

斯坦福大学 2014 机器学习教程

个人笔记 (v2.0)

摘要

本笔记是针对斯坦福大学 2014 年机器学习课程
视频做的个人笔记

最后修改: 2015-2-17

黄海广

Haiguang2000@qq.com

斯坦福大学 2014 机器学习教程中文笔记

课程概述

Machine Learning(机器学习)是研究计算机怎样模拟或实现人类的学习行为, 以获取新的知识或技能, 重新组织已有的知识结构使之不断改善自身的性能。它是人工智能的核心, 是使计算机具有智能的根本途径, 其应用遍及人工智能的各个领域, 它主要使用归纳、综合而不是演绎。在过去的十年中, 机器学习帮助我们自动驾驶汽车, 有效的语音识别, 有效的网络搜索, 并极大地提高了人类基因组的认识。机器学习是当今非常普遍, 你可能会使用这一天几十倍而不自知。很多研究者也认为这是最好的人工智能的取得方式。在本课中, 您将学习最有效的机器学习技术, 并获得实践, 让它们为自己的工作。更重要的是, 你会不仅得到理论基础的学习, 而且获得那些需要快速和强大的应用技术解决问题的实用技术。最后, 你会学到一些硅谷利用机器学习和人工智能的最佳实践创新。

本课程提供了一个广泛的介绍机器学习、数据挖掘、统计模式识别的课程。主题包括:

(一) 监督学习 (参数/非参数算法, 支持向量机, 核函数, 神经网络)。(二) 无监督学习 (聚类, 降维, 推荐系统, 深入学习推荐)。(三) 在机器学习的最佳实践 (偏差/方差理论; 在机器学习和人工智能创新过程)。本课程还将使用大量的案例研究, 您还将学习如何运用学习算法构建智能机器人 (感知, 控制), 文本的理解 (Web 搜索, 反垃圾邮件), 计算机视觉, 医疗信息, 音频, 数据挖掘, 和其他领域。

本课程需要 10 周共 18 节课, 相对以前的机器学习视频, 这个视频更加清晰, 而且每课都有 ppt 课件, 推荐学习。

本人是中国海洋大学 2014 级博士生, 目前刚开始接触机器学习, 我下载了这次课程的所有视频和课件给大家分享。中英文字幕来自于 <https://www.coursera.org/course/ml>, 主要是教育无边界字幕组翻译, 本人把中英文字幕进行合并, 并翻译了部分字幕, 对视频进行封装, 归类, 并翻译了课程目录, 做好课程索引文件, 希望对大家有所帮助。部分视频中文字幕由中国海洋大学的博士生翻译。视频已经翻译完毕 (除了 OCTAVE 教程没有翻译), 如果下载了视频, 可以直接在文档中打开视频, 内嵌中英文字幕。

目前我正在整理中文笔记, 主要是根据视频内容和中文字幕以及 ppt 来制作, 部分来源于网络, 如“小小人_v”的笔记, 并持续更新。

本人水平有限, 如有公式、算法错误, 请及时指出, 发邮件给我。

黄海广

2014-12-16 夜

■ 文档修改历史

版本号	版本日期	修改总结	修订人
1.0	2014.12.16	创建初稿	黄海广
1.1	2014.12.31	修改	黄海广
2.0	2015.02.17	修改	黄海广

目录

第 1 周	1
一、 引言(Introduction)	1
1.1 欢迎.....	1
1.2 机器学习是什么？	4
1.3 监督学习.....	6
1.4 无监督学习.....	10
二、 单变量线性回归(Linear Regression with One Variable)	15
2.1 模型表示.....	15
2.2 代价函数.....	17
2.3 代价函数的直观理解 I.....	19
2.4 代价函数的直观理解 II.....	20
2.5 梯度下降.....	21
2.6 梯度下降的直观理解.....	22
2.7 梯度下降的线性回归.....	23
2.8 接下来的内容.....	24
三、 线性代数回顾(Linear Algebra Review).....	25
3.1 矩阵和向量.....	25
3.2 加法和标量乘法.....	27
3.3 矩阵向量乘法.....	28
3.4 矩阵乘法.....	29
3.5 矩阵乘法的性质.....	30
3.6 逆、转置.....	31
第 2 周	32
四、 多变量线性回归(Linear Regression with Multiple Variables).....	32
4.1 多维特征.....	32
4.2 多变量梯度下降.....	34
4.3 梯度下降法实践 1-特征缩放.....	36
4.4 梯度下降法实践 2-学习率	38
4.5 特征和多项式回归.....	39
4.6 正规方程.....	41
4.7 正规方程及不可逆性（可选）	44
五、 Octave 教程(Octave Tutorial).....	45
5.1 基本操作.....	45
5.2 移动数据.....	46
5.3 计算数据.....	47
5.4 绘图数据.....	48
5.5 控制语句: for, while, if 语句	49
5.6 矢量化.....	50
5.7 工作和提交的编程练习	51
第 3 周	52
六、 逻辑回归(Logistic Regression).....	52
6.1 分类问题.....	52

6.2 假说表示.....	53
6.3 判定边界.....	55
6.4 代价函数.....	57
6.5 简化的成本函数和梯度下降.....	60
6.6 高级优化.....	61
6.7 多类分类：一个对所有.....	62
七、正则化(Regularization)	63
7.1 过拟合的问题.....	63
7.2 代价函数.....	65
7.3 正则化线性回归.....	67
7.4 正则化的逻辑回归模型.....	68
第 4 周	69
第八、神经网络：表述(Neural Networks: Representation).....	69
8.1 非线性假设.....	69
8.2 神经元和大脑.....	71
8.3 模型表示 1.....	75
8.4 模型表示 2.....	79
8.5 特征和直观理解 I.....	81
8.6 样本和直观理解 II.....	83
8.7 多类分类.....	85
第 5 周	86
九、神经网络的学习(Neural Networks: Learning)	86
9.1 代价函数.....	86
9.2 反向传播算法.....	88
9.3 反向传播算法的直观理解.....	91
9.4 实现注意：展开参数.....	93
9.5 梯度检验.....	94
9.6 随机初始化.....	96
9.7 综合起来.....	97
9.8 自主驾驶.....	98
第 6 周	101
十、应用机器学习的建议(Advice for Applying Machine Learning)	101
10.1 决定下一步做什么.....	101
10.2 评估一个假设.....	102
10.3 模型选择和交叉验证集.....	103
10.4 诊断偏差和方差.....	105
10.5 归一化和偏差/方差.....	107
10.6 学习曲线.....	109
10.7 决定下一步做什么.....	111
十一、机器学习系统的设计(Machine Learning System Design)	113
11.1 首先要做什么.....	113
11.2 误差分析.....	114
11.3 类偏斜的误差度量.....	115
11.4 查全率和查准率之间的权衡.....	116

11.5 机器学习的数据.....	118
第 7 周	119
十二、支持向量机(Support Vector Machines)	119
12.1 优化目标.....	119
12.2 大边界的直观理解.....	126
12.3 数学背后的大边界分类(可选)	132
12.4 核函数 1.....	139
12.5 核函数 2.....	141
12.6 使用支持向量机.....	143
第 8 周	146
十三、聚类(Clustering)	146
13.1 无监督学习：简介.....	146
13.2 K-均值算法	149
13.3 优化目标.....	151
13.4 随机初始化.....	152
13.5 选择聚类数.....	153
十四、降维(Dimensionality Reduction).....	154
14.1 动机一：数据压缩.....	154
14.2 动机二：数据可视化.....	157
14.3 主成分分析问题.....	158
14.4 主成分分析算法.....	160
14.5 选择主成分的数量.....	161
14.6 重建的压缩表示.....	162
14.7 主成分分析法的应用建议.....	164
第 9 周	165
十五、异常检测(Anomaly Detection)	165
15.1 问题的动机.....	165
15.2 高斯分布.....	167
15.3 算法.....	168
15.4 开发和评价一个异常检测系统.....	170
15.5 异常检测与监督学习对比.....	171
15.6 选择特征.....	172
15.7 多元高斯分布(可选)	174
15.8 使用多元高斯分布进行异常检测(可选)	177
十六、推荐系统(Recommender Systems).....	180
16.1 问题形式化.....	180
16.2 基于内容的推荐系统.....	182
16.3 协同过滤.....	184
16.4 协同过滤算法.....	185
16.5 矢量化：低秩矩阵分解.....	186
16.6 推荐工作上的细节：均值归一化.....	188
第 10 周	189
十七、大规模机器学习(Large Scale Machine Learning).....	189
17.1 大型数据集的学习.....	189

