. 그리고 이런 값들은 모두 다 유용하게 보인다. 하지만 값이 너무 많고 데이터는 적다면 과적합은 문제가 된다. 과적합 문제를 해결하기 위해서 우리가 취할 수 있는 두 가지의 옵션이 있다. 첫 번째는 변수의 수를 줄이는 것이다. 다시 말해, 변수의 목록을 보고 어떤 것이 중요한지, 그러니까 어떤 것을 사용하고 어떤 것을 버릴지 결정한다.

07'58"

좀 더 뒷부분에서 모델 선택 알고리즘에 대해 논할 것이다. 이것은 어떤 값을 사용할지 버릴지를 자동적으로 결정하는 알고리즘이다. 변수의 수를 줄이면 과적합 문제를 줄일 수 있다. 그리고 우리가 이 모델 선택에 대해서 논할 때 좀 더 자세히 다룰 것이다. 하지만 이 단점은 어떤 변수를 버리면 문제에 대한 정보도 버리는 셈이 된다. 예를 들어 아마도 모든 값들이 이 집의 가격을 결정하는데 유용하다고 볼 수 있다. 그래서 어떤 정보도 버리고 싶지 않을 수 있다. 두 번째 옵션은 정규화 문제인데 나중 강의에서 배우게 될 것이다. 여기 우리는 이 값들을 다 보유하지만 변수 θ 의 값을 조금 줄일 것이다. 그리고 우리가 특징을 많이 보유하고 있고, y 값의 예측에 다 조금씩 도움이 될 때 이 방법은 잘 활용될 수 있을 것이다. 또한 변수가 많은데도 하나하나가 다 가치 있기에 버리고 싶지 않을 수가 있다. 그래서 아주 높은 수준의 정규화를 진행한다. 이런 세부사항이 이해가 되지 않을 수도 있다. 하지만 다음 강의에서 이것이 무엇인지 어떻게 사용하는지에 대해 배울 것이다. 이것을 사용해서 과적합을 피하고 이것을 잘 활용하는 법을 배우자.

No.	Title
Week 3-9	Cost function

00'00"

이 영상에서 어떻게 정규화가 작동하는지에 대한 것을 배울 것이다, 그리고 여기에 우리가 사용할 비용함수를 적겠다. 여기 그린 예들을 보면, 어느 정도는 이해가 가능할 것이다. 하지만 어떻게 활용할지는 여러분이 직접 해 보라. 그리고 나서 적절한 연습을 해본다면 여러분은 직접 활용할 수 있을 것이다. 이것이 그 배경설명이다. 지난 강의에서는 우리가 2차함수를 맞추려고 한다면 데이터에 아주 잘 맞다는 것을 발견했다. 그 반면, 너무 고차원의 다항식을 맞추려 한다면 데이터에는 아주 잘 맞았지만 과적합 문제가 있었다. 왜냐하면 일반화가 거의 불가능했기 때문이다. 이것들을 고려하면서 우리는 변수 θ_3 과 θ_4 를 줄여보도록 하자. 이것이 우리의 최적화 목표 혹은 최적화 문제이다. 평소의 제곱 오차 비용함수이다. 이러한 것을 수정해서 플러스 1000 θ_3 제곱, 플러스 1000 θ_4 제곱을 하는 것으로 하자. 이렇게 적는 1000은 아주 큰 숫자이다. 우리는 이 함수를 최소화하려 한다면 이 비용함수를 작게 만드는 것은 θ_3 과 θ_4 가 매우 작을 경우이다. 아니면 θ_3 곱하기 1000을 하기에 이 비용함수는 아주 커질 것이다.

01'47"

따라서 여러분이 이 새로운 함수를 최소화할 때 θ_3 과 θ_4 를 0에 가깝게 할 것이다. 그러면 우리는

이 두 가지를 없애버리는 것처럼 된다. 이렇게 한다면 그냥 2차식만 남을 것이고 이차함수와 아주 조금의 영향을 미치는 θ_3, θ_4 가 있을 것이다. 그리고 이렇게 이차식만 남게 된다. 이것은 더 나은 가설이다. 이런 예에서 우리가 2개의 변수 값을 이렇게 조정하는 것은 매우 다른 결과를 가져온다. 더 일반화하면 이것을 정규화라고 할 수 있다. 이것은 변수가 작은 값을 가지고 있으면 좀 더 간단한 가정을 갖게 된다는 것이다.

02'59"

우리의 예를 보았을 때 θ_3, θ_4 를 이렇게 0에 가깝게 바꿈으로써 이차식만 있는 이런 간단한 가정이 된다. 좀 더 설명하자면 이렇게 모든 변수를 조정하면 더 간단한 가정이 되는데 그 이유는 이 변수가 이 예와 비슷해지기 때문이다. 이렇게 이차식만 남는 것이다. 좀 더 일반적으로 얘기하자면 변수가 작은 값을 갖게 하는 것은 좀 더 간편한 함수를 만든다는 것이다. 그리고 과적합 문제가 줄어들게 된다. 이것이 변수의 값을 작게 하여야 하는 이유이다. 여러분이 아직은 완전 친숙하지 않을 수도 있다. 여러분이 직접 경험해 보지 않는다면 설명하기 조금 난해한 문제이다.

03'57"

하지만 $\theta_{3,4}$ 를 작게 유지하고 간단한 가정을 쓰는 것이 어째서 잘 맞는 것인지에 대한 이해를 돋기를 바란다. 이 예를 보면, 집 가격 예측과 같은 경우 우리는 100개의 특징을 가질 수도 있다. 얘기했던 것처럼 x_1 은 집의 크기, x_2 는 방의 개수, x_3 는 층수 등등이다. 다항식의 예와는 다르게 우리는 θ_3, θ_4 나 다른 다항에 대해 알지 못하고 있다. 우리가 가방이 하나 있고, 100가지 특징이 있는 셋이 있을 때, 어떤 것이 덜 관련이 있는지를 먼저 알아내기란 쉽지 않다. 우리는 100 혹은 101개의 변수를 보유하고 있다. 우리가 어떤 것을 고를지 모르고, 어떤 것을 고르고 고르지 말아야 하는지도 모른다. 그러니 정규화에서 우리가 할 것은 우리의 비용함수, 선형회귀를 위한 비용함수이다. 이 비용함수를 바꾸어서 모든 변수를 줄어들게 할 것인데, 왜냐하면 어떤 변수를 줄여야 할 지 모르기 때문이다. 그래서 비용함수를 수정하여 여기 마지막에 하나를 더 추가할 것이다. 이렇게 괄호가 있다. 그리고 이렇게 마지막에 하나를 추가하여 θ_1 부터 100까지 모든 변수를 줄일 것이다.

05'37"

그런데 여기 이것이 1부터 시작하기 때문에 θ_0 를 크게 조정하지는 않을 것이다. i 가 0이 아니라 1부터 n 까지이다. 하지만 실제로는 이것이 큰 차이가 있다. 여러분이 0을 포함하느냐 안하느냐는 매우 다른 것이다. 하지만 여기서는 θ 를 100까지로만 놓자. 여기 정규화된 최적화 대상을 보면, 우리의 정규화된 비용함수를 여기 볼 수 있다. 이렇게 j θ 이며 여기 오른쪽에 적힌 것은 정규화 항이고 람다는 정규화 변수이다. 이 람다가 하는 것은 2가지의 다른 목표의 균형을 유지하는 것이다. 첫 번째의 목표는 트레이닝 데이터와 잘 맞추려는 것이다. 그리고 두 번째는 우리는 변수를 작게 유지하고자 하고, 여기 정규화 objective에 의해 이렇게 포착되었다. 이 정규화

변수인 람다는 두 가지의 목표 사이의 밸런스를 맞춘다. 그것은 트레이닝 셋과 잘 맞으면서 변수를 작게 유지하는 것이다. 그렇게 되면 가정 함수가 작아지고 과적합을 피할 수 있게 된다. 우리가 앞서 살펴본 집 가격 예를 보면 고차원의 다항식이 있으면 아주 구불구불한 곡선을 이렇게 도출하게 된다. 여기 다항식을 모두 맞추면서 대신 여러분이 이런 유일한 정규화된 대상을 사용한다면 여러분은 이차 함수가 아닌 더 매끄럽고 간단한 이런 붉은 색의 선을 도출하게 된다. 그리고 이것은 데이터에 더 알맞은 가정이라고 할 수 있다. 변수를 변경하는 것이 이런 효과를 가져오는지 이해하는 것은 어렵지만 정규화에 대해서 조금 직접 활용해 본다면 이 효과를 바로 알 수 있을 것이다.

Q) 정규화된 선형 회귀에서 우리는 θ 를 선택해서 다음을 최소화하려고 한다. 만약 λ 값이 굉장히 큰 값을 갖는 것으로 설정되었다면? 정답: 3번

08'01"

이 정규화된 선형 회귀에서, 정규화 변수 모니터가 굉장히 크게 되어 있다면 θ 1,2,3,4를 굉장히 많이 변경해야 한다. 그렇다면 우리의 가정함수는 여기 아래쪽에 적힌 이것과 같다. 우리가 θ 1,2,3,4를 변경한다면 이것을 0에 가깝게 만든다는 것을 의미한다. 이 모두가 0과 비슷하게 되는 것이다. 그렇게 된다면 이 가정함수에서 이것들을 없앨 수 있고, 집 가격은 θ_0 와 같아지는 것이다. 이렇게 평평한 직선을 도출하게 된다. 이것은 과소적합 문제가 될 수 있다. 이런 가정에서는 트레이닝 셋과 잘 들어맞지 않는다. 이것은 전혀 그 셋과 가깝지도 않다. 이것은 어떤 트레이닝 예들과도 맞지 않는다. 이것은 아주 예상이 강하거나, high bias된 경우이고, 그래서 집 가격이 θ_0 와 같아진다. 그리고 이렇게 아주 명확한 데이터에도 불구하고 이렇게 아주 평평한 수평선만을 가지게 된다. 잘 그린 것 같지는 않지만 아주 평평한 수평선이다. 정규화가 잘 활용되게 하려면 더 고려해야 하고, 정규화 람다를 잘 고르는 것도 매우 중요하다. 우리는 멀티 선택에 대해서 얘기할 때 정규화 변수 람다를 자동적으로 고르는 것에 대해서도 배울 것이다. 고수준 정규화와 비용함수에 대해서 복습하게 될 것이다. 그리고 다음 두 영상에서는 이것을 선형회귀나 로지스틱 회귀에 사용해서 과적합 문제를 해결할 것이다.

No.	Title
Week 3-10	Regularized Linear Regression

00'00"

선형회귀에 대해 2가지 학습 알고리즘에 대해 다루었다. 하나는 gradient descent에 기반한 것이고 하나는 일반 등식에 기반한 것이다. 이 영상에서 우리는 이 두 알고리즘을 사용하여 정규화된 선형 회귀에 일반화해서 사용할 것이다. 이것이 정규화된 선형회귀를 위한 최적화 목표이다. 첫 번째 파트는 우리의 선형회귀의 목표함수와 같다. 우리는 이제 또 다른 정규화 용어가 있고, 이것은 람다로 새로운 변수이다. 우리는 정규화된 비용함수 $j(\theta)$ 를 최소화하는 변수 θ 를 찾으려고 한다. 이전에는 gradient descent를 사용해서 정규화를 하지 않고 비용함수를 줄였다. 이제는 우리가 이런 regular 선형회귀라는 알고리즘이 있으니 정규화를 하지 않고 θ J 를 J 는 0,1,2에서 n까지일 경우를 계속적으로 업데이트를 해야 한다. θ_0 은 좀 분리해서 적도록 하겠다. θ_0 에 대해서는 변수 1,2,3에서 n까지와는 다르게 업데이트를 할 것이다. 그리고 이렇게 하는 이유는 우리의 정규화된 선형회귀를 위해서 우리는 $\theta_{1,2}$ 에서 n까지를 업데이트하여야 하기 때문이다. 하지만 θ_0 를 변경하지는 않을 것이다. 우리가 정규화된 선형회귀를 위한 알고리즘을 수정한다면 θ_0 에 대해서는 조금 다르게 다를 것이다.

01'49"

좀 더 자세히 말하면, 우리는 이 알고리즘을 정규화된 대상을 사용하기 위해 수정할 것인데, 우리는 이것을 여기로 가져와서 이렇게 바꿔보도록 하자. 이것에 $(-\lambda/m)^*$ θ_j 를 할 것이다. 이것을 하면 여러분은 정규화된 비용함수 $j(\theta)$ 를 최소화하기 위해 gradient descent를 활용할 수 있을 것이다. 그리고 이를 미적분을 사용해서 확인하지는 않을 것이지만 내가 괄호에 기입한 것을 보면 $j(\theta)$ 를 정규화 항을 사용해 새로이 정의했다. 그리고 유사하게 위에 노란색으로 쓴 것은 $j(\theta)$ 의 θ_0 를 반영한 편미분계수이다. 이 θ_j 의 업데이트 사항을 본다면, 굉장히 흥미로운 것을 볼 수 있을 것이다. 좀 더 자세히 말하면 θ_j 는 θ_j 곱하기 $-\lambda/m$ 으로 업데이트가 될 것이고, θ_j 와 관련이 있는 새로운 항이 생긴다. 이러한 모든 θ_j 에 관련되어 있는 항들을 한다면, 이 업데이트가 이렇게 적히는 것을 알게 될 것이다. 내가 한 것은 θ_j 이다.

Q) 여러분이 0보다 큰 트레이닝 세트 m에 대해 gradient descent를 실행한다고 하자. 0보다 크지만 꽤 작은 학습 속도 α 값과 정규화 매개변수 λ 를 사용한다고 하고 다음의 업데이트 공식을 사용한다. 다음 중 $1 - \alpha \cdot \frac{\lambda}{m}$ 항에 대해 올바르게 진술한 것은? 정답: 3번

그리고 여기 $1 - \alpha \cdot \frac{\lambda}{m}$ 은 꽤 재미있는 부분이다. 이것이 주는 효과는 꽤 흥미롭다. 이 부분은 1보다 주로 작은 숫자가 될 것인데 왜냐하면 알파 곱하기 람다/m은 양수가 되고, 여러분의 학습속도가 작고 m이 크다면 이것은 꽤 작은 숫자가 된다. 그래서 이것은 1보다 조금 작은 수가 되는 것이다. 0.99같은 수라고 생각해 보자. 그러니 이것은 θ_j 곱하기 0.99와 비슷해 지는 것이다. 이것은 θ_j 의 값을 조금씩 축소시켜 0과 비슷하게 만드는 것이다. 이것이 θ_j 의 크기를 줄인다.

04'26"

그리고 이 θ_j 의 square norm을 조금 작게 만든다. 두 번째는 정규화를 시작하기 전 우리가 다루었던 gradient descent의 업데이트와 같은 것이다. 그러나 이 gradient descent가 제대로 작동하길 바란다. 우리가 정규화된 선형회귀를 사용하고 모든 반복에서 θ_j 를 1보다 작은 어떤 수와 곱하게 한다면 변수를 조금씩이나마 줄일 수 있게 되고 이전에 했던 것과 비슷한 업데이트를 하게 될 것이다. 물론 이것은 이 업데이트가 하는 것의 일부분이다. 수학적으로 이 업데이트는 gradient descent가 이 비용함수 $J(\theta)$ 에 대해서 하는 것과 정확히 같다. gradient descent는 선형 회귀 모델에 알맞은 두 가지의 알고리즘 중 하나인 것이다. 두 번째의 알고리즘은 일반 등식을 기반으로 하고 있었고, 각각의 선이 다른 트레이닝 예에 적합했던 행렬 x 를 구성할 때 다루었던 내용이다.

05'39"

우리는 그때 벡터 y 를 만들었다. 이것이 벡터이다. 그리고 이것은 m 차 벡터이다. 그리고 이것은 내 트레이닝 셋의 이름을 땠다. x 가 m 곱하기 $n+1$ 차 행렬인 반면 y 는 m 차 벡터이다. 이 비용함수를 최소화하기 위해서 θ 를 이것과 같이 만들어야 한다. 여러분은 $\theta = (X^T X + \lambda [])^{-1} X^T y$ 와 같이 세타를 나타낼 수 있다. 람다 옆의 공간을 남겨 두었는데, 이것을 다음과 같이 채울 것이다. 우리는 정규화를 사용하지 않을 때 문제를 풀었던 것과 같이 $J(\theta)$ 를 최소화하여야 한다. 하지만 정규화를 사용할 때 최소값을 알아내려면 여러분은 각 변수에 대해서 편미분계수를 유도하여야 한다.

06'40"

이것을 0으로 설정하고, 이렇게 계산해서 비용함수를 최소화할 수 있을 것이다. 좀 더 자세히 설명하자면 여러분이 정규화를 사용하면 이 공식은 이런 식으로 바뀐다. 이 괄호 안에는 0, 1, 1, 1, 그리고 이렇게 마지막까지 1인 행렬을 얻어낼 것이다. 즉 대각선의 가장 왼쪽의 값이 0인 행렬인 것이다. 이 행렬에서 대각선을 제외한 부분은 모두 0으로 이루어져 있을 것이다. 왜냐하면 이것을 사선으로 그리고 있기 때문이다. 그리고 이것은 $n+1, n+1$ 차 행렬이다. $n=2$ 라면 이 행렬은 이렇게 변할 것이다. 0 그리고 이렇게 대각선위에는 1이 있을 것이고 그 나머지에는 0들이 있을 것이다.

07'46"

이는 너무 길고 복잡하기 때문에 어떻게 유도하는지에 대해서는 보여주지 않을 것이다. 하지만 $J(\theta)$ 에 대한 새로운 정의를 사용한다면 증명 가능하다. 그리고 이를 통해서 $J(\theta)$ 의 전역 최소값을 얻을 수 있다. 그리고 이제 비가역성에 관한 내용에 대해 논할 것이다. 이것은 좀 더 고급화된 방법이고 이것을 그냥 부수적인 것으로 생각하라. 그냥 생략하거나 이해가 가지 않더라고 해도 걱정하지 않아도 된다. 우리는 비가역성에 대한 부록 강의도 있다. 이것은 어디까지나 부록이니까 걱정하지 않아도 된다. 예시의 개수인 m 이 변수의 개수인 n 보나 작거나 같다고 하자.

08'41"

여러분이 변수의 수보다 적은 예를 가지고 있다면 이 행렬 XtX 는 전치될 수 없거나 singular이다. 혹은 이 행렬의 다른 것이 없어질 것이다. 여러분이 옥타브에 이것을 실행하고 pinv함수를 pseudo inverse를 위해 사용한다면 이것은 올바른 것이다. 하지만 여전히 이것이 좋은 가정함수를 줄지는 아직 모른다. 수학적으로 옥타브 pinv 함수는 일반적으로 좋은 결과를 제공하지만 말이다. 하지만 여러분이 이것을 다른 언어로 표현하면, 그리고 여러분이 옥타브에서 함수 inv로 표기하는 정규역행렬을 취하면 우리는 XtX 의 정규역행렬을 얻어낼 수 있다. 이러한 것이 형성되면 여러분은 $X^T X$ 는 singular이며 전치될 수 없다. 그리고 여러분이 다른 언어에서 선형대수 라이브러리를 사용해서 이 행렬을 전치하려고 한다면 행렬이 Singular이고 전치될 수 없기에 제대로 작동하지 않는다.

09'45"

다행히도 정규화가 이 문제 또한 해결해 준다. 좀 더 자세히 말하자면 정규화 변수 람다가 0보다 크다면 이 행렬 $X^T X + \lambda []$ 부분은 singular가 아니거나 전치가 가능하다. 그러니 정규화를 사용하면 $X^T X$ 가 비가역적인 문제에 대해서도 해결이 가능하다. 이제 여러분은 정규화된 선형회귀를 알고 있다. 이것을 사용하면 트레이닝 셋의 숫자가 적더라도 과적합 문제를 피할 수 있다. 그리고 이것으로 여러분은 선형 회귀를 더 많은 곳에 적용 할 수 있게 될 것이다. 다음 영상에서 이 정규화를 로지스틱 회귀에 활용할 것이다. 그래서 여러분은 로지스틱 회귀에서 과적합 문제를 해결하고 이것이 더 잘 작동하도록 하는 법을 배울 것이다.

No.	Title
Week 3-11	Regularized Logistic Regression

00'00"

로지스틱 회귀의 경우 우리는 두 가지의 최적화 알고리즘을 대해서 앞에서 다루었다. 우리는 gradient descent를 비용함수 $J(\theta)$ 를 최적화하기 위해 사용하는 법을 다루었고, 고급 최적화 방법에 대해서도 논했다. 비용함수 $J(\theta)$ 와 미분계수를 계산하기 위해서 방법이 필요했다. 이 강의에서 두 가지 테크닉을 어떻게 맞추어 사용할지에 대해 설명할 것이다. gradient descent와 고급 최적화 기술을 활용해서 이를 정규화된 로지스틱 회귀에 적용할 것이다.

00'36"

여기 우리가 보았던 것처럼 로지스틱 회귀는 여러분이 이렇게 고차원의 다향을 가지고 있다면 과적합이 일어날 가능성이 매우 높다. G 가 시그모이드 함수이고 여러분이 이런 가정함수를 도출해 냈다고 하자. 이 가정함수는 과하게 복잡하고 많이 뒤틀려있는 함수라서 이 트레이닝 셋에 알맞은 가정이 아닐 때, 그리고 일반적으로 여러분이 아주 많은 값을 가진 로지스틱 회귀가 있을 때 알맞은 가정이 아니라는 얘기이다. 이것이 꼭 다향식일 필요는 없지만, 특성이 많은 경우에도 과적합 문제를 일으키게 된다. 이것은 로지스틱 회귀를 위한 비용함수이다. 우리가 정규화를 사용해서 이것을 수정하려 하면 $J(\theta) + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$ 이 된다. 그리고 이는 $j=1$ 일때부터의 값이다. 0일때부터가 아니다. 또한 θ_1, θ_2 에서 θ_n 까지의 변수를 변경하면서 효과를 나타낼 것이다.

01'44"

그리고 이렇게 하면 아무리 고차원 다향식이더라도 잘 적용될 것이다. 여러분이 정규화를 적용하고 변수의 크기를 작게 한다면 여러분은 이렇게 결정선을 얻을 가능성이 높다. 알다시피 이것은 이렇게 생겼다. 이것은 음과 양을 나누는 것이 좀 더 합리적으로 보인다. 그러니 여러분이 많은 변수 값이 있을 때 정규화를 사용하면 과적합 문제를 좀 더 쉽게 할 수 있다는 것이다. 어떻게 이것을 활용할 수 있을까? 원래 gradient descent 알고리즘을 보면 우리는 θ 를 계속해서 업데이트해 왔다. 이 슬라이드는 선형회귀와 매우 닮아 보인다. 하지만 θ_0 를 다르게 업데이트 할 것이다. 그러면 첫 번째 선은 θ_0 를 위한 것이고 두 번째 선은 θ_1 에서 θ_n 까지의 업데이트를 뜻한다.

02'39"

왜냐하면 이 θ_0 을 분리해서 다를 것이기 때문이다. 이 알고리즘을 수정하기 위해서, 그리고 비용함수를 정규화를 위해서 우리가 하려는 것은 선형 회귀에서 했던 것과 굉장히 유사하다. 이것은 그리고 이렇게 2번째 업데이트 규칙만 바꾸는 것이다. 그리고 다시 한 번 이것은 우리가 선형 회귀에서 했던 것과 겉으로 보기에 동일해 보인다. 이것은 같은 알고리즘은 아니다. 왜냐하면 가정함수는 이것을 사용한다고 정의되기 때문이다. 그러니 이것은 정규화된 선형 회귀와는 다른 알고리즘이 것이다. 가정 함수가 다르기 때문이다. 우리는 정규화된 선형회귀를 위한 gradient descent를 활용할 것이다. 이 괄호 안에 있는 것은 새로운 편미분계수인데 이것은 비용함수 $J(\theta)$ 의 $\frac{\partial}{\partial \theta} J(\theta)$ 값을 반영한 수치이다. $J(\theta)$ 는 지난번에 정규화에서 다루었던 비용함수이다. 그러므로 이것은 정규화된 선형회귀의 gradient descent이다.

Q) 정규화된 로지스틱 회귀를 사용할 때, gradient descent가 제대로 작동하는지를 감시하는 가장 좋은 방법은 무엇인가? 정답: 3번

03'55"

더 진보된 최적화 방법을 사용해서 정규 선형회귀를 행할 수 있는지에 대해서 논해보자. 비용함수를 정의할 필요가 있는데, 이 때 변수 벡터 세터를 입력하고, 벡터의 지수는 0이다. 우리는 θ_0 에서 N 까지가 있다. 하지만 옥타브는 벡터가 1부터 시작하라고 지정하고 있다. θ_0 은 옥타브에서의 θ_1 로 표기된다. 세터1은 2, 이런 식으로 표기된다. 그리고 우리가 앞서 다루었던 비용함수를 여기에 제공하자. 우리는 비용함수에 fminunc를 사용할 것이고 그 등을 시행할 것이다. F min u and c는 제약되지 않은 F의 최소값이며, 이것은 비용함수를 최소화한 결과이다.

05'06"

비용함수의 값을 도출하기 위해서 첫 번째로 필요한 것은 J-val이다. 이것을 위해서 우리는 비용함수 $J\theta$ 를 계산하기 위해 코드를 짬다. 우리가 정규화된 로지스틱 회귀를 사용할 때, 비용함수 $J\theta$ 는 변하고, 특히 비용함수는 이 추가적인 정규화 항을 식의 마지막에 추가한다. 두 번째로 비용함수를 도출하기 위하여 필요한 것은 gradient를 미분한 것이다. 따라서 gradient(1)는 θ_0 을 이용하여 $J\theta$ 의 편미분계수를 구하고, gradient(2)는 θ_1 을 이용하며 이런 식으로 이어진다. 이 때 인덱스는 1부터 시작된다. 옥타브 코드는 항상 1부터 시작하기 때문이다. 이 부분들을 보면 우리는 지난 슬라이드에서 이것이 이것과 같다고 말했다. 이것은 변하지 않았다. 왜냐하면 θ_0 의 미분계수는 변하지 않았기 때문이다.

06'07"

정규화를 제외한 버전과 비교했을 때 다른 부분들은 변했다. 특히 θ_1 의 미분계수가 변했기 때문이다. 우리는 지난 슬라이드에서 이것을 배웠는데 이것은 원 식이 $J(\theta)$ 일때 $J(\theta) - \frac{\lambda}{m} \theta_1$ 로 주어진다. 이 부분은 그냥 넘어가자. 우리는 이렇게 괄호를 만들 것인데, 그러면 더 이상 확장되지 않는다. 그리고 보이는 것처럼 다른 부분은 이렇게 생겼고, 우리가 지난 슬라이드에서 적었던 이 부분이 더해졌다. 이것은 우리의 정규화 대상의 gradient와 일치한다. 그러니 여러분이 이 비용함수를 시행하고 fminunc나 다른 고급 최적화 기술을 적용한다면 이것은 새로운 정규화된 비용함수 $J\theta$ 를 최소화 시킬 것이다. 그리고 다른 변수는 정규화한 로지스틱 회귀와 일치할 것이다.

07'05"

이제 여러분은 어떻게 정규화된 로지스틱 회귀를 시행할지 알고 있다. 내가 거주하는 실리콘밸리를 걸어 다니면, 여기 엔지니어들은 머신 학습 알고리즘을 사용하는 회사에서 솔직히 많은 돈을 벌고 있다. 하지만 여러분이 선형 회귀, 고급 최적화 알고리즘과 정규화를 이해하고 있다면, 지금까지의 다른 많은 사람들보다 머신 러닝을 더 많이 알고 있는 것이다. 즉, 실리콘밸리의 엔지니어들, 회사에 굉장히 많은 돈을 안겨주거나 머신 러닝 알고리즘을 이용해서 제품을 생산하는 그런 사람들 다수보다도 말이다.

07'50"

축하한다. 여러분은 먼 길을 왔다. 그리고 여러분은 이것을 활용해서 많은 문제를 해결할 수 있다. 축하한다. 하지만 아직 여러분에게 강의할 내용이 많고, 다음 셋에서는 우리는 비선형 classifier에 대해서 논할 것이다. 선형회귀, 로지스틱 회귀와 달리 여러분은 다항식을 만들어 낼 수 있을 것인데, 더 효과적인 비선형 quantifier가 있어서 여러분의 다항 회귀를 해결할 수 있을 것이다. 다음 강의에서 여러분에게 자세히 알려주도록 하겠다. 다른 문제들을 해결하기 위한 더 효과적인 알고리즘이 여러분을 기다리고 있다.

Week 4

No.	Title
Week 4-1	Non-linear Hypotheses

00'00"

이번과 다음 여러 편의 강의에서 신경망이라고 불리는 학습 알고리즘에 대해 설명하려고 한다. 먼저 신경망이 어떻게 작동하는지 설명하고 다음 여러 강의에서 구체적인 학습 알고리즘을 설명하려고 한다. 신경망은 사실 꽤 오래 전에 제시된 방법이지만, 한동안 사람들의 관심을 받지 못했다. 그러나 오늘날, 신경망은 다양한 분야에 기계학습을 응용하기 위한 최첨단의 기법이다. 그렇다면 왜 신경망 알고리즘이 필요할까? 이미 선형 회귀와 로지스틱 회귀가 기계 학습에 사용되고 있다. 그런데 왜 우리는 신경망 알고리즘을 필요로 할까? 신경망 이론에 대한 관심을 불러일으키기 위해, 먼저 몇 가지 기계학습의 예를 보여주겠다. 이와 같은 문제에서는 복잡한 비선형 가설들이 필요하다.

00'43"

지도 학습 분류 문제를 생각해보자. 이 그림과 같은 훈련 데이터가 주어진 문제에 로지스틱 회귀를 적용하려 한다면, 이 식과 같이 수많은 비선형적 요소들을 사용해 로지스틱 회귀를 적용해볼 수 밖에 없을 것이다. 여기 있는 g 함수는 주로 시그모이드 함수를 나타낸다. 여기에서 시그모이드 함수를 사용하기 때문에, 수많은 다항식들을 포함할 수 있다. 만약 이 식에 충분히 많은 항들을 포함한다면, 아마도 양수 값을 가지는 데이터와 음의 값을 가지는 데이터들을 구분할 수 있는 가설을 세울 수 있다. 여기에 제시된 특별한 방법은 오직 두 개의 요소 - x_1 과 x_2 만으로 데이터가 주어졌을 때에만 제대로 동작한다. 왜냐하면, x_1 과 x_2 로 이루어진 모든 항들을 포함할 수 있기 때문이다. 그러나 많은 기계 학습 문제에서는 단지 두 개가 아닌 훨씬 많은 요소들을 고려하게 될 것이다.

01'33"

이제 문제를 주택 수급 예측 문제로 돌려보자. 만약 당신이 회귀 문제가 아닌 주택 분류 문제를 가지고 있다면, 아마도 이런 상황일 것이다. 당신은 어떤 주택이 어떤 요소와 특성을 가졌는지 알고 있을 것이다. 그 상태에서 당신이 앞으로 6개월 이내에 그 주택이 팔릴 가능성이 얼마나 될지 예측하려고 한다면, 그것은 분류 문제일 것이다. 그리고 이와 같이 우리는 아주 많은 특성을 떠올릴 수 있다. 주택마다 각각 다른 요소를 100가지쯤은 떠올릴 수 있다. 이렇게 수없이 다양한 항이 있는 문제를 푸는 데에 모든 항과 이차항을 포함하려고 한다면, 게다가 심지어 제곱항인 이차항에 다변수 이차항까지 포함한다면, 항이 엄청 많아질 것이다. x_1 제곱도 있고, x_1x_2 , x_1x_3 , 계속하자면 x_1x_4 , 결국 x_1x_{100} 까지 모두 포함된다. 다음 x_2 제곱, x_2x_3 , 등등이 계속 이어진다. 만약 당신이 단지 이차항만을, 즉 두 항의 곱, x_1 곱하기 x_1 등등 만을 포함한다고 해도 항이 100개인 경우에 5000개가 넘는 요소를 포함하게 된다. 그리고, 점근적으로, 이차항의 수는 $O(n^2)$ 으로 증가하는데, 여기에서 n 은 원래 요소의 개수이다. 주택 문제에서 떠올렸던 x_1 부터 x_{100} 까지의 변수들과 같다. 그리고, 그것은 사실 2의 n 제곱에 가깝다. 그래서, 모든 이차 요소들을 포함하는 것은 좋은 생각이 아니다. 왜냐하면 그럴 경우 너무 많은 요소들을 포함하게 되어 훈련 데이터에 너무 딱 맞는 결과를 보게 될 것이다(오버피팅). 그리고, 계산하는데 그 수많은 요소들을 다루게 된다면 오래 걸린다.

03'16"

당신이 할 수 있는 한 가지 방법은 수많은 요소들 중 어떤 부분집합만을 포함하는 것이다. 이런 경우를 생각해보자. 단지 x_1 제곱, x_2 제곱, x_3 제곱, x_{100} 제곱까지만 포함한다면, 요소의 수가 훨씬 적어진다. 이렇게 오직 100개의 이차 요소들을 뽑아낼 수 있다. 그러나 이것은 충분한 요소가 아니며 확실히 이 그림처럼 데이터 세트를 잘 가려내지는 못할 것이다. 사실, 정말로 이들 이차 요소들과 원래 x_1 에서 x_{100} 까지의 요소들을 함께 포함한다면, 실제로는 매우 흥미로운 가설을 세울 수 있다. 그러니까, 이 그림처럼 타원 모양의 구분선을 그려낼 수 있다. 하지만 틀림없이 이 그림과 같은 복잡한 구분선은 그려낼 수 없을 것이다. 요소가 5000개나 된다고 생각할 수도 있다. 하지만 만약 세제곱이나 다른 3차 항들, $x_1x_2x_3$ 라던가, x_1 제곱, x_2x_{10} , $x_{11}x_{17}$ 등을 포함하기 시작하면 얼마나 많은 요소가 있을 지 상상해 보라. 요소들은 n 이 늘어날수록 세제곱에 비례해 수가 늘어날 것이고 만약 n 이 100이면, 계산해보라. 대략 170,000개의 삼차 요소들을 포함하게 된다. 이렇게 고차 다항 요소들을 포함하는 것은 당신의 원래 요소의 수가 많을수록 더욱 급격하게 요소 공간을 필요로 한다. 이 역시 좋은 방식이 아니다. 즉 n 이 클 때, 비선형적인 분류를 위해서 요소를 추가하는 방식은 좋지 않다.

04'50"

많은 기계학습 문제에서 n 은 아주 큰 영역을 차지한다. 예를 들어보자. 컴퓨터 비전 문제를 고려해 보자. 그리고 당신이 기계학습을 이용하여 분류기를 트레이닝하여 이미지를 조사하고 이미지가 자동차인지 아닌지를 판단하게 하고 싶다고 가정해보자. 많은 사람들은 왜 컴퓨터 비전이 어려운지 궁금해한다. 당신과 내가 이 그림을 바라볼 때, 이것이 무엇인지는 매우 명백하다. 당신은 어떻게 학습 알고리즘이 이 그림이 무엇인지 아는데 실패할 수 있는지 궁금할 것이다. 왜 컴퓨터 비전이 어려운지를 이해하기 위해서, 작은 영역의 이미지를 확대해서 보도록 하자. 여기 이 빨간 사각형을 확대해보자. 우리는 자동차를 보고 있지만 컴퓨터는 다른 것을 본다. 컴퓨터가 보는 것은 행렬, 혹은 픽셀 값의 격자인데, 이 값은 이미지에서 각 픽셀의 밝기를 나타낸다. 컴퓨터 비전 문제는 컴퓨터가 픽셀 밝기 값들로 이루어진 이런 행렬을 보고, 이 숫자들이 어떤 자동차의 문 손잡이를 나타낸다고 우리에게 말해줘야 하는 문제이다.

05'54"

구체적으로, 우리가 기계학습을 이용하여, 차량 식별기를 구축하기 위해 우리가 해야 하는 것은 라벨 트레이닝 세트를 마련하는 것이다. 즉 "자동차"라고 표시된 예제들과 "자동차가 아님"이라고 표시된 다른 예제들을 준비해야 한다. 그 후에 우리는 트레이닝 세트를 학습 알고리즘에 전달하여 분류기를 학습시키고 그 다음 학습된 알고리즘을 테스트할 것이다. 이렇게, 새로운 이미지를 보여주고는 "이것은 무엇일까?"라고 질문한다. 그리곤 분류기가 그것이 차라고 대답하기를 기대한다. 왜 비선형적인 가설이 필요한지 이해하기 위해서 몇 개의 자동차 이미지와 자동차가 아닌 이미지를 보자. 우리의 학습 알고리즘에게 예제로 주었던 그 이미지이다. 이 이미지에서 몇 개의 픽셀들을 골라보자. 그러면 픽셀 1은 여기서, 픽셀 2는 여기서 고르겠다. 그리고 이 차를 도면에 표시하겠다. 그러니까, 이 좌표값은 픽셀1과 픽셀2의 밝기를 값으로

가지는 좌표값이다. 다른 이미지들도 같은 작업을 해 보자. 다른 "자동차" 예제를 고르고, 동일한 위치의 픽셀 두 개를 보자. 그러면 그 이미지의 픽셀 1은 다른 밝기를, 픽셀 2도 역시 다른 밝기를 가지고 있다. 따라서 다른 "자동차" 그림은 도표상에서 다른 위치에 놓이게 된다. "자동차"가 아닌 예제들도 도면에 표시해보자. 이 이미지는 "자동차가 아님" 예제이고, 이 이미지도 "자동차가 아님" 예제이다. 그리고 만약 우리가 더욱더 많은 예제를 사용해 "자동차"들을 나타내기 위해 +를, "자동차가 아님"을 나타내기 위해 -를 사용해 표시하면, 우리는 "자동차"와 "자동차가 아님"들이 좌표 공간에서 서로 다른 영역에 위치하게 된다는 것을 발견하게 될 것이다. 그러므로 우리가 필요로 하는 것은 어떤 비선형적인 가설이며 이를 사용해 두 가지 경우를 구분하도록 시도하는 것이다.

07'35"

특성 공간은 몇 차원일까? 우리가 단지 50×50 픽셀 이미지를 이용한다고 가정해 보자. 우리가 사용하는 이미지들은 단지 가로 길이가 50 픽셀인 작은 이미지이다. 그러나 결국 우리는 2500 개의 픽셀을 가지게 된다. 따라서 특성 공간의 차원은 n 이 2500 이며, 우리의 특성 벡터 x 는 모든 픽셀 값의 목록이다. 픽셀 1의 밝기, 픽셀 2의 밝기, 등등 마지막 픽셀의 밝기까지 이어진다. 전형적인 컴퓨터 표현에서 밝기 값은 0 에서 255 사이의 값이다. 흑백 이미지를 사용할 경우에, 즉 n 은 2500 이다. 만약 우리가 흑백 이미지들을 이용한다고 가정하에서는 그렇다. 만약 우리가 분리된 RGB 이미지를 이용한다면, 이미지는 적색, 녹색, 청색 값을 각각 가지며, n 이 7500 이 된다. 우리가 모든 이차 요소들을 포함하는 비선형 가설을 학습하려고 하면 이차 요소라는 게 x_i 곱하기 x_j 형태의 모든 항이니까 2500 픽셀이 결국엔 3 백 만개에 달하는 요소항으로 늘어난다. 이성적으로, 그건 너무 많다. 계산이 무척 오래 걸릴 것이다. 매 예제마다 300 만개의 특성을 찾고 표현해야 하기 때문이다.

08'54"

자동차를 $100 * 100$ 픽셀 이미지(흑백, RGB 아님)로 구별하기 위해 학습한다고 가정해보자. 픽셀 밝기 값을 특성으로 하고, 만약 모든 이차항을 포함하여 x_1, x_2 을 특성으로 하여 로지스틱 회귀를 훈련시킨다면 얼마나 많은 특성을 가질 수 있는가?

(정답은 3번)

단순 로지스틱 회귀를 이차, 또는 삼차 특성들과 함께하여, n 이 클 때, 복잡한 비선형 가설을 학습하는데 사용하는 것은 좋은 방법이 아니다. 왜냐하면, 너무 많은 특성들이 나타나기 때문이다. 다음 몇 편의 강의에서 신경망에 대해서 설명할 것이다. 이는 복잡한 비선형 가설을 학습하는데 훨씬 좋은 방법이다. 심지어 여러분의 입력 특성 공간 n 이 클 때의 경우에도 말이다. 중간에 나는 재미있는 영상도 몇 편 보여줄 것이다. 이들은 신경망의 역사적으로 중요한 응용이며, 우리가 나중에 함께 볼 영상이 흥미롭기를 바란다.

No.	Title
Week 4-2	Neurons and the Brain

00'00"

신경망은 아주 오래된 알고리즘이다. 원래 뇌를 모방할 수 있는 기계를 만들기 위한 모티브에서 시작된 것이다. 물론, 이번 강의에서 신경망을 강의할 것이다. 이 신경망은 다양한 기계학습문제에서도 잘 동작하고 하지만, 그렇게 논리적이지는 않아 보인다. 이 동영상 강의에서, 신경망의 배경에 대해서 이야기를 하고 싶다. 그래서, 우리는 그것이 어떻게 동작할지 감을 잡을 수 있다. 최근에 기계 문제를 활용하는 것과 또 강의를 듣는 이들 중에 진정한 지능기계를 만드는 AI에 대해 큰 꿈을 꾸고 있을지도 모르는 이들에게 도움이 될 것이다. 그렇다면 어떻게 신경망과 관련되어 있을까?

00'41"

신경망의 시작은 뇌를 모방하려고 하는 알고리즘이었고, 만약에 우리가 학습시스템을 만들기를 원하는데 아무래도 흉내 낼 수 없기 때문에, 가장 놀라운 우리가 아는 학습기계는 아마 뇌일 것이다. 신경망은 1980년대와 1990년대에 널리 사용되었고, 어떤 이유로 90년대 후반에 그 인기가 줄어들었다. 하지만 최근에, 신경망분야는 최근 다시 급부상하게 되었다.. 그 이유 중에 하나는 신경망분야가 원래 계산하는데 어느 정도 비용이 큰 알고리즘이었는데, 최근에 컴퓨터 성능이 충분히 좋아져서 방대한 양의 신경망도 돌릴 수 있게 되었다. 우리가 이후에 이야기할 다른 기술인 최신 신경망분야도 그 이유 중 하나로 태동했으며, 이는 다양한 응용분야에서 사용되는 최고의 기술이다.

01'40"

뇌를 모방한다고 했을 때, 인간의 두뇌는 정말 다양한 멋진 일들을 해내고 있다. 두뇌는 이미지를 보고, 듣고, 촉각을 통해 학습한다. 우리는 수학을 배우고, 미적분도 배우는데 두뇌는 정말 다양한 많은 일을 하고 있는 것이다. 그래서 두뇌를 모방하려고 하면 다양한 소프트웨어 조각을 활용해야만 다양한 명령을 하는 두뇌를 모방할 수 있을 거라 생각한다. 하지만 두뇌를 모방하기 위해서 수 천 가지 프로그램이 필요한 게 아니라 그저 하나의 학습 알고리즘만 필요하다면 얼마나 놀라운 가설인가?

02'19"

이는 그저 하나의 가설일 뿐이지만 이를 뒷받침하는 증거를 함께 소개하겠다. 여기 빨갛게 표시되어 있는 두뇌의 한 부분은 청각 피질이다. 지금 여러분이 내 목소리를 알아들을 수 있는 방법은 귀에서 소리 신호를 듣고 이 신호를 청각 피질로 전송하는 것이다. 그렇게 내 말이

전달되는 것이다. 신경과학자들은 다음에 설명할 엄청난 실험을 했는데, 귀와 청각피질을 연결한 선을 잘라 다시 연결하면, 이 사례는 동물의 두뇌이다. 눈에서 시신경으로 보낸 신호가 최종적으로 청각 피질로 전달된다는 것이다. 이는 청각 피질이 보는 능력을 학습했다는 것을 보여준다. 이는 우리가 알고 있는 다른 모든 감각에도 적용된다. 따라서 동물에게 이와 같은 실험을 하면, 그 동물은 이미지를 보면서 이에 따라 적절한 결정을 내리는 시각 구별 업무를 수행한다. 이는 바로 이 두뇌 조직의 한 부분 덕분에 가능한 것이다.

03'19"

여기 또 다른 예시가 있다. 붉게 표시된 부분은 체성감각 피질이다. 촉각을 담당하고 있다. 아까와 마찬가지로 비슷한 재배선 과정을 거치면 이 피질 또한 보는 능력을 학습하게 될 것이다. 이 실험과 또 비슷한 다른 여러 실험은 신경 재배선 실험이라고 불린다. 만약 같은 두뇌 조직 부분에서 시각, 청각, 촉각을 처리할 수 있다면, 이러한 감각을 처리하게 해주는 하나의 학습 알고리즘이 있을 것이라는 예측이 있다. 그러므로 수천 개의 프로그램이나 알고리즘을 시행해보는 대신에 수천 개의 멋진 일을 해내고 있는 두뇌를 알기 위해서는 몇 개의 근사치를 가지고 두뇌의 학습 알고리즘이 무엇인지, 두뇌가 다양한 형태의 데이터를 처리하는 것을 스스로 학습했는지를 시행해 보는 것이다. 매우 놀랍게도 거의 어떠한 감각도 두뇌의 어느 부분에라도 주입할 수 있는 것 같다. 그러므로 두뇌는 이를 처리하기 위해 학습할 것이다.

04'25"

몇 가지 예를 더 들겠다. 왼쪽 상단에 혀를 통해 앞을 볼 수 있는 예시다. 이는 FDA 승인을 기다리고 있는 시각장애인들을 돋기 위한 브레인포트라는 시스템이다. 이것이 처리되는 과정은 흑백 카메라를 이마에 묶고 앞을 향해 서서 여러분 앞에 있는 저해상도 흑백 이미지를 촬영한다. 그리고 전극판을 연결하여 이를 혀에 올려놓아 각각 픽셀이 어느 장소의 지도를 만들어 고전압은 어두운 픽셀로 저전압은 밝은 픽셀로 구현시키는 것이다. 오늘날 이러한 형태의 시스템으로 우리들은 수십분 후면 혀를 통해서 볼 수 있는 능력을 학습하게 된다. 두 번째 예시는 인간 반향 위치 측정 또는 인간 음파 탐지기다. 두 가지 방법이 있다. 손가락을 튕기거나, 혀를 찰 수 있다. 생각보다 어렵다. 우리 주변에 시각장애인들이 있는데, 그들은 학교에서 이와 같은 방법으로 주변 환경에서 되돌아오는 소리 패턴을 해석하는 방법을 배운다. 이게 바로 음파 탐지다. 나중에 유튜브를 검색해보면 안타깝게도 암 때문에 안구를 모두 잃은 한 소년에 대한 영상을 찾을 수 있을 것이다. 그는 말 그대로 안구가 없다. 하지만 그는 손가락을 튕기면서 어떤 것에도 부딪치지 않고 잘 걸어 다닌다. 스케이트보드도 탄다. 심지어 농구공을 골대에 넣기도 한다. 다시 말하지만 그는 안구를 잃었다.

06'00"

세 번째 예시는 바로 촉각 벨트다. 허리에 띠를 차고 버저를 누르면 가장 북쪽에 있는 것이 울리는 시스템이다. 마치 새들이 가지고 있는 북쪽이 어디인지를 아는 방향감각을 인간에게 주는 것과 비슷하다. 그리고 여기 좀 징그러운 예시가 있는데, 개구리에게 세 번째 눈을 이식하는 것이다.

그렇게 되면 개구리는 새로운 눈 또한 사용하는 법을 학습하게 된다. 어떠한 감각이라도 두뇌에 주입할 수 있고 두뇌 학습 알고리즘이 습득한 데이터를 가지고 학습하는 법을 알아내고 이를 처리한다는 것은 정말 놀라운 사실이다. 그러므로 두뇌 학습 알고리즘에 대해 우리가 알아내서, 이를 시행하거나 알고리즘의 어떠한 근사치라도 컴퓨터에 돌려본다면, AI 즉 인공지능의 꿈을 현실화해서 언젠가 진정한 인공지능 머신을 만들 수 있는 가장 최선의 접근법이 아닐까 싶다.

07'00"

아직 내가 신경망에 대해서는 강의를 하지 않았다. 이번 강의를 통해 아직은 먼 얘기 같은 인공지능에 대한 꿈에 대해 살펴보는 시간이었다. 개인적으로도 내가 계속 연구하는 주제이기도 하다. 하지만 내가 이 과정에서 신경망을 강의하는 가장 큰 이유는 오늘날 이는 기계학습 응용을 위한 매우 효과적인 최첨단 기술이기 때문이다. 이제 다음 강의에서는 신경망과 관련한 기술적인 세부사항에 대해 다뤄볼 것이다. 이를 통해 오늘날 기계 학습 응용에 적용하여 문제를 해결할 수 있다. 하지만 개인적으로 또 다른 이유는 우리가 만약 어떠한 알고리즘이 미래에 인류처럼 학습을 할 수 있는 것을 발견하게 되면, 이를 통해 우리가 하려는 일에 새로운 기회를 제공할 수 있을지도 모르기 때문이다.

No.	Title
Week 4-3	Model Representation I

00'00"

이번 강의에서 우리가 어떻게 신경망을 표현하고 있는지 먼저 알아보려고 한다. 다시 말해, 우리의 가설을 어떻게 내세우고 있는지, 또는 신경망을 이용하는 모델을 어떻게 보여주고 있는지 말이다. 신경망은 두뇌에 있는 뉴런을 또는 뉴런망을 모방하여 개발되었다. 가설에 대해 설명에 들어가기 전에 먼저 뇌에 있는 뉴런 하나가 어떻게 생겼는지 살펴보자. 우리의 두뇌는 이처럼 뉴런이 빽빽하게 가득 차 있고, 뉴런은 뇌 안에 있는 세포이다. 여기서 살펴봐야 할 중요한 두 가지가 있는데 먼저 뉴런은 이렇게 세포체를 가지고 있고, 또한 수상돌기라고 하는 입력을 받는 연결선이 있다. 입력 연결선이라고 하면 다른 장소에서 무언가가 받아들이는 역할을 한다. 또한 출력 연결선이 있는데 이는 축삭돌기라고 한다. 이 축삭돌기는 다른 뉴런에 신호를 보낼 때 사용되고, 다른 뉴런에게 메시지를 보낸다. 그러므로 뉴런이 무엇인지 간단하게 말하면, 입력 연결선을 통해 수많은 입력을 받아 계산하고는 이를 축삭돌기를 통해 내보내 다른 절이나 뇌 안의 다른 뉴런에게 보내는 계산적 단위다.

01'21"

여기에 뉴런 여러 개가 있는 그림이 있다. 뉴런이 다른 뉴런과 커뮤니케이션 하는 방법은 전기 자극을 통해서다. 이를 스파이크라고도 말하는데 이는 단순히 전기 자극만을 의미한다. 여기에 뉴런이 하나 있는데 이는 메시지를 보내고 싶을 때 축삭돌기를 통해 여기 있는 다른 뉴런에게

작은 전기 자극을 보낸다. 여기 다른 축삭돌기가 있고, 여기 입력 연결선이 있는데 이는 여기 연결 되어있는 또 다른 뉴런의 수상돌기와 연결되어있어서 들어오는 메시지를 받아서 처리한다. 그리고 반대로 여기 축삭돌기를 통해 메시지를 다른 뉴런에게 내보낸다. 이것이 바로 우리 인간 생각하는 것은 바로 이러한 과정을 통해 이루어진다. 이러한 뉴런은 어떠한 입력 신호를 받게 되면, 그 결과로 이를 처리하고 메시지를 다른 뉴런에게 전달한다. 이는 또한 우리 감각과 근육이 움직이는 과정과 같다. 신체의 어느 근육을 움직이고 싶으면, 뉴런에서 근육에 전기 신호를 보내고 이를 통해 근육이 수축한다. 눈과 같은 경우에도 뇌에 메시지를 보내야 하는데, 전기 자극을 뇌에 있는 뉴런에게 보내서 볼 수 있게 해준다.

02'43"

신경망에서 아니면 우리가 컴퓨터에 만들어 놓은 가짜 신경망에서 뉴런이 어떤 일을 하는지에 대한 아주 간단한 모델을 활용할 것이다. 여기 로지스틱 단위로 뉴런을 모형화 해봤다. 먼저 이렇게 노란 원을 그리면, 이는 분석하는 역할을 하는 뉴런의 몸체라고 생각할 것이다. 그리고 우리는 이제 이 뉴런에게 수상돌기 또는 입력 연결선처럼 몇 개의 입력을 준다. 그러면 뉴런은 이를 처리할 것이다. 그리고 이 결과로 어떠한 값을 출력 연결선(축삭돌기)을 통해 또 다른 뉴런에 보낸다. 이와 같은 도표는 수식으로는 $h\theta(x) = 1/(1 + e^{-\theta^T x})$ 을 의미한다. 그리고 x 와 θ 는 매개변수 벡터가 된다.

03'43"

이는 매우 간단하게 표현해 놓은 것이다. 그러니까 뉴런이 계산하는 과정을 매우 간단하게 표현한 예시라고 할 수 있다. x_1, x_2, x_3 와 같은 입력을 받게 되면 이를 계산해 어떠한 값을 만들어 낸다. 신경망을 수식화 할 때 보통 x_1, x_2, x_3 을 입력절로 그리는데 때로는 필요할 경우, x_0 을 추가하기도 한다. 여기서 x_0 은 bias 단위 또는 bias 뉴런이라고 불리는 데 이는 $x_0 = 1$ 이기 때문이다. x_0 는 포함될 때도 있고 아닐 때도 있는데 이는 그저 그 예시에 표기하기 편한가에 달려있다.

마지막으로 관련 용어 하나를 더 알아보면, 신경망을 이야기할 때 우리는 뉴런 또는 인공 뉴런을 시그모이드(Sigmoid) 함수 또는 로지스틱 활성 함수이라고 말한다. 이 활성 함수는 신경망 용어인 것이다. 이는 또한 $g(z) = 1 / (1 + e^{-z})$ 이렇게 비선형 함수로 표현할 수 있다. 반면에 지금까지 세타값을 모델의 매개 변수로 불렀는데, 나는 주로 이 용어를 사용할 것이다. 여기에 신경망에서는 어떤 이들은 매개 변수 대신 가중치라고 말하기도 하는데 이는 매개 변수와 같은 말이다. 이번 과정에서는 매개 변수라고 계속 해서 사용할 예정이다. 하지만 가중치라고 불리기도 하니 참고하길 바란다. 이 간단한 도표가 바로 하나의 뉴런을 의미한다.

05'33"

신경망은 무엇인가? 신경망은 다른 뉴런이 서로 강하게 연결되어 있는 집합이다. 전체적으로 여기에 입력 단위 x_1, x_2, x_3 가 있고 다시 말하지만 x_0 를 추가할 수도 하지 않을 수도 있다. 그리고 여기에 뉴런 세 개는 a_1, a_2, a_3 로 표현되어 있다. 여기 괄호 안 지수는 나중에 다시 설명하겠다. 그리고 또한 여기에도 a_0 를 집어넣어 이는 bias 단위가 된다. 여기서 a_0 는 항상

값이 1이다. 그리고 마지막으로 세 번째 절, 마지막 단계에는 $h\theta(x)$ 가설이 계산해내는 값을 출력한다. 관련 용어를 좀 더 소개하자면, 신경망에서 첫 번째 단계는 입력 계층이라고도 불리는데 이는 x_1, x_2, x_3 특성을 입력하는 곳이기 때문이다. 그리고 이 곳은 가설을 통해 최종 값을 계산 해내는 곳이다. 그리고 둘 사이에 두 번째 단계는 은닉 계층이라고도 불린다. 썩 괜찮은 용어는 아닌 것 같은데, 이러한 용어화는 아까 봤듯이 입력을 하고 정확한 출력을 얻는 과정에서 숨겨진 단계의 값은 이러한 훈련 셋업에서 관찰할 수 없다. x 값이나 y 값이 아니기 때문에 은닉 계층이라고 불린다. 또한 신경망에 은닉 계층이 한 개 이상이라고 하는데, 여기 예시에서는 첫 번째 입력 계층도 하나 있고, 또 두 번째 은닉 계층도 하나, 그리고 세 번째 출력 계층도 하나이다. 하지만 정리하자면 입력 또는 출력 단계가 아닌 것은 모두 은닉 계층이라고 불린다.

07'30"

이번에는 신경망이 어떤 역할을 하는지 더 자세히 알아보자. 이 도표에 표시된 대로 계산 과정이 이루어지는 단계로 넘어가보자. 신경망으로 표현되는 이러한 특정한 계산을 설명하려면, 여기에 또 다른 표기법이 필요하다. a 위첨자 j , 아래첨자 i 는 j 계층의 뉴론 i 또는 유닛 i 를 의미한다. 따라서 이것은 위첨자는 1이 되고, 은닉 계층인 계층 2의 첫 번째 유닛으로 활성화된다. 그러므로 이러한 활성화를 통해 계산화된 값과 출력을 의미한다. 여기에 새로운 망은 이러한 행렬에 의해 매개 변수로 표시된다. 세타 위첨자 j 는 계층 1에서 계층2 또는 계층 2에서 계층 3 함수를 조정하는 가중치의 행렬을 나타낸다.

08'32"

여기에 계산 과정을 도표로 보여주려고 한다. 여기 첫 번째 은닉 유닛은 이와 같은 계산된 값을 지닌다. a_{21} 은 다음의 시그모이드 활성함수 또는 로지스틱 활성 함수로도 불리는 시그모이드 함수와 같은 값이다. 이는 이와 같은 입력의 선형 결합에 적용된다. 두 번째 은닉 유닛은 이러한 시그모이드 활성 값을 가지게 된다. 그리고 또 세 번째 은닉 유닛도 같은 공식에 의해 계산된다. 여기에 입력 유닛과 은닉 유닛이 세 개씩 있는데 세타 1의 범위는 다른 세 유닛과 세 은닉 유닛을 나타낸 매개 변수의 행렬이다.

세타 1은 3×4 행렬이 될 것이다. 만약 네트워크에 계층 j 에 S_j 유닛이 있고, 계층 $j+1$ S_{j+1} 이 있다면 j 계층과 $j+1$ 계층을 좌우하는 세타 j 행렬의 크기는 $S_j + 1 \times S_j + 1$ 이 될 것이다. 다시 한번 확실히 설명하면, $(S_j \text{ 아래첨자 } j+1) \times (S_j \text{ 아래첨자 } j) + 1$ 이다. 그러니까 $(S_j \text{ 아래첨자 } j+1) \times (S_j \text{ 아래첨자 } j) + 1$ 이니 맨 마지막 1은 아래첨자가 아니다.

10'32"

지금까지 은닉 유닛 세 가지가 그들의 값을 계산하는 과정을 살펴보았다. 마지막으로 이 부분이 아직 비어있는데, 유닛 하나가 더 남았는데 이는 $h\theta(x)$ 는 $a(3)1$ 과 같고 그 뒤에 보이는 수식과도 같다. 여기 보면 위첨자 2를 썼는데, 이는 세타 위첨자 2가 두 번째 계층 유닛인 은닉 유닛에서 측정한 함수를 컨트롤하는 매개 변수 행렬 또는 가중치 행렬이기 때문이다.

11'13"

지금껏 배운 것을 정리해보면 여기 이 그림은 인공 신경망인데 h 함수에 x 입력값을 통해 y 값을 제공하는 것을 나타낸 그림이다. 또한 이 가설은 매개 변수에 의해 매개화 된 것을 대문자 세타로 표현하고 이 세타를 변형시키면서 다른 가설과 x 부터 y 까지 매핑하는 함수를 얻게 되는 것이다.

Q) 다음 신경망 그림을 살펴보자.

세타 1의 값은?

(정답 4번)

11'42"

따라서 이는 신경망에서 가설이 수학적 정의로 어떻게 표현되는지 알아보았다. 다음 번 강의에서는 이러한 가설이 표현하고자 하는 것과 예시 몇 개를 살펴보고 이를 효과적으로 계산하는 방법에 대해 이야기 나누겠다.

No.	Title
Week 4-4	Model Representation II

00'00"

지난 강의에서 우리는 수학적 정의를 이용하여 신경망에서 사용되는 가설을 어떻게 표현하고 계산하는지 알아보았다. 이번 강의에서는 이러한 계산 과정을 어떻게 하면 효과적으로 시행할 수 있는지 알아보고, 벡터화 구현을 보여줄 것이다. 두 번째로, 더 중요한 것은, 신경망 표현이 왜 좋은 아이디어인지와 이는 어떻게 복잡한 비선형 가설을 학습할 수 있도록 도와주는지에 대한 개념을 알아보자.

00'30"

이 신경망을 살펴보면, 이전에 우리는 가설의 출력을 계산하기 위해 필요한 단계의 시퀀스는 여기 왼쪽에 있는 이러한 방정식이라고 배웠다. 이 방정식으로 은닉 유닛 세 개의 활성값을 계산하고, 그 값을 계산하여 $h(x)$ 가설의 최종 출력을 계산한다. 이번에는 단어 몇 가지에 대해 정의해보고자 한다. 내가 밑줄 긋는 부분을 $z(2)1$ 로 표현하겠다. $a(2)1 = g(z(2)1)1$ 이 된다. 여기서 위함수 2는 무엇을 의미하느냐면 z_2 , a_2 에서 괄호 안의 위함수 2는 이 값이 계층 2, 그러니까 신경망의 은닉 계층과 연관되어있다는 뜻이다.

01'23"

지금 밑줄 긋는 부분은 비슷하게 $z(2)2$ 로 표현하겠다. 마지막으로 밑줄 그은 이 부분은 $z(2)3$ 으로 정의하겠다. 그러므로 $a(2)3 = g(z(2)3)3$ 이 된다. 여기 z 값은 선형 결합이기 때문에 x_0, x_1, x_2, x_3 입력값으로 가중된 선형 결합은 특정 뉴런으로 가게 된다. 이제 이 숫자 블록을 보면 이 숫자들이 행렬 벡터 작업 또는 행렬 벡터 곱셈과 비슷하게 대응한다는 것을 알아냈을지도 모른다. 이러한 관측을 통해 이와 같은 신경망 계산을 벡터화할 수 있게 된다.

02'22"

이제 구체적으로 벡터 x 특성을 평소처럼 x_0, x_1, x_2, x_3 벡터라고 정의하자. 여기서 x_0 는 항상 1이다. 그리고 이는 z_2 는 $z(2)1, z(2)2, z(2)3$ 처럼 이러한 z 값의 벡터라고 정하자. 여기서 z_2 는 3차원 벡터다. 이제 우리는 $a(2)1, a(2)2, a(2)3$ 계산을 벡터화하겠다. 이를 두 단계로 정리할 수 있다. 먼저 $z(2) = \theta(1)x$ 라고 하면 이는 여기 벡터 z_2 를 나타내고, $a(2) = g(z(2))$ 이다. 여기 $z(2)$ 는 삼차원 벡터이고, $a(2)$ 역시 3차원 벡터이기 때문에 여기 활성화 g 는 $z(2)$ 의 각각 요소에 시그모이드 함수 원소 연산(element-wise)을 적용한다.

03'18"

우리가 나중에 살펴볼 내용에 대한 표기법에 일관성을 유지하기 위해서, 여기 입력 계층에서는 x 가 있는데, 이는 또한 첫 번째 계층의 활성화라고 볼 수 있다. 그래서 $a(1) = x$ 라고 정의한다면, $a(1)$ 이 벡터화되어 여기에서 x 대신에 $a(1)$ 를 넣어 $z(2) = \theta(1)a(1)$ 이 성립되는데, 입력 계층에 $a(1)$ 을 활성화시키기만 하면 된다. 이제 지금까지 적은 것을 바탕으로 a_1, a_2, a_3 값을 가지고 있는데, 위첨자를 빼먹지 말아야 한다. $a(2)1, a(2)2, a(2)3$. 여기서 값이 하나 더 필요해서 $a(2)0$ 값을 가져온다. 이는 은닉 계층에 있는 bias 유닛과 상응하여 여기에 출력하게 된다. 물론 여기에도 bias 유닛이 올 수 있다. 여기에 그리지 않았을 뿐이다. 이 또 다른 bias 유닛을 사용해 $a(2)0$ 를 추가해 $a(2)0 = 1$ 이고 이 다음에는 $a(2)$ 가 4차원 특성 벡터가 되므로, 방금 전에 $a_0 = 1$ 을 추가 했기 때문에, 그리고 이는 은닉 계층에 있는 bias 유닛과 상응한다. 마지막으로 가설의 실제 값 출력 계산하기 위해서 간단하게 $z(3)$ 를 계산하면 된다. 여기서 $z(3)$ 는 지금 밑줄 긋는 부분과 대응되는데 이 안쪽 네모 친 부분이 $z(3)$ 다. 또한 $z(3) = \theta(2)a(2)$ 이므로 결국 가설 출력 $h \theta(x) = a(3)$ 이고 출력 계층에서 유일한 유닛으로 활성화되었다. 이게 실제 숫자다. $a(3)$ 나 $a(3)1$, 또는 $g(z(3))$ 로 쓸 수 있다.

05'13"

이렇게 $h \theta(x)$ 를 연산하는 과정은 전방향 전파라고 불리기도 하는데, 그 이유는 입력 단위의 활성화하며 시작하여, 그 다음에 이를 은닉계층에 전방향 전파하여 은닉 계층의 활성화를 연산하기 때문이다. 또한, 이렇게 입력에서부터 활성화된 연산 과정 다음에는 은닉, 그리고 출력 계층으로 이어지므로, 이는 전방향 전파라고 불리는 것이다. 지금까지 이러한 과정을 벡터 요소 구현해보았다. 따라서 여기 오른쪽에 있는 방정식을 활용하여 여기 이를 구현하면, $h \theta(x)$ 를 연산하는 효율적 방법을 제시하는 것이다.

05'58"

이러한 전방향 전파는 신경망의 역할과 이들이 흥미로운 비선형 가설에 대해 배워야 하는지를 이해하는 데 도움을 준다. 다음의 신경망을 보면서 이 그림의 왼쪽 면을 가려보겠다. 나머지 부분을 보면, 로지스틱 회귀와 비슷한 형태를 가지고 있는데, 우리는 이제 로지스틱 회귀 단위를 사용하여 $h_{\theta}(x)$ 를 예측하려고 한다. 예시를 보면, 가설은 $h_{\theta}(x) = g(\theta_0a_0 + \theta_1a_1 + \theta_2a_2 + \theta_3a_3)$ 출력하고 있고 여기서 a_1, a_2, a_3 는 여기 은닉 계층에서 주어진 단위다.

07'01"

아까 언급한 것의 통일성을 유지하기 위해 여기에 이렇게 위첨자 2를 추가하겠다. 그리고 여기 인덱스 1도 있는데, 출력 단위가 하나뿐이기 때문이다. 이제 파란색으로 표시된 부분을 주시해보면, 이는 매우 표준 로지스틱 회귀 모델과 흡사하다. 소문자 세타가 아니라 대문자 세타를 썼다는 사실만 빼면 말이다. 그리고 여기서는 이제 로지스틱 회귀를 할 것이다. 하지만 로지스틱 회귀에 이러한 특성이 입력되면, 은닉 계층에서 연산된 이러한 값이 나온다. 다시 말해, 이러한 신경망의 역할은 로지스틱 회귀와 같은데, 원래 특성인 x_1, x_2, x_3 대신에 새로운 특성 a_1, a_2, a_3 를 사용했다는 것이다. 아까처럼 위첨자를 추가하고, 일관성을 위해서 말이다. 이것의 멋진 점은 여기 특성 a_1, a_2, a_3 입력의 기능으로 활용될 것임을 스스로 학습한다는 것이다.

08'10"

예를 들면, layer 1부터 layer 2까지 매핑 되는 함수는, 이는 다른 매개 변수 세트, 세타 1에 의해 결정된다. 이와 마찬가지로, 신경망 또한 특성 x_1, x_2, x_3 을 로지스틱 회귀에서 사용하라고 제한되지 않고, 자신의 특성인 a_1, a_2, a_3 를 학습하여 이를 로지스틱 회귀에 적용한다. 세타 1의 매개 변수로 어떤 것을 선택하느냐에 따라서 여러분은 꽤 흥미로우면서 복잡한 특성을 학습할 수 있다. 그러므로 처음 그대로의 특성 x_1, x_2, x_3 가지고 제한을 받지 않고, x_1x_2, x_2x_3 와 같은 다항식을 선택해야 할 때보다 더 나은 가설을 만들어낸다. 대신에 이 알고리즘은 어떠한 특성도 여기 a_1, a_2, a_3 를 활용하여 마지막 유닛, 즉 로지스틱 회귀에 대입하여 즉시 학습하는 유연성을 지니고 있다.

09'11"

나는 이 예시가 어느 정도 고차원이라고 생각한다. 그래서 이 신경망에 대한 개념이, 복잡한 특성을 가지고 있으므로 아직 이해하기 어려울 것이다. 그래서 다음 두 강의에서 신경망이 이러한 은닉 계층을 활용하여 더 복잡한 특성을 연산하고 이를 최종 출력 단계에 대입하며, 이를 통해 어떻게 복잡한 가설을 학습할 수 있는지 구체적인 예시를 보여줄 것이다. 만약 이번 강의 내용이 머릿속에 잘 들어오지 않는다면, 이후 두 강의를 따라오면서 예시를 통해 이게 어떠한 과정인지를 조금 더 이해할 수 있으면 한다. 하나 짚고 넘어가자면, 신경망은 또 다른 형태의 도표로 나타날 수 있는데, 이러한 신경망이 연결된 방법은, architecture라고 불린다. architecture라는 용어는 다른 뉴런이 어떻게 서로 연결되어 있는가를 말하는 것이다. 또한 여기 두 번째 계층이, 여기 세 개의 heading 단위가 있고 이는 입력 계층의 복잡한 특성을 연산하고, 더 복잡한 특성을 세 번째 계층에서 하므로, 출력 계층에 다다를 때쯤이면, 네 번째 계층, 세 번째 계층에서 연산한 것보다 더욱더 복잡한 특성을 만날 수 있으니, 비선형 가설이 얼마나 흥미로운

가.

10'36"

이러한 네트워크에서 첫 번째 계층은 입력 계층이라고 불리고, 4번째 계층은 출력 계층 그리고 여기 두 개는 은닉 계층이다. 그러니까 입력 계층이나 출력 계층이 아니면 모두 은닉 계층이 된다.

Q) 다음 네트워크를 살펴보자. 다음과 같은 식이 주어졌을 때, $a(2)$ 를 계산하는 방법은?

정답 3번

이번 강의를 통해 신경망에서 피드 전방향 전파 단계가 작동되는지를 이해하는데 도움이 되었으면 한다. 이는 입력 계층과 전방향 전파의 활성화가 첫 번째 은닉 계층으로 시작되어 두 번째 은닉 계층으로 이어져 마지막 출력 계층으로 부분이다. 또한 우리가 이러한 연산을 어떻게 벡터화 하는지도 알아봤다.

11'13"

다음 번에, 이번 강의에서 어떻게 서로 다른 특정 계층이 이전 계층의 복잡한 특성을 연산하는지에 대해 이해하게 되었다. 하지만 이는 아직도 약간은 추상적일 수 있는데, 이는 말했다시피 추상화된 레벨이기 때문이다. 따라서 이어지는 두 강의에서, 좀더 자세한 예시를 통해 어떻게 신경망이 입력 비선형 함수를 연산하는지를 알아보고, 이를 통해 신경망에서 얻을 수 있는 복잡한 비선형 가설에 대해 이해해볼 수 있기를 바란다.

No.	Title
Week 4-5	Examples and Intuitions I

00:00

이번 강의와 다음 강의에서는 신경망이 입력 컴퓨터 비선형 함수가 어떻게 활용되는지를 알 수 있는 자세한 예시를 살펴보자. 그리고 이런 과정을 통해 왜 신경망이 복잡한 비선형 가설을 학습하는데 어떻게 활용될 수 있는지 배워보도록 하자. 다음 예시에는 특성 x_1 과 x_2 가 있는데 이는 2진이므로, 0 또는 1이다. 따라서 x_1 과 x_2 는 하나 아니면 두 개의 값만 가질 수 있다. 이

예시에서는 positive 예시 두 개, negative 예시 두 개만 표시해놨다. 이것은 좀 더 복잡한 학습 문제를 간소화한 것이라고 생각하면 되는데, 원래는 여기 오른쪽 위와, 왼쪽 아래에는 더 많은 positive 예시, 그리고 더 많은 negative 예시가 동그라미로 표시되어 있다. 우리는 이를 통해 비선형 경계 분배를 학습할 것인데 이는 positive와 negative 예시를 나누는데 필요한 개념이다.

00:53

신경망이 이를 어떻게 처리하는지를 살펴보자. 이 예시와 변수를 활용하기보다는 왼쪽에 있는 좀 더 간편한 예시를 통해 알아보겠다. 이 예시는 $y = x_1 \text{ XOR } x_2$ 를 연산하고 있다. 이는 또한 $y = x_1 \text{ XNOR } x_2$ 함수라고도 할 수 있는데 여기서 XNOR는 NOT ($x_1 \text{ XOR } x_2$) 라는 의미다. 따라서, $x_1 \text{ XOR } x_2$ 는 1이 되는 경우에만 참이 된다. 이러한 특정한 예시는 우리가 XNOR 예시를 대신 사용하면 조금 더 잘 작동하는 것을 알 수 있다. 이 둘은 물론 같다. 이는 x_1 또는 x_2 아니라는 뜻이므로, 둘 다 참이거나 둘 다 거짓은 positive 예시를 얻게 될 것이다. 둘 다 $y = 1$ 이다. 만약 둘 중 하나가 참이면 $y = 0$ 이 된다. 이제 신경망에 이와 같은 학습 세트를 적용할 수 있는지 알아보자.

01:59

XNOR 예시에 적합한 네트워크를 만들기 위해서는, 먼저 AND function에 맞는 네트워크를 보여주는 간단한 예시로 시작해보자. 예를 들어, 입력 x_1, x_2 가 있고 이것은 2진이므로 0 또는 1이 된다. 우리의 target label이 $y = x_1 \text{ AND } x_2$ 라고 하자. 이것은 logical AND다. 이 logical AND 함수를 연산하기 위해 One-unit 네트워크를 얻을 수 있을까? 그렇기 위해서는 bias unit을 추가하겠다. 이는 +1 유닛이다.

02:45

이제 이 네트워크의 가중치와 매개변수에 값 몇 개를 나눠줄 것이다. 이 도표에는 매개변수 -30, 그리고 여기는 +20, +20을 적었다. 이는 -30값을 x_0 와 연결된 값에 배분하고, 여기 +1은 여기 유닛에 들어가게 될 것이고, 그리고 +20의 매개변수 값은 x_1 과 곱해질 거고, 여기 +20값은 x_2 과 곱해진다는 것을 의미한다. 따라서, $h(x) = g(-30 + 20x_1 + 20x_2)$ 와 같다. 이처럼 때로는 이러한 weights, 매개 변수를 이렇게 적어주는 것이 편리하다. 여기서 -30은 실제로 $\theta(1)10$ 이다. 이것은 $\theta(1)11$, 이것은 $\theta(1)12$ 가 된다. 하지만 이런 것 보다는 이 네트워크에서 이러한 매개 변수와 연관해서 보는 것이 더 쉽다.

04:01

이 작은 하나의 뉴런 네트워크가 어떻게 연산되는지 살펴보자. 다시 한번 말하지만, 시그모이드 활성화 함수인 $g(z)$ 는 이와 같다. 0에서부터 시작해서 천천히 상승하다가 0.5를 지나 1을 향해 접근한다. 표지를 하기 위해서, 가로축에서 $z = 4.6$ 이라면, 시그모이드 함수는 0.99가 된다. 이는 1과 매우 가까운 수인데, 대칭을 이루고 있는 것이다. 만약 이 값이 -4.6이라면, 시그모이드 함수는 0.01 이렇게 0과 매우 가깝게 된다.

04:39

이번에는 x_1, x_2 에 가능한 4개의 입력 값을 살펴보고, 이런 사례에서 가설이 어떠한 출력을 할지 살펴보자. 만약 x_1, x_2 가 모두 0이라고 하자. 여기를 보면, x_1, x_2 가 모두 0일 경우, 가설 g 는 -30이 된다. 이는 이 그래프 원점에서 좌측으로 매우 멀어질 것이기 때문에 0과 가깝게 된다. $x_1=0, x_2=1$ 일 경우에는 여기 이 공식은 시그마 함수 $g(-10)$ 을 이 또한 좌표상 좌측으로 매우 멀어진 지점이기 때문에 0과 매우 가까워지게 된다. 이 값 또한 $g(-10)$ 이 되는데 $x_1=1, x_2=0$ 일 때, 여기 $-30 + 20$ 하면, -10이다. 마지막으로, $x_1 = 1, x_2 = 1$ 이면 $g(-30+20+20)$ 이다. 그래서 이는 $g(+10)$ 이면서 1과 매우 가까워진다.

5:39

여기 숫자 열을 보면, 이는 논리 AND 함수 과 같다. 그러니까 $h(x) \approx x_1 \text{ AND } x_2$ 이다. 다시 말해, 이는 x_1, x_2 둘 다 1일 경우에만 1을 출력한다는 것이다. 이와 같은 작은 진리표를 적으므로 우리는 신경망이 연산하는 논리 함수가 무엇인지에 대해 배워봤다.

Q) x_1, x_2 가 0,1 두 값만 가진다고 가정하자. 아래와 같은 네트워크는 논리 함수에서 어떻게 연산되는가? (힌트 : 아까 강의에서 설명한 진리표를 그려봐라.)

정답 3번

이 네트워크는 OR function을 연산한다. 먼저 이 과정을 보여주겠다. 가정은 $g(-10 + 20x_1 + 20x_2)$ 이고, 이 값을 대입하면 된다. $g(-10) \approx 0, g(10) \approx 1$, 그리고 이건 $\approx 1, \approx 1$ 이 된다. 그리고 여기 이 네 숫자는 논리 OR .함수다.

06:49

이제 하나의 뉴런이 신경망에서 AND 또는 OR 함수 등이 연산하는데 어떻게 활용되는지 이해할 수 있을 것이다. 다음 번 강의에서는 이러한 예시에서 조금 발전해서, 더 복잡한 예시를 다뤄볼 예정이다. 단위의 여러 계층으로 구성된 신경망이 XOR 함수와 XNOR 함수와 같은 더 복잡한 함수를 어떻게 계산하는지 살펴볼 예정이다.

No.	Title
Week 4-6	Examples and Intuitions II

00:00

이번 강의에서는 신경망이 복잡한 비선형 가설을 어떻게 연산하는지 그 예시를 더 살펴보도록 하겠다. 지난번 강의에서 신경망이 x_1, x_2 가 2진인, 즉, 0과 1 값을 가질 때, $x_1 \text{ AND } x_2, x_1 \text{ OR } x_2$ 를 연산하는데 활용될 수 있다는 것을 알아봤다. 또한 네트워크에서 NOT x_1 연산과 같은

부정명제를 연산할 수도 있다. 이 네트워크와 관련해서 몇 가지 적어보도록 하겠다. 이 경우에 input feature 는 x_1 하나이고, bias unit 은 +1이다. 만약에 여기에 weights + 10, -20을 적용하면, 가설은 $h_\theta(x) = g(10 - 20x_1)$ 라고 연산한다. 따라서 $x_1 = 0$ 일 때, 이 가설은 연산하면, $g(10) \approx 1$ 이 된다. 또한, $x_1 = 1$ 일 때, $g(-10) \approx 0$ 이 성립한다. 여기 이 값을 보면, 이것은 not x_1 연산이라는 것을 알 수 있다.

01:15

따라서 부정명제를 포함하기 위해서는 가장 일반적인 방법은 부정하려고 하는 변수 앞에 큰 negative weight를 놓는 것이다. 여기 $-20 * x_1$ 이런 식으로 x_1 을 부정하면 된다. 이렇게 예시를 통해 더 쉽게 이해할 수 있기를 바란다. (NOT x_1) AND (NOT x_2) 과 같은 연산하려고 한다면, 큰 negative weight를 x_1 과 x_2 앞에다가 붙이면 된다. 하지만 이는 실현가능 해야 한다. 따라서 하나의 출력 단위를 가진 신경망을 가지고 이것 또한 연산할 수 있다. 이 논리 함수 NOT x_1 AND NOT x_2 은 $x_1 = x_2 = 0$ 인 경우에만 1과 같게 된다. 이것이 논리 함수이기 때문에 NOT x_1 의 의미는 $x_1 = 0$ 이고, NOT x_2 는 $x_2 = 0$. 따라서 이 논리 함수는 은 $x_1 = x_2 = 0$ 인 경우에만 1과 같게 된다는 것이다. 이러한 논리 함수를 연산하기 위해 이와 같은 작은 신경망을 어떻게 만드는 지에 대해 이해할 수 있었기를 바란다.

Q) x_1, x_2 가 2진 (0또는 1)이라고 가정하자. 다음 중 (NOT x_1) AND (NOT x_2) 논리 함수를 연산하는 네트워크는?

(정답 1번)

02:33

이번에는 세가지 연산을 하나로 합쳐보자. x_1 AND x_2 연산망, NOT x_1 AND NOT x_2 연산망 마지막으로 x_1 OR x_2 연산망이 있다. 이 세 개의 연산망을 하나로 묶어서 x_1 XNOR x_2 를 연산해보도록 하자. 예전에 언급했듯이, 여기 x_1, x_2 가 있고, 우리가 연산하려고 하는 이 함수의 negative 예시는 여기와 여기, 그리고 positive 예시는 여기와 여기에 있다. 따라서, 이 함수는 positive와 negative 예시를 나누기 위해 비선형 결정 경계를 필요로 한다.

03:12

먼저 네트워크를 그려보자. 입력값 +1, x_1, x_2 를 가지고 여기 첫 번째 은닉 계층을 만들 것이다. 이것을 $a(2)1$ 이라고 부를 것이다, 이는 첫 번째 은닉 계층이기 때문이다. 그리고 빨간색 네트워크에서 weight 를 가져올 것이다. 그러면 -30, 20, 20이 된다. 그 다음으로는 두 번째 은닉 계층 $a(2)2$ 가 있다. 이는 layer 2의 두 번째 은닉 계층이다. 이제 가운데 있는 청록색 네트워크에서 weight 10, -20, -20를 가져온다. 그리고 진리표 값을 구해보자. 빨강 네트워크는 x_1, x_2 를 연산하는 것이기 때문에, x_1 과 x_2 의 값에 따라 약 0 0 0 1 이 나올 것이다. $a(2)2$ 의 경우, 청록색 네트워크에서는 우리는 NOT x_1 AND NOT x_2 연산에서 출력이 x_1 과 x_2 의 네 개의 값이 1 0 0 0 이라는 것을 알고 있다.

04:18

이제 출력 노드를 만들어, 이는 $a(3)1$ 이자, $h\theta(x)$ 이 된다. 이제 이전에 연산한 네트워크를 여기로 가져와보자. 여기 +1 bias unit 도 필요하니 그려주겠다. 먼저 초록색 네트워크에서 weights를 가져오겠다. 이는 -10, 20, 20 이고 이는 OR function을 연산한다는 것을 우리는 이미 알고 있다. 이제 진리표를 채워 넣어 보자. 첫 번째 칸은 0 또는 1이기 때문에 1인데, 0 또는 0 은 0, 0 또는 0 은 0, 그리고 1 또는 0은 1이 된다. $h\theta(x) = 1$ 일 때, x_1, x_2 는 둘 다 0 이거나 1이었다. 따라서 $h\theta(x)$ 는 이 두 곳에서 정확히 1을 출력하고, 그 반대에서는 0을 출력한다.

05:19

따라서 이 신경망은 입력 계층과 은닉 계층 하나 그리고 출력 계층 하나를 가지고 있고, 우리는 이제 XNOR function을 연산하는 비선형 결정 경계를 얻게 되었다. 다시 한번 살펴보면, 입력 계층은, 여기 이렇게 있고, 여기 은닉 계층은 약간 좀 더 복잡한 입력 함수를 연산했고, 바로 여기 있다. 이는 약간 더 복잡한 함수라고 할 수 있다. 그리고 여기 계층 하나를 더 추가해서 좀 더 복잡한 비선형 함수를 얻게 된다. 이는 왜 신경망이 꽤 복잡한 함수도 연산할 수 있는지에 대해 알아보는 과정이었다. 계층이 여러 개 있을 때, 두 번째 계층의 입력은 상대적으로 간단한 함수를 가지게 된다. 하지만 세 번째 계층은 좀 더 복잡한 함수를 완성하기 위해 만들어지고, 그러면 그 다음 계층 또한 더 복잡한 함수를 연산할 수 있게 된다.

06:10

이번 강의를 정리하기 위해, 신경망 응용의 재미있는 예시를 보여주려고 한다. 이를 통해 더 복잡한 함수를 연산하는 계층에 대해 이해하는데 도움이 되기를 바란다. 동료 Yann LeCun이 보내준 영상을 하나 보여주려고 한다. Yann은 뉴욕대 교수인데, 뉴욕대와 그는 신경망 연구에 선각자라고 할 수 있고 지금은 이 분야에서 전설적인 인물이다. 그의 아이디어는 전 세계에서 제품과 응용프로그램에서 사용되고 있다.

06:41

이제 그의 초창기 작업 영상을 보여줄 것인데, 그가 신경망을 가지고 손글씨를 인식하려고 하는 것인데, 손으로 쓴 숫자 인식이다. 이 강좌 초반부에, 신경망에서 가장 빠른 성공 중 하나가, 미연방우체국에서 활용할 우편 번호를 읽어내는 것이었다. 이 영상은 이를 시도하는 것이고, 이는 그 문제를 다루기 위한 알고리즘 중 하나다. 이 영상에서 이 부분은 입력 부분이 되는데, 이는 네트워크에 보여지는 숫자가 이렇다는 것을 보여준다. 왼쪽 그림은 네트워크의 첫 번째 은닉 계층에 의해 연산된 특징을 시각화 한 것이다. 따라서 네트워크의 첫 번째 은닉 계층에서 이 시각화 과정은 다른 특징들을 보여준다. 테두리와 라인 등이 보인다. 그 옆은 다음 은닉 계층을 시각화 한 것이다. 아까보다 분별하기도 이해하기도 더 어려워졌다. 은닉 계층이 더 진행될 수록, 시각화를 하면 이는 더 혼란스럽다. 이게 어떻게 진행되고 있는 건지 첫 번째 은닉 계층보다 보기 힘들 것이다. 하지만 여기 있는 모든 학습된 특징은 상위 계층으로 입력된다. 그리고 여기에

최종 결론이 나와있다. 이는 손으로 쓴 숫자의 최종 예상 값을 신경망이 생각해 낸 것이 보여지는 것이다. 이제 영상을 보도록 하자.

09:49

영상을 재미있게 봤기를 바라고, 꽤 복잡한 함수 신경망도 학습할 수 있다는 것을 알려주고 싶었다. 이렇게 입력 이미지를 얻게 되면, 입력하여 그 화소를 가지고 첫 번째 은닉 계층에서 특징 세트를 연산하다. 그러면 다음 은닉 계층은 좀 더 복잡한 특징을 연산하고, 반복된다. 이러한 특징은 로지스틱 classifiers의 첫 번째 계층에서 네트워크가 보고 있는 숫자 없이 정확한 예측을 하기 위해 활용된다.

No.	Title
Week 4-7	Multiclass Classification

0:00

이번 강의에서는 신경망을 2개 이상의 카테고리를 가진 것을 구별하는 다중분류에서 어떻게 활용하는지 알아보기로 한다. 지난 번 강의 끝 무렵에 손글씨 숫자 인식 문제에 관한 영상을 봤는데, 이는 다중 분류 문제라고 할 수 있다. 왜냐면 0부터 9까지 숫자를 인식하는 가능한 카테고리가 10개가 있기 때문이다. 이에 대해 좀 더 자세히 알아보기로 하자. 신경망에서 다중 분류를 하는 방법은 one vs all 방법의 확장이다. 컴퓨터 비전 예시가 있는데, 여기 원래 예시에 있는 것을 인식하는 것 대신에, 여기 있는 물체의 네 개의 카테고리를 인식하여 이미지가 주어졌을 때, 이를 보행자, 자동차, 오토바이, 트럭인지 결정하고 싶다. 그렇다면 네 개의 출력 단위를 가진 신경망을 만들어서 그 신경망이 4개의 숫자로 된 벡터를 출력하게 하면 된다.

1:09

따라서 출력은 4개의 숫자를 지닌 벡터가 될 것이다. 이제 분류를 시작하기 위해 첫 번째 출력을 얻도록 하자. 먼저 보행자 이미지인가? 맞다 또는 아니다. 두 번째 유닛 분류는 이것이 자동차의 이미지인가? 맞다 또는 아니다. 이 유닛 분류는 오토바이 이미지인가? 맞다 또는 아니다. 그리고 이 유닛 분류는 트럭 이미지인가? 맞다 또는 아니다. 따라서, 보행자 이미지가 있으면, 네트워크가 1,0,0,0을 출력하고, 자동차는 0, 1, 0, 0 그리고 오토바이는 0,0,1,0 등을 얻기를 원할 것이다.

1:50

지금까지는 로지스틱 회귀를 설명하면서 이야기했던 one vs all 방법을 설명했고, 여기 이제 네 개의 로지스틱 회귀 classifiers를 얻게 되었는데, 우리가 원하는 바는 이들이 4 클래스 중 하나를 구별해서 인식하는 것이다. 이제 다시 슬라이드를 정리해보면, 네 개의 출력 단위를 지닌 신경망이 있고, 이는 우리가 서로 다른 이미지를 가지고 있을 때 원하는 $h_{\theta}(x)$ 값이다. 다음과

같이 여기에 있는 학습 세트를 표현하도록 하겠다.

2:24

따라서 보행자, 자동차, 오토바이, 트럭과 같이 다른 이미지가 있는 학습 세트가 주어질 때, 이번 예시에서는, 예전에는 $y \in \{1, 2, 3, 4\}$ 과 같이 썼었는데, 이번에는 y 를 이와 같이 표현하는 대신에 이처럼 표현할 것이다. $y(i)$ 는 대응하는 이미지 $x(i)$ 가 어떤 것이냐에 따라서 $[1, 0, 0, 0]$, $[0, 1, 0, 0]$, $[0, 0, 1, 0]$, $[0, 0, 0, 1]$ 로 표현될 것이다. 따라서 학습 예시는 $(x(i), y(i))$ 가 될 것인데, $x(i)$ 는 이미지가 될 것이고, $y(i)$ 는 이 넷 중 하나의 벡터가 될 것이다.

3:10

그리고 이 신경망이 값을 출력해낼 것이다. 따라서 $h\theta(x(i)) \approx y(i)$ 가 될 것이고. $h\theta(x(i))$ 와 $y(i)$ 는 4개의 클래스가 주어졌을 때, 4차원 벡터의 형태로 예시가 된다. 이것이 바로 신경망을 통해 다중 분류 하는 과정이다. 이번 강의를 통해 가설 표현에 있어서 신경망 표현 방법을 마무리하게 되었다. 다음 강의부터는 학습 데이터를 가져오는 방법과 신경망의 매개변수를 자동으로 학습하게 되는지 알아보도록 하자.

Week 5

No.	Title
Week 5-1	Cost Function

00'00''

신경망 네트워크(Neural network)는 현존하는 가장 뛰어난 학습 알고리즘이라고 할 수 있다. 이번과 앞으로의 몇 강의에서 주어진 트레이닝 데이터를 활용하여 적합한 변수를 찾아내는 신경망 네트워크의 학습 알고리즘에 대해 학습할 것이다. 다른 학습 알고리즘과 같이 우리는 적합한 네트워크 변수를 찾기 위한 비용함수에 대해 알아보며 시작하도록 하겠다. 분류 문제에 신경망 네트워크의 적용에 대해서 초점을 맞출 것이다. 그러니 우리가 왼쪽에 보이는 것과 같은

네트워크가 있고, 이렇게 $X \mid Y \mid$ 짝을 이룬 M 트레이닝 예가 있다고 하자. 대문자 L 을 사용해서 이 네트워크의 전체 레이어(layer)의 개수를 표기하겠다.

00'45"

여기 왼쪽의 네트워크는 대문자 $L=4$ 로 표현할 수 있겠다. S_l 은 레이어 l (엘)의 유닛(unit) 개수, 즉 신경(neuron)의 개수를 나타낸다. 여기 네트워크의 l 에서 바이어스 유닛(bias unit)의 개수는 제외시킨다. 예를 들어, 여기 왼쪽 네트워크는 $S_1=3, S_2=5$ 을 가지고, $L=4$ 이기 때문에 $S_4=S_l=4$ 일 것이다. 여기 예시의 출력값 레이어(output layer)는 총 4 개의 유닛을 가졌다고 할 수 있다. 우리는 2 가지의 분류 문제를 고려할 것이다. 첫 번째는 이진법 분류인데, $y=0$ 또는 1 이다. 이진법에서는 1 개의 출력값 유닛(output unit)을 가지는데, 여기 위의 예시의 신경 네트워크는 4 개 출력 값 유닛을 가지고 있지만 우리가 이진법 분류를 적용한다면 $h(x)$ 를 계산하는 하나의 출력값 유닛만 가질 수 있다.

01'40"

그리고 이 신경망 네트워크의 출력값 $h(x)$ 는 실수(real number)이다. 그리고 이 경우에 출력 값 유닛의 숫자인 S_L 에서 L 은 마지막 레이어의 지수이다. 이 네트워크에서 우리가 가진 레이어의 갯수이니, 이 output layer 의 유닛갯수는 1 이 될 것이다. 이것을 조금 더 간소화 할 것인데, $K=1$ 이라고 지정할 것이다. 그러면 여러분은 K 가 출력 값 레이어 유닛의 갯수라고 생각하면 된다. 두 번째로 우리가 다룰 분류 문제는 멀티 클래스 분류 문제로, 이것은 K 개의 클래스로 뚜렷이 구분될 것이다. 우리의 앞선 예에서 4 가지의 분류가 있는 경우 y 라고 표기했는데, 이 경우에는 대문자 K 출력값 유닛과 가정함수 혹은 k 차원의 출력 값 벡터가 있다.

02'35"

이 경우에는 K 개의 output unit 을 가지며, $S_l=K$ 가 될 것이다. 보통 K 는 3 보다 크거나 같은데, 만약 우리가 두 가지 요인만 가지고 있다면, 한 개 대 모든 method 을 사용할 필요가 없다. K 가 V 차원보다 크거나 같을 때만 one verse all method 을 사용하기 때문에 오직 두 클래스만 있는 경우 우리는 한 개의 upper unit 만 사용할 것이다. 그러면 우리의 신경 네트워크의 비용함수에 대해서 정의해 보자. 여기서 사용하는 비용 함수는 우리가 로지스틱 회귀에서 사용했던 것을 일반화 한 것이다. 로지스틱 회귀를 위해서 우리는 J 세타 비용함수를 최소화했고, 그것은 이 비용함수에서 $-1/m$ 을 곱하고 여기 정규화 부분을 더한 것이었다. 이것은 j 가 1 부터 n 까지일 때였다. 왜냐하면 우리는 bias 항인 세타 0 를 정규화하지 않았기 때문이다.

03'32"

신경 네트워크의 경우에는 이 비용함수는 이것의 일반화된 버전일 것이다. compression output unit 하나만을 갖는 대신, K 개를 갖는 것이다. 이것이 우리의 비용함수이다. 새 네트워크는 벡터

R^k 를 나타내고, 이진법 분류 문제였다면 아마도 여기서 R 은 1일 것이다. $h(x)_i$ 를 써서 i 번째의 출력 값을 표현했다. $h(x)$ 는 k 차 벡터이고 여기 아래글자 i 는 벡터의 i 번째 요소를 골라내고 이것은 신경 네트워크의 출력 값이다. 비용함수 J 세타는 이렇게 보일 것이다. 우리가 로지스틱 회귀에서 했던 부분과 $K=1$ 에서 K 인 부분만 제외하고는 같은 식의 합에 $-1/M$ 을 한 값이다. 이 식은 K 출력 값의 합과 같다. 그러니 우리가 4 가지 출력 값 유닛이 있으면, 신경 네트워크의 마지막 레이어가 4 개의 출력 값 유닛이 있으면 이것은 $k=1\sim 4$ 까지일 때의 로지스틱 회귀 알고리즘의 비용함수의 합과 같다. 하지만 이 각각 4 개의 비용함수를 더해야 한다.

04'49"

그리고 여러분은 특별히 이것이 y_k h_k 에 적용된다는 것을 알 수 있을 것이다. 왜냐하면 우리는 저 위쪽 유닛의 K 를 y_k , 즉 벡터들 중 하나가 말하는 비용과 비교하였기 때문이다. 그리고 마지막으로 두 번째 부분인 여기는 정규화(regularization)부분인데, 이것은 우리가 로지스틱 회귀에 사용했던 것과 비슷하다. 이것은 굉장히 복잡해 보이지만 이것이 하는 것은 세타 j_i 를 더하는 것이다. 여기서 i 는 i j 와 i 의 모든 값을 의미한다. 이것을 제외하고는 로지스틱 회귀에서 사용했던 바이어스 값을 더하지 않는다. i 가 0 일때의 것들을 더하지 않는다는 것이다. 이것은 신경의 활성화 정도를 계산할 때 이와 유사한 계산을 한다. 세타 i_0 플러스 세타 i_1 곱하기 1 플러스 등등 계속 된다.

05'51"

여기 2를 대입할 것이고, 여기 처음 자리에 둔다. 그리고 저 곳에 0의 값을 넣으면 모두 0을 곱하게 되는 것이다. 이는 bias unit의 일종인데, 우리가 로지스틱 회귀에서 사용했던 것처럼 이진법적이다. 이 정규화 항은 더하지 않을 것인데 이는 우리가 정규화해서 그 값을 0으로 만들고자 함이 아니기 때문이다. 이것은 가능한 방법이기는 하다. i 가 0에서 S_i 에 이르기까지 더하더라도 결과는 그렇게 다르지 않을 것이다. 왜냐하면 이것은 꽤 일반적이기 때문이다. 그러므로 이것이 신경 네트워크에서 사용되는 비용함수이다. 다음 영상에서는 비용함수를 최적화하기 위한 알고리즘에 대해서 논할 것이다.

No.	Title
Week 5-2	Backpropagation algorithm

00'00"

이전 영상에서는 신경 네트워크의 비용함수에 대해서 다루었다. 이번 강의에서는 비용함수를 최소화하기 위한 알고리즘에 대해서 얘기할 것이다. 특별히 우리는 back propagation(역전파) 알고리즘에 대해 논해보도록 하겠다. 이것은 지난 영상에서 적었던 비용함수이다. 우리가 할 것은 변수 세타를 찾아서 J 세타 함수를 최소화 할 것이다. Gradient descent 또는 다른 고급 최적화

알고리즘 중 하나를 사용하기 위해서 우리가 해야하는 것은 입력 변수 θ 를 받아, J_θ 함수와 이 편미분계수(partial derivative term)을 계산하는 코드를 쓰는 것이다. 신경 네트워크의 변수, 세타 i j 는 실수(real number)이고 이것들이 우리가 계산해야 할 편미분계수 항이다.

00'49''

비용함수 J 세타를 계산하기 위해서 우리는 여기 위에 있는 공식을 사용할 것이고, 이 강의에서 할 것은 우리가 이 편미분계수 항 계산법을 배우는 것이다. 우리가 하나의 트레이닝 예만을 갖고 있을 경우에 대해 얘기하며 시작해보기로 하자. 여러분이 (x,y) 하나만을 가진 트레이닝 예가 있다고 하자. x_1y_1 이 아니라 이렇게 적을 것이다. xy 라고 적고 계산의 시퀀스를 이용해 이렇게 살펴보도록 하자. 첫 번째로 우리는 forward propagation 을 적용하여 가정함수가 이러한 것을 뽑아내는지 계산할 것이다. 좀 더 자세히 말하면 a_1 이 저것의 입력 값이었던 첫 번째 레이어의 활성화 값인 것이다.

01'42''

그래서 이것을 $a^{(1)} = x$ 라고 지정하면, $z^{(2)} = \theta^{(1)}a^{(1)}$ 로 계산하고 $a^{(2)}=g$, sigmoid 활성함수가 $z^{(2)}$ 에 적용되고 첫 중간 레이어의 활성화가 될 것이고, 또한 bias term 을 더할 것이다. 두 번째로 우리는 이 네 가지에 대해서, 그리고 $a_3, 4$ 를 계산하기 위한 propagation 에 대해 두 가지 단계를 더 적용할 것이다. 이는 여기 가정함수 $h(x)$ 와 같다. 이것이 벡터화된 forward propagation 의 구현이고, 신경망 네트워크에서 모든 뉴런의 값을 활성화시키는 계산을 가능하게 한다. 다음으로 미분계수들을 계산하기 위하여 propagation 이라고 불리는 알고리즘을 사용할 것이다.

이 알고리즘은 노드별 $\delta_j^{(l)}$ 를 계산할 것이며, 이것은 레이어 l 에 있는 노드 j 의 오차(error)를 의미한다. $a_j^{(l)}$ 는 레이어 l 의 j 유닛을 활성화시키는 것을 상기해보면, 이 δ 항은 neural duo 의 활성화 오류를 보여준다고 할 수 있다.

03'04''

그러므로 우리가 이 노드를 어떻게 활성화하길 원하는지와는 조금 다를 수도 있겠다. 좀 더 자세히 말하면 여기 오른쪽의 신경 네트워크에는 4 레이어가 존재한다. 그래서 대문자 L 은 4 이다. 각각의 출력 값 유닛에서 우리는 이 델타 항을 계산할 것이다. 네 번째 레이어에서 유닛 j 의 델타는 그 유닛의 활성화값에서 우리 트레이닝 예시에서 0 의 실제값이었던 것을 뺀 것과 같다. ($\delta_j^{(4)} = a_j^{(4)} - y_j$). 그래서 이 $a_j^{(4)}$ 은 $(h_\theta(x))_j$ 로 표기할 수 있다. 이 델타 항은 가정함수의 출력 값이 있을 때 y 의 값은 우리 트레이닝 셋의 y 의 값과 y_j 에서의 j 가 y 의 벡터 값일 때의 차이점과 같다. 그리고 델타 a 와 벡터 y 를 고려할 때, 이런 벡터화한 시행을 생각해 낼 수 있을 것이고, 델타 4 는 a_4-y 라는 셋을 가지게 된다. 여기서 델타 4 a_4 와 y , 각각은 벡터이고 이 네트워크의 출력 값 유닛의 숫자와 동일한 차원을 갖고 있다.

04'26''

우리는 델타 4에 대해서 계산을 했다. 다음은 앞의 레이어에서의 델타 항을 계산할 것이다. 이것은 델타 3가 세타 3t 곱하기 델타 4 일때의 델타 3 값을 위한 공식이다. 이것은 y의 multiplication operation이고 우리는 MATLAB을 통해 이것을 알고 있다. 그러니 델타 3t 델타 4, 이것은 벡터인데, g 프라임 z3 이것 또한 벡터이다. 그리고 이 점 곱하기는 두 벡터 사이의 y의 multiplication이다. g 프라임 z3은 활성 함수 g의 미분계수인데, z3이 입력 되었을 때의 경우이다. 여러분이 미적분학을 알고 있다면 여러분이 직접 시행해서 이와 같은 답을 알아내어 간소화할 수 있다. 하지만 여기서는 무엇을 의미하는 지만 설명하겠다. 여러분이 계산할 것은 이 g 프라임인데, 이것은 a3 곱하기 1 마이너스 a3이고, a3은 활성의 벡터이다. 1은 1의 벡터이고 a3은 레이어의 활성 값의 벡터이다.

05'39"

이번에는 여러분은 델타 2를 계산할 것인데 이것은 비슷한 공식을 사용할 것이다. 지금은 그냥 a2이기 때문에 여기에 증명할 것이지만, 여러분이 미적분을 알고 있다면 이 표현이 수학적으로 맞고, g 프라임으로 기입한 g 함수의 미분계수가 맞다는 것을 증명할 수 있을 것이다. 그리고 이것이 전부이고, 델타 1은 존재하지 않는다. 왜냐하면 첫 번째 레이어는 입력값 레이어와 일치하였고, 우리의 트레이닝 셋에 있었던 유일한 특징이기 때문에 아무런 오류도 존재하지 않는다. 우리가 이 값들을 변경하는 것을 원하지 않는다는 것이 아니다. 그리고 우리는 레이어 2,3에 대해서만 델타 항을 갖고 있다. back propagation은 출력값 레이어의 델타 항을 계산해서 나오는 것이고, 그러면 우리는 다른 단계로 돌아가서 델타 2를 계산해 낼 것이다. 그러니까 오류를 뒤쪽으로 전파시켜서 계산하는 것이기에 back complication이라고도 불린다.

06'52"

마지막으로 derivation은 놀라울 만큼 복잡하지만 이 계산의 조금씩 해 나간다면 조금 복잡한 수학적인 증거를 알아낼 수 있을 것이다. 만약 규칙화(regularization)를 무시하면 여러분이 얻고자 하는 편미분계수 항은 정확히 활성화와 델타 항으로 얻어진다. 이것은 ignoring 람다나 또 다른 말로는 regularization 항 람다가 0이 된다는 것이다. 이 용어의 세부사항에 대해서는 나중에 손보도록 하겠지만, 이 back propagation 시행과 델타 항 계산은 여러분의 변수의 편미분계수 항들을 꽤 빠르게 계산해 준다. 이것은 세부 사항이 많다.

07'41"

이것을 모두 한번에 모아서 여러분의 변수의 미분계수를 계산할 back propagation을 시행하는 방법에 대해서 말해보자. 이 경우에는 우리는 큰 트레이닝 셋을 가지고 있을 것이고, 하나의 예만을 가지고 있지는 않을 것이다. 우리는 m 트레이닝셋이 있다고 가정해 보자. 여기 보이는 것처럼 첫 번째 할 일은 이 델타 Δ_{ij} 를 지정하는 것이다. 이렇게 삼각형 심볼로 만들자. 이것은 사실 그리스 알파벳 델타이다. 지난번 슬라이드에 있었던 기호는 소문자 델타였다. 그러나 이 삼각형은 대문자 델타인 것이다. 이것이 모든 Δ_{ij} 에 대해 0의 값을 가지게 할 것이다. 마침내 이 대문자 델타 Δ_{ij} 는 편미분계수, J 세타함수의 세타 J_{ij} 값에 대해 계산할 것이다. 우리가 두 번째에서 보는 것처럼 이 델타는 편미분계수를 계산하기 위해 어떤 것들을 서서히 더해가는

accumulator 의 역할을 할 것이다. 다음은 우리 트레이닝 셋을 반복할 것이다. $i=1$ 에서 m 일 때, 이 i 반복에 대해 x_i, y_i 로 알아볼 것이다.

09'02''

그러니 입력 값 레이어의 활성인 set a_1 에 대해서 x_i 와 같게 만들 것이다. 이것은 i 에 대한 입력 값을 의미한다. 그리고 forward propagation 을 사용하여 레이어 2, 3, 그리고 마지막 레이어 L 까지의 활성을 계산할 것이다. 다음은 우리는 출력 값 레벨 y_i 을 사용하여 저기의 출력 값의 델타 Δ_L 의 error 항을 계산할 것이다. 델타 Δ_L 은 가정함수의 출력 값에 타겟 레이블을 뺀 것이다. 그러면 우리는 back propagation 알고리즘을 사용하여 델타 Δ_L 마이너스 1, 마이너스 2 등을 시행할 것이다. 그리고 다시 한번 이것은 델타 1 인데 우리는 이 입력값 레이어의 오차항은 고려하지 않기 때문이다.

09'57''

그리고 마지막으로 우리는 이 대문자 델타 항을 사용해서 편미분계수 항을 측적할 것이다. 이것은 지난 줄에 써 둔 것이다. 그리고 이 표현을 보면 이것의 벡터화 또한 가능하다. 자세히 말하면 행렬 델타 Δ_{ij} 를 생각하면 델타 Δ_L 이 행렬이라면 이것은 델타 Δ_L 플러스 소문자 델타 Δ_L 플러스 1 곱하기 a_{lt} 로 업데이트 될 것이다. 이것이 벡터화된 시행이고 이것은 ij 값에 대해 자동 업데이트를 할 것이다.

10'39''

마지막으로 이 four-loop 을 없애고 난 후 이것에서 벗어나 이것들을 계산하라. 우리는 대문자 D 를 계산할 것이고 j 가 0 인경우와 0 이 아닌 두 가지의 경우에 대해서 계산할 것이다. 0 인 경우는 bias 항과 일치하니, 이때는 우리는 또 하나의 정규화 항이 사라진다. 마지막으로 공식적 증명이 꽤 어려워 보이는 가운데 여러분이 이 D 항을 계산했을 때 이것은 여러분의 비용함수의 편미분계수고 이것을 gradient descent 나 advanced authorisation 알고리즘에 활용할 수 있다.

Q) 여러분의 두 개의 트레이닝 예시, $(x^{(1)}, y^{(1)})$ 과 $(x^{(2)}, y^{(2)})$ 를 가지고 있다고 하자. 다음 중 어떤 것이 descent 를 계산하는 올바른 계산법인가? 정답: 4 번

11'28''

지금까지 back propagation algorithm 에 대해 알아보았다. 이는 신경 네트워크에서 비용함수의 미분계수를 계산하는 방법이다. 이것은 정말 복잡해 보이고 굉장히 많은 단계가 있는 것처럼 보인다. 하지만 지금까지 살펴본 부분을 잘 요약해서 여러분이 알고리즘에 대해 전체적으로 이해하여, 여러분이 back propagation 을 사용해서 신경 네트워크의 비용함수의 미분계수를 계산하고 싶을 때 무엇을 해야 하는지 숙지하길 바란다.

No.	Title
Week 5-3	Backpropagation intuition

00'00"

지난 영상에서 우리는 backpropagation 알고리즘을 다루었다. 많은 사람들이 처음 접해 보았고, 첫 인상은 이것이 아주 복잡해서 놀라웠다고 말한다. 또한 각각 다른 단계가 있는데 그것이 서로 잘 맞는지 알 수 없다고 한다. 그리고 이것이 이 복잡한 단계의 블랙박스라고 할 수 있다. 여러분이 backpropagation 에 대해 이렇게 생각한다면 괜찮다. 선형회귀나 로지스틱 회귀와 비교했을 때 backpropagation 은 아마도 수학적으로 깨끗한, 수학적으로 간단한 알고리즘은 아니다. 그리고 backpropagation 을 최근 몇 년 간 성공적으로 사용하였지만 오늘날에도 이것이 어떻게 굴러가는지 때때로는 잘 이해 되지 않는다. 이것을 연습해보는 여러분은 기계적으로 다른 단계에 가서 어떻게 backpropagation 하는지를 배우므로 이것을 잘 습득할 수 있을 것이다.

01'00"

이번 영상에서 여러분에게 backpropagation 의 기계적 단계에 대해 좀 더 소개하고자 하고, 이것이 무엇인지 여러분에게 최소한 이것이 합리적인 알고리즘이라는 확신을 심어주려고 한다. 이 영상을 보고 난 뒤에도 backpropagation 이 블랙박스 같이 느껴지고 복잡하고 이해가 되지 않더라도 괜찮다. 나 역시도 수년간 이것을 다뤄 왔지만 때때로 이해하기 어렵다. 하지만 이 영상이 여러분에게 도움이 되길 바란다. 좀 더 이해를 돋기 위해서 forward propagation 이 무엇을 하는지에 대해 살펴보자. 이것은 bias unit 을 세지 않는 신경 네트워크의 2 가지의 입력 값 유닛이고, 여기 두 개의 숨겨진 유닛이 다음 레이어에 있다.

01'55"

그리고 여기 하나의 출력 값 유닛이 있다. 이것은 2, 2, 2 로 세어질 것이다. 왜냐하면 맨 윗줄을 bias unit 은 계산되지 않기 때문이다. forward propagation 을 도식화하기 위해서 이 네트워크를 좀 더 다르게 그릴 것이다. 자세히 말하면 이 신경 네트워크를 이런 타원 노드와 함께 그려 안에 텍스트를 표기할 것이다. forward propagation 을 수행할 때 우리는 예 $(x^{(i)}, y^{(i)})$ 을 사용할 것이다. $x^{(i)}$ 는 입력 레이어에 표기될 것이다. 그래서 이것은 우리가 입력 레이어에 설정한 값 $x_2^{(i)}$ 와 같이 될 것이다. 우리가 forward propagation 을 여기 첫번째 숨겨진 레이어에 하면, 우리는 z_2^1, z_2^2 를 계산할 것이다.. 이것은 입력값 유닛의 입력값의 합으로 측정된다. 그리고 우리는 로지스틱

함수의 시그모이드를 적용하면, 이 시그모이드 활성 `funcion` 은 z 값에 적용된다. 이것이 활성 값이다. 이것은 $a(2) 1, a(2) 2$ 값을 준다. 그리고 $z(3)1$ 을 알기 위해 또 forward propagation 을 한다.

03'05"

로지스틱 함수의 시그모이드를 적용하고, 활성 함수는 $a(3) 1$ 이 된다. 유사한 방법으로 $z(4)1$ 은 $a(4)1$ 이 되고, 이것이 신경 네트워크의 마지막 출력 값이다. 여기 화살표를 좀 지워보자. 이 hidden unit 에 중점을 두고 이 계산이 무엇인지 보는데, 우리는 이 여기 이 가중치를 더해야 한다. 여기에 세타 2 1 0 라는 붉은색으로 표기한 가중치가 있다. 지수는 중요하지 않다. 여기 빨간색으로 표시하고 있는 것은 세타 (2) 1 1, 여기 노랑으로 표시한 것은 세타 (2) 1 2 이다. 그러니 우리가 이 값, $z(3)1$ 은 붉은색으로 표시한 것 곱하기 이 값이다. 그러니 이것은 세타 (2) 10 곱하기 1 인 것이다. 그리고 여기에 플러스 빨간색 표시한 것 곱하기 이 값이고, 이것은 세타(2) 11 곱하기 $a(2)1$ 이다.

04'26"

그리고 이 노랑색 부분 곱하기 이 값은, 그러니 플러스 세타(2)12 곱하기 $a(2)1$ 이다. 이것이 forward propagation 이다. 조금 후에 back propagation 이 비슷한 작업을 한다는 것을 배울 것이다. 하지만 대신 방금 전의 계산은 왼쪽에서 오른쪽으로 갔지만 이 back propagation 은 오른쪽에서 왼쪽으로 흐름이 움직일 것이다. 물론 이런 비슷한 계산을 사용한다. 향후 2 슬라이드에서 무엇을 의미하는지 알게 될 것이다. back propagation 의 이해를 돋기 위해 비용함수를 보도록 하자. 이 비용함수는 우리가 출력 값 유닛이 1 개일때의 경우이다. 우리가 1 개 이상의 출력 값 유닛이 있다면 여기 k 까지의 값을 다 더해야 한다. 하지만 1 개 뿐이라면 이것이 비용함수이다. 그리고 우리는 forward propagation 과 back propagation 을 여기에 시행할 것이다. $xi yi$ 에는 출력값 유닛이 1 개 존재한다. 그러니 yi 는 실수이다. 그리고 규칙화(regularization)은 n 은 무시하도록 하자. 그렇다면 람다는 0 이다. 그리고 이 마지막 이 부분은 사라진다.

05'48"

이 계산을 좀 더 깊이 들여다 보면 트레이닝 예와 관련 있는 비용 항을 발견할 것이다. 그것은 바로 예 $xi yi$ 와 관련 있는 비용이다. 그것은 이렇게 표현할 수 있을 것이다. 그러니 이렇게 적을 수 있다. 이 비용함수의 역할은 이 네모 부분과 비슷하다. 그러니 이 복잡한 표현을 보는 것 보단 여러분이 i 의 비용이 여기 보이는 신경 네트워크 출력 값과 실제 값의 차이의 제곱의 차이에 대해 생각해 보라. 로지스틱 회귀처럼 로그를 사용해서 조금 더 복잡한 수식을 사용하려 한다. 하지만 이것을 하는 이유는 비용함수가 제곱 오차 비용함수의 종류라고 생각해보라는 것이다. 그러니 이 $cost(i)$ 는 네트워크가 얼마나 잘 i 를 올바르게 측정하는지 알게 해 줄 것이다. 또, 실제 값과 얼마나 가까운지 등을 나타낼 것이다. 이제는 back propagation 의 역할을 살펴보자.

06'49"

back propagation 은 델타 δ_j 를 계산할 것이다. 이것을 i 번째 레이어에서 unit j 에서 얻은 활성 value 의 비용오차(error of cost) 라고 생각하자. 좀 더 정확히 말하자면 이것은 미적분을 할 수 있는 사람들을 위한 것이다. 델타 항이 의미하는 것은 z_i 와 j 를 반영한 partial 미분계수다. 이것들은 이 z 항을 복잡하게 하는 입력 값의 가중치를 반영한 것의 합이다. 비용함수의 이런 것들을 반영한 편미분계수라는 것이다. 그러니 자세히 말하면 이 비용함수는 레이블 y 와 신경 네트워크의 $h(x)$ 출력값 값의 함수라고 할 수 있다. 신경 네트워크에 대해 좀 더 깊이 들어가서 z_{ij} 값에 조금 변화를 주어 신경 네트워크에 출력 값을 달리 해보자. 그러면 그것은 비용함수의 변화를 가져올 것이고, 다시 한 번 여러분이 미적분을 잘하고 편미분계수에 대해서 잘 알고 있다면 이것의 비용함수를 이 항들과 함께 고려해 보라.

08'08''

이것은 우리가 신경 네트워크의 가중치를 얼마나 바꿀까의 문제인데, 이것은 계산에 있어 이 값들을 변화시킨다. 이것이 신경 네트워크의 마지막 출력 값을 변경하니까 전체 비용에도 영향을 미치는 것이다. 이 partial 미분계수가 빠지면 이것은 말이 되지 않는다. 나머지에 대해서는 걱정하지 않아도 된다. 우리는 partial 미분계수가 꼭 필요한 것은 아니다. 하지만 back propagation 이 무엇을 하는지 보고자 한다면 출력값 레이어에서 첫 번째 셋 델타(4)1 이 있는데 이는 우리가 다루었던 y_i 도 있다. 이것은 y_i 마이너스 $a(4)1$ 이라고 하고 있다. 그러니 이것은 오차이다. 이것은 실제 값 y 마이너스 예측 값인 것이다. 그러니 우리는 델타(4)1 을 계산할 것이다. 그리고 우리는 back propagation 에 대해서 다룰 것이다. 이것에 대해 설명하고 지난 레이어에서의 델타항을 계산할 것이다.

09'10''

우리는 델타(3)1 과 2 을 구할 것이다. 그리고 이것을 다시 돌려서 델타(2)1 과 2 를 계산할 것이다. back propagation 계산은 델타(2)2 를 계산하는 것과 같다. 하지만 반대로 다시 돌아와서 계산하는 것이다. 이것이 내가 말하는 것인데, 우리가 어떻게 델타(2)2 로 왔는지를 보자. 이것은 forward propagation 과 비슷하게 가중치에 대해서 설명하겠다. 이 가중치는 청록색(cyan)으로 그릴 것이다. 이 가중치가 세타(2)1 2 라고 하고, 여기는 빨간색으로 세타(2)2 2 라고 하자. 그러니 우리가 델타(2)2 를 계산하는 것을 보면 이러한 노트와 함께 계산하는 것을 보면, 이 값을 이 가중치로 곱하고 이 값을 저 가중치로 곱한 이 값에 더하는 것이다.

10'19''

그러니 정말 이것은 델타 값에 가중치를 곱한 값의 합이라고 볼 수 있다. 그리고 가중치는 이 edge strength 와 같다. 그러니 델타(2)2 는 세타(2)1 2 와 같은데 이것은 붉은색 곱하기 델타(3)1 과 같은 것이다. 그에 더해 이 빨간색으로 표시한 것은 세타(2)2 곱하기 델타(3)2 이다. 그러니 빨간색 곱하기 이 값, 플러스 붉은색 가중치 곱하기 이 값이다. 우리는 어떻게 델타값을 구할 수 있을까. 다른 예에서처럼 이 값을 보도록 하자. 저 값은 어떻게 구할까? 이것도 비슷한 프로세스이다. 내가 초록색으로 표기할 가중치가 델타(3)1 과 같다고 하자. 그리고 여태까지는 hidden unit 에 대해서만 델타 값을 표기하였다. bias unit 을 제외하고 말이다. back

propagation 알고리즘을 어떻게 정의하느냐에 따라, 혹은 어떻게 시행하느냐에 따라 bias unit의 값 계산 시행에도 영향을 미칠 것이다.

11'46''

bias unit은 플러스 1의 값을 낳는데, 이 값을 바꿀 수는 없다. 여러분의 back propagation에 의존해서 이런 델타 값을 계산할 수 있다. 우리는 없애버렸으니 사용하지 않을 것이다. 왜냐하면 이 미분계수를 계산하기 위한 계산의 부분이 되지는 않기 때문이다.

Q) 다음의 신경 네트워크를 보라. 빨간 값이 0 예 해당한다. Backdrop propagation 을 실행한다면 다음의 값에 대해 적절한 설명은? 정답: 4 번

이 모든 설명이 여전히 어렵다면 다음 강의와 연계해서 복습해보라. back propagation이 무엇인지 더 잘 알기 바란다. 아직 잘 모르겠다면 back propagation에 대해서 좀 더 자세히 설명하도록 하겠다. 이것을 시각화하고 무엇을 하는지 확실히 알려주는 것은 아주 힘들다. 설명하거나 시각화하기 힘들긴 하지만 많은 사람들이 사용하는 효율적인 학습 알고리즘이라는 것은 사실이다.

No.	Title
Week 5-4	Implementation note: unrolling parameters

00'00''

지난 강의에서 우리는 back propagation을 사용해서 비용함수의 미분계수를 구하는 방법에 대해 논했다. 이번 영상에서는 행렬에서 벡터로 변수의 값의 unroll을 시행하는 방법에 대해 배울 것이다. 이것은 우리가 고급 최적화 루틴을 사용하기 위해 필요한 단계이다. 좀 더 자세히 말하면 여러분이 변수 세타를 가지고 이 비용함수와 이 미분계수를 구한다고 해보자. 그러면 fminun을 사용해서 해결할 수 있지만 이것이 유일한 방법은 아니다. 또 다른 고급화된 알고리즘들이 존재한다.

00'40''

하지만 이것들은 이 비용함수의 입력 값, 그리고 세타의 기본값을 사용한다. 그리고 둘 다 이 루틴들은 세타와 세타의 기본값을 변수 벡터 R_n 이나 R_{n+1} 로 가정한다. 이것은 벡터고 이는 여러분의 비용 함수가 두번째 리턴 값을 가져올 것이라고 가정하고, 이 또한 R_n , R_{n+1} 로 하자. 이것 또한 벡터이다. 우리가 로지스틱 회귀를 사용할 때는 잘 작동되었지만 우리는 현재 신경 네트워크를 사용하고 있기에 이 변수들은 벡터가 아니다. 그 대신 이것은 행렬, 수많은 세타 1, 2, 3 같은 변수 행렬이다. 이것은 옥타브에서 세타 1, 2, 3으로 표기된다. 유사하게 이 gradient 항들의 리턴에 대해서 얘기할 것이다.

01'37"

지난 강의에서 gradient 행렬을 어떻게 계산하는지 알려주었다. 대문자 $D_1 D_2 D_3$ 는 옥타브에서 이렇게 표현되었다. 이번 강의에서는 여러분에게 이런 매트릭스를 사용해서 벡터로 unroll 하는 방법에 대해서 설명할 것이다. 그렇게 되면 그러면 여기 세타나 저기 gradient에 맞는 형식으로 변경될 것이다. 좀 더 자세히 말하면 이렇게 신경 네트워크에 입력값 레이어 1은 10, 숨겨진 레이어 (hidden layer)도 10이고 출력 값 레이어는 1개라고하자. 그러면 이렇게 보일 것이다. 이 경우의 행렬의 차원은 이렇게 적을 수 있을 것이다. 예를 들어 세타 1은 10×11 행렬인 것이다. 그러니 여러분은 이렇게 행렬을 벡터로 변경할 것이다.

02'40"

우리가 여기서 할 것은 세타 1,2,3과 이 코드를 적고 이 세 가지 세타 행렬의 모든 요소들을 가져올 것이다. 그래서 이것을 크고 긴 벡터에 펼칠 것이다. 이것은 세타 vec이고 두 번째 명령은 여러분의 D 행렬을 D_{vec} 으로 펼쳐 만들어 놓을 것이다. 마지막으로 벡터를 행렬로 다시 바꿔보자면, 세타 1은 110개의 요소가 있다. 왜냐하면 10×11 행렬이었기 때문이다. 이것을 다시 reshape 명령을 내려 세타 1으로 바꿔놓을 수 있다. 세타 3과 같은 경우도 이렇게 다시 바꿀 수 있다.

Q) D_1 이 10×6 행렬이고 D_2 가 1×11 행렬이고, D_{vec} 을 다음과 같이 놓는다. 이 중 D_{vec} 에서 D_2 를 다시 얻을 수 있는 방법은? 정답: 3번

03'50"

이것이 옥타브 데모 프로세스의 일부이다. 세타 1은 아까처럼 10×11 행렬이었고, 이것은 1로만 이루어져있다. 이것을 좀 더 쉽게 하기 위해서 이것은 그것에 2를 곱한 10×11 행렬을 만들고, 이것은 3을 곱한 10×11 행렬을 만들자. 이렇게 3개의 다른 행렬, 세타 1,2,3이 생겼다. 우리는 이것을 벡터로 풀고 싶어한다. $\text{thetaVec} = \text{theta } 1; \text{theta } 2 \text{ theta } 3$. 여기는 콜론을 찍고 이것은 굉장히 긴 벡터가 될 것이다. 이것은 231개의 요소들이 있을 것이다. 내가 이것을 다 적는다면 모든 행렬의 요소가 다 나오기에 아주 길어질 것이다. 다시 원래 행렬로 돌리고자 한다면 이것을 reshape 하면 된다. 그러면 이것이 다시 10×11 행렬로 돌아간다. 이것은 세타 1이다. 다시 이것을 꺼낸다면 111에서 220까지가 될 것이고, 나는 여기 세타 2들을 다시 찾을 것이다. 221에서

마지막 요소 231 을 하면 세타 3 을 다시 만들 수 있다. 이것이 학습 알고리즘을 시행하기 위해 unrolling 하는 방법이다.

05'38"

여러분이 변수 세타 1, 2, 3 의 초기값을 가지고 있다고 하자. 우리가 할 것은 이것을 활용해서 긴 벡터를 만들어 낼 것이고, 이 fminunc 를 활용할 것이다. 우리는 비용함수를 시행해야 한다. 이것이 비용함수인데, 이것은 thetaVec 이라는 입력 값은 줄 것이고, 이 값이 펼쳐져 벡터가 될 것이다. 첫번째로 이것을 사용해서 함수를 reshape 할 것이다. 이것을 활용하여 세타 1, 2, 3 의 요소들을 사용하고 다시 돌려놓을 것이다. 이것이 내가 만들 행렬인 것이다. 이것은 forward propagation, back propagation 으로 미분계수를 계산하고 j 세타 함수를 계산할 더 편리한 형태인 것이다.

06'40 "

마지막으로 이 미분계수를 사용하여 계산 할 것인데, 이는 처음과 같은 순서일 것이다. D1, 2, 3 를 unroll 하여 gradientVec 을 얻어낼 것이다. 이렇게 얻어낼 수 있고, 여러분이 이 reshape 하고 unroll 하는 과정을 이해할 수 있기 바란다. 행렬이 편한 것은 forward, back propagation 을 할 때 행렬형태가 되어 있으면 편하기 때문이다. 하지만 반대로 벡터 형태는 진보된 최적화 알고리즘을 사용할 때 편하다. 이것들은 긴 벡터를 갖고 있다는 전제하에 움직이는 것이기 때문이다. 여러분이 이 두 가지를 바꾸는 데 익숙해져 있길 바란다.

No.	Title
Week 5-5	Gradient checking

00'00"

몇 번에 걸쳐 우리는 신경 네트워크에서의 forward 와 back propagation 에 대해 논했는데, 이것으로 미분계수를 계산했다. 하지만 back prop 은 복잡해서 시행하기 조금 난해했다. 그리고 아쉬운 특성은 버그가 좀 있다는 것이다. 그러니 여러분이 gradient descent 나 다른 최적화 알고리즘을 사용하면 이것이 잘 실행되는 것처럼 보일 것이다. 그리고 여러분의 비용함수 J 세타가 매번 gradient descent 의 반복마다 줄어들 것이다. 하지만 이것은 back prop 의 약간의 버그에도 불구하고 사실로 밝혀질 수 있다. J 세타 함수가 줄어든다면 여러분보다 버그가 많은 신경 네트워크를 사용하게 될 것이다. 그리고 이 미미한 버그가 퍼포먼스에 영향을 준다는 것을 몰랐다. 그러면 우리는 이것을 어떻게 해야할까?

00'55"

gradient checking 이라고 불리는 이런 문제들을 없애는 개념이 있다. 그러니 오늘은 신경 네트워크나 다른 적당히 복잡한 모델의 back prop이나 gradient 를 시행하여 gradient checking 을 할 것이다. 여러분이 이것을 한다면, for prop이나 back prop 이 100 퍼센트 정확하다는 것에 대한 확신을 가질 수 있을 것이다. 이것이 이런 버그 문제들을 없애는 경우를 많이 보았다. 그리고 지난 영상에서 지난 번에 델타나 그 외의 것들을 계산하기 위한 수식은 비용함수의 gradient 를 구하는 것이라고 얘기하였다. 하지만 여러분이 수치적 gradient checking 을 하게 되면, 여러분은 cross function J 의 미분계수를 정말 계산하고 있는지 확인할 수 있을 것이다. 여기 이것이 그 개념이고 해당 예들을 생각해 보자.

01'57"

함수 J 세타가 있고, 이것의 세타 값은 실수라고 하자. 그리고 이 부분에서 이 함수의 미분계수를 측정하고자 하고 이 미분계수는 이 탄젠트 1 의 기울기와 같다. 이것이 미분계수를 측정하는 방법이다. 이제 세타 더하기 앱실론의 값을 구할 것인데, 이렇게 오른쪽에 그릴 수 있다. 또한 세타 마이너스 앱실론을 여기 이렇게 그리고 이렇게 직선으로 연결할 것이고, 이 빨간 선의 기울기를 내 미분계수의 측정치로 계산할 것이다. 실제 미분계수는 저기 파란색 선의 기울기이다. 이것이 좋은 예측이었다는 것을 알 수 있을 것이다. 수학적으로 이 빨간 선의 기울기는 가로 길이로 나눈 수직의 높이이다. 그러니 이 포인트는 J 세타 플러스 앱실론인 것이다.

03'10"

여기 점 J 세타 마이너스 앱실론, 그러니 높이의 차는 J 세타 플러스 앱실론 마이너스 J 세타 마이너스 앱실론이고, 가로 길이는 2 앱실론이다. 그러니 내 예측은 j 세타의 세타가 이 값일 때의 미분계수인 것이다. 이것은 j 세타 플러스 앱실론 마이너스 J 세타 마이너스 앱실론 /2 앱실론 인 것이다. 보통 앱실론은 꽤 작은 값을 가지고 있고 보통 10 에서 -4 정도이다. 이것은 꽤 큰 범위이지만 잘 작동한다. 그리고 사실 앱실론이 정말 작으면 수학적으로 이 항은 미분계수가 된다. 이것은 이 점에서 함수의 기울기가 되는 것이다. 그래서 수학적 계산을 해야할 때는 이런 너무 작은 앱실론을 사용하지는 않는 것이다.

04'11"

그래서 보통 10 에서 -4 정도의 값을 앱실론 값으로 사용한다. 여러분 중 몇몇은 미분계수를 구하는 S 를 위한 또 다른 공식을 본적 있을지도 모른다. 이 오른쪽에 있는 것이 one-sided difference 라고 불리고, 왼쪽에 있는 공식이 two-sided difference 라고 불린다. two-sided difference 가 좀 더 정확한 측정을 내려주기에 one-sided difference 보단 이것을 사용한다. 그래서 자세히 말하면, 옥타브를 시행할 때, 옥타브에서 이것을 시행할 때 gradApprox 를 계산한다고 말한다. 여기 적힌 것처럼 미분계수에 대한 대충의 예상치이다. 그리고 이 점에서의 gradient 의 수치적 예측에 대해서 알려줄 것이다. 이것이 그 예이고, 좋은 예측처럼 보인다.

Q) $J(\theta) = \theta^3$ 으로 두고 $\theta = 1, \epsilon = 0.01$ 이고 미분 계수를 최대화하기 위해 다음 공식을 사용한다.
알맞은 값을 고르라. 정답: 2 번

05'02"

지난 슬라이드에서 세타가 실수 였을 때의 경우를 살펴보았다. 이제는 세타가 벡터 변수일 때의 좀 더 일반적인 경우를 살펴보자. 세타는 R^n 이라고 하자. 이 경우는 신경 네트워크의 변수의 unrolled version 일 수도 있다. 그러니 세타는 n 개의 요소를 가진 벡터이고, 세타 1에서 n 까지 존재한다. 자세히 말하면 비용함수의 편미분계수이고, 이것은 첫 번째 변수인 세타 1 을 반영했다. 그리고 이 값은 세타 1 을 증가시킴으로 얻을 수 있다. 그래서 여러분은 현재 J 세타 1 플러스 엡실론 등등 마이너스 이것을 하고 그것을 2 엡실론으로 나눈 값을 보고 있다. 두 번째 변수 세타 2 의 편미분계수는 다음과 같다. 이 등식은 세타 i 중 하나의 편미분계수의 수치적인 예측인 것이다.

06'25"

이제 여러분은 이것들을 시행할 것이다. 이것을 옥타브에서 시행할 것이다. $i = 1:n$ 의 경우, n 은 벡터 세타의 차수를 말한다. 주로 이것을 unrolled version 의 변수와 함께 사용한다. 그러니 세타는 신경네트워크의 변수의 긴 리스트라고 볼 수 있다. $\text{thetaPlus} = \text{theta}$ 라고 지정하고 $\text{thetaPlus of the } (i) \text{ element by epsilon}$ 라고 지정할 것이다. 이것은 thetaPlus 가 $\text{thetaPlus}(i)$ 가 아닌 경우의 세타와 같다고 할 수 있다. 왜냐하면 $\text{thetaPlus}(i)$ 는 엡실론으로 증가하고 있기 때문이다. 엡실론, 즉 세타플러스는 세타 1, 2, 3 등과 같다. 그러면 세타 i 에는 엡실론이 더해지고 세타 N 까지 내려가게 된다. 이것이 세타플러스이다. 그리고 이 유사한 두 선은 세타마이너스를 비슷한 것으로 지정하는데, 세타 i 플러스 엡실론이 아닌 이것은 세타 i 마이너스 엡실론인 경우를 제외하고 동일하다. 그리고 $\text{grandApprox}(i)$ 에 대해 시행하면 이것은 j 세타의 세타 i 의 partial 미분계수에 대한 예측을 제공할 것이다.

07'35"

우리가 신경 네트워크에서 사용할 방법은 이것을 loop 이 비용함수의 top 편미분계수를 계산하게하도록 하는 것이다. 그러면 우리는 back prop 을 통해 얻은 gradient 를 얻어낼 수 있다. 그러니 DVec 은 backprop 으로 얻어진 미분계수다. 그러니 back propagation 을 보면 미분계수, 편미분계수를 구하는 데 꽤 효율적인 방법이었다. 수치적으로 계산된 미분계수인 이 gradApprox 를 사용할 것이고 이것이 작은 수치 round up 에 이르기 까지 같거나 비슷할 수 있을 것인가 확인하는 것이다. 실제로 이것은 꽤 비슷해 보인다. 그러니 이 두가지는 같은 혹은 비슷한 답을 제공한 것이다. 그러면 내 backdrop 이 맞았다는 것을 확인할 수 있다.

08'42"

이 DVec 벡터를 gradient assents 나 다른 고급 최적화 알고리즘에 삽입하면 나는 내가 계산한 것이 맞고, 입력한 코드가 맞으며 J 세타를 잘 최적화시켰다는 것을 알 수 있다. 마지막으로 나는

모든 것을 모아서 numeric gradient checking 을 어떻게 하는지 알려주겠다. 이것이 내가 주로 하는 것들인데, 첫 번째는 DVec 을 계산하기 위해 back prop 을 사용하는 것이다. 여기 그 과정이 있고 우리는 이 행렬의 unrolled version 을 활용했을 것이다. 내가 할 것은 gradApprox 를 제한하기 위해 numeric gradient checking 을 시행할 것이다. 이것이 지난 슬라이드에서 언급했던 내용이다. 이것이 중요한 부분인데, 여러분이 학습을 위한 코드를 사용하기 전에 gradient checking 을 끄고 더 이상 gradApprox 를 계산하지 않게 두는 것이 중요하다.

09'53"

그 이유는 numeric code gradient checking code, 우리가 말했던 이것은 굉장히 계산적으로 비싸고 미분계수를 예측하는데 아주 오래 걸리기 때문이다. 반대로 우리가 예전에 설명한 backdrop 알고리즘은, Dvec 은 D1, D2 D3, 이렇게 빠르게 계산해 낼 수 있다. 그러니 여러분의 back prop 이 맞았다는 것을 알게 되면 끄고 그만 사용해야 한다. 다시 말해, 다른 gradient descent 알고리즘이나 다른 것들을 사용하기 전에 이것을 끄는 것이 중요하다.

10'39"

정확히 말하면 여러분이 매번 이 numerical gradient checking 을 사용하면 여러분의 코드가 굉장히 느려질 것이다. 왜냐하면 이것은 backdrop 보다 훨씬 느리기 때문이다. backdrop 은 이렇게 델타 4, 3, 2 등을 계산할 때 쓴 방법이다. 이것이 gradient checking 보다 훨씬 빠르다. 그러니 여러분이 계산한 값이 맞다는 것을 확인했을 때 이것을 비활성화하거나 끄는 것이 매우 중요하다.

Q) 알고리즘이 학습하는 동안 numerical gradient checking 대신 backdrop propagation 을 사용하는 이유는 무엇인가? 정답: 2 번

이것이 backdrop 이나 gradient descent 가 맞다는 것을 확인하는 방법이고, 나는 항상 내 코드가 맞는지 확인하기 위해 이를 사용한다.

No.	Title
Week 5-6	Random initialization

00'00"

지난 영상에서 우리는 네트워크를 정확히 시행하고 훈련하기 위해 모든 것들을 사용했다. 그리고 이제 살펴볼 내용은 무작위 초기화이다. 여러분이 gradient descent 를 사용할 때나 다른 알고리즘을 사용할 때, 우리는 변수 세타의 기본 값을 택해야 했다. 고급 최적화 알고리즘에서 이것은 여러분이 변수 세타의 초기값을 입력할 것이라고 전제하고 있다. 그러면 gradient descent 를 생각해 보자. 이것은 우리가 세타를 어떤 값으로 세팅해야 하고 그래야 gradient descent 를 타고 천천히 내려갈 수 있다. 내려가기 위해서, 그러니 θ_j 세타를 최소화하는 것을 하기 위해서 우리는 어떻게 세타의 초기값을 지정해야 할까? 모두 0 으로 해도 될까? 이것이 로지스틱 회귀라면 모든 변수를 0 으로 지정해도 되지만 이것은 신경 네트워크에서 사용하는 것이라 그렇게 해선 안된다.

01'03''

이 신경 네트워크를 보면, 모든 네트워크의 변수를 0 으로 지정한다고 하자. 이렇게 하면 초기값 세팅 과정에서 이 파란색 가중치는 이 수치와 같아지고 둘 다 0 이 될 것이다. 이 빨간색으로 지정한 것은 이 것과 같아지고 초록색도 이 가중치와 같아진다. 이것이 의미하는 바는 여러분의 hidden unit A1, A2 도 여러분의 입력 값과 같은 함수를 계산하게 된다는 것이다, 그러면 모든 여러분의 트레이닝 예가 $a_1^{(2)}$, $a_2^{(2)}$ 가 되는 것이다. 이것을 너무 깊이 설명하지는 않을 것이지만 이 outgoing weight 가중치가 같기에 여러분은 델타값 또한 같은 것을 볼 수 있다.

01'59''

그러니 여러분은 델타 1 1, 델타 2 1 는 델타 2 2 와 같아질 것이고, 이 그림을 좀 더 살펴보면 여러분은 비용함수의 편미분계수가 이것을 충족시킨다는 것을 알 수 있을 것이다. 네트워크의 두 파란 선의 미분계수를 벗어난 수치를 반영한 비용함수의 편미분계수인 것이다. 여러분은 이것이 둘 다 동일한 값이라는 것을 알게 될 것이다. 이것이 의미하는 것은 하나의 greater descent 업데이트 이후에 여러분은 또 이 첫 번째 파란 비율이 학습 속도이며 곱하기 이것을 할 것이고 두 번째 곱하기 이것을 할 것이다. 그러니 descent 업데이트 이후에도 이 두 파란색 비율은 같은 값이라는 것이다. 이것은 0 이 아닌 값이지만 0 과 비슷할 수는 있다. 그리고 업데이트 후에도 이 값은 저 값과 같다.

03'07''

0 이 아닌 값들이 있지만 두 빨강의 값은 동일하다. 유사한 방법이 두 녹색에도 적용된다. 둘 다 같은 변하지만 같은 값으로 변한다는 말이다. 그러니 업데이트 이후에도 이 hidden unit 의 두 가지도 동일해진다는 얘기이다. 두 녹색 가중치가 여전히 같고, 빨강도, 파랑도 같은데, 한 번의 반복 즉 gradient descent 이후에도 동일하다는 것은 무슨 의미인가, 여러분은 이 두 가지의 유닛이 입력 값과 같은 함수를 계산하고 있다는 것을 알 수 있다. $a_1(2)=a_2(2)$ 인 것이다. 이 경우로 돌아가서 여러분이 greater descent 를 하더라도 2 가지 파란색은 동일한 값일 것이다. 빨간색도 초록의 경우에도 동일한 결과일 것이다.

03'56''

이것이 의미하는 것은 신경 네트워크가 굉장히 재미있는 함수를 계산한다는 것이다. 여러분이 2 개의 hidden unit 이 아니라 아주 많은 hidden unit 이 있다고 가정하자. 이것은 여전히 같은 특징을 계산할 것이다. 어떤 hidden unit 을 입력하든 그 입력 값과 같은 결과 값이 나온다는 말이다. 이것은 아주 불필요한 재현이다. 왜냐하면 마지막 로지스틱 회귀 유닛을 찾기 위해 각 유닛의 특징을 볼 필요가 없기 때문이다. 모든 값이 같으니 하나만 보면 되는 것이다. 이것은 여러분이 다른 것을 하는 데 방해가 된다. 이 문제를 해결하려면 신경 네트워크의 변수의 초기값을 설정하는 방법이 있는데 이것을 무작위 초기화로 시행할 것이다. 좀 더 자세히 설명하자면 지난번 symmetric 문제라고 불리는 문제에서와 동일한 것이다. 이 무작위 초기화는 우리가 대칭성 붕괴 symmetry breaking 을 하는 방법이다.

04'56''

그러니 우리는 각 $\theta_{ij}^{(l)}$ 를 랜덤한 값으로 초기화해야 한다. 그 범위는 $-\epsilon$ 과 ϵ 사이일 것이다. 이 사이의 랜덤의 값이다. 그래서 이 변수의 가중치는 이 범위 사이의 값으로 초기 설정이 된다는 것이다. 여기 옥타브에 코드를 기입하는 방법은 theta_1 =rand(10,11)*(2* INIT_EPSILON)-INIT_EPSILON; 이며 rand(10,11)은 10x11 행렬의 랜덤을 정하는 것이다. 이 값들은 0 에서 1 사이이고 이것은 0 과 1 사이의 연속된 숫자를 계속 사용할 것이다. 그러니 여러분이 0 과 1 사이의 숫자를 고르면 INIT_EPSILON 의 2 배를 곱하고 마이너스 INIT_EPSILON 해야 한다. 그러면 여러분은 마이너스 앱실론과 플러스 앱실론의 사이의 값을 얻게 될 것이다. 여기 ϵ 은 우리가 gradient checking 할 때 사용한 ϵ 과 아무 상관이 없는 값을 얻게 될 것이다. Numerical gradient checking 을 할 때는 앱실론과 세타에 다른 값들을 더했다. 이것은 앱실론의 unrelated value 이다. 우리는 INIT_EPSILON 를 써서 gradient checking 을 할 때의 앱실론의 값과 구분해낼 것이다. 그리고 유사하게 세타 2 의 초기값을 설정하고 싶으면 이런 코드를 사용하면 된다.

Q) 신경 네트워크에서 매개변수들을 초기화하려고 한다. 다음 두 조건을 충족시키는 것이 가능하겠는가? 정답: 4 번

06'17''

요약하자면 신경 네트워크를 구성하기 위해서는 여러분은 0 과 유사한 마이너스 앱실론에서 플러스 앱실론 사이의 값으로 설정해야 한다는 것이다. 그 후 back drop 을 실시하여 checking 을 실행하라. 이것은 J 세타 함수를 최소화하는지 판단해야 한다. 이것은 랜덤 값으로 설정한 변수 세타의 초기값의 함수이다. symmetry breaking 이후 great gradient descent 나 다른 알고리즘이 세타의 알맞은 값을 찾을 수 있을 것이다.

No.	Title
Week 5-7	Putting it together

00'00''

신경 네트워크 학습 알고리즘에 대해서 많은 강의에 걸쳐 설명하였다. 이번 강의에서는 모든 것을 한 자리에 모아서 좀 더 큰 그림을 그려 어떻게 각각이 다 맞추어지는지와 신경 네트워크 학습 알고리즘을 시행하는 전반적인 프로세스에 대해서 다룰 것이다. 우리가 신경 네트워크를 훈련시켜 볼 때, 여러분이 가장 먼저 해야 할 것은 네트워크 구조를 골라야 하는데, 여기서 구조란 신경들간의 연결 패턴을 의미한다. 그러니 여러분은 신경 네트워크가 세 개의 입력 값 유닛과 5 개의 히든 유닛, 4 개의 출력 값 유닛을 가질지 아니면 대응 값으로 3, 5, 5, 5 를 가질지 골라야 한다. 그리고 얼마나 많은 히든 유닛을 한 레이어에 가질지가 구조 선택의 문제이다. 그래서 여러분은 어떻게 선택할 것인가?

01'00''

입력 값 유닛의 숫자는 잘 정의되었다. 여러분이 고정된 특징 x 를 입력 값 유닛의 숫자라고 했을 때 여러분 특징 $x(i)$ 의 차수는 이것에 의해 결정될 것이다. 그리고 여러분이 멀티클래스 분류를 하고 있다면 이 출력 값의 숫자는 여러분의 분류 문제에서 클래스의 숫자에 의해 결정될 것이다. 그리고 여러분은 멀티클래스 분류에서 y 는 1에서 10 사이의 값을 가진다고 했기 때문에 여러분은 10 개의 가능한 클래스를 갖고 있는 것이다. 여러분의 출력 값 y 가 벡터라는 것을 기억해야 한다. 그러니 첫 번째 클래스 대신에 여러분은 이런 벡터를 다시 코딩해야 하고, 혹은 두 번째의 클래스를 이런 벡터로 다시 코딩해야 한다. 이 응용식이 여러분의 5 번째 클래스에 적용되면 $y=5$ 이면 여러분의 신경 네트워크에서는 $y=5$ 가 아니며 10 출력 값 유닛을 가질 수 있는 위쪽 레이어에서 벡터의 값을 지정할 것이다. 이 벡터는 5 번째 위치에 있고 여기 무수한 0 들이 밑에 있다. 그러니 입력 값 유닛, 출력 값 유닛의 숫자를 고르는 것은 무척 간단하다.

02'18''

히든 유닛, 히든 레이어의 수의 경우에는 가장 합리적인 디폴트 값은 하나의 히든 레이어를 사용하고 이러한 신경 네트워크는 왼쪽에 나타나 있는데, 히든 레이어가 하나인 것이 가장 흔한 경우이다. 혹은 여러분이 1 개 이상의 히든 레이어를 사용한다면 합리적인 초기상태는 각 레이어의 히든 유닛의 숫자와 같은 숫자여야 한다. 여기 2 개의 히든 레이어가 있고, 이 히든 레이어 각각은 5 개의 히든 유닛을 가지고 있다. 그러니 3 개의 히든 레이어가 있고 각각은 같은 숫자를 가지고 있다. 그러니 5 개의 히든 유닛이 있는 것이다. 왼쪽에 있는 네트워크 구조보다 더 합리적인 초기상태이다. 히든 유닛의 수는 사실 많을수록 더 좋다. 그리고 여러분이 많이 가지고 있다면 비용은 많이 들겠지만 더 좋은 경우인 것이다. 그리고 레이어의 히든 유닛의 수는 일반적으로 특징의 수인 x 의 차원과 비교 되거나 입력값 특징의 경우 등과 같은 숫자와 비교되곤 한다.

03'34''

따라서 이런 히든 유닛의 수는 비교 가능하다. 그리고 입력값 피쳐보다 크다면 유용하다. 그러니 이것이 신경 구조의 합리적인 디폴트 설정이 되길 바라고, 여러분이 이 가이드라인을 따른다면

잘 작동하는 것을 얻을 것이다. 하지만 우리는 뒤에서 알고리즘을 어떻게 사용할지에 대한 팁을 줄 것이다. 특히 어떻게 신경 네트워크 구조를 고를지에 대한 얘기를 할 것이다. 그리고 히든 유닛을 잘 선택하는 법, 히든 레이어의 수, 등에 대해서도 다룰 것이다. 다음으로 우리가 할 것은 신경 네트워크 안에서 교환을 위해 이렇게 6 가지 단계를 행할 것이다. 여기에 4 가지가 있고 다음 슬라이드에 2 가지가 있다. 첫 번째로 할 것은 신경 네트워크를 설정하고 가중치의 값을 초기 설정하는 것이다. 우리는 0과 가까운 값으로 주로 설정한다.

04'32"

그러면 우리는 forward prop 을 사용해서, 어떠한 멋진 신경 네트워크를 사용해서, $h(x)$, 즉 y 값의 출력 값 벡터를 구할 수 있다. 또한 우리는 이 j 세타 함수를 계산하기 위해 코드를 시행할 수도 있다. 다음으로 우리는 back prop 을 시행하여 이 편미분계수 항들을 계산할 것이다. 자세히 말하자면 back prop 을 한다는 것이다. 우리는 이 트레이닝 예들로는 forward prop 을 할 것인데, 미리 들은 적이 있을지도 모르겠지만 이것은 굉장히 고도화된 벡터화 수단이고 for-loop 이나 m 트레이닝 예시가 없을 때 여러분은 back prop 을 실시하고 여러분의 코드에 for loop 이 있어야 하며, 여러분이 이 예들을 반복함으로써 x_1, y_1 을 알게 된다. 그러면 여러분은 for prop 과 back prop 을 첫 번째 예에 사용하면, 두 번째 반복에서는 여러분이 이 두 prop 을 사용하고 이런 반복이 되는 것이다. 이것은 마지막 예에 다다를 때까지 반복될 것이다. 제일 처음 시작할 때라도 여러분의 back prop 시행에는 for loop 이 있어야 한다.

05'48"

그렇다면 이것을 for-loop 없이는 시행하기 힘들다. 하지만 나는 여러분이 더욱 복잡한 버전을 시행하는 것을 권하지 않는다. 특히나 제일 처음 back prop 을 시행할 때는 더 그렇다. 그러니 자세히 말하면 우리는 m 트레이닝 예시에 for-loop 가 있고, for-loop 내에 for 과 back prop 을 이 하나의 예로 시행할 것이다. 이것은 우리가 $x(i)$ 를 취해서 입력값 레이어에 삽입하고 forward-prop 을 시행하고 back-prop 을 시행할 것이다. 그리고 이 활성화 이 델타 항이 신경 네트워크 내에 있으면 이것은 for-loop 내라고 할 수 있다. for-loop 의 정도를 나타내기 위해 이런 곡선을 그려보도록 하겠다. 이것은 물론 옥타브의 코드이고, 그리고 이것은 좀 더 sequence Java code 이고 그리고 for-loop 가 이렇게 마주친다. 우리는 이 델타 항을 계산할 것이고, 이 델타 항은 우리가 지난번에 다루었던 공식이다. $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$ 이다.

06'52"

그리고 마지막으로, 이 촉적된 델타 항을 계산하는 것 외에 우리는 다른 코드를 쓸 것이고, 이는 이 편미분계수항을 계산할 것이다. 이것들은 정규화 항인 람다를 고려해야 한다. 그렇다면 이 공식은 지난번에 주어진 것이다. 여러분은 어떻게 이것을 했었고 여러분은 이것을 계산할 코드가 있다. 다음 단계 5 번은 gradient checking 을 사용하여 우리가 계산한 것과 이 편미분계수항을 비교할 것이다. 그러나 back prop 으로 한 것과 수치적으로 계산한 것을 비교하는 것이다. 이 두 가지가 비슷한 값을 가지는지 검사하는 것이다. 이것을 하고 나면 우리의 back prop 시행이 옳은 것이라는 것을 알 수 있다. 그리고 이것을 사용하기 위해 끄는 것이 아주 중요하다.

08'00"

그리고 마지막으로 우리는 gradient descent, LB, GS 등의 최적화 알고리즘을 사용할 것이다. 이것들은 fminunc 나 다른 방법들을 탑재하고 있다. 그리고 이것을 back prop 과 함께 사용할 것이다. 그렇다면 back prop 은 이 편미분계수를 계산해 줄 것이다. 그리고 우리는 어떻게 비용함수를 계산하는지 알고 있고, 어떻게 back prop 을 사용해서 편미분계수들을 계산하는지를 알고 있으니 우리는 이것을 활용하여 J 세타 함수, 즉 변수 세터의 함수를 최소화할 것이다. 그리고 신경 네트워크에 있어서는 이 비용함수 J 세타는 볼록 함수가 아니기 때문에 이론적으로 국소 최소값에 아주 민감하고, gradient descent 와 같은 알고리즘과 다른 고급 최적화 방법들은 이론상 국소 최적값 local optimum 혹은 local minimum 에 빠지게 된다. 하지만 실제적으로 이것은 큰 문제가 아니며 우리가 이 알고리즘이 전역 최적값 global optimum 을 찾는다고 보장하지 못할 지라도 j 세타 함수는 좋은 국소 최적값을 찾아낼 것이다.

09'14

마지막으로 신경 네트워크를 위한 gradient descent 가 아직도 모호하다면 이것을 보고 이해하길 바란다. 이것은 gradient descent 를 설명하기 위해 사용했던 식과 비슷하다. 그러니 우리는 이런 비용함수가 있고, 신경 네트워크에 변수가 여럿 있다. 여기에 2 개의 변수 값을 기입하겠다. 실제로는 물론 신경 네트워크에서 아주 많은 변수가 있다. 이런 행렬의 세타 1, 2 등이 있는 것이다, 그러니 우리는 굉장히 다차원의 변수를 가지지만 여기 우리가 그릴 수 있는 것에는 한계가 있다. 우리가 이 신경 네트워크에 2 개의 변수만이 있다고 가정하겠다. 비록 명백히 실제에는 훨씬 많은 변수가 있다. 그러니 여러분이 이것을 여기에 두자. 이것은 j 세타가 꽤 낮은 점이고 이것이 변수의 세팅과 일치한다.

10'38"

이것은 우리 변수 세타의 세팅이고, 많은 트레이닝 예에서 내 가정함수의 출력 값은 $y(i)$ 와 가깝고, 이것이 사실이라면 내 비용함수를 낮추는 요인일 것이다. 하지만 반대로 여러분이 이런 값을 갖게 된다면, 신경 네트워크의 출력 값은 실제 $y(i)$ 값과 많은 차이가 있다. 그러니 이 선위의 점은 가정함수와, 신경 네트워크의 출력 값과 $y(i)$ 의 값의 차이가 크다는 것을 의미한다. 그러니 이것은 트레이닝셋과 잘 들어맞지 않는다. 그 반면 이런 낮은 값을 가진 점들은 j 세타가 낮을 때와 일치하고, 그러니 신경 네트워크와 트레이닝셋은 잘 맞는다. j 세타를 작게 하기 위해서 이것이 참일 필요가 있다.

Q) 여러분이 다음 J 함수를 최소화하기 위해 gradient descent 와 backpropagation 을 함께 사용한다고 하자. 다음 중 학습 알고리즘이 제대로 작동하는지 확인하기에 적절한 단계는 무엇인가? 정답: 3 번

11'16"

그러니 gradient descent 의 역할은 우리가 어떤 랜덤의 시작점으로부터 시작해서, 이런 점에서부터 시작해서 계속 하강하는 것을 의미한다. 그리고 back prop 이 하는 것은 gradient 의 방향을 잡는 일인데, gradient descent 는 이 경우에는 국소 최적값에 도달할 때까지 조금씩 내려오는 것을 의미한다. 그러니 여러분이 back prop 을 사용하고 gradient descent 나 다른 고급 최적화 방법을 사용한다면, 이 그림이 여러분에게 알고리즘이 어떤 것인지를 설명할 것이다. 여러분의 트레이닝 셋에서의 $y(i)$ 값과 신경 네트워크의 출력 값이 일치하는 변수의 값을 찾아내려 하고 있다. 그러니 이것이 여러분에게 다른 신경망 러닝이 어떻게 맞아 떨어지는지에 대한 이해를 높였으면 좋겠다. 여러분이 이 영상 다음에도 이것들이 어떻게 다같이 작동하는지 모르겠더라도 크게 상관은 없다.

12'20

신경망이나 back prop 은 복잡한 알고리즘이다. back prop 의 수학적 기반에 대해 아주 오랜 기간 동안 봐왔지만 아직도 이것이 의미하는 바가 무엇인지 모호할 때가 있다. 그리고 최적화 프로세스가 j 세타를 최소화하는 것이 어떻게 보이는지도 헷갈릴 때가 있다. 이것은 선형 회귀나 로지스틱 회귀처럼 아주 명확하게 잘 다를 수 있는 알고리즘이라고 생각하지 않는다. 따라서 여러분이 어렵다고 느끼더라도 괜찮다는 것이다. 하지만 여러분이 back prop 을 사용했을 때 이것이 가장 영향력 있는 알고리즘 중 하나라는 것을 알길 바라고, back prop 이나 어떤 방법들을 사용 했을 때 굉장히 복잡하고 powerful 한 비선형의 함수에 잘 맞아 떨어지기를 바란다. 이것은 오늘날 가장 효과적인 러닝 알고리즘 중 하나이다.

No.	Title
Week 5-8	Autonomous Driving

00'00"

이 영상에서는 아주 재미있고 역사적으로 중요한 예인 자동 운전을 위한 신경 네트워크를 활용한 신경 네트워크 학습에 대해서 설명할 것이다. 이것은 차가 자동으로 운전하게끔 하는 알고리즘이다. 1 분 전의 영상은 카네기멜론 대학에서 일할 때 내 동료였던 Deam Pomerleau로부터 받은 것이다. 이 비디오에서 여러분은 이러한 시각화된 것들을 볼 것이다. 그리고 이러한 시각 자료에 대해서 설명하겠다. 여기 왼쪽 아래에 무엇이 앞에 있는지, 차의 시각으로 본 것이다. 그리고 여러분은 여기 왼쪽에 길이 보이니 왼쪽으로 갈 것이고, 오른쪽으로도 조금 움직일 것이다. 그리고 여기 위쪽에 첫번째 수평선 바는 사람 운전자가 고른 방향이 보인다. 이 밝은 흰색의 밴드가 사람이 선택한 방향이고 여기 왼쪽은 왼쪽으로 꺾는 것을 의미하고 이것은 오른쪽으로 꺾는 것을 말한다.

01'07"

그러니 이 정도의 위치는 왼쪽으로 조금 가는 것, 그리고 이 왼쪽 중간의 위치는 사람이 아주 조금 왼쪽으로 트는 것을 의미한다. 그리고 여기 두 번째는 러닝 알고리즘이 선택한 방향이고, 여기 하얀색 밴드는 신경 네트워크가 여기로 방향을 틀었다는 것인데 이것은 왼쪽으로 약간 움직인 것이다. 그리고 사실 신경 네트워크가 약간 틀기 전에 여러분은 여기 네트워크가 회색을 만들어 낸 것을 알 수 있다. 그리고 이 회색은 처음에 신경 네트워크가 랜덤으로 초기 설정한 것이다. 그리고 초기에는 어떻게 운전을 해야 할지에 대한 개념이 없었거나 어떤 방향으로 가야할 지 몰랐다. 하지만 배우고 난 뒤에 이렇게 흰색 띠를 만들어 냈고, 지역의 아주 조금의 부분은 어떤 특정한 방향을 선택하는 것과 같아졌다. 이것은 신경 네트워크가 어떤 특정 방향으로 가는 것을 고르는 것을 확신하는 것이고, 회색 상태일 때와는 다르다. 이것은 계속적으로 운전 방향을 고르는 흰 띠를 만들어 내고 있다.

02'22"

ALVINN은 인공 신경 네트워크이며 사람이 운전하는 것을 보고 운전 방식을 배운다. ALVINN은 NAVLAB 2를 통제하기 위해 만들어졌는데 이것은 자동 네비게이션 실험을 위한 센서, 컴퓨터, 작동기가 설치되어 있는 군용 장갑차 Humvees를 개조한 것이다. ALVINN의 환경 설정의 첫 단계는 여기 네트워크를 만드는 것이다. 트레이닝 동안 ALVINN이 보고 있을 때 사람이 운전한다. 2초마다 한번씩 ALVINN은 앞의 길의 비디오 이미지를 수치화하고 사람의 운전 방향을 기록한다. 트레이닝 이미지는 30x32 픽셀의 해상도로 줄여지고, ALVINN의 3레이어 네트워크에게 제공된다. 이것은 backdrop 러닝 알고리즘을 사용하여, 이미지의 사람 운전자가했던 것과 동일한 운전 방향을 수행하도록 훈련 받는다. 네트워크 운전 반응은 처음에는 랜덤이다. 트레이닝 2분 후 네트워크는 인간 운전자의 운전 반응을 모방하고, 이 실험은 다른 길 유형에서도 반복된다.

04'10"

신경 네트워크는 사용자가 작동 스위치를 누르도록 훈련한 후에 ALVINN은 운전을 시작한다. 1초당 12번 ALVINN은 신경 네트워크로 오는 이미지를 수치화한다. 각각의 네트워크는 평행하게 움직이고, 이것은 운전 방향을 알아내고, 이 반응에 확신을 갖는다. 운전 방향은 가장 확신하는 네트워크에서 오는데, 이 네트워크는 차체를 통제하는데 사용된다. 교차로가 앞에 나타나면 one lane network의 확신은 줄어든다. 이것이 교차로를 건너서 가면 2개의 길이 시야에 나타나고 two lane network의 확신이 올라간다. 이 확신이 올라갈 때 이 네트워크가 운전을 하는 것이다. 그리고 이것은 2lane의 길에 이 차체를 안전하게 인도하는 것이다.

06'06"

이것이 신경 네트워크를 사용한 자동 운전이다. 물론 더 새로운 시도들도 많다. 미국, 유럽 그 외 다른 곳에서 이보다 더 획기적인 프로젝트가 진행되고 있지만 여전히 이것은 인상적이고

대단하다. 왜냐하면 backprop 을 사용해서 즉시 신경 네트워크가 운전을 잘하도록 배우기 때문이다.

Week 6

No.	Title
Week 6-1	Deciding What to Try Next

00:00

지금쯤이면 여러분은 다양한 학습 알고리즘에 대해 학습했을 것이다. 그리고 이번 강좌를 꾸준히 학습해왔다면, 최신 기계학습 기술에 대해 전문가라고 봐도 무방하다. 하지만 어떠한 학습 알고리즘을 알고 있는 사람들 중에서도 그러한 알고리즘을 효율적으로 활용할 줄 아는 사람과 반면에 이제 이번 강의에서 설명하고자 하는 자료에 익숙하지 않은 사람간에는 큰 차이가 있다. 이러한 알고리즘을 어떻게 응용하는지 알지 못하는 사람은 논리가 맞지 않은 것을 시도하다가 많은 시간을 낭비할 수도 있다.

00:34

확실히 하고 싶은 것은 여러분이 기계학습 시스템을 개발할 때, 시간을 투자할만한 전망이 있는 분야를 선택할 줄 알아야 한다는 것이다. 이번과 다음 번 강의에서 이를 위한 실용적인 제안, 조언, 가이드라인을 설명하도록 하겠다. 따라서 우리가 초점을 맞춰야 할 곳은 문제인데, 기계 학습 시스템을 개발하거나 이의 성능을 향상시키려고 한다고 하면 다음 과정을 시도하기 위한 Proxy Avenues는 무엇인지 어떻게 결정할 것이냐?

01:09

이를 설명하기 위해서 주택 가격 예측 학습 예시를 다시 활용해보자. 선형 회귀를 구현하고 regularization한다고 하자. 따라서 비용 함수 j 를 최소화시켜야 한다. 이제 학습 매개 변수를 가져온 후에, 새로운 주택 세트의 가설을 테스트하면서 주택 가격 예측에서 엄청난 오류가 발생한 것을 발견했다고 하자. 문제는 이러한 학습 알고리즘을 향상시키기 위해 다음에 무엇을 하겠느냐는 것이다. 학습 알고리즘의 성능을 향상시키기 위한 방법은 여러 가지가 있다.

01:44

먼저 한가지 방법은 더 많은 학습 예시를 얻는 것이다. 예를 들어, 전화 설문조사, 방문 조사를 통해 다양한 주택 가격 데이터를 얻을 수 있다. 슬픈 사실은 많은 사람들이 학습 예시를 모으는데 많은 시간을 투자한다는 것인데, 학습 데이터가 2배, 아니면 10배 더 많으면 더 도움이 되겠지 이렇게 생각하는 것이다. 하지만 때로는 데이터가 많다는 것이 꼭 도움이 되는 것을 의미하는 것은 아니다. 왜 그런지 다음 강의에서 알아볼 것이고, 여러분에게 도움이 되지 않는 학습 데이터를 모으는데 시간을 낭비하지 않는 방법을 설명해주도록 하겠다.

02:22

또 다른 방법은 더 작은 세트의 feature를 시도해보는 것이다. x_1, x_2, x_3 등과 같은 features 아마도 많은 수의 features가 있을 것이다. 아마도 여러분은 그 많은 셋중에서 과적합을 방지하기 위한 작은 규모의 부분집합(subset)을 신중히 선택하기 위해 시간을 보낼 것이다. 또는 추가의 feature가 필요할지도 모른다. 아마 현재 feature 세트는 충분한 정보를 주지 못할 수도 있어 여러분은 feature를 더 얻기 위해 더 많은 데이터를 원할 것이다.

다시 한번 말하지만 이러한 프로젝트는 큰 프로젝트로 스케일업 할 수 있는데, 전화 설문조사를 통해 주택이나 토지 설문조사를 통해 가능할까? 따라서 이러한 과정이 도움이 될지를 미리 알아봐야만 시간을 낭비하지 않게 된다. 또한 다행식 feature도 있는데, $(x_1)^2, (x_2)^2$, and x_1x_2 등이 있다. 이러한 것을 다 따지게 되면 시간이 많이 소모될 것이고, regularization 매개 변수인 람다를 줄이거나, 이를 증가시킬 수도 있다.

03:23

이러한 옵션 중에서 어떤 것들은 6개월 또는 그 이상이 걸리는 규모가 되기도 한다. 안타깝게도, 주로 널리 이용되는 방법은 사람들은 그저 자신의 감을 따른다는 것이다. 사람들이 그저 이러한

옵션 중에서 임의로 선택을 하게 되는데, “학습 데이터를 좀 더 모으자.”라고 하고는 너무도 쉽게 6개월을 학습 데이터를 모으는데 허비해버리고, 또 어떤 이들은 데이터 세트에 있는 주택들의 feature를 좀 더 모아보자”라고 한다. 이렇게 사람들이 말 그대로 6개월을 이러한 옵션에 매달려 있는 것을 종종 봤고, 6개월 후에서야 그렇게 도움이 되는 옵션이 아니었다는 것을 깨닫게 된다.

04:07

다행스러운 것은, 이 목록에 있는 절반은 재빠르게 제외시켜 도움이 되는 옵션에 집중할 수 있게 하는 매우 간편한 기술이 있다는 것이다. 이는 매우 간편한 기술로 이를 구현하면 이중에서 많은 옵션을 배제시킬 수 있다. 따라서 도움이 되지 않을 작업을 하는데 쓸 시간을 절약할 수 있게 된다. 이번 강의 다음 두 강의에서는 학습 알고리즘을 어떻게 평가하는지에 대해 이야기하겠다. 그리고 또 그 다음의 강의에서는 기계 학습 진단장치라 불리는 이러한 기술에 대해 설명하도록 하겠다.

04:46

여기서 진단이란 구현할 수 있는 테스트인데, 한 알고리즘이 구현되는지 아닌지에 대한 통찰력을 얻게 되는 것인데, 학습 알고리즘의 성능을 향상시키기 위해서 어떠한 작업이 전망이 좋은지에 대해 알려주는 작업이다. 특정한 진단에 대해서는 강의 후반에 설명하도록 하겠다. 하지만 먼저 이야기해주고 싶은데, 진단이란 이를 구현하기 위해 시간이 걸리고 때로는 이를 구현하고 이해하는데 시간이 꽤 걸리기도 하는데 하지만 이렇게 하면 학습 알고리즘을 개발하는데 시간을 잘 활용할 수 있게 된다. 왜냐하면 아까처럼 한 방법을 따라가다가 몇 개월을 보내다가 나중에야 그다지 쓸모가 없다는 것을 알게 되는 것을 막을 수 있기 때문이다.

Q 다음 진단에 대한 설명 중 옳은 것을 모두 고르시오.

정답 2, 3, 4번

05:32

따라서 이 후 강의에서는 먼저 학습 알고리즘을 어떻게 평가하는지 알아본 후에 여러분의 목표인 기계 학습 시스템을 향상시키기 위해서 어떠한 유용한 것을 효과적으로 선택할 수 있는 진단 방법에 대해서 설명하도록 하겠다.

No.	Title
Week 6-2	Evaluating a Hypothesis

00:00

이번 강의에서는 알고리즘을 통해 학습된 가설을 어떻게 평가하는지에 대해 알아보겠다. 이후

강의에서는 이에 대해 더 자세하게 학습하여 과적합, 시부적합과 같은 문제를 피할 수 있도록 하려고 한다. 학습 알고리즘의 매개 변수를 대입할 때, 우리는 학습 오류를 최소화할 수 있는 매개 변수를 선택하려고 한다. 학습 오류의 낮은 값을 얻는 것이 좋을 거라고 생각하겠지만, 가설이 낮은 학습 오류를 가지고 있다고 할지라도 그것이 곧 좋은 가설이라고는 할 수 없는 것이다. 또한 우리는 가설을 어떻게 과적합 할 수 있는지 예시를 살펴봤었다. 따라서 학습 세트에 있지 않은 새로운 예시를 일반화 하는데 실패했었다. 따라서 가설이 과적합 되었다면 이를 어떻게 알아볼 수 있을까? 여기 간단한 예시에서 $h\theta(x)$ 그래프를 통해 이를 알아보도록 하자. 하지만 일반적으로 2개 이상의 feature를 가진 문제, 이와 같이 많은 feature 수가 더 가진 문제는 그것이 더 어렵고 이 가설이 어떻게 생겼는지 그래프화 할 수 없을 수도 있다. 따라서 이러한 가설을 평가하기 위한 다른 방법이 필요하다.

01:13

학습된 가설을 평가하기 위한 기본적인 방법은 다음과 같다. 이와 같은 데이터 세트가 있다고 하자. 여기 10개의 학습 예시가 있다. 하지만 보통 우리에게는 더 많은, 수백, 수천 개의 학습 예시가 있을 수도 있다. 우리가 가설을 평가할 수 있는지 알아보기 위해서 먼저 데이터를 두 부분으로 나누는 것이다. 첫 번째 부분은 보통 학습 세트이고, 두 번째는 테스트 세트이다. 데이터를 학습세트와 테스트 세트로 나누는 가장 전형적인 비율은 70%, 30% 이고, 점점 학습세트에 좀 더 비중을 두고 테스트 세트를 적게 잡는다.

이제는 데이터 세트가 주어졌을 때, 학습 세트의 데이터 중 여기 70%는 학습 예시에서 주로 사용하는 m 이 주어지고, 나머지 데이터 부분은 이렇게 테스트 세트가 된다. 이제 여기서 $m_{test} =$ 테스트 예시 수라고 표시하겠다.

02:28

따라서 보통 이러한 첨자 테스트는 테스트 세트에서 온 예시를 나타내므로, 이는 $(x(1))_{test}$, $y(1)_{test}$ 와 같은 형태가 내 첫 번째 테스트 예시가 될 것이고 이는 여기 표시한 예시가 된다. 마지막으로 알아봐야 할 것은

Q) 선형 회귀를 시행하는 과정(regularization 하지 않은)에서 학습 세트를 매우 과적합하게 되었다. 이러한 경우 예상되는 결과는?

1번

여기 이렇게 처음 70%는 학습 세트가 되고, 나머지 30%는 테스트 세트가 된다고 했는데, 만약 데이터가 순서대로 정리되어 있다면, 이 중 임의로 정해서 70%는 학습 세트로, 나머지 30%는 테스트 세트로 나누는 것이 좋다. 데이터가 임의로 섞여져 있다면, 그냥 70%, 30%로 나누면 되지만, 그렇지 않을 경우 임의로 데이터를 섞거나 순서를 셔플링하여 트레이닝 셋을 준비하는 게 좋다.

03:22

그렇다면 이제 학습 알고리즘과 학습 회귀에서 매우 전형적인 학습 세트와 테스트 세트를 나누는 과정을 알아보자. 먼저 학습 세트에서 매개변수 θ 를 학습하여, $j(\theta)$ 의 학습 오류를 최소화하고, 여기서 $j(\theta)$ 는 주어진 데이터의 70%를 사용하는 것을 의미한다. 이는 학습 데이터만을 의미한다. 그 다음에는 테스트 오류를 연산할 수 있다. 그리고 테스트 오류를 $j_{text}(\theta)$ 라고 표기할 것이다. 이제는 여기 학습 세트에서 학습한 매개변수 θ 를 가져와서 여기에 대입하여 테스트 세트 오류를 연산한다. 이는 이렇게 써주면 된다. 이는 테스트 세트에서 측정한 평균 제곱 오류이다. 아마 예상한대로 일 것이다. 따라서 모든 테스트 예시를 이 매개변수 θ 가 포함된 가설을 통해 돌려보면 여러분의 가설이 m첨자 테스트, 즉 테스트 예시에 있는 제곱 오류를 측정한다. 물론 이 테스트 세트 오류 정의는 선형 회귀와 제곱 오류 매트릭스를 활용할 때 이용하는 방법이다.

04:44

이번에는 분류 문제에 있어서, 로지스틱 회귀를 대신 활용하면 어떻게 될까? 그런 경우에 학습과 테스팅 과정은 로지스틱 회귀가 매우 흡사하다고 말할 수 있다. 먼저 학습 데이터, 즉 데이터의 70%를 가지고 매개변수를 학습한다. 그럼 테스트 오류를 다음과 같이 연산할 것이다. 이는 우리가 로지스틱 회귀에서 항상 사용하는 함수와 같은 것인데 이번에는 mtest를 테스트 예시로 사용했다는 점만 다르다. 여기 테스트 세트 오류인 $j_{text}(\theta)$ 의 정의는 완벽하게 합리적으로 보인다. 때로는 해석하기 더 쉬울 수 도 있는데 대체 테스트 세트 매트릭스가 있는데, 이는 오분류 오류라고 한다. 이는 또한 0/1 오분류 예러라고 불리기도 하는데, 0/1 표기는 맞거나 틀린 예시를 얻게 된다는 것이다.

05:36

이는 다음과 같다. 먼저 오류 예측을 정의해보겠다. $Err(h(\theta)x, y) = \{1 \text{ if } h(\theta)x \geq 0.5, y = 0 \text{ or if } h(\theta)x < 0.5, y = 1\}$ 이다. 따라서 이 두 케이스는 만약 이 가설이 예시를 오분류해서 0.5가 한계점이라고 예측할 때 나올 수 있는 기본 반응이다. 여기를 보면 1이 나올 것 같았지만, 결국 0이 나왔고, 가설에서는 0이 나올 거라고 했지만, label은 사실 1이었다. 또 다른 방법으로는 이 오류 함수를 0이라고 정의하는 것이다. 만약 가설이 예시 y 를 정확하게 분류했다는 가정하에 말이다. 그 다음에 테스트 오류를 정의하는데, test error = $1/mtest \sum_{i=1}^{mtest} err(h(\theta)(x(i)test), y(i)test)$ 이와 같다. 이것이 가설이 mislabel 한 테스트 세트에 있는 예시의 일부를 내 방식대로 적어본 것이다. 이는 0/1 오분류 매트릭스의 오분류 오류를 활용한 테스트 세트 오류 정의라고 할 수 있다. 이것이 바로 학습된 가설이 얼마나 좋은지를 평가하는 표준 기술이다. 다음 번 강의에서는 이러한 아이디어를 적용하여 학습 알고리즘에서 사용하는 degree 다항식과 같은 어떠한 feature를 선택하는지, 또는 학습 알고리즘의 regularization된 매개 변수를 선택하는지에 대해 배울 것이다.

No.	Title
-----	-------

Week 6-3	Model Selection and Train/Validation/Test Sets
----------	---

00:00

몇 차 다항식을 데이터 세트에 대입해야 하는지를 또는 어떠한 feature를 학습 알고리즘에 포함해야 하는지 결정해야 한다고 가정하자. 또는 학습 알고리즘을 위한 regularization 매개 변수를 선택해야 한다고 하자. 어떻게 할 것인가? 이것이 바로 모델 선택 과정이다. 이 과정을 배우면서 데이터를 학습 또는 테스트 세트에 어떻게 나눌 것인가를 배우는 것뿐 아니라, 우리가 발견한 것으로 데이터를 바꾸는 방법은 학습 / 검증 / 테스트 세트이다. 이게 무엇인지 그리고 모델 선택을 위해 이를 어떻게 활용하는지 알아보자.

00:37

우리는 이미 과집합의 문제에 대해 많이 접해봤다. 학습 알고리즘은 학습 세트에 잘 맞는다고 해서 좋은 가설은 아니라는 것이다. 다시 말하자면, 학습 세트 오류가 새로운 예시에서 그 가설이 얼마나 잘 작동될지는 잘 예측하는 것은 아니라는 것이다. 예를 들어, 매개 변수 세트를 대입하면, 먼저 $\theta_0, \theta_1, \theta_2$ 등을 학습 세트에 대입한다. 그러면 실제로 가설은 학습 세트와 잘 맞는다. 하지만 이는 학습 세트에 보이지 않는 새로운 예시를 여러분의 가설이 얼마나 잘 일반화하는지에 대한 예측을 의미하는 것은 아니다. 더 일반적인 원칙은 매개 변수가 데이터 세트에 맞게 되면, 학습 세트나 기타 세트에, 그러면 학습 오류와 같은 가설의 오류가 같은 데이터 세트에 측정되고, 이는 실제로 오류를 일반화하는 데 있어서 좋은 평가가 되기는 어렵다. 지금까지 가설이 새로운 예시를 얼마나 잘 일반화 하는 지에 대해 알아봤다.

01:39

이제 모델 선택 문제에 대해 알아보자. 몇 차 다항식을 데이터에 대입할지 선택한다고 하자. 선형 함수, 이차 함수, 삼차함수 중 어느 것을 선택해야 할까? 이는 10차 다항식까지 생각할 수 있을 것이다. 알고리즘에 추가 매개 변수를 d 라고 표시하고 이는 다항식의 차수를 선택하고 싶은지를 나타낸다. 따라서 θ 매개 변수뿐만 아니라, d 라는 매개 변수가 하나 더 있는 것이고, 여러분의 데이터 세트를 활용해 이를 결정하려고 한다. 첫 번째 옵션은 $d = 1$ 를 선형 함수에 대입하는 것이다. $d = 2, d = 3$, 그리고 $d = 10$ 까지 이렇게 선택할 수 있다. 이 d 매개 변수를 이렇게 대입하면 된다.

02:24

예를 들어, 모델을 선택하려고 하는데, 여기 10개 모델 중 몇 차 다항식을 선택할 것인가? 이 모델에 맞게 여러분의 가설이 새로운 예시에 일반화가 얼마나 잘 되었는지 평가를 얻을 수 있다. 이제 한 가지 방법을 소개하겠다. 먼저 첫 번째 모델을 선택하여 학습 오류를 최소화한다. 그러면 매개 변수 벡터 θ 값이 나올 것이다. 그 다음에는 두 번째 모델, 2차 함수를 선택해 학습 세트에 대입하면 다른 매개 변수 벡터 θ 값을 얻을 것이다. 여기 다른 매개변수 벡터를 구분하기 위해서, (1), (2) 를 활용하겠다. 이는 이 모델을 학습 데이터에 대입하여 얻은 매개 변수라는 뜻이다.

그리고 $\theta(2)$ 는 이 2차 함수를 학습 데이터에 대입하여 얻은 매개 변수라는 뜻이고 나머지도 이와 같다. 3차 함수 모델을 대입하면, $\theta(3)$ 이 나오고, 이는 $\theta(10)$ 까지 적용된다. 다음은 이 매개 변수를 가지고 테스트 오류를 살펴본다. 테스트 세트에서 $Jtest(\theta(1))$ 그리고 $Jtest(\theta(2))$, $Jtest(\theta(3))$ 등을 연산할 수 있다. 이제 가설과 이에 대응하는 매개 변수를 가지고 테스트 세트의 성능을 측정해보겠다.

03:59

그 다음에는 이 모델 중에 하나를 선택하기 위해, 어떤 모델이 가장 낮은 테스트 오류를 가지고 있는지 볼 것이다. 이번 예시를 위해 5차 다항식을 선택했다고 하자. 지금까지는 잘 맞는 것처럼 보였다. 하지만 이제 5번째 가설을 가지고 여기, 다섯 번째 모델, 이 모델이 얼마나 잘 일반화 되는지 알아보려고 한다.

04:27

먼저 여기 $Jtest(\theta(1))$ 를 통해 5차 다항식 가설이 테스트 세트에서 얼마나 잘 맞는지 확인할 수 있다. 하지만 문제는 이것이 가설의 일반화가 얼마나 잘 되었는지를 공평하게 측정하지는 않을 것이라는 점이다. 이유는 우리가 한 것은 여기 매개 변수 d 를, 이것은 몇 차 다항식인지를 알려주는데, 이 매개 변수 d 에 대입할 것은, 여기 테스트 세트를 활용하여 가장 최고의 성능을 보여줄 값을 선택했다. 따라서 테스트 세트에 있는 $\theta(5)$ 의 성능은 오류 일반화에 있어서 너무 낙관적으로 평가 되었을 가능성이 있다. 따라서 이 매개변수 d 를 테스트 세트에 대입했기 때문에, 이제 더 이상 테스트 세트에 있는 가설을 평가하는 것은 공정하기 어렵게 되었다. 왜냐하면 이 매개 변수를 테스트 세트에 대입하였는데, 몇 차 다항식을 할지를 선택할 때, 테스트 세트를 활용하였기 때문이다. 따라서 이 가설은 전에 본 적 없는 새로운 예시에서보다 이 테스트 세트에서 더 잘 맞을 것이다. 이 것이 바로 중요한 부분이다.

05:35

다시 말하자면, 이전 슬라이드에서 매개 변수 세트 θ_0, θ_1 등을 학습 세트에 대입하면, 학습 세트에 대입된 모델을 성능은 가설이 새로운 예시를 얼마나 잘 일반화 할 것인지는 예측할 수 없다. 이는 여기 매개 변수들은 학습 세트에 대입되었는데, 그렇기 때문에 이 매개 변수들이 다른 예시에서는 잘 맞지 않을 지라도 학습 세트에 잘 맞는 것이다. 그리고 이 과정에서 여기서 설명한 것과 같은 것을 한 것이다. 다시 말해, 우리가 한 것은 이 매개 변수 d 를 테스트 세트에 대입한다. 그리고 테스트 세트에 대입된 매개 변수를 가지게 되는데, 이는 테스트 세트에서 가설의 성능이 우리가 본 적 없는 새로운 예시에서는 공정하게 평가되지 않을 수 있다는 것이다.

06:26

이러한 문제를 제기하기 위해서, 모델 선택 셋팅에서, 가설을 평가하려고 하면, 다음과 같은 방식이 있다. 데이터 세트가 주어졌을 때, 학습 테스트를 나누는 것 대신에, 이를 세 부분으로 나누는 것이다. 첫 번째 부분은 똑같이 학습 세트라고 불린다. 여기 첫 번째 파트를 학습 세트라고 하겠다. 그리고 두 번째 데이터 부분은 교차 검증 세트라고 하겠다. 교차 검증.. 이를

줄여서 CV 라고 하겠다. 때로는 교차 검증 세트 대신 검증 세트라고 하기도 한다. 그리고 나서 나머지는 테스트 세트라고 한다. 대부분 이를 나누는 비율은 60%는 학습세트로, 그리고 20%는 교차 검증 세트, 나머지 20%는 테스트 세트가 된다. 그리고 이 숫자는 조금씩 달라질 수 있는데, 하지만 이러한 비율로 나누는 것이 매우 보편적이다. 그리고 학습 세트는 이제 데이터의 60% 정도이므로 교차 검증 세트 또는 검증 세트에도 예시가 몇 개 들어 있을 것이다. 이를 Mcv 라고 표기하겠다. 이는 교차 검증 예시의 수를 의미한다.

07:52

초기 표기 방법을 따르자면, $(x(i)cv, y(i)cv)$ 로 교차 검증 예시를 표기할 수 있겠다. 마지막으로 여기 테스트 세트가 있는데, Mtest 라고 표기하고 이는 이 안에 있는 테스트 예시 수를 말한다. 이와 같이 학습, 검증 또는 교차 검증 그리고 테스트 세트를 정의했다. 또한 학습 오류, 크로스 검정 오류, 테스트 오류를 정의할 수 있다. 여기 학습 오류가 있고 $Jtrain(\theta)$ 로 표기하겠다. 이는 거의 비슷하다. 지금까지 써왔던 $J(\theta)$ 와도 거의 흡사한데, 이는 학습 세트를 측정하는 학습 세트 오류이고, $Jcv(\theta)$ 는 교차 검증 오류로, 여러분이 예상하는 것처럼 학습 검증 데이터 세트에서 학습 오류를 측정한 것과 같고, 여기는 전과 같은 테스트 세트 오류가 있다.

08:49

따라서 모델 선택 문제를 보게 된다면, 우리는 모델을 선택하기 위해 테스트 세트를 활용하지 않고, 검증 세트 또는 교차 검증 세트를 활용하여 모델을 선택할 것이다. 예를 들어, 먼저 첫 번째 가설과, 첫 번째 모델을 가지고 $\min J(\theta)$ 하면, 이는 새로운 모델을 위한 매개 변수 벡터 세타를 줄 것이다. 그리고 아까처럼 $\theta(1)$ 을 써서 이 매개 변수가 새로운 모델을 위한 것이라고 표기해준다. 2차식 모델에게도 마찬가지다. $\theta(2)$ 를 써주고, $\theta(3)$ 그리고 $\theta(10)$ 까지 10차식까지 이렇게 적용한다. 이제는 테스트 세트에 있는 가설을 테스팅하는 대신에, 이를 교차 검증 세트에 테스트 해보겠다. $Jcv(\theta)$ 를 측정해서 각자 가설이 여기 교차 검증 세트에 잘 맞는지 확인해 볼 것이다.

09:53

이번에는 가장 적은 교차 검증 오류를 가진 가설을 선택해 볼 것이다. 이를 위해 $Jcv(\theta(4))$ 가 가장 적은 교차 검증 오류를 가지고 있다고 가정해보자. 그러므로 이렇게 네 번째 다항식 모델을 선택했다. 그리고 이는 매개 변수 d 가, 여기 d가 다항식에서 차수를 나타낸다는 것 기억할 것이다. 따라서 $d = 2, d = 3, \dots, d = 10$ 까지. 이제 이 매개 변수 d를 대입할 것이다. $D = 4$ 이다. 여기에 교차 검증 세트를 활용할 것이다. 그렇게 되면 이 다항식의 차수는, 그러니까 매개 변수는 테스트 세트에 맞춰 보거나 테스트 세트를 저장하는 것이 아니라 테스트 세트를 측정 또는 선택된 모델의 오류를 일반화하는 평가를 하는데 활용할 수 있다. 이렇게 모델 선택에 대해 배워봤고, 데이터를 가지고 학습, 검증 테스트 세트로 나눌 수 있는지 알아봤다. 이제 교차 검증 데이터를 가지고 테스트 세트에 있는 모델을 선택하고 이를 평가하게 되었다.

10:59

마지막으로 설명하고 싶은 게 있다. 오늘날 활용되는 기계 학습에서 내가 아까 설명해 준 내용을 하는 사람은 그리 많지 않다. 이러한 테스트 세트를 가지고 모델을 선택하는 것을 그리 좋은 방법이 아니다. 그리고 같은 테스트 세트를 활용하여 테스트 세트에 있는 다항식 차수를 선택하면서 오류를 보고하고 오류를 일반화하는 좋은 추정이 있음에도 테스트 세트에 있는 오류를 보고한다.

Q) 교차 검증 세트를 활용한 다항식 차수를 선택하는 모델 선택 과정이 있다. 최종 모델에서 (매개 변수 θ 가 있는) 우리는 일반적으로 $J_{cv}(\theta)$ 가 $J_{test}(\theta)$ 보다 낮을 것이라고 예상할 수 있다. 그 이유는?

1. 추가 매개변수(d , 다항식의 차수)가 교차 검증 세트에 대입되었다.
2. 추가 매개변수(d , 다항식의 차수)가 테스트 세트에 대입되었다.
3. 교차 검증 세트는 보통 테스트 세트보다 작다.
4. 교차 검증 세트는 보통 테스트 세트보다 크다.

: '정답 1'

이러한 시행은 안타깝게도 많은 사람들이 되풀이 하고 있다. 테스트가 엄청 많으면, 그렇게 하기 힘든 것은 아니지만, 현역에 있는 사람들은 기계 학습에서는 이를 피하라고 권고한다. 따라서 학습, 검증, 테스트 세트를 나누는 것이 더 낫다고 여겨진다. 가끔 사람들이 같은 데이터를 검증 세트 그리고 테스트 세트에서 활용하는 것에 대해 경고를 했다. 학습 세트, 테스트 세트 해보는 것은 좋은 연습이 될 수 있으나 여전히 그런 사람들이 많다. 하지만 나는 여러분은 그렇게 하지 않기를 추천한다.

No.	Title
Week 6-4	Diagnosing Bias vs. Variance

00:00

학습 알고리즘을 구현한다면, 여러분이 원하는 만큼 잘 안 되는데, 어쩌면 대부분 잘 안될 것이다. 왜냐하면 고차원의 bias 문제 또는 분산 문제를 가지고 있기 때문이다. 다시 말해, 시부적합 또는 과적합 문제를 지니고 있다는 것이다. 이러한 사례가 bias 인지 분산 인지 아니면 둘 다 해당하는 것인지를 분별하는 것이 매우 중요하다. 이는 이 두 문제가 발생하고 있다는 것을 아는 것은 알고리즘을 향상시키는 유용하고 가능성 있는 방법에 대한 매우 강한 인지 능력을 줄 것이기 때문이다.

00:30

이번 강의에서는 이 bias와 다양한 문제를 깊게 탐구하려고 한다. 또한 이것이 bias 문제 또는 분산 문제가 있는지 아닌지를 평가하는 방법을 이해하고 더 잘 이해하고자 한다. 구현하는 학습 알고리즘의 성능을 향상시키기는 방법을 이해하는 것은 매우 중요하다. 여러분은 이러한 수치를 이미 여러 번 봤을 것이다. 여기에 이렇게 간단한 두 가설을 적용하면, 이러한 직선은 데이터를 시부적합 하고 있다.

00:59

복잡한 두 가설을 적용시키면, 학습 세트에 완벽하게 맞을 수도 있겠지만, 이는 데이터가 과적합하게 되고, 중간 난이도쯤의 가설의 경우에는 2차 다항식은 그렇게 낮지도 높지도 않다. 적당하다고 볼 수 있다. 아마도 오류 일반화에서 가장 적합한 옵션을 제공할 것이다. 이제 테스트 세트에서 학습과 검증 개념을 가지고 bias 와 변수의 개념에 대해 좀 더 이해할 수 있을 것이다. 예를 들어, 학습 오류와 교차 검증 오류가 지난 번 강의에서처럼 정의된다고 하면, 20 세트 또는 크로스 검정 세트에 측정된 바와 같이 제곱 오류, 평균 제곱 오류가 된다.

01:46

이제 다음 수치를 도표화 해보자. 수평축에 다항식의 차수를 표시하면, 오른쪽으로 갈수록 더 높은 차수의 다항식이다. 이 수치를 적용해보면, $d = 1$ 이 되고, 매우 간단한 함수를 적용할 것이다. 반면에 오른쪽 그래프의 수평축은 $d = 4$ 이므로 상대적으로 훨씬 큰 숫자가 된다. 여기에는 매우 복잡한 고차 다항식을 적용할 것이고 이는 훨씬 더 복잡한 함수를 학습 세트에 적용하게 될 것이다.

02:20

이제 학습 오류와 교차 검증 오류를 여기에 그래프화 해보자. 먼저 학습 오류다. 다항식 차수를 올리게 되면, 학습 세트가 더 잘 들어맞기 때문에, $d = 1$ 일 때, 높은 학습 오류가 된다. 만약 매우 높은 차수의 다항식을 가지게 된다면, 학습 오류는 매우 낮을 것이다. 따라서 심지어 0이 될 수도 있는데, 왜냐하면 이 학습 세트에 매우 잘 맞을 것이기 때문이다. 따라서 더 다항식의 차수를 더 증가시키면, 보통 학습 오류는 줄어들게 된다. 따라서 $J_{\text{train}}(\theta)$ 를 여기에 적어주고, 이는 학습 오류가 우리가 입력한 데이터의 다항식 차수에 따라서 줄어드는 경향을 보이기 때문이다.

03:03

다음으로는 교차 검증 오류를 살펴보자. 이 문제는 종종, 테스트 세트 오류를 살펴보면, 마치 우리가 교차 검증 오류를 그래프화 한 것과 꽤 비슷한 결과를 얻을 수 있다. 만약 $d = 1$ 일 때, 매우 간편한 함수를 대입하는데, 그래서 학습 세트가 시부적합이 될 수 도 있다. 따라서 우리는 매우 높은 교차 검증 오류를 써야 한다. 만약 중간 정도 차수 다항식을 쓰게 되면, 이번 슬라이드 예시에서는 $d = 2$ 였는데, 이 경우 훨씬 더 낮은 교차 검증 오류를 갖게 될 것이다. 왜냐하면 데이터에 더 잘 맞는 것을 찾기 위해 대입해보는 것이기 때문이다. 반대로, 만약 d 값이 너무 높을 경우, 아까는 $d = 4$ 라고 했었는데, 그럴 경우에는 과적합되어, 결국 교차 검증 오류의 높은 값을 얻게 될 것이다. 따라서 이와 같이 부드럽게 변형을 주고 곡선을 그린다면, 이러한 커브가 그려질

것이고 이는 $J_{cv}(\theta)$ 가 되고, 또는 $J_{test}(\theta)$ 도 이와 매우 비슷하다고 할 수 있다.

04:07

이러한 종류의 그래프는 bias 와 분산 문제를 더 잘 이해할 수 있게 도와줄 것이다. 예를 들어, 학습 알고리즘이 있는데, 여러분이 원하는 만큼 돌아가지 않을 때, 학습 알고리즘에 문제가 있다는 것을 어떻게 알아낼 수 있을까? 먼저 학습 알고리즘을 적용했고, 여러분이 원하는 대로 작동하지 않는다고 가정해보자. 이는 교차 검증 세트 오류 또는 테스트 세트 오류가 높다는 뜻이다. 이 학습 알고리즘이 높은 bias 때문인지 아니면 높은 분산 때문에 문제가 생겼는지 어떻게 알아낼 수 있을까? 교차 검증 오류가 높다는 설정은 여기 regime 아니면 이 regime에 해당한다. 따라서 여기 왼쪽에 있는 regime은 높은 bias 문제에 해당하고, 이는 $d = 1$ 과 같은 낮은 차수 다항식을 대입할 경우에, 데이터에 대입할 더 높은 차수 다항식이 필요할 때이다.

04:52

이와 반대로, 이 regime은 높은 분산 문제와 관련이 있다. 만약 d 그러니까 다항식 차수가, 우리가 가진 데이터 세트에 비해 너무 클 때이다. 이러한 수치는 우리에게 두 경우를 구분할 수 있도록 힌트를 준다. 다시 말해서, 높은 bias 경우에는, 시부적합인 경우, 교차 검증 오류 와 학습 오류 둘 다 높은 경우이다. 따라서 여러분의 알고리즘에 bias 문제가 있는 거라면, 학습 세트 오류 또한 높아서 교차 검증 오류 또한 높게 되는 것이다. 비슷할 수도 있고, 학습 오류보다 약간 더 높을 수도 있다. 만약 이러한 결합을 보게 된다면, 이게 바로 여러분의 알고리즘에서 높은 bias 문제가 생겼다는 것을 말한다.

05:52

반면에, 여러분의 알고리즘에 높은 분산 문제가 생긴 것이라면, 여기를 보면, $J_{train}(\theta)$, 이는 학습 오류를 나타내는 데 이게 낮다는 것이다. 이는 학습 세트를 매우 잘 대입했다는 것이다. 반면에 교차 검증 오류는 이는 제곱 오류를 최소화 하려는 것이다. 반면에, $J_{cv}(\theta) \gg J_{train}(\theta)$ 이다. 여기 \gg 표시는 훨씬 크다는 뜻이다. 이 표시는 $>$ 보다 두 배 더 크다는 것이다. 따라서 $>$ 보다 두 배 이상 더 크다는 것을 나타내는 기호다.

06:38

따라서 이러한 결합을 보게 되면, 이러한 결합 값을 보게 된다면, 여러분의 학습 알고리즘에 높은 분산 문제가 생겨 과적합 되었다는 표시다. 따라서 이 두 경우를 구분하는 방법은 높은 bias 문제가 생겼다면, 학습 세트 오류가 높아질 것이고, 가설 또한 학습 데이터와 잘 맞지 않을 것이다. 그리고 높은 분산 문제가 생긴 것이라면, 학습 세트 오류는 보통 낮을 것이며, 교차 검증 오류보다는 훨씬 낮을 것이다.

Q) 분류 문제가 생겼다고 가정하자. 이 오분류 오류는 다음과 같이 정의되는데, 교차 검증 (오분류) 오류 또한 다음과 같은 예시를 활용하여 비슷하게 정의된다. 여러분의 학습 오류가 0.10

이고, 교차 검증 오류가 0.30 이라고 한다면, 이 알고리즘에 생긴 문제로 가장 적절한 것은?

1. 높은 bias (과적합)
2. 높은 bias (시부적합)
3. 높은 분산 (과적합)
4. 높은 분산 (시부적합)

정답 3번

07:08

이제 여러분은 bias 와 분산 두 문제에 대해 더 이해할 수 있게 되었다. Bias 와 분산에 대해 아직 해야 할 이야기가 많기 때문에 다음 강의에서 계속 이어질 것이다. 하지만 이후 강의에서는 학습 알고리즘이 high bias 인지 high 분산인지를 진단할 것이다. 이를 하는 방법을 나중에 자세히 가르쳐 줄 것이다. 학습 알고리즘이 high bias 문제인지 아니면 둘 다를 포함한 것인지를 구별 하는 것을 학습하고, 이를 통해 학습 알고리즘의 성능을 향상시키는 가능성이 있는 작업을 할 수 있도록 안내할 것이다.