17.2 随机梯度下降法.....	190
17.3 微型批量梯度下降.....	191
17.4 随机梯度下降收敛.....	192
17.5 在线学习.....	194
17.6 映射化简和数据并行.....	195
十八、应用实例：图片文字识别(Application Example: Photo OCR)	196
18.1 问题描述和流程图.....	196
18.2 滑动窗口.....	197
18.3 获取大量数据和人工数据.....	199
18.4 上限分析：哪部分管道的接下去做	200
十九、总结(Conclusion).....	201
19.1 总结和致谢.....	201

第 1 周

一、 引言(Introduction)

1.1 欢迎

参考视频: [1 - 1 - Welcome \(7 min\).mkv](#)

第一个视频主要讲了什么是机器学习，机器学习能做些什么事情。

机器学习是目前信息技术中最激动人心的方向之一。在这门课中，你将学习到这门技术的前沿，并可以自己实现学习机器学习的算法。

你或许每天都在不知不觉中使用了机器学习的算法每次，你打开谷歌、必应搜索到你需要的内容，正是因为他们有良好的学习算法。谷歌和微软实现了学习算法来排行网页每次，你用 Facebook 或苹果的图片分类程序他能认出你朋友的照片，这也是机器学习。每次您阅读您的电子邮件垃圾邮件筛选器，可以帮你过滤大量的垃圾邮件这也是一种学习算法。对我来说，我感到激动的原因之一是有一天做出一个和人类一样聪明的机器。实现这个想法任重而道远，许多 AI 研究者认为，实现这个目标最好的方法是通过让机器试着模仿人的大脑学习我会在这门课中介绍一点这方面的内容。

在这门课中，你还讲学习到关于机器学习的前沿状况。但事实上只了解算法、数学并不能解决你关心的实际的问题。所以，我们将花大量的时间做练习，从而你自己能实现每个这些算法，从而了解内部机理。

那么，为什么机器学习如此受欢迎呢？原因是，机器学习不只是用于人工智能领域。我们创造智能的机器，有很多基础的知识。比如，我们可以让机器找到 A 与 B 之间的最短路径，但我们仍然不知道怎么让机器做更有趣的事情，如 web 搜索、照片标记、反垃圾邮件。我们发现，唯一方法是让机器自己学习怎么来解决问题。所以，机器学习已经成为计算机的一个能力。

现在它涉及到各个行业和基础科学中。我从事于机器学习，但我每个星期都跟直升机飞行员、生物学家、很多计算机系统程序员交流（我在斯坦福大学的同事同时也是这样）和平均每个星期会从硅谷收到两、三个电子邮件，这些联系我的人都对将学习算法应用于他们

自己的问题感兴趣。这表明机器学习涉及的问题非常广泛。有机器人、计算生物学、硅谷中大量的问题都收到机器学习的影响。

这里有一些机器学习的案例。比如说，数据库挖掘。机器学习被用于数据挖掘的原因之一是网络和自动化技术的增长，这意味着，我们有史上最大的数据集比如说，大量的硅谷公司正在收集 web 上的单击数据，也称为点击流数据，并尝试使用机器学习算法来分析数据，更好的了解用户，并为用户提供更好的服务。这在硅谷有巨大的市场。再比如，医疗记录。随着自动化的出现，我们现在有了电子医疗记录。如果我们可以把医疗记录变成医学知识，我们就可以更好地理解疾病。再如，计算生物学。还是因为自动化技术，生物学家们收集的大量基因数据序列、DNA 序列和等等，机器运行算法让我们更好地了解人类基因组，大家都知道这对人类意味着什么。再比如，工程方面，在工程的所有领域，我们有越来越大、越来越大的数据集，我们试图使用学习算法，来理解这些数据。另外，在机械应用中，有些人不能直接操作。例如，我已经在无人直升机领域工作了许多年。我们不知道如何写一段程序让直升机自己飞。我们唯一能做的就是让计算机自己学习如何驾驶直升机。

手写识别：现在我们能够非常便宜地把信寄到这个美国甚至全世界的原因之一就是当你写一个像这样的信封，一种学习算法已经学会如何读你信封，它可以自动选择路径，所以我们只需要花几个美分把这封信寄到数千英里外。

事实上，如果你看过自然语言处理或计算机视觉，这些语言理解或图像理解都是属于 AI 领域。大部分的自然语言处理和大部分的计算机视觉，都应用了机器学习。学习算法还广泛用于自定制程序。每次你去亚马逊或 Netflix 或 iTunes Genius，它都会给出其他电影或产品或音乐的建议，这是一种学习算法。仔细想一想，他们有百万的用户；但他们没有办法为百万用户，编写百万个不同程序。软件能给这些自定制的建议的唯一方法是通过学习你的行为，来为你定制服务。

最后学习算法被用来理解人类的学习和了解大脑。

我们将谈论如何用这些推进我们的 AI 梦想。几个月前，一名学生给我一篇文章关于最顶尖的 12 个 IT 技能。拥有了这些技能 HR 绝对不会拒绝你。这是稍显陈旧的文章，但在这个列表最顶部就是机器学习的技能。

在斯坦福大学，招聘人员联系我，让我推荐机器学习学生毕业的人远远多于机器学习的毕业生。所以我认为需求远远没有被满足现在学习“机器学习”非常好，在这门课中，我希望告诉你们很多机器学习的知识。

在接下来的视频中，我们将开始给更正式的定义，什么是机器学习。然后我们会开始学

习机器学习的主要问题和算法你会了解一些主要的机器学习的术语，并开始了解不同的算法，用哪种算法更合适。

1.2 机器学习是什么？

参考视频: [1 - 2 - What is Machine Learning \(7 min\).mkv](#)

机器学习是什么？在本视频中，我们会尝试着进行定义，同时让你懂得何时会使用机器学习。实际上，即使是在机器学习的专业人士中，也不存在一个被广泛认可的定义来准确地定义机器学习是什么或不是什么，现在我将告诉你一些人们尝试定义的示例。第一个机器学习的定义来自于 Arthur Samuel。他定义机器学习为，在进行特定编程的情况下，给予计算机学习能力的领域。Samuel 的定义可以追溯到 50 年代，他编写了一个西洋棋程序。这程序神奇之处在于，编程者自己并不是个下棋高手。但因为他太菜了，于是就通过编程，让西洋棋程序自己跟自己下了上万盘棋。通过观察哪种布局（棋盘位置）会赢，哪种布局会输，久而久之，这西洋棋程序明白了什么是好的布局，什么样是坏的布局。然后就牛逼大发了，程序通过学习后，玩西洋棋的水平超过了 Samuel。这绝对是令人注目的成果。

尽管编写者自己是个菜鸟，但因为计算机有着足够的耐心，去下上万盘的棋，没有人有这耐心去下这么多盘棋。通过这些练习，计算机获得无比丰富的经验，于是渐渐成为了比 Samuel 更厉害的西洋棋手。上述是个有点不正式的定义，也比较古老。另一个年代近一点的定义，由 Tom Mitchell 提出，来自卡内基梅隆大学，Tom 定义的机器学习是，一个好的学习问题定义如下，他说，一个程序被认为能从经验 E 中学习，解决任务 T ，达到性能度量值 P ，当且仅当，有了经验 E 后，经过 P 评判，程序在处理 T 时的性能有所提升。我认为经验 e 就是程序上万次的自我练习的经验而任务 t 就是下棋。性能度量值 p 呢，就是它在与一些新的对手比赛时，赢得比赛的概率。

在这些视频中，除了我教你的内容以外，我偶尔会问你一个问题，确保你对内容有所理解。说曹操，曹操到，顶部是 Tom Mitchell 的机器学习的定义，我们假设您的电子邮件程序会观察收到的邮件是否被你标记为垃圾邮件。在这种 Email 客户端中，你点击“垃圾邮件”按钮，报告某些 email 为垃圾邮件，不会影响别的邮件。基于被标记为垃圾的邮件，您的电子邮件程序能更好地学习如何过滤垃圾邮件。请问，在这个设定中，任务 T 是什么？几秒钟后，该视频将暂停。当它暂停时，您可以使用鼠标，选择这四个单选按钮中的一个，让我知道这四个，你所认为正确的选项。它可能是性能度量值 P 。所以，以性能度量值 P 为标准，这个任务的性能，也就是这个任务 T 的系统性能，将在学习经验 E 后得到提高。

本课中，我希望教你有关各种不同类型的学习算法。目前存在几种不同类型的学习算法。主要的两种类型被我们称之为监督学习和无监督学习。在接下来的几个视频中，我会给出这些术语的定义。这里简单说两句，监督学习这个想法是指，我们将教计算机如何去完成任务，而在无监督学习中，我们打算让它自己进行学习。如果对这两个术语仍一头雾水，请不要担心，在后面的两个视频中，我会具体介绍这两种学习算法。此外你将听到诸如，强化学习和推荐系统等各种术语。这些都是机器学习算法的一员，以后我们都将介绍到，但学习算法最常用两个类型就是监督学习、无监督学习。我会在接下来的两个视频中给出它们的定义。本课中，我们将花费最多的精力来讨论这两种学习算法。而另一个会花费大量时间的任务是了解应用学习算法的实用建议。

我非常注重这部分内容，实际上，就这些内容而言我不知道还有哪所大学会介绍到。给你讲授学习算法就好像给你一套工具，相比于提供工具，可能更重要的，是教你如何使用这些工具。我喜欢把这比喻成学习当木匠。想象一下，某人教你如何成为一名木匠，说这是锤子，这是螺丝刀，锯子，祝你好运，再见。这种教法不好，不是吗？你拥有这些工具，但更重要的是，你要学会如何恰当地使用这些工具。会用与不会用的人之间，存在着鸿沟。尤其是知道如何使用这些机器学习算法的，与那些不知道如何使用的人。在硅谷我住的地方，当我走访不同的公司，即使是最顶尖的公司，很多时候我都看到人们试图将机器学习算法应用于某些问题。有时他们甚至已经为此花了六个月之久。但当我看着他们所忙碌的事情时，我想说，哎呀，我本来可以在六个月前就告诉他们，他们应该采取一种学习算法，稍加修改进行使用，然后成功的机会绝对会高得多所以在本课中，我们要花很多时间来探讨，如果你真的试图开发机器学习系统，探讨如何做出最好的实践类型决策，才能决定你的方式来构建你的系统，这样做的话，当你运用学习算法时，就不太容易变成那些为寻找一个解决方案花费6个月之久的人们的中一员。他们可能已经有了大体的框架，只是没法正确的工作于是这就浪费了六个月的时间。所以我会花很多时间来教你这些机器学习、人工智能的最佳实践以及如何让它们工作，我们该如何去做，硅谷和世界各地最优秀的人是怎样做的。我希望能帮你成为最优秀的人才，通过了解如何设计和构建机器学习和人工智能系统。

这就是机器学习，这些都是我希望讲授的主题。在下一个视频里，我会定义什么是监督学习，什么是无监督学习。此外，探讨何时使用二者。

1.3 监督学习

参考视频: [1 - 3 - Supervised Learning \(12 min\).mkv](#)

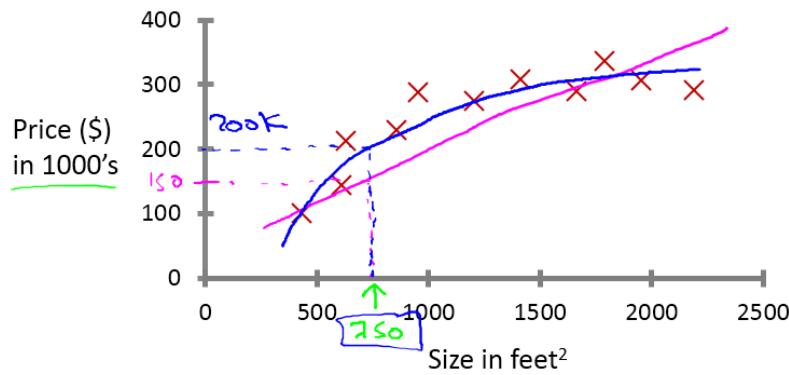
在这段视频中, 我要定义可能是最常见一种机器学习问题: 那就是监督学习。我将在后面正式定义监督学习。

我们用一个例子介绍什么是监督学习把正式的定义放在后面介绍。假如说你想预测房价。

前阵子, 一个学生从波特兰俄勒冈州的研究所收集了一些房价的数据。你把这些数据画出来, 看起来是这个样子: 横轴表示房子的面积, 单位是平方英尺, 纵轴表示房价, 单位是千美元。那基于这组数据, 假如你有一个朋友, 他有一套 750 平方英尺房子, 现在他希望把房子卖掉, 他想知道这房子能卖多少钱。

那么关于这个问题, 机器学习算法将会怎么帮助你呢?

Housing price prediction.



Supervised Learning
"right answers" given

Regression: Predict continuous valued output (price)

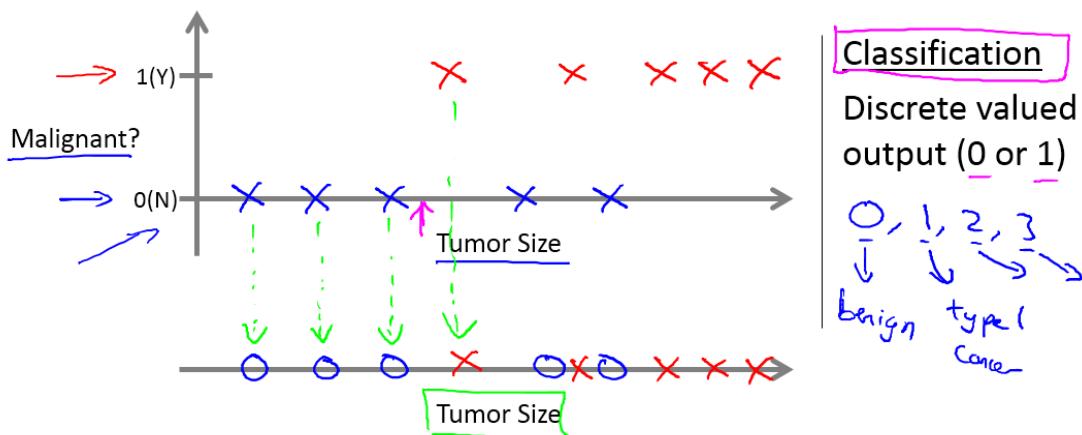
我们应用学习算法, 可以在这组数据中画一条直线, 或者换句话说, 拟合一条直线, 根据这条线我们可以推测出, 这套房子可能卖\$150,000, 当然这不是唯一的算法。可能还有更好的, 比如我们不用直线拟合这些数据, 用二次方程去拟合可能效果会更好。根据二次方程的曲线, 我们可以从这个点推测出, 这套房子能卖接近\$200,000。稍后我们将讨论如何选择学习算法, 如何决定用直线还是二次方程来拟合。两个方案中有一个能让你朋友的房子出售得更合理。这些都是学习算法里面很好的例子。以上就是监督学习的例子。

可以看出，监督学习指的就是我们给学习算法一个数据集。这个数据集由“正确答案”组成。在房价的例子中，我们给了一系列房子的数据，我们给定数据集中每个样本的正确价格，即它们实际的售价然后运用学习算法，算出更多的正确答案。比如你朋友那个新房子的价格。用术语来讲，这叫做回归问题。我们试着推测出一个连续值的结果，即房子的价格。一般房子的价格会记到美分，所以房价实际上是一系列离散的值 但是我们通常又把房价看成实数，看成是标量，所以又把它看成一个连续的数值。

回归这个词的意思是，我们在试着推测出这一系列连续值属性。

我再举另外一个监督学习的例子。我和一些朋友之前研究过这个。假设说你想通过查看病历来推测乳腺癌良性与否，假如有人检测出乳腺肿瘤，恶性肿瘤有害并且十分危险，而良性的肿瘤危害就没那么大，所以人们显然会很在意这个问题。

Breast cancer (malignant, benign)



让我们来看一组数据：这个数据集中，横轴表示肿瘤的大小，纵轴上，我标出 1 和 0 表示是或者不是恶性肿瘤。我们之前见过的肿瘤，如果是恶性则记为 1，不是恶性，或者说良性记为 0。

我有 5 个良性肿瘤样本，在 1 的位置有 5 个恶性肿瘤样本。现在我们有一个朋友很不幸检查出乳腺肿瘤。假设说她的肿瘤大概这么大，那么机器学习的问题就在于，你能否估算出肿瘤是恶性的或是良性的概率。用术语来讲，这是一个分类问题。

分类指的是，我们试着推测出离散的输出值：0 或 1 良性或恶性，而事实上在分类问题中，输出可能不止两个值。比如说可能有三种乳腺癌，所以你希望预测离散输出 0 1 2 3