No.	Title
Week 6-5	Regularization and Bias/Variance

00:00

여러분은 일반화를 통해 과적합을 막을 수 있는지 살펴보았다. 하지만 이것이 학습 알고리즘에서 bias와 분산에 어떤 영향을 끼칠까? 이번 강의에서는 bias와 분산에 대해 더 심도 있게 다뤄볼 것이며, 서로 어떻게 상호작용하고 있고, 학습 알고리즘을 일반화 하는데 영향을 주고 있는지 알아볼 것이다. 이와 같은 고차 자동 다항식을 대입한다고 하자. 하지만 과적합하는 것을 막기 위해서 이를 이와 같이 일반화해야 한다. 여기 일반화된 용어는 작은 값을 유지할 수 있게 한다. 보통 일반화는 $j=0$ 부터 m 까지 보다는 $j=1$ 에서 m 까지에서 온다. 세가지 경우를 살펴보자. 첫 번째 경우는 일반화 매개 변수 람다의 값이 매우 큰 경우다. 예로, $\lambda = 10000$ 이다. 매우 큰 값이다.

00:54

이와 같은 경우, 모든 매개 변수는 $\theta_1, \theta_2, \theta_3$ 등은 아주 많이 분류되어 있으므로, 대부분의 매개 변수는 0과 가까워 진다. 그리고 $h(\theta)x \approx \theta_0$ 이다. 따라서 이 가설은 아마도 이와 같이 평평하게 일정한 직선으로 그려질 것이다. 따라서 이 가설은 high bias 를 가지고 있고, 이 데이터 세트를 매우 시부적합하고 있다. 따라서 수평의 직선은 이러한 데이터 세트에서 그렇게 좋은 모델은

아니다. 반대의 경우에 가보면, 여기는 매우 작은 λ 값을 가지고 있다. 예를 들어 $\lambda = 0$ 이다. 이 경우에는 고차다항식을 대입한다고 하면, 보통 과적합 세팅이 된다. 그렇게 되면, 고차다항식을 대입하면, 일반화 없이 또는 최소한의 일반화를 통해, high 분산의 과적합 셋팅을 얻게 된다. 이는 $\lambda = 0$ 일 때, 우리의 일반화에 대입하여, 가설이 과적합하게 되는 것이다. 만약 중간 λ 값을 가진다면, 이는 너무 크지도 작지도 않은 매개 변수 데이터를 얻게 되어, 데이터에 적합하게 된다. 그렇다면 우리는 어떻게 일반화 매개 변수를 위한 좋은 값을 자동으로 선택할 수 있을까?

02:19

반복하자면, 여기 모델이 있고, 여기 학습 알고리즘의 객체가 있다. 우리가 일반화를 활용하는 셋팅에서, $J_{train}(\theta)$ 를 최적화 객체이지만 일반화 항 없이 좀 다르게 정의하고자 한다. 이전 강의에서 일반화를 이용하지 않았을 때에는 $J_{train}(\theta) = \text{비용함수 } J(\theta)$ 라고 정의 했었다. 하지만 일반화를 활용하여 이렇게 큰 항이 되면, $J_{train}(\theta)$, 학습 세트는 일반화는 배제한 채 학습세트에 있는 제곱 오류 또는 학습 세트에 있는 평균 제곱 오류의 합으로 정의할 수 있다. 비슷하게 교차 검증 세트 오류를 정의하여, 테스트 세트에 있는 교차 검증에서 제곱 오류의 평균 합이 되기 전에 그 오류를 테스트하여 정의를 요약할 것이다. $J_{train}, J_{cv}, J_{test}$ 의 정의를 요약하자면, 이는 평균 제곱인데, 추가 일반화 항이 없는 테스트 세트의 학습 검증에 있는 다른 제곱 기록의 절반을 의미한다. 이것이 바로 우리가 자동으로 일반화 매개변수 람다를 선택한 방법이다.

03:27

그 다음으로 람다의 다양한 값을 가지고 시도해 보려고 한다. 여기서는 일반화는 활용하지 않을 것이다. 여기에 시도해 볼 람다 값이 몇 개 있다. $\lambda = 0.01, 0.02, 0.04$ 등이 있다. 보통 여기 숫자에 2씩 곱한 값을 쓰는데, 적당히 큰 숫자까지 써준다. 이렇게 2씩 곱해주면, 이 숫자는 정확히는 10.24가 된다. 하지만 거의 가까운 수이기 때문에 괜찮다. 그리고 소수점 셋째 자리, 넷째 자리 이런 건 결과값에 큰 영향을 미치지 않는다. 따라서 여기 이렇게 12개의 모델이 있다. 이제 매개 변수 람다를 일반화한 12개 값을 선택하려고 한다. 0.01보다 작거나 10보다 큰 수를 써도 되지만, 여기서는 쉬운 예시를 사용했다. 12개의 모델을 가지고 다음과 같이 첫 번째 모델은 $\lambda = 0$ 을 가지고 $\min j(\theta)$ 를 통해 활성 데이터의 매개 변수를 얻을 것이다. 그리고 지난 강의에서와 같이, 이것을 $\theta(1)$ 이라고 표기하겠다.

04:49

다음으로 두 번째 모델은 $\lambda = 0.01$ 이고, 최소화한 cost function 또한 $\lambda = 0.01$ 이다. $\min j(\theta)$ 는 좀 다른 매개 변수 벡터 θ 를 얻기 위해 이를 $\theta(2)$ 라고 쓰겠다. 여기는 $\theta(3)$ 가 된다. 이렇게 세 번째 모델까지 구해봤다. 그리고 마지막 모델의 람다 세트는 10 또는 10.24 이고 이는 $\theta(12)$ 가 된다. 이번에는 이 가설과 모든 매개 변수를 가지고 교차 검증 세트를 활용하여 이를 검증해보자. 먼저 첫 번째 모델, 두 번째 모델, 그리고 여러 값이 일반화 매개 변수에 대입되고, 이를 교차 검증 세트를 활용해 측정하는 것인데, 이는 교차 검증 세트에 있는 제곱 벡터 매개 변수 각각의 평균 제곱 오류를 측정한 것을 기초로 한다. 또한 이 12개의 모델 중에서 교차 검증 세트에서 가장 낮은 오류를 보이는 것을 선택한다. 그럼 예시로, $\theta(5), 5$ 차 다항식을 선택하겠다.

왜냐하면 이 것이 가장 낮은 교차 검증 오류를 가지고 있기 때문이다. 그리고 나서 마지막으로는 각각 테스트 세트 오류를 보고 하고 싶다면, 내가 선택한 매개 변수 $\theta(5)$ 를 가지고 테스트 세트에서 잘 맞는지 보는 것이다. 다시 말해,

06:16

다시 말해, 여기 이렇게 매개 변수 세타를 교차 검증 세트에 대입한 것처럼, 별개의 테스트 세트를 가져와 내 매개 변수 벡터인 세타가 이전에 보지 못한 예시에 얼마나 잘 일반화 되는지 측정 할 수 있다. 이것이 일반화 매개 변수 람다를 선택하기 위해 모델 선택이 적용된 예시다. 마지막으로 교차 검증과 학습 오류를 우리가 일반화 매개 변수 람다를 달리하는 것처럼 변화를 주는 것에 대해 이해하고 넘어가도록 하겠다. 다시 한번 짚고 넘어가자면, 이것은 우리가 원래 사용한 비용함수 $j(\theta)$ 다. 하지만 변화를 주기 위해 우리는 학습 오류와 교차 검증 오류를 일반화 매개 변수를 사용하지 않고 정의해보도록 하겠다.

07:07

먼저 여기 J_{train} , J_{cv} 를 도표화해야 하는데, 이는 내 가설이 학습 세트에 잘 맞는지, 내 가설이 교차 검증 세트에서 어떻게 작동하는지를 의미한다. 일반화 매개 변수 람다를 다르게 하면, 아까 살펴봤던 것처럼 람다 값이 작으면 일반화를 많이 사용하지 않고, 과적합의 위험성이 커지게 된다. 반면에 람다의 값이 크면 여기 오른쪽 가로축에 표시한다면, 람다 값이 크면, bias 문제를 가지고 있을 위험이 높아진다.

Q) 여기 일반화된 로지스틱 회귀가 있다. J_{train} 과 J_{cv} 를 일반화 매개변수 람다의 함수로 나타낸다고 하자. 이를 가장 적합하게 표현한 그래프는?

정답 3번

07:51

따라서, J_{train} 과 J_{cv} 그래프를 그려보면, 작은 람다 값은 학습 세트에 상대적으로 잘 대입되는 것을 발견할 수 있는데, 이는 일반화를 하지 않았기 때문이다. 따라서 여기 이 부분, 일반화 항은 없어지고, 여기 제곱 오류로만 최소화한다. 따라서 람다 값이 작으면, J_{train} 값도 작아지게 된다. 반면에 람다 값이 크면 bias 문제도 커지게 되므로 이는 학습 세트에 잘 맞지 않게 되므로, 그 값은 이쯤이 된다. 따라서 $J_{train}(\theta)$ 는 람다 값이 증가할 때 따라 증가하는데, 람다값이 크면 high bias 와 대응하고, 학습 세트와 잘 맞지 않게 된다. 반면에 람다 값이 작으면 이는 데이터에 있는 고차다항식과 매우 잘 맞는다. 교차 검증 오류의 경우에는 이와 같은 수치를 얻게 된다. 여기 오른쪽은 람다 값이 큰데, 이는 과적합 될 수 있으므로 이것은 bias regime 이다. 그래서 교차 검증 오류가 높을 것이다. 여기에 $J_{cv}(\theta)$ 라고 써주자 high bias 일 때, 교차 검증 세트와 잘 맞지 않기 때문이다.

09:16

반면에 여기 왼쪽은, high 분산 regime 인데, 더 작은 두 값이 있으므로 데이터에 과적합 될 것이다. 데이터가 과적합되면, 교차 검증 오류는 높아진다. 이것이 교차 검증 오류이고, 학습 오류인데 일반화 매개변수 람다에 변화를 주면 학습 세트에서 이렇게 나타날 것이다. 다시 말하지만, 람다의 중간 값이 있는데 이는 'just right' 또는 가장 잘 맞는다고 표현하는데, 교차 검증 오류가 적고 또는 테스트 세타가 작다. 반면에 여기 그린 곡선은 어떻게 보면 만화 같고, 이상화된 것 같은데, 실제 데이터에서 얻게 되는 곡선은 이것보다는 좀 더 지저분하게 보일 수도 있다. 어떤 데이터 세트에서는 이러한 경향을 자주 보게 될 텐데, 교차 검증 오류가 지속되는 그래프를 보면서 이를 수동 또는 자동으로 교차 검증 오류를 최소화할 수 있는 지점을 선택하거나, 낮은 교차 검증 오류에 대응하는 람다 값을 선택할 수 있을 것이다.

10:29

학습 알고리즘에서 일반화 매개 변수 람다를 선택하려고 할 때, 이와 같이 수치를 그래프화 해보면 어떻게 진행이 되고 있는지 더 잘 이해할 수 있게 되고, 내가 일반화 매개 변수 모니터를 위해 좋은 값을 설정했다는 것을 확인할 수 있게 된다. 이번 강의를 통해 일반화와 일반화가 학습 알고리즘에서 bias 와 분산에 미치는 영향에 대해 더 잘 이해할 수 있는 계기가 되었으면 한다. 지금까지 여러분은 다양한 형태의 bias 와 분산에 대해 살펴봤다. 다음 번 강의에서는 지금까지 배운 것을 토대로 학습 곡선이라는 진단 프로그램에 집에 넣어볼 것이다. 이 학습 곡선은 학습 알고리즘이 bias, 분산, 또는 두 문제 다 가지고 있는지 진단할 때 자주 활용된다.

No.	Title
Week 6-6	Learning Curves

00:00

이번 강의에서는 학습 곡선에 대해 이야기하고자 한다. 학습 곡선은 그래프화 하는데 매우 유용하다. 여러분의 알고리즘이 정확하게 돌아가고 있는지 확인하고 싶거나, 알고리즘의 성능을 향상시키고 싶을 때 활용 가능하다. 학습 곡선은 실제 학습 알고리즘이 bias, 분산, 또는 두 문제 다 가지고 있는지 진단할 때 내가 자주 활용하는 툴이다.

00:27

여기에 학습 곡선이 있다. 학습 곡선을 그래프로 나타내기 위해서, 나는 보통 $j \text{train}(\theta)$ 를 활용하는데, 이는 학습 세트에 있는 평균 제곱 오류이고, 또는 $J_{cv}(\theta)$ 는 교차 검증 세트에 있는 평균 제곱 오류이다. 이를 함수 m 이라고 표시하고, m 은 내가 가지고 있는 학습 예시의 수를 가리킨다. 따라서 m 은 보통 정수인데, 100개의 학습 예시가 있지만 인위적으로 학습 세트 예시의 수를 줄일 것이다. 따라서 의도적으로 예를 들어, 10개, 20개, 30개 또는 40개 정도의 학습 예시만을 사용할 것이고, 학습 오류와 교차 검증이 무엇인지 이 작은 학습 세트 예시에 그래프로 나타내 보겠다. 그렇다면 이를 그래프화 해보자. 이와 같은 학습 예시가 하나 있을 때, 2차 함수를

적용한다고 가정하자.

01:21

학습 예시가 하나이기 때문에 완벽하게 들어 맞을 것이다. 2차 함수를 적용하면 되니까, 학습 예시에서 오류는 0이 될 것이다. 학습 예시가 2개가 되면, 2차 함수는 역시 잘 맞는다. 일반화를 활용한다고 해도, 아마 잘 들어 맞을 것이다. 뉴럴 일반화를 활용한다면, 완벽하게 들어 맞을 것이다. 이렇게 학습 예시가 세 개가 되면, 2차 함수는 잘 맞게 된다. 만약 $m=1$, 또는 $m=2$, 또는 $m=3$ 일 경우에는 학습 세트에서 학습 오류는 일반화를 사용하지 않는다고 하면 0이 될 것이다. 물론 일반화를 사용한다고 해도 0보다 조금 큰 정도이다. 학습 세트가 커지면, $j \text{ train}(\theta)$ 를 적용하기 위해 인위적으로 학습 세트 사이즈를 제한한다.

02:13

여기에 $m=3$ 대입하면, 3개의 예시만으로 학습하는 것인데, 그렇게 되면 이 수치는 내 데이터에 맞는 3개의 예시를 가지고만 학습 오류를 측정하게 되는 것이다. 따라서 100개의 학습 예시가 있는데, $m=3$ 일 때, 학습 오류를 그래프화하고 싶으면 가설 2에 맞는 예시 세 개에 있는 학습 오류를 측정하면 되는 것이다. 그리고 학습 과정에서 다른 모든 예시를 임의로 삭제한 것은 아니다.

02:45

요약해보면, 학습 세트 사이즈가 작으면, 학습 오류도 같이 작아진다. 작은 학습 세트는 학습 세트에 매우 쉽게 거의 완벽하게 잘 들어맞기 때문이다. 이제 $m=4$ 예시를 살펴 보자. 2차 함수가 이 데이터 세트에 완벽하게 들어 맞는 것을 볼 수 있다. $m=5$ 인 경우에도 2차 함수가 이렇게 되고, 학습 세트가 점점 커지는 것을 볼 수 있다. 이제 점점 예시에서 2차 함수가 완벽하게 돌아가고 있다고 확신하기가 점점 더 어려워 진다. 따라서, 학습 세트 사이즈가 커질수록 평균 학습 오류가 실제로 증가하고, 이러한 수치를 그래프화하면 학습 세트 오류는, 이는 가설에서 평균 오류를 나타내는데, m 값이 증가함에 따라 함께 증가한다. 다시 말해, m 값이 작아지면, 즉 학습 예시가 많지 않을 때, 학습 예시 하나하나를 완벽하게 대입할 수 있기 때문에 오류도 적을 것이다. 반면에 m 값이 커지게 되면, 모든 학습 예시를 완벽하게 대입하기 어려워 지므로 학습 세트 오류가 더 커지게 된다.

04:03

이번에는 교차 검증 오류를 살펴보자. 교차 검증은 처음 보는 교차 검증 세트에서 오류를 의미하는데, 학습 세트가 매우 작을 때, 이를 일반화하지는 않는다. 따라서 여기 이 가설들은 좋은 가설처럼 보이지 않는다. 이와 같이 학습 세트의 수가 커지면, 데이터가 더 잘 맞는 가설을 얻기 시작한다. 따라서 교차 검증 오류와 테스트 세트 오류는 학습 세트 사이즈가 커질수록 줄어드는 경향을 보이는데, 데이터가 많을수록, 새로운 예시에 더 잘 일반화되기 때문이다. 데이터가 많아질수록 가설이 더 잘 들어맞게 되는 것이다. 따라서 $j \text{ train}(\theta)$ 와 $J_{cv}(\theta)$ 를 그래프화하면 이와 같은 그래프를 얻을 수 있다. 이번에는 학습 곡선이 어떻게 생겼는지

살펴보고 high bias 또는 분산 문제를 가지고 있는지 알아보자.

04:58

여러분의 가설에 high bias가 있다고 가정하면, 이를 설명하기 위해 이 함수를 활용해 직선을 데이터에 대입하려고 하면, 그렇게 잘 맞는 것은 아님을 알 수 있다. 가설은 이렇게 직선을 그려줄 수 있다. 여기서 학습 세트 사이즈를 증가시키면 어떻게 되는지 보자. 여기 이렇게 예시가 5개 있는 것 대신 학습 예시가 훨씬 더 많다고 가정해보자. 여기에 직선을 이렇게 그려보면, 보다시피 이는 아까와 거의 똑같은 직선이다. 그러니까 이 데이터에 직선은 그렇게 잘 맞지 않아 보인다. 데이터가 훨씬 더 많아진다고 해도 직선이 잘 맞게 되지는 않을 것이다. 이게 아마 이 데이터에 최선으로 맞출 수 있는 직선일 것이다. 하지만 직선은 이 데이터 세트에 잘 맞지 않음을 알 수 있다. 따라서 교차 검증 오류를 그려보면, 이와 같을 것이다.

05:51

여기 왼쪽 옵션은 극소의 학습 세트 사이즈를 가지고 있다고 할 때, 학습 예시 하나만 가지고는 충분하지 않아 보인다. 하지만 학습 예시가 특정한 수에 다 다르게 되면, 거의 직선의 형태를 만족시키는 것을 알 수 있는데, 이쯤에서 학습 예시 사이즈가 더 많아진다고 해도, 즉 m 값이 훨씬 커진다고 해도 아까와 비슷한 직선의 형태를 보인다. 그러므로 교차 검증 오류는, 이렇게 써주고, 또는 테스트 세트 오류 또는 학습 예시가 특정한 수에 다 다르게 되면, 안정기에 접어들어서 평평한 상태를 유지하게 되고, 직선과 가장 비슷하게 된다. 이번에는 학습 오류에 대해 알아보자. 학습 오류는 작게 시작하는데, high bias 경우에는 학습 오류는 교차 검증 오류와 가까워지는데, 매개 변수는 많지 않고, 데이터는 많아서, m 값이 크기 때문이다. 학습 세트와 교차 검증 세트에서의 성능은 매우 비슷하다. 이것이 바로 학습 곡선의 그래프이자 high bias를 가진 알고리즘이다.

07:00

그리고 high bias 문제는 교차 검증 오류와 학습 오류가 높다는 사실로부터 반영된다. 따라서 상대적으로 높은 J_{cv} 와 J_{train} 값을 가지게 된다. 이는 또한 매우 재미있는 사실을 암시하는데, 만약 학습 알고리즘이 high bias를 가지고 있다면, 더 많은 학습 예시가 있을수록, 그러니까 이 수치가 오른쪽으로 가게 될수록, 교차 검증 오류가 아래로 많이 떨어지지 않고, 수평을 이뤄서, 이 학습 알고리즘에 high bias 문제를 가지고 있다는 것을 알게 된다. 그래서 학습 데이터의 수가 많아진다고 해서 도움이 되지 않는다는 것이다. 또한 오른쪽에 있는 수치 예시는 학습 예시가 다섯 개뿐이었고, 이렇게 직선을 그려줬고, 학습 데이터의 수가 이렇게 많아지게 되고, 거의 같은 직선의 형태를 보여준다. 따라서 학습 알고리즘이 high bias를 가질 경우, 학습 데이터의 수가 많아진다. 이는 훨씬 낮은 교차 검증 오류 또는 테스트 세트 오류를 얻는 데는 도움이 되지 않는다. 따라서 학습 알고리즘에 high bias를 가진 경우를 알아두면 유용한데, 더 많은 학습 데이터를 모으는데 시간을 낭비하는 것을 막아주기 때문이다. 이는 결국 별 도움이 안될 것이기 때문이다.

08:15

이번에는 high 분산 문제를 가진 학습 알고리즘 설정을 살펴보겠다. 여기 학습 오류를 먼저 살펴보자. 이처럼 오른쪽에는 다섯 개의 예시를 가진 매우 영리한 학습 세트가 있다. 매우 고차 다항식을 적용한다고 하면, 100차 다항식을 적어놓긴 했지만, 실제로는 사용하지 않을 것이고 예시로 적어놓은 것이다. 만약 매우 작은 λ 값을 사용한다면, 0은 아닌 작은 λ 값이면, 이를 과적합하는 함수로 이 데이터에 매우 잘 들어맞을 것이다. 따라서 학습 세트 사이즈가 작으면, 학습 오류인 $J_{\text{train}}(\theta)$ 도 작아진다.

09:03

이 학습 세트 사이즈가 증가할수록, 이 데이터를 여전히 과적합하고 있을지도 모르지만, 조금씩 이 데이터 세트에 완벽하게 맞추기가 힘들어진다. 그러니까 학습 세트 사이즈가 증가함에 따라, $J_{\text{train}}(\theta)$ 값도 증가하게 됨을 알 수 있다. 이는 예시가 많아질수록 학습 세트에 완벽하게 맞기가 어려워지기 때문인데, 학습 세트 오류는 여전히 꽤 낮은 편이다. 이번에는 분산 검증 오류를 알아보자. High variance 상태에서, 이 가설은 과적합 상태이므로 학습 예시의 수가 적절하다고 하더라도 교차 검증 오류는 높아져 있을 것이다. 따라서 교차 검증 오류는 이와 같을 것이다. 따라서 high variance 문제가 있을 때 나타나는 현상은 학습 오류와 교차 검증 오류 간에 차이가 크게 나타난다는 것이다.

09:57

이 그래프를 보면 더 많은 학습 데이터를 추가하는 것을 생각해보면, 이 수치를 가지고 오른쪽으로 점점 이어지게 되는데, 여기 파란 곡선과 분홍색 곡선이 있는데, 이 두 곡선이 서로 만나는 것을 알 수 있다. 오른쪽에 있는 이 수치를 추론해보면, 학습 오류는 이렇게 위로 올라갈 것이고, 교차 검증 오류는 이렇게 점점 내려올 것이다. 여기서 주목해야 할 것은 교차 검증 오류 또는 테스트 세트 오류다. 이러한 수치를 보면 학습 예시를 계속 추가하여 오른쪽 내용을 추론하게 되면, 교차 검증 오류는 계속 줄어들 것 이다. 따라서 high variance 상태에서 학습 데이터를 더 얻는다는 것은 도움이 될 수도 있다. 다시 말해, 학습 알고리즘이 high variance 문제를 가지고 있다는 것을 알고 있는 것이 중요한데, 이는 학습 데이터를 더 얻어야 할 가치가 있는지를 보고 판단할 수 있기 때문이다.

11:03

이번 슬라이드와 이번 슬라이드에서 꽤 깔끔한 곡선을 그렸었다. 하지만 실제 학습 알고리즘을 이러한 곡선으로 나타내면, 이와 같은 곡선의 형태로 그려질 것이다. 때로는 여기 곡선보다 더 지저분하고 정신 없는 곡선이 그려질 수도 있다. 하지만 이와 같이 학습 곡선을 그래프에 나타내는 것은 학습 알고리즘이 bias 또는 variance 또는 둘 다 문제를 약간씩 가지고 있는지를 이해하는데 도움을 준다. 따라서 학습 알고리즘의 성능을 향상시키기 위해서는, 나는 항상 이러한 학습 곡선을 그려보는 방법을 택한다. 그러면 bias 또는 variance 둘 중에 어떤 문제가 생긴 건지 쉽게 알아 볼 수 있게 된다.

Q) 다음 중 어떤 상황에서 학습 알고리즘의 성능 향상에 도움이 될 만한 많은 학습 데이터를 얻게 되는가?

1. High bias 상태의 알고리즘
2. High variance 상태의 알고리즘
3. $J_{cv}(\theta)$ 가 $J_{train}(\theta)$ 보다 훨씬 클 때
4. $J_{cv}(\theta)$ 가 $J_{train}(\theta)$ 과 거의 비슷할 때

정답 2,3번

11:44

다음 번 강의에서는 이것이 학습 알고리즘의 성능을 향상시키기 위해 어떻게 특정한 옵션을 취할 것인지 취하지 않을 것인지를 제시하는지 살펴보도록 하겠다.

No.	Title
Week 6-7	Deciding What to Do Next Revisited

00:00

지금까지 학습 알고리즘을 어떻게 평가하는지, 모델 선택 그리고 bias와 variance에 대해서도 이야기 해보았다. 그렇다면 이러한 것들이 학습 알고리즘의 성능을 향상시키기 위해 잠재적으로 유익한 것인지 아닌지를 구분하는 것을 어떻게 도와줄 것인가. 이전에 활용했던 예시로 돌아가 그 결과를 확인해보자. 여기 이전에 활용한 예시가 있다. 이를 regularization된 선형 회귀에 대입하려고 하는데, 우리가 원하는 것처럼 잘 되지 않는다고 가정하자. 그렇다면 이렇게 다양한 옵션이 있었는데, 이중에서 어떤 것이 유익한 옵션인지를 알아낼 방법은 무엇일까? 첫 번째 옵션은 학습 예시를 더 찾는 것이다. 이는 high variance 상태를 해결해준다. 예를 들어, high bias 문제가 있고, variance 문제는 없을 때, 이전 강의에서 학습 예시를 더 찾았는데, 그다지 도움은 되지 않았다. 따라서 학습 곡선을 그려서 최소한 variance 문제가 약간 있는데, 이는 교차 검증 오류를 의미하고 이것이 학습 세트 오류보다 꽤 큰 경우에만 첫 번째 옵션은 유익하게 된다. 두 번째 features 수 줄이기는 어떨까? Features 수를 줄이면, high variance 문제를 해결해준다.

01:17

다시 말해, 학습 곡선이나 여러분이 활용한 무언가를 참고하여 여기에 high bias 문제가 있다는 것을 알아낸다면, 제발 더 적은 features 세트를 선택하여 활용하려는데 시간을 낭비 하지 말자. 왜냐하면 high bias 문제가 있을 때는, features 수가 적어지면 도움이 되지 않기 때문이다. 반면에,

학습 곡선이나 여러분이 활용한 무언가를 참고하여 high bias 문제의 해결 방안 발견한다면, 더 적은 features 세트를 선택하도록 해서 시간을 낭비하지 않게 될 것이다. 다음으로 features를 추가하는 것은 어떠할까. 항상은 아니지만 보통 이것은 high bias 문제의 해결 방안으로 여겨진다. 따라서 feature를 추가하게 되는 경우는, 현재 가설이 너무 간단해서 feature를 추가해서 가설이 학습 세트에 더 잘 들어맞기를 원할 때이다. 다음으로는 고차항 추가 방법이 있다. 아까 feature를 추가하는 것의 또 다른 방법인데, high bias 문제를 해결하려는 방법이다.

02:21

예를 들어, 학습 곡선에서 여전히 high variance 문제를 가지고 있다고 하면, 이와 같은 방법을 활용하는 것은 시간을 잘 활용하지 못하는 것이다. 마지막으로 람다 값을 줄이거나 늘리는 것이다. 이 작업은 매우 빠르고 쉽게 시도할 수 있다. 여러분의 수개월을 낭비하지 않아도 되기 때문이다. 하지만 람다값을 줄이는 것은 high bias 문제 해결한다는 것을 이미 배운 바 있다. 혹시 잘 이해가 가지 않는 경우, 강의 영상을 잠시 정지하고 람다 값을 줄이면 high bias 문제를 해결하고 반면에 람다 값을 늘리면 high variance 문제를 해결한다는 것을 확실히 짚고 넘어가자. 이해가 가지 않는다면 영상을 정지하고, 이러한 케이스에 대해 확실히 알고 넘어가야 한다. 아니면, 지난 강의 후반부에서 우리가 그린 곡선을 다시 살펴보고 왜 이렇게 되는 건지 이해하기를 바란다.

03:15

이번에는 우리가 배운 모든 것을 신경망에 적용해보자. 여기에서 내가 어떻게 보통 신경망의 아키텍쳐나 연결 패턴을 선택하는 지에 대한 실용적인 조언을 해주겠다. 신경망에 대입할 때, 한 가지 옵션은 매우 작은 신경망으로 하는 것인데, 상대적으로 적은 은닉 계층, 아마 하나 정도 있을 것이다. 신경망에 대입할 때, 한 가지 옵션은 매우 작은 신경망으로 하는 것인데, 상대적으로 적은 아마 은닉 계층 하나, 아니면 상대적으로 적은 수의 은닉 계층을 가지고 있을 것이다. 따라서 이와 같은 네트워크는 상대적으로 적은 매개 변수를 가지고 있으므로 시부적합일 가능성이 있다. 이와 같이 작은 신경망의 장점은 연산 비용이 매우 저렴하다는 것이다.

04:05

또 다른 방법은 상대적으로 큰 신경망에 대입하는 것인데, 은닉 계층이 많아지거나, 여기에는 더 한 계층에 더 많이 있고, 여기에는 은닉 계층이 더 늘어나 있다. 그리고 이러한 신경망은 매개 변수가 더 많아지고, 과적합하기 더 쉬워진다. 단점 중 하나는 크게 문제되지는 않지만 생각해 볼 필요가 있는데, 네트워크에 뉴런의 수가 많아지면, 연산 비용이 비싸지게 된다. 하지만 이는 그렇게 큰 문제가 되지는 않을 것이다. 이렇게 큰 신경망에서 잠재적인 가장 큰 문제는 과적합되기 쉽다는 점이다. 따라서 신경망을 활용할 때, 보통 더 큰 신경망을 활용하기 때문에, 실제로 클수록 활용도가 높지만, 과적합되기 쉬워진다. 그럴 때는 과적합을 다루기 위해 regularization을 활용할 수 있다. 그리고 보통 regularization을 활용하여 과적합을 다루기 위해 더 큰 신경망을 활용한다. 이것은 더 작은 신경망을 활용했을 때 보다 효율적이다. 또 다른 단점은 연산 비용이 비싸질 수 있다는 것이다.

05:10

마지막으로 또 결정해야 할 것 중 하나는, 은닉 계층의 수를 몇 개로 할 것인지 정하는 것이다. 1개의 은닉 계층을 할 것인지 3개를 할 것인지 여기에 나타나 있는 것처럼, 아니면 2개를 원하는지 정해야 한다. 이전 강의에서 말했듯이, 보통 한 개의 은닉 계층이 이상적으로 생각되는데, 여러 개의 은닉 계층을 선택하고 싶다면, 학습, 교차 검증, 그리고 테스트 세트로 나누어서 학습 신경망을 하나의 은닉 계층, 또는 두 개 또는 세 개에 적용해 보는 것이다. 그리고 어떠한 신경망이 교차 검증 세트에서 가장 잘 돌아가는지 보는 것이다. 따라서 은닉 계층이 각각 하나, 둘, 세 개 있는 신경망에서 J_{cv} 교차 검증 오류와 다른 것을 모두 연산해보고, 어떤 신경망이 가장 적합한지를 선택하면 된다.

Q) 은닉 계층이 하나 있는 신경망을 학습 세트에 대입한다고 가정하자. 여기서 교차 검증 오류가 학습 오류보다 훨씬 크다는 것을 발견했다. 은닉 계층의 수를 늘리는 것이 도움이 되는가?

1. 그렇다, 매개 변수의 수가 늘어나 네트워크가 더 복잡한 함수를 표현할 수 있기 때문이다.
2. 그렇다, 현재 **high bias** 상태이기 때문이다.
3. 아니다, 현재 **high bias** 상태이기 때문에 은닉 계층을 추가하는 것은 도움되지 않는다.
4. 아니다, 현재 **high variance** 상태이기 때문에 은닉 계층을 추가하는 것은 도움되지 않는다.

정답 4번

06:02

지금까지 bias 와 variance, 학습 곡선을 가지고 문제를 진단하는 것을 살펴봤다. 그리고 어떠한 것이 학습 알고리즘의 성능을 향상시키는데 도움이 되고 아닌지를 확인해봤다. 지난 강의에 있던 내용을 이해해서 이를 효과적으로 적용해볼 수 있을 것이고, 문제에 대한 학습 알고리즘을 가지고, 큰 부분도, 여기 실리콘밸리에서 직업으로 기계 학습을 다루고 있는 대부분의 사람들처럼 다룰 수 있게 될 것이다. 이러한 bias 와 variance, 학습 곡선에 대한 진단 조언이 더 효과적이고 영향력 있는 알고리즘을 적용하고 잘 구현될 수 있도록 해줄 것이다.

No.	Title
Week 6-8	Prioritizing What to Work On

00:00

이번 강의부터는 기계 학습 시스템 디자인에 대해서 이야기하고자 한다. 먼저 복잡한 기계 학습 시스템을 디자인하는데 마주할 수 있는 주요 이슈에 대해서 소개한 다음에, 복잡한 기계 학습

시스템 디자인 전략에 대한 조언을 해주려고 한다. 이후 강의들이 연결되지 않는다는 느낌을 받을 수도 있는데, 이는 복잡한 학습 시스템을 디자인하는데 발견하게 되는 다양한 이슈를 다룰 것이기 때문이다. 이 후 강의들이 덜 수학적으로 보일지라도, 이 자료가 큰 기계 학습 시스템을 설계할 때, 매우 유용하고, 잠재적으로는 많은 시간을 절약할 수 있게 도와줄 것이다. 먼저 어떤 작업에 시간을 투자해야 하는지 전략을 세우는 것부터 시작하고자 한다. 먼저 스팸 분류 예시로 시작하고자 한다.

00:54

스팸 메일 분류기를 만들려고 있다고 하자. 여기 명백한 스팸 메일과 스팸이 아닌 메일 예시가 있다. 왼쪽에 있는 것은 무언가를 팔려는 메일이다. 그리고 스파머들이 일부러 스펠링 오타를 내는 것을 볼 수 있는데, medicine 에 1이 들어있고, 여기 mortgage도 그렇다. 그리고 오른쪽은 스팸이 아닌 메일 예시인데, 이는 사실 내 남동생이 보낸 것이다. 스팸 메일과 스팸이 아닌 메일의 학습 세트에 각각 $y = 1$ 또는 $y = 0$ 이라고 표기했다고 한다면, 지도 학습을 활용하여 이들을 구별하기 위한 분류기를 어떻게 생성할 수 있을까? 지도학습을 적용하기 위해서 첫 번째로 선택해야 할 것은 $x =$ 이메일의 feature이라고 표현하는 것이다. 이제 학습 세트에서 feature x 와 라벨 y 를 가지고 예를 들어 로지스틱 회귀를 활용하여 분류기를 학습시킬 수 있다.

01:56

여기에 이메일의 feature set를 선택하는 방법이 있다. 아마도 스팸과 논스팸을 구분하기 위한 지표로 수백 개의 리스트를 생각해 낼 수 있을 텐데, 예를 들어, 이메일에 'deal'이라는 단어가 포함되어있으면, 스팸일 가능성이 있다. 'buy'라는 단어도 스팸일 수도 있고, 'discount' 도 스팸일 수 있다. 반면에 내 이름 'Andrew' 가 포함되어있으면, 이것을 보낸 사람이 나를 알고 있으므로 스팸일 확률이 적다. 또한 'now' 라는 단어도 논스팸일 수 있는데, 보통 긴급 이메일을 많이 받기 때문이다. 이와 같이 수 백 개의 단어를 선택할 수 있다.

02:42

여기 이메일이 하나 있는데, 이것을 가지고 feature 벡터로 다음과 같이 부호화할 수 있다. 이 수백 개의 단어 목록을 가지고 알파벳 순서로 정리하겠다. 꼭 순서대로 할 필요는 없다. 여기 단어 목록이 있다. Discount, 그리고 now까지. 그리고 오른쪽과 같은 이메일을 가지고 이러한 단어가 이메일에서 등장하는지 확인해 볼 것이다. 그리고 여기 오른쪽 이메일을 feature 벡터 x 로 정의할 것이다. 내 이름은 나오지 않았으니 0이다. Buy는 여기 있으니 1이 된다. 그렇게 1 또는 0값을 가지게 된다. Buy는 두 번 나왔지만 1이라고 한다. 단어가 몇 번이나 등장했는지는 셈하지 않을 것이다.

03:37

'deal'도 있으니 1이 된다. 'discount'는 이렇게 짧은 이메일에는 없으니 0이고, now는 있으니 1이 된다. 따라서 이 feature 벡터에서는 단어가 있는지 없는지에 따라 1 또는 0 값이 주어진다. 그리고 이 예시에서 이 feature 벡터는 내가 단어 백 개를 선택해서 표현하는 데 활용할 경우

100차원 벡터가 된다. 따라서 feature X_j 는 특정단어 j 가 등장하면 1, 등장하지 않으면 0 값을 가진다.

04:25

이렇게 이메일을 가지고 feature 표현을 해봤다. 이러한 과정을 직접 손으로 백 개의 단어를 선택하는 것을 보여줬지만, 실제로 가장 많이 쓰는 방법은 학습 세트를 살펴보고, 그 안에서 가장 자주 등장하는 n 단어를 서술하면 된다. 여기서 n 은 10,000에서 50,000 개인데, 이를 features로 활용한다. 따라서 손으로 단어 백 개를 선택하는 것 보다는 여기 학습 예시를 살펴보고 10,000에서 50,000 개 단어 중에서 가장 자주 등장하는 단어를 선택해서, 이 것들이 features를 구성하고, 이를 스팸 메일 분류기를 표현하는데 활용하면 된다. 이제 스팸 메일 분류기를 생성할 때, 만나게 될 문제 중 하나는 여러분의 스팸 메일 분류기의 정확도를 높이고, 오류를 줄이기 위해서 시간을 어떻게 잘 활용해야 하는가이다.

05:18

먼저 당연히 많은 데이터를 수집해야 한다. 이렇게 생각하는 경향이 있지만, 많은 데이터가 있으면 더 나은 알고리즘을 얻을 수 있다. 하지만 스팸 메일 도메인에서, Honey Pot Projects라는 매우 진지한 프로젝트가 있는데, 여기서 가짜 이메일 주소를 생성해서 이를 스파머들에게 넘겨주어서 가짜 이메일 주소로 수많은 스팸 메일을 받게 되면, 학습 알고리즘을 학습시키는 많은 스팸 데이터를 얻을 수 있게 된다. 하지만 이전 강의에서 자주 봤듯이, 많은 데이터를 수집하는 것이 도움이 되기는 하지만, 항상 그렇지는 않다. 하지만 대부분 기계 학습 문제에서, 성능을 향상시키기 위해 할 수 있는 다양한 방법이 있다. 스팸 메일의 경우에는 이메일에서 더 복잡한 feature를 개발하는 것이다. 이메일 라우팅 정보에 기반해서 말이다. 이러한 정보는 이메일 헤더에 나와있다.

06:13

따라서 스파머가 이메일을 보내면 그들은 종종 이메일의 출처를 모호하게 할 것이고, 가짜 이메일 헤더를 사용하거나 찾아보기 힘든 컴퓨터 서비스 세트를 통해 이메일을 보낼 것이다. 여러분에게 스팸 메일을 보내기 위해 훈치 않은 루트를 사용하는 것이다. 이러한 정보는 이메일 헤더에 반영되어 있다. 또한 이메일 헤더를 보고, 이러한 스팸을 구별하기 위한 이메일 라우팅 정보 종류를 알아내기 위해 더 복잡한 feature를 개발하려고 노력한다. 또 다른 방법은 이메일 메시지 내용을 보는 것이다. 이는 이메일 본문을 말하는데, 이를 통해 더 복잡한 feature를 개발할 수 있다. 예를 들어, 'discount'와 'discounts'는 같은 단어로 취급해야 하는지, 'deal'과 'dealer'는 같은 단어로 봐야 하는지 등이 있다. 소문자와 대문자를 구분하는 것도 있다. 또한 문장 부호에 대해서도 생각해 볼 수 있는데, 예를 들어 스팸 메일에서 느낌표를 많이 사용하고 있기 때문일 수도 있다. 그리고 같은 방식으로, 아까 이메일에서 mortage, medicine, watches 등 일부러 스펠링 오류를 만들어 낸 것을 감지하는 것 같은 더 복잡한 알고리즘을 개발하고 싶을 수도 있다.

07:25

왜냐하면 스파머들은 실제로 watches에 4를 집어 넣는 것처럼 이렇게 메일을 보내고 있다. 이러한 간단한 기술로 스팸 메일 분류기는 watches와 w4tches 를 동일시 하지는 않을 것이다. 그러므로 일부러 스펠링 오류를 낸 것을 스팸이라고 감지하는 것은 어려운 작업이다. 그래서 스파머들이 이를 주로 활용한다. 이처럼 기계학습 문제를 다룰 때, 다양한 방법을 브레인스토밍해서 시도해볼 수 있다.,

Q) 다음 중 올바른 내용을 모두 고르시오.

1. 어떠한 학습 어플리케이션에서는 다양한 features (예시, 이메일 내용 feature, 이메일 라우팅 feature)를 생각해내는 것이 가능하다. 하지만 어떠한 feature가 가장 도움이 될지를 미리 예측하는 것은 어렵다.
2. 스팸 분류에서 고의로 낸 스펠링 오류를 감지하고 교정하는 알고리즘은 정확도를 엄청나게 향상시킬 것이다.
3. 스팸 분류는 고차원 벡터(feature가 50,000 단어가 있는지 없는지를 작업할 때, $n=50,000$ 이 된다.) 를 활용하기 때문에, 많은 양의 학습 데이터를 수집하기 위해 노력하는 것은 좋은 방법이다.
4. 학습 시스템의 정확도를 높이기 위해 다양한 개발 방법들이 있는데, '직감'을 활용하는 것은 추천되지는 않는다.

정답 1,4번

나 또한 스팸 문제를 한동안 연구했었다. 나도 꽤 많은 시간을 투자했다. 하지만 스팸 문제를 이해하여 이에 대해 알고 있기는 하지만, 이러한 네 가지 옵션 중에서 어떤 것이 도움이 될지를 말해주기는 어렵다. 사실 너무도 자주 리서치 그룹 또는 product 그룹이 이러한 옵션 중 하나를 임의로 선택해서 파고 듦다. 그리고 때로는 이러한 옵션이 시간을 가장 효율적으로 보낼 수 있는 방법은 아닐 때도 있는데, 어떤 사람은 그저 임의로 한 옵션만 파고들 수도 있기 때문이다. 사실 시도해볼 다른 옵션을 브레인스토밍 하는 단계에서, 이미 굴곡을 앞에 두고 있을지도 모른다. 안타깝게도 시도해볼 옵션의 목록을 생각해보는 대신에, 대부분의 사람들은 어느 날 아침에 일어나서 어떤 이유에서인지 이상한 직감으로 "큰 honeypot 프로젝트를 살펴보면서 데이터를 더 많이 수집하자." 라고 하면서 이상한 이유로 어느 날 갑자기 아침에 일어나서 한 가지에 꽂혀서 이를 6개월동안 붙잡고 있다.

09:03

하지만 우리는 더 잘 할 수 있다. 다음 번 강의에서 오류 분석의 개념과 많은 다양한 옵션 중에서 선택할 수 있는 체계적인 방법을 이야기하고자 한다. 그리고 그 후 몇 주, 또는 몇 일, 몇 달 동안을 잘 활용할 수 있는 선택을 할 수 있게 될 것이다.

No.	Title
Week 6-9	Error Analysis

00:00

지난번 강의에서 기계 학습 문제를 발견했을 때 알고리즘을 향상시키기 위한 다양한 방법이 있다는 것을 이야기했다. 이번 강의에서는 오류 분석의 개념에 대해 알아보자. 이러한 결정을 내리는 데 좀 더 체계적인 방법을 알려줄 것이다. 만약 기계 학습 문제를 다루기 시작하거나, 기계 학습 어플리케이션을 생성하려고 할 때, 복잡한 feature 가 있는 매우 복잡한 시스템을 생성하지 않는 것이 좋은 연습이라고 여겨진다. 따라서 처음에는 빨리 구현할 수 있는 매우 쉬운 알고리즘을 생성하는 것부터 시작해보자. 기계 학습 문제를 해결할 때, 나는 보통 하루 정도 시간을 말 그대로 24시간 정도 무언가를 매우 빠르게 얻어내려고 한다. 사실 전혀 복잡한 시스템이 아니고 그저 빠르게 생각해내려고 하고, 이를 구현한 다음에 교차 검증 데이터에서 테스트 해본다.

00:55

그 다음에는 학습 곡선을 그린다. 이 것은 지난 강의에서 이야기 했던 내용이다. 학습 오류와 테스트 오류 학습 곡선을 그려서 학습 알고리즘에 high bias 또는 high variance 또는 기타 문제가 있는지 살펴본다. 그리고 이를 활용해 데이터나 features가 더 필요한지 도움이 될지 결정한다. 이러한 접근은 꽤나 좋은 방법인 이유는 기계 학습 문제 해결을 그냥 시작해버리면, 이를 복잡한 feature가 더 필요한지, 데이터가 더 필요한지 아니면 다른 무언가가 더 필요한지를 미리 알아볼 방법은 실제로 없기 때문이다. 그리고 자료가 없는 상태에서, 학습 곡선도 없이 이를 미리 알아보는 것은 매우 어렵다. 어떤 옵션에 시간을 투자해야 하는지를 결정하는 것은 매우 어려운 작업이다. 또한 매우 빠르게 단순한 구현을 해보는 것, 그리고 학습 곡선을 그려보는 것이 이러한 결정을 내리는데 도움을 주기도 한다. 따라서 이러한 작업을 통해 컴퓨터 프로그래밍에서 시기상조 최적화(premature optimization)라고 불리는 것을 피할 수 있게 된다. 그리고 이러한 아이디어는 어디에 시간을 투자해야 하는지 결정하는 데 증거자료에 의해 결정을 내려야 하고 종종 잘못된 것으로 판정되는 직감을 따르지 않아야 한다는 것을 말해준다.

02:10

학습 곡선을 그려보는 것 이외에, 유용한 방법이 하나 더 있는데, 이는 오류 분석이라고 한다. 이는 스팸 메일 분류기를 생성할 때, 교차 분석 세트를 통해 알고리즘에 오류를 만드는 이메일을 직접 확인할 것이다. 그래서 알고리즘이 오분류하고 있는 스팸 메일과 논스팸 메일을 살펴보면서 어떠한 형태의 예시가 오분류 되고 있는지 시스템 상 패턴을 찾아내려고 할 것이다. 이러한 과정을 통해, 새로운 특징을 디자인하는데 영감을 줄 것이다. 아니면 시스템에서 현재 상태와 결정을 보여줄 것이다. 이를 통해 이를 향상시킬 수 있는 영감을 줄 수도 있다. 여기 구체적인

예를 들어보자.

03:00

스팸 메일 분류기를 생성했다고 하고, 교차 검증 세트에 500개의 예시가 있다고 하자. 이러한 예시는 알고리즘에서 매우 높은 오류율을 보여준다고 하자. 그리고 교차 검증 예시로 100개를 오류로 분류했다고 하자. 이제는 이 100개의 오류를 직접 살펴보면서 카테고리로 나눈다. 예를 들어 어떤 종류의 이메일인지, 어떤 신호 또는 특성이 알고리즘이 이들을 맞게 분류할 수 있게 도와줄지 생각해본다. 예를 들어, 어떤 이메일인지, 100개의 오류를 살펴보면, 이 분류기에 있는 가장 흔한 형태의 스팸 메일은 예를 들어 pharma 또는 pharmacies에 관한 것이 많았는데, 이는 약품을 팔기 위한 메일이었다. 또, 이미테이션 제품을 팔려는 이메일도 있는데, 가짜 시계, 가짜 명품 등이 있다. 또 비밀번호를 가로채려는 이메일도 있다. 피싱메일이라고 한다. 이것은 이메일에서 큰 카테고리에 속하고 아마 이것들과는 다른 카테고리일 것이다.

04:06

따라서 이메일의 종류를 구분하는 것은, 백 개의 이메일을 직접 살펴봤다고 하자. 그 중에서 12개는 약을 팔려는 이메일이고, 4개는 가짜 시계나 가짜 이미테이션을 팔려는 이메일이다. 그리고 53개의 이메일은 피싱 이메일이었는데, 이것들은 여러분의 비밀번호를 빼내가기 위한 것이다. 그리고 31개의 이메일은 기타 종류였다. 이렇게 이메일을 카테고리 별로 구분하고 보니, 이와 같은 사실을 발견했을 것이다. 이 알고리즘은 특히 비밀번호를 훔치려고 하는 이메일에 취약하다는 것이다. 이는 따라서 이와 같은 종류의 이메일에 대해서는 더 주의를 기울여야 하고, 이를 맞게 분류하기 위해 더 나은 feature를 생각해내야 한다는 것을 의미한다.

04:57

이번에는 이러한 이메일을 분류하기 위해 알고리즘에 어떠한 feature를 더 추가할 수 있는지를 살펴보는 것이다. 여기 피싱 메일을 잘 구분하기 위한 feature에 관한 가설을 몇 개 있다. 먼저 의도적인 스펠링 오류를 감지하는 것 vs 특수한 이메일 전송 경로 vs 과도한 문장부호 사용, 예를 들면 사람들이 느낌표를 많이 사용한다 등이 있다. 이제 다시 수동으로 이메일을 살펴보면, 여기는 5개, 여기는 16개, 여기는 32개 그리고 나머지는 기타라고 하자. 교차 검증 세트에서 이와 같은 결과가 나왔다면, 의도적인 스펠링 오류는 흔한 경우가 아니기 때문에 이를 감지하는 알고리즘을 생각해내려고 시간을 투자하는 것은 아깝다. 하지만 많은 스팸메들이 문장부호를 과도하게 사용하는 것처럼 보이니 이러한 문자 부호에 기반하여 복잡한 feature를 개발하는 것이 더 효율적으로 시간을 활용하는 방법일 것이다. 따라서 이와 같은 오류 분석은 알고리즘이 만든 실수를 수동으로 살펴보는 과정으로 가장 유익한 방법이 무엇인지를 안내해주고는 한다. 또한 내가 왜 먼저 알고리즘을 빠르게 구현해보라고 권하는지 그 이유를 설명해준다. 우리가 원하는 것은 알고리즘이 분류하기 가장 어려운 예시가 무엇인지를 알아내는 것이다. 그리고 다른 알고리즘, 다른 학습 알고리즘에서 예시가 비슷한 카테고리를 어려워한다. 따라서 빠르게 알고리즘을 구현해보면, 오류를 빠르게 발견하고, 어려운 예시를 찾아내는 좋은 방법이 된다. 따라서 그 오류에 좀 더 시간을 투자하면 되는 것이다.

06:49

학습 알고리즘을 개발할 때, 또 다른 유용한 팁은 학습 알고리즘을 수치화 평가하는 것이다. 학습 알고리즘을 개발할 때, 이는 무척 도움이 될 것이다. 학습 알고리즘을 평가하는 방법을 알고 있다면, 실제 숫자를 하나 줄 텐데 정확도 아니면 오류율을 나타낼 것이다. 이 실수는 여러분의 학습 알고리즘이 얼마나 잘 구현되고 있는지를 말해준다. 이후 강의에서 이 개념에 대해 더 자세히 알아보기로 하고, 예시를 먼저 살펴보자. 먼저, discount, discounts, discounted, discounting 와 같은 단어를 한 단어로 봐야 할지 아닐지를 선택해야 한다. 해결 방법은 먼저 단어의 첫 몇 글자를 보는 것이다. 단어의 첫 몇 글자를 보고 이러한 알파벳으로 시작하는 단어는 아마도 비슷한 의미를 지닐 것이라는 것을 알아낸다

07:50

보통 언어 처리에서, 이와 같은 작업을 스테밍(stemming) 소프트웨어를 활용하여 진행한다. 이를 직접 해보고 싶다면, poster stemmer 를 인터넷에서 검색해보면 되는데, 이와 같이 어간추출을 통해 discount, discounts 등을 같은 단어로 처리하는 합리적인 소프트웨어이다. 하지만 스테밍 소프트웨어를 활용하면 단어의 첫 몇 글자를 살펴보는데, 도움이 되기도 하지만 오히려 방해가 될 수도 있다. 예를 들어, universe 와 university 라는 단어를 같은 단어로 취급할 수 있기 때문이다. 이 두 단어는 같은 글자로 시작하기 때문이다. 따라서 스팸 교차 분류기에 스테밍 소프트웨어를 사용할지 안 할지를 결정하는 것은 쉬운 결정은 아니다. 특히 오류 분석은 이러한 스테밍 방법이 좋은 방법인지를 결정하는 데에 도움이 되지 않을 수도 있다. 대신에, 스테밍 소프트웨어가 분류 작업에 도움이 될지를 알아내는 최선의 방법은 그저 이를 직접 시행해보고 그 결과를 확인해 보는 것이다.

Q) 오류 분석을 수행할 때 $J_{test}(\theta)$ 를 연산하는 테스트 세트 대신 $J_{cv}(\theta)$ 를 연산하는 교차 검증 데이터를 통한 접근법이 권장되는 이유는?

1. 교차 검증 데이터 세트가 더 크기 때문이다.
2. 이러한 방법이 테스트 세트에서 더 낮은 오류율을 나타내기 때문이다.
3. 테스트 세트를 통해 새로운 feature를 개발하면 테스트 세트에서만 작동하는 feature를 선택하게 되어, 결국 $J_{test}(\theta)$ 는 새로운 예시를 일반화하는데 좋은 평가기준이 되지 못하기 때문이다.
4. 이와 같은 방법을 통해 너무 많은 feature를 선택하는 것을 피할 수 있기 때문이다.

정답 3번

09:08

이와 같은 작업을 위해, 알고리즘을 수치화 평가하는 것은 매우 도움이 될 것이다. 예를 들어,

가장 먼저 교차 검증 오류를 통해 스테밍을 활용했을 때와 아닐 때의 알고리즘의 성능을 비교해 본다. 스테밍 없이 알고리즘을 구현했을 때에는 5%의 분류 오류가 나왔다. 이번에는 스테밍을 활용하여 구현해보면, 3%의 오류가 나오고 오류율이 매우 빠르게 감소하면서 스테밍을 활용하는 것이 좋은 방법임을 알려준다. 이와 같은 특정 문제에서 실수 평가 매트릭스 즉, 교차 검증 오류가 하나 있다. 이러한 종류의 실수 평가 매트릭스는 좀 더 살펴볼 필요가 있으니 이러한 예시를 이후에 살펴보자. 이후 강의에서도 살펴보겠지만, 이런 과정을 통하여 스테밍을 활용할지 안 할지에 대한 결정을 빨리 내릴 수 있도록 도와준다.

10:08

예시 하나만 빠르게 살펴보겠다. 이번에는 대문자와 소문자를 구분해야 할지를 선택하려고 한다고 가정하자. 여기 대문자로 시작한 Mom과 소문자 mom 이 있는데, 같은 단어로 봐야 할까 아니면 다른 단어로 봐야 할까? 같은 feature 아님 다른 feature로 봐야 할까? 다시 말하지만 우리는 알고리즘을 평가할 방법을 알고 있다. 이번 경우에, 대소문자를 구별하는 것을 멈추게 되면, 오류율이 3.2% 정도 나온다고 하자. 그렇다면 스테밍만 활용했을 때보다 오류율이 더 높았다. 따라서 계속 대소문자를 구분해야 할지 말아야 할지를 결정할 수 있게 도와준다.

10:51

따라서 학습 알고리즘을 개발할 때, 다양한 아이디어와 알고리즘의 새로운 버전을 시도해 볼 것이다. 새로운 방법을 적용해볼 때마다 이렇게 많은 예시를 어떤 게 나은지 수동으로 살펴본다면, 결정을 내리는 것이 쉽지 만은 아닐 것이다. 스테밍을 사용할 것인지 아닌지? 대소문자를 구별할 것인지? 하지만 실수 평가 매트릭스 하나를 이용하여 화살표가 위로 올라갔는지 아래로 내려갔는지 살펴본다면 새로운 아이디어를 훨씬 빠르게 적용해보는데 활용할 수 있고, 또한 이러한 아이디어가 학습 알고리즘의 성능을 향상시킬지 저하시킬지를 거의 바로 알 수 있게 된다. 이렇게 되면 훨씬 더 빠르게 처리할 수 있게 된다. 따라서 오류 분석에서는 테스트 세트가 아니라 교차 검증을 통한 방법을 매우 권장하고 있다. 하지만, 사람들은 여전히 테스트 세트에서 이를 처리하고 있다. 교차 검증 세트를 통해 오류 분석을 하는 것보다 테스트 세트를 통하는 방법은 수학적으로도 덜 적절하고, 권장되지 않는 방법이다.

11:58

이번 강의를 마무리하기 전에, 새로운 기계학습 문제를 해결하기 시작할 때, 거의 항상 먼저 빠르게 구현할 수 있는 알고리즘을 활용해보라고 권유한다. 보통은 이러한 단계에 시간을 많이 투자하고 있고, 오히려 이 단계에서 시간을 너무 오래 끄는 경우를 많이 봤다. 따라서 너무 빠르게 또는 간편하게 하는 데에만 초점을 맞추지 말고 그저 빠르게 한 번 구현해볼 수 있는 것을 해보면 된다. 그렇게 초기 구현을 마치고 나면, 다음에 어느 단계를 진행해야 할지 결정을 해주는 도움을 주는 훌륭한 tool이 된다. 왜냐하면 처음에 알고리즘이 만들어내는 오류를 보고, 이 오류를 분석하여 알고리즘이 어떤 실수를 하는지를 확인해서 그 이후 개발에 도움을 줄 수 있기 때문이다. 두 번째로, 빠르고 간편한 구현을 단일 실수 평가 기법에 적용해보자. 이를 통해 새로운 아이디어를 빠르게 적용해보아 이것이 알고리즘의 성능을 향상시키는지 알아볼 수 있다.

따라서 학습 알고리즘에서 어떤 것을 적용하고 포함해야 할지에 대해 훨씬 더 빨리 결정을 내릴 수 있게 된다.

No.	Title
Week 6-10	Error Metrics for Skewed Classes

00:00

이전 강의에서 오류 분석과 오류 매트릭스의 중요성에 대해 이야기해보았는데, 이는 학습 알고리즘이 잘 구현되고 있는지를 말해주는 실수 평가 매트릭스다. 평가와 오류 매트릭스의 맥락에서 중요한 케이스가 하나 있는데, 바로 학습 알고리즘을 위한 적절한 오류 매트릭스 또는 평가 매트릭스를 생각해내는 것이 어렵다는 것이다. 이러한 경우를 *skewed classes* 라고 부르는데, 이것이 무엇을 의미하는지 자세히 알아보자.

00:36

암을 분류하는 문제를 생각해보면, 환자를 feature로 놓고, 그들이 암인지 아닌지를 결정하려고 한다. 이전에 살펴봤던 악성 vs 양성 종양 분류와 비슷한 예시다. 그렇다면, 환자가 암에 걸렸으면, $y = 1$, 그렇지 않으면 $y = 0$ 이다. progression classifier에 대해 학습했으니, 테스트 세트를 통해 분류기를 테스트해보고, 그 결과로 1%의 오류율이 나왔다. 따라서 99% 정확한 진단을 내린 것이다. 굉장히 훌륭한 결과인 것 같다. 99%의 정확도를 보였다.

01:12

하지만 학습 테스트 세트를 통해 0.5% 환자만 실제로 암을 가지고 있었다는 것을 발견했다고 하자. 따라서 검진 과정을 거친 환자의 절반가량만 암을 가지고 있는 것이다. 이런 경우에, 1%의 오류율이 그렇게 좋은 결과로 보이지 않는다. 특히, 여기 코드가 하나 있는데, 이것은 학습 코드가 아니고 feature x 를 입력하면 이를 무시해버린다. 그저 $y=0$ 이라고 설정되어있고, 항상 아무도 암에 걸리지 않았다고 예측하고, 이 알고리즘은 0.5 % 오류율을 보였다. 이는 아까 1%의 오류율보다 더 나은 결과이다. 이것은 학습 알고리즘이 아니고, 항상 $y = 0$ 이라고 예측하는 것뿐이다.

01:57

따라서 이와 같이 포지티브와 네거티브의 비율 중 한 쪽이 우세한 경우가 있는데, 여기서도 포지티브 예시가 네거티브 예시보다 훨씬 적었다. 왜냐하면 $y = 1$ 인 경우가 거의 없었기 때문인데, 이런 경우는 *skewed classes* 경우라고 부른다. 한 쪽의 예시가 다른 한쪽보다 훨씬 많았다. 그리고 $y=0$ 또는 $y = 1$ 이라고 항상 예측하는 것 만으로 알고리즘이 잘 구현되는 것을 알 수 있다. 평가 매트릭스로써 분류 오류 또는 분류 정확성을 활용하는 문제는 다음과 같다.

여기 결합 알고리즘이 하나 있는데, 99.25의 정확도를 보인다고 하자. 그럼 오류율은 0.8%가 된다. 알고리즘을 수정해서 이제 정확도가 99.5%가 되었다고 하자. 그러면 오류율은 0.5%가 된다.

03:04

그렇다면 이것은 알고리즘 성능이 향상된 것인가 아닌가? 단일 실수 측정 평가기법을 가지는 것의 장점 중 하나는 이것이 알고리즘에 도움이 되는 변화를 줄 것인지 아닌지를 빨리 결정할 수 있게 도와준다는 것이다. 99.2% 정확도에서 99.5%의 정확도로 증가하는 것. 이게 유용한 변화인지 아니면 그저 $y=0$ 이라고 더 자주 예측한 코드로 대체한 것뿐인지 말이다. 따라서 매우 skewed classes 를 얻게 되면, 분류 정확도를 활용하기 훨씬 어려워지는데, 매우 높은 분류 정확도 또는 매우 낮은 오류를 얻을 수 있기 때문이다. 따라서 이렇게 하는 것이 분류기의 품질을 향상시키는 것인지 확실치가 않은데, 이는 항상 $y=0$ 을 예측하는 것은 좋은 분류기로 보이지는 않기 때문이다.

03:53

하지만 $y=0$ 을 더 자주 예측한다는 것만으로 0.5%까지 오류가 줄어들 수 있다. 따라서 skewed classes 를 발견하게 되면, 다른 오류 매트릭스 또는 평가 매트릭스를 생각해내려고 할 것이다. 그러한 평가 매트릭스에는 precision / recall 이 있다. 이에 대해 설명하고자 한다. 테스트 세트를 통해 분류기를 평가하고자 한다. 테스트 세트에 있는 예시에서 해당 예시의 실제 class는 1 또는 0이 된다. 만약 2진 분류 문제가 생긴다면 말이다. 이 경우에 학습 알고리즘은 class의 어떤 값을 예측하고, 학습 알고리즘은 테스트 세트에 있는 예시 각각의 값을 예측하고, 예측된 값은 또한 1 또는 0이다.

04:50

이제 2×2 테이블을 다음과 같이 그려보겠다. 실제 class와 예측된 class 가 무엇인가에 따라 칸이 채워질 것이다. 실제 class가 1이고, 예측 class가 1이면, true positive 예시라고 하는데, 이는 알고리즘이 이를 포지티브라고 예측했고, 실제로도 그 예시가 포지티브라는 것이다. 학습 알고리즘이 네거티브를 예측했다면, class 0가 되고 실제 class 도 또한 class 0 이므로, true negative라고 한다. 0을 예측하고 실제로도 0이었던 것이다. 나머지 두 칸에는, 학습 알고리즘이 class 1이라고 예측했지만, 실제 class는 0이면 이는 false positive라고 한다. 알고리즘은 환자를 암이라고 예측했는데, 실제로는 그렇지 않았다는 것이다.

05:44

마지막 칸은 0, 1이다. False negative 상태인데, 알고리즘이 0을 예측했지만 실제 class는 1이었던 것이다. 이제 여기 실제 class와 예측 class를 바탕으로 한 2×2 표를 얻게 되었다. 이것은 알고리즘의 성능을 평가할 수 있는 다른 방법이다. 이제 두 숫자를 연산할 것이다. 첫 번째는 precision이다. 이것은 모든 환자 중에 우리가 암이라고 예측한 환자, 전체 중 어느 정도가 실제로 암 환자냐는 것이다. 이를 적어보면, 분류기의 precision은 true positive 수 나누기 우리가 예측한 포지티브의 수이다.

06:39

따라서 모든 환자 중에서 이 분류에 속한 환자들에게 가서 “암입니다.”라고 말한 것이다. 모든 환자 중에서 실제 암환자는 얼마나 되는가? 이것이 바로 precision이다. 또 다른 방법은 true positives 나누기 분모는 예측 positives 수인데 이는 표에서 첫 번째 행의 합과 같다. 따라서 true positives 나누기 true positives 더하기, positive를 pos라고 줄여서 쓰겠다. false positives를 더 해주면 된다. 여기 다시 pos는 positive의 약자다. 이게 바로 precision이다. Precision이 높을수록 좋은 것이다. 우리가 찾아간 모든 환자들에게 우리가 “안타깝지만, 암입니다.”라고 말한 것이다. High precision은 환자 그룹 중 대부분에게 우리가 정확한 예측을 했으면, 그들이 암환자라는 것이다.

07:38

두 번째로 연산할 수는 recall이다. 여기서 recall은 만약 테스트 세트나 교차 검증 세트에 있는 모든 환자가, 아니, 데이터 세트에 있는 모든 환자가 실제로 암환자일 때, 그들이 암환자라는 사실을 얼마나 정확하게 진단했는지를 나타낸다. 따라서 모두 암환자일 때, 우리가 그들 중 얼마나 많은 환자에게 가서 암이니까 치료가 필요하다고 이야기를 해주었느냐는 것이다. 이것을 공식화 해보면, recall은 true positives 수, 우리가 정확하게 예측한 암환자의 수를 가지고 실제 positives 수로 나눠주면 된다. 따라서 이것은 암환자 중 얼마나 많은 환자에게 치료를 받아야 한다고 말해주었느냐는 것이다.

08:40

다른 형태로 다시 써보면, 분모는 실제 positives 숫자, 즉 첫 번째 열의 합이다. 따라서 다르게 써보면, true positives 수 나누기 true positives 수 + false negatives 수가 된다. Recall이 높다는 것은 좋은 현상이다. 따라서, precision과 recall을 연산하는 것은 분류기가 잘 구현되고 있는지 알 수 있게 도와준다.

09:21

특히, $y=0$ 이라고 항상 예측하는 학습 알고리즘이 있다면, 이는 아무도 암이 아니라고 예측할 것이고, 그렇다면 이 분류기는 $\text{recall} = 0$ 가 되는데, true positives가 하나도 없기 때문이다. 따라서, $y=0$ 이라고 항상 예측하는 분류기는 그렇게 좋은 분류기는 아니라는 것을 쉽게 알 수 있다.

Q1) Precision과 Recall은 다음과 같이 정의할 수 있다.

테스트 세트에서 여러분의 알고리즘의 성능은 다음과 같다. 그렇다면 이 알고리즘의 precision은?

정답 1번

Q2) 테스트 세트에서 여러분의 알고리즘의 성능은 다음과 같다. 그렇다면 이 알고리즘의 recall은?

정답 1번

또한 매우 skewed classes 한 경우로 설정한다고 해도, 항상 $y=0$, 항상 $y=1$ 과 같은 단순한 예측으로 매우 높은 precision과 recall이 나오도록 알고리즘을 조작하는 것은 불가능하다. 따라서 매우 높은 precision과 recall이 나오는 분류기가 좋은 분류기라는 확신을 할 수 있고, 더 유용한 평가 매트릭스를 제공하는데, 이는 알고리즘이 잘 구현되고 있다는 것을 이해할 수 있는 직접적인 방법이다.

10:21

마지막으로 precision과 recall의 정의에 대해 추가 설명하자면, 우리는 precision과 recall 을 $y = 1$ 이라는 convention을 사용하고, 더 rare한 class 의 경우에 활용한다. 만약 암과 같은 rare condition 을 감지하려고 한다면, precision과 recall은 우리가 감지하려고 하는 rare class에서 $y=0$ 보다는 $y=1$ 로 정의된다. precision과 recall를 활용하면, 매우 skewed classes의 경우라도, 항상 $y=0$, 항상 $y=1$ 과 같은 단순한 예측으로 매우 높은 precision과 recall이 나오도록 알고리즘을 조작하는 것은 불가능하다. 특히, 분류기에서 high precision 과 high recall을 얻게 된다면, 매우 skewed classes의 경우라도 이 알고리즘이 잘 구현되고 있음을 확신할 수 있게 된다.

11:18

따라서 skewed classes precision recall 문제는 학습 알고리즘이 어떻게 구현되는지에 대한 직관력을 준다. 이는 또한 매우 skewed classes의 경우에 분류 오류 또는 분류 정확도 보다 우리의 학습 알고리즘을 평가할 수 있는 훨씬 좋은 방법이다.

No.	Title
Week 6-11	Trading Off Precision and Recall

00:00

지난 시간에는 skewed classes와 함께 분류 문제에서 활용한 평가기법으로 precision과 recall에 대해 살펴봤다. 많은 어플리케이션에서 precision과 recall 사이에서 trade-off를 컨트롤 하려고 하는데 이 방법에 대해 먼저 이야기한 후, 학습 알고리즘에서 평가기법으로 precision과 recall을 더 효율적으로 활용하는 방법에 대해 소개하겠다. 지난 시간에 살펴본 내용을 다시 살펴보자면, 여기 precision과 recall에 대해 정의가 있다. 지난 번에 살펴본 암 분류 예시에서 $y=1$ 은 환자가

암에 걸렸을 때를 말하고, $y=0$ 은 그렇지 않은 경우를 나타냈다. 여기 로지스틱 회귀 분류기를 학습했다고 할 때, 이 확률은 0과 1사이가 된다. 평소처럼, $y=1$ 을 예측하면 가설은 0.5보다 크거나 같고, $y=0$ 을 예측하면 가설은 0.5보다 작게 된다.

01:03

이 분류기는 precision과 recall의 어떠한 값을 제공하는데, 이번에는 어떤 환자가 암에 걸렸다고 가정할 때, 이는 매우 확실한 가정이다. 왜냐면 환자에게 가서 암이라고 말하면 이는 매우 충격적이고 매우 나쁜 소식이므로 그들은 매우 힘든 치료 과정을 거쳐야 하기 때문에 암이라는 것이 확실할 경우에만 환자에게 이 사실을 말해줘야 한다. 이를 위해 알고리즘을 변경할 건데, threshold를 0.5 대신에 0.7로 바꿔주면 이제 암에 걸릴 확률이 70% 이상이 되어야 환자에게 이야기를 해준다는 것이다. 이렇게 되면, 이 환자가 암에 걸렸다는 것이 더 확실해지고, 이 분류기는 precision이 높아지는데 이는 모든 환자한테 가서 '암인 것 같습니다.'라고 말해야 하는데 그들이 암에 걸렸다는 것이 꽤 확신을 하고 있고, 여러분이 암이라고 예측한 환자가 실제로 암일 확률이 높다는 것이므로, 확신이 있어야만 이러한 예측을 할 수 있기 때문이다.

02:34

반면에, recall은 낮아지는데 왜냐하면 $y=1$ 이라는 환자의 수가 줄어들기 때문이다. 만약 이 threshold를 0.9로 높여주면, 암이라고 90% 이상 확신이 있을 때에만 환자에게 암이라고 말해줄 수 있는 것이다. 이중에 많은 사람들이 암으로 판명 날 것이고 아까와 같이 precision은 높아지고, recall은 낮아지게 되고, 암환자를 더 정확히 판별할 수 있게 되기 때문이다. 다른 예시를 살펴보면, 많은 암환자를 구별하지 못하는 것을 막기 위해서, false negative를 피하기 위해서, 특히 환자가 실제로 암에 걸렸는데, 우리가 그 환자에게 암이라고 판정하는데 실패했다면 이는 매우 좋지 않다. 그 환자에게 암이라고 말하지 않으면, 그 환자는 치료를 하지 않을 것이고, 하지만 암이라고 나중에 밝혀지면 그들에게 암이라고 판정하는 데 실패한 것이고, 그들은 그때까지 치료를 받지 못했고, 매우 좋지 않은 결과를 낳을 수 있다. 예를 들어 죽을 수도 있다. 왜냐하면 우리가 암이라고 말해주지 않았는데 실제로 암인데 치료를 받지 못했기 때문이다.

03:48

$y=1$ 이라고 가정한다고 할 때, 그들이 암에 걸렸다고 예측하여, 이를 더 검사해보거나 암이라고 판명이 났을 때 치료를 받을 수 있도록 해야 한다. 이런 예시에서는 높은 threshold를 선택하기보다는 이 수치 대신에 낮은 0.3을 적용한다. 이는 암에 걸릴 확률이 30%가 된다는 것으로, 매우 보수적으로 대응하는 것이고, 암일 경우 필요한 경우 치료를 받으라고 하는 것이다. 이와 같은 경우에는 recall이 높아지게 되는데 더 많은 환자가 암에 걸릴 확률이 있다고 표시하는 것이고, precision이 낮아지게 되는데, 많은 환자들에게 암에 걸릴 확률이 있다고 말해주었으니, 그 중에 많은 환자들이 실제로는 암이 아니라고 판명이 될 것이기 때문이다. 덧붙이자면, 예전에 어떤 학생이 이런 질문을 했는데 recall과 precision이 높은 두 경우 모두 어떻게 활용할 수 있느냐는 것인데 사실 둘 다 활용 가능하다. 알고리즘의 세부 내용은 모두 참이기를 바라고, 더 일반적인

규칙은 높은 precision, 낮은 recall / 높은 recall, 낮은 precision, 중 어떤 것을 원하든지 $h(\theta)x \geq threshold$ 보다 크거나 같을 경우, $y=1$ 를 예측하게 된다.

05:38

일반적으로 대부분의 분류기는 precision과 recall 사이에서 trade-off하고, 여기 있는 threshold 값을 바꿔주면 이와 같이 precision과 recall을 나타내는 곡선 그래프를 그릴 수 있다. 여기 위에 값은 0.99와 같은 높은 threshold값을 말하고, 이는 $y=1$ 일 때, 암일 확률이 99%가 된다는 것이다. 확률이 99%가 된다는 것이다. 높은 precision, 낮은 recall의 경우였고, 반면에 이 지점은 0.01과 같은 훨씬 낮은 threshold값을 가진다. 이는 $y=1$ 일 때, 높은 recall, 낮은 precision이 된다. 이와 같이 threshold값을 변경하여 모든 분류기의 곡선을 추적하여 다양하게 얻을 수 있는 precision과 recall값의 범위를 확인할 수 있다. 또한, precision과 recall 곡선은 다양한 형태로 그려질 수 있는데, 때로는 이렇게, 아니면 이런 식으로 분류기의 세부 내용에 따라 달라질 수 있다.

06:57

따라서, 여기 흥미로운 질문이 하나 있는데, 이 threshold를 자동으로 선택할 수 있는가 하는 것이다. 일반적으로 다른 알고리즘이 몇 개 있는데, precision과 recall 수를 어떻게 비교할 수 있을까? 예를 들어, 이렇게 서로 다른 알고리즘 세 개가 있다고 하자. 또는 같은 알고리즘이지만 threshold가 달라서 다른 precision과 recall 값을 가진 알고리즘 일 수도 있다. 이중에 어떤 알고리즘이 가장 좋다고 말할 수 있을까? 이전에 단일 실수 평가기법(single real number evaluation metric)의 중요성에 대해 말한 적이 있다. 그 숫자는 여러분의 분류기가 얼마나 잘 작동하고 있는지를 말해준다. 하지만 이를 recall 기법으로 바꿨더니 이것이 없어졌고, 실수가 2개만 있는데, 따라서 종종 알고리즘 1과 2를 비교하게 되는 상황을 마주하게 된다. Precision은 0.5, recall은 0.4인데 여기 precision 2는 0.7, recall은 0.1 보다 좋은지 나쁜지를 비교하는 것이다. 이와 같은 알고리즘에서 비교를 해본다면, 아마 0.5/0.4가 0.7/0.1보다 좋은지 나쁜지 잘 모르겠다고 할 것이다. 이렇게 되면, 어떤 변화가 알고리즘에 효율적인지를 판단하는데 결정을 내리는 과정이 자연스럽게 만든다.

08:22

반면에, 단일 실수 평가기법에서는 숫자를 보고 알고리즘 1, 2에서 어떤 알고리즘이 더 잘 적용되는지를 빨리 결정할 수 있게 해준다. 따라서 더 빨리 알고리즘에 적용하게 될 변화를 측정할 수 있게 된다. 그렇다면 단일 실수 평가기법은 어떻게 얻을 수 있을까? 가장 자연스러운 방법은 precision과 recall의 평균을 구하는 것이다. 여기서 p 와 r 은 precision과 recall을 나타내는데, 여기서 평균값을 연산하여 어떤 것이 가장 높은 평균값을 가지고 있는지 살펴보는 것이다. 하지만 이 것은 그렇게 좋은 해결 방법이 아닌 것 같은데, 이전에 살펴본 예시에서처럼 항상 $y=1$ 이라고 예측하는 분류기에서는 높은 recall을 얻게 되는데, 이로 인해 precision은 매우 낮아진다. 반대로, 분류기가 $y=0$ 라고 거의 대부분 예측한다면, 이는 $y=1$ 을 예측하는 것과 반대로, 아까 살펴본 것처럼 매우 높은 threshold를 설정하게 되고, 결국 높은 precision과 낮은 recall을 얻게 된다.

09:40

따라서 높은 threshold 또는 낮은 threshold를 가진 두 예시 모두 좋은 분류기를 생성하지는 못할 것이다. 이를 통해, 매우 낮은 precision 또는 매우 낮은 recall을 얻게 되는 것이다. 이 평균을 가지고 보면, 여기서 알고리즘 3의 평균이 가장 높을 것을 알 수 있는데, 이는 항상 $y=1$ 을 예측하기 때문인데 이는 그리 좋은 분류기는 아니다. 여기처럼 $y=1$ 을 항상 예측한다면 이는 그다지 효율적인 분류기는 아닐 것이다. 따라서 알고리즘 1, 2가 알고리즘 3 보다 더 유용한데, 하지만 여기서는 알고리즘 3가 precision과 recall이 평균값은 가장 높게 나온다. 이를 통해 precision과 recall 이 평균값은 학습 알고리즘을 평가하는데 그렇게 좋은 방법은 아니라고 결론지을 수 있다.

10:37

반면에, precision과 recall 다르게 합하는 방식이 있는데, f score라고 하고 공식은 다음과 같다. 따라서 이렇게 f score 값을 구하면, 이 값을 통해 알고리즘 1이 가장 높고, 알고리즘 2가 그 다음이고, 알고리즘 3가 가장 낮을 값을 가진 것을 알 수 있다. 이 방식을 따르자면 알고리즘 1을 선택할 것이다. F score은 f1 score라고도 하는데, 쓸 때는 보통 f1 score라고 하는데, 사람들은 보통 f Score라고 말한다. 둘 다 사용된다. precision과 recall의 평균을 구할 때 precision과 recall의 값이 낮은 알고리즘에서는 weight가 더 높다. 여기 분수의 분자를 보면 f score는 precision과 recall의 곱을 가지는데, 따라서 precision과 recall 둘 중 하나가 0이라면 f score는 0이 된다. 따라서 이는 precision과 recall이 둘 다 커야, fscore도 크다는 것을 알 수 있다.

11:43

precision과 recall 값을 활용한 다양한 공식이 있는데, 이 f score 공식이 더 높은 확률로 사람들이 예전부터 전통적으로 기계학습에서 많이 활용한 공식이라고 할 수 있다. 그리고 f score 용어는 아무런 뜻이 없다. 따라서 f score, f1score 라고 불리는지에 대해 고민할 필요가 없다. 다만 이는 여러분이 원하는 것을 제공하는데, 따라서 precision과 recall 둘 중 하나가 0이라면 이로 인해 매우 낮은 f score를 얻게 되는데, 높은 fscore를 얻기 위해서는 precision과 recall 둘 다 1이어야 한다. 예를 들어, $p=0$ 또는 $r=0$ 일 때, $fscore = 0$ 인데, $p=1$ 그리고 $r=1$ 이면, Precision과 recall이 완벽하게 1일 때 $f score = 1$ 이다. 그리고 0과 1사이의 중간값은 보통 합리적인 분류기의 순위를 알려준다.

Q) 로지스틱 회귀 분류기를 학습하여, 다음과 같은 예측을 하려고 한다.

Threshold 매개변수의 다른 값으로, 다양한 precision과 recall 값을 얻었다. 다음 중 어떠한 방법이 threshold 값을 선택하는 합리적인 방법인가?

1. 테스트 세트에 있는 Precision과 recall을 측정해 $P+R/2$ 의 최대값인 threshold 선택

2. 테스트 세트에 있는 Precision과 recall을 측정해 $2(PR/P+R)$ 의 최대값인 threshold 선택
3. 교차 검증 세트에 있는 Precision과 recall을 측정해 $P+R/2$ 의 최대값인 threshold 선택
4. 교차 검증 세트에 있는 Precision과 recall을 측정해 $2(PR/P+R)$ 의 최대값인 threshold 선택

선택
'정답 4'

13:00

이번 시간에는 Precision과 recall를 trade-off의 개념과 우리가 활용할 threshold를 다양하게 바꿔서 $y=1$, 또는 $y=0$ 임을 결정하는 데 활용했다. 여기서 $Y=1$ 이라는 것을 예측하기 전에 최소 70% 또는 90%의 확신이 필요한데, threshold를 바꾸면서 Precision과 recall 사이의 trade-off 를 통제할 수 있었다. 또 f score에 대해 이야기해봤는데, Precision과 recall를 활용했고, 또한 단일 실수 평가기법을 제공했다. 여러분의 목표는 자동으로 threshold를 설정해 $y=1$, 또는 $y=0$ 을 선택하는 것인데, 이와 같이 하는 이유는 다양한 threshold 범위 값을 통해 교차 검증 세트에서 이러한 threshold를 측정한 후 그 threshold 값을 선택하여 교차 검증 세트에서 가장 높은 fscore를 얻게 되는지를 측정하는 것이다. 이는 분류기를 위해 자동으로 threshold 선택하려는 이유이기도 하다.

No.	Title
Week 6-12	Data For Machine Learning

00:00

지난 강의에서, evaluation metrics에 대해 이야기해보았다. 이번 강의에서는 방향을 약간 전환해서 기계학습 디자인에서 중요한 부분을 다뤄볼까 하는데, 자주 등장하게 되는 이슈로 얼마나 많은 데이터를 학습할 것인가에 관한 내용이다. 이전 강의 중에서 데이터를 수집하는데 시간을 너무 허비하지 말라고 주의를 요한 적이 있다. 왜냐하면 이것이 도움이 되는 경우는 한정되어 있기 때문이다. 하지만 특정한 상황에서는, 이번 강의에서 설명할 건데, 많은 데이터를 수집해서 학습 알고리즘의 특정 형태를 학습하는 그러한 상황에서는 많은 데이터 수집이 학습 알고리즘이 좋은 성능으로 구현될 수 있도록 해주는 효과적인 방법이 될 것이다. 이는 또한 만약 이러한 상태가 문제를 해결하는데 딱 들어맞는다면, 그리고 여러분이 많은 데이터를 얻을 수 있다면, 고성능 학습 알고리즘을 구현하는 매우 효과적인 방법이 될 것이다. 이제 더 자세히 알아보자.

00:58

먼저 스토리를 들려주겠다. 옛날에 내가 아는 연구원 두 명이 있었는데, Michelle Banko 와 Eric Broule이다. 그들은 다음과 같은 멋진 연구를 진행 중이었다. 그들은 다른 학습 알고리즘을

활용하는 효과와 그것들을 다른 학습 세트 sciences 에서 시도해보는 연구하는데 흥미를 가지고 있었다. 그들은 헛갈리는 단어를 구분하는 문제에 대해 연구하고 있었는데, 예를 들어, For breakfast I ate () eggs. 라는 문장에서 to 인지 two 인지 too 가 들어가야 하는지 여부다. 이 경우에는 I ate two eggs가 맞다. 여기 한 가지 예시를 보여줬고, 이건 다른 예시 세트다. 그들은 기계 학습 문제를 다루는데, 즉 지도 학습 문제를 가지고 영어 문장에서 특정 위치에 어떠한 적절한 단어가 들어가야 하는지를 분류하려고 했다. 그들은 다른 학습 알고리즘 몇 개를 가지고, 이는 그들이 연구하던 당시 2001년에는 굉장히 획기적이었는데, 변수 그러니까 로지스틱 회귀의 변수를 사용했는데, 이는 바로 Perceptron 이다.

02:03

다른 알고리즘도 있었는데, 그 당시에는 꽤 사용했지만 지금은 많이 사용하지 않는 알고리즘이 있다. 따라서 Winnow 알고리즘이 회귀와 비슷해서, 물론 다르기는 하겠지만, 오늘날에는 그렇게 많이 활용되고 있지 않다. 또한 기억 기반 학습 알고리즘도 요즘에는 많이 사용하지 않는다. 하지만 나중에 다시 설명해주겠다. 그들은 또한 Naïve Bayes 알고리즘을 활용했는데, 이에 대해서 이제부터 설명하고자 한다. 설명한 알고리즘에 대해 자세히 알아야 할 필요는 없다. 이것은, 네 개의 각기 다른 분류 알고리즘이 있다고 알려주는 것이지 세부 내용까지 다 알아야 하지는 않다. 그들은 학습 세트 사이즈를 달리하여 이 학습 알고리즘을 학습 세트 사이즈의 범위에 적용해봤고, 이것이 그 결과물이다.

02:03

그리고 이 트렌드는 매우 분명해 보인다. 먼저, 대부분의 알고리즘은 거의 비슷한 성능을 보여주고 있다. 두 번째로 학습 세트 사이즈가 증가할수록, 여기 가로축이 단위가 백만인 학습세트 사이즈인데, 여기 10만부터 10억까지 예시가 있다. 그리고 알고리즘의 성능이 모두 거의 감소하지 않고 증가하고 있다는 것이다. 사실, 어떠한 알고리즘을 선택해도, 하위 알고리즘을 선택하게 될 것인데, 그 하위 알고리즘에 더 많은 데이터를 대입하면, 이와 같은 예시에서 상위 알고리즘도 이길 수도 있다. 따라서 이러한 연구가 매우 영향력이 있기 때문에, 이와 비슷한 결과를 보여주는 다양한 연구가 있었다. 많은 다양한 학습 알고리즘에서 내용에 따라 다르지만, 때로는 매우 비슷한 성능 범위를 보여주는 경향이 있다. 하지만 성능에 이끌 수 있는 것은 그 알고리즘에 많은 양의 학습 데이터를 제공하는 것이다. 이와 같은 결과는 기계 학습에서 누가 가장 좋은 알고리즘을 가졌느냐가 아니라 누가 가장 많은 데이터를 가지고 있는지가 승패를 결정한다고 말한다. 그렇다면 이것은 언제 사실인고 언제 거짓인가? 이 중에서 정확히 어떤 아이템을 활용해야 할지 고민하는 것보다 데이터가 많은 것이 고성능 알고리즘을 얻는 최선의 방법인 학습 알고리즘도 있다.

04:21

이번에는 많은 학습 세트를 가지는 것이 도움이 될 것이라고 생각되는 가정을 살펴보자. 기계 학습 문제에서 feature x 는 충분한 정보를 가지고 있고 y 값을 정확하게 예측하는데 활용할 수

있다고 하자. 예를 들어, 이전 슬라이드에서 살펴봤던 헛갈리는 단어 예시를 모두 가져오자. feature x가 우리가 채워 넣으려고 하는 빈칸에 들어갈 단어 예시 데이터를 수집한다고 하자. 이제 feature 는 for breakfast I have () eggs 라는 문장을 수집했다. 내가 빈칸에 넣을 단어는, two 인데, to도 아니고, too도 아니라는 것을 쉽게 알 수 있다. 따라서 feature는 이러한 단어 세트 중 하나를 수집하여 충분한 정보를 제공해주고 이러한 헛갈리는 단어를 사이에서 y 즉, 어떤 단어를 선택해서 빈칸을 채워야 하는지를 꽤 명료하게 결정할 수 있게 해준다.

05:26

이과 같이 feature x가 y값을 도출하기 위해 충분한 정보가 있는 예시를 살펴봤다. 이제는 반대다. 주택 가격을 예측하는 문제인데, 주택 사이즈만 있고 다른 feature는 없다. 예를 들어, 500 평방피트짜리 집이 하나 있는데, 그 외에 다른 feature는 전혀 제공하지 않는 것이다. 그 주택이 도시에 비싼 지역에 위치해 있는지도 말해주지 않았고, 그 집에 방이 몇 개 있는지, 가구는 잘 구비되어있는지, 새 집인지 오래된 집인지도 말해주지 않았다. 주택이 500 평방 피트라는 것 외에 아무런 정보도 제공하지 않는다면, 주택 사이즈 말고도 주택 가격에 영향을 줄 요소들이 무척 많기 때문에, 그 사이즈만 알고 있다면, 주택 가격을 정확히 예측하기 매우 어려워진다. 이와 같이 features가 어느 정도의 정확성을 가지고 가격을 예측할 수 있는 충분한 정보가 있는 가정과 반대되는 예시를 살펴봤다.

06:23

이러한 가정을 테스트하는 방법으로 자주 생각하는 것 중 하나로 나는 종종 스스로 이렇게 물어보고는 한다. Feature x 입력값, features 등 학습 알고리즘에 주어지는 것과 같은 정보가 주어졌을 때, 만약 인간 전문가가 있다면, 과연 인간 전문가는 정확하게 또는 자신 있게 y값을 예측할 수 있느냐는 것이다. 먼저 첫 번째 예시를 풀기 위해 인간 전문 영어 사용자를 찾아가보자. 영어를 잘하는 사람을 찾아간다. 그렇다면 그 인간 영어 전문가는, 즉 우리 같은 사람은 이것을 읽고 나서 쉽게 예측할 수 있으므로 확신을 가지고 x를 통해 y값을 정확하게 예측할 수 있다. 반대로 이번에는 가격 전문가를 찾아가보자. 아마 부동산업자나 업으로 집을 파는 사람 정도가 될 것이다. 내가 그들에게 단지 주택 사이즈만 말해주고, 그 가격을 말해달라고 하면, 주택 가격과 판매 전문가라고 할지라도 말하지 못할 수 있는데, 이는 주택 가격 예시에서 사이즈만을 아는 것은 그 주택의 가격을 예측하는데 충분한 정보를 제공하지 않고 있다는 것을 말해준다.

07:37

이번에는 이 가정을 살펴보면서 데이터가 많은 것이 도움이 되는지 알아보도록 하자. Features 가 y값을 예측할 수 있는 충분한 정보를 가지고 있다고 가정하자. 또한 매개 변수가 많은 학습 알고리즘을 활용한다고 하면, 아마 많은 features를 가진 로지스틱 회귀 또는 선형 회귀 정도 될 것이다. 아니면 내가 가끔 활용하는 방법인데, 은닉 계층을 많이 가지고 있는 신경망일 수도 있다. 이 또한 많은 매개 변수를 가진 학습 알고리즘 중 하나이다. 따라서 이렇게 많은 매개 변수를 가진 효과적인 학습 알고리즘이 있고, 매우 복잡한 함수를 적용할 수 있다. 이 예시 모두를 low-

bias 알고리즘이라고 하겠다. 왜냐하면 매우 복잡한 함수도 적용할 수 있기 때문이다. 우리에게 효과적인 학습 알고리즘이 있기 때문에 매우 복잡한 함수도 적용할 수 있는 것이다.

08:31

이러한 알고리즘을 데이터 세트를 통해 구현할 때, 학습 세트에 잘 맞을 것이고, 학습 오류 또한 작을 것이다. 이제 우리가 엄청 큰 학습 세트를 활용할 건데, 이런 경우에 큰 학습 세트가 주어졌을 때, 매개 변수가 많이 있을지라도, 학습 세트가 매개 변수의 수보다 훨씬 많아서 이러한 앨범은 과적합하지 않게 될 것이다. 왜냐하면 학습 세트가 엄청 크기 때문이다. 여기서 과적합 하지 않을 것이라는 의미는 학습 오류가 테스트 오류가 가깝다는 뜻이다. 마지막으로 여기 두 수식을 보면, 학습 세트 오류가 작고, 테스트 세트 오류가 학습 오류와 가깝다라는 것은 테스트 세트 오류 또한 작을 것을 의미한다.

09:31

고성능 학습 알고리즘을 얻기 위해 생각해봐야 할 다른 방법은 high bias 나 high variance 가 되어서는 안 된다. bias 문제는 매개 변수가 많은 학습 알고리즘을 가지고 있으면 이는 low bias 알고리즘을 얻게 된다. 그리고 큰 학습 세트를 활용하는 것은 variance 문제도 없을 것이라는 것을 보장해준다. 그래서 이 알고리즘에 variance가 없을 것이다. 이 두 개를 적용하면 low bias 그리고 low variance 상태의 학습 알고리즘을 얻게 되므로 테스트 세트에서 잘 구현될 것이다.

기본적으로 feature가 충분한 정보를 가지고 있다는 것은 가정에서 중요한 구성요소이다. 또한 다양한 class의 함수를 가지고 있다는 것은 low bias를 보장해주고, 많은 학습 세트를 가지고 있다는 것은 low variance를 의미한다.

Q) 큰 학습 세트를 갖게 되는 것은 학습 알고리즘 성능 향상에 큰 도움을 준다. 하지만 큰 학습 세트가 도움이 되지 않는 경우는?

1. Feature x 가 y 를 정확히 예측하기 위한 충분한 정보를 가지고 있지 않고(예를 들어, 주택 사이즈만으로 주택 가격을 예측하려는 경우), 로지스틱 회귀와 같은 단순한 알고리즘을 활용할 때
2. 많은 feature를 가진 학습 알고리즘을 활용할 때(low bias 상태인 알고리즘)
3. 은닉 계층의 수가 많은 신경망을 활용하지만 feature x 가 y 를 정확히 예측하기 위한 충분한 정보를 가지고 있지 않을 때(예를 들어, 주택 사이즈만으로 주택 가격을 예측하려는 경우)
4. 일반화를 활용하지 않을 때(일반화 매개변수 $\lambda = 0$)

정답 1,3번

10:27

따라서 데이터를 많이 가지고 있고, 많은 매개 변수를 가진 학습 알고리즘을 학습 할 때 생기는 문제에는 고성능 학습 알고리즘을 활용하는 것이 좋은 방법이라는 것을 이해할 수 있다. 내가 종종 스스로에게 묻는 주요한 테스트 문제는 첫째, 인간 전문가가 feature 를 보고 자신 있게 y 값을 예측할 수 있느냐는 것이다. 이는 feature x 로부터 y 값은 정확하게 예측 될 수 있다는 것과 같은 이러한 증명이기 때문이다. 두 번째로 우리가 실제로 큰 학습 세트를 얻을 수 있는가 그리고 학습 세트에 있는 많은 매개 변수를 가진 학습 알고리즘을 학습할 수 있는가 인데, 둘 다 할 수 있다면 매우 고성능의 학습 알고리즘을 얻게 될 것이다.

Week 7

No.	Title
Week 7-1	Optimization Objective

00:00

지금쯤이면, 여러분은 다양한 학습 알고리즘에 대해 살펴보았을 것이다. 지도 학습에서, 다양한 학습 알고리즘의 성능은 매우 비슷한데, 여러분이 학습 알고리즘 a 또는 b 중 어느 것을 활용하는지 보다는 이들 알고리즘을 적용 할 수 있는 여러분의 기술과 알고리즘을 위해 얼마나 많은 데이터를 수집했는지가 중요하다. 예를 들어, 학습 알고리즘에 적용할 feature 선택하는 것과 색상화 매개변수 선택하기 같은 것이다. 그런데, 산업 분야와 학계 모두에서 매우 효과적이고 널리 사용되고 있는 알고리즘 하나 더 있는데, 이는 서포트 벡터 머신이다. 로지스틱 회귀와 신경망과 비교해보면, 서포트 벡터 머신, 즉 SVM은 복잡한 비선형 함수를 학습하는데 더 확실하고 효과적인 방법이다. 이제 이에 대해서 강의를 시작하겠다. 강의 후반부에서는 다양한 지도 알고리즘에 대해 간단하게 살펴볼 건데, 매우 간략하게 설명해줄 것이다. 그 대중성과 강력함을 따져봤을 때, 서포트 벡터 머신은 다른 학습 알고리즘 개발과 마찬가지로 내가 이만큼 시간을 투자해서 설명할 마지막 지도 알고리즘이 될 것이다. 이제 최적화 objective에 대해 알아보자. 이제 이 알고리즘에 대한 설명을 시작하겠다.

01:31

서포트 벡터 머신을 설명하기 위해, 로지스틱 회귀를 가지고 이를 약간 변형시켜서 서포트 벡터 머신이 무엇인지에 대해 설명해 주겠다. 로지스틱 회귀에서 여기 친숙한 가설이 있고, 시그모이드 활성 함수는 오른쪽에 이렇게 있다. 이제 수학적으로 설명하기 위해, $z = \theta^T x$ 라고 하겠다. 이제 로지스틱 회귀를 어떻게 활용할지 살펴보자. 예로, $y = 1$ 의 의미는 학습 세트 또는 테스트 세트 또는 교차 검증 세트에 예시가 있는데, $y = 1$ 이면, $h(\theta)x \approx 1$ 를 바란다. 그러니까 이 예시를 정확하게 분류하기를 원한다. $h(\theta)x \approx 1$ 은 $\theta^T x$ 가 0 보다 훨씬 커야 함을 의미한다. 따라서

여기 훨씬 크다는 기호는 0보다 훨씬 더 크다는 것이다. 그러니까 $z = \theta(T)x$ 인데 이게 0보다 훨씬 크려면 이렇게 멀리 오른쪽에 있어야 하는데, 이는 로지스틱 progression 출력이 1과 가까워 진다는 것을 의미한다. 이렇게 로지스틱 progression 출력이 1과 가까워진다. 반대로, y 가 0과 가까운 예시가 있는데, 우리가 원하는 것은 이 가설 출력값이 0과 가까운 것이다. 그리고 이는 $\theta(T)x$ 가 0 보다 훨씬 작아야 함을 의미한다. 왜냐하면 여기 이렇게 가설을 대입하면 출력값이 0과 가까워지기 때문이다.

03:02

로지스틱 회귀의 비용 함수를 살펴보면, (x,y) 각각의 예시가 비용 함수 전체의 항을 되어줌을 알 수 있다. 따라서 전체 비용함수는 여기 1부터 m 까지 예시의 합이 될 것이고, 이와 같은 표현은 하나의 학습 예시가 전체적인 objective 함수에 기여함을 알 수 있다. 이제 여기 4개의 가설을 정의를 가지고, 여기 이렇게 포함되어있는데, 각각 학습 예시가 여기 이 항에 적용되어, 1부터 m 값을 무시하고 로지스틱 회귀의 비용 함수 전반에 적용된다. 이번에는 $y=1, y=0$ 의 경우를 살펴보자. 먼저 $y = 1$ 이라고 가정해보자. 이 경우에는 여기 objective 수식에서 첫 번째 항만 활용하는데, $y=1$ 일 때, $1-y = 0$ 이기 때문이다. 따라서 $y = 1$ 일 때, (x,y) 예시는 $y=1$ 일 때 우리는 이 로그항을 얻게 되는데, 아까 이 마지막 항과 비슷하다. 여기서 $z = \theta(T)x$ 로 정의한다.