0 代表良性，1 表示第一类乳腺癌，2 表示第二类癌症，3 表示第三类，但这也是分类问题。

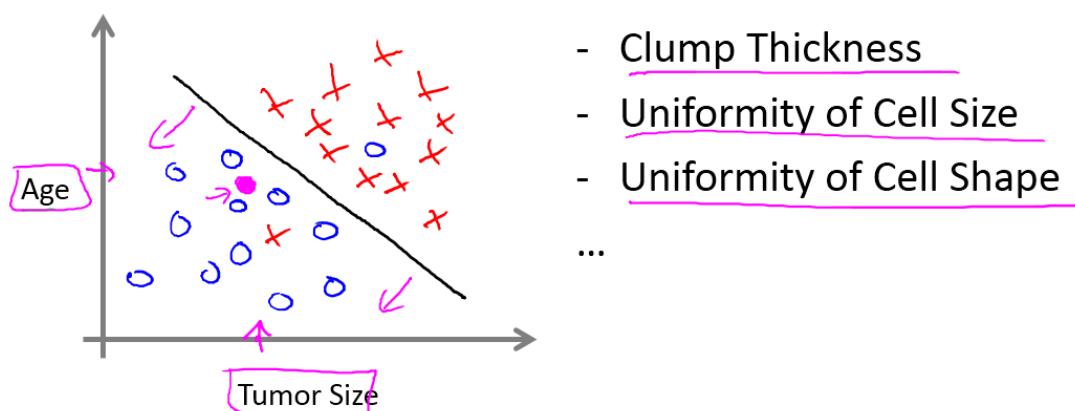
因为这几个离散的输出分别对应良性，第一类第二类或者第三类癌症，在分类问题中我

们可以用另一种方式绘制这些数据点。

现在我用不同的符号来表示这些数据。既然我们把肿瘤的尺寸看做区分恶性或良性的特征，那么我可以这么画，我用不同的符号来表示良性和恶性肿瘤。或者说是负样本和正样本。现在我们不全部画 X ，良性的肿瘤改成用 O 表示，恶性的继续用 X 表示。来预测肿瘤的恶性与否。

在其它一些机器学习问题中，可能会遇到不止一种特征。举个例子，我们不仅知道肿瘤的尺寸，还知道对应患者的年龄。在其他机器学习问题中，我们通常有更多的特征，我朋友研究这个问题时，通常采用这些特征，比如肿块密度，肿瘤细胞尺寸的一致性和形状的一致性等等，还有一些其他的特征。这就是我们即将学到最有趣的学习算法之一。

那种算法不仅能处理 2 种 3 种或 5 种特征，即使有无限多种特征都可以处理。



上图中，我列举了总共 5 种不同的特征，坐标轴上的两种和右边的 3 种，但是在一些学习问题中，你希望不只用 3 种或 5 种特征。相反，你想用无限多种特征，好让你的算法可以利用大量的特征，或者说线索来做推测。那你怎么处理无限多个特征，甚至怎么存储这些特征都存在问题，你电脑的内存肯定不够用。我们以后会讲一个算法，叫支持向量机，里面有一个巧妙的数学技巧，能让计算机处理无限多个特征。想象一下，我没有写下这两种和右边的三种特征，而是在一个无限长的列表里面，一直写一直写不停的写，写下无限多个特征，事实上，我们能用算法来处理它们。

现在来回顾一下，这节课我们介绍了监督学习。其基本思想是，我们数据集中的每个样本都有相应的“正确答案”。再根据这些样本作出预测，就像房子和肿瘤的例子中做的那样。我们还介绍了回归问题，即通过回归来推出一个连续的输出，之后我们介绍了分类问题，其目标是推出一组离散的结果。

现在来个小测验：假设你经营着一家公司，你想开发学习算法来处理这两个问题：

1. 你有一大批同样的货物，想象一下，你有上千件一模一样的货物等待出售，这时你想预测接下来的三个月能卖多少件？
2. 你有许多客户，这时你想写一个软件来检验每一个用户的账户。对于每一个账户，你要判断它们是否曾经被盗过？

那这两个问题，它们属于分类问题、还是回归问题？

问题一是一个回归问题，因为你知道，如果我有数千件货物，我会把它看成一个实数，一个连续的值。因此卖出的物品数，也是一个连续的值。

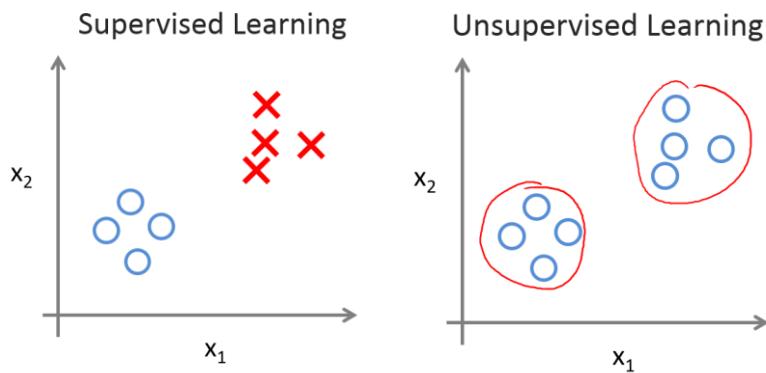
问题二是一个分类问题，因为我会把预测的值，用 0 来表示账户未被盗，用 1 表示账户曾经被盗过。所以我们根据账号是否被盗过，把它们定为 0 或 1，然后用算法推测一个账号是 0 还是 1，因为只有少数的离散值，所以我把它归为分类问题。

以上就是监督学习的内容。

1.4 无监督学习

参考视频: [1 - 4 - Unsupervised Learning \(14 min\).mkv](#)

本次视频中, 我们将介绍第二种主要的机器学习问题。叫做无监督学习。

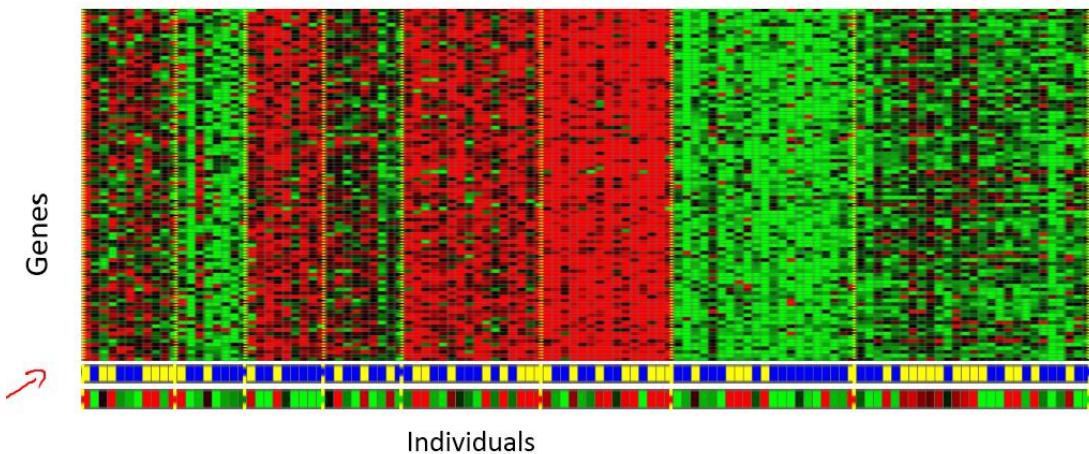


上个视频中, 已经介绍了监督学习。回想当时的数据集, 如图表所示, 这个数据集中每条数据都已经标明是阴性或阳性, 即是良性或恶性肿瘤。所以, 对于监督学习里的每条数据, 我们已经清楚地知道, 训练集对应的正确答案, 是良性或恶性了。

在无监督学习中, 我们已知的数据。看上去有点不一样, 不同于监督学习的数据的样子, 即无监督学习中没有任何的标签或者是有相同的标签或者就是没标签。所以我们已知数据集, 却不知如何处理, 也未告知每个数据点是什么。别的都不知道, 就是一个数据集。你能从数据中找到某种结构吗? 针对数据集, 无监督学习就能判断出数据有两个不同的聚集簇。这是一个, 那是另一个, 二者不同。是的, 无监督学习算法可能会把这些数据分成两个不同的簇。所以叫做聚类算法。事实证明, 它能被用在很多地方。

聚类应用的一个例子就是在谷歌新闻中。如果你以前从来没见过它, 你可以到这个 URL 网址 news.google.com 去看看。谷歌新闻每天都在, 收集非常多, 非常多的网络的新闻内容。它再将这些新闻分组, 组成有关联的新闻。所以谷歌新闻做的就是搜索非常多的新闻事件, 自动地把它们聚类到一起。所以, 这些新闻事件全是同一主题的, 所以显示到一起。

事实证明, 聚类算法和无监督学习算法同样还用在很多其它的问题上。



其中就有基因学的理解应用。一个 DNA 微观数据的例子。基本思想是输入一组不同个体，对其中的每个个体，你要分析出它们是否有一个特定的基因。技术上，你要分析多少特定基因已经表达。所以这些颜色，红，绿，灰等等颜色，这些颜色展示了相应的程度，即不同的个体是否有着一个特定的基因。你能做的就是运行一个聚类算法，把个体聚类到不同的类或不同类型的组（人）……

所以这个就是无监督学习，因为我们没有提前告知算法一些信息，比如，这是第一类的人，那些是第二类的人，还有第三类，等等。我们只是说，是的，这是有一堆数据。我不知道数据里面有什么。我不知道谁是什么类型。我甚至不知道人们有哪些不同的类型，这些类型又是什么。但你能自动地找到数据中的结构吗？就是说你要自动地聚类那些个体到各个类，我没法提前知道哪些是哪些。因为我们没有给算法正确答案来回应数据集中的数据，所以这就是无监督学习。

无监督学习或聚集有着大量的应用。它用于组织大型计算机集群。我有些朋友在大数据中心工作，那里有大型的计算机集群，他们想解决什么样的机器易于协同地工作，如果你能够让那些机器协同工作，你就能让你的数据中心工作得更高效。第二种应用就是社交网络的分析。所以已知你朋友的信息，比如你经常发 email 的，或是你 Facebook 的朋友、谷歌+圈子的朋友，我们能否自动地给出朋友的分组呢？即每组里的人们彼此都熟识，认识组里的所有人？还有市场分割。许多公司有大型的数据库，存储消费者信息。所以，你能检索这些顾客数据集，自动地发现市场分类，并自动地把顾客划分到不同的细分市场中，你才能自动并更有效地销售或不同的细分市场一起进行销售。这也是无监督学习，因为我们拥有所有的顾客数据，但我们没有提前知道是什么的细分市场，以及分别有哪些我们数据集中的顾客。我们不知道谁是在一号细分市场，谁在二号市场，等等。那我们就必须让算法从数据中发现

这一切。最后，无监督学习也可用于天文数据分析，这些聚类算法给出了令人惊讶、有趣、有用的理论，解释了星系是如何诞生的。这些都是聚类的例子，聚类只是无监督学习中的一种。

我现在告诉你们另一种。我先来介绍鸡尾酒宴问题。嗯，你参加过鸡尾酒宴吧？你可以想像下，有个宴会房间里满是人，全部坐着，都在聊天，这么多人同时在聊天，声音彼此重叠，因为每个人都在说话，同一时间都在说话，你几乎听不到你面前那人的声音。所以，可能在一个这样的鸡尾酒宴中的两个人，他俩同时都在说话，假设现在是在个有些小的鸡尾酒宴中。我们放两个麦克风在房间中，因为这些麦克风在两个地方，离说话人的距离不同每个麦克风记录下不同的声音，虽然是同样的两个说话人。听起来像是两份录音被叠加到一起，或是被归结到一起，产生了我们现在的这些录音。另外，这个算法还会区分出两个音频资源，这两个可以合成或合并成之前的录音，实际上，鸡尾酒算法的第一个输出结果是：

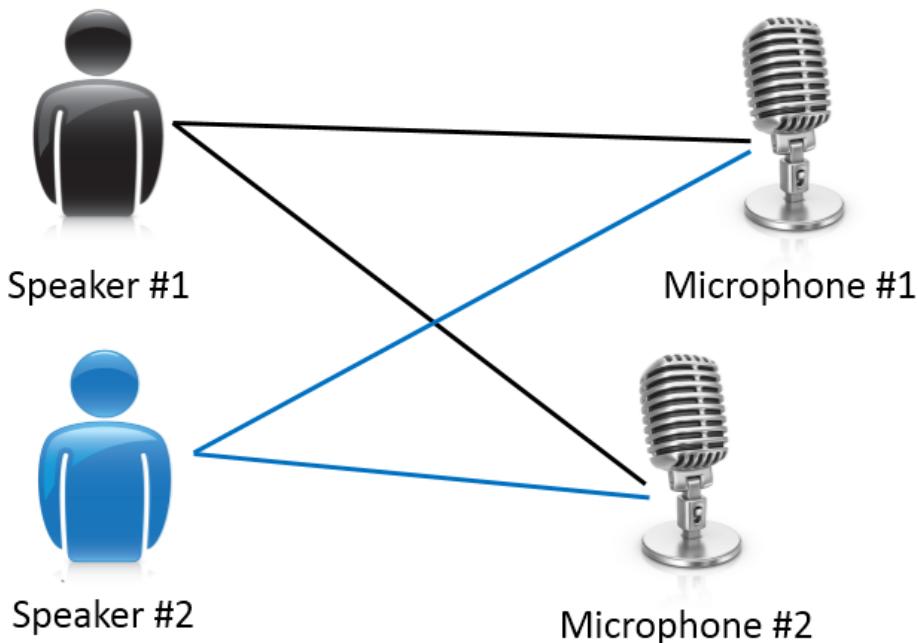
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

所以，已经把英语的声音从录音中分离出来了。

第二个输出是这样：

1, 2, 3, 4, 5, 6, 7, 8, 9, 10.

Cocktail party problem



看看这个无监督学习算法，实现这个得要多么的复杂，是吧？它似乎是这样，为了构建

这个应用，完成这个音频处理似乎需要你去写大量的代码或链接到一堆的合成器 JAVA 库，处理音频的库，看上去绝对是个复杂的程序，去完成这个从音频中分离出音频。事实上，这个算法对应你刚才知道的那个问题的算法可以就用一行代码来完成。

就是这里展示的代码： `[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');`

研究人员花费了大量时间才最终实现这行代码。我不是说这个是简单的问题，但它证明了，当你使用正确的编程环境，许多学习算法是相当短的程序。所以，这也是为什么在本课中，我们打算使用 Octave 编程环境。Octave 是免费的开源软件，使用一个像 Octave 或 Matlab 的工具，许多学习算法变得只有几行代码就可实现。

后面，我会教你们一点关于如何使用 Octave 的知识，你就可以用 Octave 来实现一些算法了。或者，如果你有 Matlab（盗版？），你也可以用 Matlab。事实上，在硅谷里，对大量机器学习算法，我们第一步就是建原型，在 Octave 建软件原型，因为软件在 Octave 中可以令人难以置信地、快速地实现这些学习算法。这里的这些函数比如 SVM（支持向量机）函数，奇异值分解，Octave 里已经建好了。如果你试图完成这个工作，但借助 C++ 或 Java 的话，你会需要很多很多行的代码，并链接复杂的 C++ 或 Java 库。所以，你可以实现这些算法，借助 C++ 或 Java 或 Python，它只是用这些语言来实现会更加复杂。

我已经见到，在我教机器学习将近十年后的现在，发现，学习可以更加高速，如果使用 Octave 作为编程环境，如果使用 Octave 作为学习工具，以及作为原型工具，它会让你对学习算法的学习和建原型快上许多。

事实上，许多人在大硅谷的公司里做的其实就是，使用一种工具像 Octave 来做第一步的学习算法的原型搭建，只有在你已经让它工作后，你才移植它到 C++ 或 Java 或别的语言。事实证明，这样做通常可以让你的算法运行得比直接用 C++ 实现更快，所以，我知道，作为一名指导者，我必须说“相信我”，但对你们中从未使用过 Octave 这种编程环境的人，我还是要告诉你们这一点一定要相信我，我想，对你们而言，我认为你们的时间，你们的开发时间是最有价值的资源。我已经见过很多人这样做了，我把你看作是机器学习研究员，或机器学习开发人员，想更加高产的话，你要学会使用这个原型工具，开始使用 Octave。

最后，总结下本视频内容，我有个简短的复习题给你们。

我们介绍了无监督学习，它是学习策略，交给算法大量的数据，并让算法为我们从数据中找出某种结构。

好的，希望你们还记得垃圾邮件问题。如果你有标记好的数据，区别好是垃圾还是非垃圾邮件，我们把这个当作监督学习问题。

新闻事件分类的例子，就是那个谷歌新闻的例子，我们在本视频中有见到了，我们看到，可以用一个聚类算法来聚类这些文章到一起，所以是**无监督学习**。

细分市场的例子，我在更早一点的时间讲过，你可以当作**无监督学习**问题，因为我只是拿到算法数据，再让算法去自动地发现细分市场。

最后一个例子，**糖尿病**，这个其实就像是我们的乳腺癌，上个视频里的。只是替换了好、坏肿瘤，良性、恶性肿瘤，我们改用糖尿病或没病。所以我们把这个当作**监督学习**，我们能够解决它，作为一个监督学习问题，就像我们在乳腺癌数据中做的一样。