04:29

여기 비용함수에서처럼 마이너스가 있어야 하지만, $y = 1$ 이기 때문에, 이 항도 1과 같아진다. 표현을 간략하게 하기 위해서 이렇게 썼다. 이 함수를 함수 z 로 그려주면, 여기 왼쪽 아래에 있는 것과 같은 곡선이 그려진다. 즉, z 값이 크면, $\theta(T)x$ 값과 커지고, 이는 z 값이 비용 함수에서 매우 적은 기여를 하는 것이다. 이는 왜 로지스틱 회귀가 $y=1$ positive 예시로 보는지를 설명해준다. 이는 $\theta(T)x$ 를 매우 크게 설정하는데, 이 비용 함수의 항에 대응하면 매우 작아지기 때문이다. 이번에는 서포트 벡터 머신에 대입해보겠다. 이 비용 함수를 가지고 약간 변경을 한다. 여기 1지점을 가지고 비용 함수를 그려보겠다. 새로운 비용함수는 이렇게 평평하게 그려지고, 여기서부터는 직선으로 위로 올라간다. 이는 로지스틱 회귀와 비슷한 직선으로 그려지지만, 이 부분은 직선이다. 따라서 내가 방금 자홍색으로 그린 곡선은 로지스틱 회귀를 활용한 비용 함수에 매우 근접한 추정이다. 이렇게 한쪽은 평평한 직선 왼쪽은 이렇게 두 직선으로 이루어졌다는 사실만 제외하면 말이다. 여기 이렇게 기울기 직선 부분은 크게 신경 쓰지 않아도 된다. 그리 중요하지 않다. 하지만 이는 $y=1$ 일 때 우리가 활용하게 될 새로운 비용 함수이고, 이것이 로지스틱 함수와 꽤 비슷한 함수라는 것을 알 수 있다.

06:29

이것이 서포트 벡터 머신의 연산에 도움을 줄 것이고, 나중에 살펴볼 최적화 문제에서 소프트웨어가 이를 더 쉽게 해결할 수 있게 만들어 줄 것이다. 지금까지 $y=1$ 인 경우를 살펴봤다. 이번에는 $y=0$ 이다. 이 경우에는 비용을 살펴보면 여기 두 번째 항만 필요하다 왜냐하면 첫 번째 항은 없어지기 때문이다. 따라서 $y = 0$ 일 때, 여기 0을 대입하면, 위에 있던 두 번째 항만 남기

때문이다. 따라서 예시의 비용, 즉 비용 함수의 기여는 여기 이 항을 통해 이루어진다. 이를 함수 z 로 그려보면, 가로축에 z 가 있고, 이와 같은 그래프가 생성된다. 서포트 벡터 머신에서는 이 파란선 비슷한 것을 대체하는데, 동시에 새로운 비용으로 여기는 0이므로 평평하고, 이제 여기서부터는 상승 직선이 그려진다.

07:27

이제 이 두 함수에 이름을 붙여보자. 왼쪽에 있는 것은 $\text{cost1}(z)$, 오른쪽은 $\text{cost0}(z)$. 여기서 아래첨자는 이 비용이 $y = 1$ 일 때 대응한다는 것이고, 여기는 $y = 0$ 인 경우이다. 이제 정의를 구했으니, 서포트 벡터 머신을 생성할 준비가 되었다. 여기 로지스틱 회귀 $j(\theta)x$ 가 있다. 이 방정식이 눈에 잘 들어오지 않는다면, 이전에는 $-$ 표시가 밖에 있었지만, 이제 이렇게 마이너스 표시를 안에 넣었기 때문에 달라 보이는 것이다. 서포트 벡터 머신에서 이 부분을 $\text{cost1}(z)$ 로 대체할 건데, 이것이 바로 $\text{cost1}(z) = \text{cost1}(\theta^T x(i))$ 이고, 이 부분은 $\text{cost0}(z) = \text{cost0}(\theta^T x(i))$ 이다. 이전 슬라이드에서 살펴본 cost 1 함수는 이렇게 그려졌고, cost 0 함수는 이렇게 그려졌다. 따라서 서포트 벡터 머신 최소화 문제에서는 다음과 같이 공식을 써볼 수 있다. 이제 서포트 벡터 머신을 위해 조금 다르게 써보려고 한다. 이것을 약간 다르게 다시 매개변수화 하려고 한다. 먼저 $1/m$ 항을 없앤다. 이것은 보통 사람들이 서포트 벡터 머신과 progression을 비교할 때 활용하는 방법과 조금 다르다.

09:44

이렇게 하면 된다. 여기 $1/m$ 항을 없애고, θ 최적값을 얻게 되는 것이다. $1/m$ 항은 내가(삭제) $1/m$ 항은 항상 같으므로 따라서 내가 최소화 문제를 해결 했는 지와 관계없는 θ 최적값을 얻게 될 것이기 때문이다. 따라서 예시를 보여주면, 최소화 문제를 해결하려고 한다. $\min_u (u - 5)^2 + 1$ 를 최소화하면 $U = 5$ 가 된다. 이번에는 이 objective 함수에 곱하기 10을 하려고 한다. 여기 이렇게 최소화 문제가 있는데, $\min_u 10(u - 5)^2 + 10$ 이므로 $u = 5$ 그대로다. 따라서 최소화하려고 하는 것에 무언가를 곱해주면, 똑같이 나오는데, 여기서는 10을 곱해주어도 함수를 최소화하는 u 값은 달라지지 않았다. 다시 말해, 여기 m 값을 삭제하고, objective 함수에 항수 m 을 곱해줘서 θ 값은 변하지 않고 최소값을 구할 수 있게 된다.

11:04

두 번째로 변형할 방법은, 로지스틱 회귀 대신에 SVM을 활용할 때 쓰는 더 일반적인 방법인데, 이는 다음과 같다. 로지스틱 회귀에서 objective 함수에 두 항을 더하였다. 첫 번째는 이 항이고, 비용을 나타내고, 학습 세트에서 왔다. 두 번째는 이 일반화 항이다. 그리고 이 항들을 좀 간략하게 적어보면, $A + \text{정규화 매개 변수 } \lambda x B$ 가 된다. 여기서 A 는 이 항을 가리키고, B 는 여기 람다 없이 이 두 번째 항을 가리킨다. 따라서 우선순위를 매기는 대신에, $A + \lambda B$ 에 여기 정규화 매개 변수 람다 대신에 다른 값을 적용하여, 학습 세트가 A 를 얼마나 최소화할 것인지 아니면 매개 변수 값을 작게 유지할 것인지 중에 결정하여 상대적인 가중치로 바꿔주면 된다. 따라서 B 가 매개변수가 된다.

12:15

서포트 벡터 머신에서 보통 다른 매개 변수를 활용한다. 따라서 람다 대신에 첫 번째와 두 번째 항 사이에 있는 relative weight를 조절할 것이다. 따라서 다른 매개 변수를 활용할 것인데, 보통 C라고 하고, C 곱하기 A 더하기 B 가 된다. 로지스틱 회귀에서 람다 값을 매우 크게 잡으면, B에 매우 고가중치를 주는 것이다. 여기서는 C에 매우 작은 값을 설정하여, B에 A보다 큰 가중치를 주게 된다. 이는 trade off 를 조절하는 다른 방법인데, 첫 번째 항을 최적화 할 것이냐 두 번째 항을 최적화 할 것이냐에 따라 우선수위를 매기는 방법이 달라진다. 그러니까 여기서 $C = 1/\lambda$ 의 역할을 하고 있다고 보면 될 것이다. 이 두 방정식이 같아 질 것이라는 것은 아니다. C는 $1/\lambda$ 와 같고, 이것은 그렇지 않다. $C = 1/\lambda$ 일 때, 여기 두 최적화 objectives 는 같은 값, 같은 최적 θ 값을 출력해야 한다는 것이다. 그래서 여기 람다값을 지우고, 여기에 항수 C를 대입하면 된다.

13:37

지금까지 최적화 objective 함수에 대해 전반적인 내용에 대해 살펴봤다. 여기 이 함수를 최소화하면, SVM으로 학습한 매개 변수를 얻게 되는 것이다.

Q) 다음과 같이 최소화 문제가 있다.

이 두 최적화 문제는 같은 θ 값을 출력할 것이다(같은 θ 값이 두 문제의 최적의 값을 출력한다). 어떠한 경우가 해당되는가?

정답 3번

마지막으로 로지스틱 회귀와 다르게, 서포트 벡터 머신은 확률을 출력하지 않는다. 여기 비용 함수가 있고, 매개변수 θ 값을 얻기 위해 이를 최소화하면, 여기서 서포트 벡터 머신은 $y = 1$ 또는 0 이라고 바로 예측하는 것이다. 따라서 이 가설은 $\theta^T x \geq 0$ 이면 1을 예측하고, 그 외의 경우에는 0을 예측한다. 따라서 매개변수 θ 를 학습하면, 서포트 벡터 머신에서 가설은 이러한 형태를 가지게 된다. 지금까지 서포트 벡터 머신의 수학적 정의를 살펴봤다. 이후 강의에서는 최적화 objective가 어떻게 구현되는지 또는 가설 SVM의 자료가 무엇을 학습할 것 인지에 대해 알아보겠다. 이것을 약간 변형하여 복합 비선형 함수를 배워보도록 하자.

No.	Title
Week 7-2	Large Margin Intuition

00:00

사람들은 때로 서포트 벡터 머신을 large margin classifier 라고 말하기도 한다. 이번 강의에서는 이것의 의미와 SVM 가설이 어떻게 생겼는지에 대한 개요를 그려 볼 것이다. 여기 서포트 벡터 머신의 비용 함수가 있다. 여기 왼쪽에 $\text{cost}_1(z)$ 함수를 그려서 positive 예시로 활용할 것이다. 오른쪽에 $\text{cost}_0(z)$ 를 그렸는데, 여기 가로축이 z 값을 나타낸다. 이제 이 비용 함수를 작게 만드는 방법을 살펴보자. positive 예시가 있을 때, $y = 1$ 일 때, $\text{cost}_1(z) = 0$ 이 되는데, 이는 $z \geq 1$ 한 경우일 때만 그렇다. 즉, positive 예시가 있을 때, $\theta(T)x \geq 1$ 이라는 것이다. 반대로 $y = 0$ 일 때, 여기 $\text{cost}_0(z)$ 함수를 보면, 이 구역에서만, $z \leq 1$ 한 경우일 때만 $\text{cost}_0(z) = 0$ 이 된다.

01:10

이것은 서포트 벡터 머신의 흥미로운 특성인데, positive 예시가 있을 때, $y = 1$ 일 때, 우리에게 필요한 것은 $\theta(T)x \geq 0$ 뿐이다. 이는 우리가 정확히 분류한다는 것인데, 만약 $\theta(T)x \geq 0$ 일 경우, 가설이 0을 예측할 것이기 때문이다. 비슷한 예로, negative 예시에서 필요한 것은 $\theta(T)x < 0$ 뿐이고, 우리가 정확한 예시를 골랐다는 확신을 준다. 하지만 서포트 벡터 머신은 여기에 좀 더 필요한 것이 있다. 그저 이렇게 잘 맞는 예시를 얻지 못한다. 그러니 0보다 조금 크다고 하면 안 된다. 이것이 0보다 훨씬 크다고 해야 한다. 따라서, $\theta(T)x \geq 1$, 그리고 여기는 0보다 훨씬 작아야 하니까 $\theta(T)x \leq -1$ 로 고쳐준다. 이렇게 서포트 벡터 머신에 추가할 safety 요소 또는 safety margin 요소를 생성했다.

02:04

로지스틱 회귀도 이와 비슷한데, 이 것이 서포트 벡터 머신에서는 어떠한 결과로 나타날지 살펴보자. 예를 들면, 이번에는 항수 C 를 매우 큰 값으로 놓고, 그러니까 C 의 값을 매우 크게 잡고, 한 10만 정도 큰 숫자로 잡는다. 그렇게 되면 서포트 벡터 머신이 어떻게 되는지 보자. 만약 C 가 매우 큰 수이고, 이 최적화 objective를 최소화할 때, 우리는 의욕적으로 값을 정해게 되는데, 여기 첫 번째 항은 0이 된다. 이러한 맥락에서 최적화 문제에서 이 첫 번째 항이 0과 같다고 하려면 어떻게 해야 하는지를 이해하도록 하자. 우리는 C 를 매우 큰 항수로 설정하고, 이를 통해 서포트 벡터 머신이 어떠한 종류의 가설을 학습하게 될지에 대해 추가적인 정보를 얻을 수 있을 것이다.

03:06

따라서 학습 예시가 $y = 1$ 이라고 주어질 때 여기 첫 번째 항을 0으로 만들려면, θ 값을 찾는 것인데, 따라서 $\theta(T)x^{(i)} \geq 1$ 이 된다. 이와 비슷하게, 0이 들어간 예시가 있을 때는, $\text{cost}_0(z)$ 에서 우리가 필요한 값을 얻기 위해서는 $\theta(T)x^{(i)} \leq -1$ 와 같은 수식이 필요하다. 만약 최적화 문제에서 매개 변수를 선택해서 여기 첫 번째 항이 0과 같다는 것을 보여주면, 다음과 같은 최적화 문제가 남아있다. 우리는 첫 번째 항 0을 최소화할 것인데, 왜냐하면 매개변수를 선택하고, 이는 0과 같으므로 $Cx_0 + 1/2$ 그리고 여기 두 번째 항 $\sum_{i=1}^n \theta_j^2$ 이 있다. 여기 Cx_0 은 0이기 때문에 지워주고 따라서 $\theta(T)x^i \geq 1$ 은 $y^{(i)} = 1$ 일 경우이고, $\theta(T)x^i \leq -1$ 은 $y^{(i)} = 0$ 일 경우인 것이다.

04:34

이 최적화 문제를 해결할 때, 매개변수 θ 함수를 최소화할 때, 매우 흥미로운 결정 경계를 얻게 된다는 것이다. 예를 들어, positive와 negative 예시가 있는 데이터 세트를 보면, 이 데이터는 선으로 나눌 수 있다는 것이다. 여기 직선이 존재하여, 물론 다른 많은 직선이 있기는 하지만, 이렇게 positive와 negative 예시를 완벽하게 나눌 수 있다는 것이다. 예를 들어, 여기 positive와 negative 예시를 완벽하게 나눌 수 있는 결정 경계가 있다고 하자. 하지만 이는 그렇게 자연스러워 보이지 않는다. 또는 더 이상한 선을 그려보면, positive와 negative 예시를 겨우 분리하는 결정 경계가 또 하나 생긴다. 하지만 둘 다 좋은 선택 같지는 않아 보인다.

05:20

서포트 벡터 머신은 아까와 다른 이와 같은 결정 경계를 선택한다. 검정색으로 그렸다. 이는 아까 그린 자홍색 또는 초록색 결정경계보다 훨씬 나은 결정 경계로 보여진다. 검정선이 훨씬 더 확실한 경계선처럼 보이고, positive와 negative 예시를 분리하는 역할을 더 잘 수행하는 것 같다. 수학적으로는, 검정 결정 경계는 거리(distance)가 더 떨어져있다. 여기서 distance는 여백(margin)을 의미하는데, 여기 파랑색 두 선을 추가로 그리면, 여기 검정 선이 어떠한 학습 예시에서부터라도 최소한의 거리가 큰 것을 알 수 있다. 반면에 자홍색 초록색 선은 학습 예시와 무척 가깝기 때문에 이는 positive와 negative 예시를 분리하는 역할을 검정색 선보다 잘 해내지 못한 것이다. 따라서 이 거리는 서포트 벡터 머신의 여백이 되는 것이고, 이는 SVM에게 특정한 안정성을 주는데, 최대한으로 거리를 두어 데이터를 분리했기 때문이다. 그래서 서포트 벡터 머신이 때로는 large margin classifier라고 불리는 것이고, 이는 이전 슬라이드에 적었던 최적화 문제의 결과로 볼 수 있다.

06:39

이것이 왜 이전 슬라이드에서 봤던 최적화 문제인지, 이것이 어떻게 large margin classifier로 이어지는지 궁금해 할 수도 있다. 아직 설명하지 않았으니 당연하다. 다음 강의에서는 최적화 문제가 large margin classifier로 이어지는 지에 대해 자세히 설명해주도록 하겠다. 하지만 이것은 SVM이 어떠한 가설을 선택할지를 이해하기 위해 기억해둬야 할 유용한 feature가 있는데 이는 positive와 negative 예시를 최대한 분리시키려 하는 것이다.

Q) 아래의 학습 세트에서 x 는 positive 예시 ($y=1$)을 o 는 negative 예시 ($y=0$)을 의미한다고 하자. $SVM(\theta_0 + \theta_1x_1 + \theta_2x_2 \geq 0$ 일 때, 1을 예측)을 학습한다고 가정할 때, SVM에서 출력할 $\theta_0, \theta_1, \theta_2$ 값은?

정답 2번

이번에는 large margin classifiers에 대해 마지막으로 하나 더 짚고 넘어가자. 우리는 이와 같은 large margin classification 설정을 활용했는데, 이는 C가 매우 클 때, 아마 10만이라고 했던 것 같다. 이와 같은 데이터 세트를 가지고 이렇게 positive와 negative 예시를 large margin으로

나누어줄 결정 경계를 선택할 것이다. 이제 SVM는 이 large margin이 보이는 것보다 약간 더 복잡해졌다. 특히, large margin classifier를 활용하면, 여러분의 학습 알고리즘이 outlier에 민감하게 반응할 수 있는데, 그렇다면 화면에 보이는 것과 같이 먼저 positive 예시를 더 추가해보자.

07:56

만약 예시가 하나 주어졌고, large margin으로 데이터를 구분하려고 하면, 결정 경계는 이와 같이 자홍색 선으로 그려질 것이다. 따라서 하나의 outlier, 예시가 추가 된다고 해도 이 결정 경계를 검정색 선에서 자홍색 선으로 바꿔야 할지는 확신이 서지 않는다. 만약 regularization 정규화 매개변수 C 가 매우 크다면, SVM은 다음과 같이 결정 경계를 검정색 선에서 자홍색 선으로 바꿀 것이다. 하지만 C 값이 작다면, 그렇게 크지 않다면, 검정색 선으로 결정 경계를 선택하게 된다. 물론 데이터가 선으로 분리되지 않는다면, 여기에 positive 예시, 여기에 negative 예시가 있었다면, SVM은 이에 맞게 구현될 것이다.

08:53

따라서 이 large margin classifier는 regularization 정규화 매개 변수 C 가 매우 큰 경우에만 해당되는 그래프를 보여주고 있다. 다시 말해, C 의 역할은 $1/\lambda$ 와 같은데, 여기서 람다는 이전에 설명했다시피, regularization 정규화 매개 변수다. 따라서 $1/\lambda$ 이 매우 크거나, 람다값이 매우 작을 때, 여기 자홍색 선과 같은 결정 경계를 얻을 수 있게 된다. 하지만 실제로 서포트 벡터 머신을 적용해보면, C 가 이와 같이 엄청 크지 않을 때, 여기 outliers 이상치를 제외하고 잘 구현될 수 있다. 또한 여러분의 데이터가 직선으로 구별되지 않는다고 해도 잘 구현될 수 있다.

09:43

하지만 bias 와 variance를 서포트 벡터 머신의 맥락에서 이야기해보면, 이는 나중에 또 다뤄볼 것이지만, 이러한 regularization 매개 변수를 포함한 trade-off 상호 절충은 그 때가 되면 더 분명해질 것이다. 이번 강의를 통해 large margin classifier 가 large margin 으로 데이터를 분리하려고 할 때 서포트 벡터 머신이 어떻게 작동하는지에 대해 살펴보았다. 엄밀히 말해, 아까의 그래프는 매개변수 C 가 매우 클 때만 맞는 내용이고, 서포트 벡터 머신에 대해 설명한 효율적인 방법이다. .

이번 강의에서 한 가지 내용이 빠졌는데, 이 슬라이드에 적었던 최적화 문제다. 이것들이 어떻게 large margin classifier를 유도하는지 이번 강의에서 설명하지 않았지만, 다음 강의에서 수학적으로 이를 설명하여, 최적화 문제가 어떻게 large margin classifier에 이러한 결과를 도출해냈는지 알아보도록 하자.

No.	Title
-----	-------

00:00

이번 강의에서는 large margin classification를 수학적으로 표현해보도록 하겠다. 이번 강의는 선택사항이므로, 꼭 수강하지 않아도 좋다. 이번 강의는 서포트 벡터 머신에서 최적화 문제와 이것이 어떻게 large margin classification로 이어지는지에 대해 자세히 다뤄볼 것이다.

먼저 벡터의 내적의 특성 몇 가지를 살펴보도록 하자. 여기 이렇게 벡터 U 와 벡터 V 가 있다고 하자. 둘 다 2차원 벡터다. 그리고 $U^T V$ 또한 벡터 U 와 벡터 V 사이에 있는 내적이다. 여기 2차원 벡터를 활용하여, 그 수치를 그려보겠다. 이게 벡터 U 라고 하면, 여기 가로축이 u_1 값을 가지고, 세로축에는 u_2 값을 가진다고 하자. U 는 벡터의 두 번째 구성요소다.

01:10

이제 유용한 기호를 하나 활용하겠는데, 바로 $\|u\|$ 다. 여기 양쪽에 바 2개는 벡터 U 의 길이를 의미한다. 즉, 벡터 U 의 유clidean 거리를 말한다. 피타고拉斯 정의에 의하면, $\|u\| = \sqrt{u_1^2 + u_2^2}$ 이고, 이는 벡터 U 의 길이를 말한다. 이는 실수이다. 여기서 말하는 벡터의 길이란 여기 이렇게 그린 화살표의 길이를 말한다. 이제 벡터 V 를 살펴보면서 내적을 연산해보자. 따라서 V 는 이렇게 v_1, v_2 값을 가지게 된다. 벡터 V 는 이렇게 V 를 향해서 이렇게 그려진다. 이제 벡터 U 와 벡터 V 사이에 내적을 연산하는 방법을 살펴보자. 방법은 다음과 같다. 벡터 V 를 가지고 벡터 U 위에 투영시킨다. 이와 같이 직각 또는 90도로 내려와서 벡터 U 에 투영하는 것이다. 그리고 여기 그려진 빨간 선의 길이를 측정할 것이다. 여기 빨간 선을 P 라고 하겠다. P 는 길이 또는 벡터 V 가 벡터 U 위에 투영된 정도다. 다시 적어보자. P 는 P 는 벡터 V 가 벡터 U 위에 투영된 길이를 말한다. 또한, $U^T V = P * \|u\|$ 가 된다. 이것이 내적을 연산하는 하나의 방법이다.

03:23

실제로 기하학으로 계산해보면, P 값과 $\|u\|$ 을 계산해보면, 아까와 같은 방법으로 내적을 연산하는 것과 같은 답을 도출해 낼 것이다. 연산을 해보면, $U^T V$ 에서 U 를 전치하면, 이렇게 1×2 행렬과 행렬 V 를 곱해주면 되는데, 따라서, $U^T V = u_1 v_1 + u_2 v_2$ 가 된다. 선형 대수학 정의에서 이 두 공식은 같은 값을 도출한다. 또한 $U^T V = V^T U$ 도 성립한다. 따라서 이와 같은 과정을 반대로 V 를 U 에 투영하는 것이 아니라 U 를 V 에 투영하는 것이다. 그렇게 같은 과정을 거치면, 벡터 U, V 의 행이 반대로 되고, 실제로 어떤 값이든지 간에 같은 값을 얻게 된다.

04:17

다시 말하자면, 이 $\|u\|$ 의 값은 실수이고, P 의 값 또한 실수다. $U^T V$ 는 두 실수, p 의 길이와 $\|u\|$ 의 일반적인 곱셈인 셈이다. 한가지 더 살펴보자면, P 는 사실 부호를 지니고 있다. 따라서 양수 또는 음수다. 즉, U 벡터가 이와 같다면, V 는 이렇게 있다고 하자. 이처럼 U 와 V 사이 각도가 90도를 넘으면, V 를 U 에 투영하려고 하면, 이처럼 투영하면 된다. 그래서 P 의 길이가 된다. 이 경우에도

$U^T V = P * \|u\|$ 가 성립하는 것이다. 하지만 이 경우에 P 값은 음수가 될 것이다. U 와 V 사이 각도가 90도 보다 작으면, P 길이는 빨간 선처럼 positive 값이 되고, U 와 V 사이 각도가 90도를 넘는 경우에는, P 길이는 여기 작은 부분인 negative 값이 될 것이다. 따라서 두 벡터 사이의 내적은 둘 사이의 각도가 90보다 크면 negative 값을 가지게 된다.

05:43

이것이 벡터 내적이 처리되는 과정이다. 이러한 벡터 내적의 특성을 활용하여 서포트 벡터 머신에서 최적화 objective를 이해하는 데 활용해볼 것이다. 여기 전에 살펴본 서포트 벡터 머신에서 최적화 objective 가 있다. 이 슬라이드에서 사용할 목적으로 간소화, 즉 objective를 분석하기 쉽게 만들려고 한다. 따라서 $\theta_0 = 0$ 이라고 하자. 이를 간편하게 그리기 위해서 feature의 수인 $n=2$ 이라고 하겠다. 그래서 여기에는 feature가 x_1, x_2 2개 있다. 이제 objective 함수, SVM의 최적화 objective 를 살펴보자. $n=2$ 일 때, feature는 두 개뿐이다. 이는 $1/2(\theta_1^2 + \theta_2^2)$ 과 같은데, 매개변수가 θ_1, θ_2 2개뿐이기 때문이다. 이를 다시 써보면, $1/2(\sqrt{\theta_1^2 + \theta_2^2})^2$ 와 같이 쓸 수 있는데, 어떤 수 w 는 $w = (\sqrt{w^2})^2$ 이 성립하기 때문이다. 제곱근과 제곱을 계산하면 그 안의 것만 남는다.

07:08

여기 보면 이 부분이 $\|\theta\|$ 또는 벡터 θ 의 길이와 같다는 것을 알 수 있다. 즉, 벡터 $\theta = [\theta_1; \theta_2]$ 라고 하면, 여기 빨간색 밑줄 그은 부분이 $\|\theta\|$ 또는 벡터 θ 의 길이와 같다. 아까 살펴본 벡터의 노름(norm)의 정의와 같은 것이다. 따라서 $\theta_0, \theta_1, \theta_2$ 쓴다고 해도 벡터 θ 의 길이와 같게 된다. 이는 여기 가정한대로, $\theta_0 = 0$ 일 때 또는 θ_1, θ_2 의 길이가 되는데, 여기서는 θ_0 는 무시하도록 하자.

그러므로 여기서 θ 는 보통 θ 로 θ_1, θ_2 만 활용하도록 하자. θ_0 를 추가하든 하지 않은 연산은 가능하고, 나머지 연산에 영향을 미치지 않는다. 따라서 이것은 $1/2\|\theta\|^2$ 과 같다. 따라서 최적화 objective 에서 서포트 벡터 머신의 역할은 이 $\|\theta\|^2$, 즉 매개변수 벡터 θ 의 길이를 최소화하는 것이다.

08:28

이제 $\theta^T X$ 항의 역할을 살펴보도록 하자. 주어진 $\theta^T X(i)$ 는 어떻게 구할 수 있을까? 이전 슬라이드에서 다른 벡터 U 와 V 를 가지고 $U^T V$ 에 대해 살펴봤다. 아까 내린 정의를 가지고 θ 와 $X(i)$ 가 u, v 의 역할을 하면 어떻게 그려지는지 알아보자. 먼저 학습 예시 하나를 그려보도록 하겠다. 여기 positive 예시 하나를 x 표시로 그려주고, 이게 예시 $X(i)$ 라고 하자. 이는 가로축에는 $X(i)$ 1, 세로축에는 $X(i)$ 2 값을 가진다는 것이다. 이렇게 학습 예시 하나를 그려봤다. 이것을 벡터라고 생각하지 않았지만, 이는 원점 (0,0)에서 시작해 이어지는 학습 예시다.

09:37

이제 매개 변수 벡터를 가지고 벡터로 이렇게 그려보도록 하겠다. 여기는 θ_1 , 여기가 θ_2 가 된다. 그렇다면 $\theta^T X(i)$ 는 어떻게 구할 수 있을까? 아까 방법을 사용하면, 예시를 가지고 매개변수 벡터

θ 에 투영하는 것이다. 여기 빨갛게 표시한 부분의 길이를 알아보도록 하겠다. 그리고 매개변수 벡터 θ 에 i 번째 학습 예시를 투영한 것을 $P(i)$ 라고 하겠다. 그렇게 되면, $\theta^T X(i) = P * \|u\|$ 아까 슬라이드에서 살펴본 것이다. 이는 물론 $\theta_1 X(i)_1 + \theta_2 X(i)_2$ 와 같다. 따라서 이 둘은 θ 와 $x(i)$ 사이의 내적을 연산하는데 동등하게 유효한 방법이다.

10:57

그렇다면, 이것은 무엇을 의미할까? 이러한 제약은 $\theta^T X(i) \geq 1$ 또는 $\theta^T X(i) \leq -1$ 은 무엇을 의미하는가? 이러한 제한이 $P(i)X \geq 1$ 으로 대체될 수 있다는 것이다. 이는 $\theta^T X(i) = P * \|u\|$ 이기 때문이다. 최적화 objective로 나타내면 다음과 같다. $\theta^T X(i)$ 대신에 $P(i) * \|u\|$ 를 써줬다. 여기 다시 한번 정리하면, 여기 최적화 objective는 $1/2\|\theta\|^2$ 와 같게 된다. 여기 아래에 있는 학습 예시를 살펴보자. 여기 단순화로 $\theta_0 = 0$ 을 계속 활용하도록 한다. 이제 어떤 결정 경계를 서포트 벡터 머신이 선택할지 알아보자. 이 예시를 살펴보자. 이 서포트 벡터 머신이 이와 같은 결정 경계를 선택하려고 한다. 이는 그리 좋은 선택은 아닌데, 바로 margin이 너무 작기 때문이다. 그래서 결정 경계가 학습 예시와 무척이나 가깝다. 서포트 벡터 머신이 이와 같은 선택을 하지 않는지 알아보자.

12:14

이러한 매개 변수를 선택할 때, 매개변수 벡터 θ 가 결정 경계와 90도를 이루 가능성이 있다. 그렇기 때문에 여기 초록색 결정 경계는 이 방향을 가리키는 매개변수 벡터 θ 와 대응한다. 여기 단순화로 $\theta_0 = 0$ 이 있는데, 이는 결정 경계가 여기 있는 원점 $(0,0)$ 을 반드시 통과해야 한다는 것이다. 이것이 최적화 objective에서 무엇을 의미하는지 살펴보자. 여기 예시를 살펴보겠다. 이게 내 첫 번째 예시, x_1 이라고 하자. 이 예시를 매개 변수 θ 에 투영해보자. 이렇게 하면 되는데, 여기 작은 빨간 선 부분이다. 이 부분은 p_1 이 된다. 매우 작은 부분이다. 또한, 여기 예시를 보면, 이 것은 x_2 , 두 번째 예시가 된다. 그리고 이 예시를 θ 에 투영하면, 이것을 자홍색으로 그려보겠다. 여기 작은 자홍색 선으로 표시한 부분이, P_2 가 되는 것이다. 이는 두 번째 예시가 매개 변수 벡터 θ 의 방향으로 이렇게 투영된 것을 나타낸다. 따라서 이 작은 투영된 선 부분은 매우 작다. 여기서 p_2 음수가 되는데, p_2 가 반대 방향으로 가고 있기 때문이다. 이 벡터가 매개변수 θ 와 이루는 각도가 90도보다 크기 때문에, 이는 0보다 작다.

13:50

따라서, $P(i)$ 는 매우 작은 수라는 것을 알 수 있다. 최적화 objective를 보면, positive 예시에서 $P(i) * \|u\| \geq 1$ 이인데, 여기에서 $P(i)$ 가 있는데, 만약 p_1 의 값이 매우 작을 경우, $\|u\|$ 값이 매우 커져야 한다는 것을 의미한다. $P_1(\theta)$ 값이 작으면, $P(1) * \|\theta\| \geq 1$ 를 성립하려면, $P(1)$ 이 작을 경우, 여기 $\|u\|$ 의 값을 크게 하는 것 만이 이 수식을 성립할 수 있게 하는 유일한 방법이다. negative 예시도 똑같이 살펴보면, $P(2) * \|\theta\| \leq -1$ 이 필요한데, 우리는 이미 p_2 값이 매우 작을 것이라는 것을 알고 있기 때문에, 이 공식을 성립하게 하려면, $\|u\|$ 값이 아까처럼 커져야 한다. 하지만 우리가 최적화 objective를 다룰 때, $\|\theta\|$ 값이 작은 매개 변수를 찾으려고 하기 때문에

이는 매개변수 벡터 θ 의 방향으로 그다지 좋은 방법은 아닌 것 같다.

15:13

반면에, 여기 다른 결정 경계가 있다. 여기서 SVM이 이와 같은 결정 경계를 선택했다고 하자. 그러면 아까와 매우 다르다. 만약 결정 경계가 이렇다면, 이것이 이에 대응하는 θ 방향이다. 따라서 여기 결정 경계는 수직선인데, 이와 대응하는 것인데, 이와 같이 초록색 결정 경계를 얻는 방법은 선형 대수학을 통해 보여줄 수 있다. 벡터 θ 를 90도로 잡는 것이다. 이제 여러분의 데이터를 벡터 x 에 투영하는 과정을 알아보자. 이것을 예시 x_1 이라고 하자. 이것을 θ 에 투영하면, 여기 이 길이가 p_1 이 된다. 또 다른 예시를 보면, 이것을 x_2 라고 하고 똑같이 투영하면, 여기 길이가 p_2 가 된다. 그리고 이것은 0보다 작다. 여기 보면, p_1, p_2 의 투영 길이가 훨씬 길어진 것을 볼 수 있다. 만약 아직 이러한 제한이 있다면, $P(1) * \|\theta\| \geq 0$ 에서 $p(1)$ 이 훨씬 커졌기 때문에, 이제 $\|\theta\|$ 는 작아질 수 있다.

16:41

따라서 이는 왼쪽 대신 오른쪽과 같이 결정 경계를 선택한 경우 SVM이 $\|\theta\|$ 을 훨씬 작게 만들 수 있다는 것이다. 따라서 우리가 $\|\theta\|$ 를 작게 만들면, $\|\theta\|$ 의 제곱도 작아지므로 SVM이 오른쪽 가설을 선택하게 되는 것이다. 이것이 바로 SVM이 large margin certification effect **효과**를 증가시키는 방법이다. 중요한 것은, 여기 초록색 선, 초록색 가설을 보면, positive negative 예시가 θ 에 크게 투영되기를 원했고, 이게 참이 되기 위해서는 이렇게 초록색 선을 주변을 감싸는 것이다. 여기 보면 이렇게 margin이 크게 있고, 이 큰 gap은 positive와 negative 예시를 분리시켜주고, 여기 gap, 즉 margin의 정도가 p_1, p_2, p_3 등의 값이 된다. 따라서 margin이 커지면, 여기 p_1, p_2, p_3 가 커지면, SVM이 $\|\theta\|$ 작은 값을 얻을 수 있고, 이것은 바로 이 objective에서 얻고자 하는 것이다. 이것이 왜 이 머신이 large margin classifiers이 되는지를 말해주는데, p_1 값을 최대로 만들어서 학습 예시와 결정 경계 사이의 거리를 최대로 만들기 때문이다.

18:14

마지막으로 우리는 단순화 $\theta_0 = 0$ 를 활용하여 이 모든 연산을 작업했다. 이로 인한 효과는, 아까 간략하게 설명했지만, $\theta_0 = 0$ 일 경우, 결정 경계가 원점을 지나게 되는데, 만약 θ_0 가 0이 아니게 되면, 이는 이와 같이 원점을 지나지 않는 결정 경계 또한 고려해야 한다는 것이다. 이와 같은 경우를 연산해보지는 않을 것이다. 결국 이만큼의 large margin 증명 작업은 거의 똑같을 것이다.

이러한 주장을 일반화하여 이미 살펴보았고, 이러한 최적화 objective를 가질 때, θ_0 가 0이 아닐 경우 SVM은 C가 가장 큰 경우에 대응할 것이다. 하지만 $\theta_0 \neq 0$ 경우라도 이 서포트 벡터 머신은 positive와 negative 예시를 넓은 margin으로 분리시킬 방법을 찾으려고 노력할 것이다.

Q) 우리가 활용한 SVM 최적화 문제는 다음과 같다

$P(i)$ (양수나 음수) 는 $x(i)$ 가 θ 에 투영된 것을 가리킨다. 위와 같은 학습 세트가 주어졌다. θ 가 최적값을 가질 때, $\|\theta\|$ 값은?

정답 2번

따라서 이는 서포트 벡터 머신이 large margin classifier라는 것을 설명한다. 다음 강의에서는 이러한 SVM 아이디어를 가지고 복잡한 비선형 classifier를 생성하는데 적용해보도록 하겠다.

No.	Title
Week 7-4	Kernels I

00:00

이번 강의에서는 서포트 벡터 머신을 적용하여 복합 비선형 classifier를 개발하도록 하겠다. 이와 같은 작업을 하는 주요 기술을 kernels라고 부른다. Kernels가 무엇인지 그리고 어떻게 활용하는지 알아보자. 이와 같이 학습 세트가 주어졌을 때, 비선형 결정 경계를 찾아서 positive와 negative 예시를 구분하려고 한다. 결정 경계는 이와 같이 생성될 것이다. 한 가지 방법은 복합 다항 feature를 생각해내는 것이다. 이렇게 feature 세트가 준비되면, 모든 θ 항의 합이 0보다 크거나 같으면 $h(\theta)x = 1$ 이고, 그 외의 경우에 $h(\theta)x = 0$ 이 된다. 또 다른 방법으로는 내가 나중에 활용할 표기법의 잠깐 설명하자면, 이 것은 결정 경계를 연산하는 가설에서 이와 같이 사용할 수 있다. $\theta_0 + \theta_1f_1 + \theta_2f_2 + \theta_3f_3 \dots$ 이렇게 쭉 이어진다. 여기 f_1, f_2, f_3 새로운 표기법을 활용하여, 여기 연산하려고 하는 새로운 feature를 표기하는데 활용할 것이다. $F_1 = x_1, F_2 = x_2, F_3 = x_1x_2, F_4 = x_1^2, F_5 = x_2^2$ 등이 된다.

01:38

우리는 이와 같이 고차다항식으로 다양한 feature를 생각해낼 수 있다는 것을 배운 적 있다. 여기서 문제는 여기에서 다른 feature를 선택할 수 있는지 또는 이 고차다항식 말고 더 나은 종류의 feature가 있는가이다. 이는 이 고차다항식이 우리가 원하는 것인지 확실치 않다는 것이다. 예전에 컴퓨터 비전에 대해 이야기 했을 때, 이미지에 많은 픽셀을 입력할 때, 고차다항식을 활용하는 것은 이와 같은 고차다항이 많기 때문에 연산량이 매우 많아질 수 있다. 그렇다면, 이와 같은 가설에 입력할 수 있는 다른 또는 더 나은 feature를 선택할 수 있을까?

02:18

먼저 새로운 feature f_1, f_2, f_3 를 정의하는 방법이다. 여기서는 feature 3개만 정의하겠지만, 하지만 실제 문제에서는 훨씬 더 많은 수를 정의하게 될 것이다. 여기 feature x_1, x_2 가 있고, 여기서는 interceptor x_0 를 포함하지 않을 것이다. 여기 x_1, x_2 에서 수동으로 몇 지점을 선택하겠다. 먼저

여기는 l_1 , 또 다른 곳은 l_2 , 그리고 세 번째로 l_3 가 있다. 이번에는 이렇게 3 지점을 수동으로 선택해봤다. 이 세 지점을 landmark라고 부르고, $landmark1$, $landmark2$, $landmark3$ 라고 하겠다. 이제 여기 새로운 features를 다음과 같이 정의하겠다.

03:07

예시 x 이 주어졌을 때, $f_1 = \text{similarity}(x, l(1))$ 이고, 유사성을 측정하기 위해 활용할 이 특정 공식은 $\text{similarity}(x, l(1)) = \exp(-\|x - l(1)\|^2 / 2\sigma^2)$ 이다. 이전에 선택사항이었던 강의를 봤다면 알겠지만, 이 표시는 벡터 w 의 길이를 말한다. 따라서 여기서는 $\|x - l(1)\|^2$ 은 유clidean 거리를 말하는데, 이 유clidean 거리는 x 지점과 l_1 사이 거리를 말한다. 이에 대해서는 후에 더 설명해주겠다. 이게 첫 번째 feature고 두 번째 feature $f_2 = \text{similarity}(x, l(2)) = \exp(-\|x - l(2)\|^2 / 2\sigma^2)$ 이고, $f_3 = \text{similarity}(x, l(3))$ 그리고 위의 식과 마찬가지로 정의 될 수 있다. 여기서 유사도 함수(similarity function)는 수학적인 용어로 kernel 함수가 된다.

04:54

그리고 여기서 활용하는 특정 kernel은 Gaussian kernel이라고 한다. 이 공식은 여기 유사도 함수(similarity function) 중 특정한 형태로 Gaussian kernel이라고 한다. 이 용어는 추상적으로 이렇게 다른 유사도 함수(similarity function)를 kernels라고 하고, 서로 다른 유사도 함수(similarity function)와 특정 예시를 가질 수 있는데, 이는 Gaussian kernel이라고 한다. 다른 kernel에 대한 예시도 살펴볼 것이지만, 지금은 유사도 함수(similarity function)로 일단 기억해두자. 따라서 x 와 l 사이를 쓸 때, 이 kernel을 사용하여, $k(x, l(i))$ 로 쓰기도 한다. 이제 이 kernel이 어떤 역할을 하는지, 이러한 종류의 similarity 함수가 그리고 이러한 표현이 맞는 건지 알아보도록 하자.

05:46

먼저 첫 번째 landmark부터 보자. $landmark l_1$ 은 내가 방금 선택한 지점 중 하나이다. 여기서 x 와 l_1 사이에서 similarity of kernel은 이와 같다. 모두 잘 따라오고 있는지 확인 차원에서, 여기서 numerator 분자항은 다음과 같은 합계 공식으로도 거리를 나타낼 수 있다. 여기서는 벡터 x 와 벡터 l 사이의 component wise 거리를 나타낸다. 그리고 이번 슬라이드에서도 x_0 을 무시하겠다. Intercept 항인 $x_0 = 1$ 은 무시해도 된다. 이것이 바로 x 와 landmark 사이의 kernel with similarity를 연산하는 방법이다. 이번에는 이 함수가 어떻게 활용되는지 살펴보자. $X \approx l(1)$ 라고 가정하자. 그럼 유clidean 거리 공식과 numerator는 0과 가까워진다. 이는 여기 이 항의 x 를 활용한 거리는 0과 가까워 진다. $F_1 \approx \exp(-0^2/2\sigma^2) \approx 1$ 이 성립한다. 여기 \approx 근접하다는 기호를 사용한 이유는 이 거리가 0이 아닐 수 도 있기 때문이다. 하지만 만약 x 가 landmark 와 가까워지면, 이 항은 0과 가까워질 것이고, 그럼 f_1 도 1과 가까워진다.

07:13

반대로, x 가 l_1 과 거리가 먼 경우, 여기 첫 번째 feature, $f_1 = \exp(-(large\ number)^2/2\sigma^2) \approx 0$ 이 된다. 이와 같은 features는 x 가 여러분의 landmark 중 하나와 얼마나 비슷한지를 측정하고, x 가 landmark에 가까워지면, feature f 는 1과 가까워진다. 또한 x 가 landmark와 멀어지면, feature x 는

0이거나 0과 가까워진다. 여기 각각의 landmark는 이전 페이지에서 이 landmark를 l1, l2, l3 이렇게 표시했었다. 이 각각의 landmark l1, l2, l3은 각각 feature f1, f2, f3를 의미한다. 학습 예시 x 가 주어졌을 때, 방금 적은 l1, l2, l3이 주어지면, 이제 3개의 새로운 feature f1, f2, f3를 연산할 수 있다.

08:12

하지만 먼저 여기 \exp 함수를 살펴보자. 여기 similarity 함수를 보고 수치 몇 개를 그려보면서 어떤 함수인지 잘 이해하도록 하자. 예시로, 여기 x_1, x_2 feature 두 개가 있다. 첫 번째 landmark $l_1 = [3; 5]$ 라고 하자. 그리고 이번에는 시그마의 제곱이 1이라고 하자. 이 수식을 그려보면, 이와 같은 그림이 나온다. 따라서 세로축은 표면의 높이 f_1 값이고, 여기 아래 가로축은 학습 예시가 주어졌다면, x_1, x_2 가 된다. 이와 같이 특정 학습 예시가 주어지면, x_1, x_2 값을 가지는 학습 예시가 되고, 여기 높이는 f_1 값이 된다. 여기 아래는 같은 수치로 그림 그래프인데, 수량화 그래프를 활용하여, 가로축은 x_1 , 세로축은 x_2 로 표시하고, 여기 있는 수치를 3D 등고선으로 나타내면 이렇게 된다.

09:15

$X = [3; 5]$ 일 때, f_1 값은 1이 되고, 이게 최대값이기 때문이다. 그리고 X 가 이쪽으로 움직여서 멀어지면, 이 값은 0과 근접한다. 따라서 이것이 f_1 이 x 가 첫 번째 landmark와 얼마나 가까운지를 측정한 것이다. 그리고 x 가 첫 번째 landmark와 얼마나 가까운지에 따라 그 값은 0부터 1 범위에서 달라진다. 이번에는 여기 매개변수 시그마 제곱 값을 다르게 할 때 어떻게 달라지는지를 살펴볼 것이다. σ^2 는 Gaussian kernel 매개변수인데, 이를 바꿔주면 조금씩 다른 결과가 나온다. 먼저 $\sigma^2 = 0.5$ 라고 하고 값을 구해보자. $\sigma^2 = 0.5$ 라고 하고 여기서 kernel은 비슷해 보이지만, 여기 너비가 줄어들었다. 등고선 또한 줄어들었다. 따라서 $\sigma^2 = 0.5$ 일 때, $X = [3; 5]$ 부터 시작해보면, 이 방향으로 옮기면, feature f_1 은 더 급격히 0으로 떨어진다. 반대로, $\sigma^2 = 3$ 으로 증가시키면 이 지점은 $l_1, [3; 5]$ 인데, σ^2 값이 커지면 l_1 에서부터 0에 가까워질 때 feature 값은 천천히 감소한다.

Q1) 다음은 x_1 값을 가진 1차원 그래프 예시다. $l(1) = 0.5$ 라고 가정하자. $\sigma^2 = 1$ 일 때, $f_1 = \exp(-||x - l(1)||^2 / 2\sigma^2)$ 이 성립한다. $\sigma^2 = 4$ 로 바꿔면, 새로운 값 σ^2 를 그린 f_1 그래프는?

정답 3번

11:03

이렇게 feature의 정의가 주어졌을 때, 어떠한 가설 자료를 학습 할 수 있는지 알아보자. 학습 예시 x 가 주어졌을 때, feature f_1, f_2, f_3 의 값을 연산해보자. 여기서 $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$ 일 때, 가설은 1을 예측한다. 이 예시에서는 학습 알고리즘을 이미 알고 있고, 이러한 매개 변수 값을 얻었다고 하자. $\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0$ 이다. 이제 학습 예시가 주어져서 그 위치를 자홍색 점으로 이런 식으로 표시하면 어떤 변화가 생기는지 알아보자. 여기 학습 예시

x 가 주어졌을 때 이 가설은 무엇을 예측할까? 이 공식을 보면, 이 학습 예시 x 는 I_1 에 근접하므로, $f_1 \approx 1$, 또한 학습 예시 x 는 I_2, I_3 와 멀기 때문에, $f_2 \approx 1, f_3 \approx 0$ 가 된다. 이제 이 공식에 대입해보면, $\theta_0 + \theta_1 * 1 + \theta_2 * 0 + \theta_3 * 0$ 이 되는데, θ_2, θ_3 뒤의 0은 정확히 0은 아니고 0과 가깝다는 것이다. 이제 이 값을 대입해보면 $-0.5 + 1 = 0.5 \geq 0$ 가 나온다.

12:50

이번에는 $y = 1$ 을 예측해보자. 이는 0보다 크거나 같기 때문이다. 조금 다르게 생각해보자. 다른 관점에서 다른 색으로, 청록색으로 여기 다른 학습 예시 x 를 표시했다. 이제 아까와 비슷하게 연산을 해보면, f_1, f_2, f_3 는 모두 0에 근접하다. 따라서 $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \approx 0.5$ 가 나오는데, $\theta_0 = -0.5$ 이고, f_1, f_2, f_3 은 모두 0이기 때문이다. 따라서 -0.5이고 이는 0보다 작다. 따라서 여기 이 지점은 $y=0$ 이라고 예측할 수 있는 것이다. 여러분이 혼자서 다른 지점으로 연산을 해본다면, I_2 와 가까운 학습 예시를 가지고 이와 같은 지점에 표시한다면, 이 것은 $y=1$ 을 예측할 것이다. 이제 이 경계를 넓게 보면, 이 공간에서 I_1 과 I_2 과 가까운 지점은 positive를 예측하고, I_1 과 I_2 이 두 landmark와 먼 지점은 0을 예측한다는 것을 알 수 있다. 즉, 이 가설의 결정 경계는 이와 같이 그려질 것이고, 여기 빨간 결정 경계 안에서는 $y=1$ 로 예측되고, 그 바깥은 $y=0$ 으로 예측될 것이다.

14:34

지금까지 landmark와 kernel function 의 정의에 대해 알아봤다. 방금 그린 것과 같이 이 두 개의 landmark와 가까운 지점은 positive를 예측하는 꽤 복잡한 비선형 결정경계도 학습했다. 또 이 landmark와 먼 곳은 negative를 예측했다. 따라서 이것이 kernel의 개념이고, 이를 서포트 벡터 머신에서 어떻게 활용하는지, 이는 landmark와 similarity 함수를 이용하여 추가 features를 정의하고, 더 복잡한 비선형 분류기를 학습해보았다. 이 강의를 통해 kernel이 무엇인지, 이 것이 서포트 벡터 머신에서 새로운 feature를 정의하기 위해 어떻게 활용되는지를 이해할 수 있었기를 바란다. 하지만 아직 답을 얻지 못한 문제가 몇 개 더 남았다. 하나는 이러한 landmark를 어떻게 얻는가? 또 어떻게 선택하는가 이고, 다른 하나는 다른 similarity 함수는 뭐가 있으며, 있다면 우리가 활용한 Gaussian kernel 이외에 다른 것도 활용할 수 있는가? 다음 강의에서는 이 질문에 대한 답을 얻고, 다 하나로 합쳐서 kernel를 이용한 서포트 벡터 머신이 복합 비선형 함수를 학습하는데 얼마나 효과적인지 살펴보도록 하자.

No.	Title
Week 7-5	Kernels II

00:00

지난 시간에는 kernel에 대해서 설명했는데 이것이 서포트 벡터 머신에서 새로운 feature를

어떻게 정의하는데 활용되는지 살펴봤다. 이번 시간에는 지난 번에 설명하지 못한 내용을 이어가려고 한다. 그리고 실제로 이러한 아이디어를 어떻게 활용하는지에 대해 설명하고자 한다. 예를 들어, 서포트 벡터 머신에서 bias, variance, trade-off와 어떻게 관련이 있는지 살펴보겠다. 지난 시간에 landmark를 선택하는 과정에 대해 이야기했었다. l_1, l_2, l_3 는 이와 같은 kernel이라고 하는 similarity 함수를 정의할 수 있게 해주었다. 이 예시처럼 similarity 함수가 있을 경우, 이것은 Gaussian kernel이라고 했다. 그리고 이와 같은 가설 함수를 세울 수 있도록 했다. 하지만 이러한 landmark를 어디서 얻게 되는가? 어디서 l_1, l_2, l_3 를 얻게 되는가? 복잡한 학습문제에서는 세 개 이상의 landmark가 더 필요할 것이고, 손으로 직접 선택해야 할지도 모른다.

00:54

실제로는 landmark는 다음과 같이 정해진다. 바로 이렇게. 기계 학습 문제에서 positive 와 negative 예시가 있는 데이터 세트가 있다. 이 예시를 가지고, 여기 있는 모든 학습 예시를 가지고 가서 학습 예시가 표시되어 있던 그 자리에 똑같이 landmark를 표시한다. 여기에 x_1 예시가 하나 있고, 이제 첫 번째 학습 예시가 있던 똑같은 자리에 있는 지점을 첫 번째 landmark라고 정한다. 여기 다른 예시 x_2 가 있다. 그리고 두 번째 학습 예시가 있던 똑같은 자리에 있는 지점을 두 번째 landmark라고 한다. 오른쪽에 있는 수치에는 이를 표시하기 위해서 빨간색과 파랑색 점을 활용했다. 이 수치의 색상은, 점의 색상은 중요하지 않다. 이 방법을 활용하여, m landmark 를 활용하여 $l_1, l_2 \dots, l_m$ 까지 학습 예시를 얻게 되고, m 개의 학습 예시가 있고, 학습 예시마다 그와 같은 지점에 하나의 landmark가 있는 것이다. 내 feature가 학습 세트에서 봤던 예시와 얼마나 가까운지를 측정하기 때문에 좋은 자료다.

02:12

이러한 개념을 좀더 확실히 정리해보면, m 개의 학습 예시가 주어질 때, landmark의 위치를 m 학습 예시의 위치와 거의 일치하게 선택할 것이다. x 예시가 있을 때, 이 x 는 학습 세트에 있는 것인데, 교차 검증 세트에 있을 수도 있고, 테스트 세트에 있을 수도 있다. 주어진 x 예시로 f_1, f_2 등을 연산하려고 할 때, 여기서 l_1 는 x_1 과 같다. 이것은 feature 벡터를 생성한다. 여기서 f 를 feature 벡터라고 하자. 이 f_1, f_2 등을 가지고 feature 벡터를 안에 넣는다. 그래서 이렇게 f_m 까지 써준다. 관례상 여기에 feature f_0 를 넣을 수 있다. 이는 항상 1과 같다. 우리가 전에 봤던 것과 x_0 , interceptor와 비슷한 역할을 한다. 예를 들어, 학습 예시 $(x(i), y(i))$ 가 있을 때, 이 학습 예시를 연산할 features는 다음과 같다. $x(i)$ 는 $f_1(i) = sim(x(i), l(1))$ similarity는 다 쓰지 않고, 약자로 sim이라고 쓰겠다. $f_2(i) = sim(x(i), l(2))$ 그리고 $f(m) = sim(x(i), l(m))$ 과 같다. 그리고 여기 중간에, 여기 목록 안에, i 번째 요소가 있는데, $f_i(i) = sim(x(i), l(i))$ 가 된다. 여기서 $l(i)$ 는 $x(i)$ 와 같다. 따라서 $f_i(i)$ 는 $x(i)$ 와 $x(i)$ 사이의 similarity 를 구하는 것이 된다. 만약 Gaussian kernel을 활용하면, $f_i(i) = exp(-0 / 2\sigma^2)$ 이다. 따라서, 1과 같다. 따라서 이 학습 예시의 feature중 하나는 1이 된다.

04:33

이것은 여기 위의 내용과도 비슷하다. 여기 m feature를 모두 가지고 feature 벡터로 통합 정리한다. 이 예시를 다 표현하는 대신, $x(i) = R(n+1)$ 차원 벡터가 된다. 어떻게 정의하느냐에

따라 $R(n)$ 또는 $R(n+1)$ 이 된다. 이제 학습 예시를 표현하는 것 대신 이 feature 벡터 f 를 활용해보자. $F(i)$ 는 여기 있는 것을 벡터 안에 집어 넣을 것이다. $F(i) = [f_1(i) \dots f_m(i)]$ 가 되고, 그리고 원한다면, $f_0(i)$ 를 추가해도 좋다. 여기서 $f_0(i) = 1$ 이다. 따라서 여기 벡터는 내 학습 예시를 표현할 새로운 feature 벡터를 제공한다. 이와 같은 kernel 과 similarity 함수가 주어졌을 때, 서포트 벡터 머신을 어떻게 활용하는지 알아보자.

05:34

매개 변수 θ 의 학습 세트가 이미 주어졌다면, x 값이 있고 예측을 하려고 할 때, feature f 값을 연산하면 되는데, $f \in R(m+1)$ 이다. m 학습 예시가 있기 때문에 여기 m 이 있는데, landmark m 이 된다. 이제 $\theta^T f \geq 0$ 일 때, 1을 예측해보자. $\theta^T f = \theta_0 + \theta_1 f_1 + \dots + \theta_m f_m$ 이 된다. 그리고 $\theta \in R(m+1)$ 이 된다. 여기서 m 은 landmark의 수는 학습 세트 사이즈와 같음을 나타낸다. 따라서 m 은 학습 세트 사이즈를 나타내고, $\theta \in R(m+1)$ 이 되는 것이다. 따라서 이와 같이 매개 변수 θ 의 셋팅이 주어졌을 때, 예측을 할 수 있게 된다. 그렇다면 매개변수 θ 는 어떻게 구하는가? 바로 SVM 학습 알고리즘을 활용하는 것인데, 특히 이 최소화 문제를 해결하는 것이다.

$\min_{\theta} C$ 곱하기 이 비용 함수 최소화를 전에도 다뤄본 적 있다.

06:52

이제 $\theta^T x(i)$, 처음 feature 였던 $x(i)$ 를 활용해 예측하는 것 대신에, 처음에 활용한 feature $x(i)$ 대신에 이를 대체하여, $\theta^T x(i)$ 로 i 의 학습 예시를 예측하는데 활용하고자 한다. 이를 살펴보면, 여기 이 부분이고 이 최소화 문제를 해결하기 위해서 서포트 벡터 머신에 활용할 매개 변수를 구한다. 하나 더 설명하자면, 이 최적화 문제에서 n 은 m features 와 같다는 것이다. 여기 이 부분인데. 우리가 가진 feature의 개수다. Feature의 유효한 숫자는 f 차원이다. 따라서 $n = m$ 이 성립한다. 원한다면, 이것을 합계라고 생각하면 되는데, 이는 사실 $j=1$ 부터 m 까지의 합을 의미한다. 그리고 이것을 $n=m$ 이라고 생각할 수 있는데, f 가 만약 새로운 feature가 아니라면, $m+1$ feature 가 있게 되고, 여기서 $+1$ 은 interceptor에서 오는 것이다. 그리고 여기서는 $j=1$ 부터 m 까지 합계를 구하는데, regularization에서 다른 것과 비슷하게 매개변수 θ_0 를 regularize하지 않았기 때문이다. 그렇기 때문에 이 합계는 $j=0$ 부터 m 까지의 합이 아니라 $j = 1$ 부터 m 까지의 합이 되는 것이다.

08:22

이것이 바로 서포트 벡터 머신 학습 알고리즘이다. 내가 따로 수학적으로 설명하지 않은 부분이 있는데, 서포트 벡터 머신이 구현되는 방법이다. 여기 마지막 항이 사실 좀 다른 게 구현되었다. 서포트 벡터 머신을 활용하기 위해서 이러한 세부 내용까지 알아야 할 필요는 없다. 그래서 여기 써있는 방정식으로 여러분에게 필요한 내용은 다 이해하게 된 것이다. 하지만 서포트 벡터 머신이 구현되는 방법은, $\sum_j \theta_j^2$ 여기 이 항은 다르게 표현할 수 있는데, $\theta^T \theta$ 이다. 여기서 매개변수 θ_0 는 무시했다. 따라서 $\theta = [\theta_1; \dots; \theta_m]$ 이다. 그리고 여기 $\sum_j \theta_j^2$ 는 $\theta^T \theta$ 로 쓸 수 있다.

서포트 벡터 머신 구현에서 가장 많이 쓰이는 것은 $\theta^T \theta$ 를 대신하여 $\theta^T M \theta$ 인데 M 은 행렬이고, 어떤 kernel를 사용하느냐에 따라 어떤 행렬이 들어갈지 정해진다. 따라서 조금은 다른 distance

metric를 제공한다. 여기서 최소화 대신에 조금 다른 방법을 활용하는데, $\|\theta\|^2$ 는 여기서 약간 비슷하게 최소화 한다는 것이다. 이는 kernel에 따라서 매개 변수 벡터 θ 의 재설계 버전인 셈이다. 하지만 이는 수학적인 세부사항으로 서포트 벡터 머신 소프트웨어가 더 효율적으로 돌아갈 수 있게 해준다.

09:57

서포트 벡터 머신이 이와 같은 변경을 하는 이유는 훨씬 큰 학습 세트 크기를 변경할 수 있게 해주기 때문이다. 예를 들어, 10,000개의 학습 예시가 있는 학습 세트가 있다고 하자. 여기서 landmark를 정의하는 방법은 10,000개의 landmark를 얻게 된다. 따라서 θ 는 10,000 차원이 된다. 이정도 까지는 잘 돌아갈 수도 있다. 하지만 m 이 정말 커지면 이러한 모든 매개 변수 값을 구하는 것은, 예를 들어 $m=50,000$ 또는 $100,000$ 이 되면 이 모든 매개 변수를 구하는 것은 서포트 벡터 머신 최적화 소프트웨어에서 여기 그린 최소화 문제를 해결하려면 시간이 오래 걸리게 된다. 이는 세부적인 수학적 내용이므로 꼭 알고 있을 필요는 없다. 사실 이는 최적화를 위해 마지막 항을 조금 변경하는데, 이는 $\|\theta\|^2$ 를 최소화하는 것과는 약간 다르다. 여러분이 원한다면, 이 내용을 이행적 내용이라고 생각하면 되는데, 이는 objective를 약간 변경한다. 하지만 이는 주로 연산의 효율성을 위한 것이므로 크게 걱정하지 않아도 된다.

11:07

왜 kernel를 로지스틱 회귀와 같은 다른 알고리즘에 적용하지 않는지가 궁금할 수 있는데, 원한다면, 로지스틱 회귀에서 kernel을 적용하여, landmark를 활용하여 feature의 소스를 정의할 수 있다. 하지만 서포트 벡터 머신에 적용되는 연산 트릭은 로지스틱 회귀와 같은 다른 알고리즘에서는 잘 일반화 되지 않는다. 그렇기 때문에 로지스틱 회귀에서 kernel을 활용하는 것은 매우 느릴 것이다. 반면에 이와 같은 연산 트릭이 포함되면, 이를 어떻게 변경하는지, 서포트 벡터 머신 소프트웨어가 어떻게 구현될지, 서포트 벡터 머신과 kernel은 특히 함께 잘 구현된다. 반면에, 로지스틱 회귀와 kernel은 할 수는 있지만 매우 느리게 돌아갈 것이다. 따라서 사람들이 알아낸 서포트 벡터 머신이 kernel과 함께 구현되는 특정 케이스에 사용될 고차원 최적화 기술의 이득을 얻을 수 없다.

12:00

하지만 여기 이 부분은 여러분이 실제로 비용 함수를 최소화하는 소프트웨어를 구현하는 할 때에만 적용된다. 이 내용은 다음 시간에 더 살펴보기로 하자. 하지만 이 비용 함수를 최소화하는 소프트웨어를 어떻게 쓰는지에 대해서는 알 필요가 없는데, 이러한 작업을 소프트웨어를 쉽게 구할 수 있기 때문이다. 그리고 행렬을 도치시키거나 제곱근을 연산하기 위해 코드를 직접 쓰는 것을 추천하지는 않는데, 비용 함수 최소화하는 소프트웨어도 쓰는 것을 추천하지 않는다. 대신에 사람들이 개발해놓은 소프트웨어 패키지를 활용하는 것을 추천하고 그러한 소프트웨어 패키지는 이러한 수 최적화 트릭을 이미 포함하고 있다. 그러니 걱정할 필요는 없다. 하지만 이를 알아두면 좋은 점이 하나 있는데, 서포트 벡터 머신을 적용할 때, 서포트 벡터 머신의 매개 변수를 어떻게 선택하는가이다. 이제 마지막으로 서포트 벡터 머신을 활용할 때, bias, variance, trade-off에 대해

설명하도록 하겠다.

12:58

Svm을 활용할 때, 매개 변수 C를 선택해야 하는데, 이는 최적화 objective에 있고, 여기서 C는 $1/\lambda$ 와 같은 역할을 했었다. 여기서 λ 는 로지스틱 회귀에서 정규화 매개변수였다. Large C가 있으면, 이는 로지스틱 회귀에서 λ 값이 적은 것을 의미하는데, 이는 정규화를 많이 활용하지 않는다는 것이다. 이때, lower bias 와 higher variance 상태인 가설을 얻는다. 반면에, Small C에서는 로지스틱 회귀에서 λ 값이 큰 것을 의미하는데, higher bias 와 lower variance 상태인 가설을 얻게 된다. 따라서 large C의 가설은 higher variance이므로 과적합하기 쉽다. 반면에 small C 가설은 higher bias이므로 시부적합하기 쉽다. 따라서 이 매개변수 C는 우리가 선택해야 할 매개 변수 중 하나다.

14:00

또 다른 하나는 매개변수 σ^2 이고, 이는 Gaussian kernel에서 등장한다. Gaussian kernel σ^2 가 크면 Similarity 함수에서, 다음과 같다. 여기 예시를 들면, feature x_1 이 하나 있는데, 이 지점에 landmark를 선택하면, σ^2 가 클 때, Gaussian kernel은 상대적으로 천천히 내려간다. 이게 feature $f(i)$ 가 된다. 따라서 이 매끈한 함수는 그 값이 부드럽게 변하고, 이는 higher bias 와 lower variance 상태의 가설을 제공할 것인데, 이는 천천히 감소하는 Gaussian kernel에서는 입력 x 값을 바꾸면 가설도 천천히, 매끄럽게 변하기 때문이다. 반면에, σ^2 이 작으면, 여기 landmark와 feature x_1 이 있고, Gaussian kernel은, 즉 similarity 함수는, 더 급격하게 변화할 것이다. 두 경우 모두 1을 선택했고, 만약 σ^2 이 작으면, feature가 덜 매끄럽게 변화할 것이다. 따라서 경사가 높고, higher derivatives 할 경우, 이를 활용하여, lower bias, higher variance 상태가 되는 것이다. 이번 주의 연습문제를 통해 이 내용에 대해 숙지하여 이를 직접 활용할 수 있도록 하자.

Q) SVM을 학습하는 도중, 학습 데이터를 과적합하는 것을 발견했다. 다음에 취할 조치로 적절한 것은? (복수 선택 가능)

1. C 증가시키기
2. C 감소시키기
3. σ^2 증가시키기
4. σ^2 감소시키기

정답 2,3번

지금까지 서포트 벡터 머신에서 kernel 알고리즘에 대해 알아봤다. 여기서 알아본 bias와 variance를 통해 이 알고리즘이 어떻게 구현될지 예상할 수 있게 되었기를 바란다.

No.	Title
Week 7-6	Using An SVM

00:00

지금까지는 SVM에 대해 꽤 추상적인 수준에서 이야기해봤다. 이번 시간에는 SVM을 활용하기 위해서 실제로 필요한 것에 대해 이야기해보겠다. 서포트 벡터 머신 알고리즘은 특정한 최적화 문제를 제기한다. 하지만 지난 시간에 잠시 언급했던 것처럼, 매개 변수 θ 를 구하기 위해 직접 소프트웨어를 쓰는 것을 추천하지는 않는다. 오늘날, 몇 안 되는 사람들만, 아마 어느 누구도 코드를 직접 써서 행렬을 전치시키거나 제곱근의 수를 구하지 않을 것이다. 이런 것은 라이브러리 함수를 활용한다. 같은 방식으로 SVM 최적화 문제를 해결하기 위한 소프트웨어는 매우 복잡하다, 그리고 연구원들도 수 최적화 연구를 수년 동안 연구하고 있다. 따라서 이를 해결하기 위해서는 좋은 소프트웨어 라이브러리와 소프트웨어 패키지가 필요하다. 따라서 이를 직접 구현하려고 하는 것 보다는 최적화된 소프트웨어 라이브러리 중 하나를 선택하는 것을 강력하게 추천한다.

00:58

실제로도 좋은 소프트웨어 라이브러리가 많이 있다. 가장 많이 활용하는 tool은 liblinear와 libsvm인데, 이것 말고도 좋은 소프트웨어 라이브러리가 많이 있다. 학습 알고리즘을 코드화하는데 활용하기 위해 다양한 주요 프로그래밍 언어를 접하게 될 것이다. 직접 SVM 최적화 소프트웨어를 쓰는 것은 안되지만, 여러분이 해야 할게 몇 가지 있다. 먼저 매개 변수 C 를 선택해야 한다. 지난 시간에 bias/variance 특성에 대해서 잠깐 살펴보았다. 두 번째로, 활용할 kernel 또는 similarity(유사도) 함수를 선택해야 한다. 따라서 kernel을 사용할지 안 할지를 선택하는 것이 된다. Kernel을 활용하지 않는 아이디어를 linear kernel이라고 한다. 만약 누가 나는 linear kernel인 SVM을 활용한다고 하면, 이는 kernel을 활용하지 않고, SVM을 활용하는 것이고, 이는 $\theta^T X$ 를 활용하는 SVM이다. 이는 $\theta_0 + \theta_1X_1 + \dots + \theta_nX_n \geq 0$ 일 경우, $y=1$ 일 예측한다. linear kernel라는 용어는 이 SVM 버전은 표준 linear classifier를 제공한다는 말이다.

02:13

어떤 문제에 있어서 합리적인 선택이 될 것이고, 소프트웨어 라이브러리가 많이 있는데, 예를 들어 liblinear와 같은, kernel 없이 SVM을 학습하는 소프트웨어 라이브러리가 있는데 이는 linear kernel이라고 한다. 그렇다면 왜 이렇게 하려고 하는가? Feature의 수가 많을 경우, 즉, n 이 클 경우, 학습 예시의 수인 m 은 작다. feature의 수가 많을 때, 여기서 $x \in \mathbb{R}^{n+1}$ 이다. Feature 수가 이미 많고, 학습 세트는 작다면, 데이터가 충분하지 않으므로 복잡한 비선형 함수가 아니라 선형 결정 경계를 대입할 것이다. 그러면 과적합 될 위험이 있다. 고차원 feature 공간에 있는 매우 복잡한 함수를 대입하려고 해도 학습 세트가 작다. 따라서 이런 경우에는 kernel 활용하지

않는 것, 즉 linear kernel를 활용하는 것이 합리적인 셋팅으로 보인다.

03:14

kernel에서 필요한 두 번째 선택은 Gaussian kernel 인데 이는 전에 언급한 내용이다. 이를 선택했다면, 다음 선택은 σ^2 인데, 지난 시간에 bias, variance, tradeoffs에 대해 설명했다. 또한, σ^2 가 클 때, higher bias, lower variance classifier가 되고, σ^2 작으면, higher variance, lower bias classifier가 된다. 그렇다면 이 Gaussian kernel은 언제 선택하는가? 만약 $x \in R(n)$ 이고, n이 작고, m은 클 때, 예시로 이와 같은 2차원 학습 세트를 전에 그려본 적 있다. 여기서 n=2지만 이는 꽤 큰 학습 세트다. 학습 예시의 꽤 많은 수를 그렸기 때문에, 여기에 kernel을 활용해 더 복잡한 비선형 결정 경계를 대입하고자 한다면, Gaussian kernel 이 좋은 방법이다. 이와 관해서는 수업 후반부에 linear kernel, Gaussian kernel을 언제 선택하는지에 대해 좀 더 설명하겠다.

04:27

하지만 Gaussian kernel 를 활용하기로 결정했다면, 다음과 같이해야 한다. 어떤 서포트 벡터 머신 소프트웨어 패키지를 이용하느냐에 따라, kernel 함수 또는 similarity 함수 중에 고르라고 할 것이다. 따라서 SVM 구현할 때, 옥타브 또는 맷랩을 사용한다면, kernel의 특정 feature를 연산하기 위한 함수를 제공하라고 요구할 것이다. 따라서 여기서는 f_i 를 연산하여 i 값을 구하는데, 여기 있는 f 값은 실수이고, $f(i)$ 라고 쓰는 것이 더 적절하겠다. Kernel 함수를 쓰기 위해서 입력값이 필요는데, 학습 예시나 테스트 예시 중 벡터 x 와 landmark 값으로 가져와서, 여기서는 x_1, x_2 만 가져왔는데, landmark도 학습 예시이기 때문이다. 이제 입력값을 가져와서 x_1, x_2 소프트웨어를 쓰고, 이 사이에 있는 similarity 함수를 연산하여 실수 값을 얻는다.

04:27

서포트 벡터 머신 패키지는 이 x_1, x_2 를 입력값으로 kernel 함수에 제공하여, 실수 값을 얻는 역할을 한다. 그리고 거기서부터 자동으로 모든 feature를 생성하는데, 자동으로 X 는 f_1, f_2, \dots, f_m 값을 가지고, 함수를 활용해 모든 feature를 생성하고 거기서부터 서포트 벡터 머신을 학습시킨다. 하지만 때로는 이 함수를 여러분이 제공해야 할 때도 있다. Gaussian kernel을 활용한다고 해도 어떤 SVM 구현은 Gaussian kernel과 다른 kernel 또한 포함해야 할 때도 있는데, Gaussian kernel이 가장 흔히 활용되는 kernel이기 때문이다. Gaussian 과 linear kernel 은 지금까지 가장 널리 사용되는 kernel이다.

06:18

이를 구현 할 때, 한가지 참고해야 할 점은 매우 다양한 범위의 feature가 있다면, Gaussian kernel를 활용하기 전에 feature scaling을 하는 것이 중요하다. 왜냐하면 $\|x - l\|^2$ 을 연산한다고 할 때, 여기 이 식은 여기 이 문자와 같다. 이 $\|x - l\|^2$ 은 $x-l$ 벡터 v 값을 구하는 것이다. 그렇다면 이제 $\|v\|$, 즉 x 사이의 차를 구하면 된다. $\|v\|^2 = v_1^2 + v_2^2 + \dots + v_n^2$ 인데 $x \in R(n)$ 또는 $x \in R(n+1)$ 이므로 n항까지 있다. 하지만 여기서 x_0 는 무시하고, $x \in R(n)$ 이라고 하면, 제곱의 왼쪽 부분이 이를 맞게 해준다. 따라서 $\|x - l\|^2 = \|v\|^2 = v_1^2 + v_2^2 + \dots + v_n^2$ 이

성립한다. 또 달리 표현해보면, $(x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2$ 이 된다.

07:29

이제 여러분의 feature는 매우 다양한 값을 가지게 되었다. 이제 집값 예측 예시를 대입해보자. 이 데이터가 주택과 관련된 데이터라고 하자. X 는 수천 평방피트 범위에 있고, 이는 첫 번째 feature, x_1 이다. 두 번째 feature x_2 는 방의 수다. 따라서 1부터 5까지 방의 수라면, $x_1 - l_1$ 사이 간격이 매우 커질 것이다. 여기서는 천의 제곱피트를 다루고 있고, 이에 비해 $x_2 - l_2$ 는 훨씬 작다. 이 경우에 이 항에서 두 항 사이의 거리는 주택의 크기에 의해 결정되고, 방의 수는 거의 무시될 것이다.

08:16

따라서 이러한 상황을 피하고 머신이 잘 돌아가게 하려면, feature scaling 작업을 거쳐야 한다. 그렇게 되면 SVM은 다양한 feature 모두를 비슷하게 주목할 것이고, 방금 예시의 주택 사이즈처럼 더 큰 움직임이 없어야 한다. 서포트 벡터 머신을 활용할 때, 가장 많이 활용하는 두 kernel은 linear kernel인데, 이는 kernel을 활용하지 않는다는 뜻이고, 또 다른 하나는 아까 살펴본 Gaussian kernel이다. 한가지 당부하자면, 모든 여러분이 생각해내는 모든 similarity 함수가 유효한 kernel은 아니라는 것이다. Gaussian kernel과 linear kernel 그리고 기타 kernel은 모두 기술적인 조건을 만족한다. 이는 Mercer's Theorem 이라고 하는데, 이를 해야 하는 이유는 서포트 벡터 머신의 알고리즘 또는 SVM의 구현이 매개변수 θ 값을 효과적으로 구하기 위한 더 영리한 numerical 최적화 트릭을 많이 가지고 있기 때문이다.

09:14

처음 디자인 구상 단계에서 이러한 결정은 우리의 관심을 kernel에게만 제한시켜서 Mercer's Theorem 이라는 기술적 조건을 만족시키게 된다. 이것의 역할은 모든 SVM 패키지, SVM 소프트웨어 패키지가 큰 class 최적화를 활용할 수 있게 해주고, 매개 변수 θ 값을 매우 빠르게 얻을 수 있다. 대부분의 사람들이 Gaussian kernel 또는 linear kernel 을 활용하는데, Mercer's theorem를 만족시키는 또 다른 kernel이 몇 개 더 있다. 다른 사람들이 이를 활용하는 것을 보게 될 수도 있는데, 개인적으로 두 개 이외의 다른 kernel은 거의 사용하지 않는다. 아마 전혀 사용하지 않을 수도 있다. 단지 다른 사람들이 활용할 수도 있기에 잠시 살펴보면, 하나는 polynomial kernel이다. 여기서 $k(x, l)$ 은 다양한 옵션이 있지만, $(x^T l)^2$ 라고 정의해보자. 이제 이를 측정해보자. 만약 x 와 l 이 서로 가깝다면, 내적은 커질 것이다. 따라서 이는 훈련 않은 kernel 이다. 이는 자주 사용되지 않지만, 다른 이가 사용하는 것을 보게 될 수는 있다. 이게 polynomial kernel의 한 버전이다.

10:32

또 다른 버전은 $(x^T l)^3$ 이고, 이러한 것들이 polynomial kernel 예시다. $(x^T l + 1)^3$, $(x^T l + 5)^4$ 등이 있다. polynomial kernel은 매개 변수가 2개 있다. 하나는 여기에 어떤 수를 더할 수 있을까? 바로

0이다. 따라서 여기에는 실제로 $+ 0$ 을 해주고 또한 이 몇 차항 인지가 있다. 여기 이 숫자가 이를 나타낸다. 그리고 polynomial kernel의 일반적인 형태는 $(x^T I + \text{정수})^{\text{차수}}$ 이고, polynomial kernel에는 이렇게 이 두 매개 변수가 있다. 따라서 polynomial kernel은 거의 항상 또는 보통 그리 잘 구현되지는 않는다. Gaussian kernel도 이를 많이 활용하지는 않지만 이를 보게 될 수는 있다. 이는 보통 데이터가 x 와 $|y|$ 이 negative가 아닐 때 활용된다. 따라서 내적이 negative가 아니라는 것을 보장한다. 또한, x 와 $|y|$ 서로로 비슷하다는 것을 말해주고, 그렇다면 둘 사이의 내적은 클 수도 있다. 또 다른 특성도 있지만 그리 많이 활용되지 않는다.

11:48

여러분이 하려는 작업에 따라 다르겠지만, 소수만 알고 있는 kernel도 발견할 수 있다. 이중 하나는 string kernel인데, 입력 데이터가 텍스트 string 이거나 다른 형태의 string일 때 활용한다. chi-square kernel, histogram intersection kernel 등도 있다. 이와 같이 다른 두 물체의 similarity를 측정하는데 활용되는 난해한 kernel들이 더 있다. 예를 들어, 텍스트 분류 문제를 해결하려고 하면, x 입력값이 문자열이라면 string kernel을 활용하여 두 문자열 사이의 similarity를 찾을 수도 있다. 하지만 개인적으로 아마도 이러한 kernel을 활용할 기회는 거의 없거나 아예 없을 수도 있다. 아마 chi-square kernel을 딱 한 번 정도 사용한 적 있고, histogram kernel도 한두 번 정도 밖에 사용해 본 적이 없다. string kernel은 단 한번도 활용한 적이 없다. 하지만 다른 어플리케이션에서 이와 같은 것을 발견하게 될 경우 대비해 한 번 살펴봤다. 구글이나 빙 검색 엔진에서 kernel의 정의에 대해 검색하면 이러한 것들이 kernel로 분류되어 있다.

Q) 여러 종류의 kernel 중 하나와 이 매개 변수 C , σ^2 를 선택하려고 한다. 가장 올바른 선택은?

1. 학습 데이터에서 가장 잘 구현되는 것 선택
2. 교차 검증 데이터에서 가장 잘 구현되는 것 선택
3. 테스트 데이터에서 가장 잘 구현되는 것 선택
4. 가장 큰 SVM margin을 주는 것 선택

정답 2번

12:51

이제 두 가지만 더 살펴보도록 하겠다. 하나는 다중 분류다. 이렇게 4개의 class로 보통은 k 개의 class를 가지고 다양한 class 사이에 적절한 결정 경계를 출력하려고 한다. 대부분의 SVM, SVM 패키지에 이미 다중 분류 기능이 내장되어 있다. 따라서 이와 같은 패키지를 활용하여, 그 기능을 활용하면 될 것이다. 그렇지 않을 경우, 한 가지 방법은 로지스틱 회귀 개발에서 배웠던, one-vs-all 방법을 활용하는 것이다. 방법은 k class가 있을 때, k SVM을 학습하면 되는데, one은 클래스 각각을 다른 클래스로부터 분리시키는 것이다. 그러면 k 매개 변수가 벡터가 생성되어, $\theta(1)$ 을 제공하여 class $y=1$ 으로 다른 class와 분리할 것이다. 두 번째 매개변수 벡터 $\theta(2)$ 는 $y=2$ 이므로

positive class로 나머지는 모두 negative class가 된다. 이와 같이 매개 변수 벡터 $\theta(k)$ 에 가서 이 매개 변수 벡터는 마지막 class K를 나머지와 구분하게 된다.

13:59

그리고 마지막으로, 이것은 로지스틱 회귀에서 다뤘던 one-vs-all 방법과 똑같다고 보면 되는데, class i를 $\text{largest}(\theta(i)^T x)$ 을 통해 예측하는 것이다. 다중 분류는 이 정도에서 마무리하자. 더 흔한 경우는 어떠한 소프트웨어 패키지를 사용한다고 해도 다중 분류 기능이 내장되어 있을 확률이 꽤 높으니 이러한 결과에 대해서 걱정하지 않아도 된다.

14:24

우리는 서포트 벡터 머신을 로지스틱 회귀에서 시작해서 개발했고, 그 다음에 비용 함수를 조금 변경해봤다. 이번 시간에 마지막으로 살펴볼 내용은 간단하게 살펴보면, 이와 같은 두 알고리즘 중에 하나를 사용하게 되는 경우이다. 여기서 n은 feature의 수이고, m은 학습 예시의 수이다. 따라서 우리는 언제 이 두 알고리즘을 사용하는 것일까? 먼저 n이 학습 세트 사이트보다 상대적으로 크면, 예를 들어 n은 feature의 수이고, $n \geq m$ 일 때, 예를 들어 텍스트 분류 문제가 있을 경우, feature 벡터의 차원은 $n=10,000$ 이고 학습 세트 사이즈는 $m = 10 - 1000$ 이라고 하자. 스팸 분류 문제를 생각해보면, 스팸 메일이 있을 때, 10,000 feature가 있는데, 이는 10,000 개의 단어를 말하고, 10부터 1,000개 사이의 예시가 있다고 하자. 따라서 n이 m 보다 상대적으로 크면, 보통 로지스틱 회귀 또는 kernel 없이 SVM을 즉, linear kernel를 활용한다.

15:31

작은 학습 세트에서 많은 feature가 있으면, 선형 함수는 잘 돌아가겠지만, 매우 복잡한 비선형 함수에 대입할 만한 자료가 충분하지 않기 때문이다. 이제 n은 작고, m은 중간이라면, $n = 1 - 1000$ 이면 1은 매우 작은 수가 된다. 하지만 1000개의 feature에서 학습 예시의 수가 10 - 10,000, 아니 50,000까지라면 $m = 10,000$ 은 꽤 큰 수지만 백만에 비하면 큰 수는 아니다. 따라서, m이 중간 사이즈 수라면 SVM에서 선형 kernel은 잘 구현될 것이다. 우리는 이에 대해 예전에 한 분명한 예시와 함께 다뤄봤는데, 이렇게 2차원 학습 세트가 있다. $N=2$ 일 경우, 이렇게 꽤 많은 수의 학습 예시를 그려 놓는다. 따라서 So Gaussian kernel은 positive 와 negative classes를 잘 분류해낼 것이다.

16:30

세 번째 셋팅은 n이 작고, m이 큰 경우다. $N = 1 - 1000$ 이거나 더 클 수도 있다. 하지만 m이 50,000 또는 백만보다 더 클 수도 있다. 50,000, 100,000, 백만, 1조. 엄청 큰 학습 세트 사이즈를 가지게 된다. 이럴 경우, Gaussian Kernel의 SVM이 뭔가 천천히 구현될 것이다. 요즘 SVM package에서 Gaussian Kernel을 활용할 경우, 약간 더딜 수도 있다. 그래서 50,000 정도는 괜찮을 건데, 백만 개의 학습 예시가 있거나 십만도 많은 축에 속한다. 요즘 SVM 패키지는 매우 훌륭하지만, Gaussian Kernel 를 활용하여 이렇게 큰 학습 세트 사이즈를 구현하는 데는 조금

버벅거릴 수 있다. 따라서 이와 같은 경우, 그냥 수동으로 feature를 더 생성해서 로지스틱 회귀 또는 kernel 없는 SVM를 활용한다.

17:33

여기 슬라이드를 보면, 로지스틱 회귀 또는 kernel 없는 SVM가 써있는 것을 볼 수 있는데, 둘 다 같은 내용이다. 이는 로지스틱 회귀와 kernel 없는 SVM은 거의 비슷한 알고리즘이고, 둘 다 꽤 비슷한 성능을 가지고 있고, 구현할 세부내용에 따라서 둘 중 어떤 것을 사용할 지가 정해진다. 보통 이 중 하나의 알고리즘이 적용되는 곳에, 즉 로지스틱 회귀가 적용되고, kernel 없는 SVM가 적용된다면 다른 하나의 알고리즘도 잘 작동한다. 하지만 SVM에서 다른 kernel을 활용해 복잡한 비선형 함수를 학습할 때, 예시의 수가 10,000개 또는 50,000개까지 있을 수도 있으므로 굉장히 큰 숫자다. 이는 매우 흔히 볼 수 있는데, 이러한 사례에서 서포트 벡터 머신에서 Gaussian Kernel을 활용할 때 빛을 발할 것이다. 이보다 더 구현하기 어려운 것은 로지스틱 회귀가 필요할 것이다.

18:38

이번에는 신경망은 어디에 적용할 수 있을까? 이러한 모든 문제, 규칙에서 잘 설계된 신경망은 잘 구현된다. 하나 단점이자 신경망을 활용하지 않는 경우가 있는데, 이러한 문제에서 신경망은 느리게 학습할 수 있다는 것이다. 하지만 여러분이 좋은 SVM 구현 패키지를 가지고 있다면, 신경망보다는 좀 더 빠르게 작동될 것이다. 이 전에는 보여주지 않았지만, SVM이 가지고 있는 최적화 문제는 convex 최적화 문제로, 좋은 SVM 최적화 소프트웨어 패키지라면 전역 최솟값 또는 이와 근접한 것을 항상 찾아낼 것이다. 따라서 SVM에서 local optima에 대해서는 걱정하지 않아도 된다. 실제로 local optima는 신경망에서 그리 큰 문제는 아니지만 결국 해결 할 수 있으므로, SVM을 활용한다면 크게 신경 쓰지 않아도 된다. 또한 문제가 어떤 것이냐에 따라서 신경망은 매우 느려질 수 있는데, 이와 같은 SVM에서 특히 그럴 수도 있다.

19:40

여기서 설명한 내용이 모호하거나 이해하는데 어려움이 있다면, 예를 들어 이 알고리즘을 써야 할지 저 알고리즘을 써야 할지에 대한 확신이 서지 않는다고 해서 걱정할 필요는 없다. 기계 학습 문제에서 나 또한 어떠한 알고리즘을 사용하는 것이 최선일지 확신이 서지 않는다. 하지만 이전 시간에 배웠듯이 알고리즘 자체보다는 데이터의 양에 달려있다. 그리고 얼마나 여러분이 능숙한지, 오류 분석과 학습 알고리즘 디버깅, 새로운 feature 설계, 그리고 어떤 feature를 통해 학습 알고리즘을 얻을 수 있는지 등이 더 중요하다. 그러한 것들이 어떠한 로지스틱 함수 또는 SVM를 활용하는지 보다 더 중요한 경우도 많다는 것이다.

20:23

그럼에도 불구하고, SVM은 여전히 가장 강력한 학습 알고리즘으로 평가 받고 있고, 복잡한 비선형 함수를 학습할 수 있는 매우 효율적인 방법을 가지고 있다. 따라서 로지스틱 회귀, 신경망, SVM 모두를 통틀어, 이러한 빠른 학습 알고리즘을 활용하는 것은 최신 기계 학습 시스템을 잘

구현하여 다양한 어플리케이션에 활용할 준비가 되어있는 것이다. 이는 또한 여러분에게 매우 강력한 무기가 되어줄 것이다. 이는 실리콘밸리, 또는 이 산업군과 학계에서 고성능 기계 학습 시스템을 개발하기 위해 활용되고 있다.

Week 8

No.	Title
Week 8-1	Unsupervised Learning: introduction

00'00"

본 강의에서는 클러스터링을 다룰 것이다. 이것은 신나는 일인데 label된 데이터를 살펴보는 대신 살펴볼 우리의 첫 비지도학습(Unsupervised Learning) 알고리즘이기 때문이다. 자율 학습은 무엇인가? 강의 시작쯤에 이에 대해 간단하게 언급한 적이 있다. 여기서는 지도 학습과 비교해 보는 것이 유용할 것이다. 여기 전형적인 지도 학습 문제가 있다. Label된 트레이닝 셋이 주어졌고 포지티브 라벨 예시(positive label example)와 네거티브 라벨 예시(negative label example)를 구별짓는 결정 경계(decision boundary)를 찾는 것이 목표이다. 이 때 지도 학습 문제는 주어진 label 집합에 가설을 맞추는 것이다. 반대로 자율 학습 문제는 주어진 데이터와 관련 있는 label이 존재하지 않는다.

00'48"

우리에게 주어진 데이터는 다음과 같다. 여기 label되지 않은 일련의 점이 있다. 그래서 우리의 트레이닝 셋은 label y 가 없이 단순히 x_1, x_2 에서 x_m 로 표현된다. 여기 그래프의 점들에 label이 없는 이유다. 따라서 자율 학습에서는 label되지 않은 트레이닝 셋을 가지고 알고리즘을 실행시키고 이 알고리즘이 우리를 위해서 자료의 구조를 찾도록 할 것이다. 여기 주어진 데이터 셋을 보면 우리가 알고리즘으로 하여금 찾도록 할 가지 유형의 구조는 다음과 같다. 데이터 셋 점이 두 개의 무리(클러스터)로 나뉘고 알고리즘이 내가 동그라미 한 클러스터를 찾는다. 이는 클러스터링 알고리즘이라 불린다. 이것이 바로 우리가 배울 첫 번째 자율 학습이다. 클러스터가 아니라 다른 유형의 구조나 데이터 패턴을 찾는 자율 학습 알고리즘에 대해서도 차후에 논의할 테지만 클러스터링에 대해 얘기한 후에 이를 살펴볼 것이다.

01'52"

그러면 클러스터링의 장점은 무엇인가? 강의 초반에 몇 가지 활용법을 얘기한 적이 있다. 한 가지는 고객의 데이터 베이스를 가지고 마케팅 차별화를 할 때이다. 고객을 각각 다른 부류로

그룹 지어 각각의 그룹에 대해 다른 판매전략을 취하거나 서로 다른 그룹들에 더 최적화할 수 있다. 즉, 소셜 네트워크 분석이다. 실제로 사람들의 소셜 네트워크를 그룹 지은 사람들이 있다. 페이스 북, 구글+에 올린 정보나 여러분이 가장 자주 이메일을 보내는 사람들이나 그 사람들이 자주 메일을 보내는 또 다른 사람들을 통해 친밀한 사람들을 그룹 지을 수 있다. 이것은 클러스터링 알고리즘의 또 다른 예로 소셜 네트워크 상에서 가장 친한 그룹의 친구를 찾도록 해준다. 내 친구가 실제로 한 일인데 그 친구는 클러스터링을 사용해서 컴퓨터 클러스터들을 조직하고 데이터 센터들을 더 잘 조직한다. 만약 여러분이 클러스터 안에서 어떤 컴퓨터가 데이터 센터와 더 긴밀하게 일하는지 안다면, 여러분은 자원과 네트워크의 구성을 재조직하여 데이터 센터의 커뮤니케이션을 디자인 할 수 있다. 마지막으로 또 다른 친구가 한 일인데 이 친구는 천문학 데이터를 통해서 은하의 생성을 이해하기 위해 클러스터링 알고리즘을 사용했다.

Q) 다음 중 어떤 진술이 참인가? 해당되는 것을 모두 고르시오.

- ① 자율 학습에서 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ 트레이닝 셋은 label되지 않은 $y^{(i)}$ 가 없다.
- ② 클러스터링은 자율 학습의 한 예이다.
- ③ 자율 학습에서는 label되지 않은 데이터 셋을 주고 데이터의 구조를 찾으라고 한다.
- ④ 클러스터링은 자율 학습의 유일한 알고리즘이다.

정답 2,3 번

이렇게 쓰일 수 있는 클러스터링 알고리즘이 우리가 처음 살펴 볼 자율 학습 알고리즘의 예이다. 다음 강의에서 특정한 클러스터링 알고리즘을 배워보자.

No.	Title
Week 8-2	K-Means Algorithm

00'00"

클러스터링 문제를 풀 때는 label되지 않은 데이터 셋이 주어질 것이다. 그리고 우리는 알고리즘이 자동적으로 데이터를 서로 상관관계를 갖는 부분 집합이나 클러스터로 그룹 지어 주기를 바랄 것이다. K Means 알고리즘은 클러스터링 알고리즘 가운데 지금까지 가장 인기가 많고 광범위하게 사용되는 알고리즘이다. 따라서 이번 강의에서는 K Means 알고리즘이 무엇이고 어떻게 작동하는지를 살펴보겠다.

00'26"

K Means 클러스터링 알고리즘은 다음 그림에 잘 나타나 있다. 예를 들어 다음과 같이 Label되지 않은 데이터 셋을 2 개의 클러스터로 나누고 싶다고 해보자. 만약 K Means 클러스터링 알고리즘을 실행한다면 다음과 같이 하면 된다. 첫 번째 단계는 클러스터 포인트인 점 두 개를 무작위로 뽑아 초기화하는 단계이다. 이 점들을 클러스터 중심점(Cluster centroids)라 한다. 여기

두 X 표시된 점이 클러스터 중심점이다. 여기서는 두 점을 선택했는데 데이터를 두 그룹으로 나누고 싶기 때문이다. K Means는 반복 알고리즘으로 여기서 두 가지 작업을 한다. 첫 번째는 클러스터 배정 단계이며 두 번째는 중심점 이동 단계이다. 이제 이것들이 무엇인지 설명하겠다. K means 알고리즘의 두 단계 중 첫 번째는 클러스터 배정 단계이다. 이것이 뜻하는 바는, 여기 초록 점들은 각각 빨간 클러스터 중심점과 파란 클러스터 중심점 중 가까운 가까운 클러스터에 배정되는 과정을 거친다.

01'37"

구체적으로 보면 여러분이 가진 데이터 셋을 빨간 점들의 중심과 파란 점들의 중심 중 어디에 가까운지에 따라, 지금 내가 한 것처럼 색깔을 표시한다. 이것이 바로 클러스터 배정 단계이다. K-means 반복문 내에서의 또 다른 단계는 중심점 이동 단계이다. 우리는 여기서 두 개의 클러스터 중심을, 즉 여기서처럼 빨간 X점과 파란 X점을 선택해서, 같은 색으로 표시된 점들의 평균점으로 옮길 것이다. 이제 우리는 빨간 점들을 보고 그 평균을 계산할 것이다. 모든 빨간 점들의 위치의 평균을 계산한다. 그리고 빨간 점들의 중심을 이동시킬 것이다. 같은 작업을 파란 점들의 중심에서도 반복한다. 모든 파란 점들을 보고 평균을 계산한 후 파란 점들의 중심을 그곳으로 옮긴다. 지금 한 번 해보자. 점들의 중심을 다음과 같이 이동시킬 것이다. 이제 새로운 평균점으로 옮겨졌다. 빨간 점은 다음과 같이 움직였고 파란 점은 다음과 같이 움직였다. 이제 우리는 이제 또 다른 클러스터 배정 단계로 돌아가자. 이제 우리는 다시 한 번 모든 lable 안된 예들이 빨간 점들의 중심에 가까운지 아니면 파란 점들의 중심에 가까운지를 본다. 이에 따라 빨간 색이나 파란 색으로 표시할 것이다. 각각의 점들을 두 개의 클러스터 중심 중 하나로 배정할 것이다. 지금 한 번 해보자.

02'50"

이제 몇몇 점들의 색깔이 바뀌었다. 이제 또 다른 중심점 이동 단계로 나가자. 모든 파란 점들의 평균을 계산하고 모든 빨간 점들의 평균을 계산해서 클러스터 중심을 다음과 같이 옮긴다. 다시 한 번 해보자. 클러스터 배정 단계를 한 번 더 하겠다. 각각의 점들을 어느 쪽에 가까운지 보고 빨간 색, 아니면 파란 색으로 표시한다. 그리고 중심을 이동시킨다. 그리고 끝났다. 여기서 K means 알고리즘을 추가적으로 계속 반복하면 클러스터 중심은 더 이상 바뀌지 않을 것이고 점들의 색깔 또한 더 이상 바뀌지 않을 것이다. 그리고 이것이, 바로 이 점이 K means가 수렴하는 점이다. 이 데이터에서 두 개의 클러스터를 잘 찾았다. K means 알고리즘을 좀 더 공식적으로 표시해보자.

03'42"

K means 알고리즘은 입력 값이 두 개 필요하다. 하나는 변수 K로 여러분이 이 데이터 상에서 찾고자 하는 클러스터의 숫자이다. 이후에 k를 선택하는 법을 알려줄 것이다. 지금은 우리가 찾고 싶어하는 클러스터 개수를 결정했다고 치자. 이제 알고리즘에 데이터 셋에 얼마나 많은 클러스터가 있다고 생각하는지를 입력할 것이다. K means 알고리즘은 또한 lable되지 않은 여러 개의 x로 구성된 트레이닝 셋을 입력값으로 가진다. 이것이 자율 학습이기 때문에 우리는 y를 더

이상 lable할 필요가 없다. K means의 자율 학습을 위해 나는 $x^{(i)} \in R^n$ 차원의 벡터를 사용할 것이다. 즉 트레이닝 예들은 N+1차원의 벡터가 아니라 N 차원의 벡터이다. 이제 K means 알고리즘이 하는 일을 살펴보자.