好了，以上就是无监督学习的视频内容，在下一个视频中，我们将深入探究特定的学习算法，开始介绍这些算法是如何工作的，和我们还有你如何来实现它们。

二、单变量线性回归(Linear Regression with One Variable)

2.1 模型表示

参考视频: [2 - 1 - Model Representation \(8 min\).mkv](#)

之前的房屋交易问题为例，假使我们回归问题的训练集（Training Set）如下表所示：

Size in feet ² (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

我们将要用来描述这个回归问题的标记如下：

m 代表训练集中实例的数量

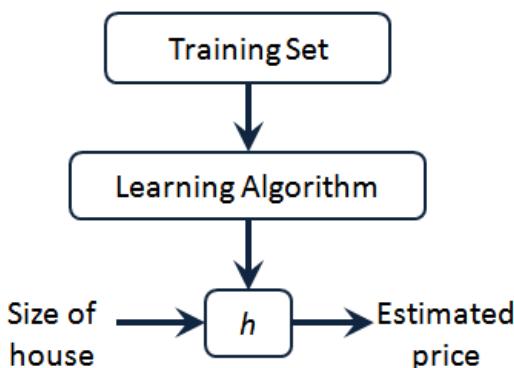
x 代表特征/输入变量

y 代表目标变量/输出变量

(x,y) 代表训练集中的实例

$(x^{(i)},y^{(i)})$ 代表第 i 个观察实例

h 代表学习算法的解决方案或函数也称为假设（hypothesis）



因而，要解决房价预测问题，我们实际上是要将训练集“喂”给我们的学习算法，进而学习得到一个假设 h ，然后将我们要预测的房屋的尺寸作为输入变量输入给 h ，预测出该房屋的交易价格作为输出变量输出为结果。那么，对于我们的房价预测问题，我们该如何表达 h ？

一种可能的表达方式为: $h_{\theta} = \theta_0 + \theta_1 x$, 因为只含有一个特征/输入变量, 因此这样的问题叫作单变量线性回归问题。

2.2 代价函数

参考视频: [2 - 2 - Cost Function \(8 min\).mkv](#)

如图:

Training Set	Size in feet ² (x)	Price (\$ in 1000's) (y)
	2104	460
	1416	232
	1534	315
	852	178
...

$m = 47$

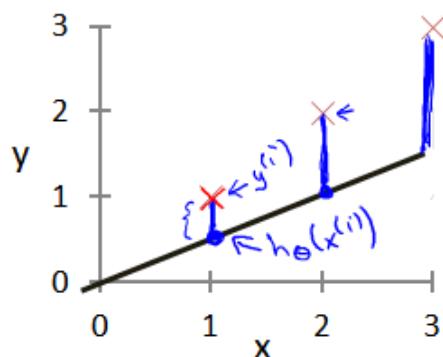
Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

θ_i 's: Parameters

How to choose θ_i 's?

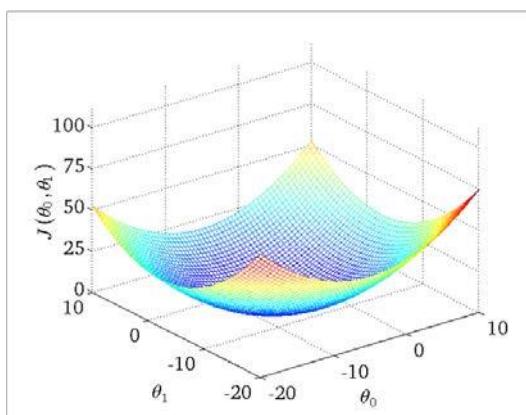
我们现在要做的便是为我们的模型选择合适的参数 (parameters) θ_0 和 θ_1 , 在房价问题这个例子中便是直线的斜率和在 y 轴上的截距。

我们选择的参数决定了我们得到的直线相对于我们的训练集的准确程度 模型所预测的值与训练集中实际值之间的差距 (下图中蓝线所指) 就是建模误差 (modeling error)。



我们的目标便是选择出可以使得建模误差的平方和能够最小的模型参数。即使得代价函数 $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$ 最小。

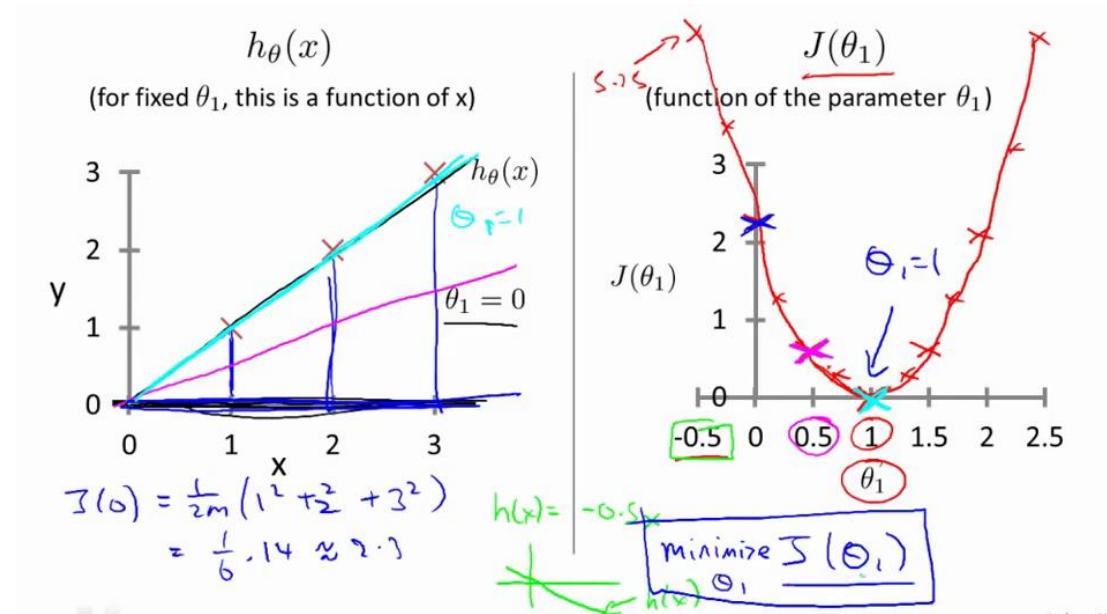
我们绘制一个等高线图, 三个坐标分别为 θ_0 和 θ_1 和 $J(\theta_0, \theta_1)$:



则可以看出在三维空间中存在一个使得 $J(\theta_0, \theta_1)$ 最小的点。

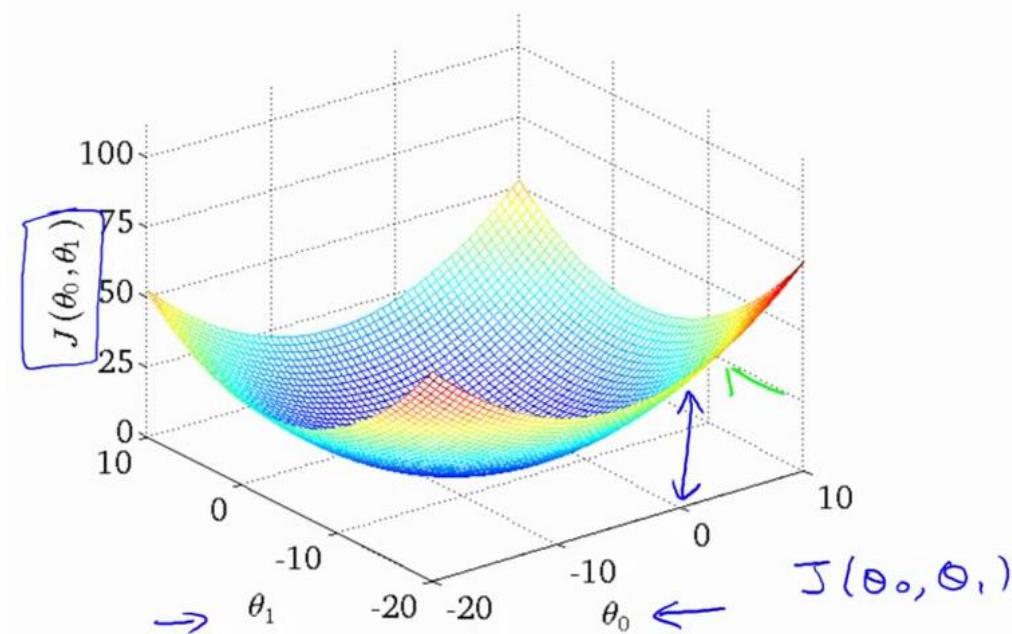
2.3 代价函数的直观理解 I

参考视频: [2 - 3 - Cost Function - Intuition I \(11 min\).mkv](#)