04'27"

이제 첫 번째 단계는 k 클러스터 중심점을 무작위로 초기화한다. 이것을 $\mu_1, \mu_2, \dots, \mu_k$ 라 부른다. 초기 다이어그램에서 클러스터 중심점은 빨간 X점과 파란 X점의 위치와 대응되었다. 그래서 우리는 클러스터 중심점을 두 개 가졌다. 빨간 X점은 아마도 μ_1 과, 파란 X점은 μ_2 였을 것이다. 그리고 이러한 식으로 일반적으로 두 개보다 더 많은 K개의 클러스터 중심점을 잡을 수 있다. k means 알고리즘의 내부 반복문에서 우리는 다음을 반복할 것이다. 먼저 각각의 트레이닝 예시에서, 나는 변수 C^i 를 $x^{(i)}$ 에 가장 가까운 클러스터 중심점의 index 1에서 K로 설정할 것이다. 이것이 나의 클러스터 배정 단계로, 우리는 각각의 예시를 가까운 클러스터 중심점에 따라 빨간 색 아니면 파란 색으로 표시했다. C^i 는 1에서 K에 속하는 숫자로, 빨간 X점에 가까운지, 아니면 파란 X점에 가까운지를 알려줄 것이다. 또 다른 방법은 $\|x^{(i)} - \mu_K\|$ 을 사용해서 각각의 클러스터 중심점과의 거리를 측정하는 것이다. 여기서 이것은 μ 이고 μ_K 에서 대문자 K는 모든 중심점 개수이다. 나는 소문자 k를 사용해서 다른 중심점을 가리킬 것이다. 하지만 C^i 는, 나는 k값을 최소화할 것이고 $x^{(i)}$ 와 클러스터 중심점의 거리를 최소화하는 K값을 구할 것이다. 그리고 이를 최소화하는 k값이 C^i 에 들어간다. C^i 가 무엇인지를 표현하는 다른 방식이 있다. 바로 $\|x^i - \mu_k\|$ 이다. 이는 트레이닝 예 x^i 와 클러스터 중심점 μ_k 간의 거리이다. 여기서 k는 소문자 k이다. 대문자 K는 클러스터 중심점의 총 개수를 나타내고 이 소문자 k는 1과 K 사이의 숫자를 나타낸다. 나는 소문자 k를 사용해서 다른 클러스터 중심점들을 나타낼 것이다. 다음은 소문자 k이다. 이것은 예시와 클러스터 중심점 간의 거리이다. 나는 K값을 찾아서, 이것을 최소화하는 k를 찾아서, 이 값을 C^i 로 둘 것이다.

06'15"

보통의 경우처럼 나는 x^i 와 클러스터 중심점의 거리를 썼는데, 보통 사람들은 이것을 제곱한다. 우리는 C^i 가 나의 트레이닝 예 x^i 와 클러스터 중심점 간 가장 작은 거리의 제곱으로 결정된다고 생각한다.

Q) K means 알고리즘을 실행하고 이 알고리즘이 수렴된 뒤에 여러분은 $c^{(1)} = 3, c^{(2)} = 3, c^{(3)} = 5 \dots$ 를 가진다. 다음 중 참인 진술은? 해당되는 것을 모두 고르시오.

정답 2번

하지만 최소화된 제곱 거리, 최소화된 거리는 C^i 값을 주겠지만 우리는 대개 여기에 제곱을 쓰는데, 일반적으로 사람들이 K means 알고리즘에서 이렇게 한다. 이것이 클러스터 배정 단계이다. K means의 반복문은 또한 중심점을 옮기는 단계를 포함한다. 따라서 클러스터 중심점을 소문자 $k=1 \sim K$ 각각에 대해서 μ_k = 클러스터에 배정된 점들의 평균으로 설정한다.

구체적인 예를 들어 클러스터 중심점 중 하나, 중심점 2가 x^1, x^2, x^6, x^{10} 을 트레이닝 예로 가진다고 해보자. 이것이 실제로 의미하는 것은 $c^1 = 2, c^2 = 2, c^6 = 2$, 그리고 마찬가지로 $c^{10} = 2$ 이라는 것이다. 이것을 클러스터 배정 단계에서 얻었다면, 이는 x^1, x^2, x^6, x^{10} 이 클러스터 중심점 2에 배정되었다는 것을 의미한다. 중심점 이동 단계에서는 이 네 개의 평균을 구할 것이다. 따라서, $x^1 + x^2 + x^6 + x^{10}$ 하고 이 값을 평균 낸다. 그러니까 이 클러스터 중심점에 배정된 네 개의 점들을 $\frac{1}{4}$ 로 나눈다. 이제 μ_2 는 n차원 벡터가 될 것이다. 예시 x^1, x^2, x^6, x^{10} 이 각각 n차원 벡터였기 때문이다. 나는 이것을 모두 더해서 4로 나눌 것이다. 왜냐면 이 클러스터 중심점에 배정된 점이 네 개이기 때문이다. 그리고 중심점 이동 단계에서는 중심점 μ_2 를 이동시킨 후 끝낼 것이다.

07'52"

이것은 μ_2 를 여기 적힌 네 개의 점의 평균으로 이동시키는 효과를 가진다. 내가 받은 질문 가운데 하나는, 예를 들어 μ_k = 클러스터에 배정된 점들의 평균으로 놓았을 때 만약 클러스터에 배정된 점이 하나도 없다면 어떻게 하냐는 것이다. 이 같은 경우에는 이 클러스터 중심점을 없애는 것이 보편적인 해결법이다. 그렇게 하면 k 클러스터 대신 K-1 클러스터를 갖게 된다. 만약 k 클러스터가 정말 필요한 경우라면, 아무 점도 없는 클러스터 중심점으로 할 수 있는 방법은 무작위로 클러스터 중심점을 재 초기화하는 것이다. 하지만 K means 알고리즘을 실행하는 동안 아무 점도 없는 클러스터 중심점을 가진 클러스터는 제거하는 것이 일반적이다. 이런 경우가 있기도 하지만 실제로는 자주 일어나는 상황은 아니다. 지금까지 K means 알고리즘을 살펴보았다.

09'58"

이번 강의를 마치기 전에 K means 알고리즘의 보편적인 활용법 가운데 하나를 얘기하고 싶다. 그것은 바로 잘 구분되지 않은 클러스터 문제이다. 이런 얘기다. 지금까지 우리는 K Means를 살펴보았고 이를 여기에 보이는 것과 같은 데이터 셋에 적용해왔다. 여기 보이는 클러스터들은 꽤 잘 나뉘어져 있다. 그리고 우리는 알고리즘을 사용해서 3개의 클러스터를 찾고 싶다. 하지만 결과적으로 K Means 알고리즘이 이처럼 잘 나누지 않은 여러 개의 클러스터가 있는 데이터 셋에도 자주 적용되는 것으로 드러났다. 여기 티셔츠 사이즈 같은 적용 예가 있다.

10'34"

예를 들어 여러분이 티셔츠 제조자라고 해보자. 여러분은 티셔츠를 판매하고 싶은 지역으로 가서 지역 사람들의 키와 몸무게를 조사해서 예시들을 수집했다. 사람들의 키와 몸무게는 아마도 양의 값을 가지기 때문에 이렇게 생긴 데이터 셋을 가지게 될 것이다. 그리고 여러 사람들의 몸무게와 키의 예를 예시의 집합으로 사용한다. 가령 여러분이 티셔츠 사이즈를 재기를 원한다고 해보자. 여러분이 세 가지 사이즈의 티셔츠(스몰, 미디움, 라지)를 디자인해서 팔고 싶다고 해보자. 스몰 사이즈는 어느 정도여야 할까? 미디움 사이즈는 어느 정도여야 할까? 라지 사이즈 티셔츠는 얼마나 크게 만들어야 할까? 한 가지 해결방법은 오른쪽의 이 데이터 셋을 가지고 k means 클러스팅 알고리즘을 실행하는 것이다. 이 알고리즘은 이 부분의 모든 점들을 하나의 클러스터로 묶을 것이고 이 점들을 두 번째 클러스터로 묶을 것이고 이 점들을 세 번째

클러스터로 묶을 것이다. 따라서 이전에는 3 개의 구분된 클러스터처럼 보이지 않았을지라도, K Means 알고리즘을 사용하면 친절하게도 여러분을 위해 클러스터를 여러 개 만들어 줄 것이다. 여러분이 할 수 있는 일은 이 첫 번째 그룹 사람들의 키와 몸무게를 보고 스몰 사이즈의 티셔츠를 디자인하는 것이다. 이 티셔츠는 첫 번째 그룹 사람들에게 잘 맞을 것이다. 같은 방법으로 미디움, 라지 사이즈 티셔츠를 제작한다. 이것은 사실 K Means 알고리즘을 사용해서 시장을 세 부분으로 나눈 차별화 마케팅의 예와 비슷하다. 여러분은 세 그룹의 사람들에게 맞는 세 가지 다른 사이즈의 티셔츠(스몰, 미디움, 라지)를 디자인할 수 있다.

12'17"

이것이 바로 K Means 알고리즘의 역할이다. 이제 여러분은 K Means 알고리즘을 어떻게 실행하는지, 문제 해결에 어떻게 사용하는지를 배웠다. 다음 강의에서는 K means 알고리즘의 핵심을 좀 더 깊이 살펴보고 실제로 잘 실행하는 법에 대해 좀 더 얘기하겠다.

No.	Title
Week 8-3	Optimization objective

00'00"

지금까지 살펴본 대부분의 지도 학습 알고리즘, 즉 선형 회귀, 로지스틱 회귀, 등등은 최소화하려고 하는 최적화 대상이나 비용 함수를 가지고 있었다. K means 알고리즘 역시 최소화하려고 하는 최적화 대상이나 비용 함수를 가지고 있는 것으로 드러났다. 이번 강의에서는 이 최적화 대상에 대해 얘기하겠다. 이것을 다루려는 이유는 두 가지 목적에서 유용하기 때문이다. 첫번째, kmeans 알고리즘의 최적화 대상을 안다는 것은 학습 알고리즘을 디버깅하는데 도움을 줘서 kmeans 알고리즘이 제대로 실행되도록 해준다. 아마도 더 중요할 수 있는 두번째 목적에 대해서는, 나중에 강의에서 최적화 대상을 이용해서 어떻게 kmeans 알고리즘의 더 적합한 비용을 찾고 국소 최적화를 피하는지 이야기할 것이다. 하지만 이것은 후반 강의에서 다룰 예정이다.

00'49"

K means 알고리즘이 실행되는 동안 짧게 복습하자면 우리는 두 세트의 변수를 추적할 것이다. 첫 번째는 클러스터 번호를 나타내는 c^i 로, 현재는 x^i 가 배정되어 있다. 우리가 사용할 또 다른 변수 세트는 μ_k 이다. 이 변수는 클러스터 중심점 k 의 위치를 가리킨다. K means 알고리즘에서는 대문자 K로 클러스터의 총 개수를 나타낸다. 여기 소문자 k 는 클러스터 중심점의 인덱스이다. 즉, $k = \{1, 2, \dots, K\}$ 이다. 좀 더 자세히 살펴보자. $\mu_{c(i)}$ 는 x^i 가 배정된 클러스터의 클러스터 중심점을

나타낸다. 좀 더 설명하면, 예를 들어 x^i 가 5번 클러스터에 배정되었다고 해보자. 이것이 의미하는 바는 x^i 의 인덱스인 $c^i = 5$ 라는 것이다. 즉, x^i 가 5번 클러스터에 배정되었다는 것의 의미는 $c^i = 5$ 라는 것이다. 그리고 $\mu_{c^i} = \mu_5$ 가 될 것이다. $c^i = 5$ 이기 때문이다. 따라서 μ_{c^i} 가 x^i 가 배정된 클러스터 5의 클러스터 중심점이 될 것이다.

02'30"

이제 우리는 k-means 클러스터링 알고리즘의 최적화 대상을 공부할 준비가 되었다. 여기를 보자. k-means의 비용 함수는 이 모든 변수, $c^{(1)}$ 에서 $c^{(m)}$ 까지, μ_1 에서 μ_K 까지의 J 함수를 최소화한다. k-means는 알고리즘이 실행되면서 다양해진다. 최적화 대상은 여기 오른쪽에 보이는 $\sum_{i=1}^m \|x^{(i)} - \mu_{c^i}\|$ 이다. 여기 빨간 색 박스 표시 부분이다. 그리고 이것을 제곱한다. 좀 더 설명해보겠다. 이것이 트레이닝 예시 $x^{(i)}$ 의 위치이고 이것이 $x^{(i)}$ 가 배정된 클러스터 centroid의 위치이다. 이것을 그림으로 설명해보면 여기가 x_1 이고 여기가 x_2 이고 이 점이 $x^{(i)}$ 라면, 그리고 $x^{(i)}$ 가 어떤 클러스터 중심점에 배정되었다면, 이 클러스터 중심점에 십자표를 하겠다. 여기가 μ_5 라고 하자. 저기 예를 든 것처럼 $x^{(i)}$ 가 클러스터 중심점 5에 배정되었다면 이 제곱거리, $x^{(i)}$ 점과 $x^{(i)}$ 점이 배정된 클러스터 중심점 간의 거리의 제곱이다. 그리고 k-means는 변수 C^i 와 μ_i 를 정의하려는 것처럼 보일 것이다. 즉, J 비용 함수를 최적하기 위해 C와 μ 를 찾는 것처럼 보일 것이다. 이 비용함수는 때로 왜곡비용함수, 또는 왜곡 k-means 알고리즘이라고 불린다.

04'10"

좀 더 자세히 살펴보자. 여기 k-means 알고리즘이 있다. 우리가 앞장 슬라이드에서 적은 그대로의 알고리즈다. 이 알고리즘의 첫 단계는 우리가 각각의 점들을 가까운 중심점으로 배정하는 클러스터 배정 단계였다. 그리고 클러스터 중심점을 μ_1 에서 μ_K 고정값으로 놓고 변수 $c^{(1)}$ 에서 $c^{(m)}$ 까지에 대해 어떤 클러스터 배정 단계가 J를 최소화하는지를 수학적으로 보일 수 있다. 클러스터 배정 단계가 하는 일은 클러스터 중심점을 바꾸지 않지만 비용 함수를 최소화하기 위해 또는 J 함수 왜곡을 최소화하기 위해 $c^{(1)}$ 에서 $c^{(m)}$ 까지 변수들을 정확하게 택하는 일이다. 이를 수학적으로 증명할 수 있지만 여기서는 넘어가자. 하지만 굉장히 직관적인 의미를 지닌다. 그러면 각각의 점들을 가장 가까운 클러스터 중심점으로 배정하자. 왜냐면 클러스터 중심점의 점들 간 거리의 제곱을 최소화하는 방법이기 때문이다.

05'28"

k-means의 두 번째 단계는 중심점 이동 단계였다. 이것 역시 증명하지는 않겠다. 하지만 수학적으로 중심점 이동 단계가 하는 일은 J를 최소화하는 μ 값을 택해서 비용 함수를 최소화하는 것이다. μ_1 에서 μ_K 까지의 클러스터 중심점 위치를 고려해서 J를 최소화하는 것을 나는 줄여서 wrt라 부른다. 즉, 실제로 하는 일은 여기 두 세트의 변수를 이렇게 두 부분으로 구분 짓는 것이다. C 세트 변수와 μ 세트 변수를 이렇게 나눈다. 처음으로 하는 일은 c변수를 고려해 J를 최소화하는 것이고 그리고 나서 μ 변수를 고려해 J를 최소화하고 계속 반복된다.

Q) k-means 알고리즘을 실행해서 제대로 돌아가는지 확인하려고 한다. 비용 함수 J를 다음과

같이 그렸다면 이것이 뜻하는 바는 무엇인가?

정답 4번

이것이 바로 k-means 알고리즘이 하는 일이다. 이제 k-means 알고리즘이 비용 함수 J 를 최소화하려고 한다는 것을 이해했으므로 이를 이용해 다른 알고리즘도 디버그 해볼 수 있다. 그리고 우리가 k-means 알고리즘이 제대로 작동하도록 만들었는지 확인할 수 있다.

06'40"

이제 우리는 k-means 알고리즘이 왜곡 함수라고도 불리는 비용 함수 J 를 최소화하려고 한다는 사실을 이해했다. 이것을 활용해 k-means 알고리즘을 디버그하고 k-means 알고리즘이 제대로 수렴되며 실행되는지 확인할 수 있다. 다음 강의에서는 이것을 사용해 k-means 알고리즘이 더 나은 클러스터를 찾고 k-means 알고리즘이 국소 최적값을 회피하도록 돋는 법을 살펴볼 것이다.

No.	Title
Week 8-4	Random Initializing

00'00"

이번 강의에서는 K-means를 초기화하는 법에 대해 얘기하고 나아가 더욱 중요한 주제인 K-means가 국소 최적값을 피할 수 있도록 만드는 방법을 살펴보자. 여기, 앞서 살펴본 K-means 클러스터링 알고리즘이 있다. 우리가 별로 언급하지 않았던 한 가지 단계는 바로 무작위로 클러스터 중심점을 초기화하는 것이다. 클러스터 중심점을 초기화하기 위해서 사용할 수 있는 방법에는 여러 가지가 있다. 하지만 이 중에서도 다른 방법들보다 훨씬 더 추천할 만한 방법이 한 가지 있다. 가장 잘 작동하는 것처럼 보이는 이 방법에 대해서 얘기해보자.

00'38"

여기 내가 주로 클러스터 중심점을 초기화하는 방법이 있다. K-means 알고리즘을 실행할 때, 클러스터 중심점 개수인 K 를 트레이닝 예시 M 의 개수보다 작게 놓아야 한다. 클러스터 중심점 개수를 내가 가진 트레이닝 예의 개수와 같거나 이보다 많게 놓은 채로 K-means 알고리즘을 실행하는 것은 정말 이상할 것이다. 그래서 내가 주로 K-means를 초기화하는 방법은 바로 무작위로 k 트레이닝 예를 선택하는 것이다. 그 다음은 μ_1, \dots, μ_K 를 이 k 예시들과 같다고 놓자. 좀 더 구체적인 예를 보자. $k=2$ 라고 가정하자. 그리고 여기 오른쪽 예에서 클러스터를 두 개 찾고 싶다고 하자. 이제 내가 할 일은 클러스터 중심점을 초기화하기 위해서 2 가지 예시를 무작위로 뽑는 것이다. 예를 들어 이 점과 이 점을 뽑았다고 하자. 내가 클러스터 중심점을 초기화하는 방식은, 나는 클러스터 중심점을 이 오른쪽에 보이는 두 개의 점으로 초기화할 것이다. 따라서 이 점이 나의 첫 번째 클러스터 중심점이고 이 점이 두 번째 클러스터 중심점이다. 그리고 이것은

K-means 알고리즘을 초기화하는 한 가지 랜덤한 방식이다. 지금 고른 두 점은 특허나 잘 고른 것 같다. 하지만 이만큼 운이 좋지 않아서 이런 점을 첫 랜덤 초기화 예로, 이 점을 두 번째 예로 고르는 경우도 있다. 그리고 이렇게 두 점을 선택하는 이유는 $k=2$ 이기 때문이다. 이렇게 무작위로 트레이닝 예 두 개를 선택했으면 나는 이 점을 나의 첫 클러스터 중심점으로, 이 점을 클러스터 중심점의 두 번째 초기화 점으로 가지게 될 것이다.

02'19"

이것이 바로 무작위로 클러스터 중심점들을 초기화하는 방법이다. 이렇게 초기화하면 여러분의 첫 클러스터 중심점 μ_1 은 x^i 와 같은데 여기서 i 는 무작위로 선택된 값이다. μ_2 는 x^j 와 같은데 여기서 j 는 무작위로 선택된 값이다. 만약 여러분이 클러스터와 클러스터 중심점을 많이 가지고 있다면 같은 방식으로 계속된다. 이전 강의에서 처음 K-means 알고리즘을 설명할 때 그때 슬라이드에서 단 한가지 목적은 K-means 알고리즘을 보여주는 것이었다. 그래서 클러스터 중심점을 초기화할 때 다른 방식을 사용했었다. 하지만 지금 슬라이드에서 설명하는 방식이 정말 추천할 만한 방법이다. 오른쪽에 보이는 이 두 가지 방식을 추천한다. 이제 여러분은 K-means 알고리즘은 클러스터들이 초기화되는 방식, 즉 랜덤 초기화이기 때문에 K-means 알고리즘이 다른 해들로 수렴한다고 생각할 수도 있다.

03'19"

특히나, K-means 알고리즘은 국소 최소값을 가지는 것으로 끝날 수 있다. 다음과 같은 데이터가 주어진다면 말이다. 보시다시피, 클러스터가 세 개 있는 것으로 보인다. 여러분이 K-means 알고리즘을 실행하고 좋은 국소 최소값으로 끝난다면, 여기 보이는 점들은 좋은 국소 최소값들인데, 클러스터링으로 끝날 수 있다. 하지만 운이 없다면, 무작위 초기화를 통해 K-means 알고리즘이 서로 다른 국소 최적화값 사이에 끼이게 될 것이다. 여기 왼쪽에 보이는 것처럼 파란색 클러스터는 왼쪽에 있는 점들을 상당히 많이 포섭했다. 하지만 녹색 클러스터는 상대적으로 적은 점을 포섭했다. 따라서 이것은 안 좋은 국소 최적값에 대응한다. 왜냐면 여기 클러스터 두 개를 하나로 만들었고 하나로 통합했으며 나아가서 두 번째 클러스터를 두 개의 부분 클러스터로 나눴기 때문이다. 즉, 두 번째 클러스터를 이렇게 두 개의 다른 부분 클러스터로 나눈다. 따라서 오른쪽 하단의 이 예들은 K-means 알고리즘의 다른 국소 최적값에 대응한다. 사실, 여기 예시처럼 빨간 클러스터는 한 가지 최적값만 포함했다.

04'29"

한 가지 덧붙이면 국소 최적값이라는 용어는 왜곡 함수 J 의 국소 최적값을 의미한다. 그리고 왼쪽 하단의 해들은 이 국소 최적값들이 대응하는 값으로 K-means 알고리즘이 국소 최적값에 끼어있다는 것을 의미하기 때문에 이는 왜곡 함수 J 를 최소화하지 못하고 있다는 뜻이다. 따라서 K-means 알고리즘이 국소 최적값에 끼인 것이 걱정되고 K-means 알고리즘이 가장 좋은 클러스터를 찾을 확률을 높이고 싶다면 여기 꼭대기처럼, 우리는 무작위 초기화를 여러 번 시행할 수 있다. 따라서 K-means 알고리즘을 한번 초기화하고 제대로 실행되기를 기대하는 대신 K-means 알고리즘을 여러 번 초기화하고 K-means 알고리즘을 여러 번 실행해서 가장 좋은 해,

가장 좋은 국소 혹은 전역 최적값을 얻도록 시도하는 것이다. 구체적으로 보자.

05'20"

여기 실행 방법이 있다. 예를 들어 내가 K-means 알고리즘을 100번 실행하기로 결정했다고 하자. 나는 이 반복문을 백 번 반복할 것인데 50에서 1,000 사이가 일반적인 시행 횟수이다. 자 그래서 K-means 알고리즘을 100번 실행하기로 결정했다고 하자. 이것은 우리가 K-means 알고리즘을 무작위로 초기화한다는 것을 의미한다. 각각의 무작위 초기화 100번에 대해 우리는 K-means 알고리즘을 실행할 것이고 그 결과 클러스터와 클러스터 중심점 여러 개 생성된다. 그리고 나서 왜곡 함수 J 를 계산할 것이다. 우리가 얻은 여러 개의 클러스팅과 클러스터 중심점에 대해 이 비용 함수를 계산하는 것이다. 최종적으로 이 전체 과정을 백 번 반복한다. 그러면 여러분은 데이터를 클러스팅하는 백 개의 다른 방법을 사용하게 된 것이고 이 중에서 가장 낮은 비용 값을 가진 하나를 선택하면 된다. 그러면 왜곡이 가장 작게 된다. K-means 알고리즘을 무척 작은 클러스터 개수로 실행하게 되면, 물론 여러분은 클러스터 개수가 2개에서 10개 사이 정도라는 사실을 이미 알고 있지만, 무작위 초기화를 여러 번 수행하면 더 나은 국소 최적값을 발견하는데 도움이 된다. 더 나은 클러스팅 데이터를 발견하도록 하라. 하지만 만약 K 가 굉장히 크다면, 예를 들어 K 가 10보다 크다면, 물론 우리가 클러스터를 수백 개 찾으려고 할 때 얘기지만, 무작위 초기화를 여러 번 수행하는 것은 그다지 차이를 만들어 내지 않을 것이다. 오히려 처음 실행한 무작위 초기화가 꽤 괜찮은 해를 도출했을 확률이 더 높다. 무작위 초기화를 여러 번 실행하는 것은 아주 조금 더 나은 해를 구하도록 해 줄 수 있지만 그렇게 큰 차이가 생기는 것은 아니다. 하지만 여러분이 상대적으로 클러스터를 적게 가지고 있는 체제라면, 특히 예를 들어 무작위 초기화를 시행할 클러스터를 2개, 3개, 또는 4개 가지고 있다면 왜곡 함수 J 를 최소화하고 좋은 클러스팅을 가져온다는 측면에서 큰 차이를 보일 수도 있다.

Q) k-means를 초기화하는 데는 다음 중 어떤 방법이 적합한가?

정답 3번

07'20"

지금까지 K-means 알고리즘과 무작위 초기화를 살펴보았다. 만약 여러분이 비교적 적은 개수의 클러스터, 즉 2개, 3개, 4개, 5개, 또는 6개, 7개를 가지고 무작위 초기화를 여러 번 실시한다면 이것은 때때로 데이터에 대해 더 좋은 클러스팅을 하는 결과를 가져올 수 있다. 하지만 많은 개수의 클러스터를 가지고 있을 때에도, 내가 강의에서 설명한 것처럼, 초기화, 무작위 초기화 방법은 K-means 알고리즘에 좋은 클러스터를 만들 적절한 시작점을 제공해 줄 것이다.

No.	Title
Week 8-5	Choosing the number of Clusters

00'00"

이번 강의에서는 K-means 클러스터링의 마지막 남은 사항에 대해 얘기하겠다. 바로 클러스터 개수를 선택하는 방법이다. 또는 변수 capsule K의 값을 선택하는 방법이다. 솔직히 말해서 여기에는 딱히 좋은 방법이나 자동적으로 해결하는 방법이 없기 때문에 지금까지 클러스터 개수를 선택할 때 가장 많이 쓰는 방법은 여전히 직접 보고 선택하거나 클러스터링 알고리즘의 출력값 등을 보고 선택하는 방법이다. 나는 클러스터 개수를 선택하는 방법에 대한 질문을 꽤 자주 받기 때문에 현재 사람들이 이에 대해 가지고 있는 생각과 클러스터 개수를 직접 선택하는 가장 보편적인 방법을 얘기하고 싶었다.

00'42"

클러스터 개수를 선택하는 일이 언제나 쉽지 않은 이유 중 가장 큰 부분을 차지하는 것은 데이터에 얼마나 많은 클러스터가 있을 지가 대체로 모호하기 때문이다. 이 데이터 셋을 보면 여러분 중 일부는 클러스터 개수가 네 개로 보일 것이고 이는 $k=4$ 라는 것을 의미한다. 하지만 다른 일부는 두 개로 보일 것이고 그러면 $k=2$ 이다. 또 다른 사람들은 세 개라고 생각할 수도 있다. 따라서 이 같은 데이터 셋을 보는 것은, 즉 진정한 클러스터 개수는 실제로 정말 모호하다. 그리고 나는 여기에 정답이 있다고 생각하지 않는다. 이것은지도 학습의 일부이다. Label이 주어지지 않았기 때문에 항상 딱 맞는 정답이 있는 것이 아니다. 그리고 문제를 더 어렵게 만드는 것은 자동적인 알고리즘을 사용해서 클러스터 개수를 결정하는 것이다.

01'32"

사람들이 클러스터 개수를 결정하는 방법에 대해 얘기할 때, 가끔 회자되는 방법이 Elbow Method이다. 여기에 대해 좀 더 설명하고 장단점을 얘기하겠다. Elbow Method는 클러스터의 총 개수인 K값을 다양하게 하는 것이다. 이제 클러스터 1개로 K-means 알고리즘을 실행하자. 이것은 모든 점이 하나의 클러스터로 무리 지어 진다는 것이고 비용 함수나 J 왜곡 함수를 계산해서 여기 그린다는 것이다. 그리고 나서 우리는 클러스터 2개로 K-means 알고리즘을 실행할 것이다. 여러 개의 무작위 agent를 가질 수도 있고 아닐 수도 있다. 그리고 우리는 클러스터 2개와, 아마도 작은 왜곡 값을 가질 것이다. 여기 그려라. 이제 클러스터 3개로 K-means 알고리즘을 실행한다. 이번엔 더더욱 작은 왜곡값(Distortion)을 가질 수도 있다. 여기 그려라. 클러스터 4개, 5개, 이후도 마찬가지이다. 그 결과 왜곡이 어떤지 보여주는 곡선, 즉 함수가 클러스터 개수가 늘어날수록 감소하는 것을 볼 수 있다. 그러면 이렇게 하강하는 곡선을 얻게 된다. Elbow Method가 이 곡선을 보면 "이 점을 봐. 꼭 팔꿈치(elbow)처럼 생겼네."라고 할 것이다. 그렇다. Elbow Method는 아마도 인간의 팔을 비유적으로 가져온 것일 것이다. 팔을 뻗는다고 상상해보자. 그러면 여기가 어깨 관절, 이 지점이 팔꿈치 관절, 마지막 끝 부분이 아마 손일 것이다. 그래서 Elbow Method라고 불린다. 여러분은 여기서 패턴을 발견할 수 있는데 바로 1에서 2, 2에서 3으로 가면서 왜곡이 급격하게 줄어드는 것을 볼 수 있다. 그리고 여기 팔꿈치 부분인 3에서부터 왜곡이 줄어드는 속도가 매우 느려진다. 그렇기 때문에, 클러스터 3개를 사용하는 것이 적절한 개수인 것처럼 보일 수 있다. 왜냐면 이 곳이 이 곡선의 팔꿈치 부분이기 때문이다. $K=3$ 이 될

때까지 왜곡은 급격하게 줄어든다. 그리고 이후에는 완만하게 감소한다. 따라서 이것이 클러스터 개수를 선택하는 합리적인 방법일 수 있다.

03'35"

Elbow Method는 그렇게 자주 사용되지는 않는다. 한 가지 이유는 여러분이 만약 클러스트링하는데 Elbow Method를 사용한다면 종종 이처럼 모호해 보이는 곡선을 얻게 되기 때문이다. 그리고 이것을 보면, 분명하게 팔꿈치처럼 보이는 부분이 없고 왜곡이 끊임없이 내려가는 것처럼 보인다. 30이 괜찮은 숫자일 수도, 4일 수도 있고, 5도 괜찮아 보인다. 실제로 이렇게 되면 여기 왼쪽처럼 점을 찍으면 괜찮다. 이 경우에는 분명한 답이 나오겠지만 실제로는 여기 오른쪽 이렇게 생긴 곡선으로 끝날 수도 있는데 이렇게 되면 팔꿈치가 시작되는 부분이 명확하지 않게 된다. 이렇게 되면 Elbow Method를 사용해서 클러스터 개수를 선택하는 것이 어려워진다. 따라서 짧게 요약하면 Elbow Method는 시도할 만한 가치가 있지만 나라면 굳이 시도하지 않을 것이다. 어떤 특정한 문제를 해결하는데 대한 기대가 크기 때문이다.

Q) $k=3$, $k=5$ 를 사용하는 k-means를 실행한다고 하자. $k=3$ 보다 $k=5$ 일 때 비용함수 J 값이 훨씬 높다고 하자. 이것이 의미하는 바는?

정답 3번

04'30"

마지막으로 k 값을 선택하는 또 다른 방법이 있다. 사람들은 종종 나중을 위해 클러스터를 얻으려고 또는 downstream 목적으로 K-means 알고리즘을 실행한다. 여러분들도 T-shirt 사이즈 결정 예시에서처럼 시장 구분을 목적으로 K-means 알고리즘을 실행하고 싶을 수 있다. 어쩌면 computer 클러스터를 개선시키거나 다른 여러 가지 목적으로, market segmentation같은 downstream 후자의 해당하는 목적으로, 클러스터를 학습하기 위해서 K-means 알고리즘을 실행하고 싶을 수 있다. 평가방법이 제시되어 있다면 일반적으로 클러스터 개수를 선택하는 더 나은 방법은 다른 개수의 클러스터가 later downstream 목적을 얼마나 충실히 달성하는지를 보는 것이다. 구체적인 예시를 통해 살펴보자. 다시 한번 티셔츠 사이즈 예를 보자. 내가 과연 세 가지 사이즈의 티셔츠를 원하는지를 결정하려고 한다. 따라서 $k=3$ 으로 두면 스몰, 미디움, 라지의 세 가지 사이즈의 티셔츠를 의미한다. $K=5$ 로 두면, 엑스트라 스몰, 스몰, 미디움, 라지, 엑스트라 라지의 다섯 가지 사이즈의 티셔츠를 갖게 된다. 따라서 티셔츠 사이즈 종류를 3개, 4개, 5개 등 선택할 수 있다. 4개를 선택할 수도 있지만 간단히 하기 위해서 여기서는 3개와 5개를 선택할 경우를 살펴보겠다.

05'46"

$K=3$ 인 K-means 알고리즘을 실행한다면 이 클러스터는 스몰 사이즈, 이 클러스터는 미디움 사이즈, 이 클러스터는 라지 사이즈이다. 반면 $K=5$ 로 두면 이 클러스터가 엑스트라 스몰, 이 클러스터가 스몰, 이 클러스터가 미디움, 이 클러스터가 라지, 이 클러스터가 엑스트라 라지가 된다. 이 예시의 좋은 점은 우리가 3개, 4개, 혹은 5개 클러스터를 원하든 간에 선택할 수 있다는

것이다. 특히, 여러분이 할 수 있는 일은, 이것을 티셔츠 비즈니스의 관점에서 생각하고 물어보자. “만약 사이즈 종류가 5개라면 티셔츠가 고객에게 얼마나 잘 맞을 것이며 얼마나 많이 팔 수 있을 것인가? 고객은 얼마나 행복해할 것인가?” 이것은 티셔츠 비즈니스의 관점에서, 티셔츠 사이즈 종류가 더 많은 고객에게 더 잘 맞는다는 관점에서 합리적이다. 아니면 티셔츠 사이즈 종류를 줄여서 티셔츠 종류를 적게 생산하고 싶을 수도 있다. 그리고 고객에게 좀 더 낮은 가격에 판매할 수도 있을 것이다. 이렇듯 티셔츠 생산 사업은 클러스터 개수를 세 개로 할 것인지 아니면 다섯 개로 할 것인지 선택권을 준다. 어떤 종류의 티셔츠를 생산할지 결정하는 문제처럼 later downstream purpose의 예를 준다. 즉 클러스터의 수를 결정하는 평가방법을 제시한다. 프로그래밍 연습을 하는 분들은 이번 주 프로그래밍 연습문제 가운데 K-means 알고리즘 부분을 보면 이미지 압축을 위해 K-means 알고리즘을 사용하는 예시가 있을 것이다. 이 문제에서 클러스터 개수를 선택할 때 다시 한 번 이미지 압축의 평가방법을 사용해서 클러스터 개수 K를 선택할 수 있다. 다시 말해, 이미지 대 파일 사이즈 압축 문제로 이미지를 얼마나 잘 보이게 할 것이냐의 문제이다. 프로그래밍 연습을 직접 해보면 지금 내가 하는 말이 더 잘 와 닳을 것이다.

07'54"

요약하면, 대부분의 경우, 고객 수 K는 여전히 사용자의 관찰과 사용자의 혜안으로 직접 결정된다. 또 다른 방법은 Elbow Method를 사용하는 것이지만, 그다지 잘 작동하지는 않는다. 하지만 클러스터 개수를 선택하는 문제에 대해 생각해보고 어떤 목적으로 K-means 알고리즘을 실행하는지 물어보기에 좋은 방법이다. 그리고 K-means 알고리즘을 실제로 실행하는 목적이 무엇이건 간에 클러스터 개수 K가 그 목적에 부합하는지 생각해보자.

No.	Title
Week 8-6	Motivation 1-Data compression

00'00"

이번 강의에서는 자율 학습 문제의 두 번째 유형인 차원 감소(dimensionality reduction)에 대해 얘기하겠다. 사람들이 차원 감소를 하고 싶어하는 몇 가지 다른 이유가 있다. 하나는 데이터 압축이다. 나중에 몇 강의 뒤에 보겠지만, 데이터 압축은 우리가 데이터를 압축하도록 해 줄 뿐만 아니라 컴퓨터 메모리와 디스크 공간을 덜 차지하게 하고, 또 학습 알고리즘의 속도를 높여준다. 하지만 먼저 차원 감소에 대해 얘기하자.

00'35"

동기가 부여되는 예시로서, 우리가 굉장히 많은 값을 가진 데이터 셋을 수집했다고 가정하자. 여기 그 중 2가지를 그렸다. 우리에게 알려지지 않았던 두 가지 값 중 하나는 무언가의 cm으로 표시되는 길이이고 다른 하나인 x_2 는 인치로 표시되는 무언가의 길이이다. 따라서 반복되는 표현이므로, 둘 다 길이를 측정하는 것이기 때문에, x_1 , x_2 라는 두 개의 분리된 기호로 나타내는 대신에 데이터를 1차원으로 감소시켜서 길이를 측정하는 기호를 하나로 만들고 싶을 수 있다. 이 예시가 부자연스럽다면, 사실 센티미터와 인치 예시는 비사실적이지만 업계에서 일어나는 일과 크게 다르지는 않다. 만약 여러분이 수 백 혹은 수 천 개의 값을 가지고 있다면, 여러분이 정확히 어떤 값을 가지고 있는지 파악하지 못하게 되기 쉽다. 어떤 경우에는 엔지니어링 팀이 여러 개 있을 때, 한 엔지니어링 팀은 200개의 값을 주고, 두 번째 엔지니어링 팀은 300개의 값을 주고, 세 번째 엔지니어링 팀은 500개의 값을 준다면, 여러분은 다 합쳐서 1,000개의 값을 갖게 되고 이렇게 되면 어떤 팀에서 어떤 값을 받았는지 정확하게 파악하기 어려워진다. 따라서 이렇게 반복되는 값을 갖고 싶지 않을 것이다. 센티미터로 측정할 때 길이가 가장 가까운 센티미터로 반올림되고 인치도 가장 가까운 인치로 반올림되었다. 이것이 바로 이 X점들이 직선으로 정렬되지 않은 이유이다. 가장 가까운 센티미터와 인치로 반올림 될 때의 에러 때문이다. 만약 우리가 데이터를 2차원 대신 1차원으로 감소시킬 수 있다면 반복을 줄일 수 있다.

02'15"

덜 부자연스러운 다른 예시를 보자. 수년간 나는 무인 헬리콥터 조종사 또는 유인 헬리콥터 파일럿과 일해 왔다. 만약 여러분이 이 다른 파일럿들을 대상으로 조사나 테스트를 진행한다면, 여러분은 헬리콥터 파일럿들의 기술을 가리키는 샘플 하나, x_1 과 파일럿의 만족도를 나타내는 x_2 를 가질 것이다. 이것은 파일럿이 비행을 얼마나 즐기는지를 나타낸다. 이 두 샘플들은 높은 상관관계가 있다. 여러분이 신경 쓰는 것은 이 방향과 파일럿 적성을 측정하는 다른 샘플일 것이다. 나는 이 샘플을 명명하려고 하는데 다시, 이 두 샘플이 높은 상관관계를 가진다면 여러분은 차원을 감소시키고 싶을 것이다. 차원을 감소시킨다는 것이 무엇을 의미하는지 좀 더 설명하자면, 2차원, 즉 2D의 데이터를 1차원, 또는 1D로 가져온다는 것이다.

03'22"

다른 색을 사용해서 표시해보겠다. 이 경우에 차원을 감소시킨다는 것은 이 투영선을 찾아서, 즉 이 데이터들이 위치한 선을 찾아서 이 모든 데이터를 이 일직선에 위치시킨다는 것이다. 그리고 이렇게 함으로써, 나는 선 위의 각각의 점의 위치를 측정할 수 있다. 나는 새로운 z_1 을 만드는데, 선 위의 위치를 특정하는 데는 단 하나의 새로운 수치만 필요하므로, 이제 새로운 z_1 이 녹색 선 위의 점들의 위치를 특정한다. 이것이 의미하는 바는 이전에 내가 가진 예시 $x^{(1)}$, 이 점이 $x^{(1)}$ 이라고 하자. $x^{(1)}$ 을 표시하려면 이차원의 숫자, 이차원의 벡터가 필요했다. 하지만 이제 $z^{(1)}$ 은 일차원의 벡터로 표시할 수 있다. 이제 $z^{(1)}$ 을 사용해서 첫 번째 예를 나타낼 것이다 이것은 실수가 될 것이다. 같은 방식으로 나의 두 번째 예인 $x^{(2)}$ 도 나타내려면 두 가지 숫자가 필요했다. 여기 그래프에서 검은 점에 해당한다. 이제는 실수 $z^{(2)}$ 만 있으면 여기 선 위에 표시할 수 있다. 이를 모든 M개의 예시까지 적용한다.

04'58"

요약하면, 원래 가지고 있던 데이터 셋에서 오리지널 예시들을 이 녹색 선위에 위치시키면 한 가지 숫자만으로 나타낼 수 있다. 실수만 있으면 특정 점의 선 위의 위치를 구체화할 수 있다. 따라서 내가 가진 트레이닝 예시들을 녹색 선위에 위치시킨 후 이 한 가지 숫자만 사용해서 각각의 위치를 나타낼 것이다. 이것은 오리지널 트레이닝 셋을 대략적으로 나타낸 것이다. 왜냐면 내가 가진 트레이닝 예시들을 모두 선 위에 위치시켰기 때문이다. 하지만 이제부터는 각각의 예시에 대해 한 숫자만 유지할 필요가 있다. 이것은 메모리나 공간 요구량, 혹은 여러분이 데이터를 저장한 어떤 장소를 이등분한다. 더 흥미롭고 중요한 점은, 나중 강의에서 다루겠지만, 이것이 우리의 학습 알고리즘의 속도를 높여준다는 것이다. 이는 데이터 압축을 메모리나 디스크 저장 공간을 줄이는 것보다 더 흥미롭게 활용하는 법이다.

06'10"

이전 슬라이드에서 우리는 데이터를 2차원에서 1차원으로 감소시키는 예를 보았다. 이번 슬라이드에서는 차원에서 2차원으로 감소시키는 예를 보자. 차원 감소의 전형적인 예는 1,000개의 차원 데이터 혹은 1000D 데이터를 갖는 것이다. 이 데이터의 차원을 100 차원, 혹은 100D로 감소시키려고 한다. 하지만 여기 슬라이드에 적을 수 없다는 한계 때문에 3D에서 2D로, 또는 2D에서 1D에 해당하는 예시를 사용할 것이다. 여기 보이는 데이터 셋을 가진다고 하자. 즉, $x^{(i)} \in R^3$ 로 표시된다. 이것은 3차원 예시이다. 이 슬라이드 상으로 보기 어려울 수도 있겠지만, 3D 점 cloud를 보여주겠다. 여기 잘 보이지 않을 수도 있는데, 이 모든 데이터는 여기 평면에 놓여 있다. 이제 차원을 감소시키려면 이 데이터를 꺼내서 2차원 평면에 놓는다. 나도 이 데이터를 들어내서, 데이터를 전부 놓았다. 이제 평면에 놓여있다.

07'21"

이제 마지막으로 평면 내에서 점의 위치를 나타내려면 숫자가 두 개 필요하다. 우리는 이 축을 따라 점의 위치를 나타내려고 한다. 그리고 이 축도 이용할 것이다. 따라서 두 개의 숫자, z_1 과

Z_2 을 가지고 평면 내 점의 위치를 나타낼 수 있다. 이것이 의미하는 것은 우리가 각각의 예시, 각각의 트레이닝 예를 이 두 숫자, 내가 여기 쓴 Z_1 과 Z_2 로 나타낼 수 있다는 것이다. 우리의 데이터는 벡터를 사용해서 $z^{(1)} \in R^2$ 로 나타낼 수 있다. 그리고 여기 z 집합 Z_1, Z_2 은 $z^i, z_1^{(i)}$, $z_2^{(i)}$ 로 나타낼 수 있다. 이전 슬라이드에서 1차원으로 감소시킬 때는 Z_1 만 있었던 것을 기억하는가? $z_1^{(i)}$ 이 이전 슬라이드에서는 Z_1 이었지만 여기서는 2차원 데이터를 가지고 있기 때문에 Z_1, Z_2 두 개를 데이터로 사용한다.

08'40"

이 값들이 말이 되는지 확인해 보자. 3개의 그래프를 3D 버전으로 다시 한 번 띄워보자. 우리가 거친 과정은 최적화된 데이터 셋으로 가운데에 2D로 나타나 있다. 오른쪽에 2D 데이터 셋은 Z_1 과 Z_2 을 축으로 가진다. 좀 더 자세히 살펴보자. 여기 왼쪽에 오리지널 데이터 셋이 있다. 이렇게 3D cloud 점들로 시작했었다. 축은 x_1, x_2, x_3 로 표시되어 있다. 3D 점도 있지만 데이터의 대부분은 대체로 2D 평면에 가깝게 누워있다. 따라서 이 데이터를 가지고 2D 평면에 나타낼 것이다. 이제 대부분 2D 평면에 위치해 있다. 모든 점을 평면에 위치시켰기 때문에 이제 모든 데이터가 평면 위에 놓인 것을 볼 수 있다. 이것은 2가지 숫자, Z_1 과 Z_2 를 가지고 평면 위 점들의 위치를 나타낼 수 있다는 뜻이다. 3D를 2D로 줄이기 위해 우리가 거쳐야 할 과정이다.

Q) 우리가 이 m 데이터 셋에 대해 차원 감소를 적용한다고 가정하자. 이 때 $x^{(i)} \in R^n$ 이다. 그 결과 우리가 얻는 것은 무엇인가?

정답 3번

09'45"

이것이 바로 차원 감소이고 이를 사용해 데이터를 압축하는 법이다. 나중 강의에서 이를 사용해 학습 알고리즘을 실행시킬 것이지만 이것은 나중 강의에서 다루자.

No.	Title
Week 8-7	Motivation 2-Visualization

00'00"

이전 강의에서 우리는 데이터를 압축하기 위해 차원 감소를 살펴보았다. 이번 강의에서는 차원감소와 데이터를 시각화하는 두 번째 활용법을 알아보겠다. 수 많은 머신 러닝 활용 가운데 데이터를 좀 더 잘 알면 효과적인 학습 알고리즘을 개발하는데 정말 도움이 된다. 데이터를 더 잘 시각화하는 방법이 있는데, 즉 차원 감소는 이를 이루는 또 다른 유용한 도구이다. 예시로 시작해보자.

00'31"

전세계 많은 국가에 대한 통계와 사실들로 구성된 큰 데이터 셋을 수집했다고 가정하자. 첫 번째로 X_1 은 국가의 GDP, 혹은 국내 총 생산, X_2 는 1인당 GDP를, X_3 은 인간개발지수, 기대 수명을, X_5 , X_6 도 이런 식으로 계속된다. 이렇게 우리가 방대한 데이터 셋을 가지고 있을 때, 예를 들어 각 국가마다 50개 정도의 정보를 가지고 있을 때, 우리는 국가에 관한 방대한 데이터 셋을 가지게 된다. 이 데이터를 더 잘 이해하기 위해 우리가 할 수 있는 방법이 있는가? 이 방대한 숫자의 표를 고려해 볼 때, 우리는 이 데이터를 어떻게 시각화하는가? 우리는 50개의 표본을 가지고 있다고 상정할 때 50 차원의 데이터를 그리기란 매우 어렵다. 이 데이터를 점검하는 좋은 방법은 무엇인가? 차원을 감소시켜서 각 나라를 이 벡터 X^i 를 사용해서 나타내는 대신, 예를 들어 캐나다 같은 나라를 $X^i \in R^{50}$ 으로 나타내는 대신에 Z 벡터인 R^2 을 사용해보자.

01'48"

그렇다면 Z_1 과 Z_2 의 숫자 쌍이 50개의 표본을 요약하게 되었다. 이제 이 국가들을 벡터 R^2 에 그린다. 그리고 이를 사용해서 다른 국가들을 좀 더 잘 이해할 수 있다. 우리가 할 수 있는 일은 50차원의 데이터를 2차원으로 감소시키는 것이다. 그러면 2차원에 표현할 수 있고 그렇게 되면, 차원 감소 알고리즘을 보면, 그리고자 하는 이 새로운 표본들에 물리적인 의미를 부여하지는 않는다. 따라서 이 표본들을 해석하는 것은 대개 우리의 몫이다.

02'38"

우리가 이 표본들을 썼다면 다음을 알 수 있다. 여기서 국가들은 Z^i 로 나타낼 수 있고 $Z^i \in R$ 이다. 이 점들과 숫자는 국가를 나타낸다. 여기가 Z_1 이고 Z_2 이다. 그리고 여기 몇몇 점들을 강조한다. 여기서 가로 축인 Z_1 축이 대응하는 것은 국가의 크기나 국가의 경제활동 규모이다. 따라서 GDP, 또는 한 국가의 경제 크기이다. 반면 수직 축은 1인당 GDP에 해당한다. 또는 개인의 웰빙, 또는 개인의 경제적인 활동을 뜻하며 여러분은 이 50개의 표본을 이렇게 2차원으로 놓았다. 여기 이 점을 국가, 예를 들어 미국이라고 하면, 미국은 상대적으로 GDP크기가 크고, 상대적으로 1인당 GDP도 높다. 반면 이 점이 싱가포르라고 해보자. 1인당 GDP는 매우 높지만 국토가 작기 때문에 전체 경제 크기는 미국보다 훨씬 작다. 그리고 여기 이 점에 해당하는 국가는 불행하게도

국민들이 상대적으로 부유하지 못하고 아마 기대 수명도 더 짧을 것이다. 건강 보험도 더 적고 경제 성숙도도 낮을 것이다. 이 점에 해당하는 더 작은 규모의 국가들은 꽤 경제 활동 규모가 크지만 국민들이 누리는 부는 상대적으로 적다. 따라서 이 Z_1 과 Z_2 축이 2차원의 변수들 가운데 여러 국가의 특징을 가장 간결하게 잘 알아보도록 도와줄 것이다. 예를 들어 어떤 국가의 전반적인 경제 규모로 투영할 수 있는 경제활동 규모나 1인당 GDP로 측정되는 개인의 웰빙, 1인당 건강보험 등이 해당한다.

05'01"

Q) $x^{(i)} \in R^n$ 인 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ 인 데이터 셋을 가지고 있다고 하자. 이를 시각화하기 위해서 차원 감소를 실시해서 k차원인 $z^{(i)} \in R^k$ 인 $\{z^{(1)}, z^{(2)}, \dots, z^{(m)}\}$ 을 얻었다. 여기서 참은 무엇인가?

정답 3번

지금까지 차원 감소를 하는 법을 살펴보았다. 데이터를 50 차원에서 2차원으로 혹은 3차원으로 감소시켜서 표현할 수 있게 되고 데이터를 더 잘 이해할 수 있다. 다음 강의에서는 PCA, 또는 Principal Component Analysis라 불리는 알고리즘을 살펴본다. 이 알고리즘으로 차원 감소를 해보고 앞서 언급한 데이터 압축에 활용하는 법을 살펴보자.

No.	Title
Week 8-8	Principal Component Analysis Problem Formulation

00'00"

차원을 감소시킬 때 지금껏 가장 인기가 있고 보편적으로 쓰이는 알고리즘이 바로 Principal Component Analysis 또는 PCA 알고리즘이다. 이번 강의에서는 이 알고리즘의 문제 구체화에 대해 얘기하겠다. 다시 말해, 이 알고리즘으로 정확히 무엇을 할지 공식화 해보자.

00'20"

다음과 같은 데이터 셋을 가지고 있다고 하자. 즉, $X \in R^2$ 이다. 이 데이터의 차원을 2차원에서 1차원으로 감소시키고 싶다고 하자. 다시 말해, 이 데이터를 위치시킬 선을 찾고 싶다. 따라서 데이터를 위치시킬 좋은 선은, 바로 다음과 같이 그리면 상당히 좋은 선택이 될 것이다. 이것이 좋은 선택이 될 것 같은 이유는 여기 점들이 표시되는 것을 보면 알 수 있는데, 여기, 그리고 여기 아래쪽에 표시하겠다. 즉, 점들이 여기, 여기, 여기, 그리고 여기 위치된다. 그리고 각각의 점과 선 사이의 거리가 꽤 짧다는 사실을 알 수 있다. 그러니까 여기 푸른 색 선들이 꽤 짧다는 것이다. PCA가 하는 일은 더 낮은 차원의 곁 표면을 찾아서 그러니까 지금 경우에는 선이지만, 데이터를 위치시키는 것이다. 그러면 이 파란 선으로 표시되는 거리 값의 총합의 제곱이 최소화된다. 파란 선의 길이는 때때로 투영 오류라고 불리기도 한다. 따라서 PCA가 하는 일은

데이터를 위치시킬 위치를 잘 찾아서 오류를 최소화시키는 것이다. 그리고 PCA를 실행하기 전에, 먼저 할 일은 값들을 평균 정규화해서, 즉 x_1 와 x_2 값을 조정해서 평균이 0이 되게 해야 한다. 그래서 값을 비교할 수 있게 만든다. 이 예시에 대해서는 이미 했으므로 PCA의 관점에서 값을 조정하고 정규화하는 법에 대해서는 나중에 다시 다루겠다.

02'00"

다시 예시로 돌아와서 내가 방금 그은 빨간 선과 대조적으로 내가 데이터를 위치시킬 또 다른 선을 이렇게 그린다. 바로 이 자주색 선이다. 그리고 여기 보이듯이 이 선은 데이터를 위치시키기가 더 힘든 방향이다. 그래서 만약 내가 이 자주색 선 위에 나의 데이터를 위치시키면 이렇게 될 것이다. 그리고 투영 에러는, 이 파란 색들이 굉장히 크다. 따라서 이 점들이 자주색 선 위에 표시되려면 굉장히 긴 거리를 움직여야 한다. 이것이 바로, PCA, principal components analysis가 여기 자주색 선보다 빨간 선을 선택하는 이유이다.

02'44"

PCA 문제를 좀 더 공식적으로 써보자. PCA의 목표는, 만약 우리가 데이터를 2차원에서 1차원으로 감소시키려고 한다면, 우리는 벡터를 찾기를 원할 것이다. 즉 $u^1 \in R^n$ 이다. 그리고 이 경우에는 R^2 일 것이다. 그래서 데이터를 위치시킬 방향을 이렇게 찾을 것이다. 이것은 투영 에러를 최소화하기 위함이다. 그래서 이 예시에서 나는 PCA가 이 벡터를 찾기를 바란다. 나는 이것을 $u^{(1)}$ 으로 놓을 것이다. 그런데 이 벡터를 확장해서 그은 이 선위에 데이터를 위치시킬 때 작은 재구축 오류가 발생하는 경우가 많다. 데이터 reference는 다음과 같다. 그리고 PCA가 $u^{(1)}$ 을 도출하든지 $-u^{(1)}$ 을 도출하든지 상관없다는 점을 짚고 넘어가겠다. PCA가 이 방향으로 나가는 양의 벡터 $u^{(1)}$ 을 줘도 괜찮고 아니면 반대 방향으로 향하는 반대 값의 벡터 $-u^{(1)}$ 을 줘도 괜찮다. 여기 대신 파란색으로 다시 그리겠다. 다시 말해 양의 벡터 $u^{(1)}$, 음의 벡터 $-u^{(1)}$ 이든 관계없다. 왜냐면 둘 다 내가 데이터를 위치시키는 빨간 선 위에 있기 때문이다.

03'54"

지금까지 데이터를 2차원에서 1차원으로 감소시키는 경우였다. 더 보편적인 경우에 우리는 n 차원 데이터를 가지고 이를 k 차원으로 감소시키려고 한다. 이 경우, 우리는 데이터를 위치시킬 단 하나의 벡터를 찾기를 원할 뿐 아니라 데이터를 위치시킬 k 차원도 찾고 싶어 한다. 이는 투영 오류를 최소화하기 위함이다. 여기 예시가 있다. 3차원 점들이 여기 구름처럼 많이 있다. 그러면 나는 벡터를 찾고 싶다. 즉 벡터 한 쌍을 찾고 싶다. 이것을 벡터라 부를 것이다. 벡터들을 빨간 색으로 그리겠다. 나는 한 쌍의 벡터를 여기 중심지에서 찾을 것이다. 이것이 벡터 $u^{(1)}$ 이고 여기가 나의 두 번째 벡터 $u^{(2)}$ 이다. 그리고 이 두 벡터가 평면을 만든다. 즉, 이차원 평면을 만든다. 이런 평면에 나는 데이터를 위치시킬 것이다. 선형 대수가 친숙한 분들이라면, 올해는 정말로 선형 대수의 전문가들이신데, 이것의 공식적인 정의는 벡터들의 집합 $u^{(1)}, u^{(2)}, \dots, u^k$ 를 찾는다는 것이다.

05'08"

그리고 우리가 할 일은 k 벡터들의 집합이 형성(span)한 이 선형 부분 공간(subspace)에 데이터를 위치시키는 것이다. 하지만 여러분이 선형 대수를 잘 모른다면 데이터를 위치시킬 한 방향 대신 k 방향들을 찾는다고 생각해보라. 따라서 k 차원 평면은 이 경우에 이차원 평면을 찾는 것이었고, 바로 이 그림처럼 말이다. 그리고 우리는 k 방향을 사용하는 평면의 점들의 위치를 정의할 수 있다. 이것이 바로 PCA에서 우리가 데이터를 위치시킬 k 벡터를 찾기를 원하는 이유이다. PCA에서 우리가 하고자 하는 것은 투영 거리를 최소화할 수 있도록 데이터를 위치시키는 방법을 찾는 것이다. 이 투영거리는 점들과 투영 위치 사이의 거리이다. 따라서 이 3차원 예시에서는 바로 이 선이다. 이 점을 찍고 2차원 평면에 나타낸다. 평면은 이미 만들었다. 따라서 투영 오류는 이 점과 이 점이 2차원 평면에 위치될 지점 사이의 거리가 될 것이다. PCA가 하는 일은 선이나 면, 혹은 무엇이든 간에 나의 데이터를 위치시킬 수 있는 것을 찾아서 90도 투영값이나 혹은 직각투영오차(orthogonal projection error)의 제곱을 최소화하는 것이다.

06'20"

마지막으로 내가 자주 듣는 질문은, PCA가 선형 회귀와 어떻게 관련되는가 하는 문제이다. PCA를 설명할 때, 가끔 이렇게 생긴 다이아그램으로 끝나기도 하기 때문에 선형 회귀처럼 보인다. 하지만 PCA는 선형 회귀가 아니다. 곁보기에는 매우 유사하지만 실제로 이 둘은 완전히 다른 종류의 알고리즘이다. 우리가 선형회귀를 하고 있다면, 우리가 할 일은 왼쪽처럼 x 축의 정보를 가지고 변수 y 의 값을 투영하는 것이다. 선형 회귀는, 보이는 것처럼 직선을 그려서 점과 직선간의 오차 제곱을 최소화하는 것이다. 그리고 우리가 최소화하는 것은 이 파란색 선들의 크기이다. 지금 내가 파란 선들을 수직으로 그렸다는 것에 주의하라. 이 파란 선들은 점과 가정에서 투영한 값 사이의 직선 거리이다. 반면, PCA는 이 파란 선들, 사선으로 그려진 선들을 최소화 하려고 한다. 이것이 바로 가장 짧은 직각 거리이다. 즉 x 점과 이 빨간 선 사이의 가장 짧은 거리이다. 이는 데이터 셋에 따라 매우 다른 효과를 가져온다. 일반적으로 선형 회귀를 할 때, 여기 우리가 투영하려고 하는 구별되는 변수 y 가 있다. 모든 선형회귀는 x 값을 가지고 y 값을 구한다. 반면 PCA는 우리가 구하려는 특별한 변수 y 가 없다. 하지만 대신 여러 가지 값이 있다. 즉, x_1, x_2, \dots, x_n 이 있다. 이 값들은 똑같이 취급 받으며, 어느 하나 특별하지는 않다.

08'06"

마지막 예로 내가 3차원 데이터를 가지고 있고 데이터를 3차원에서 2차원으로 감소시키려고 한다면 나는 아마도 내 데이터를 위치시킬 수 있는 2가지 방향, u^1 과 u^2 를 찾고 싶을 것이다. 그러면 나는 3가지 값, x_1, x_2, x_3 를 갖게 된다. 이 값들은 똑같이 취급된다. 세 값은 똑같이 취급 받으며 여기에는 내가 투영하려고 하는 특별한 변수 y 가 없다. 따라서 PCA는 선형 회귀가 아니며 곁으로는 상관관계가 있는 것처럼 보일지라도 이 둘은 실제로는 매우 다른 알고리즘이다.

Q) 여러분이 다음과 같은 데이터 셋으로 PCA 알고리즘을 수행한다고 하자. 다음 중 무슨 값이 데이터를 투영시킬 적당한 $u^{(1)}$ 값이겠는가?

정답 4번

여러분이 이제 PCA가 하는 역할을 이해했기를 바란다. 데이터를 위치시킬 수 있는 더 낮은 차원의 면을 찾아서 오류를 최소화한다. 각 점과 투영 위치 간의 제곱 거리를 최소화한다. 다음 강의에서 데이터를 위치시킬 실제로 낮은 차원의 면을 구하는 법을 알아보자.

No.	Title
Week 8-9	Principal Component Analysis Algorithm

00'00"

강의에서는 principle components analysis 알고리즘에 대해 얘기하겠다. 이번 강의가 끝나면 여러분은 PCA를 여러분 힘으로 실행할 수 있게 될 것이다. 그리고 데이터의 차원을 감소시킬 때 사용할 수 있다. PCA를 적용하기 전에 데이터 전처리 단계가 있는데 이는 여러분이 항상 해야 하는 단계이다. label되지 않은 트레이닝 예시 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ 을 생각해 볼 때 항상 평균 정규화를 수행하는 것은 중요하다. 그리고 데이터에 따라 값을 조정하라. 이것은 지도 학습의 평균 정규화와 값을 조정하는 작업과 굉장히 유사하다. 사실 label되지 않은 데이터, $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ 에 지금 하는 작업만 빼면 사실 똑같은 과정이다.

00'43"

평균 정규화를 하려면 먼저 각각의 표본의 평균을 구한 후 각 $x_j^{(i)}$ 를 $x_j - \mu_j$ 로 바꾼다. 그러면 각 값이 정확하게 평균값 0을 갖도록 만든다. 그리고 두 번째로 각각의 값이 각기 다른 scale을 가진다면, 예를 들어 만약 x_1 이 집의 크기이고 x_2 가 침실 개수라면 이전 예시에서 했던 것처럼 우리는 각각의 값을 비교 가능한 값이 되도록 조정할 수 있다. 지도 학습에서 했던 것처럼 우리는 $x_j^{(i)}$, J값을 가지고 계산한다. 즉, $\frac{x_j^{(i)} - \mu_j}{s_j}$ 가 되게 한다. 여기서 s_j 는 j의 베타값을 측정하는 변수이다. 이것은 최대값 - 최소값이 될 수 있으며 더 일반적으로는 j 값의 표준편차이다. 이 같은 데이터 전처리 과정을 끝냈으면 여기 PCA 알고리즘이 있다.

01'40"

이전 강의에서 우리는 PCA가 하는 일을 보았다. PCA는 데이터를 위치시킬 수 있는 더 낮은 차원의 부분 공간(sub space)을 찾으려고 한다. 따라서 예측 오류 제곱을 최소화시키려면, 예측 오류 제곱의 합을 최소화시키려면 이 파란색 선들의 길이를 제곱해서 벡터, $u^{(1)}$ 을 찾는다. 이 벡터는 이 방향을 향한다. 2차원의 경우에는 벡터 $u^{(1)}$ 과 $u^{(2)}$ 를 찾을 수 있는데 이 두 벡터는 데이터를 위치시킬 이런 모양의 평면을 만든다. 그래서 데이터 평균의 차원을 감소시키는 법을 빨리 되짚어보면, 원쪽을 보면 주어진 예는 $x^i \in R^2$ 이다. 우리가 하려고 하는 것은 우리의 데이터를 대표하는 $Z^{(i)} \in R$ 을 만족시키는 값의 집합을 찾는 것이다. 이것이 바로 2차원에서 1차원으로 감소시킨다는 것의 의미이다.

02'39"

저기 빨간 선에 데이터를 위치시킨다고 하자. 선 위의 점들의 위치를 구체화하는 데는 숫자가 1개만 필요하다. 이것을 Z_1 으로 놓겠다. 여기서 Z 는 실수이고 이것은 1차원 벡터와 같다. 따라서 Z_1 은 1×1 행렬의 첫 번째 구성요소를 뜻하거나 이 1차원 벡터를 뜻한다. 우리는 이 점의 위치를 구체화하는데 숫자 1개만 있으면 된다. 여기 예가 내가 가진 $x^{(1)}$ 이었다면 여기 표시될 것이다. 이 예가 $x^{(2)}$ 였다면 여기 표시될 것이다. 여기 이 점은 $Z^{(1)}$ 이 되고 같은 방식으로 다른 점들, $x^{(3)}, x^{(4)}, x^{(5)}$ 도 여기 $Z^{(1)}, Z^{(2)}, Z^{(3)}$ 으로 표시될 것이다. PCA가 할 일은, 우리가 해야 할 일은 두 가지를 계산할 방법을 찾는 것이다. 한가지는 이 벡터, $u^{(1)}$ 을 계산하는 것이고 이 경우에는 $u^{(1)}$ 과 $u^{(2)}$ 이다. 다른 하나는 이 숫자들, Z 를 어떻게 계산할까 하는 문제이다.

03'50"

왼쪽의 예, 즉 데이터를 2차원에서 1차원으로 줄이는 예이고 여기 오른쪽의 예는 3차원 데이터를 2차원으로 줄이는 것이다. 즉, $x^{(i)} \in R^3$ 에서 $Z^{(i)} \in R^2$ 로 바꾼다. 그래서 이제 Z 값들은 2차원이다. 즉, Z_1, Z_2 이다. 우리는 이 새로운 변수 Z_1, Z_2 를 그 데이터를 이용하여 계산하여야 한다. 그래서 우리는 이 모든 값을 어떻게 계산하는가? 수학적인 유도 방법, 수학적인 증명에 의해 $u^{(1)}, u^{(2)}, Z^{(1)}, Z^{(2)}$ 등의 맞는 값을 구할 수 있지만 이는 매우 복잡하고 이 강의가 다루는 차원을 벗어난다.

04'33"

여러분이 모든 변수 값을 구하면 이 과정이 $u^{(1)}$ 값을 찾기 위함이었다는 것을 알 수 있다. 이는 그렇게 어렵지는 않지만 이 값이 옳은 값이라는 수학적 증명은 내가 다루고자 하는 범위를 벗어난다. 하지만 이 벡터값들, $u^{(1)}, u^{(2)}, Z$ 를 계산하기 위해 여러분이 실행해야 하는 구체적인 과정을 설명해보겠다. 다음과 같다. 데이터를 n 차원에서 k 차원으로 감소시키고 싶다. 우리가 먼저 할 일은 공분산 행렬(covariance matrix)을 계산하는 것이다. covariance matrix는 보통 Greek 대문자 알파벳 시그마로 나타낸다. Greek 알파벳 시그마가 총합을 구하는 기호와 같은 것은 조금 불행한 일이다. 이 Greek 알파벳 시그마를 사용해서 행렬을 표현한다. 여기 이것이 총합을 구하는 기호 시그마이다. 다행스럽게도 여기 슬라이드에서는 matrix sigma와 행렬, 합을 나타내는 시그마 사이의 모호함이 없을 것이다. 다행히도 내가 각각을 사용하는 것은 문맥상 분명히 구별될 것이다. 이 matrix sigma는 어떻게 계산하는가? 예를 들어 우리가 중괄호로 묶인 U,S,V, svd Sigma를 계산한다고 하자. 우리가 할 일은 시그마 행렬의 고유벡터(eigenvalues of the matrix sigma)를 계산하는 것이다. 즉 아래 수식을 사용한다. 여기서 svd는 singular value decomposition의 약자이다. 이것은 훨씬 고급화된 single value composition이다.

06'14"

이것은 여러분이 알아야 하는 수준보다 고급 선형대수이다. 하지만 시그마가 covariance matrix와 같을 때 이 eigenvectors를 계산하는 방법이 있다. 여러분이 만약 선형 대수를 잘 안다면 그리고 이 벡터를 들어본 적이 있다면 여러분은 또 다른 같은 용도로 계산하는데 사용될 수 있는 eigenvector가 있다는 것을 알 것이다. 그리고 SVD 함수와 I 함수 둘 다 똑 같은 벡터값을

내놓는 것으로 판명되었다. 차이가 있다면 SVD 수치적으로 안정적이라는 것이다. 그래서 나는 SVD를 사용하는 편이다. 하지만 내 친구들은 I 함수를 사용해서 똑 같은 작업을 한다. covariance matrix sigma에 적용시키면 똑같은 값이 도출될 것이다. 이것은 covariance matrix이 갖는 수학적인 특성, symmetric positive definite을 충족시키기 때문이다. 여러분은 이것의 의미를 알 필요는 없는데 SVD함수와 I 함수가 다른 함수지만 covariance matrix에 적용되면 이 수학적인 특성을 언제나 충족시키고 둘 다 같은 값을 도출한다는 사실을 숙지하면 된다.

07'18"

이 부분은 여러분이 알 필요가 있는 범위를 넘어선 선형대수에 가까웠다. 여기까지 하나도 이해되지 않았다고 해서 걱정할 필요는 없다. 여러분이 알아야 할 것은 바로 이 수식을 함수에 넣어야 한다는 것이다. 만약 여러분이 Octave 또는 MATLAB이 아니라 다른 언어로 실행하려 한다면 여러분은 SVD 또는 singular value decomposition을 계산할 수 있는 numerical linear algebra library를 찾아야 한다. 거의 모든 주요 프로그래밍 언어에는 이런 library가 굉장히 많다. 사람들은 이 library를 사용해서 covariance matrix sigma의 SVD를 계산한다. 좀 더 자세히 알아보면, 이 covariance matrix sigma는 $n \times n$ 행렬이다. 이것을 알아보는 한 가지 방법은, 정의를 보면 이 이 값은 $n \times 1$ 벡터이다. 그리고 이 값은 $1 \times n$ 이다. 따라서 이 부분은 $n \times n$ 행렬이 될 것이다. $1 \times N$ 을 전이시키고 $1 \times n$ 을 전이시켜서 $n \times n$ 행렬이 된다. 더해도 여전히 $n \times n$ 행렬이 된다.

08'34"

그리고 svd 결과로 U,S,V 세 개의 행렬이 출력된다. 이 외에 우리가 정말 필요한 것은 u 행렬이다. 이 행렬 역시 $n \times n$ 행렬이다. u 행렬의 열을 보면 바로 이 벡터들, $u^{(1)}, u^{(2)}, \dots, u^{(n)}$ 로 이루어져 있다. 따라서 u는 $n \times n$ 행렬이다. 우리가 데이터를 n 차원에서 k 차원으로 감소시키려고 한다면 먼저 이 행렬에서 k 벡터들을 꺼내고 이는 $u^{(1)}, \dots, u^{(k)}$ 를 줄 것이다. 이는 우리가 데이터를 위치시키고자 하는 k 방향들을 줄 것이다.

09'22"

SVD numerical linear algebra routine의 나머지 작업으로부터 우리는 U 행렬을 얻는다. 그리고 이 열들을 $u^{(1)}, \dots, u^{(n)}$ 으로 둔다. 나머지 과정을 짧게 정리해보면 먼저 SVD numerical linear algebra routine에서 우리는 이 u, s, d행렬을 얻는다. 우리는 이 행렬의 K개 열을 사용해서 $u^{(1)}, \dots, u^{(k)}$ 을 얻는다. 그리고 해야 하는 다른 작업은 원래의 $x \in R^n$ 를 낮은 차원의 $Z \in R^k$ 로 바꾸는 것이다. 따라서 U행렬에서 K 행을 꺼내서 이 행렬을 구성한다.

10'12"

이 행렬의 열에 $u^{(1)}, \dots, u^{(k)}$ 를 쓰는다. 알다시피 이 행렬에서 이 부분을 꺼낸다. 즉, K열의 첫 번째 부분이다. 이것은 $n \times k$ 행렬이 될 것이다. 나는 이 행렬에 이름을 붙이려고 한다. 즉 U_{reduce} 행렬이라 부를 것이다. 일종의 U행렬의 간소화 버전이라고 할 수 있다. 나는 이 행렬을 사용해서 내가 가진 데이터의 차원을 감소시킬 것이다. 그리고 내가 Z를 계산하는 방법은 위의

수식과 같다. 즉 $U_{reduce}^T * X$ 인데, 이는 U_{reduce} 를 transpose해서, 즉 해당 행렬의 행과 열을 바꿔서 X 를 곱해 Z 를 도출한다.

11'19"

이 차원들이 말이 되는지 확실히 하기 위해 여기 행렬은 $k \times n$ 이 될 것이다. 그리고 여기 X 는 $n \times 1$ 이 될 것이다. 따라서 이 값은 $k \times 1$ 이 될 것이다. 그리고 Z 는 우리가 바란 것처럼 k 차원의 벡터가 된다. 그리고 여기 보이는 오른쪽 X 들은 우리의 트레이닝 셋이나 우리의 교차검증 세트, 또는 테스트 셋의 예가 될 수 있다. 예를 들어서 내가 트레이닝 예 i 를 쓰고 싶다고 한다면 x_i 를 X^i 로 놓은 다음 이를 이용하여 $Z^{(i)}$ 를 얻을 수 있다.

11'55"

요약하자면 여기 PCA 알고리즘을 슬라이드 한 장에 정리했다. 평균 정규화를 한 다음에 모든 값이 평균값 0을 갖도록 해주고 값을 보고 서로 다른 범위 값을 가져서 조정이 필요하다면 추가로 조정할 수도 있다. 이 전처리 단계를 마친 후 공분산행렬 (covariance matrix Sigma)를 계산한다. 만약 여러분에게 주어진 행렬이 다음과 같다면 데이터는 다음과 같은 행들로 주어진다. 만약 시계열 자료들이 행으로 주어진 행렬 X 가 있고, X^T transpose가 다음과 같다면, 이의 covariance matrix sigma는 굉장히 벡터화가 잘 된 것이다. 여러분은 octave를 실행할 수 있다. 여러분은 다음과 같은 수식을 실행할 수 있는데 이 수식은 공분산행렬을 얻는 방법에 대한 행렬 표현식이다. 오늘 이 식을 증명하지는 않겠다. 이것이 올바른 벡터를 도출했는지 여러분이 스스로 수학적인 확인법, octave를 써볼 수 있다. 이 값과 이 값이 똑같은 해를 도출하는지를 본다. 또는 스스로 수학적으로 증명해 본다. 어떤 방법이든 이 것이 올바르게 벡터값을 얻었다는 것을 보여줄 것이다. 이제 망설임 없이 우리는 SVD routine을 사용해서 U , S , D 값을 얻을 수 있다. 그리고 우리는 감소시킨 U 행렬에서 첫 k 행을 쓰고, 여기 X 값에서 감소된 차원의 Z 값을 얻는다.

13'38"

k Means 알고리즘과 유사하게 PCA 알고리즘을 실행하면 $x \in R^n$ 을 사용하지 $x_0 = 1$ 을 사용하지 않을 것이다. 이것이 바로 PCA 알고리즘이었다. 내가 한 가지 하지 않은 것은 이 과정이 k 차원의 subspace에서 k 차원의 면으로 데이터를 위치시키는데 대한 수학적 증명을 하지 않았다. 이는 예측 오류 제곱을 최소화한다. 왜냐면 이를 수학적으로 증명하는 것은 본 강의가 다루는 범위에서 벗어나기 때문이다. 다행히 PCA 알고리즘은 코드를 많이 사용하지 않고도 실행할 수 있다. 그리고 여러분이 이것을 octave 알고리즘으로 실행한다면 굉장히 효과적인 차원 감소 알고리즘을 얻을 것이다.

Q) PCA에서 우리는 $x \in R^n$ 에서 다음과 같은 $z \in R^k$ 를 얻었다. 다음 중 z_j 값으로 올바른 것은?
정답 4번

14'27"

이것이 바로 PCA 알고리즘이었다. 내가 하지 않았던 한 가지는 $u^{(1)}, u^{(2)}, Z$ 에 대한 수학적인

증명이다. 이 예측 오류 제곱을 최소화할 증명을 하는 것은 전적으로 여러분에게 달렸다. 이제 배운 것을 상기해 보자. PCA가 하려는 것은 예측 오류 제곱을 최소화할 수 있는 방식으로 데이터를 위치시킬 면이나 선을 찾는 것이었다. 그래서 나는 이 부분, 저 부분을 수학적으로 증명하지 않았는데 이는 본 강의를 벗어나는 것이기 때문이었다. 다행인 점은 PCA 알고리즘은 octave 상에서 간단한 코드로 실행할 수 있다는 것이다. 이걸 실행하든 이걸 실행하든 혹은 이 알고리즘을 실행하든 모두 다 잘 작동할 것이지만 이 알고리즘을 사용하면 굉장히 효과적인 차원 감소 알고리즘을 얻게 된다. 이는 예측 오류 제곱(square projection error)을 최소화하는데 제 역할을 한다.

No.	Title
Week 8-10	Reconstruction from Compressed Representation

00'00"

이전 강의에서 압축 알고리즘으로서의 PCA에 대해, 예를 들어, 1,000 차원의 데이터를 100 차원의 벡터로 압축하는 법을 살펴보았다. 또 3차원의 데이터를 2차원으로 압축하는 법을 알아보았다. 따라서 이것이 압축 알고리즘이라면 이 압축된 차원에서 원래의 고차원 데이터로 돌아가는 방법도 있을 것이다. 예를 들어 주어진 값 $Z^{(i)}$ 가 100 차원이라면, 예를 들어, 원래 1,000 차원이었던 $x^{(i)}$ 로 어떻게 돌아갈 수 있을까? 이번 강의에서는 돌아가는 방법을 알아보자.

00'40"

PCA 알고리즘에서 우리는 다음과 같은 예를 가질 수 있다. 이 점을 $x^{(1)}$ 으로 놓고 이 점을 $x^{(2)}$ 로 놓자. 우리가 할 일은 이 점들을 이 1차원에 표시하는 것이다. 그리고 이 점들을 1차원에 표시한 다음에 실수인 Z_1 을 사용하여 이 점들의 위치를 구체화한다. 여기 이 점을 $Z^{(1)}$ 이라 하면 여기서 어떻게 다시 원래의 이차원 공간으로 돌아갈 수 있을까? 특히, $Z \in \mathbb{R}$ 을 다시 $X \in \mathbb{R}^2$ 로 돌릴 수 있을까? 그러니까 지금 쓰고 있는 공식과 같을 경우, 만약 반대 방향으로 가기를 원한다면 이런 공식에서는 다음과 같은 공식을 쓸 것이다. 다시 한번 차원을 확인하자. 여기 U_{reduce} 는 $n \times 1$ 차원 벡터이고 Z 는 $k \times 1$ 차원 벡터이다.

이 둘을 곱하면 $n \times 1$ 차원 벡터가 된다. 따라서 X_{approx} 는 n 차원 벡터가 될 것이다. 따라서 PCA의 의도와 같이, 만약 투영 오류 제곱이 크지 않다면 이 X_{approx} 는 원래 값이 무엇이었던 간에 원래 X 값에 가깝게 될 것이다. 애초에 이 X 값은 Z 를 유도하기 위해 사용했었다.

02'17"

그래프로 나타내면 어떻게 될 것인지 보자. 바로 다음과 같다. 여러분이 다시 이 과정을 반복하면, 즉 여기 녹색 선위에 놓인 점들에서 시작한다. 처음 들었던 예로 돌아가서 $x^{(1)}$ 값에서

시작하면 이 $Z^{(1)}$ 값을 얻을 수 있고 이 공식에 $Z^{(1)}$ 값을 대입해서 $X_{approx}^{(1)}$ 을 얻는다. 바로 여기 이 점이 $X_{approx}^{(1)}$ 이 될 것이고 $X_{approx}^{(1)} \in R^2$ 가 될 것이다. 그리고 같은 과정을 반복하면 이 점이 $X_{approx}^{(2)}$ 가 될 것이다. 이는 원래 데이터 값과 상당히 유사할 것이다.

02'54"

바로 이렇게 해서 낮은 차원의 Z에서 압축되기 전의 데이터 차원으로 돌아갈 수 있다. 원래 데이터 X에 가까운 값을 얻었다. 우리는 이것을 원본 데이터의 재건축 과정이라 부른다. 압축된 값에서 원래 X값의 복구를 시도하는 과정이다.

Q) k=n으로 두고 PCA 알고리즘을 실행한다고 하자. 따라서 데이터 차원은 감소되지 않았다. 보존된 분산이 다음과 같다면 $\frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^n s_{ii}}$, 다음 중 어느 것이 참인가?

정답 1, 2, 3번

이제 여러분은 label되지 않은 데이터 셋에 PCA를 어떻게 적용해서 고차원 값 X를 얻고 이를 어떻게 보다 낮은 차원 평면 Z에 나타내는지 알 것이다. 그리고 본 강의를 통해서 여러분은 이 낮은 차원의 Z를 원래의 고차원 데이터로 돌리는 법도 배웠다. 이제 PCA를 적용하고 실행하는 방법을 배웠으므로 다음에는 실제 PCA를 활용법의 메커니즘을 알아보자. 다음 강의에서는 k값 선택법, 그리고 벡터 z의 차원을 얼마까지 감소시킬지 선택하는 방법에 대해 알아보자.

No.	Title
Week 8-11	Choosing the Number of Principal Components

00'00"

PCA 알고리즘에서 우리는 N차원 값들을 K차원 값으로 감소시킨다. 이 숫자 K는 PCA 알고리즘의 매개변수이다. 숫자 K는 principle components의 수 또는 우리가 보존하는 principle components의 수라고 불린다. 이번 강의에서 가이드라인을 제시해서 사람들이 PCA를 위한 매개변수 K를 어떻게 선택하는지 알려주겠다. K를 선택하기 위해서는, 그러니까 principal component의 수를 선택한다는 것인데, 여기 몇 가지 유용한 개념이 있다. PCA 알고리즘은 투영 오류 제곱 평균을 최소화 하려고 한다. 그래서 지금 내가 쓰는 이 값을 최소화하려고 한다. 이 값은 지난 강의에서 다뤘었던 원래 데이터 X와 투영 값인 $X_{approx}^{(1)}$ 간의 거리이다. 즉, 더 낮은 차원의 면의 표시값과 X값 사이의 거리 제곱을 최소화하려고 한다. 이것이 투영 오류 제곱 평균이다.

01'04"

그리고 total variation in the data를 정의해 보겠다. 이는 $\frac{1}{m} \sum_{i=1}^m \|X^{(i)}\|^2$ 이다. 따라서 total variation in the data는 내가 가진 트레이닝 예시들 각각의 길이의 평균이다. 이를 보고 누군가는 “평균적으로 내가 가진 트레이닝 예들은 이 벡터로부터, 0벡터로부터 얼마나 멀리 떨어져 있는가?”라는 질문을 할 수도 있다. 나의 트레이닝 예들은 원점에서 평균적으로 얼마나 멀리 떨어져 있는가? K를 선택하려고 할 때, 가장 보편적인 방법은 더 작은 값들을 선택해서 이 비율이 0.01과 같거나 적도록 만드는 것이다. 달리 말하면 K를 선택하는 보편적인 방식은 투영 오류 제곱 평균을 생각하는 것이다. 즉, x 값과 그 투영 값 사이의 평균 거리를 데이터의 총변동으로 나눈 것이다. 이것이 데이터의 변화 값이다. 우리는 이 비율이, 예를 들어, 0.01보다 작기를 원한다고 하자. 혹은 또 다른 접근법으로서, 1%보다 작기를 원한다고 하자. 그리고 대다수 사람들은 k를 선택할 때 k를 직접 택하지 않고 보통 사람들이 얘기하는 방식은, 이 숫자가 무엇이든 간에, 즉 0.01이거나 다른 숫자이건 간에, 그리고 만약 k값이 0.01이라면 이를 달리 표현하면, PCA 언어를 사용하면, “변수의 99%가 보존된다”이다.

02'31"

나는 이 표현이 기술적으로 무엇을 의미하는지는 신경 쓰고 싶지 않다. 하지만 “변수의 99%가 보존된다”는 표현은 여기 왼쪽 값이 0.01보다 작다는 뜻이다. 따라서 여러분이 PCA를 사용하고 있다면 그리고 누군가에 여러분이 principle component를 얼마나 많이 보존하고 있는지 말하고 싶다면 이렇게 하는 것이 자연스러울 것이다. “나는 k를 선택해서 99%의 변수가 보존되었어.” 이것을 알고 있는 편이 좋다. 왜냐면 이는 투영 오류 제곱 평균이 기껏해야 1%인 총변동으로 나눠졌다는 뜻이기 때문이다.

03'06"

이는 여러분에게 혜안을 주는 표현이다. 여러분이 “나는 principle component 를 100개 가지고 있어” 또는 “1,000차원에서 k=100이야”라고 하면 다른 이들이 이를 해석하기 힘들 것이다. 이것이 사람들이 숫자 0.01를 자주 사용하는 이유이다. 자주 사용되는 다른 값은 0.05인데 이 값은 5%로 나타낼 수 있고 따라서 95%의 변수를 보존한다고 말할 수 있다. 같은 방식으로 90%, 85%로 보존한다고 말할 수 있다. 따라서 90%는 0.10에 대응되고 이것은 10%이다. 따라서 변수 값으로는 보편적인 범위의 값은 90, 95, 99, 어쩌면 85%까지 낮아질 수 있다. 따라서 95에서 99까지는 우리들이 가장 보편적으로 사용하는 범위 값이라 할 수 있다.

04'04"

놀랄 수도 있지만 많은 데이터 셋에서 여러분은 변수의 99%를 보존할 수 있다. 데이터의 차원을 크게 감소시키고도 많은 변수들을 보존할 수 있다. 실생활 데이터의 대부분에서 변수 값들의 상관관계가 높기 때문에 데이터를 크게 압축하고 나서도 99%나 95% 가량의 변수를 보존할 수 있다.

04'27"

그러면 실행 방법은 무엇인가? 여기 우리가 사용할 알고리즘이 하나 있다. K값을 선택하려면 먼저 $k=1$ 에서 시작한다. 그리고 PCA를 실행한다. 그리고 U_{reduce} , $Z^{(1)}$, $Z^{(2)}$ $Z^{(m)}$ 을 계산한다. 그리고 $X_{approx}^{(1)}$ $X_{approx}^{(m)}$ 도 계산한다. 그리고 나서 변수 값의 99%가 보존되었는지 확인한다. 이대로 잘 도출된다면 $k=1$ 을 사용한다. 하지만 그렇지 않는 경우라면 다음으로 $k=2$ 를 써본다. 그리고 잘 도출될 때까지 다시 전체 과정과 확인을 반복한다. 즉, 이 표현식이 만족될 때까지 반복하는 것이다. 0.01보다 작게 나오지 않으면 다시 반복해야 한다. $k=3$ 으로 두고 안되면 $k=4$ 로 놓고 계속해서 $k=17$ 이 되어서 우리가 데이터의 99%가 보존될 때까지 반복한다. 이렇게 되면 $k=17$ 을 사용한다. 이것이 변수의 99%를 보존하는 가장 작은 k값을 구하는 한 가지 방법이다.

05'24"

하지만 여러분이 보시다시피 $k=1$, $k=2$ 하는 식으로 이 모든 계산을 반복하는 것은 굉장히 비효율적이다. 다행히도 우리는 이 과정에서 PCA를 실행해서 이 모든 값을 좀 더 쉽게 계산할 수 있다. 구체적으로, 여러분이 이 u , s , 그리고 d행렬을 구하기 위해 svd (Sigma)를 사용할 때, 공분산행렬(covariance matrix sigma)의 SVD (Sigma)를 사용하면, S 행렬을 얻을 수 있고, S 는 $n \times n$ 정사각행렬이 될 것이다. 그리고 대각행렬이다. 즉 행렬의 대각선 성분은 $S_{11}, S_{22}, S_{33} \dots S_{nn}$ 이다. 그리고 이 성분들은 이 행렬에서 유일하게 값이 0이 아닌 성분들이다. 즉 행렬의 나머지 부분 성분 값은 0인 것이다.

06'13"

그러니까 여기 이 커다란 동그라미들은, 이 행렬의 이 대각선 부분의 성분들은 0의 값을 가질 것이다. 보여주는 것은 가능하지만 여기서 증명하지는 않을 것이다. 또한 바로, 주어진 k값, 즉 여기 이 값을 훨씬 쉽게 계산할 수 있는 법을 보일 것이다. 즉, k값은 다음과 같이 계산될 수 있다. $1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}}$ 이다. 다르게 표현하자면, 아니면 이를 설명할 수 있는 다른 관점에서 보면, $k=3$ 이라고 하자. 그리고 우리는 분자의 합을 계산할 텐데, S_{ii} 에서 $i=1 \sim 3$ 을 합하는 것이고 이 행렬에서 보면 S_{11}, S_{22}, S_{33} 성분의 합인 것이다.

07'11"

이게 분수이고, 분모는 이 모든 대각선 성분의 합이다. 그리고 $1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}}$ 를 계산하면 파란색으로 표시한, 왼쪽의 이 공식을 계산한 값이 나온다. 그래서 우리가 할 수 있는 일은 이 값이 0.01과 같거나 작은지 테스트하는 것이다. 또는 $\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$ 를 계산하면 된다. 변수의 99%가 보존되었는지 확인하고 싶다면 말이다. 그리고 우리는 서서히 k 를 증가시킨다. 즉, $k=1$, $k=2$, $k=3$ 으로 계속 두고 테스트해서 변수를 99%까지 보존하게 해주는 가장 작은 k값을 얻을 때까지 반복한다. 이 과정을 반복하려면 여러분은 SVD 함수를 한 번만 사용하면 된다. 이 함수가 S 행렬을 주기 때문에 이 행렬을 갖게 되면 변수의 K값을 증가시키면서 계산을 계속하면 되기 때문에 다른 K값을 테스트하기 위해 SVD 함수를 반복해서 사용할 필요가 없다. 따라서 이 방법이 훨씬 효율적일 뿐만 아니라 PCA를 여러 번 반복해서 굵어울 필요 없이 k값을 택하도록

해준다. SVD를 한번 실행하기만 하면 모든 대각선 성분들, $S_{11}, S_{22}, S_{33}, \dots, S_{nn}$ 을 얻을 수 있다. 그리고 여기 k값을 변화시켜서 99%의 변수를 보존할 수 있는 가장 작은 k값을 얻을 수 있다.

08'48"

요약하면, 압축을 위해 PCA를 실행할 때, 내가 k값을 선택할 때 자주 사용하는 방법은 covariance matrix에서 SVD를 한 번 사용하는 것이다. 그리고 이 공식, $\frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^m s_{ii}} \geq 0.99$ 을 사용해서 이 공식을 만족시키는 가장 작은 k를 사용한다. 그리고 다른 k값을 뽑는다면, k값을 직접 뽑는다면, 그러니까 여러분이 만약 1,000 차원의 데이터를 가지고 있고 이 중에서 k=100인 k를 선택하고 싶다고 하자. 그리고 여러분이 다른 사람에게 여러분이 한 일을 설명하고 싶다면 PCA 실행의 성과를 잘 설명하는 한 가지 방법은 $\frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^m s_{ii}}$ 를 계산해서 몇 퍼센트의 변수가 보존되었는지 설명하는 것이다. 이 숫자를 얘기할 때, PCA에 익숙한 사람이라면 여러분의 100 차원 값이 원래 데이터 셋에 얼마나 가까운지 이해할 수 있을 것이다. 왜냐면 99%의 변수가 보존되었기 때문이다.

09'46"

이것이 바로 여러분의 구성 오류 제곱을 계산하는 방식이다. 이 비율이 0.01이라면 사람들은 여러분이 PCA를 잘 실행해서 원래 데이터 셋에 가까운지 본능적으로 잘 알 수 있다. 따라서 이를 여러분이 k를 잘 택하는 방법으로 활용하기를 바란다.

Q) 이전에 PCA는 투영 오차를 최소화하도록 데이터가 투영되는 $u^{(1)}$ 방향을 선택한다고 했었다. 이를 다르게 표현하면? 정답 3번

여러분의 데이터를 어떤 차원으로 감소시킬 것인지 선택하는 것에 대해 얘기해보면, 예를 들어 여러분이 PCA를 고차원 데이터 셋에 적용한다고 하면, 예를 들어 1,000차원에 데이터에 적용하면, 데이터 셋이 서로 높은 상관 관계 값을 가지게 되어서, 대부분의 데이터 셋이 이에 해당할 것인데, 여러분은 PCA가 대부분의 경우 변수의 99%를 보존하게 만들어 준다는 것을, 99%에서 95%의 범위로, 여러분이 데이터를 굉장히 많이 압축해도 변수의 대부분을 보존할 수 있다.

No.	Title
Week 8-12	Advice for Applying PCA

00'00"

이전 강의에서 PCA가 학습 알고리즘의 속도를 높일 수 있다고 얘기했다. 이번 강의에서는 어떻게 이것이 가능한지 그리고 PCA에 어떻게 적용할지에 대해 얘기하겠다. 여기 우리가 PCA를 사용해서 학습 알고리즘의 속도를 높일 수 있는 방법이 있다. 이 지도 학습 알고리즘의 속도가 높아지는 것은 내가 개인적으로 PCA를 가장 흔하게 사용하는 방법이다. 가령 여러분이 지도 학습 문제를 가지고 있다고 하자. 여기 값이 매겨진 x, y 값이 있다. 그리고 여러분의 예시 $x^{(i)}$ 가 굉장히 고차원적이라고 하자. 이렇게 설정하자. $x^{(i)} \in R^{10,000}$. 한 가지 예는, 여러분이 computer vision 문제를 풀고 있다고 하자. 여러분은 100x100 이미지를 가지고 있고 이는 10,000 픽셀이다. 그러면 $x^{(i)}$ 는 여러분의 10,000 픽셀의 강도를 포함한 벡터 값이다. 여러분은 10,000 차원의 벡터 값을 가지게 된다.

01'16"

따라서 이렇게 매우 고차원의 벡터 값을 가지고 학습 알고리즘을 수행하면 속도가 느릴 수 있다. 여러분이 만약 로지스틱 회귀에 10,000 차원의 벡터 값을 적용하거나 벡터 기계를 지원하는 새로운 네트워크에 적용하면, 데이터 값이 너무 커서, 무려 10,000이다, 여러분의 학습 알고리즘을 느리게 만들 수 있다. 다행히 PCA를 사용해서 우리는 이 데이터의 차원을 감소시켜서 알고리즘이 더 효율적으로 실행되게 만들 수 있다.

01'43"

방법을 알아보자. 우리가 할 일은 먼저 lable된 트레이닝 셋을 확인하고 입력 값을 추출하는 것이다. 여기서는 잠깐 y 값을 제쳐두고 x 값들을 추출한다. 그러면 lable되지 않은 트레이닝 셋인 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ 을 가지게 된다. 그리고 10,000 차원의 데이터, 10,000개의 예시를 가지고 있다. 즉, $x^{(1)}, x^{(2)}, \dots, x^{(m)} \in R^{10,000}$ 이다. 그러니까 입력 값 벡터 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ 을 추출한다. 그리고 PCA를 실행하면 감소된 차원의 데이터 값을 얻을 수 있다. 따라서 10,000 차원의 데이터 대신 이제는 1,000 차원의 벡터 값을 가질 수 있을 것이다. 따라서 이것은 10배로 데이터를 절약한 셈이다.

02'25"

이는 새로운 트레이닝 셋을 준다. 방금 내가 사용한 예 $x^{(1)}, y^{(1)}$ 은, 내가 처음 사용한 입력값은, 이제 $z^{(1)}$ 으로 표시된다. 그래서 우리는 이제 새로운 트레이닝 예시를 갖게 되었는데 $z^{(1)}, y^{(1)}$ 의 짝이다. 똑같이 $z^{(2)}, y^{(2)}, \dots, z^{(m)}, y^{(m)}$ 이다. 이 트레이닝 예시들이 이제 훨씬 낮은 차원 값인 $z^{(1)}, z^{(2)}, \dots, z^{(m)}$ 으로 표시된다. 마지막으로 나는 이 차원이 감소된 트레이닝 예시들을 학습 알고리즘인 신경망 네트워크, 로지스틱 회귀 등에 적용해서 H가정을 배워서 이 입력값, 낮은 차원의 z 값들을 이용해서 투영할 수 있다. 그래서 만약 내가 로지스틱 회귀를 사용한다면, $h_{\theta}^z = \frac{1}{1+e^{-\theta^T z}}$ 이다. 여기서 Z 벡터 값을 입력 값으로 사용해서 투영한다. 마지막으로 새로운 예가 있다면, 예를 들어, 새로운 테스트 샘플 X 가 있다고 하자. 이 테스트 샘플 X 를 가지고 PCA로 같은 방식으로 대응시켜서 이에 대응하는 Z 를 찾는다. 그러면 Z 는 이 가정에 들어맞는다. 그리고 이

가정은 여러분의 입력값 X 에 대한 투영을 할 것이다.

03'50"

마지막으로 PCA가 하는 일은 X 에서 Z 까지 맵핑을 정의하는 것인데 이는 트레이닝 셋을 가지고 PCA를 실행함으로써 정의되어야만 한다. 특히 이 맵핑은 PCA가 학습하는데, 이 맵핑은, 매개 변수들의 셋을 계산한다. 즉 값을 조정하고 평균 정규화 과정을 한다. 그리고 U_{reduce} 행렬 또 한 계산한다. 하지만 이 모든 것들은, 즉 U_{reduce} 행렬은 PCA가 학습한 매개변수와 같다. 그리고 우리는 우리의 매개변수들을 우리가 가진 트레이닝 셋에만 적용해야지 cross validation이나 다른 테스트 셋에 적용해서는 안 된다. 따라서 U_{reduce} 행렬은 같은 것들은 여러분이 가진 트레이닝 셋으로 PCA를 수행해서 얻어야 한다. 그리고 U_{reduce} 를 얻었으면, 또는 평균 정규화를 수행하거나 값을 조정해서 이 값을 비교 가능하게 만든다. 트레이닝 셋의 이 모든 매개 변수들을 똑 같은 맵핑이나 다른 예시에 적용할 수 있는데 다른 예시란 cross-validation 셋이나 여러분의 테스트 셋이다.

05'00"

요약하면 여러분이 PCA를 실행할 때, cross-validation 셋이나 여러분이 가진 데이터의 테스트 셋 부분이 아니라 트레이닝 셋 부분에서만 실행해야 한다. 이것은 x 에서 z 값의 맵핑을 정의하고 여러분은 이 맵핑을 교차검증 셋이나 여러분의 테스트 셋에 적용할 수 있다. 이 예에서 나는 데이터를 10,000차원에서 1,000 차원으로 데이터를 감소시키는 법에 대해 얘기했는데 이는 비현실적인 수치는 아니다. 많은 문제에 있어서 우리는 실제로 데이터의 차원을, 5x나 10x로 감소시킨다. 그리고도 여전히 많은 변수들을 보존할 수 있고 성능에도 거의 영향을 주지 않는다. 즉 분류 정확도 차원에서 보면, 학습 알고리즘의 분류 정확도에 거의 영향을 미치지 못한다. 그리고 낮은 차원의 데이터로 학습 알고리즘을 수행하면 속도가 훨씬 더 빨라진다.

05'50"

요약하면, 우리는 지금껏 다음과 같은 PCA의 활용법에 대해 얘기했다. 한 가지는 압축하는데 적용하는 것이다. 즉, 데이터를 저장할 수 있는 메모리나 디스크 공간을 축소시킬 수 있다. 그리고 또 한 가지는 학습 알고리즘의 속도를 높이는데 사용하는 것이다. 이 활용법들에서 k 를 선택하기 위해서, 보존되는 변수의 비율을 찾기 위해서 이렇게 한다. 이 활용법, 즉, 학습 알고리즘의 속도를 높여도 99% 변수를 보존하게 된다. 이것이 k 값을 고르는 보편적인 방법이다. 이 압축 활용을 위해서 여러분의 k 를 택하는 방법이다. 반면 시각화 활용법을 살펴보면, 우리는 대체로 2차원이나 3차원 데이터를 표현하는 법을 알지만 시각화 활용법에서는 $k=2$, $k=3$ 을 선택한다. 주로 2차원이나 3차원 데이터 셋만을 그릴 수 있기 때문이다.

06'58"

PCA의 주요 활용법을 이 여러 가지 활용법을 위한 k 값을 택하는 방법과 함께 요약하면, PCA를 잘못 사용하기 쉬운 예를 언급해야겠다. 여러분은 가끔 다른 사람들이 이런 실수를 하는 것을 들을 것이고 이런 경우가 많지 않기를 바란다. 내가 이것을 언급하려는 이유는 여러분이 같은

실수를 하지 않도록 하기 위함이다. 이것이, PCA를 과적합(over-fitting)을 막기 위해 사용한 것이 바로 PCA를 잘못 사용한 예이다. 하지만 여기에는 이유가 있다. 이것은 PCA를 잘 사용한 것은 아니지만 이유가 있다. 즉, 만약 우리가 $x^{(i)}$ 를 가지고 있고 n 값들을 가지고 있는데 데이터를 압축하려고 대신 $z^{(i)}$ 를 사용한다. 그리고 값을 훨씬 낮은 차원일 수 있는 k 로 감소시킨다. 그 결과 훨씬 작은 숫자의 값을 가지게 되면, 예를 들어 k 가 1,000이고 n 이 10,000이면 우리는 1,000 차원 데이터를 갖게 되고 10,000 차원 데이터를 사용할 때 보다 과적합할 확률이 낮아지게 된다. 일부는 과적합을 막기 위해 PCA를 사용한다고 생각한다. 하지만 이 기능만을 강조하는 것은 PCA를 잘못 활용하는 것이고 나는 이것을 추천하지 않는다. 그렇다고 이 활용법이 나쁜 것은 아니다. 이 방법을 사용해서 데이터의 차원을 감소시켜 과적합을 막고 싶다면 아마도 제대로 잘 작동할 것이다.

08'10"

하지만 이것은 과적합을 해결하는 좋은 방법이 아니다. 대신 과적합이 걱정된다면 이를 해결할 훨씬 좋은 방법이 있다. PCA를 사용하는 대신 정규화(regularization)를 해서 데이터의 차원을 감소시키는 것이다. 그 이유는 PCA가 작동하는 방식을 생각해보면, PCA가 label된 y 를 사용하지 않기 때문이다. 여러분이 입력값 $x^{(i)}$ 를 보고 있다면 그리고 이를 사용해 데이터의 더 낮은 차원을 찾으려고 한다. 이 때 PCA가 하는 일은, 일부 정보를 던져버리는 것이다. y 값이 무엇인지 모르고서 데이터의 차원을 감소시키거나 던져버린다. 따라서 PCA를 사용하는 것이 괜찮을 수도 있지만, 예를 들어 변수의 99%가 보존된다면 괜찮지만, 여러분이 데이터의 대부분을 보존하고 싶어도 일부 중요한 정보를 날릴 수 있다. 결국 여러분이 변수의 99%나 95%나 비슷하게 보존하고 싶다면 정규화를 사용하는 것이 과적합을 피하는 좋은 방법이 될 것이다. 그리고 정규화가 더 잘 작동하는 이유는 여러분이 선형 회귀나 로지스틱 회귀, 또는 다른 방법에 정규화를 실시할 때, 이 최소화 문제가 y 값이 무엇인지 알기 때문에 일부 중요한 정보를 버릴 확률이 낮다. 반면 PCA는 label된 값을 사용하지 않기 때문에 중요한 정보를 버릴 확률이 높다. 요약하면, 만약 여러분이 주 목표가 학습 알고리즘 속도를 높이는 것이라면 PCA를 사용하는 것이 좋다. 하지만 과적화를 피하기 위해서라면 PCA를 사용하는 것이 그다지 좋지 않다. PCA 대신 정규화를 사용하는 것이 많은 사람들이 추천할 방식이다.

09'55

끝으로 PCA의 마지막 남은 잘못된 활용법을 보자. PCA는 매우 유용한 알고리즘이라 할 수 있으며 나는 시각화 목적으로 압축하는데 자주 사용한다. 하지만 나는 가끔 사람들이 PCA를 사용해서는 안될 때 사용하는 것을 본다. 여기 내가 자주 보는 실수가 있다. 누군가가 머신 러닝 시스템을 고안하고 있는데 그들은 계획을 다음과 같이 적을 수 있다. 그러니까 학습 시스템을 만들어 보자. 이렇게 생긴 트레이닝 셋을 가지고, 우리는 PCA를 수행할 것이다. 그리고 로지스틱 회귀를 훈련한다. 그리고 데이터 셋을 테스트한다. 종종, 프로젝트의 출발은 누군가가 프로젝트 계획을 다음과 같이 적고 PCA를 사용해서 이 네 단계를 수행하자고 말하는 것이다.

10'36"

이처럼 PCA를 포함하는 프로젝트 계획을 쓰기 전에, 한 가지 좋은 질문은, 우리가 PCA를 사용하지 않고 전체를 해야 한다면 어떻게 하겠는가 하는 것이다. 사람들은 종종 복잡한 계획안을 구상하고 PCA를 실행하는 등의 단계 이전에 이 단계를 생각하지 않는다. 그리고 가끔, 구체적으로 얘기하면, 나는 사람들에게 PCA를 수행하기 전에 내가 우선 추천하는 것은 그들이 가진 원래의 가공되지 않은 데이터 $x^{(i)}$ 를 가지고 할 수 있는 것을 다 해보라고 조언한다. 원하는 결과가 나오지 않으면 그때 PCA를 실행하고 $z^{(i)}$ 를 사용하는 것을 고려해보는 것이 좋다.

11'18"

따라서 PCA를 쓰기 전에, 데이터의 차원을 감소시키는 대신, 나는 우선 이 PCA 단계를 버리는 것을 고려할 것이다. 그리고 원래의 데이터로 학습 알고리즘을 훈련하는 것을 생각할 것이다. 예를 들어 나의 가공되지 않은 원래 입력값이 $x^{(i)}$ 라고 하면, 나는 PCA를 알고리즘에 넣기 전에, $x^{(i)}$ 로 할 수 있는 일을 다 해볼 것을 추천한다. 이렇게 해서 잘 작동하지 않는다는 생각이 들면, 그러니까 여러분의 학습 알고리즘의 속도가 굉장히 느리거나, 메모리나 디스크 공간을 너무 차지해서 여러분이 데이터 값을 감소시키고 싶다면, $x^{(i)}$ 를 사용하는 작업을 잘 안될 경우에만, $x^{(i)}$ 를 사용하는 것이 잘 작동하지 않는다는 강한 확신이나 증거가 있을 때만, PCA를 실행하고 데이터 값을 감소시키는 것을 고려해보라. 사람들이 PCA를 포함시킨 프로젝트 계획을 짜면서 시작하는 모습을 보기 때문이다. 하지만 대신 PCA를 사용하지 않고도 사람들의 이러한 방식도 잘 작동할 수 있다. 그러므로 PCA를 얻기 위해서, k가 무엇인지 찾아내려고 너무 많은 시간을 보내기 전에 이를 대안으로 고려해 보라.

Q) PCA의 활용법으로 괜찮은 것을 모두 골라라. 정답 1, 4번

12'18"

지금까지 PCA였다. 마지막에 여러 가지 충고를 했지만 그럼에도 불구하고 여러분이 적절하게 쓰기만 한다면 PCA는 엄청나게 유용한 알고리즘이다. 나 또한 PCA 알고리즘을 무척 자주 사용한다. 나는 주로 학습 알고리즘의 실행 시간을 줄이는데 사용한다. 하지만 내 생각에 PCA의 가장 대중적인 활용은 데이터를 압축하는 것이다. 이는 메모리와 디스크 공간을 덜 차지하기 위해서이다. 그리고 또 데이터를 시각화하기 위해서이다. PCA는 자율 학습 알고리즘 중에서도 가장 자주 쓰이고 사장 강력한 알고리즘이다. 본 강의에서 배운 내용을 가지고 여러분이 PCA를 실행하고 이 모든 목적을 위해서도 잘 활용하기를 바란다.

Week 9

No.	Title
-----	-------

Week 9-1	Problem Motivation
----------	--------------------

00:00

이번 시간부터는 Anomaly Detection(이상 탐지)이라는 문제에 대해 이야기하고자 한다. 이는 널리 활용되고 있는 기계 학습의 형태다. 한가지 흥미로운 점은 이는 보통 자율 학습 문제에서 다뤄지는데, 어떤 면에서는 지도 학습 문제와 비슷한 측면을 보인다는 것이다. 그렇다면 이상 탐지란 무엇일까. 먼저 이를 설명하기 위해 예시를 들어보겠다. 여러분이 비행기 엔진 설계자인데, 비행기 엔진이 공정과정을 거칠 때, 품질 보증 테스트를 거치는데 테스트 내용으로 다음과 같은 비행기 엔진의 feature를 점검한다. 열 발생, 진동 등이 있다. 오래 전에 이와 같은 문제를 친구들과 함께 논의한 적이 있는데, 이는 실제 비행기 엔진에서 수집한 feature이다.

00:57

여기 x_1 부터 x_m 까지 데이터 세트가 주어졌다. 만약 m 비행기 엔진을 제조한다고 할 때, 이 데이터를 그려보면 이와 같이 생겼을 것이다. 따라서 여기 x 표는 분류되어있지 않은 예시다. 이상 탐지 문제는 다음과 같다. 다음날 새로운 비행기 엔진을 공정하려고 하는데, 새로운 엔진에 $x\text{-test}$ 라는 feature 세트를 가지고 있다. 이상 탐지 문제란 이 비행기 엔진에서 변칙적인 것이 발견되었는지, 즉 이 엔진이 테스트를 더 거쳐야 하는지 아니면 문제가 없어 보이므로 추가 테스트 없이 고객에게 배송되어도 되는지를 알고 싶다. 따라서 새 비행기 엔진이 여기 있는 지점이라면, 이 것은 지금까지 봤던 비행기 엔진과 별로 달라 보이지 않으니 문제가 없어 보인다. 하지만 새 엔진이, $x\text{-test}$ 가 이 지점을 가리킨다면, x_1, x_2 이 새로운 예시의 feature이다. $x\text{-test}$ 가 이렇게 집합에서 떨어져있게 되면 이를 anomaly(이상)이라고 말한다. 따라서 그 엔진은 다른 엔진들과 매우 다르기 때문에 고객에게 전달되기 전에 추가 테스트를 거쳐야 한다.

02:20

이상 탐지 문제를 공식으로 알아보면, x_1 부터 x_m 까지 예시가 주어지고, 이러한 m 예시를 보통 보통의 이상 없는 예시라고 한다. 그리고 새 예시 $x\text{-test}$ 에 이상이 있는지를 말해줄 알고리즘이 필요하다. 우리가 활용할 접근은 이러한 학습 세트, 분류되어있지 않은 학습 세트가 주어졌을 때, 모델 $p(x)$ 를 정하는 것이다. 즉, x 의 확률을 위한 모델을 생성하는 것이다. 여기서 x 는 비행기 엔진의 feature를 말한다. 이렇게 $p(x)$ 를 생성한 후에, 새 엔진은 $p(x_{\text{test}}) < \varepsilon$ 이고 이를 이상이라고 표시하자. 새 엔진이 있는데, $p(x)$ 모델의 경우 데이터로 측정해 볼 때, 이상이 있을 확률이 매우 낮다. 하지만 $p(x_{\text{test}})$ 의 경우에는 $p(x_{\text{test}}) \geq \varepsilon$ 일 경우에 이상이 없다고 본다. 그래서 이와 같은 학습 세트에서, 이 모델을 그려 넣으면, 모델 $p(x)$ 는 아마도 이 부근 아니면 여기 가운데쯤에 있을 확률이 매우 높다. 여기를 좀더 벗어나면 확률은 낮아진다. 더 밖으로 나가게 되면 확률은 더 낮아지고, 아예 멀어진 지점이라면 이상으로 간주할 것이다. 하지만 여기 가운데 있는 지점에서는 문제가 없는데, 중간에 있는 $p(x)$ 는 그 구역에 다른 예시들이 많이 밀집해 있기 때문이다.

04:03

이번에는 이상 탐지 예시를 적용해보도록 하자. 이상 탐지에서 가장 흔히 활용되는 어플리케이션은 fraud detection이다. 유저가 많고 그 유저가 다양한 활동을 한다면, 웹사이트 등에서 다양한 유저의 행동 feature x_i 를 연산할 수 있다. 먼저 모델을 생성해서, 다른 유저의 다양한 행동 방식을 확률, 유저 행동 feature의 특정 벡터 확률로 연산하는 것이다. 따라서 웹사이트에서 유저 행동 feature의 예시는 x_1 은 얼마나 자주 로그인 하는지, x_2 는 방문한 페이지 수 또는 트랜잭션 수 그리고 x_3 는 포럼에 포스팅 수, x_4 는 유저의 타자 속도라고 할 수 있다. 어떤 웹사이트에서는 실제로 유저의 글자당 타자 속도를 추적하기도 한다. 따라서 이와 같은 데이터를 기반으로 $p(x)$ 를 모델화할 수 있다.

05:07

마지막으로 $p(x) < \varepsilon$ 을 통해 웹사이트에서 이상한 행동을 하는 유저를 확인할 수 있다. 따라서 추가 검토를 할 수 있도록 유저의 프로필을 보내거나 그러한 유저의 추가적인 신원 확인을 요구할 수도 있고 웹사이트에서 사기와 같은 이상한 행동을 하지 못하도록 막을 수도 있다. 이와 같은 기술은 이상한 행동을 보이는 유저를 표시해 두는데, 유저뿐 아니라 그러한 행동까지 표시해둔다. 뭔가를 계속 훔치거나 이상한 행동 그 자체뿐 아니라 그러한 행동을 하는 유저도 찾아낸다.

이와 같은 기술은 실제로 많은 온라인 판매 웹사이트에서 행해지고 있고, 사기 또는 컴퓨터의 계좌를 훔치는 것과 같은 이상한 행동을 보이는 유저를 찾아내는 역할을 한다.

06:03

이상 탐지의 또 다른 예시는 manufacturing이다. 비행기 엔진에 대해 먼저 살펴봤는데, 이상이 있는 엔진을 찾아내서 추가 검토를 하도록 보내는 것이다. 세 번째 예시는 데이터 센터에 있는 모니터링 컴퓨터다. 이를 연구 중인 친구들이 몇 명 있다. 컴퓨터 클러스터 또는 데이터 센터에 기계가 많이 있을 경우, 각 기계마다 feature를 연산할 수 있다. 따라서 feature의 예로, 메모리를 얼마나 쓰고 있는지, 디스크 접근 횟수, CPU 작업 등이 있다. 또한, 좀 더 복잡한 feature로는 이 기계에서 CPU 작업 나누기 이 기계에서 네트워크 트래픽 양이 있다. 이와 같은 여러분의 데이터 센터에 있는 컴퓨터가 어떻게 작동하는지에 대한 데이터 세트에서, x 확률을 모델화할 수 있는데, 이러한 기계가 메모리를 얼마나 쓰고 있는지, 디스크 접근 횟수, CPU 작업 등의 확률을 모델화할 수 있다.

07:00

만약 $p(x) < \varepsilon$ 인 기계가 있다면, 그 기계는 특이한 행동을 보이고, 곧 고장이 날 수도 있으므로, 이를 시스템 관리자에게 살펴보도록 하기 위해 표시해두어야 할 것이다. 이것이 오늘날 다양한 데이터 센터를 가지고 기계에서 일어나는 특이한 행동을 살펴보는 방식이다.

Q) 이상 탐지 시스템은 $p(x) \leq \varepsilon$ 일 때, x 로 이상이 있음을 표시한다. 시스템에 이상이 없음에도 너무 많은 이상을 탐지해 표시할 경우(지도 학습에서와 비슷한 경우, 이러한 실수는 false

positive라고 함) 취해야 할 적절한 조치는?

1. ϵ 값을 증가시켜 본다
2. ϵ 값을 감소시켜 본다

정답 2번

이것이 바로 이상 탐지다. 다음 강의에서는 가우스 분포 (Gaussian distribution)와 가우스 확률분포(Gaussian probability distribution)의 리뷰 특성에 대해서 살펴볼 것이고 이를 가지고 이상 탐지 알고리즘에 적용해보도록 하겠다.

No.	Title
Week 9-2	Gaussian Distribution

00:00

이번 시간에는 정규 분포(normal distribution) 이라고도 하는 가우스 분포(Gaussian distribution)에 대해 살펴보도록 하겠다. 이미 가우스 분포에 대해 잘 알고 있다면 이번 강의는 패스해도 좋다. 하지만 가우스 분포 또는 정규 분포 이 뭔지 확실치 않고, 이를 학습한지 오래되었다면, 이 강의를 끝까지 들어주기를 바란다. 이제 가우스 분포를 적용하여 이상 탐지 알고리즘을 개발해보도록 하겠다. 먼저 x 는 열 값의 임의의 변수이고, 따라서 x 는 열 번호다. x 의 확률 분포가 Gaussian이고, 평균 μ , 변수 σ^2 이라고 하자. X 는 임의의 변수고, \sim 는 \sim 으로 분포된다는 것이다. 그리고 가우스 분포를 표기하면, $X \sim N(\mu, \sigma^2)$ 와 같다. 여기서 스크립터 N 은 normal을 가리키는데 Gaussian과 normal은 여기서 같은 의미다. 그리고 가우스 분포는 두 개의 매개 변수로 매개 변수화 되는데, 평균 매개 변수는 μ 로, 변수 매개 변수는 σ^2 라고 표기한다.

01:28

가우스 분포 또는 가우스 확률 밀도를 그려보면, 종 모양의 곡선 그래프가 될 것이고, 아마 전에 본 적이 있을 것이다. 여기 종모양 곡선 그래프는 두 개의 매개 변수, μ, σ^2 로 매개 변수화 되었다. 여기 종모양 곡선 그래프에 중심은 평균 μ 이다. 그리고 종모양 곡선 그래프의 너비는, 약 이 정도는 매개 변수 σ 로, 하나의 표준 편차라고 한다. 따라서 x 가 다른 값을 가질 확률을 명시한다. X 가 여기 중심에서 가지는 값은 꽤 높은데, 가우스 밀도가 꽤 높기 때문이다. 반면에 x 값이 이렇게 점점 중심에서 멀어지면, 확률을 점점 낮아진다. 이제 가우스 분포의 공식을 정리해보자. X 의 확률은, $p(x; \mu, \sigma^2)$ 이렇게 써주면, 이는 확률 x 는 두 매개 변수 μ, σ^2 로 인해 매개변수화 되었다는 것을 의미한다. 그리고 가우스 밀도 공식은 $(1 / \sqrt{2\pi} \sigma) * \exp(- (x -$

μ)²/2 σ^2) 이다. 이 공식을 외울 필요는 없다. 이는 단순히 왼쪽에 이는 종모양 곡선 그래프를 나타내는 공식일 뿐이다. 외울 필요는 없다. 이를 활용해야 할 때에는 찾아보면 된다.

02:55

따라서 여기 왼쪽의 수치는 μ 의 고정된 값과 σ 의 고정된 값으로 그린 그래프고, 여기 곡선 그래프는 여기 곡선 그래프는 $p(x)$ 가 $p(x ; \mu, \sigma^2)$ 를 함수로 그린 것이다. 때로는 σ^2 을 Variance(분산)라고 생각하면 편한데, 여기서 시그마는 표준 편차라고 하고, 이는 가우스 확률 밀도에서 너비를 명시한다. 또한 σ^2 은 Variance(분산)라고도 한다. 이제 가우스 분포가 어떻게 생겼는지 그 예시를 살펴보자. $\mu = 0, \sigma = 1$ 일 때, 가우스 분포는 0을 중심으로 그려지는데, 왜냐하면 이게 평균이고, 가우스 너비이므로 표준 편차는 여기 있는 시그마다. 가우스 예시를 몇 개 살펴보자. $\mu = 0, \sigma = 1$ 일 때, 가우스 분포는 0을 중심으로 그려지는데, 이는 평균이 0이고, 가우스의 너비가 매개 변수 시그마값에 의해 정해지기 때문이다.

04:17

또 다른 예시를 보겠다. 평균은 똑같이 0이고, 시그마는 0.5이다. 따라서 표준 편차는 0.5다. 따라서 σ^2 는 0.25가 된다. 이 가우스 분포의 예시에서, 가우스 확률 밀도는 이와 같다. 중심은 똑같이 0인데, 너비가 훨씬 작아졌다. 왜냐하면 시그마값이 작아져서 가우스 밀도의 너비가 절반 정도 줄어들었기 때문이다. 하지만 곡선 아래 부분이 확률 분포이기 때문에, 여기 색칠한 부분의 너비 값은 1이고, 이는 확률 분포의 특성이다. 따라서 이 그래프는 가우스 밀도가 훨씬 높은데, 여기 반은 표준 편차이고, 이는 두 배 더 크다.

05:08

또 다른 예시에서 $\sigma = 2$ 일 때, 더 둉뚱하고 옆으로 넓어진 가우스 밀도 그래프를 얻게 된다. 여기서 시그마 매개 변수가 가우스 분포를 정하기 때문에 너비가 넓어지는 것이다. 즉, 커브 아래의 부분은 여기 빗금 친 부분의 너비는 항상 1이고, 이는 확률 분포의 특성 중 하나다. 이게 넓어졌고, 여기 반은 양쪽이 똑같다. 마지막 예시에서는 평균과 시그마 둘 다 바꿔 보겠다. 0을 중심으로 하는 게 아니라 이 가우스 분포는 3을 중심으로 하는데, 이 것으로 가우스 분포 전체가 움직이기 때문이다.

05:50

마지막으로 매개변수 추정 문제를 살펴보자. 매개 변수 추정 문제란 무엇인가? x_1 부터 x_m 까지 예시가 있는 데이터 세트가 있는데, 예시 각각은 열번호라고 하자. 이 그래프에서 데이터 세트 예시 x 를 그려놨다. 여기서 가로축이 x 축이 되고 그 위에 x 예시를 이와 같이 표시했다. 매개변수 추정 문제는 이러한 예시가 가우스 분포에서 온 것이라고 하자. 따라서 각각 예시 $x(i) \sim$ 이 물결무늬 뜻은 분포되었다는 것이다. 여기 예시 각각이 정규 분포 또는 가우스 분포에서 어떤 매개변수 μ 와 σ^2 따라 분포되어 있다고 하자. 하지만 이 매개 변수의 값을 알지 못한다. 매개변수 추정의 문제는 주어진 데이터 세트에서 μ 와 σ^2 값을 추정해내는 것이다. 따라서 이와 같은 데이터 세트가 주어졌을 때, 이 데이터를 기반으로 가우스 분포를 추정해보면, 대략 이런 그림이

나올 것이다. M 가 분포 가운데에 있고, σ 는 가우스 분포에서 너비를 정하는 표준 편차를 의미한다. 이는 데이터를 잘 표현한 그래프처럼 보인다. 왜냐하면 이 데이터가 중심 지역에 있을 높은 확률을 보여주고, 옆으로 퍼져나갈수록 점점 낮은 확률을 보여주기 때문이다. 따라서 이것은 적절한 μ 와 σ^2 값의 추정이라고 할 수 있고, 이처럼 가우스 분포 함수를 그려볼 수 있다.

07:35

따라서 이번에는 공식을 써볼 텐데, 매개 변수 μ 와 σ^2 를 추정하는 표준 공식을 알아보자. 먼저 μ 를 추정하는 공식을 써보면, 다음과 같이 예시의 평균을 구한 것과 같다. μ 는 평균 매개변수이다. 학습 세트를 가지고 m 개의 예시의 평균을 구하는 것이다. 이는 이러한 분포의 중간값이 될 것이다. 그렇다면 σ^2 는 어떻게 구할까? 이 변수는, 표준 공식을 먼저 써보면, 다음과 같은데, 여기서 μ 는, 여기서 이 공식을 활용하여 연산한다. 여기서 변수란, 이 항을 보면 예시에 있는 두 값의 차이의 제곱을 나타내는데, $-\mu$ 의 분포다. 변수에서 두 예시의 차이, $-\mu$ 의 제곱의 평균을 추정할 것이다. 통계학 전문가에게만 덧붙이자면, 통계학을 잘 알고 있고 최대 우도 측정(maximum likelihood estimation)에 대해 들어봤다면, 이러한 매개변수는 μ 와 σ^2 의 소수값 최대 우도 측정인 것이다. 하지만 이에 대해 전에 들어본 적이 없다면, 걱정하지 않아도 된다. 여러분이 알아야 할 내용은 이 둘이 주어진 데이터 세트에서 μ 와 σ^2 값을 구하는 표준 공식이라는 것이다.

09:04

마지막으로, 이전에 통계학 수업을 들었던 사람들을 위해 덧붙이자면, 여기 m 대신에 $m-1$ 을 넣은 공식을 본 적이 있을 것이다. 따라서 여기 첫 번째 항이 $1/m$ 이 아니라 $1/(m-1)$ 이 된다. 기계 학습에서 보통 $1/m$ 공식을 배우지만, 실제로는 m 값이 큰 학습 세트 사이즈라고 가정할 때에는 $1/M$ 이든 $1/(M-1)$ 차이는 없다. 따라서 이렇게 다른 버전을 본 적이 있다면, 어떠한 버전을 활용해도 같은 값을 얻게 되는데, 기계 학습에서는 대부분 사람들이 이 $1/m$ 공식을 활용한다는 것이다. 이 두 버전은 다음 수학적 특성과 같은 이론적 특성에서는 약간 다르다. 하지만 실제로는 거의 차이가 없다고 봄도 된다.

Q) 가우스 밀도 공식은 다음과 같다. 다음 중 그림을 나타내고 있는 공식은?

정답 3번

지금까지 학습 세트가 주어졌을 때, 가우스 분포에서 μ 와 σ^2 값을 추정하는 방법과, 가우스 분포가 어떻게 생겼는지에 대해 배워봤다. 다음 시간에는 이 내용을 바탕으로 이상 탐지 알고리즘을 개발하는데 적용해보도록 하겠다.

No.	Title
Week 9-3	Algorithm

00:00

지난 시간에는 가우스 분포에 대해 학습했다. 이번 시간에는 이를 활용하여 이상 탐지 알고리즘을 개발해보도록 하겠다. 여기 분류되어있지 않은 학습 세트 m 개 예시가 있다고 하자. 각 예시는 $x \in \mathbb{R}^n$ 이다. 따라서 학습 세트는 feature 벡터 m 부터 마지막 제작된 비행기 엔진 m 또는 m 유저 feature 등이 될 수 있다. 우리가 이상 탐지를 해결하는 방법은, 먼저 데이터 세트에서 $p(x)$ 를 모델로 정한다. 이제 높은 확률 feature와 낮은 확률 feature를 알아낼 것이다. 따라서 x 는 벡터고, $p(x) = p(x_1 \dots x_n)$ x 의 첫 번째 요소이다. 여기에 곱하기 $p(x_2 \dots)$ 이는 두 번째 feature의 확률, 곱하기 $p(x_3 \dots)$ 세 번째 feature의 확률이고, 이제 마지막 feature $p(x_n \dots)$ 까지 곱해준다. 팔호 안에 곧 내용을 추가할 거라서 공간을 남겨 두었다. 이제 $p(x_1), p(x_2)$ 이 항들을 어떻게 만들 것이냐고 하면, x_1 이 가우스 분포에 의해 분포되었다고 가정하고, 이는 $x_1 \sim (\mu_1, \sigma_1^2)$ 라고 할 수 있다. 따라서 $p(x_1)$ 이 가우스 확률 분포라고 하면, $p(x_1; \mu_1, \sigma_1^2)$ 이라는 것이다. 비슷하게, x_2 를 가우스 분포해보면, $x_2 \sim (\mu_2, \sigma_2^2)$ 가 되고, \sim 는 μ_2, σ_2^2 에 의해 가우스 분포되었다는 의미다. 이것은 다른 가우스에 의해 분포되었는데, 이 가우스는 μ_2 와 σ_2^2 를 매개 변수로 한다. 또한, x_3 도 다른 가우스로 분포되는데, 이와 같이 μ_3 와 σ_3^2 다른 feature와 다른 평균과 표준 편차를 가지고 있으며, 이는 x_n 까지 이어진다. 이렇게 모델을 생성했다.

02:17

통계학을 공부한 학생들을 위해 설명을 덧붙이자면, 이 방정식은 독립 feature x_1 부터 x_n 까지 값에 대한 독립성 가정(independence assumption)과 대응한다. 하지만 실제로 이러한 부분의 알고리즘은 결국 이러한 feature들이 독립성에 가까운지 여부를 떠나서 잘 구현된다. 또한 독립성 가정이 참이 아닐 때도, 이 알고리즘은 잘 구현된다. 하지만 이러한 용어를 잘 알고 있지 않는다면, 여기서는 독립성 가정을 활용했지만, 걱정하지 않아도 된다. 이 알고리즘에 대해 이해하고 잘 구현할 수 있게 될 것이다. 방금 설명은 그저 통계학에 대해 잘 알고 있는 이들을 위한 추가 설명일 뿐이었다.

02:57

아까 내용을 이어서 설명하자면, 이 표현을 가지고 좀 더 단순화하여 표현해보면 다음과 같다. $\prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$ 여기 이 기호는 그리스 알파벳 대문자 파이다. 이 기호는 값의 세트의 product에 대응한다. 따라서 각종 표기법을 알고 있다면 이 기호는 $1 + 2 + 3 + \dots + n$ 까지의 합을 의미한다. 반면에 이 product 기호는, i 부터 n 까지를 나타나는데, 위의 각종 표기법과 같은데, 이번에는 곱셈을 하는 것이다. 따라서, $1x2x3\dots x_n$ 까지의 곱이 되는 것이다. 따라서 이와 같은 product 표기법을 활용하면, $\prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$ 이렇게 단순하게 표기할 수 있는 것이다. 여기 모든 내용을 이렇게 간단하게 표기하는 것이다. 왜냐하면 $x_j; \mu_j, \sigma_j^2$ 를 모두 곱해주는 것이기 때문이다.

04:20

이러한 $p(x)$ 분포를 추정하는데 문제는 밀도 추정 문제가 발생할 수 있는데, 여기 슬라이드의 제목이다.

Q) $\{x_1, \dots, x_m\}$ 과 같은 학습 세트가 있을 때, μ_j, σ_j^2 를 추정하는 공식은? ($\mu_j \in \mathbb{R}, \sigma_j^2 \in \mathbb{R}$)

정답 4번

그렇다면 지금까지 배운 내용을 토대로 이상 탐지 알고리즘을 살펴보자. 첫 번째 단계는 변칙 예시를 나타내는 feature x_i 를 준비하는 것이다. 시스템에서 사기 행각을 벌이는 이상한 유저를 발견하거나 이전에 살펴본 비행기 엔진 예시에서 엔진에서 뭔가 이상한 것을 발견하는 등의 feature를 생각해내는 것이다. 이상하게 큰 값이나 작은 값을 변칙 예시 feature x_i 로 정한다. 더 일반적인 방법은, feature를 선택하고, 수집하고 있는 데이터에 내용의 일반적인 특성이라고 생각하는 것을 설명하는 것이다. 다음으로 학습 세트 M 이 있을 때, 예시는 분류되지 않았고, $x(1)$ 부터 $x(m)$ 까지 있는데, 이 매개 변수를 μ_1 부터 μ_n 까지, 그리고 σ_1^2 부터 σ_n^2 까지 대입하는 것이다. 그리고 이 공식은 이전 강의에서 배웠던 공식과 흡사한데, 이러한 매개 변수의 추정을 활용할 것이다. 다시 설명하면, 여기 μ_j 는 j feature 값의 평균이고, $p(x_j; \mu_j, \sigma_j^2)$ 이 된다. 여기서 μ_j 는 j feature 값의 학습 세트의 평균을 말한다. 여러분이 이러한 공식을 연산하게 되면, j 값은 1부터 m 까지다. 따라서 이 공식을 활용해 $\mu_1, \mu_2, \dots, \mu_n$ 을 추정할 수 있는 것이다. σ^2 도 마찬가지이다.

06:17

이를 벡터화한 버전도 생각해볼 수 있다. 만약 μ 을 벡터라고 가정하면, $\mu = [\mu_1; \mu_2; \dots; \mu_n]$ 이 되는데, 매개 변수 세트의 벡터화 버전으로도 다음과 같이 $1/m \sum_{i=1}^m x(i)$ 쓸 수 있다. 따라서 방금 적어본 이 공식은 이 x_i 를 feature 벡터로 추정하여 μ 의 모든 n 값을 동시에 추정한다. 이것 또한 σ_j^2 를 추적하는 벡터화 공식을 만들어볼 수 있다. 다음으로, 새로운 예시가 있을 때, 새로운 비행기 엔진이 있을 때, 이 엔진에 이상이 있는지를 알고 싶다. 이 경우, $p(x)$ 를 연산해 새 예시의 확률을 연산하면 된다. 따라서 $p(x)$ 는 여기 이 공식과 같고, 여러분이 구현 또는 연산해야 할 것은 여기 이 공식이다. 이 공식은 가우스 확률을 구하는 것이고, 이를 연산하여 확률이 작다고 나오면, 이를 이상이 있다고 표시하면 된다.

07:27

이 방법을 적용한 예시가 있다. 이 슬라이드의 왼쪽 상단에 이 데이터 세트를 그래프화했다. 이것을 보면, feature x_1 이 있는데, 이 데이터 세트에서 평균이 여기쯤 되고, feature x_1 은 평균이 약 5정도 되고, 표준 편차의 경우, 이 데이터 세트의 x_1 값을 보면 표준 편자는 약 2가 된다. 그래서 여기가 σ_1 이 되고, 세로축에서 측정한 feature x_2 의 평균값은 약 3이고, 표준 편자는 약 1이다. 따라서 이 데이터 세트를 가지고 $\mu_1, \mu_2, \sigma_1, \sigma_2$ 를 추정하면, 이와 같은 값을 얻게 된다.

여기서는 표준 편차로 시그마를 썼는데, 이전 슬라이드에서 공식은 시그마 제곱을 활용했다. σ_1^2, σ_2^2 . 따라서 σ_1, σ_2 를 쓸지 σ_1^2, σ_2^2 를 쓸지 주의해야 한다. 예를 들면, σ_1^2 는 2를 제곱하니 4와 같다.

08:30

다음 그림에서는 $p(x_1; \mu_1, \sigma_1^2)$, $p(x_2; \mu_2, \sigma_2^2)$ 는 이와 같은 분포 그래프로 나타난다. 만약 $p(x)$ 를 그린다면, 여기 두 개를 말하는데, 이와 같이 생긴 표면도를 얻게 될 것이다. 이게 $p(x)$ 표면도이고, 여기 높이는 특정 표면 지점부터 높이를 말하는데, x_1, x_2 와 같은 특정한 값은, $x_1 = 2, x_2 = 2$ 일 때, 이 지점이고, 이 3차원 표면도에서 높이가 $p(x)$ 가 된다. 따라서 $p(x)$ 는 이 그래프의 높이는, $p(x_1; \mu_1, \sigma_1^2)$ 곱하기 $p(x_2; \mu_2, \sigma_2^2)$ 가 된다. 이것이 바로 매개 변수를 이 데이터에 대입하는 방법이다. 이번에는 새로운 예시를 대입해보기로 하자. 이렇게 새로운 예시를 여기에 대입해보면, 이것은 변칙인가 아닌가? 또는 다른 두 번째 예시를 여기에 그려보자. 이것은 또 변칙인가 아닌가? 이를 알아보기 위해, ε 값을 정해야 한다. 여기서는 $\varepsilon=0.02$ 라고 하자. 이 값을 정하는 방법은 나중에 설명해주겠다.

09:55

아까 첫 번째 예시를 $x(1)_{\text{test}}$ 라고 하고, 두 번째 예시를 $x(2)_{\text{test}}$ 라고 하자. 이제 $p(x(1)_{\text{test}}) = 0.0426$ 를 연산하는 것인데, 이는 꽤 큰 값이다. 특히 $0.0426 \geq \varepsilon$ 가 된다. 따라서 이는 최소 ε 보다 큰 높은 확률을 가지게 되므로, $x(1)_{\text{test}}$ 는 변칙이 아니다라고 할 수 있다. 반면에 $x(2)_{\text{test}}$ 를 연산해보면, 훨씬 작은 수다. 따라서 이는 $0.0021 < \varepsilon$ 이므로 $x(2)_{\text{test}}$ 변칙이 되는 것이다. 이는 우리가 선택한 ε 보다 훨씬 작기 때문이다. 여기서 다시 설명해주겠다. 다시 말해, 3차원 표면도를 통해 보면, x_1, x_2 값이 표면에서부터 높이가 높기 때문에, 이는 변칙이 아닌 표준 예시가 되는 것이다. 반면에 이 지점과 먼 곳은 낮은 확률은 가지고 있고, 이러한 지점을 변칙이라고 표시하는 것이다. 따라서 이와 같은 지역을 정의하면, 이렇게 그려질 것이고, 이 밖의 모든 지점은 변칙이라고 표시되고, 그 안은 문제가 없는, 변칙이 아닌 예시라고 간주한다. 따라서 $x(2)_{\text{test}}$ 예시는 이 지역 밖에 위치하고 있으므로, 확률이 매우 작으므로 우리는 이를 변칙 예시라고 간주하는 것이다.

11:22

이번 강의에서는 x 의 확률인 $p(x)$ 를 어떻게 추정하는지 알아봤고, 이를 통해 이상 탐지 알고리즘을 개발할 수 있었다. 이번 강의를 끝으로, 데이터 세트를 가지고, 매개 변수를 대입하여, 매개 변수를 추정하는 모든 과정을 학습했다. 매개변수 μ 와 σ 에 새로운 예시를 가져와서 새로운 예시가 변칙인지 아닌지를 살펴봤다. 다음 강의에서는 이 알고리즘에 대해 더 자세히 탐구하여 이를 실제로 어떻게 구현하는지에 대해 알아보도록 하자.

No.	Title
-----	-------

00:00

지난 시간에는 이상 탐지 알고리즘을 개발해봤다. 이번 시간에는 이상 탐지 문제에 특정 어플리케이션을 개발하는 것, 특히 이상 탐지 알고리즘을 평가하는 문제에 대해 알아보겠다. 지난 강의에서, 실수 평가의 중요성에 대해 이야기했었고, 특정 어플리케이션을 위한 학습 알고리즘을 개발하고자 할 때 필요한 아이디어였고, 다양한 선택이 뒤따르는데, 예를 들어, 어떠한 feature를 선택하고 활용할지 등이 있다. 여러분의 학습 알고리즘을 평가하는 방법을 알고 있을 때, 이러한 선택에 대해 결정을 하는 것은 훨씬 쉬워지는데, 학습 알고리즘이 연산을 통해 수를 제공해주기 때문이다. 따라서 무언가 결정을 하려고 할 때, 추가 feature에 대한 아이디어가 있는데, 이것을 추가할지 말지 고민한다. 만약 여러분이 feature를 가지고 알고리즘을 구현하고, feature 없이 알고리즘을 구현할 때, 어떠한 값을 얻을 수 있는데, 이는 이러한 feature를 추가해서 성능이 더 향상되었는지 그 반대일까? 그런 다음에 더 나은 방법, 더 쉬운 방법으로 이러한 feature를 추가할지 말지를 결정할 수 있게 된다.

01:07

따라서 이상 탐지 시스템을 빠르게 개발할 수 있게 된다면, 이상 탐지 시스템을 평가하는 방법을 터득하는데 매우 도움이 될 것이다. 이를 위해서, 이상 탐지 시스템을 평가하기 위해서 분류된 데이터를 가지고 가정해 볼 것이다. 지금까지 분류되지 않은 데이터를 활용하여 이상 탐지를 비지도 학습 문제로 간주했다. 하지만 변칙 예시와 그렇지 않은 예시를 명시한 분류된 데이터가 있다면, 이것이 바로 이상 탐지 시스템을 평가하는 표준 방법이라고 생각한다. 따라서 비행기 엔진 예시를 다시 들면, 과거에 제조된 어떤 비행기 엔진에서 변칙 예시를 분류한 데이터가 있고, 이는 결함이나 뭔가에 문제가 생겼다고 판단이 내려졌다. 또 전혀 문제가 없는 정상 예시도 있다고 하자. 이제 $y=0$ 은 정상 또는 이상 없는 예시라고 하고, $y=1$ 은 변칙 예시라고 하겠다. 이상 탐지 알고리즘을 개발하고 평가하는 과정은 다음과 같다.

02:27

이를 학습 세트라고 하고, 테스트 세트에 있는 교차 검증에 대해서는 나중에 다시 설명하겠다. 하지만 학습 세트는 보통 아직 분류되지 않은 학습 세트라고 여겨진다. 여기 정상, 이상 없는 예시가 있는데, 혹시라도 분류되지 않은 학습 세트에 몇 개의 변칙 예시가 들어있어도 상관없다. 다음으로 교차 검증 세트와 테스트 세트를 정의해볼 건데, 이는 특정한 이상 탐지 알고리즘을 평가하기 위함이다. 따라서 교차 검증 세트와 테스트 세트는 다음과 같이 추정할 건데, 교차 검증 세트에 예시 몇 개를 추가할 수 있고, 테스트 세트는 변칙 예시라고 알려진 예시를 추가할 수 있다. 따라서 테스트 세트는 $y=1$ 과 같은 예시를 포함하고 있는데, 이는 이상 있는 비행기 엔진과 대응한다.

03:17

여기 자세한 예시가 있다. 여기 있는 모두가 우리에게 주어진 데이터라고 하자. 엔진 10,000 예시를 제조했고, 우리는 이것이 모두 정상인 엔진이라고 알고 있다. 또한 이 10,000개 예시 안에 몇 개의 결함이 있는 엔진이 발견된다고 해도 이는 크게 문제되지 않는다. 하지만 이 10,000개의 예시의 대다수는 정상이고 이상이 없는 엔진이라고 가정하게 된다. 그렇다면, 이번에는 공장이 오래되다 보니 제조 공정에서 결국 24개에서 28개 정도의 결함이 있는 엔진이 발견되었다고 하자. 이상 탐지에서 보통은 변칙 예시의 수는 $y=1$ 으로, 보통 20에서 50개 정도는 발견된다. 이는 매우 전형적인 예시의 수의 범위고, $y=1$ 예시의 수이다. 하지만 보통 좋은 예시의 수가 훨씬 더 많다.

04:20

따라서 이러한 데이터 세트가 주어졌을 때, 이를 학습 세트와 교차 검증 세트, 그리고 테스트 세트로 나누는 방법은 다음과 같다. 먼저 10,000개의 정상 비행기 엔진에서 6,000개를 분류되지 않은 학습 세트에 넣는다. 이 분류되지 않은 학습 세트라고 하고, 여기 모든 예시는 $y=0$ 에 대응한다고 알고 있다. 그리고 이는 $p(x)$ 에 대입하면 된다. 여기 6,000개 엔진을 $p(x)$ 에 대입하는데, 여기서 $p(x) = p(x_1; \mu_1, \sigma_1^2), \dots, p(x_n; \mu_n, \sigma_n^2)$ 이 성립한다. 여기 6,000 예시는 $\mu_1, \sigma_1^2, \dots, \mu_n, \sigma_n^2$ 를 평가하는데 활용될 것이다. 이것이 우리의 학습 세트이고, 정상 예시의 대부분을 차지한다. 다음으로 정상 비행기 엔진을 가지고 이 중 몇 개를 교차 검증 세트에 넣고, 또 몇 개를 테스트 세트에 넣을 것이다. 따라서 $6,000 + 2,000 + 2,000$ 으로 10,000개의 정상 비행기 엔진 예시가 나눠졌다. 이제 여기서 결함이 있는 비행기 엔진이 20개가 있고, 이를 가지고 10개는 교차 검증 세트, 그리고 10개는 테스트 세트에 넣을 것이다. 그리고 나서 다음 슬라이드에서 이상 탐지 알고리즘을 평가하는데 이를 어떻게 활용할지 이야기해보겠다.

05:47

여기 설명한 내용은 분류된 예시와 분류되지 않은 예시를 나누는 좋은 방법을 하나 보여준 것이다. 정상과 결함이 있는 비행기 엔진을 정상 엔진은 60%, 20%, 20%로 나누었고, 결함 엔진은 교차 검증 세트와 테스트 세트에 넣었다. 다음 슬라이드에서 왜 이렇게 했는지 설명하겠다. 또 다른 방법으로 사람들이 이상 탐지 알고리즘을 어떻게 적용하는지 살펴보면, 다른 방법으로 데이터를 나눈 것을 발견할 수 있다. 다른 대안으로, 이는 그리 추천하는 대안은 아니지만, 10,000개의 정상 엔진에서 6,000 정도를 학습 세트에 넣고 4,000개를 교차 검증 세트와 테스트 세트에 넣는다. 우리는 교차 검증 세트와 테스트 세트가 완전히 다른 데이터 세트라는 것을 알고 있다. 하지만 이상 탐지에서는 같은 정상 엔진 세트를 교차 검증 세트와 테스트 세트에 활용하는 이들을 볼 수 있을 것이고, 결함 엔진의 똑같은 세트를 교차 검증 세트와 테스트 세트에 적용하는 경우도 있다. 따라서 이러한 사례를 볼 때, 이는 그다지 추천하고 싶은 방법은 아니다. 교차 검증 세트와 테스트 세트에서 같은 데이터를 활용하는 것은 좋은 기계 학습 연습이 아니라고 여겨진다. 하지만 사람들이 종종 이렇게 하는 경우가 있다.

07:09

따라서 학습, 교차 검증, 테스트 세트가 주어졌을 때, 이를 어떻게 평가하고 알고리즘을 개발하고

평가하는지 설명하도록 하겠다. 먼저 학습 세트를 가져와서 모델 $p(x)$ 를 대입한다. 이 모든 가우스를 m 개의 분류되지 않은 비행기 엔진 예시에 대입한다. 이는 분류되지 않은 예시라고 하겠다. 하지만 이것은 정상 비행기 엔진이라고 우리는 간주하고 있다. 이상 탐지 알고리즘이 예측을 하고 있다고 하자. 따라서 테스트 세트의 교차 검증에서 테스트 예시 x 는, 이 알고리즘이 $y=1$ 을 예측한다고 할 때, $p(x) < \varepsilon$ 이 되고, $y=0$ 을 예측하면, $p(x) \geq \varepsilon$ 라고 하자. 따라서 x 를 가지고 예측을 하려고 할 때, 라벨은 $y=1$ 은 이상이 있다는 것이고, $y=0$ 은 정상 예시라는 것이다.

08:06

그렇다면, 학습, 교차 검증, 테스트 세트에서 알고리즘은 어떻게 개발하는가? 더 구체적으로 말하자면, 이상 탐지 알고리즘을 어떻게 평가하는가? 첫 번째로 분류되지 않은 학습 세트에 모델 $p(x)$ 학습 데이터를 대입한다. 이를 가지고, 분류되지 않은 학습 세트에서 여기 예시는 정상 비행기 엔진의 대다수라고 간주하고 있는데, 이는 이상이 발견되지 않았기 때문에, 모델 $p(x)$ 를 대입하는 것이다. 이 데이터에 있는 모든 가우스의 매개 변수에 적용될 것이다. 다음으로, 교차 검증 세트와 테스트 세트에서 이상 탐지 알고리즘을 y 값을 예측하기 위해 활용할 것이다. 따라서 테스트 예시에서, $(x(i)\text{test}, y(i)\text{test})$ 가 있는데, 여기서 y 는 이것이 변칙 예시인지 아닌지에 따라 1 또는 0이 된다. 테스트 세트 입력값 x 에서 이상 탐지 알고리즘은 $p(x) < \varepsilon$ 라면 $y=1$ 로 예측할 것이다. 이상이라고 판단할 확률은 매우 낮을 것이다. 이 알고리즘이 $y=0$ 을 예측한다면, 이는 $p(x) \geq \varepsilon$ 인 경우다. 따라서 정상 예시를 예측하는 것은 $p(x)$ 값은 큼 것이다.

09:27

우리는 이제 이상 탐지 알고리즘을 여기 테스트 세트 또는 교차 검증 세트에 있는 y 로 분류된 값이 어떤 값인지 예측을 하는데 활용하게 된다. 이는 지도 학습 셋팅과 비슷한 과정을 보여준다. 라벨 테스트 세트가 있을 때, 알고리즘은 이러한 라벨을 예측하는데, 따라서 이러한 라벨을 맞게 분류하는가에 따라 이를 평가할 수 있게 된다. 물론 여기 라벨은 매우 편향되어 있을 텐데, 그 이유는 $y=0$ 이고, 이는 정상 예시이고, 변칙 예시 $y=1$ 보다 훨씬 더 흔한 경우이기 때문이다. 하지만 이는 평가 metrics 자료와 매우 가깝기 때문에, 이를 지도 학습에서 활용할 수 있다.

Q) 모델 $p(x)$ 를 다음과 같이 대입하였다. 교차 검증 세트 또는 테스트 세트에서 평가할 때, 알고리즘은 다음과 같이 예측한다. 여기에서 분류 정확도를 활용하는 것은 알고리즘의 성능을 평가하기 위한 좋은 방법인가?

1. 그렇다, 교차 검증 세트 / 테스트 세트에 데이터가 있기 때문이다.
2. 아니다, 교차 검증 세트 / 테스트 세트에 라벨이 없기 때문이다.
3. 아니다, 편향된 분류되어 있기 때문이다. (이 알고리즘은 $y=0$ 을 예측하는 분류 정확도가 높다)
4. 교차 검증 세트에서는 아니지만, 테스트 세트에서는 맞다.

정답 3번

10:12

따라서 활용하기 좋은 평가 metric은 뭘까? 이 데이터가 매우 편향되었기 때문에, 왜냐면 $y=0$ 인 경우가 훨씬 많기 때문에, 분류 정확도는 좋은 평가 metric은 아니다. 이전 강의에서 이에 대해 설명한 바 있다. 한쪽으로 편향된 데이터 세트가 있을 때, $y=0$ 을 항상 예측할 때, 이는 매우 높은 분류 정확도를 가지고 있다. 따라서 평가 metrics를 활용해야 하는데, true positives, false positives, false negatives, true negatives 의 부분을 연산하는 것 또는 이 알고리즘의 정확성 또는 recall을 연산하는 것 또는 이는 실수의 위치나 recall 번호를 요약하는 f1-score를 연산하는 것 등이 있다. 이와 같이 테스트 세트 또는 교차 검증 세트에 있는 이상 탐지 알고리즘을 평가하는 방법에 대해 알아봤다.

11:01

다음으로 이상 탐지 알고리즘 설명 초반에 이와 같은 매개변수 ε 가 등장했다. ε 는 어떤 것이 이상이 있다는 것을 표시하기 할 때 활용하는 것이다. 따라서 교차 검증 세트가 있을 때, 이와 같은 매개 변수 ε 를 선택하는 다른 방법은 ε 의 다양한 값을 시도해보고, ε 값을 선택하는 것인데, f1-score 값을 최대화하는 ε 값을 선택하거나 크로스 검증 세트에서 잘 작동되는 것을 선택하는 것이다. 일반적으로 학습/테스트/교차 검증 세트를 줄이는 방법은 결정을 하려고 할 때, 예를 들어 어떤 feature를 포함할지, 매개 변수 ε 를 조정할 지와 같은 예시가 있는데, 그때 교차 검증 세트에서 알고리즘을 계속해서 평가하여 어떠한 feature를 사용했는지, ε 값을 어떻게 설정했는지와 같은 결정을 내려서, 교차 검증 세트의 알고리즘을 평가하는데 활용하고, feature 세트를 선택해서 만족할만한 ε 값을 찾았을 때, 최종 모델을 가지고 이를 평가하여 테스트 세트의 최종 평가를 할 수 있게 된다.

12:12

이번 시간에는 어떻게 이상 탐지 알고리즘을 평가하는지 그 과정에 대해 살펴봤는데, 알고리즘을 평가하기 위해서는, 실수 평가와 f1-score와 같은 수를 가지고 하는 평가는 여러분이 이상 탐지 시스템을 개발하려고 할 때, 시간을 더 효율적으로 활용할 수 있게 해준다. 또한 우리는 다음과 같은 결정을 내려야 하는데, ε 값을 선택해야 하고, 어떤 feature를 추가할지 등이 있다. 이번 시간에는 분류된 데이터를 이상 탐지 알고리즘을 평가하기 위해 조금 활용해봤는데, 이는 지도 학습 셋팅에 좀 더 다가가게 된 것이다. 다음 시간에는 이에 대해 더 살펴볼 것이다. 특히, 언제 이상 탐지 알고리즘을 활용해야 하는지 그리고 이를 대신해 언제 지도학습을 활용해야 할지 그리고 이러한 두 공식에 차이점은 무엇인지에 대해 살펴보겠다.

No.	Title
-----	-------

00:00

지난 시간에는 이상 탐지 알고리즘을 평가하는 과정에 대해 배워봤다. 이를 통해 예시가 있는 label 데이터를 활용해봤고, 이는 이상이 있는지 정상인지 $y=1$ 인지 $y=0$ 인지 알게 되었다. 따라서 이와 같은 의문점이 생겼는데, label 데이터가 있을 때, 예시가 있는데 이상이 있거나 그 중에서는 정상도 있을 것이다. 그럼 그냥 이 알고리즘에 supervisor를 활용하면 어떨까? 이는 로지스틱 회귀나 신경망을 통해 분류된 데이터를 가지고 $y=1$ 인지 혹은 0 인지를 예측하면 어떨까? 이번 시간에는 이상 탐지 알고리즘을 언제 활용해야 하는지, 알고리즘에서 supervisor를 활용하는 것 대신에 더 효율적일지에 대한 아이디어와 가이드라인을 설명하도록 하겠다.

00:46

이 슬라이드는 여러분이 이상 탐지와 지도 학습 중 어느 것을 활용하는 것이 효율적 일지를 보여준다. 만약 positive 예시의 수가 매우 적은 문제가 있는데, $y=1$ 은 이상 예시라는 것을 기억해두자. 그렇다면 여기서 이상 탐지 알고리즘을 활용하는 것을 고려해 볼 것이다. 0-20개의 positive 예시가 있고, 50개까지가 주로 활용된다. 따라서 positive 예시의 수가 매우 적을 경우, 이를 테스트 세트에 있는 교차 검증 세트에 저장한다. 반대로, 보통 이상 탐지 세팅에서, 정상 비행기 엔진에서 정상 예시처럼 negative 예시의 수가 상대적으로 많은 경우가 종종 있다. 그럴 경우, 이와 같은 매우 많은 수의 negative 예시를 $p(x)$ 에 대입하기 위해 활용하게 된다. 따라서 많은 이상 탐지 어플리케이션에서 positive 예시는 적고, negative 예시가 많다. 그리고 $p(x)$ 를 예측하는 과정에서, 모든 가우스 매개변수를 대입할 때, negative 예시만 필요하게 된다. 따라서 negative 데이터가 많으면 $p(x)$ 를 잘 대입할 수 있다.

02:05

반대로, 지도 학습에서는 보통 positive 와 negative 예시의 수가 둘 다 많다. 이런 경우에 문제 해결 방법은 이상 탐지 알고리즘을 쓸지, 지도 학습 알고리즘을 쓸지를 정해야 한다. 다음은 사람들이 이상 탐지에 대해 생각하는 예시다. 이상 탐지 어플리케이션에서, 매우 다양한 종류의 변칙들이 있다. 어떤 것이 잘못되는 다양한 방법을 생각해보자. 비행기 엔진에서도 이러한 예시를 찾아볼 수 있다. 이런 경우, positive 예시의 수가 적을 때, 알고리즘이 변칙이 어떠한 것인지를 학습하는 것이 어려울 수 있다. 특히, 나중에 등장할 변칙이 지금까지 보지 못한 것일 수도 있다. 따라서 positive 예시 세트에서는 비행기 엔진에 이상이 생길 수 있는 예시가 5개나 10개 또는 20개 정도 있다고 하자. 하지만 내일이 되면, 완전히 새로운 변칙 세트를 탐지해야 할지도 모른다. 지금껏 한번도 보지 못한 비행기 엔진이 고장 날 수 있는 완전 새로운 방법이다. 이런 경우, 하기 어려운 positive 예시 대신 이와 같이 가우스 $p(x)$ 모델을 가지고 negative 예시를 다루는 것이 더 도움이 될 것이다. 왜냐하면 내일 발생할지 모르는 변칙이 지금껏 보지 못한 새로운 종류일 수도 있기 때문이다.

03:32

반면에, 다른 문제에서, positive 예시는 이에 대해 파악할 수 있을 정도로 충분할 수 있다. 특히, 이후에 등장할 positive 예시가 학습 세트에 있는 것들과 유사할 거라고 생각하게 된다. 그러한 설정에서 모든 positive 예시와 모든 negative 예시를 보고 이를 구분하는데 활용하는 알고리즘에 supervisor를 활용하는 것이 합리적이다. 특정 문제를 해결할 때, 이상 탐지 알고리즘을 활용하지 또는 지도 학습 알고리즘을 활용할지에 대한 선택을 하는데 도움이 되었으면 한다. 이상 탐지에서 특히 구별되는 점은 positive 예시가 적어서, 학습 알고리즘이 positive 예시에 대해 학습하지 못하는 경우에 있다. 따라서 이럴 경우, negative 예시를 많이 가져와서 이를 많이 학습하도록 하는 것이다. 비행기 엔진의 negative 예시에 있는 $p(x)$ 만 학습하도록 하는 것이다. 그리고 적은 수의 positive 예시로 알고리즘을 평가하여 이를 교차 검증 세트 또는 테스트세트에서 활용하는 것이다.

04:41

여기 '다양한 형태의 변칙 부분'을 더 쉽게 설명해보도록 하겠다. 이전 강의 내용 중에서 이메일 스팸 예시를 다른 적이 있다. 그 내용 중, 스팸 메일 형태가 다양했다. 물건을 팔려고 하는 스팸 메일도 있었고, 비밀번호를 훔쳐가거나 이는 피싱이라고 하고, 이처럼 다양한 종류가 있었다. 하지만 스팸 문제에서는 스팸 메일의 예시가 충분해서 거의 모든 스팸 메일 종류를 알고 있기 있는데, 이는 스팸 메일 예시가 많기 때문이다. 그래서 종류가 다양한 스팸 메일 예시를 지도 학습 세팅이라고 보는 것이다.

05:21

다음으로, 이상 탐지가 활용되는 예시로는 사기 탐지(fraud detection)가 있다. 사람들이 사기 행각을 벌이는 종류는 다양하고, 웹사이트에서 그러한 사기 행각을 벌이는 유저의 수는 상대적으로 적은 경우, 이상 탐지 알고리즘을 적용할 수 있다. 만약 여러분이 주요 웹사이트에서 상품을 판매하고 있다면, 많은 사람들이 해당 웹사이트에서 사기 행각을 벌이려고 할 것이다. 따라서 $y=1$ 의 예시는 많은데, 여기서 사기 탐지는 사실 지도 학습 부분에서 다룰 수 도 있다. 하지만 웹사이트에서 수상한 행동을 하는 유저의 예시를 많이 보지 못했기 때문에, 더 자주 발생하는 사기 탐지는 학습 지도 알고리즘 보다는 이상 탐지 알고리즘으로 간주된다. 다음 예시는 전에 살펴봤던 제조업에 관한 것이다. 많은 예시를 보게 되는데, 그것들에서 이상이 발견되지 않기를 바라지만, 제조 과정에서는 많은 양을 처리하다 보면, 불량 예시가 나오기 마련이다. 따라서 제조업 또한 지도 학습 쪽으로 넘어갈 수 있다. 하지만 많은 불량 예시를 발견하지 않았다면, 데이터 센터에서 이상 탐지 모니터링 기계에서 이상 탐지 알고리즘을 적용할 수 있다.

06:45

반면에, 이 쪽에는 분류, 날씨 예측, 암 여부 판별 등이 있다. 만약 positive와 negative 예시의 수가 같을 경우, 이를 보통 supervisor 문제로 간주하다.

Q) 다음 중 어떠한 예시가 지도 학습 알고리즘 보다 이상 탐지 알고리즘에 더 적절한가? (복수 선택 가능)

1. 전기 공장(고객에게 전력 제공)을 운영하는데, 전기 시설 중 어디에 이상이 있는지를 모니터 하려고 할 때
2. 전기 공장을 운영하는데, 내일 필요한 전력량을 예측하려고 할 때(이를 통해 적절한 전력 발생 용량을 증가시킬 계획을 세울 수 있음)
3. 컴퓨터 시각/보안 어플리케이션을 통해, 회사 주차장에서 수상한 행동을 하는 사람이 있는지를 영상을 통해 감시하려고 할 때
4. 컴퓨터 시각 어플리케이션을 통해, 가게에 들어오는 손님의 성별을 구분하기 위해 이미지를 조사하려고 할 때

정답 1번 3번

학습 문제의 특성에 대해 이해하여 이상 탐지 문제와 지도 문제를 구분할 수 있는 시간이 되었기를 바란다. 이외에도 다양한 기술 회사 등에서 다양한 문제를 직면하게 될 텐데, 때로는 positive 학습 예시가 거의 없거나 때로는 하나도 없는 경우도 있다. 또한 우리가 전에는 본적이 없는 이상에 대한 예시도 정말 다양하다. 이와 같은 문제에서 이상 탐지 알고리즘이 가장 흔하게 활용되고 있다.

No.	Title
Week 9-6	Choosing What Features to Use

00:00

지금까지 여러분은 이상 탐지 알고리즘에 대해, 그리고 이상 탐지 알고리즘을 평가하는 방법에 대해서 살펴봤다. 이상 탐지를 적용할 때, 가장 큰 영향을 미치는 것은 바로 이상 탐지 알고리즘에서 어떠한 feature를 사용하고, 선택하는지로 밝혀졌다. 이번 시간에는 이상 탐지 알고리즘을 디자인하고 어떠한 feature를 선택해야 할지에 대한 제안과 가이드라인을 설명하도록 하겠다. 이상 탐지 알고리즘에서 이와 같은 $p(x_i; \mu_i, \sigma^2_i)$ 가우스 분포를 활용하여 feature를 정하는 것이었다. 내가 주로 활용하는 방법은 데이터를 좌표로 그리거나, 히스토그램을 그려서 이를 이상 탐지 알고리즘에 적용하기 전에 데이터가 약간 가우스처럼 보이는지 확인한다. 그럴 경우, 데이터가 가우스가 아닐지라도 보통 잘 적용이 되는데, 구현하기 전에 확인해볼 수 있는 좋은 sanitary check다.

01:05

만약 데이터가 가우스처럼 생기지 않았을 경우라도, 해당 알고리즘은 잘 구현이 되는 것을 알 수 있다. 예를 들어, 이와 같이 데이터를 그려보면, 이건 히스토그램을 그린 건데, 히스토그램을 그리기 위해서는 옥타브에서 HIST 명령어를 사용한다. 여기를 보면 약간 가우스와 비슷하게 생겼는데, 따라서 feature가 이렇게 생겼을 경우, 알고리즘에 잘 적용할 수 있는 것이다. 하지만 데이터를 히스토그램으로 나타낼 때, 이와 같이 생겼다면, 이는 좀 모양 곡선으로 생기지 않았고, 이는 매우 불균형적인 분포다. 한 쪽으로 완전히 쏠려있다. 만약 데이터가 이와 같이 그려진다면, 나는 보통 이를 가우스처럼 보이게 만들기 위해서 다른 형태를 적용한다. 이런 경우에도 알고리즘이 구현되겠지만, 하지만 가우스와 비슷하게 변환을 조금 준다면 더 잘 구현될 것이다. 따라서 이러한 데이터 세트가 주어졌을 때, 데이터에 $\log(x)$ 변환을 주어, 이 히스토그램을 다시 그려보면, 이 예시의 히스토그램은 이렇게 된다. 이제 좀 더 가우스처럼 보인다. 이는 좀더 보통의 종 모양 곡선처럼 생겼고, 이와 같은 평균과 매개변수 시그마를 적용할 수 있게 된다.

02:21

따라서 로그 변환이라는 것은, feature x_1 이 있을 때, x_1 의 히스토그램이 이렇게 생겼고, 이 feature x_1 을 가지고 $\log(x_1)$ 으로 변환하여 새로운 x_1 값을 얻게 되고, 오른쪽과 같은 가우스의 형태와 비슷한 히스토그램을 얻게 되는 것이다. 로그 변환 말고 다른 방법으로는, 여기 feature x_2 가 있을 때, 이를 $\log(x_2+1)$ 으로 대체하여 아니면 $\log(x_2 + c)$ 가 있는데 이 항수 c 를 가지고 가우스와 비슷한 형태로 작업을 할 수 있다. x_3 는 이렇게 제곱근을 씌워서 대체할 수 있는데, 이는 x_3 의 $1/2$ 제곱으로 바꿔 쓸 수 있다. 그럼 또 여기 매개변수 다른 예시 $1/2$ 을 가지고 작업을 해보면, x_4 에는 $1/3$ 제곱을 해주는 것이다. 여기 지수 매개 변수, 매개변수 C 는 매개 변수의 예시가 되고 이를 가지고 좀 더 가우스와 비슷한 데이터를 만들 수 있는 것이다.

03:44

이번에는 실제 데모를 통해 데이터를 변환하여 좀 더 가우스와 비슷하게 만드는지 알아보자. 옥타브에 feature x 세트를 이미 로딩해놨다. 여기 수천 개의 예시가 로딩되어있다. 이제 데이터의 히스토그램을 불러오겠다. Hist x 명령어를 사용한다. 여기 히스토그램이 있다. 자동으로 빈이 10개인 히스토그램을 활용하고 있다. 하지만 나는 좀 더 좋은 그리드 히스토그램으로 만들고 싶어서, $hist(x, 50)$ 을 입력하면, 이렇게 50개의 다른 빈이 생성된다. 이제 더 보기 좋다. 하지만 아직 가우스처럼 보이지는 않는다. 이제 데이터를 조금 작업해보자. 이번에는 $hist(x.^{0.5}, 50)$ 를 입력해보자. 데이터에 제곱근을 씌워서 이를 히스토그램으로 그려보는 것이다. 이제 좀 더 가우스처럼 보이지만 아직 부족하다. 이번에는 $hist(x.^{0.2}, 50)$ 를 입력하면 좀 더 가우스처럼 보인다. 0.1을 더 줄여보면, 이제 가우스와 거의 비슷한 형태를 뜨고 있다. 0.1을 활용하면 된다. 그럼 0.05로 더 줄여보면, 이제 완전 가우스처럼 보인다. 이 새로운 feature를 $x_{\text{NEW}} = x.^{0.05}$; 라고 정의할 수 있다. 이제 새로운 feature x 는 이전 히스토그램 보다 더 가우스와 비슷한 형태를 띄게 되었고, 이를 대신해 새로운 feature를 이상 탐지 알고리즘에 활용할 것이다.

05:10

물론 다른 방법도 있다. $Hist(\log(x), 50)$ 을 입력하여 이와 같은 변환 예시로 활용할 수 있다.

이것도 매우 가우스처럼 생겼다. 따라서 $x_{\text{NEW}} = \log(x)$; 와 같이 정의할 수 있고, 이는 활용하기 좋은 feature 선택의 예가 될 수 있다. 정리하자면, 데이터를 히스토그램으로 그렸는데, 가우스와 꽤 다른 형태를 띠고 있다면, 이와 같이 학습 알고리즘에 적용하기 전에 약간의 변형을 주어 데이터가 가우스처럼 보일 수 있게 작업을 해볼 필요가 있다. 물론 하지 않아도 구현이 되기는 할 것이다. 하지만 나는 보통 이러한 작업을 거친다.

05:45

이제 두 번째로, 이상 탐지 알고리즘에서 활용할 feature를 어떻게 생각해내는지 알아보자. 나는 보통 오류 분석 과정을 통해서 하는데, 이는 지도 학습에서 배운 오류 분석 과정과 흡사한데, 여기서 우리는 완성된 알고리즘을 학습하고, 이를 교차 검증 세트에서 구현하고, 예시를 보고 만약 잘못되었을 경우, 추가 feature를 적용하여 알고리즘이 교차 검증 세트에서 오류가 난 예시에 잘 구현될 수 있도록 하는 과정이다. 그럼 이러한 과정을 예시를 통해 살펴보자. 이상 탐지에서 우리는 $p(x)$ 가 정상 예시보다는 크고, 이상 예시보다는 작기를 바란다. 여기서 흔히 발생하는 문제는 $p(x)$ 가 정상과 이상 예시를 비교할 수 있을 만큼 충분한 값을 가지고 있느냐 것이다.

06:43

예시를 좀 더 자세히 살펴보자. 여기 분류되지 않은 데이터가 있다. 여기에 x_1 feature를 가지고 여기에 가우스를 적용해보겠다. 그럼 내가 데이터에 적용한 가우스는 이와 같이 구현될 것이다. 그리고 이상 예시가 있는데, 여기서 $p(x) = 2.5$ 라고 하자. 그럼 이렇게 이상 예시를 적용한다. 이는 정상 예시가 많이 밀집되어있는 곳 가운데 놓여지게 되는데, 이 이상 예시는 초록색으로 그렸고, 파란 곡선의 높이로 보니 꽤 높은 확률인데, 이 알고리즘은 이를 이상 예시로 표시하는데 실패할 것이다. 만약 이게 비행기 엔진 예시였다면, 학습 예시를 보고 특정 비행기 엔진에서 고장 난 부분이 무엇인지를 살펴보고 이 예시를 통해 새로운 x_2 feature를 생각해내서 나머지 빨간색 예시의 정상 비행기 엔진과 다른 이상 예시를 구분해 낼 수 있는지 알아볼 것이다.

07:52

그렇다면 이런 작업을 거쳐서 새로운 feature x_2 를 생성해내어 데이터를 다시 그려보면, 학습 세트에 있는 모든 정상 예시가 이와 같은 빨간 x 로 나타난 것을 발견할 수 있다. 그리고 이 것을 이상 예시라고 한다면, feature x_2 는 특이한 값을 가질 것이다. 따라서 여기 초록색 예시는 이상 예시인데, x_1 값은 여전히 2.5이다. 그리고 x_2 값은 여기에 이렇게 3.5 값이든지 아니면 매우 작을 값을 가질 것이다. 하지만 해당 데이터를 모델화하면 이상 탐지 알고리즘이 데이터의 중간부분에 높은 확률을 보여주고, 이렇게 점점 낮은 확률을 보여준다는 것을 발견하게 된다. 따라서 이 밖에 있는 예시의 경우, 이 알고리즘은 매우 낮은 확률을 측정할 것이다. 따라서 이와 같은 작업은 이 알고리즘이 오류를 일으키는 것을 확인할 수 있다. 이 알고리즘이 이상이라고 표기하기 실패한 것을 보고, 이를 할 수 있는 새로운 feature를 생성하도록 하는 것이다. 따라서 비행기 엔진에서 이상이 발견되면, 이를 새로운 feature로 생성하여, 이 새로운 feature가 정상 예시 중에서 이상 예시를 구분하기 쉽게 만들어준다. 따라서 오류 분석 과정은 이상 탐지를 할 새로운 feature를

만드는데 활용된다.

09:17

마지막으로 이상 탐지에서 feature를 선택하는 방법에 시행할 때 나의 의견을 이야기해주도록 하겠다. 나는 보통 feature를 선택할 때, 매우 큰 값 아니면 매우 작은 값을 선택하는데, 이 값은 이상 예시가 될 것이다. 예를 들어 데이터 센터의 컴퓨터 모니터링을 다시 얘기해보면, 데이터 센터에 아마 수천, 수만 개의 기계가 있을 것이다. 그 중에 기계 하나가, 컴퓨터 하나가 이상하게 작동하는지를 알고 싶다. 따라서 이 예시에서 선택할 수 있는 feature는 사용된 메모리, disc 접근 횟수, CPU 로딩, 네트워크 트래픽 등이 있다. 이번에는 이 오류 사례 중 하나를 감지하려고 하는데, 내 데이터 세트에 CPU 로딩과 네트워크 트래픽이 선형으로 증가하고 있다고 하자. 만약 너무 많은 웹서버를 구현하고 있고, 여기에 유저가 많은 서버가 있고, CPU로딩도 높고, 네트워크 트래픽도 높다고 하자.

10:20

하지만 컴퓨터 중 하나가 무한 루프에 빠지게 되었다고 의심이 든다고 하자. 따라서 기계 중 하나가 고장이 났다고 하면, 웹 서버, 서버 코드가 무한 루프에 빠지게 되었다면, CPU 로딩이 증가하게 되지만, 네트워크 트래픽은 그렇지 않은데, 이는 그저 선회하기만 하기 때문이다. 따라서 CPU 작업량이 많아지게 되고, 이는 무한 루프에 빠지게 된다. 그럴 경우, 이상의 형태를 감지하기 위해서, 새로운 feature x_5 를 생성할 건데, $x_5 = \text{CPU 로딩} / \text{네트워크 트래픽}$ 이 된다. 따라서 기계 중 하나가 CPU 로딩이 매우 크지만 네트워크 트래픽은 그렇게 크지 않으므로 x_5 는 엄청 큰 값을 가지게 되는데, 따라서 이 feature는 특정한 형태의 이상 탐지를 잡아내는데 도움을 줄 것이다.

11:15

또한 다른 feature도 생각해 볼 수 있는데, $x_6 = \text{cpu load}$ 의 제곱 / 네트워크 트래픽이다. 이는 x_5 와 같은 CPU 로딩이 높고, 네트워크 트래픽은 이에 비례하여 작은 기계의 이상을 잡아내는 또 다른 feature의 변수가 될 수 있다. 이와 같은 feature를 생성하면 특이한 feature 값이 결합을 보여주는 이상 예시를 잡아내는 작업을 착수할 수 있다.

Q) 이상 탐지 알고리즘이 잘 구현되지 않아 교차 검증 세트에 많은 정상 예시 $p(x)$ 와 이상 예시 $p(x)$ 를 출력했다. 다음 중 어떠한 변화를 줘야 알고리즘의 성능을 향상시킬 수 있는가?

1. feature의 수를 줄인다.
2. 정상 예시와 이상 예시를 구분할 수 있는 feature를 더 생성한다.
3. $P(x)$ 를 적용할 수 더 큰 (정상 예시) 학습 세트를 얻는다.
4. ϵ 값을 변경한다.

정답 2번

11:50

이번 시간에는 feature를 정하여 이상 탐지 알고리즘에 적용하기 전에 이를 약간 변환하여 가우스와 비슷한 형태를 만드는 방법과 다른 형태의 이상 예시를 탐지하기 위해 feature를 생성하는 과정에 있는 오류 분석에 대해 살펴봤다. 이와 같은 가이드라인을 통해 이상 탐지 알고리즘에 좋은 feature를 선택해서 모든 형태의 이상 예시를 감지하는데 도움이 되었으면 한다.

No.	Title
Week 9-7	Multivariate Gaussian Distribution

00:00

이번 강의와 다음 번 강의에서 우리가 지금까지 개발해온 이상 감지 알고리즘에 사용 가능한 확장자에 대해 이야기하고자 한다. 이 확장자는 다변량 가우스 분포를 활용하는데, 이는 장단점도 있고, 초기의 알고리즘이 잡아내지 못한 이상 예시도 감지할 수 있다. 먼저 예시를 살펴보자. 분류되지 않은 데이터를 다음과 같이 그려봤다. 데이터 센터에서 모니터링 기계, 모니터링 컴퓨터 예시를 활용하겠다. 여기서 feature가 두 개 있는데, x_1 은 CPU 로딩, x_2 는 메모리 사용이다. 여기 x_1, x_2 feature를 가우스로 표현하면, 여기 x_1 feature 그래프, x_2 feature 그래프가 있다. 여기에 가우스를 적용하면, 이와 같이 그려질 것이고, 여기는 $p(x_1; \mu_1, \sigma_1^2)$, 메모리 사용은 $p(x_2; \mu_2, \sigma_2^2)$ 가 될 것이다. 이것이 바로 이상 탐지 알고리즘이 x_1, x_2 를 모형화한 것이다.

01:19

이제 테스트 세트에서 이와 같은 예시가 있다고 하자. 여기 초록색 x 의 위치는, x_1 값은 0.4 정도 되고, x_2 는 1.5정도 된다. 데이터를 보면, 거의 대부분의 데이터가 이 지역에 밀집되어있고, 초록색 x 는 이 데이터들과 멀리 떨어져있는 것을 볼 수 있다. 이는 이상 예시라고 볼 수 있는 것이다. 데이터에서 정상 예시는 CPU 로딩, 메모리 사용에서 함께 선형으로 증가하는 것을 볼 수 있다. 따라서 많은 CPU를 활용하는 기계를 가지고 있다면, 메모리 사용도 높을 것이고, 반면에 이 초록색 예시에는, CPU 로딩이 매우 낮은데, 메모리 사용은 매우 높다. 따라서 이는 학습 세트에서 보지 못한 예시다. 따라서 이는 이상 예시처럼 보인다. 이럴 때 이상 탐지 알고리즘이 어떻게 되는지 보자.

02:15

CPU 로딩에서는 0.5 정도 가 되니까 다른 예시들과 그렇게 멀리 떨어져 있지 않고, 높은 확률을 보여주고 있다. 반면에 메모리 사용에 있어서는 1.5정도가 되는데, 여기쯤이다. 이는 그렇게

이상한 가우스는 아니지만, 여기 값은 우리가 봤던 다른 예시와 크게 다르지 않게 된다. 따라서 $p(x_1)$ 은 꽤 높은 값을 가지게 된다. $P(x_2)$ 도 마찬가지다. 여기 오른쪽 그래프를 보면, 여기 이 지점은 그렇게 달라 보이지 않고, 여기서도 그렇게 달라 보이지 않는다. 즉, 더 많은 메모리를 사용하는 예시도 있고, 더 적은 CPU를 활용하는 예시도 있는데, 이러한 예시는 이상 예시로 보이지 않기 때문이다. 따라서 이상 탐지 알고리즘은 이 지점을 이상이 있다고 표시하는데 실패하는 것이다.

03:06

이 이상 탐지 알고리즘은 이 파란색 타원 높은 확률 부분이고, 여기에 있는 예시는 확률이 높고, 여기 이 원에는 확률이 높고, 다음 원에 들어갈 예시는 더 낮은 확률이고 그 다음은 더 낮아지게 된다. 따라서 여기 초록색 x 표시된 부분은 꽤 확률이 높은 편이다. 이 구역에 있는 모든 예시는 이 거의 같은 확률이라고 간주한다. 이 정도에 있는 예시가 훨씬 더 낮은 확률이라고 생각하지 않는다. 이를 해결하기 위해, 이상 탐지 알고리즘의 변형된 버전을 개발해보겠는데, 이는 다변량 가우스 분포 또는 다변량 정규 분포라고 한다. 방법은 다음과 같다.

04:09

Feature $x \in \mathbb{R}^n$ 이 있을 때, $p(x_1), p(x_2)$ 이렇게 별개 예시 대신에 $p(x)$ 로 한번에 연산하려고 한다. 따라서 다변량 가우스 분포의 매개 변수는 $\mu \in \mathbb{R}^n$ 는 벡터고, $\sigma \in \mathbb{R}^{nxn}$ 는 $n \times n$ 행렬인데, 이는 공분산 행렬로 이는 우리가 PCA(주성분분석)를 다룰 때 봤던 공분산 행렬과 비슷하다. 이제 다변량 가우스 분포의 공식을 써보자. $p(x; \mu, \Sigma)$ 는 다음과 같은 공식과 같은데, 이 공식을 외울 필요는 없다. 여러분이 필요할 때 찾아서 가져다 쓰면 되는 것이다. 여기 시그마의 절대값은 이러한 기호를 쓰는데, 이는 시그마의 제지(determent)라고 하고 이는 행렬의 수학적 기능을 나타내고, 행렬의 제지가 무엇인지는 알 필요가 없다. 그저 이를 옥타브에서 옥타브 명령어 $\det(\Sigma)$ 를 활용해 연산할 수 있다는 것만 기억하면 된다. 다시 한번 짚고 넘어가면, 여기 이 시그마는 $n \times n$ 행렬을 말한다. 여기서는 합을 나타내는 것이 아니라 그저 $n \times n$ 행렬을 말한다.

05:47

지금까지 $p(x)$ 공식에 대해 알아봤다. 하지만 더 흥미롭고 중요한 사실은, 이 $p(x)$ 가 어떻게 그래프로 표현되느냐이다. 이제 다변량 가우스 분포의 예시를 살펴보자. 여기 2차원 예시가 있다. $N=2$ 이고, feature 는 x_1, x_2 두 개다. $\mu = [0; 0]$ $\Sigma = [1 0; 0 1]$ 이라고 하자. 여기 대각선으로는 1이 있고, 나머지는 0인 항렬은 항등 항렬이라고 부른다. 이 경우에, $p(x)$ 는 다음과 같이 그릴 수 있는데, 이 그래프에서 특정 x_1 값과, 특정 x_2 값이 만날 때, 이 표면에서부터 높이는 $p(x)$ 값이 되는 것이다. 따라서 이와 같은 설정에서, x_1, x_2 가 0일 때 매개 변수 $p(x)$ 는 가장 높은 값을 가지는데, 이 것이 가우스 분포의 정점이 되는 것이고, 여기서부터 시작해서 이와 같은 2차원 가우스 또는 종모양 2차원 표면도에서 확률은 점점 낮아지는 것이다.

06:55

아래 그림은 같은 내용이지만, 등고선 그래프에 다른 색상을 활용하여 그린 것이다. 여기 보면

중앙은 짙은 빨간색으로 그려져 있고, 이는 가장 높은 값을 의미한다. 그리고 그 값은 줄어들면서 노란색으로 표현되어있고, 좀 더 내려가면 청록색, 더 내려가면 이렇게 짙은 파란색으로 되어있다. 이는 같은 수치를 표현한 그래프이지만 꼭대기에서 내려다 본 모습을 다른 색으로 표현한 것이다. 또한 이러한 분포에서 $(0,0)$ 을 중심으로 가장 확률이 높고, 그리고 이 원점에서 멀어질수록 x_1, x_2 확률이 점점 낮아진다. 이번에는 매개 변수를 다양하게 해보고 어떤 변화가 있는지 살펴보자.

07:39

먼저 시그마를 가지고 변화를 줘보는데, 시그마 값을 조금 줄여보자. 시그마는 공분산 행렬인데, 이는 feature x_1, x_2 의 변수 또는 변동성을 측정한다. 이와 같이 시그마 값이 줄었을 때, 이 범프의 너비가 줄어들고, 높이는 약간 늘어난다. 이는 표면 아래의 지역은 1이고, 즉 표면 아래의 적분양은 1이기 때문에, 확률 분포는 1로 통합되어야 하기 때문이다. 하지만 변수를 줄이면, 이는 시그마의 제곱을 줄이는 것과 같기 때문에, 더 좁은 분포를 가지게 되고 범프가 조금 높아지는 것이다. 여기도 보면 동심 등고선이 약간 줄어든 것을 볼 수 있다. 반면에 이번에는 시그마를 대각선에 2, 2값을 가지는 행렬로 증가시키면, 이제는 두 배로 넓고 평평해진 가우스를 얻게 되는 것이다. 너비는 훨씬 넓어지게 되고, 잘 안보이지만 여전히 종 모양 범프이고, 아래로 마니 평평하게 되어서 훨씬 넓어졌고, x_1, x_2 의 변수와 변동성은 더 넓어지게 되었다.

08:49

여기 몇 가지 예시가 더 있다. 이번에는 시그마의 원소 하나를 변형시켜보자. 이처럼 여기는 0.6 여기는 1이 된다. I 는 첫 번째 feature x_1 의 변수를 줄여주고, 두 번째 feature x_2 의 변수는 그대로 유지한다. 따라서 이러한 매개 변수 설정으로 이와 같은 모델을 얻을 수 있다. X_1 의 변수는 더 작아졌고, x_2 의 변수는 더 커졌다. 반대로, 행렬에 2, 1 이렇게 값을 주면, 이와 같은 예시를 얻을 수 있는데, 여기서는 x_1 의 값의 범위가 넓어지고, 반면에 x_2 의 값은 상대적으로 좁아지게 된다. 이는 이 그래프에도 반영되어 있는데, 분포가 x_1 이 0에서 멀어질수록 천천히 낮아지고, x_2 이 0에서 멀어질수록 급격하게 낮아진다.

09:50

마찬가지로 이전 슬라이드처럼 행렬의 원소값을 바꿔주려고 하는데, 여기서는 x_2 가 값이 매우 작은 0.6 이었고, x_2 에 원래 예시보다 작은 값의 범위를 적용했었는데, 이번에는 시그마값을 2라고 하면, x_2 는 훨씬 더 큰 값의 범위를 가지게 된다. 다변량 가우스 분포에서 장점은 이들 데이터 사이의 상관 관계를 활용할 수 있다는 것이다. 즉, x_1 과 x_2 가 예시로서 서로 상관 관계가 있다는 사실을 모형화 할 수 있다는 것이다. 예를 들어, 먼저 이 공분산 행렬에서 반대 대각선의 수를 바꾸기 시작하면, 다양한 형태의 가우스 분포를 얻을 수 있게 된다. 따라서 반대 대각선의 수를 0.5에서 0.8로 변화를 주면, 이 분포는 $x = y$ 라인을 따라 그 꼭대기가 점점 가늘어지는 것을 볼 수 있다. 여기 등고선에서는 x 와 y 는 함께 커지는 데 확률이 높은 것에서, x_1 의 값이 크면 y_2 값이 커지고 아니면 x_1 이 작아지면, y_2 도 작아진다. 아니면 그 중간에 있게 된다. 여기 숫자가 0.8로 커지면서, 이와 같은 x 가 약 y 와 같은 값인 좁은 지역에 모든 확률이 모인 가우스 분포를 얻게 된다. 이렇게 높고 좁게 이 선을 따라서 중앙 지점에 분포하게 되고, x 와 y 값이 거의

같은 지점이다.

11:36

이 세 예시는 positive 예시를 가지고 그래프화 한 것이다. 반대로 negative 값을 설정하면, 이는 -0.5에서 -0.8로 감소한다. 이 경우 negative x_1 과 negative x_2 가 상호작용하는 부분에 대부분의 확률이 들어가고, 대부분의 확률이 $x_1 = x_2$ 가 아니라 $x_1 = -x_2$ 가 되는 구역에 포함되어 있다. 따라서 x_1, x_2 사이의 negative 상호 작용을 보여주고 있다. 따라서 다변량 가우스 분포가 보여주는 다양한 분포에 대해 이해할 수 있기를 바란다.

12:19

다변량 행렬 시그마를 바꿔주는 내용에 이어서 또 다른 방법은 평균 매개 변수 μ 값을 달리하는 것이다. 원래 $\mu = [0; 0]$ 값을 가지고 있었고, 따라서 $x_1 = 0, x_2 = 0$ 인 지점을 중심으로 분포가 이루어졌다. 여기 분포의 정점이 있고, 반면에 μ 값에 변동을 주면, 이는 분포의 정점에도 변화를 주게 된다. 따라서 $\mu = [0; 0.5]$ 일 때, 정점은 $x_1 = 0, x_2 = 0.5$ 인 지점이다. 그리고 이와 같이 분포의 중심인 정점이 이동하게 된다. 또 $\mu = [1.5; -0.5]$ 가 되면, 또 분포의 정점은 다른 곳으로 이동하게 되고, 여기서 $x_1 = 1.5, x_2 = -0.5$ 가 된다. 따라서 μ 값을 달리해주는 것은 전체 분포의 중심을 이동시키는 것뿐이다.

Q) 다음과 같은 다변량 가우스 분포가 있다.

이러한 분포에서 μ, Σ 값은?

정답 3번

13:20

이와 같이 다양한 그래프를 통해 다변량 가우스 분포가 보여주는 확률 분포에 대해 이해할 수 있게 되었기를 바란다. 여기서 가장 큰 장점은 두 개의 다른 feature를 가지고 positive 또는 negative로 상호작용하고 있는 것을 발견할 수 있다는 것이다. 다음 시간에는 다변량 가우스 분포를 가지고 이를 이상 탐지에 적용해보도록 하겠다.

No.	Title
Week 9-8	Anomaly Detection using the Multivariate Gaussian Distribution

00:00

지난 시간에 다변량 가우스 분포와 매개 변수 μ, Σ 값에 변화를 주어 다변량 가우스 분포 변화를 예시를 통해 알아봤다. 이번 시간에는 이러한 내용을 바탕으로 다양한 이상 탐지 알고리즘을

개발하는 데 적용해보자. 다변량 가우스 분포 또는 다변량 정상 분포의 개요를 다시 말하자면, 두 개의 매개 변수 μ , Σ 가 있다. 여기서 $\mu \in \mathbb{R}^n$ 이고, $\Sigma \in \mathbb{R}^{n \times n}$ 이다. 그리고 여기 p 의 확률은 다음과 같이 μ , Σ 로 매개변화 된다. 그리고 μ , Σ 값에 변화를 주면, 다양한 분포 범위를 얻을 수 있게 된다. 여기 지난 시간에 살펴본 세 가지 예시가 있다. 이제 매개 변수 대입 또는 매개 변수 추정 문제에 대해 알아보자.

00:56

문제는 예시 세트가 x_1 부터 x_m 까지 있을 때, 여기 각각 예시는 n 차원 벡터이고, 이 예시는 다변량 가우스 분포에서 얻을 수 있다. 그렇다면 매개 변수 μ , Σ 를 어떻게 추정할 수 있는가? 먼저 이들을 추정하는 표준 공식은 다음과 같이 학습 예시의 평균으로 μ 를 놓는 것이고, 이것과 Σ 가 같다고 하는 것이다. 이는 사실 우리가 PCA(주성분분석)에서 활용한 시그마와 같은 것이다. 따라서 이 두 공식을 통해 매개 변수 μ , Σ 값을 추정할 수 있는 것이다. 그렇다면 이와 같은 데이터 세트에서는 μ , Σ 값을 어떻게 추정할까? 이 방법을 통해 이를 이상 통제 알고리즘에 적용하면 된다. 그렇다면 이 모든 것을 어떻게 적용하여 이상 탐지 알고리즘을 개발할 수 있을까? 그 방법은 다음과 같다.

01:56

먼저, 학습 세트를 가지고 모델 $p(x)$ 를 이전 슬라이드에서 봤던 μ , Σ 에 공식에 대입하는 것이다. 그 다음으로 새로운 예시 x 가 있을 때, 즉 테스트 예시가 있을 때, 이전 예시를, 새로운 예시를 적용해본다. 이게 테스트 예시가 된다. 이제 새로운 예시 x 를 가지고, $p(x)$ 를 연산해볼 건데, 여기 다변량 가우스 분포의 공식을 활용할 것이다. 만약 $p(x)$ 값이 매우 작다면, 이를 이상 예시로 표시할 것이다. 반면에 $p(x)$ 값이 매개 변수 ϵ 보다 크다면 이를 이상 예시라고 표시하지 않는다. 이는 우리가 다변량 가우스 분포를 이 데이터에 적용하면, 초록색 예시를 제외한 빨간 x 표만 있을 텐데, 가우스 분포는 여기 중앙 부분에 위치할 확률이 높고, 이렇게 중앙에서 멀어질수록 그 확률은 조금씩 낮아질 것이고, 이 부분에서는 굉장히 낮을 것이다.

03:01

따라서 다변량 가우스 분포를 이 예시에 적용하면, 이 예시를 정확하게 이상 예시로 표시할 것이다. 다음으로 다변량 가우스 분포 모델과 원모델 사이의 관계에 대해 이야기를 하자면, $p(x)$ 는 $p(x_1), p(x_2), \dots, p(x_n)$ 까지의 곱의 결과물인데, 이를 수학적으로 증명할 수 있다. 여기서 이를 증명하지는 않겠지만, 다변량 가우스 분포 모델과 원모델 사이의 관계에 대해서 수학적으로 증명이 가능하다는 것이다. 특히, 원모델은 다변량 가우스 분포와 대응하여 가우스 분포의 등고선은 항상 축을 따라 정렬되어 있다. 따라서 여기 가우스 분포의 예시 세 개를 모두 원모델을 활용하여 적용할 수 있다. 이는 다변량 가우스 분포와 대응하여 여기 타원은 분포의 등고선을 나타내는데, 이는 이 모델이 다변량 가우스 분포의 특별한 케이스와 대응한다는 것이다. 특히, 이 특별한 케이스는 $p(x)$ 의 가변량 가우스 분포를 제한함으로써 정의된다. 따라서 확률 밀도 함수, 확률 분포 함수의 등고선은 축을 중심으로 배열된다. 그래서 다변량 가우스 함수 $p(x)$ 를 얻게 되는데, 이는 다음 세 그래프와 같다.

04:44

이 예시 세 개에서 지금 그리고 있는 이 타원은 x_1, x_2 축을 따라 정렬되어있는 것을 알 수 있다. 지금 우리에게 없는 건 이러한 각도의 등고선이다. 이와 같은 예시는 시그마 값이 $[1, 0.8 ; 0.8, 1]$ 이렇게 반대쪽 대각선이 0이 아닌 이와 같은 행렬과 대응한다. 이 모델이 다변량 가우스 분포지만 제한된 형태라는 것을 수학적으로 증명할 수 있음을 발견했다. 그리고 여기서 제한은 공분산 행렬 시그마는 반대쪽 대각선에 0이 있어야 한다. 특히 여기 공분산 행렬 시그마에서 이렇게 대각선이 $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$ 이와 같고, 그 외의 부분에서는 여기 위쪽과 아래쪽은 0이어야 한다. 만약 이 시그마 값을 $\sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$ 여기에 공분산 행렬에 대입하면, 이 두 모델은 똑같아진다. 이 새 모델이 다변량 가우스 분포를 활용하여 원모델과 대응하여, 만약 공분산 행렬 시그마가 대각선을 제외한 부분에 0만을 포함한다면, 이는 가우스 분포와 대응하고, 이 분포 함수의 등고선에서는 축을 중심으로 정렬되어있다. 따라서 이 다른 두 feature 사이에 상호작용을 모델화할 수는 없다.

06:30

따라서 원모델은 사실 다변량 가우스 모델의 특별한 케이스라는 말이다. 그렇다면 이 모델을 각각 언제 활용하는 것인가? 원모델은 언제 활용하고, 다변량 가우스 모델은 언제 활용하는가? 이전 모델이 더 자주 활용되는 반해, 다변량 가우스 분포는 덜 활용된다. 하지만 feature 간의 상호작용을 캡쳐할 수 있는 장점을 가지고 있다. 따라서 다른 두 예시 x_1, x_2 이 있을 때, 이전 예시처럼 독특한 결합을 보여줄 때, 이상 예시를 캡쳐하려고 한다. 이전 예시에서 CPU 로딩과 메모리 사용에서 이상이 있는 것을 살펴봤었다. 원모델을 활용하여 캡쳐하려고 할 때, 이 둘이 특이한 결합을 가지게 된다면 feature $x_3 = x_1/x_2$ 와 같은 추가 feature를 생성하는 것이다. 이는 CPU 로딩 / 메모리 사용과 같을 것이다. 만약 x_1, x_2 값이 각각 정상 값으로 보여도 이 둘이 특이한 결합을 가지게 된다면 추가 feature를 생성해야 한다.

08:01

하지만 수동으로 이와 같은 추가 feature를 생성하고자 한다면, 원모델은 잘 구현될 것이다. 반면에 다변량 가우스 모델은 각기 다른 feature 사이의 상호 관계를 자동으로 캡쳐할 수 있다. 하지만 원모델은 더 중요한 장점을 가지고 있는데, 그 중 하나는 연산하는 게 저렴하다는 것인데, 이는 즉 많은 양 n개의 값, 그리고 더 많은 양의 feature를 더 잘 측정한다는 건데, n이 10,000 개 또는 100,000개 일지라도 원모델은 잘 구현될 것이다. 반면에, 다변량 가우스 모델은 이 부분을 살펴보면, 시그마 행렬의 역을 연산하려고 할 때, 여기서 시그마는 $n \times n$ 행렬이고, 만약 이 시그마가 $100,000 \times 100,000$ 행렬이라면, 이는 연산하기 시간이 오래 걸릴 것이다. 그리고 다변량 가우스 모델은 n개의 수가 많아지면 잘 구현되지 않는다.

09:08

원모델은 또한 상대적으로 작은 학습 세트에서도 잘 작동된다. 이것은 $p(x)$ 모델에 활용하는 분류되지 않은 작은 예시인데, m값이 5,000 이 정도로 작아도 잘 구현된다. 반면에 다변량 가우스

분포에서는, 이 알고리즘은 $m > n$ 과 같은 수학적 특징을 가지고 있어서, 예시의 수가 가지고 있는 feature의 수보다 많아야 한다. 우리가 매개변수를 추정하는 수학적 특징을 활용하는 방법이 있는데, 만약 이것이 참이 아니고 $m \leq n$ 이라면 이 행렬은 가역 할 수 없고, 이는 또한 비정칙(singular) 행렬이기 때문에 여기에 어떠한 변동을 주지 않는 이상 다변량 가우스 모델을 활용할 수 없다. 하지만 경험상 나라면 m 의 값이 n 보다 훨씬 큰 경우에만 다변량 가우스 모델을 활용할 것이다. 따라서 이는 한정된 수학적 조건인데, 실제로 m 이 n 보다 약간 큰 경우에 다변량 가우스 모델을 활용했다.

10:10

따라서 $m \geq 10n$ 이라면, 이는 합리적인 경험 법칙이 될 것이고, 만약 이를 충족하지 못할 경우, 다변량 가우스 모델은 매개 변수가 많은데, 공분산 행렬 시그마는 $n \times n$ 행렬이고, 이는 대략 n 제곱 매개 변수를 가지고 있는데, 이는 대칭 행렬이고, $\frac{n^2}{2}$ 매개변수에 근접하기 때문이다. 하지만 이는 많은 매개 변수이기 때문에, m 의 값이 많이 있는지, 모든 매개 변수를 대입할 수 있는 충분한 데이터가 있는지 확인해야 한다. 그리고 $m \geq 10n$ 은 이 대변량 행렬 시그마를 꽤 괜찮게 추정할 수 있다는 것을 확인시켜주는 합리적인 경험 법칙이다. 따라서 실제로 왼쪽에 있는 원모델이 더 자주 활용된다는 것을 알 수 있다. 만약 이게 의심스럽다면, 여러분은 직접 feature 사이의 상호 관계를 캡쳐해야 한다. 사람들은 보통 수동으로 이와 같은 추가 feature를 생성해서 값의 특정한 이상 결합을 캡쳐한다.

11:10

하지만 많은 학습 예시 세트를 가지고 있거나 m 값이 매우 크고, n 값이 그렇게 크지 않을 때, 다변량 가우스 모델은 고려해 볼만한 방법이고, 실제로 더 잘 구현될 수도 있다. 그리고 값의 특정한 이상 결합을 찾기 위해 수동으로 추가 feature를 생성하느라 시간을 보내지 않아도 된다. 이번에는 기술적인 속성에 대해 알아보겠는데, 다변량 가우스 모델을 대입할 때, 공분산 행렬 시그마가 비정칙일 때, 또는 이 것을 가역 할 수 없을 때, 이를 해결하는 방법 두 가지가 있다. 하나는 $m > n$ 조건을 만족하는 것이고, 다음은 여분의 feature를 가지고 있다면, 여기서 여분의 feature는 같은 feature 2개가 있는 것을 말한다. 어쩌다 보니 똑같은 feature 2개가 있게 되어, $x_1 = x_2$ 이다. 또는 $x_3 = x_4 + x_5$ 가 될 수도 있다.

12:15

그렇다면, 이와 같이 여분의 feature가 있다고 할 때, $x_3 = x_4 + x_5$ 여기서 x_3 에 대한 어떠한 추가 정보도 없는 상태다. 그럼 여기 feature 2개를 가지고 더해준 것뿐이다. 이와 같은 여분 feature, 복제 feature 에서 시그마 행렬은 가역 할 수 없게 된다. 따라서 여기 디버깅 세트가 있다. 그전에 이는 거의 발생하지 않아서, 아마 발견하지 않을 가능성이 높고, 이에 대해 걱정하지 않아도 되지만, 혹시나 다변량 가우스 모델을 구현하게 된다면, 시그마 행렬은 가역 할 수 없다. 먼저, $M > N$ 을 충족하는지 확인한다. 그 다음으로는 여분 feature를 확인한다. 그리고 같은 feature 2개가 나온다면, 그 중 하나를 삭제하거나 $x_3 = x_4 + x_5$ 같은 여분 feature가 있다면, 이 여분 feature를 삭제하면, 다시 잘 작동할 것이다.