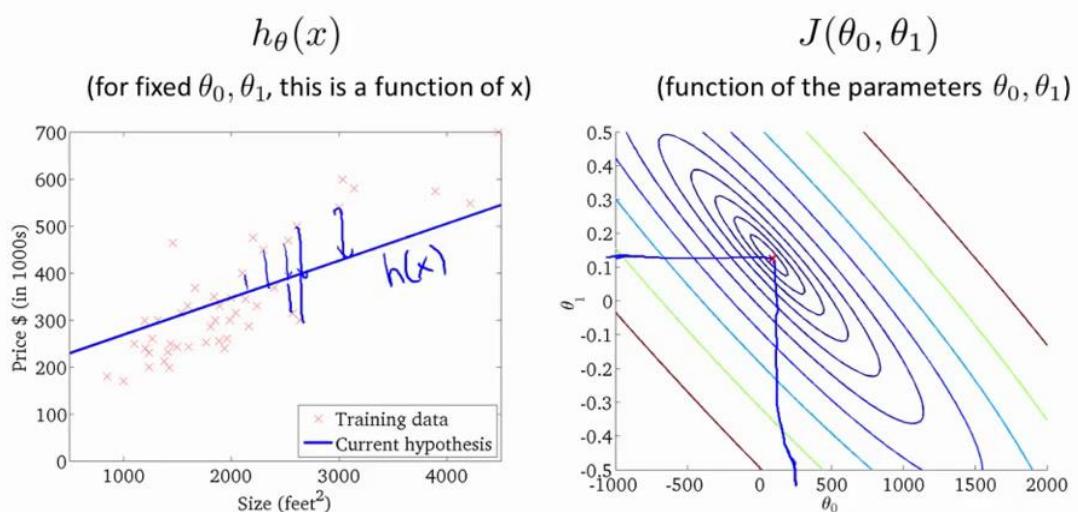


2.4 代价函数的直观理解 II

参考视频: [2 - 4 - Cost Function - Intuition II \(9 min\).mkv](#)



代价函数的样子

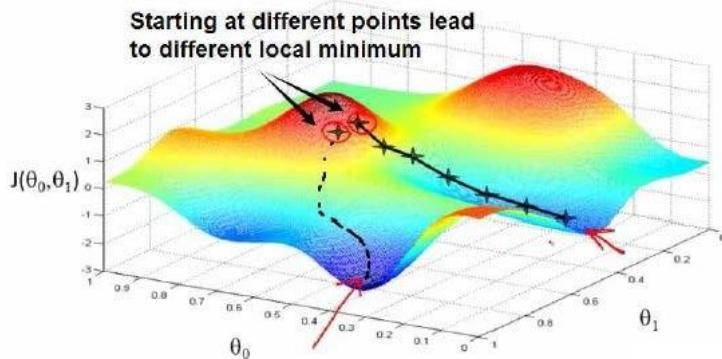


2.5 梯度下降

参考视频: [2 - 5 - Gradient Descent \(11 min\).mkv](#)

梯度下降是一个用来求函数最小值的算法，我们将使用梯度下降算法来求出代价函数 $J(\theta_0, \theta_1)$ 的最小值。

梯度下降背后的思想是：开始时我们随机选择一个参数的组合 $(\theta_0, \theta_1, \dots, \theta_n)$ ，计算代价函数，然后我们寻找下一个能让代价函数值下降最多的参数组合。我们持续这么做直到到到一个局部最小值 (local minimum)，因为我们并没有尝试完所有的参数组合，所以不能确定我们得到的局部最小值是否便是全局最小值 (global minimum)，选择不同的初始参数组合，可能会找到不同的局部最小值。



批量梯度下降 (batch gradient descent) 算法的公式为：

```
repeat until convergence {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$    (for  $j = 0$  and  $j = 1$ )
}
```

其中 α 是学习率 (learning rate)，它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大，在批量梯度下降中，我们每一次都同时让所有的参数减去学习速率乘以代价函数的导数。

2.6 梯度下降的直观理解

参考视频: [2 - 6 - Gradient Descent Intuition \(12 min\).mkv](#)

梯度下降算法如下图:

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

描述: 对 θ 赋值, 使得 $J(\theta)$ 按梯度下降最快方向进行, 一直迭代下去, 最终得到局部最小值。其中 α 是学习率 (learning rate), 它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大。

2.7 梯度下降的线性回归

参考视频: [2 - 7 - GradientDescentForLinearRegression \(6 min\).mkv](#)

梯度下降算法和线性回归算法比较如图:

Gradient descent algorithm

```
repeat until convergence {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 
    (for  $j = 1$  and  $j = 0$ )
}
```

Linear Regression Model

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

对我们之前的线性回归问题运用梯度下降法, 关键在于求出代价函数的导数, 即:

$$\frac{\partial}{\partial \theta_j} J(\theta_0 \theta_1) = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$j=0$ 时:
$$\frac{\partial}{\partial \theta_0} J(\theta_0 \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$$

$j=1$ 时:
$$\frac{\partial}{\partial \theta_1} J(\theta_0 \theta_1) = \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

则算法改写成:

```
Repeat {
     $\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})$ 
     $\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$ 
}
```

2.8 接下来的内容

参考视频: [2 - 8 - What 's Next \(6 min\).mkv](#)

在接下来的一组视频中，我会对线性代数进行一个快速的复习回顾。如果你从来没有接触过向量和矩阵，那么这课件上所有的一切对你来说都是新知识，或者你之前对线性代数有所了解，但由于隔得久了，对其有所遗忘，那就请学习接下来的一组视频，我会快速地回顾你将用到的线性代数知识。

通过它们，你可以实现和使用更强大的线性回归模型。事实上，线性代数不仅仅在线性回归中应用广泛，它其中的矩阵和向量将有助于帮助我们实现之后更多的机器学习模型，并在计算上更有效率。正是因为这些矩阵和向量提供了一种有效的方式来组织大量的数据，特别是当我们处理巨大的训练集时，如果你不熟悉线性代数，如果你觉得线性代数看上去是一个复杂、可怕的概念，特别是对于之前从未接触过它的人，不必担心，事实上，为了实现机器学习算法，我们只需要一些非常非常基础的线性代数知识。通过接下来几个视频，你可以很快地学会所有你需要了解的线性代数知识。具体来说，为了帮助你判断是否有需要学习接下来的一组视频，我会讨论什么是矩阵和向量，谈谈如何加、减、乘矩阵和向量，讨论逆矩阵和转置矩阵的概念。

如果你十分熟悉这些概念，那么你完全可以跳过这组关于线性代数的选修视频，但是如果你对这些概念仍有些许的不确定，不确定这些数字或这些矩阵的意思，那么请看一看下一组的视频，它会很快地教你一些你需要知道的线性代数的知识，便于之后编写机器学习算法和处理大量数据。

三、线性代数回顾(Linear Algebra Review)

3.1 矩阵和向量

参考视频: [3 - 1 - Matrices and Vectors \(9 min\).mkv](#)

如图: 这个是 4×2 矩阵, 即 4 行 2 列, 如 m 为行, n 为列, 那么 $m \times n$ 即 4×2

$$\rightarrow \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

↓ ↑ ↑
 4×2 matrix
 $\rightarrow \boxed{\mathbb{R}^{4 \times 2}}$

矩阵的维数即行数 \times 列数

矩阵元素 (矩阵项):

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

A_{ij} 指第 i 行, 第 j 列的元素。

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

向量是一种特殊的矩阵, 讲义中的向量一般都是列向量, 如:

向量 (4×1)。

如下图为 1 索引向量和 0 索引向量, 左图为 1 索引向量, 右图为 0 索引向量, 一般我们用 1 索引向量。

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

3.2 加法和标量乘法

参考视频: [3 - 2 - Addition and Scalar Multiplication \(7 min\).mkv](#)