13:08

선형 대수학에 대해 잘 알고 있는 사람들을 위해 덧붙이자면, 여기 여분 feature에서 feature를 가리키는 공식 용어는 선형종속(linearly dependent)이다. 하지만 실제 의미는 이러한 문제 중 하나가 알고리즘에서 발견되면, feature의 여분이 생기지 않도록 하라는 것이다. 그러면 시그마 행렬을 가역 할 수 없는 문제를 해결할 수 있게 된다. 하지만, 다시 말하지만 이와 같은 일이 발생할 경우는 희박하기 때문에, 시그마를 가역 할 수 없을 것을 걱정하지 않고 $m \geq n$ 이라면, 다변량 가우스 모델을 적용하면 된다.

Q) $x(i) \in \mathbb{R}^n$ 일 때, 학습 세트 $x(1) \dots x(m)$ 을 활용하여 이상 탐지에 적용해보려고 한다. 다음 설명 중 옳은 것은? (중복 선택 가능)

- 원 모델은 다변량 가우스 모델과 대응하고, 이 등고선 그래프는 축을 중심으로 정렬된다.
- 다변량 가우스 모델은 m (학습 세트 사이즈)이 매우 작은 경우($m < n$)에 활용할 수 있다.
- 다변량 가우스 모델은 x 에 다른 feature 사이의 상호 관계를 자동으로 캡쳐할 수 있다.
- 원모델은 다변량 가우스 모델에 비해 연산이 효율적인데, n (feature의 수)의 값이 클 때 더 잘 구현된다.

정답 1,3,4번

지금까지 다변량 가우스 분포를 가지고 이상 탐지에 대해 살펴봤다. 이러한 방법을 적용하여 이상 탐지 알고리즘을 얻을 수 있게 된다면, 자동으로 다른 feature 사이의 positive 와 negative 의 상호관계를 자동으로 캡쳐 할 수 있게 되고, feature 값의 특이한 결합이 발견되면 이를 이상 예시로 표시할 수 있다.

No.	Title
Week 9-9	Problem Formulation

00:00

이번 강의부터는 추천 시스템(recommender system)에 대해 이야기해보도록 하겠다. 추천 시스템에 대해 강의를 하는 이유는 두 가지다. 첫 번째는 이는 기계 학습에서 중요한 어플리케이션이기 때문이다. 지난 몇 년간 실리콘밸리에 있는 다양한 기술 회사에 방문하고는 했는데, 거기에서 기계 학습 어플리케이션을 연구하는 사람들과 이야기를 하면서, 기계 학습에서 가장 중요한 어플리케이션은 무엇인지 아니면 기계 학습 어플리케이션에서 그 성능을 가장 향상시키고 싶은 것은 무엇인지를 물어봤다. 실리콘밸리에서 일하는 사람들에게서 가장 많이 돌아온 답변 중

하나는 더 나은 추천 시스템을 개발하는 것이라고 했다. 아마존, 넷플릭스, 이베이, 애플이 만든 아이튠즈 지니어스 등과 같은 웹사이트와 시스템에서 새로운 제품을 사용해보라고 추천하고 있다. 아마존에서는 새로운 책을 추천하고, 넷플릭스는 새로운 영화를 추천할 것이다. 이러한 종류의 추천 시스템은 여러분이 어떠한 책을 이전에 구입했는지, 어떠한 영화에 별점을 매겼는지 살펴본다. 이러한 시스템은 아마존의 수익 중 상당한 부분을 차지하고 있다. 그리고 넷플릭스와 같은 기업도 추천 시스템으로 고객들이 영화를 보도록 하는데, 이는 고객들이 영화를 보게 하는 상당한 부분에 기여하고 있다.

01:19

따라서 추천 시스템의 성능을 향상 시키는 것은 이러한 기업들에게 잠재적인 그리고 즉각적인 영향을 미치는 핵심 요소가 될 것이다. 추천 시스템은 재미있는 문제인데, 학술 기계 학습은, 그러니까 학술 기계학습 컨퍼런스에서 추천 시스템의 문제는 상대적으로 주목을 거의 받지 못하고 있고, 적어도 학계에서는 그리 중요한 문제라고 여겨지지 않는다. 하지만 지금 돌아가는 사정을 살펴보면, 많은 기술 기업들에게 이러한 시스템을 개발하는 것은 그 기업의 우선 순위로 보여진다. 이것이 내가 추천 시스템을 강의하려고 하는 첫 번째 이유다.

01:58

추천 시스템에 대해 이야기하고자 하는 두 번째 이유는 지난 몇 개의 강의에서 기계 학습에 대한 큰 아이디어 몇 개를 이야기하며 공유하고 싶었다. 기계 학습에서 이슈화 되고 있는 이야기 말이다. 우리는 이 강좌에서 기계학습에서 feature가 얼마나 중요한지, 그리고 여러분이 선택한 feature가 학습 알고리즘의 성능에 얼마나 큰 영향을 미치는지에 대해 학습했다. 따라서 기계 학습에서 주요 이슈는, 어떤 문제에 있어서는, 모든 문제는 아니겠지만, 어떤 문제에서는 여러분을 위해 자동으로 좋은 feature 세트를 학습하도록 시도하는 알고리즘이 있다. 따라서 직접 feature를 설계하고 코드화하는 것 보다는 지금까지 우리가 해온 대부분은, 알고리즘을 얻을 수 있는 셋팅이 몇 개 있고, 어떠한 feature를 활용해야 할지 학습했다는 것이다. 여기서 추천 시스템은 그러한 셋팅의 한 예시일 뿐인 것이다.

02:48

다른 예시도 많이 있지만, 추천 시스템을 통해 새겨진 것은 feature를 학습하는데 아이디어를 제공해 줄 것이고, 이러한 예시의 최소 하나 정도는 알 수 있게 될 것이며, 기계 학습의 주요 이슈에 대해 알 수 있게 될 것이다. 이제 본론으로 넘어가서 추천 시스템 문제 형성에 대해 이야기해보자. 예시로서 영화 평점을 예측하는 현대적인 문제를 활용하도록 하겠다. 여기 문제가 있다. 여러분의 웹사이트나 기업에서 영화를 팔고 대여한다고 하자. 아마존, 넷플릭스 그리고 아이튠즈와 같은 모든 기업에서 이와 같이 하고 있는데, 유저들에게 다른 영화를 1부터 5까지 별점을 매기라고 해보자. 유저들은 1,2,3,4, 또는 별 다섯 개로 평가할 것이다. 더 나은 예시를 위해, 별점을 0부터 5까지라고 해보자. 보통 웹사이트에서 1부터 5까지 별점을 매기지만, 그저 산수가 편하라고 이렇게 한 것이다.

03:52

여기 영화 다섯 개가 있다. Love at Last, Romance Forever, Cute Puppies of Love, Nonstop Car Chases, 그리고 Swords vs. Karate. 이다. 여기 네 명의 유저가 있다. Alice, Bob, Carol, 그리고 이니셜 A, B, C, D 다. 유저 1,2,3,4라고 하겠다. 앤리스가 Love at Last와 Romance Forever 영화가 좋아서 별 점 5개를 주었다고 하자. 그녀는 아직 Cute Puppies of Love를 보지 않았기 때문에 아직 평점이 없고, Car Chases 와 Swords vs. Karate는 좋아하지 않았다. 다른 유저 2 Bob은 다른 세트의 영화를 평가했는데, Love at Last는 좋았고, Romance Forever는 아직 안 봤고, 나머지는 4, 0, 0 점으로 평가했다고 하자. 유저 3은 0, 이건 안 봤고, 0, 5, 5 라고 평가했고, 그냥 이렇게 숫자를 기입해보자.

04:51

여기서 사용할 표기를 설명하자면, 이 표기들을 쭉 활용할 것이다. N_u 는 유저의 숫자를 가리키므로 $N_u=4$ 가 된다. 따라서 여기서 u 는 user를 의미한다. N_m 은 영화의 수를 의미하고, 영화 다섯 편이 있으니 $N_m = 5$ 가 된다. 여기 예시를 대략 3개의 로맨틱 코미디 영화 그리고 나머지 둘을 액션 영화라고 분류했다. 여기 예시를 살펴보면, Alice와 Bob은 로맨틱 코미디에 높은 평점을 줬고, 액션 영화에는 매우 낮은 평점을 매겼다. Carol 과 Dave 반대의 경우다. 유저 3,4 는 액션 영화는 좋아해서 높은 평점을 줬지만, 로맨스 물은 별로 좋아하지 않았다.

05:49

예를 들어, 이러한 추천 시스템 문제에서 다음과 같은 데이터가 있다고 하자. 데이터는 다음의 내용을 포함하고 있다. 먼저 $r(i, j)$ 값이 있는데, 이는 만약 유저 j 가 영화 i 를 평가했다면, 1이 된다. 따라서 유저가 몇 편의 영화만 평가했다면, 여기 보면 몇 개의 평점은 없다. 따라서 $r(i, j) = 1$ 일 경우, 유저 j 가 영화 i 에 평점을 매기면, 우리는 이 숫자 $y(i, j)$ 를 얻게 되는데, 이는 유저 j 가 영화 i 에 매긴 평점이다. 따라서 $y(i, j)$ 의 숫자는 해당 유저가 특정 영화에 매긴 별 점에 따라 0부터 5사이가 될 것이다. 따라서 추천 시스템 문제는 이와 같은 데이터를 가지고, 이와 같이 $r(i, j)$, $y(i, j)$ 세트에서 모든 데이터를 보면서 아직 별 점이 매겨지지 않은 영화를 보고 이 둘을 대신에 들어갈 값이 무엇인지를 예측하는 것이다.

06:46

이와 같은 예시에서 영화 수가 몇 편 없고, 유저의 수도 적고, 대부분의 유저가 대부분의 영화에 평점을 매겼는데, 실제로 유저들이 평가한 영화는 그리 많지 않을 것이다. 하지만 이와 같은 데이터에서, Alice와 Bob이 모두 로맨스 영화를 좋아한다고 할 때, Alice는 이 영화에도 5점을 줄 수도 있다. Bob 또한 4.5점 또는 더 높은 점수를 줄 수 도 있다. 또한 Carol과 Dave는 이 영화에 매우 낮은 점수를 줄 것이다. Dave의 경우 액션 영화를 좋아했는데, 따라서 Swords and Karate에 4점 또는 5점을 줄 수도 있다.

07:22

따라서 추천 시스템을 개발하는 데 있어서 우리의 역할은 이와 같은 평점이 매겨지지 않은 값을

자동으로 채워 넣는 학습 알고리즘을 개발해내는 것이다. 이를 통해, 해당 유저가 아직 보지 않은 새로운 영화를 추천하는 것이다. 또한 유저가 재미있어할 영화가 무엇인지 예측하는 것도 필요하다.

Q) $r(i, j) = 1$ 일 때, 유저 j 가 영화 i 에 평점을 매겼고, $y(i,j)$ 는 유저가 영화에 매긴 평점을 가리킨다. 다음 예시를 보고 물음에 답하시오. (영화 수 = 2, 유저 수 = 3)

그렇다면, $r(2,1)$ 과 $y(2,1)$ 값은?

정답 3번

이와 같이 추천 시스템 문제 형성에 대해 알아봤다. 다음 시간에는 이러한 문제를 해결하기 위해 학습 알고리즘을 개발해보도록 하겠다.

No.	Title
Week 9-10	Content Based Recommendations

00:00

지난 시간에는 추천 시스템 문제에 대해 알아보면서 영화 집합과 사용자 집합 그리고 누가 어떠한 영화의 부분집합에 어떤 평점을 매겼는지에 대한 예시를 알아봤다. 유저들은 영화에 1-5점 또는 0-5점 사이의 평점을 매겼다. 그리고 우리는 유저들이 매긴 평점을 보고 그들이 아직 평가하지 않은 영화를 어떻게 평가할지 예측하고자 했다. 이번 시간에는 추천 시스템을 개발하는데 첫 번째 접근 방식에 대해 이야기하고자 한다. 이는 콘텐츠 기반 추천 이라고 부른다. 여기 지난 시간에 살펴본 데이터 세트가 있고 다시 한번 설명하자면, N_u 는 유저의 수 따라서, $N_u = 4$, N_m 은 영화의 수니까 $N_m = 5$ 가 된다. 그렇다면 여기 매겨지지 않은 값은 어떻게 예측할 수 있는가?

00:52

먼저 영화 각 편에 적용할 feature 세트를 준비한다. 이와 같이 각 영화마다 feature가 두 개씩인데 이를 x_1, x_2 라고 표기하겠다. 여기서 x_1 은 로맨스 영화인지의 정도를, x_2 는 액션 영화인지의 정도를 측정한다. 따라서 *Love at Last* 영화를 보면, 로맨스 정도가 0.9라고 되어있다. 따라서 이는 로맨스 영화일 확률이 높고, 액션 영화일 확률은 0이다. 따라서 이 영화에서 액션은 나오지 않는다. *Romance Forever*의 경우에는 로맨스가 1.0, 액션은 0.01 이다. 영화에서 작은 자동차 사고와 같은 액션 장면이 있을지도 모른다. 건너뛰어서 *Swords vs karate* 를 살펴보자. 이

영화는 로맨스는 0이므로 로맨스는 전혀 등장하지 않지만, 액션이 가득하다. Nonstop car chases는 로맨스가 약간 등장하지만, 주로 액션을 다룬다. Cute puppies of love는 액션이 전혀 등장하지 않는 로맨스 영화이다.

01:55

따라서 이와 같은 feature가 주어졌을 때, 각각 영화는 feature 벡터를 표현하고 있는 것이다. 첫 번째 영화를 살펴보자. 이렇게 영화 1, 2, 3, 4, 5라고 하자. 첫 번째 영화, Love at last는 feature 값으로 0.9, 0을 가지고 있다. 이것은 feature x_1, x_2 라고 할 수 있다. 이번에도 다른 feature를 추가하려고 하는데, feature $x_0 = 1$ 이다. 이제 이를 함께 놓고, 여기 feature x_1 이 있다. 이를 가지고 feature x_1 을 생성할 텐데, $x(1)$ 은 첫 번째 영화의 feature 벡터를 의미하고, 이 feature 벡터는 1과 같은데, 이는 x_0 이고, 그리고 0.90, 0 이렇게 포함하게 된다. 따라서 Love at last는 feature 벡터 $x(1)$ 이고, Romance forever는 feature 벡터 $x(2)$ 이고 또한, Swords vs karate는 feature 벡터 $x(5)$ 값을 가지게 된다.

02:55

또한 이전에 우리가 활용하던 표기법과 통일시키기 위해서 n 을 feature 수라고 정하고, 여기서 x_0 은 포함하지 않겠다. 따라서 $n=2$ 인데, 로맨스와 액션의 정도를 측정하는 feature x_1, x_2 가 두 개 있기 때문이다. 이제 예측을 해볼 것인데, 먼저 유저들의 평점을 각각의 선형 회귀 문제로 간주할 것이다. 예를 들어, 유저 j 가 매개 변수 벡터 $\theta(j)$ 를 학습 할 건데, $\theta(j) = \mathbb{R}^3$ 이다. 보통 $\theta(j) = \mathbb{R}^{n+1}$ 의 형태인데, 여기서 n 은 feature의 수를 나타내고, 여기에는 x_0 은 포함하지 않는다. 이제 유저 j 가 영화 i 에 매기는 평점을 예측 하면, $(\theta(j))^T x(i)$ 이와 같다. 이제 예시를 살펴보자. 먼저 유저 1은 Alice다. Alice와 관련해, 매개변수 벡터 $\theta(1)$ 이라고 하자. 유저 2 Bob은 다른 매개변수 벡터 $\theta(2)$ 가 된다. Carol은 다른 매개변수 벡터 $\theta(3)$, Dave는 다른 매개변수 벡터 $\theta(4)$ 가 된다.

04:17

그렇다면, Alice가 Cute puppies of love 영화에 어떤 평점을 매길지 예측해보자. 이 영화는 매개변수 벡터 $x(3)$ 인데, 여기서 $x(3)$ 은 x_0 의 1, 0.99, 0 값을 가진다. 따라서 예를 들어, Alice의 매개변수 벡터 $\theta(1)$ 값이 주어졌다고 할 때, 이 매개 변수 벡터값을 구하는 방법은 이따가 다시 설명했다. 하지만 지금은 일단 불특정한 학습 알고리즘이 매개 변수 벡터 $\theta(1)$ 을 학습했고, 이는 0,5,0 값을 가진다. 따라서 이 공간에 들어갈 값은, 영화 3 Cute puppies of love의 feature 벡터 예측 값은 $\theta(1)^T x(3)$ 이와 같다. 따라서 이 두 벡터 사이의 내적은 $5 \times 0.99 = 4.95$ 가 된다. 따라서 이 값에 대한 예측은 4.95가 된다. 이 값은 매개변수 벡터 $\theta(1)$ 이 맞는다면, 타당한 값으로 보인다.

05:38

이와 같이 다른 유저의 선형 회귀 값을 복사하여 적용한 것인데, Alice는 그녀의 평점을 예측하는데 우리가 활용한 매개변수 벡터 $\theta(1)$ 가 있었고, 이는 그 영화에 얼마나 로맨스 또는 액션이 포함되어있는지를 나타낸다. 그리고 Bob, Carol, Dave의 경우, 모두 로맨스와 액션의 정도

에 따른 각기 다른 선형 함수 값을 가지고 있었고, 이를 통해 우리는 그들의 별점을 예측할 수 있는 것이다.

Q) 다음과 같은 영화 평점 표가 있다. $\theta(3)$ 값으로 가장 적절한 것은? ($x_1 = 0$)

정답 4번

06:15

이제 공식으로 이를 정리해보겠다. 먼저 만약 유저 j 가 영화 i 를 평가했을 때 $r(i,j) = 1$ 가 되고, $y(i,j)$ 는 그 영화의 평점인데, 평점이 먼저 있어야 한다. 유저가 먼저 그 영화를 평가를 해야 얻을 수 있는 값이다. 이전 슬라이드에서 살펴봤듯이, $\theta(j)$ 는 유저의 매개변수 벡터이고, $x(i)$ 는 특정 영화의 feature 벡터 값이다. 그리고 개별 유저마다 각 영화에 대한 평점을 예측하는 것은 다음과 같다. 이번에는 잠시 등장하는 $m(j)$ 값에 대해 설명했다. 여기서 $m(j)$ 는 유저 j 가 평가한 영화의 수를 의미한다. 이 표기법은 이 부분에서만 필요하다. 매개변수 벡터 $\theta(j)$ 값을 구하는 방법은 다음과 같다. 이는 선형 회귀 문제로 볼 수 있다. 따라서 매개변수 벡터 $\theta(j)$ 값을 선택해서, 여기 값을 예측하는 것인데, 주어진 학습 세트, 즉 데이터에서 관찰한 값과 최대한 가까우면 된다.

07:19

이제 공식을 적어보자. 매개변수 벡터 $\theta(j)$ 값을 구하기 위해서, 매개변수 벡터 $\theta(j)$ 값을 최소화 해보자. 그리고 유저 j 가 평가한 모든 영화의 합을 구하려고 한다. 먼저 $\sum_{i:r(i,j)=1}$ 라고 써주면 여기 합계 문법을 읽는 방법은 i 값의 모든 합이다. 여기서 $r(i,j) = 1$ 이다. 따라서 유저 j 가 평가한 모든 영화의 합을 구하는 것이다. 그리고 나서 $(\theta(j))^T x(i)$ 를 연산한다. 이는 유저 j 가 영화 i 에 매긴 평점을 가지고 예측하는 것이다. 여기에 $-y(i,j)$ 를 해주고 이는 실제 관찰된 평점이고, 여기에 제곱을 해준다. 그 다음에 이를 유저 j 가 평점을 매긴 영화의 수인 $1/2m(j)$ 로 나누어준다. 이는 최소자승 회귀분석(least squares regressions)과 같다. 이는 선형 회귀로 매개 변수 벡터 $\theta(j)$ 를 구해서 이와 같은 형태의 제곱 오류형을 최소화 한다.

08:36

또한 여러분이 regularization 항을 추가하기를 원한다면, $\lambda/2m(j)$ 를 더해주면 되는데, 우리에게 $m(j)$ 예시가 있기 때문에, 여기는 $2m(j)$ 가 된다. 유저 j 는 이만큼 영화를 많이 평가했지만, $m(j)$ 매개 변수에 대입할 많은 데이터 점수를 우리가 가지고 있지 않다. 이제 여기에 regularization 항을 추가해주는데, $\sum_{k=1}^n (\theta(j)_k)^2$ 와 같다. 이 합계는 $k=1$ 부터 n 까지고, 여기서 $\theta(j)$ 는 $n+1$ 차원 벡터가 된다. 이전 예시에서 $n = 2$ 였다. 하지만 더 넓은 의미에서, n 은 영화 각각에 포함된 feature의 수를 말한다. 따라서 $\theta(0)$ 값과 같은 bias 항은 보통 regularize하지 않는다. 따라서 합계는 $k=1$ 부터 n 까지다. 따라서 이 공식을 $\theta(j)$ 함수로 최소화하면 이는 좋은 해결책이 될 것이고, 유저 j 의 영화 평점을 예측하기 위한 꽤 좋은 측정 $\theta(j)$ 값을 구할 수 있게 된다.

09:40

추천 시스템에서는 이러한 표기를 조금 바꿔보려고 한다. 이 공식을 단순화하기 위해서, 여기 $m(j)$ 항을 지운다. 그럼 이렇게 항수만 남는다. 따라서 이러한 최적화 공식에서 $\theta(j)$ 값을 바꾸지 않고 이를 삭제할 수 있게 된다. 따라서 이 전체 방정식에 $m(j)$ 값을 곱하고 항수를 삭제해도, 이를 최소화하면 $m(j)$ 값은 아까와 똑같다. 따라서 이전 슬라이드에 내용을 옮겨오면, 여기 최적화 objective가 있다. 유저 j 의 매개변수인 $\theta(j)$ 값을 구하기 위해 여기 최적화 objective에서 $\theta(j)$ 를 최소화 해보겠다. 여기 제곱 오류 항이 있고, 여기 regularization 항이 있다. 추천 시스템을 개발하기 위해서 단순히 한 유저의 매개변수만 구하는 것이 아니라 모든 유저의 매개변수를 구해야 한다. 여기 보면, $\theta(nu)$ 까지 모든 유저의 매개 변수를 구할 것이다.

10:37

따라서, 여기 최적화 objective에다가 혼합된 합계 공식을 추가해주는 것이다. 따라서 여기 이 절반 정도 부분은 여기 위에 있는 공식과 같은 것이다. 다른 점은 특정 유저 $\theta(j)$ 값만 구하는 게 아니라 모든 유저의 objective 합계를 구하고 여기 있는 최적화 objective 총합, 즉 모든 비용값을 최소화할 것이다. 이 곳을 $\theta(1), \dots, \theta(nu)$ 의 함수로 최소화하면, 유저마다 각각의 매개변수 벡터를 얻게 된다. 그렇게 되면, 이를 가지고 모든 유저, n 명의 유저에 대해 예측을 할 수 있게 된다. 이제 모든 공식을 합쳐보면, 이렇게 최적화 objective가 공식이 위에 있다. 여기에 이름을 붙여주자면, $J(\theta(1), \dots, \theta(nu))$ 라고 하겠다. 여기서 J 는 여기서 최소화하려고 하는 최적화 objective를 말한다.

11:41

이제 실제로 최소화를 해보면, 여기서 기울기 하강 업데이트를 살펴보면, 이러한 방정식이 있다. $\theta(j), \theta(k)$ 가 있고, $-\alpha$, 여기서 알파는 학습률이다. 곱하기 여기 오른쪽에 있는 항을 해주면 된다. 여기 보면 $k=0$ 일 때와 $k \neq 0$ 일 때의 두 케이스가 있는데, 여기 있는 regularization 항은 $\theta(j)k$ 값만 regularize하는데 이 때, $k \neq 0$ 이다. 따라서 우리는 $\theta(j)$ 를 regularize 하지 않고, 이를 약간 다르게 업데이트하여 $k=0$ 일 때와 $k \neq 0$ 일 때의 두 케이스로 나누는 것이다. 예를 들어, 여기 있는 이 항은 최적화 objective의 매개 변수에 대한 편도 함수인데, 따라서 이는 기울기 하강이고, 도함수를 먼저 연산해서 여기에 대입해두었다.

12:38

만약 여기 기울기 하강 업데이트가 우리가 선형 회귀에서 봤던 것과 많이 비슷한 이유는 이는 사실 선형 회귀와 같기 때문이다. 주요 차이점은 선형 회귀에서 $1/m$ 항이 있는데, 이는 사실 $1/m(j)$ 이다. 하지만 전에 최적화 objective 공식을 활용할 때, 이를 삭제했으므로 여기에도 $1/m$ 항이 없는 것이다. 하지만 이는 학습 예시 중 하나로, Σ 곱하기 $x(k)$ 더하기 여기 regularization 항 인데 이 regularization 항 도함수에 기여한다. 따라서 기울기 하강을 활용할 때, 비용 함수 J 가 모든 매개 변수를 학습할 수 있도록 최소화하는 방법은 다음과 같다. 또한 도함수에 활용하는 이 공식들은 원활 경우, conjugate gradient 또는 LBFGS와 같은 더 고차원의 최적화 알고리즘에 적용할 수 있다. 이를 또한 비용 함수 J 값을 최소화 하는데 활용할 수 있다.

13:37

이번 강의를 통해 여러분이 각기 다른 유저의 다양한 영화 평점을 예측하기 위한 편차를 선형 회귀에 적용할 수 있기를 바란다. 이러한 알고리즘은 콘텐츠 기반 추천 또는 콘텐츠 기반 접근이라고 하는데, 이는 다른 영화마다 feature를 적용할 수 있는데, 따라서 이러한 feature가 이 영화의 콘텐츠가 로맨스 인지 액션인지를 알아낸다. 그리고 이러한 영화의 feature, 콘텐츠를 활용하여 예측을 하는 것이다. 하지만 많은 영화에 이러한 feature가 사실 없거나 우리가 판매하려고 하는 어떠한 아이템을 위한 모든 영화에서 feature를 얻기는 매우 어렵다. 따라서 다음 시간에는 콘텐츠 기반이 아닌, 데이터 세트에 있는 모든 영화를 위한 feature가 주어지지 않은 추천 시스템에 이야기 해보겠다.

No.	Title
Week 9-11	Collaborative Filtering
개요	
Machine Learning Specialization 9-11 번역본	

00:00

이번 시간에는 추천 시스템 개발 접근법 중 하나인 협업 필터링(collaborative filtering)에 대해 강의하도록 하겠다. 이 알고리즘은 매우 흥미로운 특징을 가지고 있는데, 이는 특징 학습(feature learning)이라고 한다. 이는 어떠한 feature를 사용해야 할지 스스로 학습하기 시작하는 알고리즘을 뜻한다. 여기에 전에 사용한 데이터 세트가 있는데, 누군가가 우리에게 영화가 얼마나 로맨틱했는지, 액션이 얼마나 들어있는지에 대해 이야기해준 내용이다. 하지만 보다시피, 이와 같이 영화를 보고 영화가 얼마나 로맨틱했는지, 얼마나 많은 액션이 들어있는지, 거기에 이 둘 이외에 feature에 대한 정보를 누군가에게 얻기란 매우 어렵고 시간이 필요하며 오래 걸리는 작업이다. 그렇다면 이러한 feature는 어디서 얻게 되는 건가? 먼저 문제를 조금 바꿔서 이러한 feature의 값을 모르는 데이터 세트가 있다고 하자. 여기 영화 데이터 세트가 있고, 유저들이 매긴 평점이 있는데, 영화가 얼마나 로맨틱한지, 얼마나 많은 액션이 담겨있는지는 알 수 없다. 따라서 모두 물음표로 표시해놨다.

01:10

이번에는 조금 다른 가정을 해보자. 우리가 유저를 각각 찾아가서 그 유저들은 우리에게 로맨스 영화가 얼마나 좋았는지, 액션으로 가득한 영화가 얼마나 재미있었는지를 이야기해준다. 따라서 앤리스는 $\theta(1)$, 밥은 $\theta(2)$, 캐롤은 $\theta(3)$, 데이브는 $\theta(4)$ 이다. 이것도 같이 활용해보면, 앤리스는

로맨스 영화가 정말 좋았다고 말하며 여기 5점이 있는데, 이는 $x(1)$ 이다. 또한 앤리스는 액션 영화가 별로 좋지 않았다고 하며 0점을 줬다. 밥도 이와 비슷한 리뷰를 말해줬고, 여기 $\theta(2)$ 가 있다. 반면에 캐롤은 액션 영화가 좋았다고 말했고, 여기 5점이 있다. 이는 $x(2)$ 에 해당하고, 여기서 $x_0=1$ 이라는 것을 기억하자. 캐롤이 로맨스 영화를 좋아하지 않았다고 했고, 데이브도 이와 비슷하다. 이와 같이 우리가 각 j 유저들에게 $\theta(j)$ 값을 이야기해주었다고 하자. 따라서 이는 그들이 다른 장르의 영화를 얼마나 좋아했는지에 대해 명시하고 있다.

02:23

만약 유저들에게 여기 매개변수 θ 값을 얻게 되면, 각 영화의 x_1, x_2 값을 추론할 수 있게 된다. 예시를 살펴보자. 영화 1을 보면, 영화 1은 feature 벡터 x_1 과 연관되어 있는데, 이 영화 제목은 Love at last 이지만 이를 무시해보자. 우리가 영화에 대해 전혀 모른다고 가정하고, 영화 제목도 모른다고 하자. 우리가 아는 것이라고는 앤리스가 이 영화를 좋아했다는 것이다. 밥도 이 영화를 좋아했다. 캐롤과 데이브는 이 영화를 싫어했다. 그렇다면 이를 통해 무엇을 추론할 수 있는가? 우리는 feature 벡터를 통해 앤리스와 밥이 로맨스 영화를 좋아한다는 것을 알 수 있는데 여기 5점을 줬기 때문이다. 반면에 캐롤과 데이브는 로맨스 영화는 싫어하고 대신 액션 영화를 좋아하는 것을 알 수 있다. 따라서 이러한 매개 변수 벡터는 유저 3, 4 인 캐롤과 데이브가 우리에게 알려준 정보다.

03:20

따라서 앤리스와 밥이 영화 1을 좋아했고, 캐롤과 데이브는 영화 1을 싫어했다는 데이터에 따르면, 우리는 이 영화가 로맨스 영화라는 사실을 유추할 수 있을 것이다. 아마도 액션 영화 쪽은 아닐 것이다. 이 예시는 약간 수학적으로 간소화되어 있지만 우리가 필요한 것은 어떠한 feature 벡터가 x_1 인지, 그래서 $\theta(1)^T x_1 \sim 5$ 이건 앤리스의 평점이고, $\theta(2)^T x_1 \sim 5$, $\theta(3)^T x_1 \sim 0$ 이건 캐롤의 평점이고, $\theta(4)^T x_1 \sim 0$ 라는 것을 알 수 있다. 그리고 보다시피 x_1 은 여기 x_0 값이 들어가서, [1, 1.0, 0.0] 이기 때문에, 앤리스, 밥, 캐롤, 데이브의 영화 선호도에 따라서 그들이 어떻게 영화에 평점을 매겼는지를 알 수 있게 해준다. 이와 같은 방법으로 여기 다른 물음표에도 어떠한 수치가 들어갈지 예측할 수 있다.

Q) 다음과 같이 영화 평점이 있다.

여기 x_1 feature 하나만 주어졌을 때,

$x(1)$ 값(상단 표에 물음표 표시)으로 적절한 것은?

4번 보기 모두 가능

(정답 1번)

04:38

이제 feature $x(i)$ 를 학습하여 이 문제를 공식으로 써보자. 유저들에게 영화 선호도 정보를 얻었다고 하자. 유저들이 찾아와서 $\theta(1), \dots, \theta(n)$ 까지 값을 알려줬는데, 우리는 영화 1의 feature 벡터 $x(i)$ 값을 얻고자 한다. 이제 여기서 다음과 같이 최적화 문제를 제기할 수 있다. 먼저 영화 i 의 평점인 모든 인덱스 j 의 합계를 구하고자 하는데, 왜냐하면 영화에 대한 feature를 학습하려고 하는데, 그게 feature 벡터 $x(i)$ 다. 다음으로는 여기 오류 제곱을 최소화하고자 하는데, feature $x(i)$ 를 선택해서, 유저 j 가 영화에 내린 평점을 예측한 값은 제곱 오류 안에서 유저 j 의 영화 i 에 대한 평가인 실제 $y(i, j)$ 값과 비슷할 것이다. 따라서 다시 정리해보면, 이 항은 feature $x(i)$ 를 선택해서, 이 영화에 평점을 매긴 모든 유저 j 에게 이 알고리즘은 해당 유저가 이 영화를 어떻게 평가할지 그 값을 예측하는데, 이는 오류 제곱 안에서 유저가 해당 영화를 실제로 평가한 값과 크게 다르지 않다는 것이다. 여기까지 제곱 오류 항을 살펴봤다. 또한 이전처럼 여기 regularization 항을 추가하여 feature값이 너무 커지지 않도록 할 수 있다.

06:13

이처럼 한 특정 영화에 대한 feature들을 학습하는 방법을 알아봤는데, 이번에는 모든 영화의 모든 feature를 학습해보기로 하자. 먼저 여기 추가로 합계항을 추가하고, 여기서는 $N(m)$ 까지 영화의 합을 구하고, 여기 위에 있는 objective를 최소화하여 모든 영화의 합을 구한다. 그러면 최적화 문제를 활용하게 되는 것이다. 이를 최소화하면, 모든 영화의 feature를 얻게 될 것이다. 이번에는 이전 시간에 살펴본 알고리즘과 방금 살펴본 알고리즘을 살펴보자. 이전 시간에는 영화 평점이 있을 때, $r(i, j)$ 데이터가 있고, 평점 $y(i, j)$ 값이 있다.

Q) 다음을 최소화하기 위해 기울기 하강을 활용한다고 하자.

$i \neq 0$ 일 때, 가장 적절한 기울기 하강 업데이트 규칙은?

(정답 4번)

이와 같은 여러 영화의 feature를 가지고, 다른 유저들의 매개변수 θ 값을 학습할 수 있다. 지난 시간에는 유저가 매개 변수 값을 제공하면, 다른 영화의 feature값을 측정할 수 있었다. 이는 닮이 먼저냐 계란이 먼저냐 문제와 같다. x 값들을 구할 수 있는 θ 값을 먼저 구할지, x 값들을 구해서 θ 값들을 구할지 뭐가 먼저냐는 말이다. 이와 같은 경우, 이는 실제로 잘 구현되는데, 임의의 θ 값을 정하는 것이다. 그리고 이렇게 정한 임의의 θ 값을 가지고 다른 영화에서 feature를 학습하기 위해 이야기했던 과정을 거치면 되는 것이다. 이제 영화의 임의의 feature를 가지고, 이전 강의에서 했던 것처럼 여기 첫 번째 공식에 대입하여 더 나은 매개 변수 θ 값을 얻을 수 있다. 이제 그렇게 얻은 θ 값을 가지고, 더 나은 feature를 구할 수 있다. 이러한 과정을 반복하면, θ, x, θ, x , 이런 식으로 반복해보면 잘 구현될 뿐만 아니라, 여러분의 앨범에 합리적인 영화의 feature 세트를, 다른 유저의 매개 변수 세트를 모으게 될 것이다.

08:35

지금까지 기본적인 협업 필터링 알고리즘에 대해 알아봤다. 하지만 이는 우리가 활용할 마지막 알고리즘은 아니다. 다음 시간에는 이 알고리즘을 더 향상시켜서 더 효율적인 연산이 가능하게 만들어볼 것이다. 이번 강의를 통해 문제를 만들어내서 동시에 매개 변수와 다른 영화의 feature를 학습하는 것을 이해할 수 있는 계기가 되었으면 한다. 이러한 추천 시스템 문제는, 유저 각각이 다양한 영화를 평가할 때에만 가능한데, 영화마다 여러 유저가 평가를 해야 좋다. 이러한 과정을 반복하면서 θ 와 x 값을 예측할 수도 있다.

09:13

정리하자면, 이번 시간에는 기본적인 협업 필터링에 대해서 알아봤다. 여기서 협업 필터링이라는 용어는 이러한 큰 세트의 알고리즘을 구현할 때, 유저들이 협업처럼 효율적으로 모든 이에게 더 나은 영화 평점을 얻기 위해 무언가를 하는 것의 관찰을 의미한다. 왜냐하면 모든 유저는 영화를 평가하는데, 이러한 평점을 통해 알고리즘이 더 나은 feature를 얻을 수 있게 돋는다. 영화 평점을 스스로 매기면, 시스템이 더 좋은 feature를 학습할 수 있도록 돋는 것이고, 이러한 feature는 모든 유저들에게 더 나은 영화 추천 서비스를 제공할 수 있는 시스템을 만드는데 활용된다. 따라서 모든 유저들이 모두를 위해 더 나은 feature를 얻기 위해 서로 협업한다는 개념이다. 이것이 바로 협업 필터링이다. 다음 시간에는 이러한 아이디어를 통해 실제 구현해보면서, 더 나은 알고리즘을, 더 나은 협업 필터링 기술을 개발해보도록 하겠다.

No.	Title
Week 9-12	Collaborative Filtering Algorithm

00:00

지난 두 강의 시간에 영화에 관한 feature가 주어졌을 때, 이를 유저의 매개 변수 데이터를 학습하는 데 활용하는 방법에 대해서 알아봤다. 두 번째로, 유저의 매개 변수가 주어졌을 때, 이를 영화의 feature를 학습하는데 활용할 수 있었다. 이번 시간에는 그러한 아이디어를 바탕으로 협업 필터링 알고리즘을 개발해보도록 하겠다. 우리가 이전 시간에 다뤄본 내용 중에 영화의 feature가 있을 때, 유저의 매개 변수 θ 를 구해 최소화 문제를 해결 할 수 있었다. 또한 매개 변수 θ 가 주어졌을 때, 이를 feature x 를 측정하는데 활용할 수 있었고, 이는 또한 최소화 문제를 해결을 통해 가능하다. 따라서 한가지 방법은 왔다갔다하며 연산하는 것이다. 먼저 임의의 매개 변수를 정하고 이를 가지고 θ 값을 구한 다음, 또 x 값을 구하고, 또 θ 값을 구하고, 또 x 값을

구하는 것이다. 하지만 더 효율적인 알고리즘이 있는데, 이는 θ 와 x 를 왔다 갔다 하지 않아도 되고, θ 와 x 를 동시에 구할 수 있다.

01:04

방법은 다음과 같다. 먼저 여기 최적화 objectives 둘을 가지고 같은 objective에 넣는 것이다. 이제 새로운 최적화 objective j 를 정의해보겠는데, 이는 비용 함수로, 이 것은 feature x 의 함수 그리고 이것은 매개 변수 θ 의 함수다. 이는 위에 있던 두 최적화 objective인데, 이렇게 두 개로 합쳤다. 이를 설명하기 위해서는, 먼저 여기 제곱 오류 항은 위에 있는 항과 같은 것이고, 여기서 합계는 조금 달라 보이는데, 이 역할에 대해서 먼저 알아보자. 첫 번째 합계는 모든 유저 j 를 더하고, 그 다음에 그 유저가 평점을 매긴 모든 영화의 합니다. 따라서 이는 (i, j) 모두의 합을 구하는 것이고, 이는 유저가 평점을 매긴 영화와 대응한다. 여기 j 의 합계는 모든 유저를 말하고, 해당 유저에 의해 평가가 된 모든 영화의 합을 말한다.

02:03

여기 있는 합계는 반대의 순서로 연산한다. 모든 영화 i 의 합계를 구하고, 해당 영화에 평가를 매긴 모든 유저 j 의 합을 구한다. 따라서 이 두 합계는 (i, j) 의 합을 구하는 것이고, $(i, j) : r(i, j) = 1$ 이다. 이는 유저와 평점이 있는 영화를 짹지은 것이다. 따라서 여기 두 항은 여기 이 첫 번째 항과 같은 것이고, 여기에 합계를 $(i, j) : r(i, j) = 1$ 의 합계라고 명시했다. 다음으로는 결합된 최적화 objective를 정의하는 것인데, 이를 최소화하여 x 와 θ 를 동시에 구하는 것이다. 그리고 최적화 objective에서 이 다른 항은 regularization θ 항인데, 이것이 여기로 내려온다. 그리고 마지막 항 x 의 최적화 objective 항이고, 이는 여기로 옮겨왔다.

03:15

그리고 여기 최적화 objective j 는 흥미로운 특성을 지니고 있는데, x 의 항수값을 가지고 있을 때, θ 에 대해 최소화하면, 이 문제의 답을 구할 수 있다. 반면에 이를 반대로 하면, θ 의 항수를 가지고 x 에 대하여 j 를 최소화하면, 이 것과 같아진다. 왜냐하면 각각 x 또는 각각 θ 를 최소화하면 이 두 항 모두 항수이기 때문이다. 따라서 여기 최적화 objective는 여기 x 와 θ 의 비용 함수를 합친 것이다. 단 하나의 최적화 문제를 생성하기 위해서는, 여기 비용 함수를, feature x 와 유저 매개변수 데이터의 함수로 간주하고, 이 전부를 x 와 θ 의 함수로 최소화하는 것이다. 그러면 이것과 이전 알고리즘의 차이점은 이전처럼 $\theta \rightarrow x \rightarrow \theta \rightarrow x$ 로 최소화 하는 연산을 하지 않아도 된다는 것이다. 이 새로운 버전에는 두 매개 변수 x 와 θ 의 세트를 계속해서 왔다갔다할 필요 없이, 두 매개 변수 세트를 동시에 최소화하는 것이다.

04:40

다음으로 우리가 feature를 이와 같이 학습 할 때, 이전에는 $x_0 = 1$ 이라는 것을 활용했었다. 이와 같이 feature를 학습할 시 정형화를 활용할 때, 실제로는 이와 같은 관습을 따르지 않는다. 따라서 우리가 학습할 feature x 는 $x \in R^n$ 이다. 반면에 이전 feature x 는 $x_0=1$ 을 포함하여 $x \in R^{n+1}$ 이다. 태을 삭제하고, 이제 $x \in R^n$ 이다. 이와 비슷하게, 매개 변수 θ 는 같은 차원에 있기 때문에, $\theta \in$

R^n 이 되고, 이는 x_0 가 없을 경우, 매개변수 θ_0 도 필요가 없기 때문이다. 따라서 이전 방법을 쓰지 않는 이유는 이제 모든 feature를 학습하는데, feature가 항상 1인 변경이 불가한 코드는 필요하지 않기 때문이다. 만약 알고리즘에서 항상 1인 feature를 필요로 한다면, 스스로 1을 학습하도록 선택할 수 있다. 따라서 알고리즘이 선택한다면, $x_1 = 1$ 이라고 설정할 수 있다. 따라서 $x_0 = 1$ 이라는 feature가 필요하지 않는 것이고, 알고리즘은 스스로 이를 학습할 수 있는 유연성을 지니고 있기 때문이다.

05:55

이렇게 여러 공식을 하나로 합쳐서 협업 필터링 알고리즘을 생성해보자. 먼저, x 와 θ 를 작은 임의의 값으로 초기화한다. 이는 신경망 학습과 조금 비슷한데, 신경망에서도 모든 매개 변수를 작은 임의의 값으로 초기화한다. 다음으로, 기울기 하강 또는 고도의 최적화 알고리즘을 활용하여 비용 함수를 최소화한다. 따라서 도함수를 적용하면, 이와 같은 기울기 하강을 얻을 수 있는데, 여기 이 항은 비용 함수의 부분 도함수이다. 이를 다 적지는 않겠지만, feature 값 $x(i)k$ 에 대한 값이 있고, 여기 이 항은 우리가 최소화하는 매개 변수 θ 에 대한 비용 함수의 부분 도함수 값이다.

06:49

다시 하면 언급하자면, 이 공식에는 더 이상 $x_0 = 1$ 는 포함하지 않는다. 따라서, $x \in R^n$ 이고, $\theta \in R^n$ 이다. 이와 같은 새로운 정형화에서, 모든 매개 변수 θ 하나하나를, 모든 매개 변수 θ 를 규칙화하고 있다. 이제 더 이상 다르게 규칙화된 θ_0 는 없고, 이는 매개변수 θ_1 에서 θ_n 까지 항에 비교해서는 규칙화되지 않았다. 따라서 더 이상 θ_0 는 없고, 이와 같은 업데이트에서 $k=0$ 라는 특수한 경우를 가져오지 않은 것이다. 따라서 우리는 기울기 하강을 활용해 feature 와 매개 변수 θ 에 대한 비용 함수 j 를 최소화하는 것이다.

07:33

마지막으로, 유저가 매개변수 θ 를 가질 때, 학습된 feature x 를 가진 영화가 있을 때, 해당 유저가 그 영화에 매겨진 평점을 $\theta^T x$ 로 예측할 수 있다. 이를 써보면, 만약 유저 j 가 영화 i 에 아직 평점을 매기지 않았다면, 해당 유저 j 가 영화 i 에 매길 평점을 $(\theta(j))^T(x(i))$ 로 예측하는 것이다.

Q) 우리가 설명한 알고리즘에서, 다음과 같이 $x(1), \dots, x(nm)$ 과 $\theta(1), \dots, \theta(Nu)$ 를 작은 임의의 값으로 초기화했다. 그 이유는?

1. 이 단계는 선택사항이므로, 모두 0으로 초기화해도 작동될 것이다.
2. 임의의 초기화는 어떤 문제이든 기울기 하강을 활용할 때, 항상 필요한 작업이다.
3. 어떠한 i, j 에서도 $x(i) \neq \theta(j)$ 가 성립하게 해준다.
4. 이는 대칭성 봉괴(신경망의 매개 변수를 임의로 초기화 하는 것과 비슷)를 일으켜,

알고리즘이 이와 같이 서로 다른 feature $x(1), \dots, x(Nm)$ 를 학습할 수 있게 해준다.

이것이 바로 협업 필터링 알고리즘이고 이러한 알고리즘을 구현하면, 매우 훌륭한 알고리즘을 얻게 되는데, 이는 동시에 모든 영화의 좋은 feature를 학습하고, 모든 유저의 매개 변수를 학습 할 것이며, 다른 유저들이 다른 영화에 어떤 평점을 매길지를 잘 예측할 수 있게 된다.

No.	Title
Week 9-13	Vectorization: Low Rank Matrix Factorization

00:00

지난 시간에는 협업 필터링 알고리즘에 대해 이야기해봤다. 이번 시간에는 이러한 알고리즘에 벡터화 구현하는 것에 대해 이야기해보겠다. 또한 이 알고리즘의 활용 방법에 대해서 이야기하겠다. 예를 들어, 먼저 하나의 물품이 주어졌을 때, 이와 관련된 다른 물품을 찾을 수 있을까? 예를 들어, 한 유저가 하나의 물품을 최근에 보고 있다. 이와 관련된 다른 물품을 해당 유저에게 추천할 수 있을까? 이를 어떻게 해결 할 수 있을지 살펴보자. 먼저, 협업 필터링 문제를 예측하는 것 대신에 대안을 찾아볼 것이다. 여기 영화 다섯 편이 포함된 데이터 세트가 있다. 여기 모든 유저들이 매긴 모든 평점을 뒤어서 행렬로 만들 것이다. 여기 보면 영화가 다섯 편, 유저가 네 명 있다. 따라서 이 행렬 y 는 5×4 행렬이 된다. 여기 있는 모든 원소 데이터를 가지고 물음표를 포함해서 여기 행렬에 그대로 가져다 넣은 것이다. 따라서 이 행렬의 원소는, 행렬의 (i, j) 원소는 우리가 이전에 $y(i, j)$ 라고 표기했던 것으로 유저 j 가 영화 i 에 준 평점을 의미한다.

01:15

이와 같이 모든 평점을 포함한 행렬 y 가 있을 때, 알고리즘의 모든 예측 평점을 쓰는 대안이 있다. 특히, 어떤 특정한 유저가 특정 영화에 하는 예측이 뭔지, 유저 j 가 영화 i 의 준 평점을 예측하는 것은 $(\theta(j))^T(x(i))$ 이 공식으로 가능하다. 평점을 예측하는 행렬은 (i, j) 로 입력되어 다음과 같다. 이는 유저 j 가 영화 i 에 주는 평점에 대응하는데, 이는 $(\theta(j))^T(x(i))$ 와 같다. 따라서 이 행렬에서 첫 번째 원소 $(1, 1)$ 는 유저 1이 영화 1에 매긴 평점이고, $(1, 2)$ 원소는 유저 2가 영화 1에 매긴 평점이다. 이것은 유저 1이 마지막 영화에 대해 매긴 평점을 예측한 것이다. 이러한 평점은 이러한 값을 예측했고, 이 평점은 이 값을 예측했다.

02:35

이제 예측 평점 행렬을 가지고, 이를 더 단순하게 벡터화해서 쓰는 방법을 알아보자. 먼저 행렬 x 를 정의하면, 선형 회귀에서 다뤘던 행렬과 같은데, $(x(1))^T, (x(2))^T, \dots, (x(Nm))^T$ 이렇게 써줄 수 있다. 따라서 모든 영화에 대한 feature를 이렇게 줄 세웠다. 각각 영화를 하나의 예시로 간주하고, 여러 영화의 모든 feature를 이렇게 행으로 세운 것이다. 이번에는 행렬 θ 를 구해 보면,

유저의 매개 변수 벡터 값을 다음과 같이 행으로 쭉 쌓는다. 따라서 여기 $(\theta(1))^T$ 는 첫 번째 유저의 매개 변수 벡터이고, $(\theta(2))^T$ 가 있고, 이렇게 행으로 $(\theta(N_u))^T$ 항까지 쌓아서, 행렬 θ 를 정의할 수 있다.

03:50

이제 주어진 행렬 x 와 행렬 θ 의 정의를 가지고, 모든 예측의 행렬을 벡터화 연산을 하기 위해서 $x\theta^T$ 를 활용하여 이 행렬의 벡터로 연산한다. 우리가 지금까지 활용한 협업 필터링 알고리즘에 다른 이름을 붙여주자면 지금 활용한 이 알고리즘은 low rank matrix factorization 이라고 한다. 따라서 사람들이 low rank matrix factorization에 대해 이야기 하는 것은 바로 이 알고리즘에 대해 이야기하는 것이다. 그리고 이러한 용어는 $x\theta^T$ 행렬이 지닌 선형 대수학에서 수학적 특성 low rank matrix 에서 온 것인데, 이를 통해 이 알고리즘이 low rank matrix factorization라는 이름을 얻게 되었는데, 이는 $x\theta^T$ 행렬의 low rank 특성 때문이다.

04:54

Low rank의 뜻을 모르거나 low rank 행렬에 대해 모른다고 해도 걱정하지 않아도 된다. 이 알고리즘을 활용하기 위해서 이에 대해 알 필요는 없기 때문이다. 하지만 선형 대수학을 잘 알고 있다면, 거기서 이 알고리즘에 low rank matrix factorization이라는 이름을 붙여준 것이다.

Q) 다음과 같이 행렬 x 와 행렬 θ 가 있다. 이를 다음과 같은 행렬로 표현하려면 어떤 공식을 활용해야 하는가?

정답 3번

다음으로, 협업 필터링 알고리즘을 구현할 때, 또 다른 방법이 있는데, 이는 연관된 영화를 찾기 위해 학습된 feature를 활용하는 것이다. 각각 물품 i 에서(영화 i) feature 벡터 $x(i)$ 값을 학습했다. 특정한 feature가 다른 feature가 뭔지 미리 알지 못한 채로 학습하게 되는데, 하지만 이 알고리즘을 구현하면, 해당 feature가 여러 영화 또는 물품 등이 가진 중요한 요소를 완벽하게 캡쳐 할 것이다. 여기서 중요한 요소는 유저가 어떤 영화를 좋아하는지, 다른 유저는 어떤 다른 영화를 좋아하는지에 대한 것이다.

05:52

따라서, 다음과 같은 feature를 얻게 되는데, 예전에 배운 것처럼 $x_1 = \text{romance}$, $x_2 = \text{action}$, $x_3 = \text{comedy}$, $x_4 = \dots$ 등이 될 것이다. 총 N 개의 feature를 가지고 있고, 이러한 feature를 학습하게 된 후, 학습된 feature를 가지고 이러한 feature가 무엇인지를 사람이 이해할 수 있게 해석해 내는 것은 꽤 어려워진다. 실제로 feature를 시각화하기 어렵고, 이러한 feature가 무엇인지 알아내기 어렵다. 보통 캡쳐할 만한 의미가 있는 유저들이 좋아하거나 좋아하지 않은 것을 결정할 가장 중요한, 가장 핵심적인 영화의 feature를 학습할 것이다.

06:41

이번에는 다음과 같은 문제를 다뤄보도록 하자. 여기에 어떤 영화 i 가 있고, 해당 영화와 관련 있는 다른 영화 j 를 찾으려고 한다. 왜 이렇게 하려고 하는가? 영화를 유저가 있고, 그들은 영화를 둘러보다가 현재 영화 j 를 보고 있다. 그렇다면 그들이 영화 j 를 보고 난 다음에 추천해줄 만한 적당한 영화는 무엇일까? 또는 누군가가 영화 j 를 최근에 구매했다면 그들에게 구매해보라고 추천할 수 있는 적당한 다른 영화는 뭐가 있을까? 여기 feature 벡터를 학습했으므로, 이는 매우 편리한 방법으로 두 비슷한 영화를 측정할 수 있게 된다. 예를 들어, 영화 i 는 feature 벡터 $x(i)$ 를 가지고 있는데, 다른 영화 $x(j)$ 가 있을 때, $\|x(i) - x(j)\|$, 둘 사이의 거리가 작으면, 영화 i 와 영화 j 가 비슷하다는 것을 나타내는 매우 강한 표시다. 최소한 영화 i 를 좋아한 사람들은 영화 j 를 좋아할 확률이 높다는 것을 의미한다.

07:47

다시 말해, 유저가 어떤 영화 i 를 보고 있는데, 이와 가장 비슷한 영화 5편을 찾아서 유저에게 추천을 해주려고 할 때, 여러 영화의 feature 사이의 거리가 가장 작은 다섯 영화 j 를 찾는 것이다. 그러면 유저에게 여러 영화를 추천해 줄 수 있게 된다. 이번 강의를 통해, 모든 유저의 모든 영화에 대한 예측 평점을 벡터화 연산하는 방법을 배웠고, 학습 feature를 활용하여 서로 관련이 있는 영화나 물품을 찾는 방법을 이해할 수 있을 것이다.

No.	Title
Week 9-14	Implementational Detail: Mean Normalization

00:00

지금까지 추천 시스템 알고리즘 또는 협업 필터링 알고리즘에 대한 주요 내용을 모두 살펴봤다. 이번 시간에는 마지막 구현 디테일인 평균 정규화(mean normalization)에 대해 소개하도록 하겠다. 이는 때로 알고리즘이 더 잘 작동할 수 있게 도와준다. 평균 정규화의 개념에 대해 설명하기 위해 예시를 살펴보자. 이 예시에서 한 유저는 어떠한 영화에도 평점을 매기지 않았다. 따라서 여기 Alice, Bob, Carol, Dave 네 명의 유저에 다섯 번째 유저 Eve를 추가하겠다. 그녀는 어떠한 영화에도 평점을 매기지 않았다. 해당 유저에게 협업 필터링 알고리즘이 어떻게 작용하는지 살펴보자. $N=2$ 라고 하면, 두 feature를 학습할 것이고, 매개 변수 벡터 $\theta(5)$ 를 학습할 것인데, $\theta(5) \in \mathbb{R}^2$ 이다. 여기서 벡터는 이제 $Rn+1$ 이 아니고, Rn 이라는 것을 기억하자. 이제 유저 5 Eve의 매개 변수 벡터 $\theta(5)$ 를 학습해보자.

00:59

여기 첫 번째 항을 보면 이는 최적화 objective인데, 유저 Eve는 어떠한 영화도 평가를 하지 않아서, 유저 Eve에게는 $r(ij) = 1$ 을 만족하는 영화가 없다. 따라서 여기 첫 번째 항은 $\theta(5)$ 값을 결정하는데 필요하지 않다. 이는 Eve가 평점을 한 영화가 없기 때문이다. 따라서 $\theta(5)$ 값에 영향을 주는 항은 이 항뿐이다. 따라서 $\theta(5)$ 를 구해서 마지막 규칙화 항을 최대한 작게 하려고 한다. 즉, $\lambda/2[(\theta(5)1)^2 + (\theta(5)2)^2]$ 를 최소하려고 한다. 따라서 규칙화 항에서 유저 5와 대응하는 부분이고, 여러분의 목표가 이 항을 최소화하는 것이라면, $\theta(5) = [0 ; 0]$ 이 되는 것이다. 왜냐하면 이 규칙화 항은 매개 변수가 0과 가까워지게 하는데, 이렇게 매개변수를 0에서 멀어지게 할 데이터가 없으면, 이 첫 번째 항은 $\theta(5)$ 에 영향을 주지 않으므로 모든 원소가 0인 $\theta(5) = [0 ; 0]$ 가 되는 것이다.

02:17

또한, 유저 5가 어떠한 영화에 평점을 어떻게 매길지를 예측할 때는, $\theta(5)^T x(i) = 0$ 이다. 왜냐하면 $\theta(5)$ 어떠한 x 값을 가져도 내적은 0이 될 것이다. 따라서 Eve가 모든 영화를 0점으로 평가했다고 예측해보자. 하지만 이는 별로 도움이 되지 않을 것 같아 보인다. 여기 영화를 살펴보면, 첫 번째 영화 Love at Last에 두 명의 유저가 5점으로 평가했다. Swords vs. Karate 영화의 경우, 어떤 유저는 5점으로 평가했다. 따라서 어떤 이는 특정 영화를 좋아하는 것이다. Eve가 모든 영화를 0점으로 평가한다고 예측하는 것이 별로 도움이 되지 않을 것처럼 보인다. 사실 Eve가 모든 영화를 0 점으로 평가한다고 예측하면, 그녀에게 어떠한 영화를 추천할 좋은 방법이 없게 되는 것이다. 따라서 여러분은 모든 영화가 Eve에게는 똑같은 평점을 받을 것이라고 예측하고 있기 때문에, 그녀에게 추천할 만한 예측된 평점이 조금이라도 높은 영화가 하나도 없다. 이는 별로 좋지 않다.

03:24

평균 정규화가 여기서 이러한 문제를 해결해준다. 이제 그 방법을 살펴보자. 먼저 모든 영화 평점을 행렬 Y 에 집어넣도록 하자. 따라서 여기 모든 평점을 가지고 행렬 Y 로 가지고 간다. 여기 모두 물음표로만 되어있는 열은 Eve가 어떠한 영화에도 평점을 매기지 않았다는 것을 의미한다. 이제 평균 정규화를 적용해보면, 모든 영화가 얻은 평점의 평균을 연산하는 것이다. 그리고 이를 μ 벡터에 넣도록 하겠다. 따라서 첫 번째 영화는 5점 두 개, 0점 두 개이므로 평균은 2.5점이다. 두 번째 영화는 2.5 별점을 받았다. 그리고 마지막 영화는 0,0,5,0 점을 받았으므로, 이의 평균값은 1.25점이다. 이제 모든 영화의 평점을 보고, 여기서 평균 점수를 빼도록 하겠다. 따라서, 첫 번째 원소 5에서 여기 2.5를 빼면 2.5가 나온다. 두 번째 항은 $5 - 2.5 = 2.5$ 가 나온다. 그리고 0,0, -2.5를 하면, -2.5가 나온다. 즉, 영화 평점 행렬을 가지고 여기 y 행렬에서 영화의 평균 평점을 행에 맞춰서 빼주는 것이다. 이는 모든 영화를 정규화하여 평균 평점으로 0을 가지게 하는 것이다.

04:52

마지막 예시를 들어보겠다. 여기 마지막 행은 0, 0, 5, 0 이다. 여기서 -1.25를 빼주면 이런 값이 나온다. 이제 물론 물음표는 그대로 물음표로 남아있다. 따라서 새로운 행렬 y 에 있는 모든 영화의 평균 평점은 0이다. 이제 이 평점 세트를 가지고 협업 필터링 알고리즘에 적용해보겠다. 이것이 내가 유저에게 얻은 실제 데이터라고 하겠다. 이제 이 데이터 세트를 가지고, 여기 평균 정규화 영화 평점을 가지고, 매개 변수 $\theta(j)$ 와 feature $x(i)$ 를 학습하도록 하겠다.

05:40

이제 다음과 같이 영화 평점에 대한 예측을 해보겠다. 유저 j 가 영화 i 가 있을 때, $(\theta(j))^T(x(i))$ 를 예측하려고 하는데, 여기서 x 와 θ 는 매개 변수로 여기 평균 정규화 데이터 세트에서 학습한 것이다. 하지만 영화 i 에 대한 예측을 하기 위해 데이터 세트에서 평균을 빼버렸기 때문에, 다시 평균을 더해주어야 한다. 따라서 μ_i 값을 다시 더해준다. 이것이 바로 학습 데이터에서 모든 평균을 빼버렸을 때 얻게 되는 예측이다. 따라서, 예측을 할 때, 이와 같이 영화 i 의 평균 μ_i 를 다시 더해줘야 한다. 예를 들어, 유저 5 Eve가, 아까 슬라이드에서 봤던 argument가 이번에도 똑같이 적용되는데, Eve는 어떠한 영화도 평점을 주지 않았고, 따라서 유저 5의 학습된 매개변수 $\theta(5)$ 는 여전히 $\theta(5) = [0;0]$ 이다.

06:38

따라서, 우리는 이제 특정 영화 i 에 대하여 eve의 평점을 예측하기 위해, $(\theta(j))^T(x(i)) + \mu_i$ 를 활용한다. $\theta(5)$ 가 0일 때, 여기 이 부분은 0이 된다. 또한 영화 i 는 μ_i 로 예측할 건데, 이는 이해하기 쉽다. 이는 영화 1을 이브가 2.5로 평가하고, 영화 2를 2, 5로 평가하고, 영화 3을 2라고 평가한다는 것과 같다. 이는 말이 되는데, 만약 Eve가 어떠한 영화도 평가하지 않았다면, 이 새로운 유저 Eve에 대해 아는 것이 하나도 없다. 그래서 모든 영화의 평균 평점을 예측하는 것이다.

07:29

마지막으로, 이번 시간에는 평균 정규화에 대해 이야기해봤는데, 행렬 y 의 각 열을 규정화하여 평균값이 0이 되도록 했다. 평점이 없는 영화가 있을 경우, 이는 아무 영화도 평가하지 않은 유저와 비슷한 경우인데, 아무런 평점이 없는 영화가 있다면, 알고리즘을 통해 아까처럼 다른 행마다 규정화하여 평균이 0이 되게 한 것 대신에, 이번에는 다른 열마다 규정화하여 평균이 0이 되도록 하면 된다. 이는 좀 덜 중요할 수도 있는데, 영화에 평점이 전혀 없는 경우, 그 영화를 누구한테도 추천하는 것은 바람직하지 않다. 따라서, 어느 영화에도 평점을 하지 않은 유저를 다루는 케이스가 평점이 하나도 없는 영화의 케이스보다 더 중요할 것이다.

Q) 지금까지 평균 정규화에 대해 알아봤다. 하지만 feature scaling 과 같은 다른 어플리케이션과 달리 영화 평점은 최대-최소 범위로 나누지 않았다. 그 이유는?

1. 실제 값을 측정하는 데 이러한 scaling은 유용하지 않다.
2. 모든 영화 평점은 이미 비교할 수 있게 되어있다(0점부터 5점까지). 따라서 이미

나누어져 있는 것이다.

3. 평점을 빼는 것은 범위로 나누는 것과 수학적으로 동일하다.
4. 이는 모든 알고리즘을 매우 효율적으로 연산할 수 있게 해준다.

정답 2번

정리하자면, 지금까지 평균 정규화에 대해 배웠고, 이는 협업 필터링 이전의 작업 단계라고 볼 수 있다. 데이터 세트에 따라서 이는 가끔 여러분의 알고리즘 구현이 좀 더 잘 될 수 있도록 도울 것이다.

Week 10

No.	Title
Week 10-1	Learning With Large Datasets

00'00"

지금부터는 large scale 머신 러닝을 학습하겠다. 이는 빅 데이터를 이용한 알고리즘이다. 머신 러닝의 최근 5년이나 10년간 역사를 돌아보면 알고리즘 학습이 5년 전과 비교해서 훨씬 잘 작동하는 이유는 알고리즘을 훈련할 수 있는 데이터 양이 엄청나게 늘어났기 때문이라는 것을 알 수 있다. 앞으로 강의에서는 엄청난 데이터를 가지고 알고리즘을 처리하는 법을 살펴보겠다.

00'31"

방대한 양의 데이터를 사용하려는 이유는 무엇인가? 앞서 살펴보았듯이 고성능의 머신 러닝 시스템을 도출하는 가장 좋은 방법 중 하나는 저편향(low bias) 학습 알고리즘을 사용해서 많은 양의 데이터를 통해 훈련시키는 것이다. 앞서 살펴본 예시 중 하나는 헷갈리는 어휘를 구별하는

것이었다. 예를 들어, 아래 예시에서 볼 수 있듯이 아침으로 계란 2개를 먹었다면, 그 결과는, 알고리즘에 많은 양의 데이터를 탑재하는 한, 잘 돌아가는 것처럼 보일 것이다. 따라서 결과적으로 머신 러닝에서는 가장 좋은 알고리즘을 가진 사람이 아니라 가장 많은 데이터를 가진 사람이 승자가 된다. 따라서 여러분은 방대한 데이터를 얻고 싶을 것이다. 최소한 이를 얻을 수 있는 시기라도 알고 싶을 것이다.

01'21"

하지만 여기에도 문제가 있다. 구체적으로는, 계산상의 문제가 있다. 예를 들어 트레이닝 셋 크기가 $M=100,000,000$ 이라 하자. 이 수치는 최근의 데이터로 볼 때 상당히 현실적인 수치이다. 미 인구조사통계국의 데이터를 보면 미국 인구는 3억명인데 여기서 수억 명의 자료기록을 얻을 수 있다. 인기 있는 웹사이트들의 트래픽 양을 살펴보면 수 억 건이 넘는 트레이닝 셋을 쉽게 얻을 수 있다. 선형 회귀 모델, 혹은 로지스틱 회귀(linear regression) 모델을 훈련하고 싶다면 기울기 하강 알고리즘을 적용하면 된다. Gradient를 계산하기 위해서 무엇을 해야 하는지 살펴본다면, 여기서 하는 얘기인데, 변수 term들을 계산해서 1단계 decent를 수행하기 위해서는 M 값이 1억이면 합계를 1억 번 수행해야 한다. 단지 기울기 하강을 한 단계 계산하기 위해 1억 번 엔트리를 합해야 하기 때문에, 앞으로 강의에서 이를 대체할 수 있는 방식이나 변수를 좀 더 효율적으로 계산하는 법을 살펴보겠다. 이번 large scale 머신 러닝 강의가 끝나면 심지어 오늘날의 1억개가 넘는 데이터에 대해서 선형 회귀(linear regression), 로지스틱 회귀(logistic regression), 신경망(neural network) 등 어떤 모형을 적용할지를 배우게 될 것이다. 물론, 1억 개의 예시로 모델을 훈련하는데 힘을 쏟기 전에 먼저 스스로에게 물어보자. 왜 그냥 예시 1,000개를 먼저 사용하지 않는가? 1억 개 예시 가운데 1,000개 예시로 이루어진 부분집합을 무작위로 선택하자. 그리고 1,000개의 예시를 가지고 알고리즘을 연습하자.

Q) $m=100,000,000$ 인 굉장히 큰 데이터 셋을 가지고 있고 supervised learning 문제에 맞닥뜨렸다. 데이터의 더 작은 부분 집합을 사용하는 대신 이 큰 데이터를 사용하는 것이 훨씬 좋은 성과를 낼 것이라고 어떻게 말할 수 있겠는가?

정답 4번

엄청나게 큰 모델을 실제로 개발하기 전에 이처럼 1,000개의 예시를 단위로 연습해보는 것은 대체로 좋은 sanity check를 해보는 셈이 된다.

03'27"

온전성 확인(sanity check)를 하는 법은 훨씬 적은 트레이닝 셋을 가지고 똑같은 방식으로 수행하는 것이다. 즉, 똑같이 잘 작동할 것 같은 훨씬 작은 $n=1,000$ 트레이닝 셋을 이용하는 것이다. 이것은 학습 곡선을 그리는 대표적인 방식이다. 만약 당신이 학습 곡선을 만들 예정이고 훈련 대상이 이처럼 생겼다면 이것은 J train θ 다. 만약 교차 검증 셋 목표를 가지고 있다면 J_{cv} of θ 는 이런 모양일 것이다. 그러면 이는 고분산(high-variance) 학습 알고리즘처럼 보이고 우리는 다른 훈련 예시를 더하면 더 나은 결과가 도출될 것이라는 자신감을 가질 수 있다. 반대로 학습 곡선을 그리려고 할 때 훈련 대상이 이렇게 생겼다면, 그리고 교차 검증 대상이 이런 모양이라면

classical high-bias 학습 알고리즘이 도출될 것이다. 이 경우, 예를 들어 $m=1,000$ 을 만들어내려고 해서 $m=500$ 에서 $m=1,000$ 이 된다면 m 을 1억으로 증가시킨다고 해도 더 나아질 것으로 보이지 않는다. 그렇다면 알고리즘을 조정하는 법을 알아내기 위해 애쓰기보다는 $n=1,000$ 을 고수하는 편이 낫다. 물론, 오른쪽에 가까운 상황이라면 한 가지 해결방법은 다른 수치를 더하거나 숨겨진 extra units을 신경망 알고리즘에 더하는 것이다. 그러면 왼쪽에 보이는 상황과 유사하게 끝날 것이다. 즉, $n=1,000$ 이 되고 이는 1,000개의 예시를 넘어서서 수치를 더해서 알고리즘을 바꾸려는 노력에 자신감을 불어넣어 줄 것이다. 이는 시간을 잘 활용하는 방법이다.

05'15"

Large-scale 머신 러닝에서 무척 큰 데이터 셋을 다룰 때, 우리는 계산상 합리적인 방법이나 효율적인 방법을 사용하고 싶어한다. 다음 강의에서는 두 가지 주요 아이디어를 살펴 볼 예정이다. 매우 큰 데이터 셋을 다루는 첫 번째 아이디어는 stochastic 기울기 하강이고 두 번째 아이디어는 맵리듀스(Map reduce)이다. 이러한 아이디어를 배우고 나면 아마도 학습 알고리즘을 빅 데이터에 맞춰서 조정할 수 있게 되고, 각양각색의 애플리케이션에서의 성능이 훨씬 좋아질 것이다.

No.	Title
Week 10-2	Stochastic Gradient Descent

00'00"

선형 회귀, 로지스틱 회귀, 신경망 네트워크를 포함해 많은 학습 알고리즘의 경우, 알고리즘을 유도하는 방식은 비용 함수를 이용하거나 최적화 객체를 사용하는 것이다. 그 다음, 기울기 하강 같은 알고리즘을 사용해 비용 함수를 최소화한다. 기울기 하강 트레이닝 셋은 커야 하며 이는 계산적으로 시간이 매우 오래 걸린다. 이번 강의에서는 기본적인 기울기 하강 알고리즘인 확률적 기울기 강하(Stochastic Gradient Descent)의 변형을 살펴보고 이를 통해 알고리즘을 훨씬 큰 트레이닝 셋에 적용시키는 법을 알아본다.

00'37"

기울기 하강을 이용해 선형 회귀 모델을 훈련한다고 생각해보자. 짧게 다시 떠올려보면, 가설은 다음과 같고, 비용함수는 m 개의 트레이닝 샘플에 대한 가설의 평균 오차 제곱들의 합의 $\frac{1}{2}$ 과

같으며, 활과 같은 모양이다. 따라서 $\theta=0$ 과 $\theta=1$ 변수에 대한 함수이므로, J 비용함수 또한 활 모양이다. 여기서 기울기 하강 알고리즘은 다음과 같은데, 해당 알고리즘의 내부 반복 문에서는 반복적으로 변수 θ 를 업데이트한다. 이제 이 강의의 남은 부분에서는 선형회귀를 실행 가능한 예제로 계속 사용할 것이다. 그러나 확률적 기울기 강하(Stochastic Gradient Descent)은 매우 일반적인 아이디어이며 로지스틱 회귀, 신경망, 그리고 다른 특정 트레이닝 셋에 기울기 하강을 훈련시키는 학습 알고리즘에도 응용된다.

01'38"

여기 이 그림이 기울기 하강의 역할을 보여준다. 만약 매개변수들이 저쪽에 있는 점들에서 초기화되었다면 기울기 하강 알고리즘을 반복한 결과, 매개변수들은 전역 최소점으로 수렴할 것이다. 즉, 매개변수들의 이동방향을 보면 바로 전역 최소점으로 향한다는 것을 알 수 있다. 이제 기울기 하강에서 문제는 m 이 클 때이다. 이 변수 term을 계산하는 것은 매우 오래 걸릴 수 있는데, m 예시들을 모두 더해야 할 수도 있기 때문이다. m 이 3억이라고 가정해보자. 미국 인구는 대략 3억명이다. 미국의 인구통계자료가 그 정도의 기록을 가지고 있을 것이다. 선형회귀모형이 이에 적합 하려면 3억개의 자료를 이용하여 덧셈을 수행하여야 한다. 그리고 이 작업은 매우 오래 걸릴 것이다. 따라서 이를 해결하기 위하여 특별한 버전의 기울기 하강이 필요한데, 이는 배치 기울기 하강(Batch gradient descent)이라 불린다. 배치(Batch)라는 용어는 우리가 모든 트레이닝 예들을 한번 살펴보는 행위를 가리킨다. 우리는 이를 모든 트레이닝 예시의 배치(batch)의 정렬이라고 부른다. 그리고 이것은 사실 가장 적절한 이름은 아니지만 보통 사람들이 이 같은 기울기 하강을 이렇게 부른다. 디스크에 3억 개의 인구통계조사 자료가 들어있다고 가정해보자. 알고리즘을 작동시키려면 미분계수들을 계산하기 위해 3억 개의 기록이 담긴 컴퓨터 메모리를 읽어야 한다. 그리고 이 기록들을 지속적으로 컴퓨터로 스트리밍 해야 한다. 왜냐면 이 모든 기록을 컴퓨터 메모리에 저장할 수 없기 때문이다. 따라서 당신은 이 기록을 꼼꼼하게 그리고 천천히 읽고 미분계수를 계산하기 위해 덧셈을 수행하여야 한다. 이 모든 과정을 끝마치면 기울기 하강이라는 다음 단계로 나아갈 수 있다. 이 단계에서는 같은 작업을 다시 한 번 더 해야 한다. 즉, 3억 개의 기록을 다시 한번 스캔하고 총합을 구한다. 이 작업이 끝나면 기울기 하강을 사용해서 또 다른 단계로 조금씩 나아갈 수 있다. 그리고 다음 단계에서 이 같은 작업을 다시 수행한다. 이 작업이 끝나면 세 번째 단계인데 같은 일을 반복한다. 따라서 알고리즘의 값이 수렴하는 데에는 굉장히 오랜 시간이 걸릴 것이다. Batch 기울기 하강과 대조적으로, 이번에는 매번 모든 트레이닝 예시를 살펴 볼 필요 없이 한 반복당 한 개의 트레이닝 예만 사용하는 알고리즘을 만들어 보자.

03'54"

새로운 알고리즘을 보기 전에 다시 한 번 여기 나와있는 Batch 기울기 하강 알고리즘을 보자. 이게 비용 함수이고, 업데이트 부분이고, 여기 term은 물론 기울기 하강 규칙을 사용한 것이다. J train의 θ 의 최적화 대상으로서, 매개변수 θ J 에 대해서 편미분계수(partial derivative)이다.

04'19"

이제 큰 데이터 셋에 잘 맞는 좀 더 효율적인 알고리즘을 살펴보자. Stochastic 기울기 하강 알고리즘을 끝내려면 이 벡터를, 비용함수를 다른 방법으로 정의한다. 즉, 매개변수 θ 의 비용을, 내 가설이 해당 $x(i)$, $y(i)$ 예들에 대해 얻는 제곱 합의 $\frac{1}{2}$ 이 되도록 할 때의 $x(i)$, $y(i)$ 에 맞추어 정한다. 이 비용 항은 내가 세운 가정이 1회 예시 $x(i)$, $y(i)$ 에서 얼마나 잘 작동하는지를 측정한다. 이제 전반적인 비용 함수 j 훈련이 이 동등한 form으로 적힐 수 있는지 주목하라. j train은 $x(i)$, $y(i)$. 예들에 대한 내 가설의 m 개의 트레이닝 예시들의 비용의 평균이다. 선형 회귀의 비용 함수라는 관점으로 무장하고 Stochastic 기울기 하강을 써보자. 확률적 기울기 강하의 첫 번째 단계는 데이터 셋을 무작위로 섞는 것이다. 무작위로 섞는다는 것은 m 트레이닝 예시들의 순서를 재조정하는 것이다. 이것은 일종의 표준적인 전처리 과정(pre-processing step)으로 1분 이내에 이러한 결과가 돌아온다. Stochastic 기울기 하강의 주 작업은 그 다음에 시작된다. 먼저 $i=1$ 에서 m 까지 반복할 것이다. 트레이닝 예시들을 반복해서 스캔하고 이어지는 업데이트를 수행한다. θ_j 변수를 $\theta_j - \alpha \times \frac{x(i)-y(i)\times j}{h}$ 로 업데이트 할 것이다. 모든 j 값을 위해 이러한 업데이트를 해보자. 여기서 이 항은 우리가 Batch 기울기 하강의 덧셈 연산 안에서 했던 것과 정확히 일치한다. 여러분이 미적분학을 할 수 있다면 여기 이 항이 $x(i)$, $y(i)$ 의 θ_j 에 대한 편미분계수와 같다는 것을 알 수 있을 것이다. 물론 비용은 이전에 정의한 것이다. 알고리즘을 정리하기 위해 먼저 여기 중괄호 부분을 닫겠다.

06'58"

확률적 기울기 강하가 하는 일은 트레이닝 셋을 스캐닝 하는 것이다. 먼저 첫 트레이닝 예시로 $x(1)$, $y(1)$ 을 보자. 이 예시들을 보는 것은 첫 트레이닝 예시의 비용이라는 관점에서 기본적으로 아주 작은 기울기 하강 단계이다. 달리 말해, 첫 번째 예시를 보고 변수들을 조금 변형해서 첫 번째 트레이닝 셋에 잘 들어맞도록 한다. 안쪽 루프 내에서 이 작업을 하면 두 번째 트레이닝 셋으로 나아갈 수 있다. 두 번째 트레이닝 셋에서는 변수 공간에서 이루어 질 것이다. 따라서 변수들을 조금 조정해서 두 번째 트레이닝 셋에 잘 맞도록 한다. 그리고 나서 세 번째 트레이닝 셋을 시작한다. 변수들을 변형해서 세 번째 트레이닝 예시에 좀 더 잘 맞도록 한다. 전체 트레이닝 셋을 완전히 익힐 때까지 이 과정을 반복한다. 이 최종적 반복문(ultra repeat loop)은 전체 트레이닝 셋을 여러 번 거칠 수도 있다. 이렇게 보면 Stochastic 기울기 하강은 왜 우리가 무작위로 데이터 셋을 섞기를 원하는지에 대한 동기를 부여한다. 하지만 여기서 트레이닝 사이트를 스캔할 때는 이것을 알 수 없다. 그래서 우리는 무작위로 분류된 순서에 따라 트레이닝 예시에 접근하는 것으로 끝나고 만다. 여러분의 데이터가 이미 무작위로 분류되었는지 아니면 원래 다른 특이한 순서로 분류되었는지에 따라 실제로는 Stochastic 기울기 하강의 수렴 속도를 조금 더 높일 수 있다. 데이터가 무작위로 분류되었는지 확실하지 않을 때는 무작위적으로 데이터 셋을 섞는 것이 좀 더 안전하다. 하지만 Stochastic 기울기 하강을 보는 더 중요한 시각은 확률적 기울기 강하가 gradient terms를 모든 m 트레이닝 예시에 대해 더하는 것보다는 강하법(descent)에 가깝다는 것이다. 따라서 우리는 이 gradient 항을 단일 트레이닝 예시로 사용해서 변수를 더 좋게 만들기 시작할 것이다. 따라서 3억개에 달하는 미인구조사통계국의 기록을 살펴볼 때까지 기다리는 것보다, 즉, 변수를 조금 변형하여 전역 최소값에 다가가기 위해, 모든 트레이닝 셋을 스캔하는 것보다 낫다. 대신 Stochastic 기울기 하강 대신에 단일 트레이닝

예를 볼 필요가 있다. 그러면 우리는 이미 변수들에 대해, 전역 최소값으로 다가가게 만드는 데에 진전을 보이는 셈이 된다.

09'24"

여기서 알고리즘을 다시 써보면 첫 번째 단계는 무작위로 데이터를 섞는 것이고 두 번째 단계는 실제로 작업을 수행하는 것으로, 단일 트레이닝 예시 $x(i)$, $y(i)$ 에 대해 업데이트가 이루어지는 것이다. 이제 이 알고리즘이 변수에 하는 것을 보자. 이전에 Batch 기울기 하강을 살펴볼 때, Batch gradient descent는 시간 안에 모든 트레이닝 셋을 살펴보는 알고리즘이었다. 이 알고리즘은, 알다시피, 이 같은 전역 최소값을 얻기 위해 적절한 직선 궤적(trajectory)을 그릴 것이다. Stochastic 기울기 하강과 비교해볼 때 모든 반복문 수행이 빠를 것이다. 왜냐면 우리는 모든 트레이닝 예시들을 합할 필요가 없기 때문이다. 하지만 모든 반복에서 단일 트레이닝 예를 더 잘 맞추는데 쓰인다. 따라서, 우리가 확률적 기울기 강하를 시작하려고 한다면, stochastic 기울기 하강을 이 같은 포인트에서 시작하자. 첫 번째 계산 과정은, 알다시피, 아마도 저 방향으로 변수를 가져갈 것이고 두 번째 계산 과정은, 운이 좋다면 두 번째 예시가 되겠지만, 하지만 이 같은 변수라면 운이 없어서 아마도 잘못된 방향으로 향하고 있을 것이다. 세 번째 계산 과정에서는 변수를 트레이닝 예시에 더 잘 맞게 맞춰서 이 방향으로 나가는 것으로 끝날 수도 있다. 이제 네 번째 샘플에서도 반복할 것이다. 다섯 번째, 여섯 번째, 일곱 번째도 마찬가지이다. 확률적 기울기 강하를 실행하면서 여러분은 이 알고리즘이 전역 최소값 방향으로 변수들을 밀기 시작한다는 사실을 발견할 것이다. 하지만 언제나 그런 것은 아니다. 전역 최소값을 보기 위해 좀 더 무작위적인 원형 경로를 택하자. 사실 Stochastic 기울기 하강을 실행하면 이 알고리즘은 Batch 기울기 하강과 같은 방식으로 수렴하지 않는다. 그리고 전역 최소값에 가까운 지역을 끊임없이 맴돌지만 전역 최소값에 가거나 거기에 머물지는 않는다. 하지만 실제로 이는 문제가 아니다. 변수들이 전역 최소값에 상당히 가까운 지역에 머무르는 것으로 끝나기만 하면 되기 때문이다. 그래서, 변수들은 전역 최소값에 가깝게 끝나고 이는 확률적 기울기 강하를 실행하면 대개의 경우 변수들을 전역 최소값 근처로 가지고 갈 수 있다는 좋은 가정을 할 수 있게 해주고 실제적인 목적으로는 이로써 충분하다.

11'44"

마지막으로 디테일을 하나 더 보자. Stochastic 기울기 하강에서 내부 반복문을 여러 번 수행하기 위해 외부 반복문을 반복하는 것을 보았다. 이를 얼마나 반복해야 하는가? 트레이닝 셋의 크기에 따라 다르지만 아마 한 번이면 충분할 것이다. 그리고 최대는 아마도 10 번 정도가 보통일 것이다. 따라서 이 내부 반복을 최소 1 번에서 최대 10 번까지 반복하면 될 것이다. 만약 여러분이 정말 엄청난 양의 데이터 셋, 예를 들었던 미 인구조사통계국의 기록 같은 예시, 즉, 3 억개의 예시를 가지고 있다면 트레이닝 셋을 단 한번만 살펴보는 것도 가능하다. 여기서 $i=1$ 부터 3 억까지이다. 이 반복을 한번 돌렸을 때 아마 완벽하게 좋은 가설을 얻을 수 있을 것이다. m 값이 굉장히 커서 내부 반복문을 딱 한 번만 해도 되는 경우가 있다. 대개, 1 부터 10 까지 데이터 셋을 패스(pass)하는 것이 꽤 보통일 것이다. 하지만 이것은 여러분이 가진 트레이닝 셋의 크기에

달려있다.

Q) Stochastic 기울기 하강에 대한 다음 설명 중 어느 것이 참인가? 정답 1,2,4 번

그리고 만약 이것을 Batch 기울기 하강과 비교해 본다면 전체 트레이닝 셋에 걸쳐 한번 살펴본 다음에 얻은 것은 단 한 단계의 기울기 하강을 행한 것과 같다. 따라서 이는 기울기 하강을 위한 작은 단계 중 하나로, 이 단계가 끝나면 또 작은 기울기 하강 단계를 밟게 된다. 이것이 바로 Stochastic 기울기 하강이 훨씬 속도가 빠른 이유이다. 이것이 바로 Stochastic 기울기 하강 알고리즘이다. 이 알고리즘을 실행하면, 아마도 여러분의 학습 알고리즘을 훨씬 큰 데이터 셋에 맞출 수 있고 성능 또한 좋아질 것이다.

No.	Title
Week 10-3	Mini-Batch 기울기 하강

00'00"

이전 강의에서 우리는 확률적 기울기 강하에 대해, 그리고 이 알고리즘이 Batch 기울기 하강 보다 얼마나 빠를 수 있는지 살펴보았다. 이번 강의에서는 Mini-batch 기울기 하강이라 불리는 또 다른 변수를 살펴보자. Mini-batch 기울기 하강은 Stochastic 기울기 하강보다도 좀 더 빠를 수 있다.

00'19"

지금까지 배운 알고리즘을 정리해보자. Batch 기울기 하강을 사용할 때 우리는 각 generation에서 m 을 예시로 사용할 것이다. 반면 Stochastic 기울기 하강에서는 각 generation마다 단일 예를 사용할 것이다. Mini-batch 기울기 하강은 이 두 알고리즘의 중간이다. Mini-batch 기울기 하강 알고리즘에서는 계산 과정마다 b 예시를 사용할 것이고 여기서 b 는 mini batch 크기라 불리는 변수이다. 그렇기 때문에 이 알고리즘이 Batch 기울기 하강과 Stochastic 기울기 하강의 중간인 것이다. 여기서 내가 훨씬 작은 batch 크기를 사용한다는 점을 빼고 batch 기울기 하강과 같다. 전형적인 예는 b 값을 10으로 잡는 것이다. 보통 범위는 $b=2$ 에서 $b=100$ 이다. 이것이 Mini-batch 크기로 전형적인 값이다. 한번에 한 가지 예나 m 예시를 사용하는 대신 b 예시를 사용할 것이다. 다시 제대로 써보면 b 값을 얻을 수 있다. 예를 들어 $b=10$ 이라 가정하자. 훈련 예시에서 다음 10개 예시를 얻을 수 있는데 그 중에는 x_i 나 y_i 가 포함될 수 있다. 만약 예가 10개라면 인덱싱은 $x_{(i+9)}$, $y_{(i+9)}$ 이 될 것이다. 따라서 예시는 모두 10개이고 이 예시를 사용해서 결국 기울기 하강을 수행할 것이다. 그래서 이는 $\text{rate} \times \frac{1}{10} \times \sum_{k=i}^{i+a} x(k) - y(k) \times x(k) \times j$ 이다. 이 표현식은 gradient terms를 10개 예에 대해 합한다. 이 10번, 내 미니 batch 크기가 다시 $i+a$ 가 되는데, 이는 b 선택에서 나오고, i 를 10까지 증가시키면 다시 다음 10개 예에 대해 작업을 수행하고 이런 식으로 계속된다. 이는 전체 알고리즘을 수행하기 위해서이다.

02'50"

여기 오른쪽 꼭대기를 인덱싱하는 작업을 간단히 하기 위해서 크기가 10인 mini-batch와 크기가 1,000인 트레이닝 셋을 사용하여 분류(sort)할 것이다. 여기서 한 스텝 당 $i=1$ 부터 21까지로 잡고 10개씩 예를 보기 때문에 10단계로 수행한다. 이 같은 기울기 하강 분류를 수행할 때 한 번에 10개의 예를 사용한다. 따라서 이 10과 $i+9$ 값은 내가 mini-batch값을 10으로 잡았기 때문에 얻은 결과이다. 이 최종 four loop는 여기서 991로 끝난다. 트레이닝 셋이 1,000개 있고 크기 10인 step을 100번 하면 얻을 수 있다. 이것이 바로 mini-batch 기울기 하강이다. 이를 batch 기울기 하강과 비교해보면 훨씬 진행이 빠를 것이다. 다시 한 번 실행 예시로 3억개의 미국 인구통계조사 데이터를 사용하자. 그러면 처음 10개 예만 살펴보아도 매개변수 θ 를 좋게 만들 수 있기 때문에 전체 트레이닝 셋을 다 스캔할 필요가 없다. 처음 10개의 예를 사용하고 이를 이용해서 progress를 만들고 그 다음에 다음 예 10개를 살펴보고 변수를 조금 조정하는 과정을 반복한다. 이것이 바로 Mini-batch 기울기 하강이 batch 기울기 하강보다 빠른 이유이다. 다시 말해, 3억 개의 예시를 다 스캔할 때까지 기다릴 필요 없이 단지 10개의 예만 살펴보고도 변수 조정이 가능하기 때문이다. 그러면 Mini-batch 기울기 하강과 확률적 기울기 강하를 비교해보면 어떨까. 확률적 기울기 강하를 하듯이 한 번에 단일 예로 사용하지 않고 한 번에 b 예시를 사용하기를 원하는 이유는 무엇인가? 그 답은 바로 벡터화(vectorization)이다. 특히, 좋은 벡터화된 입력 값을 가지고 있을 때 Mini-batch 기울기 하강이 Stochastic 기울기 하강을 능가할 가능성이 높다. 이 경우, 10개 예시의 합은 좀 더 벡터화된 방식으로 수행될 수 있는데 이는 10개 예시의 계산을 부분적으로 병렬화(parallelize)할 수 있다. 달리 말해, 적절한 벡터화를 사용하여 나머지 항을 계산함으로써, 부분적으로 좋은 수치 대수 라이브러리 (good numerical algebra libraries)를 사용할 수 있으며 gradient 계산을 b 예시를 가지고 병렬화할 수 있다. 반면 Stochastic 기울기 하강을 single example of time으로 보고 있다면 한번에 한 가지 예로 병렬화할 것이 많이 없을 것이다. 최소한 병렬화 작업을 덜 할 수 있다. Mini-batch 기울기 하강의 한 가지 단점은 당신이 다뤄야 할 Mini-batch 크기인 extra 매개변수가 b 이기 때문에 시간이 걸릴 것이라는 것이다. 하지만 좋은 벡터화된 초기 값(good vectorized implementation)을 가지고 있다면 Stochastic 기울기 하강보다도 더 빨리 돌아갈 수도 있다.

Q) 크기가 m 인 트레이닝 셋에서 Mini-batch 기울기 하강을 실행한다고 하자. 그리고 Mini-batch 기울기 하강의 크기는 b 이다. 이 알고리즘은 b 크기가 다음 중 어느 것과 같을 때 Batch 기울기 하강과 같아지겠는가?

정답 3번

05'48"

이제 Mini-batch 기울기 하강이 어떤 의미에서 확률적(Stochastic) 기울기 강하와 Batch 기울기 하강의 중간쯤이라는 표현을 이해할 수 있을 것이다. 이 알고리즘에서 합리적인 값을 b 로 선택한다고 가정하자. 나는 주로 $b=10$ 으로 놓고 다른 값은 2에서 100사이로 두는데 이것이 일반적인 선택이다. B 값을 선택하고 좋은 벡터화된 초기 값을 사용하면 확률적 기울기 강하나

Batch 기울기 하강 보다 더 빠를 수도 있다.

No.	Title
Week 10-4	Stochastic Gradient Descent Convergence

00'00"

이제 여러분은 확률적 기울기 강하 알고리즘을 배웠다. 이 알고리즘을 돌릴 때, 이 알고리즘이 어떻게 완벽하게 디버그 되고 제대로 수렴하는지 확실히 할 수 있을까? 또한 학습 속도(learning rate) 알파를 어떻게 확률적 기울기 강하로 조정하는가? 이번 강의에서는 이 같은 의문을 해결하는 기술에 대해 얘기할 것이다. 즉, 수렴을 확실히 하는 법, 그리고 학습 속도 알파를 택하는 법에 대해 알아볼 것이다.

00'27"

batch 기울기 하강을 사용하던 때를 떠올려 보자. 기울기 하강이 수렴하는지를 확실히 하는 표준적인 방식은 반복 수의 함수로써 최적화된 비용 함수를 그리는 것이다. 이것이 비용 함수이고 이 비용 함수가 매번 계산할 때마다 감소한다면 수렴한다. 트레이닝 셋의 크기가 작을 때는 이렇게 할 수 있다. 왜냐면 합계를 꽤 효율적으로 계산할 수 있기 때문이다. 하지만 트레이닝 셋의 크기가 굉장히 크다면 이 같은 방식으로 합하기가 어려워 알고리즘을 주기적으로 멈춰야 할 것이다. 여러분은 이 비용 함수를 계산하기 위해 확률적 기울기 강하를 주기적으로 멈추고 싶지 않을 것이다. 왜냐면 트레이닝 세트 전체 크기의 합을 내야 하기 때문이다. stochastic gradient의 요지는 알고리즘을 실행하는 도중에 전체 트레이닝 셋을 때때로 스캐닝하지 않고도 하나의 예시를 보는 것으로 대체할 수 있다는 것이다.

01'29"

따라서 확률적 기울기 강하에 있어서, 알고리즘이 수렴하는 것을 체크하기 위해서, 그 대신 우리가 할 수 있는 방법이 있다. 이전에 살펴보았던 비용의 정의를 가져오자. 단일 트레이닝 예의 변수 θ 의 비용은 그 트레이닝 예의 오차제곱의 $\frac{1}{2}$ 이다. 우리가 특정한 예로 트레이닝하기 직전에 확률적 기울기 강하를 학습한다. 확률적 기울기 강하에서는 x_i 와 y_i 예시를 순서대로 볼 것이다. 그리고 여기에 작은 업데이트를 할 것이다. 그리고 나서 다음 예시인 x_{i+1} , y_{i+1} 을 볼 것이다. 이것이 바로 stochastic 기울기 하강을 하는 방법이다. 알고리즘이 예시 x_i , y_i 에서 실행되고 있지만 변수 θ 를 예시를 가지고 업데이트하기 전에, 예시의 비용을 계산하자. 같은 말을 반복하지 않기 위해 조금 다른 표현을 사용하겠다. stochastic 기울기 하강은 특정 트레이닝 예시인 $x(i)$, $y(i)$ 를 사용해 θ 를 업데이트하기 전에 우리가 가진 트레이닝 세트를 스캐닝한다. 우리의 가정이 트레이닝 예시에 얼마나 잘 들어맞는지를 계산해보자. 우리는 이 작업을 θ 를 업데이트하기 전에 하고 싶어한다. 예시를 사용해 θ 를 업데이트 했다면 현재 예상되는 것보다 해당 예시에서 더 좋은 결과를 낼 것이다.

02'52"

마지막으로 확률적 기울기 강하의 수렴을 체크하기 위해 우리가 할 수 있는 일은 모든 계산 과정, 예를 들어, 천 번에 걸친 모든 계산 과정에서 이전 단계에서의 비용을 계산하는 것이다. 우리는 이 비용 평균을 계산할 수 있다. 예를 들어 알고리즘이 의해 계산된 마지막 1,000 개 예의 비용 평균 말이다. 이렇게 하면, 알고리즘이 잘 작동하는지에 대한 예측치를 얻을 수 있다. 그러니까, 1,000개 트레이닝 예시에서 여러분의 알고리즘이 잘 작동하는지 알 수 있다. 전체 트레이닝 셋을 스캔 해야 하는, J_{train} 을 주기적으로 계산하는 것과 대조적으로 확률적 기울기 강하의 부분으로서, 이 작업을 함으로써, 변수 θ 를 업데이트 하기 전에 이 비용들을 계산하지 않아도 된다.

03'48"

여기 그래프가 어떤 모양일지 보여주는 예시가 몇 개 있다. 1,000개의 예시를 가지고 비용 평균을 그렸다고 가정해보자. 1,000개의 예시에 대한 평균이기 때문에 조금 매끄럽지 않을 수 있다. 그리고 매번 계산 과정마다 감소하지 않을 수도 있다. 그러면 다음과 같은 그래프를 얻을 수 있다. 예를 들어 천 개의 트레이닝 셋의 작은 부분 집합으로 평균을 내면 그려진 그래프가 깔끔하지 않다. 다음과 같은 그래프를 얻는다면 알고리즘이 꽤 괜찮게 작동한다고 할 수 있다. 아마도 여기서부터 시작해서 비용이 내려가고 여기 고원 부분이 평평해진 것처럼 보인다. 아마도 여러분의 비용이 이렇게 되고 학습 알고리즘이 수렴한 것처럼 보일 것이다. 더 작은 학습 속도를 사용하면, 알고리즘이 애초부터 천천히 학습하고 비용 또한 천천히 하락한다. 하지만 결국에는 더 작은 학습 속도를 통해 알고리즘이 아마 아주 조금 더 나은 솔루션을 내놓을 수 있을 것이다. 그래서 빨간 선은 확률적 기울기 강하의 더 작은 학습 속도를 이용할 때의 모습을 나타낸다. 이렇게 나타나는 이유는, 기억하겠지만, stochastic 기울기 하강이 전역 최소값으로 수렴하지 않고 변수들이 전역 최소값 주위를 왔다 갔다 할 것이기 때문이다. 따라서 더 작은 학습 속도를 사용하면 더 작은 최소값 변동 폭을 가질 수 있다. 이 작은 차이는 때로 간과되지만 이 작은 차이로 변수를 위한 더 좋은 값을 얻을 수 있다.

05'21"

여기 일어날 수 있는 또 다른 상황이 있다. 확률적 기울기 강하를 실행하고 있다고 가정해보자. 그리고 이 비용들을 그리면서 천 개의 예시를 평균한다. 이것이 바로 다른 그래프들 중 하나의 결과이다. 이것은 수렴된 것처럼 보인다. 이 숫자, 1,000을 가지고 5,000개 예시를 평균할 때까지 증가시킨다. 그러면 이렇게 생긴 더 완만한 곡선을 도출할 수 있다. 1,000개 대신 5,000개의 예시를 평균하면 이 같은 더 부드러운 곡선을 얻을 수도 있다. 이것이 바로 여러분이 평균하는 예시의 개수를 증가시키는 작업의 효력이다. 하지만 예시 개수를 너무 많이 늘리는 것의 단점은 5,000개 예시 모두에서 one date point를 얻는다는 것이다. 여러분의 학습 알고리즘이 잘 하고 있는지에 대한 피드백은 지연되는데 이는 1,000개 대신 5,000개의 예시에서 매번 one data point를 그려야 하기 때문이다.

06'22"

같은 관점에서 보면, 가끔 기울기 하강을 실행하고 이렇게 생긴 그래프로 끝날 수도 있다. 그리고 다음과 같은 그래프에서 비용은 전혀 증가하지 않는 것처럼 보인다. 전혀 학습하지 않는 알고리즘이처럼 보인다. 여기 평평한 곡선과 비용이 떨어지지 않는 것처럼 보인다. 하지만 여러분이 많은 예시에 대한 평균을 늘리고 싶다면 여기 빨간 선 같은 것을 볼 수 있고 비용이 사실 감소하는 것처럼 보인다. 두 세 개의 예를 평균하는 파란 선이 있다. 이 파란 선은 깔끔하지 않아서 실제로는 감소하는 비용의 실제 추이를 볼 수 없으며, 1,000개 대신 5,000개의 예시를 평균하는 것이 가능하다는 것을 못 보게 한다. 물론 우리는 많은 개수의 예시를 평균했으며 여기서는 5,000개의 예시를 평균했다. 다른 색을 사용하겠다. 여기서 학습 곡선이 이렇게 끝나는 것을 볼 수 있다. 많은 개수의 예시를 평균할 때 학습 곡선이 여전히 평평하다. 그렇다면 그 이유가 무엇이건 이 알고리즘이 불행하게도 그다지 많이 학습하고 있지 않다는 확실한 증거가 될 것이다. 따라서 학습 속도를 변화시키거나 값을 변화시키거나 알고리즘의 무언가를 바꿔야 한다.

07'35"

마침내, 마지막으로 볼 수 있는 것은 이 곡선들을 그리면 이렇게 생긴 곡선을 볼 수 있는데 실제로 증가하는 것처럼 보인다. 그렇다면 이것은 알고리즘이 발산한다는 신호다. 그러면 여러분은 학습 속도 알파의 더 작은 값을 사용해야 한다. 이것은 여러분이 예시들의 범위에 대한 비용 평균을 그릴 때 현상의 범위에 대한 감각을 알도록 해주면서 동시에 다른 그래프에 대한 반응으로 여러분이 시도할 다른 것들을 제안해 줄 것이다. 따라서 만약 그래프가 깔끔하지 못하거나 상하 변동이 크다면 여러분이 평균 내는 예시의 개수를 늘려서 순서도의 전반적인 추이를 잘 볼 수 있게 하라. 그리고 예러가 실제로 증가하는 것 그리고 가격이 증가하는 것을 본다면 더 작은 알파 값을 사용해 보라.

08'28"

마지막으로, 학습 속도에 대해 좀 더 살펴보는 것은 그만한 가치가 있다. 확률적 기울기 강하를 실행할 때, 알고리즘이 여기서 시작해서 최소값을 향해 간다는 것을 보았다. 그리고 이 알고리즘은 실제로 수렴하는 대신 최소값 주변을 영원히 맴도는 것을 보았다. 그리고 변수 값이 전역 최소값에 가까이 있지만 전역 최소값에 정확하게 다가가지는 않는다. 확률적 기울기 강하에서 가장 평범한 가정은 학습 속도 알파를 일정하게 유지하는 것이다. 그리고 우리는 다음과 같은 결과를 가지게 된다. 확률적 기울기 강하를 사용해서 실제로 전역 최소값으로 수렴하고 싶다면 한 가지 방법이 있다. 바로 시간을 들여 학습 속도 알파를 서서히 증가시키는 것이다. 이렇게 하려면 보통 α 가 다음과 같다고 놓는 것이다. 여기서 i 는 반복 횟수이므로, 트레이닝 예시의 개수이다. const1과 const2는 알고리즘의 추가적인 변수들로 더 좋은 성능을 위해서 설정해야 한다. 사람들이 이를 기피하는 이유 중 하나는 이 두 가지 추가적인 변수들을 설정하는데 시간을 쏟아야 하고 따라서 이 알고리즘이 까다롭게 되기 때문이다. 잘 알다시피, 알고리즘을 잘 작동시키려면 변수들을 더 잘 다루어야 한다. 하지만 변수들을 잘 조정하게 되면 최소값을 향해 가는 알고리즘을 얻을 수 있다. 하지만 여러분이 학습 속도를 줄임에 따라

최소값에 가까이 갈 수 있고, 변동치가 전역 최소값에 갈 때까지 점점 더 작아질 것이다. 이를 잘 이해하기를 바란다. 이 공식이 말이 되는 이유는 알고리즘이 실행될 때 계산 과정 숫자가 커지기 때문이다. 그래서 알파가 서서히 작아지고 전역 최소값으로 수렴하기 전까지 여러분은 더 작은 단계를 밟을 것이다. 그래서 천천히 알파를 0까지 감소시키면 더 좋은 가정을 얻을 수 있다. 하지만 상수들을 다루는데 추가적인 노력이 필요하고 솔직히 말해 전역 최소값에 가까운 변수 값이라면 무엇이든 좋다. 다른 사람들이 두 가지 방법 모두를 사용하는 것을 볼 수 있지만 대개는 알파를 서서히 줄이는 과정은 잘 시행하지 않고 학습 속도 알파를 일정하게 유지하는 것이 확률적 기울기 강하에서는 좀 더 보편적인 접근법이다.

11'00"

이번 강의를 요약하면 우리는 확률적 기울기 강하의 비용 함수를 최적화하는 법을 모니터링하는 방법을 살펴보았다. 그리고 이 방법이 전체 트레이닝 세트에 대한 비용 함수를 계산하기 위해 트레이닝 세트 전부를 주기적으로 스캐닝할 필요가 없다는 점을 살펴보았다. 대신 마지막 1,000 개 정도 예만 살펴보면 된다. 그리고 이 방법을 사용해 확률적 기울기 강하가 잘 작동해서 수렴하는지를 알거나 학습 속도 알파를 조정하는데 사용할 수 있다.

No.	Title
Week 10-5	Online Learning

00'00"

이번 강의에서는 온라인 학습 셋팅이라 불리는 새로운 대규모 데이터에 대한 머신 러닝 셋팅을 다룰 것이다. 온라인 학습 셋팅은 끝없이 밀려드는 데이터의 흥수 또는 스트림 속에서 우리가 갖는 문제들을 모델화하고 알고리즘이 학습할 수 있게 해준다. 오늘날 많은 대규모 웹사이트들, 혹은 대규모 웹사이트 회사들은 각기 다른 버전의 온라인 학습 알고리즘을 사용해서 계속 들어오고 나가는 사용자들의 흥수로부터 학습하도록 한다. 구체적으로는 여러분의 웹사이트에 접속하는 사용자의 끝없는 스트림에서 생성되는 데이터의 끊임없는 스트림 속에서 여러분이 할 수 있는 일은 온라인 학습 알고리즘이 데이터의 스트림으로부터 사용자의 선호를 학습하도록, 그리고 웹사이트 상에서 일부 결정들을 최적화하도록 하는 것이다.

00'50"

여러분이 배송 업체를 운영한다고 가정해 보자. 고객이 와서 패키지를 장소 A에서 장소 B로 배송해달라고 요청한다. 아니면 여러분이 웹사이트를 운영하는데 유저들이 여러 번 방문해서 패키지를 어디서 어디로 보내고 싶다고 (즉, 출발지와 도착지) 계속 요청한다고 생각해보자. 여러분은 웹사이트 상에서 일정 가격을 받고 패키지 배송 서비스를 제공한다. 예를 들어 어떤 패키지는 50달러에, 어떤 패키지는 20달러에 배송해준다. 그리고 고객에게 제공하는 가격에 따라, 고객은 배송 서비스를 선택한다. 이것은 긍정적인 예시이다. 때로는 고객들이 가버리거나 여러분의 배송 서비스를 선택하지 않기도 한다. 여기서 만약 학습 알고리즘을 사용해서 사용자에게 제공하고 싶은 요구 비용을 최적화하고 싶다고 해보자. 구체적으로, 사용자들의 특성을 잡아내는 기능을 고안해냈다고 해보자. 만약 우리가 인구구조에 대해 아는 바가 있다면, 우리는 패키지의 출발지와 도착지를, 사용자들이 패키지를 배송하고 싶어하는 장소를 캡쳐할 수 있을 것이다. 또한 패키지 배송을 제시하는 가격도 알 수 있다. 여기서 우리가 알고 싶은 것은 고객들이 패키지 배송을 선택할 확률이다. 즉, 주어진 조건들 하에서 고객들이 우리의 배송 서비스를 이용할 것인지 여부이다. 그리고 노파심에서 말하는데 이러한 값 x 는 우리가 요구하는 가격 또한 나타내어야 한다.

02'14"

그렇다면 우리는 고객이 주어진 가격 하에서 우리의 서비스를 이용할 확률을 추정할 수 있다. 그렇게 되면 고객이 꽤 높은 확률로 우리의 웹 사이트를 선택하도록 하는 가격을 정할 수 있다. 동시에 고객에게도 꽤 높은 편익을 제공할 수 있게 된다. 즉, 패키지 배송 서비스를 이용하는 것이 이익이 되도록 만들어 줄 수 있다. 따라서 우리는 어떤 주어진 가격과 조건에서 $y=1$ 일 때의 성질을 배워서 새로운 사용자가 진입할 때 적절한 가격을 선택할 수 있다.
따라서 $y=1$ 인 확률을 모델화하기 위해서 우리는 로지스틱 회귀나 신경망 네트워크 또는 이와 유사한 다른 알고리즘을 사용할 수 있다. 여기 온라인 학습 알고리즘의 경우를 보자. 나는 이 과정을 영원히 반복할 것이다. 이는 우리의 웹 사이트가 계속 갱신된다는 것을 뜻한다.

03'11"

이제 이 웹사이트 상에서 벌어지는 일은 때로 사용자가 방문하고 이 사용자를 위해서 우리는 웹사이트의 사용자나 고객에게 대응하는 x, y 쌍을 얻는 것이다. 여기 x 들은 고객이 지정한 출발지와 도착지이고 우리가 당시 고객에게 서비스를 제공한 가격이다. y 는 1이거나 0이고 이는 고객이 우리의 서비스를 이용하거나 이용하지 않는다는 것을 뜻한다. 이제 이 $\{x, y\}$ 를 얻고 나서 온라인 학습 알고리즘이 하는 일은 이것을 사용해 변수 θ 를 업데이트하는 것이다. 그리고 특히 우리는 변수 θ_j 를 $\theta_j - \alpha(h_\theta(x) - y) * x_j$ 로 학습속도 알파와 회귀에 대한 기울기 하강 rule을 이용하여 업데이트 할 것이다. $J=0$ 에서 n 까지 이렇게 한다. 그리고 대괄호를 닫을 것이다.

04'20"

다른 학습 알고리즘에 대해 나는 X_i, Y_i 를 썼으나 이번 온라인 학습 셋팅에서는 고정된 트레이닝 셋이 있다는 생각을 버리고 $X-Y$ 를 이용하여 알고리즘을 사용하자. 우리가 예를 얻고 나면 우리는 그 예를 사용하는 법을 배우고 또 그 예를 버리는 법을 배운다. 한 번 버리고 나면 그 예를 다시

사용하지 않을 것이다. 우리가 한 번에 한 가지 예만 사용한 이유이다. 우리는 이 예시로부터 배우고 폐기한다. 이것이 바로 우리가 i로 표시된 고정된 트레이닝 셋이 있다는 식의 생각을 버리는 이유이다. 그리고 만약 여러분이 사용자 스트림이 끝없이 이어지는 규모가 큰 웹사이트를 정말로 운영한다면 이 같은 온라인 학습 알고리즘이 무척 합리적인 선택이 될 것이다.

05'10"

여러분이 데이터를 많이 가지고 있다면 본질적으로 데이터에서 자유롭다고 할 수 있으며 데이터는 사실 무제한적이고 따라서 트레이닝 예시를 한 번 이상 볼 필요가 없다. 물론 우리의 사용자 수가 적다면, 온라인 학습 알고리즘을 사용하는 대신, 우리의 데이터를 고정된 트레이닝 셋에 저장하고 트레이닝 셋을 이용해서 알고리즘을 실행하는 편이 낫다. 하지만 여러분이 끝없는 데이터 스트림을 가진다면, 온라인 학습 알고리즘은 매우 효과적일 수 있다. 온라인 학습 알고리즘의 한 가지 흥미로운 효과를 언급하겠다. 그것은 바로 사용자 선호에 맞춰 변할 수 있다는 것이다. 예를 들어, 만약 시간이 지나면서 경제에서 어떤 변화가 생기면 사용자들은 가격에 좀 더 민감하게 변하고 높은 가격을 지불하는 것을 꺼리게 된다.

06'03"

혹은 가격에 덜 민감해져서 반대로 높은 가격을 지불할 수도 있다. 또는 사용자에게 다른 요소가 좀 더 중요해질 수도 있다. 새로운 유형의 사용자에 여러분의 웹사이트에 유입된다면 말이다. 이 같은 종류의 온라인 학습 알고리즘은 사용자 선호에 맞춰 변할 수 있고 가격을 지불할 용의가 있는 사용자 구성이 바뀌는 것을 계속 놓치지 않고 파악할 수 있다. 여러분의 사용자 풀(pool)이 변하기 때문에 여러분의 매개변수 θ 의 업데이트가 가장 최근의 사용자 풀(pool)에 맞도록 매개변수를 서서히 조정하기 때문이다. 온라인 학습 알고리즘의 또 다른 활용 예가 있다. 제품 조사에 온라인 학습 알고리즘을 적용해서 사용자에게 질 좋은 조사 목록을 제공할 수 있다. 예를 들어 여러분이 휴대폰을 판매하는 온라인 매장을 운영한다고 하자. 즉, 모바일 폰이나 휴대폰을 판매한다. 그리고 여러분은 사용자가 방문해서 "안드로이드 폰 1080 화소 카메라"라는 키워드를 검색창에 칠 수 있는 인터페이스를 구축했다. 1080 화소는 여러분이 전화기, 휴대폰, 그리고 모바일 폰에 장착되어 있는 카메라 유형이다.

07'08"

예를 들어 여러분의 매장에 휴대폰이 백 개 있다고 하자. 여러분의 웹 사이트가 정렬된 방식 때문에 사용자가 검색 창에 제품 유형을 입력하면, 다른 종류의 휴대폰이 10개 뜰 것이다. 우리가 하려는 것은 학습 알고리즘을 사용해서 100개의 휴대폰 가운데 어떤 10개를 찾아내서 여기 보이는 것처럼 사용자의 검색에 대응하는 것이다. 여기 문제 해결법이 있다. 각각의 휴대폰과 사용자 문의를 가지고 벡터 x 를 구성할 수 있다. 벡터 x 값은 휴대폰의 여러 기능을 잡아낼 수 있다. 즉, 사용자의 검색 문의가 휴대폰 기능에 얼마나 맞아떨어지는지를 판단한다. 사용자가 검색한 내용과 휴대폰의 설명이 얼마나 맞아떨어지는지 등을 본다. 그러니까 x 값은 휴대폰의 기능과 사용자의 검색문의 내용이 얼마나 유사한지를 다른 차원들을 따라 잡아내는 것이다.

08'10"

우리가 하려는 것은 사용자가 특정한 폰을 보려고 링크를 클릭할 확률을 계산하는 것이다. 우리는 사용자가 구매하기를 원할 만한 휴대폰을 보여주고 싶기 때문이다. 사용자가 웹 브라우저에서 클릭할 확률이 높은 폰을 보여주기를 원한다. 따라서 나는 사용자가 폰을 클릭하는 것을 $y=1$ 로 놓고 그렇지 않은 경우를 $y=0$ 으로 놓을 것이다. 나는 사용자가 주어진 휴대폰 검색 목록에서 특정한 휴대폰을 클릭할 확률을 알고 싶다. 여기 x 값은 휴대폰 문의와 휴대폰의 기능이 얼마나 맞는지를 확인한다.

08' 40"

이러한 웹사이트를 운영하는 사람들의 언어로 이 문제에 이름을 붙이면, 이런 학습의 문제는 클릭률(click-through rate) 예측 문제라 불린다. 줄여서 CTR문제라 부른다. 사용자가 주어진 검색 결과 가운데서 특정 링크를 클릭할 확률을 의미한다. CTR은 click through rate의 약자이다. 여러분이 만약 어떤 특정 폰에 대해 CTR을 측정한다면, 우리가 할 수 있는 일은 먼저 이것을 사용해서 사용자에게, 가장 클릭할 확률이 높은 폰 10개를 보여주는 것이다. 100개의 폰에 대해 우리는 $p(y = 1|x; \theta)$ 를 계산해서 사용자가 가장 클릭할 확률이 높은 폰 10개를 고른다. 이는 어떤 폰 10개를 사용자에게 보여줄지를 결정하는 굉장히 합리적인 방법이다.

09' 26"

분명하게 하자면, 사용자가 매번 검색할 때마다, 우리는 열 개의 검색 결과로 돌아간다. 그러면 우리는 (x,y) 쌍을 10개 갖게 되는데, 이것은 사용자가 우리 웹사이트를 방문할 때마다 우리에게 10개의 트레이닝 예시를 준다는 것이다. 우리가 사용자에게 보여줄 휴대폰 10개에 대해 각각 벡터 x 값을 얻고 사용자에게 폰 10개를 보여줄 때마다 y 값을 얻는다. URL을 클릭하는지의 여부에 따라 y 값을 얻는다. 이러한 웹사이트를 운영하는 하나의 방법은 사용자에게 끊임없이, 여러분이 보기에도 사용자가 가장 좋아할만한 선택지 10개를 제공하는 것이다. 따라서 사용자가 방문할 때마다 여러분은 10개의 예시, (x,y) 쌍을 10개 갖게 되고, 이를 온라인 학습 알고리즘에 사용해, 매개변수를 업데이트하는데 사용한다. 이 10개의 예시에 대해 기울기 하강의 10 단계를 사용해서 데이터를 버릴 수 있다. 사용자가 끊임없이 웹사이트에 유입된다면 이 방법이 여러분의 알고리즘의 매개변수를 업데이트하는 무척 합리적인 방법이 될 것이다. 이렇게 해서 사용자에게 가장 클릭할 확률이 높은 폰 10개를 보여줄 수 있다.

10'37"

지금까지 제품 검색 문제 또는 폰을 정렬하는 법, 폰을 검색하는 법을 배웠다. 다른 것들도 짧게 언급하겠다. 하나는 만약 여러분이 웹사이트를 가지고 있고 사용자에게 어떤 특별한 제안을 할 것인지 결정하려고 한다면, 이는 폰과 유사한 경우이고, 여러분이 웹사이트를 가지고 있어서 다른 사용자에게 다른 뉴스 기사를 제시하려고 한다고 하자. 그래서 만약 여러분이 뉴스 기사를 모아 놓은 웹사이트를 운영한다면, 여러분은 앞서 살펴본 것과 유사한 선택 시스템을 사용해서 사용자에게 사용자가 가장 흥미를 가지고 클릭할 확률이 높은 뉴스 기사를 제공할 수 있다. 특별 제안에 관해서는 우리는 '추천'을 통해 이익을 남길 수 있다.

11'15"

사실, 여러분이 협력적 필터링 시스템(collaborative filtering system)을 가지고 있다면 여러분은 이를 활용해서 로지스틱 회귀 분류기(classifier)를 사용해서 추가적인 값을 얻어서 여러분이 사용자에게 추천할 다른 여러 가지 제품의 클릭률을 예측할 수 있다. 물론, 나는 이 중 어떤 문제도 고정된 트레이닝 셋을 가지는 표준적인 머신 러닝 문제로 공식화될 수 없었다는 점을 알려야겠다. 어쩌면 여러분은 웹사이트를 며칠 운영하고 트레이닝 셋, 고정된 트레이닝 셋을 저장해서 이것을 가지고 학습 알고리즘을 실행할 수도 있다. 하지만 실제 현장에서 이러한 문제를 해결하는 방식을 보면, 여러분은 대기업들이 데이터가 너무 많아서 고정된 트레이닝 셋을 저장할 필요 없이 온라인 학습 알고리즘을 사용해서 사용자가 웹사이트에서 생성하는 데이터로부터 끊임없이 학습하는 것을 볼 것이다.

Q) 다음 중 온라인 알고리즘의 장점은?

정답 1,3번

12'04"

지금까지 온라인 학습 설정에 대해서 보았다. 여러분이 보았듯이 이 알고리즘은 stochastic 기울기 하강 알고리즘과 굉장히 유사했다. 차이가 있다면 고정된 트레이닝 셋을 스캐닝하는 대신 한 사용자로부터 한 개의 예시를 얻어서 이 예시에서 학습하고 버리고 나아가는 것이었다. 따라서 데이터 스트림이 끊임없다면 이 알고리즘을 사용하는 것을 고려해보라. 물론 온라인 학습의 장점은 여러분의 사용자 집단이 바뀌거나 여러분이 예측하려는 것들이 사용자의 기호에 따라 서서히 변하는 경우에 온라인 학습 알고리즘이 가장 최근의 사용자 행동 변화에 맞춰서 여러분의 설을 변화시킨다는 점이다.

No.	Title
Week 10-6	Map Reduce and Data Parallelism

00' 00"

앞선 강의에서 우리는 stochastic 기울기 하강을 살펴보았다. 뿐만 아니라 다른 알고리즘, stochastic 기울기 하강 알고리즘, 온라인 학습 알고리즘의 활용 등등도 배웠다. 하지만 이 모든 알고리즘들은 한 개의 머신에서 작동하거나 한 개의 컴퓨터에서 작동한다. 그리고 어떤 머신 러닝 문제들은 한 개의 머신으로 작동시키기에 너무 클 수 있다. 여러분이 가진 데이터가 너무 많아서 어떤 알고리즘을 사용하든 간에 하나의 컴퓨터에서 모든 데이터를 작동시키고 싶지 않을 수도 있다.

00' 28"

따라서 이번 강의에서는 대규모 머신 러닝에 대한 다른 접근법인 맵리듀스(map reduce)에 대해 얘기하겠다. stochastic 기울기 하강을 여러 강의를 통해 배웠고 맵리듀스는 상대적으로 짧은

시간을 할애하지만 이 두 가지를 내가 강의한 시간에 비례해 중요성을 저울질하지 않기를 바란다. 많은 사람들이 맵리듀스가 똑같이 중요하다고들 한다. 어떤 사람들은 심지어 맵리듀스가 기울기 하강보다 더 중요하다고 한다. 단지 설명하기 더 쉽기 때문에 할애하는 시간이 많지 않을 뿐이다. 하지만 맵리듀스를 사용하면 여러분은 stochastic 기울기 하강을 사용할 때보다 훨씬 더 큰 문제들에 학습 알고리즘을 조정할 수 있게 될 것이다.

01'25"

여기 방법이 있다. 우리가 선형 회귀 모델이나 로지스틱 회귀 모델 등등에 적용하고 싶다고 가정하자. 먼저 다시 한번 batch 기울기 하강으로 시작하자. 이것이 batch 기울기 하강 학습 규칙이다. 여기 적힌 공식을 추적하기 위해 나는 $m=400$ 예를 쓸 것이다. 물론, 우리의 관점에서 보면, 대규모 머신 러닝의 관점에서는 m 값이 상당히 작은 수치이지만 이 수치는 4억처럼 큰 수치보다는 문제들에 자주 적용될 수 있다. 하지만 이 슬라이드에 적은 것을 간단히 하기 위해서 나는 400을 사용할 것이다. 그래서 이 경우, batch 기울기 하강 학습 규칙은 여기 400과 $i=1$ 부터 $i=400$ 까지의 합을 가진다. 만약 m 값이 크다면 이는 계산적으로 시간이 오래 걸릴 것이다.

02'16"

그래서 맵리듀스가 하는 일은 다음과 같다. 맵리듀스는 두 명의 연구자, Jeff Dean과 Sanjay Ghemawat가 개발했다. Jeff Dean은 실리콘 밸리의 전설적인 엔지니어 중 한 명으로 오늘날 구글이 운영하는 것들의 구조적 기반 중 대부분을 만들었다. 여기 맵리듀스가 무엇인지 보자. 예를 들어 내가 가진 트레이닝 셋이 이렇다고 하자. 여기 상자 모양으로 그리겠다. 여기 $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$ 이다. 이것이 나의 400개 트레이닝 예들이다.

03'04"

맵리듀스에서는, 이 트레이닝 셋을 다른 부분 집합으로 나눈다. 그리고 내가 트레이닝 셋을 병렬적으로 처리할 컴퓨터 4대, 혹은 머신 4 대를 가지고 있다고 하자. 4대의 머신으로 데이터를 나눌 것이기 때문이다. 만약 여러분이 10대 혹은 100대를 가지고 있다면 트레이닝 셋을 10개, 100개로 나눌 수 있다. 내가 이 4대의 머신을 가지고 할 일은 내 트레이닝 셋의 처음 $\frac{1}{4}$ 부분을, 그러니까 100개의 예시를 먼저 가지고 시작한다. 특히, 이 합계식으로 첫 예시 100개를 계산한다. 즉 여기 써보면 $temp_j^{(1)} = \sum_{i=1}^{100} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$ 이다. 시그마 다음 부분은 위의 이 식에서 가져온다. 그러니까 여기 위에 있는 기울기 하강 공식이다.

04'27"

비슷하게, 나는 4등분한 데이터의 두 번째 부분을 두 번째 머신으로 보낼 것이다. 두 번째 머신은 $(x^{(101)}, y^{(101)}) \dots (x^{(200)}, y^{(200)})$ 트레이닝 예시를 사용할 것이다. 그리고 비슷한 변수인 $temp_j^{(2)} = \sum_{i=101}^{200} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$ 을 통해서 101부터 200까지 예시를 계산한다. 같은 방식으로 세 번째 머신과 네 번째 머신에서 세 번째 부분 예시, 네 번째 부분 예시를 사용한다. 이제 각각의 머신은 400개의 예시를 합하는 대신에 100개만 합하면 된다. 이는 전체 작업량의 $\frac{1}{4}$ 만 수행하면 되므로 아마도 4배 정도 속도가 빨라질 것이다. 마지막으로 이 4대가 작업을 끝내면 나는 temp

변수들을 사용해서 중간 결과물을 합산할 것이다. 이 변수들을 중앙화된 마스터 서버로 보내서 이 결과물을 합산할 것이다. 특히 이는 나의 매개변수 θ_j 를 업데이트 시킬 것인데 θ_j 는 다음과 같은 공식에 따라 업데이트 된다. $\theta_j = \theta_j - \alpha \cdot \frac{1}{400} (\text{temp}_j^{(1)} + \text{temp}_j^{(2)} + \text{temp}_j^{(3)} + \text{temp}_j^{(4)})$. 여기서 여러분도 알겠지만, $j=0, \dots, n$ 까지 독립적으로 계산해야 한다. 이 값들 안에서 말이다. 따라서 이 방정식을 따로 따로 계산하는 과정이 분명하게 이해되기를 바란다.

06'07"

이 방정식이 하는 일은 정확히 말해서 바로 이 공식과 같다. 여러분이 중앙화된 마스터 서버로 이 결과물을 보내서, 즉 $\text{temp}_j^{(1)}, \text{temp}_j^{(2)}, \text{temp}_j^{(3)}, \text{temp}_j^{(4)}$ 를 보내서 이 값을 합하면, 이 네 개 값들의 합은, 바로 우리가 원래 구하려고 했던 batch stream descent의 $\sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$ 값과 같다. 그러면 여러분은 α 곱하기 $\frac{1}{400}$, α 곱하기 $\frac{1}{100}$ 를 가지면 이는 정확히 batch 기울기 하강과 같다. 유일한 차이점은 한 개의 머신에서 트레이닝 예시 400개를 합할 필요 없이 이를 4 대의 머신으로 나눠서 작업한다는 것이다.

07'00"

여기 맵리듀스 기술이 어떤 것인지를 보여주는 그림이 있다. 여기 트레이닝 셋이 있고, 우리는 4대의 머신에 걸쳐 병렬화하려고 한다. 이 트레이닝 셋을 똑같이 나눈다. 우리가 할 수 있는 한 똑같이 4개의 부분으로 나눈다. 나눈 4개의 데이터를 4대의 다른 컴퓨터로 보낸다. 그리고 각각의 컴퓨터는 이 결과를 계산해서 결과물을 중앙화된 서버로 보내면 이 서버가 이 결과물을 합산한다. 우리가 방금 살펴본 예에서는 대부분의 작업이 기울기 하강이었다. 즉 $\sum_{i=1}^{400} (\dots)$ 이나 $\sum_{i=1}^m (\dots)$ 을 계산하는 것이었다. 이제 4대의 컴퓨터가 각각 작업량의 $\frac{1}{4}$ 만 수행하면 되므로 여러분은 속도를 잠정적으로 4배 높일 수 있다.

07'59"

특히나 만약 네트워크 레이턴시(network latencies)가 없다면 데이터를 주고 받는 네트워크 커뮤니케이션에 어떤 비용도 들지 않고, 여러분은 속도를 잠정적으로 속도를 4배 끌어올릴 수 있다. 물론 실제로는, 네트워크 레이턴시 때문에 나중에 결과물을 합산하는 오버헤드나 다른 요소들 때문에, 여러분은 실제로는 4배 보다 좀 더 적게 속도를 높일 수 있을 것이다. 그럼에도 불구하고, 이 같은 맵리듀스 접근법은 하나의 컴퓨터를 사용하기에는 무척 많은 데이터 셋을 처리하도록 해준다.

08'32"

만약 여러분이 다른 컴퓨터들에 계산을 병렬화시켜서 속도를 높이기 위해 맵리듀스를 다른 알고리즘에 적용하려고 한다고 생각해보자. 이때 핵심적인 질문이 있다. 바로 여러분의 학습 알고리즘이 트레이닝 셋의 합계로 표현될 수 있는가 하는 것이다. 많은 학습 알고리즘이 사실 부분 트레이닝 셋의 합을 계산함으로써 표현될 수 있는 것으로 드러났다. 그리고 연산비용이 많이 들에도 불구하고 굳이 큰 데이터를 이용하는 이유는 굉장히 큰 데이터를 합해야만 알고리즘이 잘 작동하기 때문이다. 따라서 여러분의 학습 알고리즘이 트레이닝 셋의 합으로

표현될 때마다, 그리고 학습 알고리즘의 작업량이 트레이닝 셋의 합으로 표현될 때, 맵 리뷰(map review)가 여러분의 학습 알고리즘을 굉장히 좋은 데이터 셋을 통해 조정하는데 사용하기 좋은 방법이다. 예를 하나 더 보자.

09'18"

예를 들어 여러분이 진보된 최적화 알고리즘 가운데 하나를 사용하고 싶다고 하자. 그러니까 l , b , f , g , s constant gradient 등을 사용하고 싶다. 그리고 알고리즘을 이 중 로지스틱 회귀를 통해 훈련시키고 싶다. 이를 위해서는 두 개의 주요 값을 계산해야 한다. 하나는, l , b , f , g , s 와 constant gradient 같은 진보된 최적화 알고리즘을 위해서는, 비용 함수의 최적화 목표를 계산하는 추가 알고리즘(루틴)을 제공해 주어야 한다. 그리고 로지스틱 회귀에 대해서는, 이 같은 트레이닝 셋의 합계를 가지는 이런 비용함수를 기억하는가? 여러분이 10대의 머신에 걸쳐 병렬화시키고 싶다면, 트레이닝 셋을 10대의 머신에 걸쳐 나누고 각각의 머신은 이 같은 식을 트레이닝 데이터의 $\frac{1}{10}$ 에 대해 계산할 것이다.

10'10"

진보된 최적화 알고리즘에 필요한 또 다른 것은 바로 이 편미분계수를 계산하는 루틴이다. 다시 한번, 로지스틱 회귀를 위한 이 미분 계수들은 이 트레이닝 셋의 합계로 표현된다. 그리고 다시 한 번, 이전에 살펴본 예시와 비슷하게 여러분은 각각의 머신이 계산한 트레이닝 데이터의 작은 일부분을 합하게 될 것이다. 마지막으로 이 모든 값을 계산했으면, 이 결과물을 중앙서버로 보내서 이 부분 합들을 합산한다. 이는 $temp^{(i)}$ 나 $temp_j^{(i)}$ 같은 변수들을 합하는 것에 상응하는데 여기서 i 는 i 번호의 머신으로 계산된다는 뜻이다. 그래서 중앙서버가 이 것들을 더해서 전체적인 비용함수를 구하고 전체적인 부분 미분계수를 구하면 여러분은 이 진보된 최적화 알고리즘을 무사히 실행시킬 수 있다.

11'04"

그래서 좀 더 크게 보면, 다른 학습 알고리즘을 가지고 이것들을 합하는 형태로 표현하거나 아니면 트레이닝 셋에 대한 함수들의 합의 계산으로 나타냄으로써, 여러분은 맵리듀스 기술로 다른 학습 알고리즘 또한 병렬화할 수 있다. 그리고 매우 큰 트레이닝 셋에도 조정할 수 있다. 마지막으로 한 가지 덧붙이면, 지금까지 우리는 맵리듀스 알고리즘을 살펴보았다. 이를 이용하면 여러분은 여러 대의 컴퓨터를 사용해서 병렬화할 수 있는데, 컴퓨터 클러스터의 컴퓨터일 수도 있고 데이터 센터의 컴퓨터들일 수도 있다. 여러분이 가진 컴퓨터가 한 대라도 맵리듀스를 사용할 수 있다. 특히, 많은 싱글 컴퓨터들이 요즘은 프로세스 코어를 여러 개 가지고 있다. 여러분은 CPU를 여러 개 가질 수 있고, 각각의 CPU가 여러 개의 프로세스 코어를 가지고 있을 수 있다.

11'57"

만약 여러분이 가진 트레이닝 셋의 크기가 크다면 여러분이 할 수 있는 일은, 여러분에게 4개의 컴퓨팅 코어를 가진 컴퓨터가 있다면, 컴퓨터가 한 대뿐이라도 트레이닝 셋을 나눠서 싱글 박스

내의 다른 코어들로 보낼 수 있다. 이것은 데스크 탑 컴퓨터나 싱글 서버 모두에 적용될 수 있다. 그리고 맵리듀스를 사용해서 작업량을 분배한다. 각각의 코어는 합을 계산할 수 있다. 예를 들어 여러분의 트레이닝 셋의 사분의 일에 해당하는 데이터를 계산하면, 여러분은 이 부분합들을 더해서 전체 트레이닝 셋의 합을 얻는다. 맵리듀스의 장점은 다음과 같이 생각할 수 있다. 여러 대의 머신에 병렬화하기보다 이처럼 한 대의 머신에 병렬화하면 여러분은 네트워크 레이턴시에 대해 걱정할 필요가 없게 된다. 왜냐면 모든 커뮤니케이션, 즉 temp 변수들을 전송하고 수신하는 과정이 한 대의 머신에서 이루어지기 때문이다. 따라서 데이터 센서 내의 여러 대 컴퓨터를 사용할 때보다 네트워크 레이턴시의 문제가 적어진다.

12'58"

마지막으로 멀티 코어 머신으로 병렬화 할 때를 살펴보자. 여러분이 실행할 때의 디테일에 따라, 즉 여러분이 멀티 코어 머신을 가지고 있고 특정한 수치 선형 대수 라이브러리(numerical linear algebra library)들을 가지고 있다면, 이것이 자동적으로 여러분의 선형대수 작업을 머신 내멀티 코어에 걸쳐 수행하는 것으로 나타났다. 따라서 운이 좋다면 여러분은 이 수치 선형 대수 라이브러리를 사용할 수 있는데 이는 모든 단일 라이브러리에 적용되는 것은 아니다. 하지만 여러분이 이 라이브러리를 사용하고 굉장히 좋은 벡터화한 표현식을 학습 알고리즘에 적용할 수 있다면, 여러분은 표준적인 학습 알고리즘을 벡터화 방식으로 적용하고 병렬화에 대해 걱정하지 않아도 될 것이다. 그리고 수치 선형 대수 라이브러리가 이 작업의 일부를 대신 해줄 수 있다. 따라서 여러분은 맵리듀스를 실행할 필요가 없다. 하지만 다른 경우에는 내가 언급한 이 같은 맵리듀스를 사용해서, 이 맵리듀스의 공식을 사용해서 병렬화하는 것이 좋은 아이디어일 것이고 여러분의 학습 알고리즘의 속도를 높여줄 것이다.

13'58"

본 강의에서 우리는 데이터 센터의 많은 컴퓨터에 걸쳐 데이터를 나누는 병렬화 머신 러닝을 하는 맵리듀스에 대해 알아보았다. 비록 싱글 컴퓨터의 멀티 코어에도 병렬화를 적용할 수 있다는 이 아이디어들이 중요하지만 오늘날에는 맵리듀스를 실행하는 좋은 오픈소스가 있어서, 많은 사용자들이 하둡(Hadoop)이라고 불리는 오픈소스 시스템을 사용하거나 그들만의 방식을 사용하거나 아니면 다른 누군가의 오픈소스 소프트웨어를 사용한다. 여러분은 이를 이용해 학습 알고리즘을 병렬화할 수 있으며 한 대의 머신을 사용하는 것보다 훨씬 큰 규모의 데이터 셋에서 실행할 수 있다.

Week 11

No.	Title
Week 11-1	Problem Description and Pipeline

00:00

이번 시간과 다음 시간에 이어질 강의에서 기계 학습 어플리케이션 예시 또는 기계 학습 어플리케이션 역사에 대해 알아볼 건데, Photo OCR라는 어플리케이션을 중심으로 알아 볼 것이다. 이것에 대해 다루고자 하는 이유는 세 가지가 있는데, 먼저 복잡한 기계 학습 시스템이 하나로 합쳐질 수 있는지 그 예시를 보여주고자 함이다. 두 번째로, 먼저 기계 학습 파이프라인의 개념에 대해 말한 다음, 다음 단계에 무엇을 할지 결정하기 위해 자료를 어떻게 할당할지 설명하려고 한다. 이는 큰 어플리케이션에서 여러분이 스스로 작업하는 상황일 수도 있고, 복잡한 어플리케이션을 다른 개발자들과 함께 팀으로 작업하는 상황일 수도 있다. 마지막으로 Photo OCR 문제를 통해 기계 학습의 흥미로운 사실에 대해서 설명해주려고 한다. 하나는 컴퓨터 시각(영상정보를 수집한 컴퓨터가 이미지를 분석해서 유용한 정보를 생성하는 기술) 문제를 다루는데 어떻게 머신러닝을 적용할 수 있느냐는 것이고, 두 번째는 이 후 강의에서 살펴볼 인공 데이터 합성에 대한 이야기다. 그렇다면 먼저 Photo OCR 문제에 대해 살펴보도록 하자.

01:00

Photo OCR 은 Photo Optical Character Recognition(사진 광학 문자 인식)의 약자이다. 디지털 사진 분야와 휴대폰 카메라의 성장으로 우리는 이제 어느 곳에서나 시각 사진을 찍을 수 있게 되었다. 많은 개발자들의 흥미를 불러일으킨 내용은 컴퓨터가 어떻게 사진의 콘텐츠를 좀 더 잘 이해할 수 있을까이다. Photo OCR 문제는 우리가 찍은 사진 속에 있는 텍스트를 온전하게 읽어내는 것에 집중하고 있다. 이와 같은 이미지가 있을 때, 이미지 속 텍스트를 컴퓨터가 읽어낼 수 있다면, 이 사진을 나중에 다시 찾아보려고 할 때, lula b's antique mall 이라고 검색해보면, 자동으로 이 사진이 검색되어 나타날 것이고, 수백, 수천 장의 사진첩을 다 뒤지느라 시간을 낭비하지 않아도 된다.

01:51

Photo OCR 문제는 이와 같이 여러 개의 단계로 나타날 수 있다. 먼저 주어진 이미지에서 이미지를 살펴보고 텍스트의 위치를 감지한다. 위치를 감지한 후 텍스트 영역을 살펴보면서 실제의 텍스트를 읽고 해당 이미지에 표시된 텍스트가 무엇이든 간에 저장하게 된다. 물론 텍스트 영역 인지가 성공적으로 이루어지고, 텍스트를 정확하게 읽어 낸다는 가정하에서이다. 스캔된 문서에 OCR 또는 광학 문자 인식을 적용하는 것은 상대적으로 문제가 쉬운 반면에, 사진에 OCR을 적용하는 것은 오늘날 매우 어려운 머신러닝 문제로 인식되고 있다. 따라서 이를 개발해낸다면, 컴퓨터가 이미지에 담긴 내용을 더 잘 이해할 수 있게 될 뿐만 아니라 어플리케이션도 개발할 수 있게 된다. 예를 들어 시각 장애인을 돋는 어플리케이션도 개발할 수 있는데, 시각 장애인에게 그들 앞에 거리 표지판과 같은 것을 인식하고, 표지판 내용을 말해줄 수 있는 카메라를 제공해 줄수 있게 된다. 또한 자동차 네비게이션 시스템에도 적용된다면 자동차가 거리 표지판을 읽으면서, 목적지까지 운전하는데 도움을 줄 수 있다.

03:04

Photo OCR을 수행하기 위해서는 다음의 단계를 거친다. 먼저 이미지를 스캔 한 후, 텍스트 이미지가 있는 지점을 찾는다. 그렇게 Photo OCR 시스템이 찾을 수 있는 텍스트 이미지의 한 예를 보여주고 있다. 다음으로 그림처럼 직사각형으로 표시 된 텍스트 지점이 주어졌을 때 문자 분할을 할 수 있는데, 여기 "Antique Mall"이라고 써있는 텍스트 박스를 가지고 이를 분할하여, 알파벳 하나하나를 나누는 것이다. 마지막으로 알파벳 하나하나를 분할한 것을 가지고 분류기를 구현할 수 있는데, 이는 알파벳의 이미지를 보고 첫 번째 이미지는 알파벳 A, 두 번째는 N, 세 번째는 T라는 것을 알아낸다. 이러한 과정을 통해 이 문구가 LULA B'S ANTIQUE MALL 이라는 것을 알아내고, 이 이미지 속에 있는 다른 텍스트도 인식하는 것이다.

04:02

다른 photo OCR 시스템도 있는데, 더 복잡한 작업을 한다. 예를 들어, 마지막에 스펠링을 수정하기도 한다. 예를 들어, 알파벳 분할과 알파벳 분류 시스템은 이와 같은 cleaning 단어를 보고, 철자 교정 시스템은 이 단어가 'cleaning' 이라고 말해줄 것이고, 문자 분류

알고리즘은 1 대신 L이 들어가야 한다고 말해줄 것이다. 하지만 이번 시간에 우리의 목적은 마지막 단계는 생략하고, 문자 감지, 문자 분할, 문자 분류의 세 단계에 집중할 것이다. 이와 같은 시스템은 기계 학습 파이프라인이라고 부르는데, 여기에 photo OCR 파이프라인을 보여주는 이미지가 있다. 여기 이미지가 있는데, 먼저 문자 감지 시스템 단계를 거쳐, 텍스트 지점을 찾아내고 마지막으로 텍스트에서 알파벳을 하나씩 분류한다. 최종적으로 개별 알파벳을 인식하게 된다.

05:06

많은 복잡한 기계 학습 시스템에서, 이러한 파이프라인은 흔히 볼 수 있는데, 여기서 다양한 모듈을 가지고 있다. 예를 들어, 문자 감지, 알파벳 분류, 알파벳 인식 모듈이 있는데, 이들 중 어떤 것은 기계 학습 요소이고, 어떤 것은 기계학습 요소가 아닐 수도 있다. 하지만 원하는 출력을 생산하기 위해 어떠한 데이터에 대해 차례로 작업이 이루어지는 모듈 세트를 가지게 되면, 이 경우에는 photo OCR 예시에서 이미지 상에 있는 문자를 기록해 낼 것이다. 기계 학습 시스템을 디자인할 때, 가장 중요한 결정사항은 연결하고 싶은 파이프라인지 정확히 무엇인가이다. 즉, photo OCR 문제에서는 이 문제를 해결하기 위해 다른 모듈의 순서대로 어떻게 나눌까이다. 그렇게 파이프라인을 디자인하면, 파이프라인에 속하는 모듈 각각의 성능도 디자인하는 것은 알고리즘의 최종 성능에 큰 영향을 준다.

06:01

이와 같은 문제를 해결하기 위해 팀으로 작업할 때, 모듈마다 각자 맡아서 작업하는 경우가 많이 있다. 여기서 문자 감지 단계에 1-5 명 정도가, 문자 분류에 1-5명 정도가, 그리고 문자 인식에 1-5명 정도의 엔지니어가 작업할 것임을 쉽게 예측할 수 있다. 따라서 파이프라인은 엔지니어링 팀에서 멤버들끼리 업무량을 분할하는 좋은 방법을 제공한다. 물론 원한다면 한 사람이 모든 단계를 혼자서 작업할 수도 있다.

Q) 누군가 기계 학습 파이프라인을 언급할 때, 여기서 그 의미는?

1. Photo OCR 시스템
2. 문자 인식 시스템
3. 기계 학습을 활용하는 다양한 단계 / 요소를 거치는 시스템
4. 배관 어플리케이션 (하하;)

정답 3번

복잡한 기계 학습 시스템에서 파이프라인의 개념은 매우 흔히 활용되고 있다. 지금까지 Photo OCR 파이프라인이 어떻게 작동되는지를 살펴보았다. 다음 시간에는 이 파이프라인에 대해 좀 더 설명하면서, 이를 기계 학습에 주요 콘셉트를 설명하는 예시로 활용할 계획이다.

No.	Title
Week 11-2	Sliding Windows

00:00

이전 시간에는 photo OCR 파이프라인과 어떻게 구현되는지를 살펴봤다. 이는 이미지를 가지고 와서 어떠한 기계 학습 요소의 순서를 거쳐서 이미지 상의 텍스트를 읽어내는 과정이다. 이번 시간에는 파이프라인의 요소 각각이 어떻게 작동되는지를 자세히 살펴볼 것이다. 이번 강의의 대부분은 슬라이딩 윈도우 분류기라는 것에 대해 이야기할 것이다. 첫 번째 단계는 텍스트 감지였는데, 여기서 이와 같은 이미지를 살펴보고 이미지 상의 텍스트가 있는 지점을 찾아낸다. 텍스트 감지는 컴퓨터 비전에서 흔치 않은 문제다. 왜냐하면 이는 감지하려고 하는 텍스트의 길이에 따라 다르겠지만, 여기 여러분이 찾으려고 하는 이 직사각형은 다른 관점을 가질 수 있다. 따라서 이미지에서 사물을 감지하는 것에 대해 이야기하자면, 먼저 보행자 감지와 같은 간단한 예시를 살펴본 후에, 보행자 감지에서 활용된 개발을 문자 감지에 적용해보도록 하겠다.

01:05

보행자 감지에서 이와 같은 이미지를 가지고 이미지 상에 등장한 모든 보행자를 감지하려고 한다. 여기 한 명이 있고, 두 명, 세 명, 네 명, 그리고 다섯 명 마지막 여섯 명까지. 여기서 문제는 텍스트 감지보다 더 간단할 수 있는데, 보행자의 종횡비가 단순하기 때문이다. 그렇다면 이러한 직사각형으로 고정된 종횡비를 가지고 감지하고자 하는 이미지를 찾는 것이다. 여기서 종횡비는 직사각형의 높이와 너비를 일컫는다. 다른 보행자들이지만 그들의 종횡비는 모두 같다. 하지만 텍스트 감지에서는 텍스트의 길이에 따라 종횡비가 다 다르다. 보행자 감지에서도 카메라와 떨어진 거리에 따라서 직사각형의 높이가 달라질 수 있지만 그래도 종횡비는 같다.

01:57

보행자 감지 시스템에서는 다음과 같이 하면 된다. 먼저 종횡비를 82x36으로 표준화한다고 하면, 어림수로 80*40으로 할 수 있지만, 82*36으로 해보자. 그 다음에는 positive 와 negative 예시의 큰 학습 세트(Learning set)를 수집한다. 여기 82*36은 이미지는 보행자에게는 맞지만, 여기 다른 예시에는 맞지 않는다. 이 슬라이드에는 12개의 positive 예시인 $y=1$ 이 있고, 12개의 negative 예시인 $y=0$ 가 있다. 더 일반적인 보행자 감지 어플리케이션에서는 1,000에서 10,000개 또는 가능하다면 그 이상의 학습 예시를 볼 수 있다. 다음으로 신경망 또는 다른 학습 알고리즘에 82*36 이미지를 입력하여, y 를 분류하는데, 이미지에 보행자가 있는지 없는지를 분류하는 것이다.

02:55

여기서는 이미지에 보행자가 있는지 없는지를 결정하기 위해 지도 학습을 적용하게 된다. 이제 새로운 이미지가 있는데, 이와 같은 테스트 세트 이미지를 통해 이미지상의 보행자를 찾으려고 한다. 먼저 이 이미지의 직사각형 패치를 적용한다. 여기 그려져 있는데, 이는 82*36 패치이다. 이

이미지 패치를 우리가 활용하는 분류기에 돌려보면 이 이미지 패치에 보행자가 있는지 없는지를 결정하게 된다. 그리고 만약 보행자가 없다면 분류기는 $y=0$ 값을 돌려 줄 것이다. 다음으로는 이 초록색 직사각형을 옆으로 조금 옮겨서 새로운 이미지 패치에 분류기를 돌려 보행자가 있는지를 판단하게 한다. 그 다음에 이 윈도우를 오른쪽으로 조금 옮겨서 또 분류기를 돌려본다.

03:56

이렇게 직사각형을 옆으로 조금씩 이동하는 양이 매개 변수이고, 이는 매개 변수의 step size라고 하거나 stride 매개 변수라고도 한다. 이렇게 한번에 1픽셀씩 움직이면, step size 또는 stride1이라고 표현할 수 있는데, 이런 단위가 가장 효율적으로 작동된다. 따라서 한 번에 4 픽셀, 8 픽셀처럼 큰 수의 step size를 적용하는 것이 더 흔한 경우이고, 이렇게 되면 직사각형을 한번에 더 많이 움직일 수 있다. 이러한 과정을 거쳐 직사각형을 한번에 조금씩 오른쪽으로 옮기면서 분류기에 이미지 패치를 돌리다가, 결국 이 윈도우를 이미지에서 다른 장소에 돌리게 되면, 처음에는 첫 번째 줄에서 시작했고 이제 다음 줄로 가게 되었는데, 그렇게 되면 결국 분류기를 통해 모든 다양한 이미지 패치를 특정 step size 또는 stride로 돌려보게 되는 것이다.

04:56

아까 예시는 매우 작은 직사각형이었는데, 이는 특정 크기의 보행자만 감지하게 될 것이다. 이번에는 더 큰 이미지 패치를 찾아보도록 하자. 이번에는 이와 같이 더 큰 이미지 패치를 가지고, 분류기에 돌려보도록 하자. 여기서 큰 이미지 패치라는 말은 이와 같은 이미지 패치를 말하는데, 이 이미지 패치를 가지고 이를 $82*36$ 으로 사이즈를 줄이는 것이다. 따라서 큰 이미지 패치를 작은 이미지로 줄여서 작은 이미지를 가지고 분류기에 돌려서 보행지가 이 패치에 있는지를 결정하는 것이다. 또한 이렇게 더 큰 이미지를 가지고 윈도우 옆으로 끝까지 넘겨볼 수 있다. 그리고 모든 과정이 끝나면, 여러분의 알고리즘은 이미지에 보행지가 있는지를 감지하게 될 것이다. 이와 같이 분류기를 학습하는 방법을 알아봤고, 슬라이딩 윈도우 분류기 또는 슬라이딩 윈도우 감지기를 활용하여 이미지 속 보행자를 찾을 수 있는 것이다.

06:03

이제 텍스트 감지 예시로 다시 돌아와서 photo OCR 파이프라인에 있는 이 단계에 대해 이야기해보는데, 이 유닛에서 텍스트가 있는 지점을 찾는 게 우리의 목표다. 보행자 감지와 비슷하게, 텍스트가 있는 지점에 대응한 positive와 negative 예시가 분류된 학습 세트가 여기 있다. 이번에는 보행자를 감지하는 대신 텍스트를 감지해볼 것이다. 따라서 positive 예시는 텍스트가 있는 이미지 패치를 말한다. 그리고 negative 예시는 텍스트가 없는 이미지를 말한다. 이와 같이 분류기 학습을 통해, 이를 테스트 세트 이미지에 있는 새로운 이미지에 적용할 수 있다.

06:42

여기 우리가 예시로 활용한 이미지가 있다. 지난 번에 이 예시를 가지고 하나의 지정된 크기로 슬라이딩 윈도우를 돌려봤을 때, 이는 우리가 직사각형 크기 하나만을 활용한다는 것이다. 그런데

여기 이렇게 작은 슬라이딩 윈도우 분류기가 작은 이미지 패치로 이렇게 많이 있다고 하자. 그렇게 되면, 이와 같은 결과를 얻게 되는데, 여기서 하얀 지점은 이 텍스트 감지 시스템이 텍스트가 있다고 감지한 부분이고, 따라서 여기 두 수치의 측은 같게 된다. 여기 위에 지점이 있고, 여기에도 있는데, 따라서 여기 검정색 부분은 분류기가 텍스트가 없다고 찾은 부분이다. 반면에 여기 하얀색 부분이 많이 있는데 이는 분류기가 이미지에서 많은 텍스트를 찾은 부분이다.

07:35

여기 이미지 왼쪽 하단은 하얀 부분은 분류기가 텍스트가 있다고 생각한 부분이다. 그리고 회색 부분은 분류기가 출력한 확률에 대응하는데, 이와 같은 회색 그림자 부분은 텍스트가 있기는 하지만, 확실하지는 않다는 것이고, 여기 밝은 흰색 부분은 분류기가 높은 확률로 그 곳에 텍스트가 있다고 생각하는 것이다. 아직 텍스트 감지가 끝난 것이 아닌데, 왜냐하면 여기 텍스트 지점에 직사각형을 그려서 한 번 더 분류기에 출력하려고 할 건데, expansion operator라고 하는 것을 적용해 볼 것이다. 그 역할은 이미지를 가지고 여기 하얀 부분을 확대해보는 것이다. 수학적으로 이와 같은 과정을 통해 오른쪽의 이미지를 얻을 수 있고, 이와 같은 이미지를 얻기 위해, 오른쪽 이미지에 있는 모든 픽셀이 왼쪽 이미지에 있는 흰색 부분 픽셀의 범위 안에 있는지를 묻는 것이다. 따라서, 특정 픽셀 범위 내에서, 왼쪽 이미지 속 하얀색 픽셀이 5 픽셀, 10 픽셀이라고 하면, 오른쪽 이미지에 있는 픽셀도 하얀색으로 칠하게 되는 것이다. 이 결과로 왼쪽에 있는 하얀 부분을 약간 확대시켜서 조금 확대해서 근처에 하얀 픽셀이 있는지를 보고 근처 픽셀도 하얀색으로 칠하는 것이다.

09:08

마지막으로, 이제 거의 다 끝나간다. 오른쪽에 이는 이미지를 가지고 연결 요소를 찾아서 하얀 지점을 보고 그 둘레에 박스를 그려준다. 예를 들어, 여기 보면 이렇게 하얀 부분이 여기, 여기, 여기 이렇게 있는데, 여기에 간단한 발견법을 활용하여 종횡비가 다른 직사각형을 제외하려고 하는데, 텍스트 상자는 보통 높이보다는 너비가 더 넓기 때문이다. 따라서 이렇게 길고 가는 부분인, 이 것과 이것을 제외하면, 이 부분은 너무 길고 가늘기 때문에 제외하겠다. 종횡비가 텍스트 지점의 비율과 맞는 지점에 직사각형을 그려주면, 이와 같이 텍스트를 묶은 상자는 이 지점과 저 지점은 Lula B's antique mall 로고와 open 표시를 가리킨다. 이 예시에서는 사실 텍스트 하나를 감지하지 못했는데, 이는 잘 보이지 않지만 텍스트가 하나 더 있다. 여기 Lula B's라고 또 써있는데, 여기 종횡비가 틀려서 이를 제외시켰다. 이 이미지에서는 잘 감지했지만, 분류기가 텍스트 예시 하나를 텍스트로 제대로 잡아내지 못했다. 이는 사실 유리창에 써있는 텍스트였기 때문에 읽어내기가 어려웠다. 이 것이 바로 슬라이딩 윈도우를 활용한 텍스트 감지였다. 여기 텍스트가 있는 직사각형 부분을 이미지에서 잘라내서 텍스트를 확인하기 위해 파이프라인에서 이후 단계를 적용하면 된다.

Q) 20x20 픽셀 이미지 패치를 활용하는 텍스트 감지기를 구현한다고 하자. 200x200 픽셀 이미지에 분류기를 돌리는데, 슬라이딩 윈도우를 활용할 때, 4픽셀을 한번에 감지한다고 하자. (이번 문제는 한 가지 크기만 알고리즘에 적용한다고 하자.) 이미지 하나를 모두 감지하기 위해서

분류기를 몇 번 적용해야 하는가? (가장 가까운 횟수를 고르시오.)

- 1) 100번
- 2) 400번
- 3) 2,500번
- 4) 40,000번

정답 3번

10:47

이번에는 파이프라인의 두 번째 단계를 기억해보면, 이는 문자 분류 단계였는데, 이와 같은 이미지가 있을 때, 이미지에 알파벳 각각을 어떻게 분류할 수 있을까? 이번에도 positive 와 negative 예시 세트가 있는 지도 학습 알고리즘을 가지고 이미지 패치를 살펴보고, 이 이미지 패치에서 중간에 두 알파벳이 나눠지는 공간이 있는지를 알아볼 것이다. 첫 번째 positive 예시를 보면, 이미지 패치는 중간 이미지를 보여주고 있는데, 두 알파벳이 나눠진 것처럼 보인다. 두 번째 예시도, 가운데 선을 그을 수 있게 두 알파벳이 나눠져 있으므로 positive 예시다. 여기도 positive 예시인데 이렇게 두 다른 알파벳 사이에 가운데 공간이 있고 나눠져 있기 때문이다.

11:39

반면에 negative 예시의 경우, 가운데를 두 알파벳으로 나눌 수가 없는데, 이는 negative 예시를 뜻하고, 두 알파벳 사이에 중간 갭이 없음을 보여준다. 이번에는 신경망과 같은 다른 학습 알고리즘을 활용하여 분류기를 학습할 건데, 이를 통해 positive와 negative 예시를 구별하려고 한다. 이와 같은 분류기를 학습하면, 이러한 텍스트에 텍스트 감지 시스템을 적용할 수 있다. 여기 직사각형을 그려보면, 초록 직사각형의 가운데 부분이 두 알파벳 사이에 있는 것처럼 보이는지를 묻는 것이다. 만약 분류기가 아니라고 말하면, 윈도우를 슬라이드해서, 이는 1차원 슬라이딩 윈도우 분류기가 되는데, 이는 여기 다른 줄이 없기 때문에 딱 한 줄만 원쪽에서 오른쪽으로 윈도우를 슬라이드 할 것이기 때문이다. 딱 한 줄만 있다.

12:33

하지만 분류기가 이 위치에 있는 경우, 우리는 두 알파벳을 분리하거나 직사각형 가운데를 둘로 나눌 수 있는가를 물을 것이다. 그럼 분류기는 $y=1$ 를 출력할 것이고, 이렇게 둘 사이에 선을 그어 두 알파벳을 나눌 수 있게 된다. 그렇게 윈도우를 또 슬라이드 하면, 여기는 나눠지지 않고, 또 옆으로 옮기면, 이렇게 나눠지고, 이렇게 반복된다. 이렇게 오른쪽으로 분류기를 조금씩 슬라이딩하면, 나중에는 이를 positive 예시의 하나로 분류하게 될 것이다. 이렇게 오른쪽으로 윈도우를 슬라이딩하면, 분류기는 매번 우리에게 두 알파벳을 분리시킬 알맞은 자리를 이야기 해줄 것이고, 이 이미지를 두 알파벳으로 나눌 것이다. 이렇게 슬라이딩 윈도우를 통해 문자를 분류해봤다.

13:24

여기 photo OCR 파이프라인이 다시 등장했다. 이번 시간에는 텍스트 감지 단계에 대해

이야기했고, 그 다음으로 텍스트를 감지하는 슬라이딩 윈도우를 살펴봤다. 여기서 1차원 슬라이딩 윈도우를 통해 문자 분류 작업을 거쳐 이를 분류했는데, 이와 같은 텍스트 이미지에서 알파벳을 분류한 것이다. 파이프라인의 마지막 단계인 문자 확인 단계인데, 이는 이번 시간에 지도 학습에서 다뤄서 이미 알고 있는 내용이다. 여기서 표준 지도 학습을 신경망 또는 다른 것에 적용하여 이와 같은 이미지 입력을 통해 알파벳을 A부터 Z까지 26개를 분류하거나, 36개의 전산 숫자를 분류할 수도 있다. 멀티클래스 분류 문제는 알파벳이 포함된 이미지를 입력하여 이미지에 어떠한 알파벳이 있는지를 결정한다.

14:21

지금까지 photo OCR 파이프라인과 다른 요소를 가지고 photo OCR 시스템을 개발하기 위한 슬라이딩 윈도우 분류기에 대해 알아봤다. 다음에 이어질 강의는 photo OCR을 활용하여 이와 같은 어플리케이션을 개발할 때 벌어질 수 있는 재미있는 이슈에 대해 살펴보겠다.

No.	Title
Week 11-3	Getting Lots of Data and Artificial Data

00:00

고성능 기계 학습 시스템을 얻는 가장 믿을만한 방법은 low bias 학습 알고리즘을 선택하여 이를 많은 학습 세트에 학습하는 것이다. 그렇다면 그렇게 많은 학습 데이터는 어디에서 구할 수 있는가? 기계 학습에는 인공 데이터 생성 및 합성 (artificial data synthesis)이라고 하는 멋진 아이디어가 있는데, 이는 모든 문제에 적용할 수 있는 것은 아니다. 따라서 이를 특정 문제에 적용하기 위해서는 어떠한 생각과 혁신 그리고 통찰력이 필요하다. 하지만 이러한 아이디어가 여러분의 기계에 적용된다면, 문제에, 여러분의 학습 알고리즘에 많은 학습 세트를 제공할 수 있는 쉬운 방법이 될 수도 있다. 인공 데이터 합성은 두 번수로 이루어져 있는데, 첫 번째는 새로운 데이터를 Scratch를 통해 얻는 것이고 두 번째는 우리에게 이미 작은 라벨 학습 세트가 있을 때, 학습 세트를 증폭시키거나 작은 학습 세트를 큰 학습 세트로 바꾸기도 한다. 이번 시간에는 이러한 아이디어에 대해 다뤄볼 예정이다.

00:59

인공 데이터 합성 아이디어에 대해 설명하기 위해 photo OCR 파이프라인의 글자 부분을 활용하도록 하겠다. 여기 입력 이미지를 가지고 어떤 글자인지를 인식하려고 한다. 만약 우리가

큰 라벨 데이터 세트를 수집한다면 이와 같다. 이와 같은 예시에서 정사각형 비율을 선택했다. 여기서는 정사각형 이미지 패치를 활용할 것이다. 우리의 목표는 이미지 패치를 활용해 이미지 패치 중간에 있는 글자를 인식하는 것이다. 간편하게 하기 위해, 이 이미지를 칼라 이미지 대신 흑백 이미지로 하겠다. 왜냐하면 칼라 이미지는 이와 같은 문제에서는 크게 도움이 되지 않기 때문이다. 여기 이미지 패치를 보면, 알파벳 T라는 것을 알 수 있고, 이 경우에는 'S', 또 다른 경우에는 'I'를 인식할 수 있다.

01:53

여기 줄 예시 이미지에서 이보다 훨씬 큰 학습 세트는 어떻게 찾을 수 있을까? 오늘날 컴퓨터는 많은 폰트 라이브러리를 가지고 있는데, 워드프로세서를 활용하면, 어떤 것을 활용하는지에 다르겠지만, 이와 같이 엄청 많은 폰트가 이미 내장되어있다. 또한 실제로 다른 웹사이트에 가도 인터넷 상에서 다운 받을 수 있는 다양한 무료 폰트 라이브러리가 수백, 수천 개가 넘는다. 따라서 더 많은 학습 세트가 필요하다면, 다른 폰트로 된 글자를 가져와서 이를 임의로 배경에 적용하는 것이다. 여기서 C를 가져와서 아무데나 붙여주면 된다. 이제 글자 C의 이미지 학습 예시가 생성된 것이다.

02:45

이와 같은 작업 이후에, 이제 데이터를 진짜처럼 보이게 합성을 해준다. 약간의 작업을 거쳐 이와 같은 합성 학습 세트를 얻게 된다. 오른쪽에 보이는 모든 이미지는 실제로 합성된 이미지다. 폰트를 가져가서, 웹사이트에서 임의의 폰트를 다운 받아서 이미지에 글자 하나를 붙여 넣기하거나 폰트에서 여러 글자를 가져와서 임의의 배경에 붙여 넣기 하는 것이다. 그리고 앱 파인더를 약간 흐리게 처리하면, 약간 왜곡하면 되는데, 여기서 앱 파인더는 공유하고, 측정하고, 회전하는 작업을 말한다. 그렇게 되면 다음과 같은 합성 학습 세트를 얻을 수 있게 된다. 이렇게 합성 데이터를 진짜처럼 보이게 하는 작업을 너무 대충하면 나중에 잘 활용하지 못할 수 있다. 하지만 이와 같이 합성 데이터를 보면 실제 데이터와 거의 흡사한 것을 알 수 있다. 따라서 이러한 합성 데이터를 활용하게 되면, 인공 학습 합성을 위한 학습 예시를 무제한으로 얻을 수 있게 되는 것이다. 만약 이와 같은 자료의 합성 데이터를 활용할 때, 글자 인식 문제에서 지도 학습 알고리즘을 생성할 라벨 데이터를 무제한으로 얻을 수 있게 되는 것이다. 이와 같이 인공 데이터 합성 예시를 살펴봤는데, 이를 통해 새로운 데이터를 Scratch에서 완전히 새로운 이미지를 생성할 수 있게 되었다.

04:14

인공 데이터 합성에 또 다른 주요 접근법은 현재 가지고 있는 미가공 이미지 데이터를 가지고, 가공된 추가 데이터를 생성하는 것이다. 다시 말해 학습 세트를 증폭시키기 위한 것이다. 따라서 여기 합성 이미지가 아닌 실제 이미지가 있는데, 여기에 그리드 선을 그려놨다. 원래는 이 그리드 선이 없었다. 다음으로 이 알파벳 이미지를 가지고 인공 뒤틀림, 인공 왜곡을 적용하여 다음과 같은 16개의 새 예시를 얻었다. 이와 같은 방법으로 이 작은 라벨 학습세트를 가지고 학습 세트를 증폭시켜서 이와 같이 다양한 예시를 얻게 되었다. 다시 말해, 이를 적용하기 위해서는

왜곡의 적용할 적당한 세트는 무엇인지 이와 같은 방법으로 학습 세트를 증폭시킬 것인지에 대한 생각과 통찰력이 필요하다. 글자 인식과 같은 특정 예시에서는 이와 같은 왜곡 방법을 소개하는 것은 당연한 선택이지만, 다른 기계 학습 어플리케이션에서는 다른 왜곡이 더 잘 맞을 수도 있다.

05:24

이번에는 음성 인식의 완전히 다른 영역 예시를 보여주겠다. 음성 인식에서 오디오 클립이 있는데, 이 음성에서 말하는 단어를 인식하는 것을 학습하려고 한다. 여기 분류된 학습 세트를 하나 살펴보자. 여기 분류된 학습 예시가 있는데, 누군가 특정 단어 몇 개를 말한다. 먼저 오디오 클립을 들어보자. 0-1-2-3-4-5. 이와 같이 누군가 0부터 5까지 세는 음성인데, 이 학습 알고리즘을 적용하여 단어를 인식하도록 해보자. 그렇다면 데이터 세트를 어떻게 증가시킬 수 있을까? 한 가지 방법은 오디오 왜곡을 통해 데이터 세트에 예시를 추가하는 것이다. 여기에 배경 소리를 추가하여 통화 상태가 안 좋은 상황을 연출하겠다. 이와 같은 빠하는 소리가 날 때, 이는 말하는 이의 잘못이 아니라 오디오 트랙에 포함되어있는 것이다. 이 때 여기서 이를 재생해보겠다. 0-1-2-3-4-5.

06:25

따라서 이와 같은 오디오 클립을 듣고 소리를 인식하게 된다면, 활용하기 유용한 학습 예시가 될 것이다. 여기 또 다른 소음 예시가 있다. 0-1-2-3-4-5. 자동차가 빠르게 지나가고 주변에 사람들이 걸어 다니는 상황이다. 하나 더 있는데, 이처럼 명확하게 012345가 들리는 원본 오디오 클립을 가지고 자동으로 여기 추가 학습 예시를 합성하여 하나의 학습 예시를 가지고 4개의 다른 학습 예시로 증가시킬 수 있다. 여기 마지막 예시를 들어보면, 0-1-2-3-4-5 이와 같이 분류된 예시 하나 만을 수집하는 노력을 통해, 왜곡된 사운드를 합성하여 새로운 배경 소리를 입력하면, 이와 같이 하나의 예시로 다양한 예시를 만들 수 있게 된다. 많은 노력 필요 없이 자동으로 이와 같은 원본 음성에 배경음을 추가해주면 된다.

07:30

이와 같은 왜곡을 더한 데이터 합성시 한가지 당부를 하자면, 스스로 왜곡 작업할 때, 테스트 세트에서도 볼 수 있는 소음 또는 왜곡을 대표할 수 있는 자료를 활용해야 한다는 것이다. 글자 인식 예시에 이와 같은 뒤틀림 예시는 적당하다고 보는데, 왜냐하면 테스트 세트에서 볼 수 있는 이미지이기 때문이다. 여기 오른쪽 위에 있는 그림도 우리가 볼 수 있는 이미지라고 할 수 있다. 음성 예시에서는 만약 통화 상태가 좋지 않거나, 배경 소음이 있다고 할지라도 음성을 인식하려고 한다. 따라서 음성에서 합성된 이미지는 우리가 분류하여 정확하게 인식하고자 하는 이러한 예시를 대표할 수 있는 예시가 필요하다.

08:18

반면에, 실제로 데이터에 임의의, 무의미한 소음이 더해진다면 이는 그다지 도움이 되지 않는다. 이게 잘 보일지 모르겠지만, 여기 이미지를 가지고 4개의 이미지로 복사해 각각 임의의 가우스 잡음을 추가하였다. 각각 픽셀은 픽셀 밝기이며, 임의의 가우스 잡음을 이미지마다 추가했다.

이는 완전히 무의미한 잡음이다. 따라서 이와 같은 픽셀 방식 잡음을 여러분의 테스트 세트에서 보기로 원하지 않는 이상, 이와 같은 임의의 무의미한 잡음은 쓸모가 없다. 하지만 인공 데이터 합성 과정은 예술이라고도 말할 수 있는데 이를 시도해보고 작동하는지를 살펴봐야 한다. 만약 이와 같은 왜곡을 추가하기로 결정했다면, 어떠한 의미 있는 왜곡을 추가하여 여러분의 테스트 세트에서 볼 수 있는 비슷한 종류의 대표적인 이미지의 추가 학습 예시를 생성할 수 있을지 생각해봐야 한다.

Q) 예시 m 개의 선형 회귀 모델을 다음과 같이 최소화하여 학습하려고 한다. 모든 예시를 두 개로 복제한다고 할 때, 원래 $(x(i), y(i))$ 예시가 있었는데, 이제 $2m$ 개의 예시가 있다. 이게 도움이 되는가?

1. 그렇다, 학습 세트 사이즈의 증가는 변수를 줄여준다
2. 그렇다, 많은 feature를 활용한다면 ('low bias' 학습 알고리즘 일 경우)
3. 아니다, 다른 매개 변수 θ 를 얻을 텐데, 원래 학습 세트에서 학습한 것들보다 성능이 더 나을게 없다.
4. 아니다, 결국 데이터를 복제하기 전과 같은 매개 변수 θ 를 얻게 된다.

(정답 4번)

09:18

이 강의를 마무리하기 전에, 인공 데이터 합성을 통해 많은 데이터를 얻는 방법에 대해 이야기를 하려고 한다. 항상 많은 노력을 기울이기 전에, 어떻게 인공 학습 예시를 생성할지를 생각해내야 하는데, low biased 분류기를 가지고 있는지, 그리고 많은 학습 데이터를 가지고 있으면 많은 도움이 될 것이다. 가장 기본적인 방법은 학습 곡선을 그려서 낮거나 높은 변수 분류기를 가지고 있으면 된다. 만약 낮은 변수 분류기가 없을 경우, 시도해 볼만한 또 다른 방법은 분류기의 feature의 수를 점차 증가시키는 것인데, 신경망에서 low bias 분류기를 얻을 때까지 은닉 계층의 수를 증가 시키는데, 그 때 많은 인공 학습 세트를 생성하기 위해 노력해야 한다. 그러니까 피해야 할 것은 일주일 내내 또는 몇 달 동안을 인공 학습 데이터 세트를 생성하는데 시간을 낭비하는 것이다. 그렇게 시간을 다 보내고 나서야 그렇게 많은 데이터를 가지고도 여러분의 학습 알고리즘의 성능이 그다지 향상되지 않았다는 것을 알 수 있게 된다. 따라서, 많은 테스팅에서 조언하는 것은 많은 학습 세트를 얻기 위해 많은 시간을 투자하는 노력하기 전에 많은 학습 세트를 활용할 수 있다는 것이다.

10:31

두 번째로, 기계 학습 문제를 다룰 때, 함께 일하는 팀 동료들 또는 학생들에게 자주 묻는 질문인데, 지금 가지고 있는 데이터의 10배를 얻기 위해서는 얼마나 많은 작업이 필요할까이다. 새로운 기계 학습 어플리케이션을 발견하게 되면, 팀 동료들과 앉아서 이러한 똑 같은 질문을 하는데, 계속되는 질문을 통해 이에 대한 답이 이와 같다는 것에 놀라웠다. 예를 들어, 그렇게

어렵지 않다. 길어봐야 며칠이면 현재 가지고 있는 기계 학습 어플리케이션 데이터의 10배를 얻는 것이 가능하다는 것이다. 또한 10배 많은 데이터는 여러분의 알고리즘이 더 잘 구현될 수 있게 해준다. 따라서 팀으로 기계 학습 어플리케이션을 작업하게 된다면, 여러분 스스로 또는 멤버들에게 물어볼 수 있는 매우 좋은 질문인 것이다. 이와 같은 몇 분의 브레인스토밍을 통해 여러분의 팀이 말 그대로 10배의 데이터를 생성할 수 있다면, 그 팀에서 여러분은 영웅이 될 것이다. 10배의 데이터를 가지고 학습을 하면 훨씬 좋은 성능을 얻을 수 있게 될 것이기 때문이다.

11:39

이와 같이 이번 시간에는 인공 데이터 합성의 방법에 대해 자세히 살펴봤다. 크게 두 가지 방법이 있는데 이는 임의의 폰트를 활용하여 Scratch에서 데이터를 생성하는 방법이 있었고, 두 번째는 이미 가지고 있는 예시를 활용해 이를 왜곡시켜 학습세트를 증가하는 방법이 있었다. 또 다른 방법으로는 데이터를 스스로 수집해서 분류하는 것이다. 내가 자주 활용하는 유용한 연산인데, 특정한 수의 예시를 얻기 위해 몇 시간 또는 몇 분이 걸리는 가를 측정하는 것이다. 그래서 앉아서 생각을 해보는데, 1개의 예시를 분류하는데 10초가 걸린다고 하면, 이 어플리케이션에서는 1000개의 분류된 예시가 있는데, 이와 같은 데이터의 10배를 얻으려고 하면, 이는 $m = 10,000$ 이 된다.

12:39

많은 데이터를 얻는 두 번째 방법은 데이터를 수집하여 스스로 분류하는 것이다. 나는 보통 가만히 앉아서 현재의 데이터의 10배를 얻기 위해서 나 또는 누군가가 데이터를 수집하고 분류하기 까지, 얼마나 많은 시간이, 며칠이 걸릴지를 계산하는 것이다. 예를 들어, 기계 학습 어플리케이션에서 1,000 예시가 있다고 하자. $m = 1000$ 이다. 이제는 앉아서 하나의 예시를 수집하여 분류하는 데 얼마나 시간이 걸릴지를 묻는 것이다. 때로는 10초 정도 걸린다고 하면, 이의 10배는 얼마나 걸릴지를 계산하는 것이다. 하나의 학습 예시를 얻는데 10초가 걸린다면, 10배의 데이터를 얻고자 할 때, 데이터 10,000개를 얻고자 하는 것인데, 계산해보면, 직접 10,000의 예시를 분류하기 위해서는, 한 개의 예시를 분류할 때, 10초가 걸렸다.

13:46

이를 계산하면 이는 며칠이면 해결 할 수 있는 그런 간단한 작업이라는 것에 많은 사람들이 놀라는 것을 발견한다. 팀으로 작업 할 때도, 이렇게 많은 데이터를 얻는 게 생각보다 간단한 작업이라는 것에 놀랐고, 이를 통해 학습 어플리케이션의 성능을 향상시킬 수 있고, 이를 해낸다면 무엇을 연구하고 있든지 간에 팀에서 여러분은 영웅이 될 것이다. 이 과정을 통해 훨씬 좋은 성능을 얻게 되었기 때문이다.

14:16

세 번째이자 마지막으로 많은 데이터를 얻는 좋은 방법 중 하나는 크라우드소싱(crowdsourcing)을 활용하는 것이다. 오늘날 인력을 구하는 웹사이트나 서비스는

많은 학습 세트를 쉽게 분류하고 있다. 이러한 크라우드소싱 또는 크라우드 소스 데이터 분류(crowd sourced data labeling) 개념은 모든 학문에서 그런 것처럼 문제가 있는데 labeler 신뢰도와 관련이 있다. 전 세계적으로 수십만의 labeler가 있는데, 데이터를 분류하는 것을 쉽게 도와준다. 아까 말했듯이 이와 같은 방법이 있다는 것이다. 또한 Amazon Mechanical Turk 시스템이 크라우드소싱 옵션 중 가장 인기 있는 시스템일 것이다. 이와 같은 시스템은 구현하기 작업이 좀 필요하지만, 고품질의 분류를 원한다면, 고려해볼 만한 시스템 중 하나이다. 만약 웹사이트를 통해 사람을 고용하고자 할 경우, 다양한 데이터를 제공해 줄 것이다.

Q) 여러분은 최근 지난 12개월 동안 1,000개의 예시를 활용해 기계 학습 어플리케이션을 개발하고 있는 팀에 합류하게 되었다. 수동으로 예시를 수집하고 분류해야 한다고 할 때, 하나의 추가 예시를 얻는 데 10초가 걸린다고 하자. 매일 8시간 작업한다고 할 때, 10,000 예시를 얻기 위해 며칠이 걸릴까? (가장 가까운 값을 고르시오)

- 1) 1일
 - 2) 3.5일
 - 3) 28일
 - 4) 200일
- 정답 2번

15:25

이번 시간에는 인공 데이터 합성의 개념을 통해 Scratch에서 새로운 데이터를 생성하거나 임의의 폰트를 활용한 예시, 가지고 있는 학습 세트를 증가시키기 위해 이를 왜곡하여 추가 예시를 생성하는 방법에 대해 알아봤다. 마지막으로 이번 시간에서 여러분이 기억해야 할 내용에 대해 다시 말해주자면, 기계 학습 문제에서 두 가지를 하는 것이 도움이 된다. 첫 번째는 학습 곡선을 활용한 sanity check 인데, 이는 데이터가 많으면 도움이 될 것이다. 두 번째는 가정을 해보는 것인데, 앉아서 스스로에게 진지하게 질문을 해보는 것이다. 지금 가지고 있는 데이터의 10배를 얻기 위해서는 작업이 얼마나 필요할까? 항상은 아니지만 가끔 생각보다 며칠 또는 몇 주 안에 결과를 얻을 수 있다는 사실에 놀라게 될 것이다. 이를 통해 학습 알고리즘에 성능도 향상시킬 수 있기 때문이다.

No.	Title
Week 11-4	Ceiling Analysis: What Part of the Pipeline to Work on Next

00:00

지난 여러 강의에서, 기계 학습 시스템을 개발할 때 개발자로서 가장 중요한 자원은 바로 시간이라고 강조했는데, 이는 다음 단계에 어떤 작업을 할지를 선택하는 문제다. 또는 기계 학습

문제에 대해 팀으로 작업할 경우에도 엔지니어나 개발자들이 작업하는데 가장 중요한 자원은 시간이다. 따라서 여러분 또는 여러분의 동료가 어떤 요소 때문에 많은 시간을 낭비하는 것을 피해야 한다. 몇 주 또는 몇 달을 보내고 나서야 최종 시스템의 성능에는 그다지 큰 변화를 주지 않는다는 것을 알게 될 것이다. 이번 시간에는 ceiling 분석에 대해 알아보겠다. 여러분이 팀으로 시스템에서 파이프라인 기계를 작업할 때, 어떤 파이프라인을 활용하는 것이 시간을 가장 효율적으로 활용할 수 있는지에 대해 매우 강한 신호 또는 안내를 해 줄 것이다.

00:59

ceiling 분석에 대해 이야기하기 위해, 계속해서 photo OCR 파이프라인 예시를 활용하겠다. 여기 박스마다 텍스트 감지, 글자 분류, 글자 인식이 써있는데, 박스 각각마다 소규모의 엔지니어링 팀이 작업을 하고 있을 수 있다. 또는 여러분 혼자서 모든 시스템을 개발할 수도 있다. 여기서 문제는 어디서 자원을 할당해야 하는 것이다. 이러한 요소 중에서 하나, 둘 또는 세 요소 모두에 시간을 투자해서 성능을 향상시켜야 할지를 결정하는 것이다. 이것이 바로 ceiling 분석의 개념이다. 다른 기계 학습 시스템을 개발하는 과정에서도 시스템을 개발하는데 있어서 무엇을 할지를 결정하는 것은 이러한 학습 시스템의 하나의 룰번호 측정 metric에 매우 도움이 된다.

02:17

먼저 글자 레벨 정확도를 선택해보자. 이와 같은 테스트 세트 이미지가 있을 때, 테스트 이미지에 있는 알파벳 또는 글자의 부분을 가지고 이를 정확하게 인식할 수 있는가? 또는 원한다면 하나의 룰번호 측정을 선택해도 좋다. 하지만 어떠한 측정 방법을 선택한다고 해도, 전반적인 시스템은 현재 72% 정도의 정확도를 보여준다. 다시 말해, 테스트 세트 이미지가 있는데, 테스트 세트 이미지 각각에서 텍스트 감지, 글자 분류, 그리고 글자 인식 과정을 거치는 것이다. 그리고 이 테스트 세트에서 전체 시스템의 전반적인 정확도는 어떠한 metric를 선택한다고 해도 72%가 나온다는 것이다. 이제 ceiling 분석에 깔려있는 개념에 대해 알아보겠는데, 기계 학습 파이프라인에서 첫 번째 모듈인 텍스트 감지를 가지고 설명해보도록 하겠다.

03:08

이제 테스트 세트를 조금 돌아가는 방식으로 활용해보겠다. 모든 테스트 예시는 정확한 텍스트 감지 출력을 제공할 것인데, 이는 테스트 세트에서 수동으로 알고리즘에 테스트 예시에 있는 텍스트가 어떤 것인지 말해주는 것이다. 즉, 이미지에서 텍스트 감지 하는데 있어서 100% 정확도를 보이는 텍스트 감지 시스템으로 가정해보는 것이다. 이는 매우 간단하다. 학습 알고리즘이 이미지 상 텍스트를 감지하라고 하는 대신, 이미지를 보고 수동으로 테스트 세트 이미지 상에 텍스트의 위치를 표시해주는 것이다. 그렇게 되면 이는 테스트 세트에 텍스트가 어디에 있는지 정확한 검증 자료 분류가 된다. 그런 다음 이러한 검증 자료를 다음 단계 파이프라인인 글자 분류 파이프라인에 적용해 보는 것이다. 다시 말해, 여기에 체크 표시를 하는 것은, 이 테스트 세트에 가서 파이프라인의 텍스트 감지 부분에 옳은 답을, 옳은 분류를 입력해준다는 것이다. 그렇게 되면 이 테스트 세트에 완벽한 텍스트 감지 시스템을 가지게 되는

것이다.

04:23

이제 이 데이터를 가지고 나머지 파이프라인 글자 분류와 글자 인식에 적용시켜보자. 그리고 아까와 같이 같은 측정 metric를 활용하여 전체 시스템의 전반적인 정확도를 측정해보도록 하자. 이와 같이 텍스트 감지가 100% 완벽하다면 성능은 향상될 것이다. 이 예시에서는 89%까지 향상되었다. 또 성능은 계속 향상될 것이다. 이제 다음 파이프라인으로 넘어가면, 글자 분류 단계이다. 이번에도 테스트 세트를 보고 정확한 텍스트 감지 출력을 입력하여 정확한 글자 분류 출력이 나오도록 해보겠다. 먼저 테스트 세트에 가서 수동으로 텍스트를 글자 하나하나로 정확히 분류하고 이것이 얼마나 도움을 주는지 보면 된다. 전체 시스템의 정확도가 90%까지 올라갔다고 하자. 따라서 모든 시스템의 정확도는, 글자 인식 시스템의 최종 출력이 무엇이든지 간에, 전체 파이프라인의 최종 출력이 무엇이든지 간에, 이 정확도를 측정하게 된다.

05:22

마지막으로 글자 인식 시스템을 생성하여 여기에도 정확한 분류를 제공할 건데, 이렇게 하고 나면, 100%의 정확도를 얻을 수 있게 된다. 이와 같은 분석 작업의 장점은 이와 같은 요소의 상승 잠재력 향상을 확인할 수 있다는 것이다. 따라서 완벽한 텍스트 감지를 얻게 되면, 이 성능은 72%에서 89%로 증가하게 된다. 17%가 증가하게 된 것이다. 이는 현재 시스템을 가지고 텍스트 감지를 개선시키기 위해 시간을 투자하면, 시스템의 성능을 17% 가량 향상시킬 수 있다는 것이다. 이는 시간을 투자할 가치가 있는 작업이다.

06:02

반면에, 텍스트 감지에서부터 거의 완벽한 글자 분류로 넘어갔을 때, 성능은 오직 1%만 증가했는데, 슬픈 결과다. 따라서 글자 분류에 많은 시간을 투자한다고 해도 상승 잠재력은 매우 작다는 것인데, 따라서 글자 분류에는 많은 엔지니어들이 작업하기를 원하지 않을 것이다. 이와 같은 분석은 완벽한 글자 분류를 가지고도 성능은 경우 1%만 상승할 수 있다는 것을 보여준다. 이는 이러한 요소 각각에서 작업을 한다면, 시스템의 성능을 얼마나 향상 시킬 수 있는 ceiling(최대 한계)이 무엇인지, 상한이 무엇인지를 예측한다.

06:43

마지막으로, 글자에서부터 더 나은 글자 인식 형태를 얻게 되면 정확도는 10% 가까이 더 증가한다. 다시 말해, 이는 10% 향상이라고 할 수 있는데, 이는 어느 정도의 가치가 있을까? 이는 파이프라인의 마지막 단계에 더 많은 시간을 투자하는 것이, 시스템의 성능을 향상시킬 수 있는 방법이라는 것을 말해준다. 다르게 생각해보면, 이와 같은 분석을 통해 각각 요소를 향상시키는 상승 잠재력은 어느 정도 되는가 또는 이러한 요소 중 하나가 완벽하게 구현이 되면 전반적인 성능이 얼마나 향상될 수 있을까? 이는 시스템의 성능의 상한을 두는 것이다. 따라서 ceiling 분석은 매우 중요한 개념이다. 이 개념에 대해 다른 예시로 설명하고자 하는데 이는 좀 더 복잡하다.

07:31

이미지에서 얼굴 인식을 하려고 한다. 사진을 보고 사진에 있는 사람이 여러분의 친구인지 등 이미지 속에서 얼굴을 인식하는 것이다. 이는 약간 예술과 관련된 예시인데, 실제로 이와 같은 방법으로 얼굴 인식이 이루어지지는 않는다. 하지만 예시를 통해 그렇게 설정을 하여 ceiling 분석 과정이 어떤 건지에 대해 살펴보고자 한다. 여기 카메라 이미지가 있는데, 다음과 같이 파이프라인을 디자인한다고 하자. 첫 단계는 이미지를 사전 처리하는 것이다. 오른쪽 상단에 있는 이미지를 가지고 배경을 먼저 제거해보자. 먼저 이미지를 분석하고 배경을 없앤다. 다음으로 사람의 얼굴을 감지하려고 하는데, 이를 학습을 통해서 이루어진다. 따라서 여기서 슬라이딩 윈도우 분류기를 통해 사람의 얼굴에 박스를 그려준다. 얼굴을 감지하고 나면, 사람을 인식하고자 할 때, 눈이 중요한 열쇠가 된다는 것을 발견할 수 있다. 친구의 얼굴을 인식할 때, 눈이 가장 중요한 단서가 된다는 것이다.

08:38

이번에는 사람의 눈을 감지하기 위해 분류기를 적용해보자. 눈을 분류하고 나면 이는 사람을 인식하는데 유용한 특성을 제공하는 것이다. 그 다음은 얼굴의 다른 부분의 신체적 특징을 찾아보자. 코를 분류하고, 입을 분류할 수 있겠다. 그럼 눈, 코, 입을 찾아서 이를 선형 회귀 분류기에 활용할 유용한 특성을 제공하게 되는 것이다. 그럼 이 분류기의 역할은 우리에게 전반적인 분류를 제공하여 이 사람이 누구인지 분류하여 찾을 수 있게 해준다. 이와 같이 복잡한 파이프라인은 여러분이 실제로 사람을 인식하려고 할 때 활용할 것보다 좀 더 복잡할 것이다. 하지만 이는 ceiling 분석을 설명하기 위한 유용한 예시였다.

09:22

그렇다면, 이와 같은 파이프라인에서 ceiling 분석은 어떻게 진행될까? 먼저 이와 같은 부분을 한번에 하나씩 거치는 것이다. 전체적인 시스템의 정확도가 85%라고 하자. 먼저 테스트 세트에 가서 수동으로 전경과 배경의 실측 정보를 제공한다. 이렇게 수동으로 테스트 세트에 접근하여 포토샵 또는 다른 프로그램을 가지고 배경이 어디 있는지를 확인해서 수동으로 실제 배경을 삭제해서 정확도가 어떻게 바뀌는지 살펴본다. 이번 예시에서는 정확도가 0.1% 상승했다. 이는 거의 완벽한 배경 분류를 가지고 있다고 해도 완벽하게 배경을 지웠다고 해도 시스템의 성능이 그렇게 많이 상승되지 않는다는 것을 분명히 보여준다. 따라서 미리 분석하고, 배경을 삭제하는데 그렇게 많은 노력을 기울일 필요가 없다는 것이다.

10:08

이번에는 테스트 세트로 넘어가서 정확한 얼굴 인식 이미지를 입력하고 눈코입 분류를 어떠한 순서를 정해서 순서대로 단계를 진행한다. 먼저 눈의 정확한 위치를 제공하고, 그 다음 코의 위치, 마지막으로 입의 위치를 제공한다. 그러면 마지막으로 모든 것을 정확하게 제공하면 100% 정확도를 얻게 된다. 따라서 이와 같이 시스템을 거쳐 더 많은 요소를 추가하게 되면, 테스트 세트의 정확한 분류는 전체 시스템의 성능을 향상되고 각 단계별로 성능이 얼마나 상승하게

되었는지를 확인할 수 있다. 따라서 완벽한 얼굴 인식에서부터 시스템의 전체 성능이 5.9% 정도 크게 상승했다. 이는 얼굴 인식에 더 작업할 가치가 있다는 것이다. 여기서는 4% 증가했고, 여기는 1% 증가했고, 여기는 1% 여기는 3%가 증가했다.

11:01

따라서 여기서 시간을 투자할 가치가 있는 요소는 완벽한 얼굴 시스템을 추가했을 때 성능이 5.9%나 상승했고, 눈을 분류했을 때는 4%, 마지막으로 로지스틱 분류기는 3% 추가로 증가했다. 이와 같이 작업을 집중해야 할 요소에 대해 말해주고 있다.

11:25

이번에는 실제 경고성 이야기를 하나해주겠다. 여기 사전 탐지(배경 제거)를 추가한 이유는 실제 사례를 알고 있기 때문인데, 연구팀이 일년 반, 그러니까 18개월 동안 더 나은 배경 제거를 하기 위해서 작업한 것이다. 하지만 그 이유에 대한 내용은 명료하지 않은데, 어쨌든 컴퓨터 비전 어플리케이션이 있었는데, 두 명의 엔지니어가 1년 반을 더 나은 배경 제거를 위해 작업을 했다는 것이다. 이는 매우 복잡한 알고리즘을 연구하는 것이었고, 결국 연구 논문을 하나 발간하기까지 했다. 하지만 그러한 작업이 끝난 후에, 그들이 작업했던 실제 어플리케이션의 전체적인 성능에는 큰 변화가 없었다는 것을 발견했다. 그들 중 누구라도 ceiling 분석을 해봤더라면, 일찍 알아차릴 수 있었을 텐데 말이다. 그 중 한 명이 내게 나중에 말해주기를, 이런 종류의 분석을 미리 했었더라면, 이것이 18개월이 걸리는 작업이라는 것을 미리 알 수 있었을 것이고, 그랬더라면 배경 제거에만 18개월을 작업하는 대신에 다른 요소에 집중할 수 있었을 것이다.

Q) 파이프라인 기계 학습 시스템에서 ceiling 분석을 하려고 한다. 어떠한 요소에 실제 측정한 분류를 추가하려고 할 때, 전반적인 시스템의 성능이 매우 조금 증가하는 것을 발견했다. 이것이 의미하는 것은?(복수 선택 가능)

1. 해당 요소를 위해 더 많은 데이터를 수집해야 한다.
2. 해당 시스템 요소를 향상시키기 위해서 엔지니어링 자원을 투자할 의미가 없다.
3. 만약 그 요소가 기울기 하강을 학습하는 분류기 학습이라면, 10배로 만들기 위해 기울기 하강을 활용하여 더 나은 분류기 매개변수를 모을 필요가 없다.
4. 그 요소에 더 많은 feature를 선택하여 bias를 줄일 수 있게 돋고, 그 요소의 feature의 수를 줄이는 (변수를 줄여) 것은 도움이 되지 않는다.

정답 2,3번

12:33

요약하자면, 파이프라인은 복잡한 기계 학습 어플리케이션에서 광범위하게 이루어지고 있다. 여러분이 큰 기계 학습 어플리케이션을 작업하고 있다면, 개발자로서 여러분의 시간은 매우

귀중하다. 그러니 영향력이 없는 것에 매달리느라 시간을 낭비하지 말자. 이번 시간에는 ceiling 분석의 개념에 대해서 이야기해봤는데, 이는 집중해야 할 요소가 무엇인지를 확인할 수 있는 매우 좋은 툴이다. 큰 변화는 최종 시스템의 전체 성능에 크게 영향을 줄 것이다. 따라서 몇 년 동안 기계 학습을 작업하면서 어떠한 요소에 시간을 더 투자할지를 내 직감대로 결정하지 않게 되었다. 기계 학습을 오랫동안 다뤄왔지만 아직도 기계 학습 문제를 보면 직감을 따라 자주 이 요소를 더 파헤쳐보자 하고 시간을 다 보내버리는 경우가 있다. 하지만 시간이 지나면서 내 직감을 믿지 않게 되었다. 대신에 기계 학습 문제에서 ceiling 분석과 같은 단계를 거쳐서 이는 어느 단계에 시간을 투자해야 성능을 향상시킬 수 있을지를 알아낼 수 있는 훨씬 더 좋은, 믿을만한 방법이다. 이를 통해 전체 시스템의 최종 성능에 큰 영향을 주게 된다는 것을 확신할 수 있게 될 것이다.

No.	Title
Week 11-5	Summary and Thank You

00:00

기계 학습 강좌의 마지막 강의에 오신 걸 환영합니다. 지금까지 다양한 강의를 함께 했다. 이번 시간에는 이번 강좌의 주요 논제에 대해 요약해보고, 이 강의를 마무리 하기 전에 몇 마디 전하도록 하겠다. 지금까지 학습한 내용을 살펴보자. 이번 강좌에서는 지도 학습 알고리즘에 대해 이야기를 많이 했는데, 선형 회귀, 로지스틱 회귀, 신경망, svm 등이 있었다. 문제에서는 분류된 데이터와 분류된 예시 $(x(i), y(i))$ 가 있었다. 또한 자율 학습 분야에서는 k-평균 군집화(k-means clustering), 차원 축소를 통한 주성분 분석법, 분류되지 않은 데이터 $x(i)$ 만 있을 때, 이상 탐지 알고리즘이 있었다. 이상 탐지는 알고리즘을 평가하기 위해 분류된 데이터 또한 활용할 수 있다. 또한 특수 어플리케이션과 주제에 대해 이야기해봤는데, 추천 시스템과 병렬식 또는 빠른 사용 시스템과 같은 큰 규모 기계 학습 시스템이 있었고, 컴퓨터 비전을 위한 슬라이딩 윈도우 물체 분류와 같은 특별한 어플리케이션도 다뤄봤다.

01:07

마지막으로 기계 학습 시스템을 개발하는데 많은 다양한 관점에서 이야기해봤다. 이는 기계 학습 알고리즘이 작동하고, 작동하지 않게 하는지를 알아봤다. 먼저 bias와 변수에 대해 이야기했고, 변수 문제를 해결에 도움을 주는 규칙화에 대해 이야기했다. 다음 단계에 무엇을 할지를

선택하는 것에 살펴봤었다. 기계 학습 시스템을 개발 할 때, 어느 것에 우선 순위를 둬서 시간을 활용해야 할지에 대한 것이다. 학습 알고리즘 평가에 대해 이야기하면서 정밀성, 재호출성, f1 점수와 같은 평가법을 다뤄봤고, 학습/교차검증/테스트 세트와 같은 평가의 실용적 면모도 살펴봤다. 또한 디버깅 학습 알고리즘을 통해 학습 알고리즘이 확실히 구현되도록 했다. 학습 곡선과 같은 진단 프로그램도 살펴봤고, 오류 분석 또는 ceiling 분석도 살펴봤다. 이와 같은 다양한 툴은 기계 학습 시스템을 개발하는데 있어서 다음 단계에 무엇을 할지를 결정하는 것을 도와 소중한 시간을 아낄 수 있도록 도와준다. 기계학습의 툴을 얻게 되는 것에 더하여 지도 학습 또는 자율 학습 등의 툴을 얻게 되는 것으로 툴을 얻게 되는 것뿐만 아니라 이러한 툴을 적용하여 효율적인 기계 학습 시스템을 디자인할 수 있게 될 수 있기를 바란다.

02:33

여기까지가 우리가 살펴본 내용이다. 이번 강의를 통해 배운 모든 내용이고 이제 여러분은 기계 학습의 전문가라고 생각해도 좋다. 알다시피 기계 학습은 과학, 기술, 산업에 큰 영향력을 미치는 기술이다. 이제 여러분은 이러한 기계 학습 툴을 매우 효과적으로 활용할 수 있게 되었다. 이번 강의를 들은 여러분이 좋은 시스템, 어플리케이션, 제품을 만드는데 이를 활용할 수 있기를 바란다. 또한 기계 학습을 통해 여러분의 더 나은 삶을 영위할 뿐 아니라 다른 이들의 삶에도 도움을 줄 수 있기를 바란다. 나 역시 이번 강의를 통해 수업을 하면서 무척 재미있었다. 고맙게 생각한다. 마무리 하기 전에 마지막으로 한마디 더 하겠다. 그리 오래 전은 아니지만 내가 학생이었을 때, 심지어 지금도 다양한 강의를 듣고, 시간이 있을 때 새로운 것을 배우려고 노력한다. 이러한 것을 배울 때 많은 시간이 필요하다는 것은 잘 알고 있다. 여러분의 삶이 이것 말고도 많은 것들을 하느라 바쁘다는 것도 알고 있다. 하지만 여러분이 시간을 내서 모든 강의를 들었고, 몇 시간 동안 강의가 이어진 적도 있다. 여러분 중 많은 이가 시간을 내서 복습 문제를 풀어보고, 프로그래밍 연습 문제를 다뤄봤다. 이는 매우 길고 복잡한 프로그래밍 연습문제였다. 고맙게 생각한다. 이번 강의를 보며 열심히 공부하고 시간을 투자하며 많은 노력을 한 것을 알고 있다. 따라서 이번 강의를 통해 많은 것을 얻어갈 수 있었으면 한다. 이 강좌를 열심히 들어줘서 고맙다.