矩阵的加法: 行列数相等的可以加。

例:
$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

矩阵的乘法: 每个元素都要乘

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} \times 3$$

组合算法也类似。

3.3 矩阵向量乘法

参考视频: [3 - 3 - Matrix Vector Multiplication \(14 min\).mkv](#)

矩阵和向量的乘法如图: $m \times n$ 的矩阵乘以 $n \times 1$ 的向量, 得到的是 $m \times 1$ 的向量

$$\begin{array}{c}
 \begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix} \\
 \text{3x2} \quad \text{2x1} \quad \text{3x1 matrix}
 \end{array}$$

$1 \times 1 + 3 \times 5 = 16$
 $4 \times 1 + 0 \times 5 = 4$
 $2 \times 1 + 1 \times 5 = 7$

算法举例:

$$\begin{array}{c}
 \begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix} \\
 \text{3x4} \quad \text{4x1} \quad \text{3x1}
 \end{array}$$

$1 \times 1 + 2 \times 3 + 1 \times 2 + 5 \times 1 = 14$
 $0 \times 1 + 3 \times 3 + 0 \times 2 + 4 \times 1 = 13$
 $-1 \times 1 + (-2) \times 3 + 0 \times 2 + 0 \times 1 = -7$

3.4 矩阵乘法

参考视频: [3 - 4 - Matrix Matrix Multiplication \(11 min\).mkv](#)

矩阵乘法:

$m \times n$ 矩阵乘以 $n \times o$ 矩阵, 变成 $m \times o$ 矩阵。

如果说这样不好理解的话就举一个例子来说明一下, 比如说现在有两个矩阵 A 和 B, 那么它们的乘积就可以表示为图中所示的形式。

$$\text{例如: } A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix} =$$

$$\begin{pmatrix} a_{11}b_{11} + a_{12}b_{12} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{12} & a_{21}b_{12} + a_{22}b_{22} \\ a_{31}b_{11} + a_{32}b_{12} & a_{31}b_{12} + a_{32}b_{22} \end{pmatrix}$$



3.5 矩阵乘法的性质

参考视频: [3 - 5 - Matrix Multiplication Properties \(9 min\).mkv](#)

矩阵乘法的性质:

矩阵的乘法不满足交换律: $A \times B \neq B \times A$

矩阵的乘法满足结合律。即: $A \times (B \times C) = (A \times B) \times C$

单位矩阵: 在矩阵的乘法中, 有一种矩阵起着特殊的作用, 如同数的乘法中的 1, 我们称这种矩阵为单位矩阵。它是个方阵, 一般用 I 或者 E 表示, 本讲义都用 I 代表单位矩阵, 从左上角到右下角的对角线 (称为主对角线) 上的元素均为 1 以外全都为 0。如:

$$\begin{array}{ccc} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ 2 \times 2 & 3 \times 3 & 4 \times 4 \end{array}$$

对于单位矩阵, 有 $AI=IA=A$

3.6 逆、转置

参考视频: [3 - 6 - Inverse and Transpose \(11 min\).mkv](#)

矩阵的逆: 如矩阵 A 是一个 $m \times m$ 矩阵 (方阵), 如果有逆矩阵, 则:

$$AA^{-1} = A^{-1}A = I$$

我们一般在 OCTAVE 或者 MATLAB 中进行计算矩阵的逆矩阵。

矩阵的转置: 设 A 为 $m \times n$ 阶矩阵 (即 m 行 n 列), 第 i 行 j 列的元素是 $a(i,j)$, 即:

$$A=a(i,j)$$

定义 A 的转置为这样一个 $n \times m$ 阶矩阵 B , 满足 $B=a(j,i)$, 即 $b(i,j)=a(j,i)$ (B 的第 i 行第 j 列元素是 A 的第 j 行第 i 列元素), 记 $A^T=B$ 。(有些书记为 $A'=B$)

直观来看, 将 A 的所有元素绕着一条从第 1 行第 1 列元素出发的右下方 45 度的射线作镜面反转, 即得到 A 的转置。

$$\begin{vmatrix} a & b \\ c & d \\ e & f \end{vmatrix}^T = \begin{vmatrix} a & c & e \\ b & d & f \end{vmatrix}$$

例:

矩阵的转置基本性质:

$$(A \pm B)^T = A^T \pm B^T$$

$$(A \times B)^T = B^T \times A^T$$

$$(A^T)^T = A$$

$$(KA)^T = KA^T$$

matlab 中矩阵转置:

直接打一撇, $x=y^T$ 。

第 2 周

四、多变量线性回归(Linear Regression with Multiple Variables)

4.1 多维特征

参考视频: [4 - 1 - Multiple Features \(8 min\).mkv](#)

目前为止, 我们探讨了单变量/特征的回归模型, 现在我们对房价模型增加更多的特征, 例如房间数楼层等, 构成一个含有多个变量的模型, 模型中的特征为 (x_1, x_2, \dots, x_n) 。

Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

增添更多特征后, 我们引入一系列新的注释:

n 代表特征的数量

$x^{(i)}$ 代表第 i 个训练实例, 是特征矩阵中的第 i 行, 是一个向量 (vector)。

比方说, 上图的 $\mathcal{X}^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$,

$x_j^{(i)}$ 代表特征矩阵中第 i 行的第 j 个特征, 也就是第 i 个训练实例的第 j 个特征。

如上图的 $\mathcal{X}_3^{(2)} = 2 \quad \mathcal{X}_3^{(2)} = 2$

支持多变量的假设 h 表示为:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

这个公式中有 $n+1$ 个参数和 n 个变量, 为了使得公式能够简化一些, 引入 $x_0=1$, 则公式转化为:

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

此时模型中的参数是一个 $n+1$ 维的向量，任何一个训练实例也都是 $n+1$ 维的向量，特征矩阵 X 的维度是 $m \times n+1$ 。因此公式可以简化为： $h_{\theta}(x) = \theta^T X$ ，其中上标 T 代表矩阵转置。

4.2 多变量梯度下降

参考视频: [4 - 2 - Gradient Descent for Multiple Variables \(5 min\).mkv](#)

与单变量线性回归类似，在多变量线性回归中，我们也构建一个代价函数，则这个代价函数是所有建模误差的平方和，即：

$$J(\theta_0, \theta_1, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

其中： $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

我们的目标和单变量线性回归问题中一样，是要找出使得代价函数最小的一系列参数。
多变量线性回归的批量梯度下降算法为：

```
Repeat {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$ 
}
```

即：

```
Repeat {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$ 
}
```

求导数后得到：

```
Repeat {
     $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)})$ 
    ( simultaneously update  $\theta_j$ 
    for  $j=0,1,\dots,n$  )
}
```

当 $n >= 1$ 时，

$$\begin{aligned}\theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}\end{aligned}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

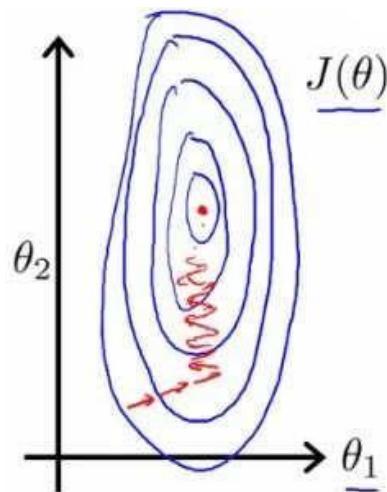
我们开始随机选择一系列的参数值，计算所有的预测结果后，再给所有的参数一个新的值，如此循环直到收敛。

4.3 梯度下降法实践 1-特征缩放

参考视频: [4 - 3 - Gradient Descent in Practice I - Feature Scaling \(9 min\).mkv](#)

在我们面对多维特征问题的时候，我们要保证这些特征都具有相近的尺度，这将帮助梯度下降算法更快地收敛。

以房价问题为例，假设我们使用两个特征，房屋的尺寸和房间的数量，尺寸的值为 0-2000 平方英尺，而房间数量的值则是 0-5，以两个参数分别为横纵坐标，绘制代价函数的等高线图能，看出图像会显得很扁，梯度下降算法需要非常多次的迭代才能收敛。



解决的方法是尝试将所有特征的尺度都尽量缩放到-1 到 1 之间。如图：

Feature Scaling

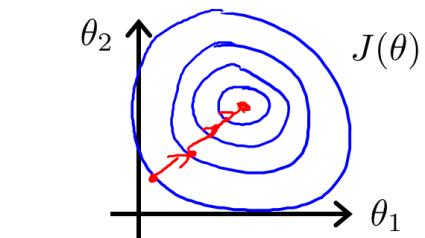
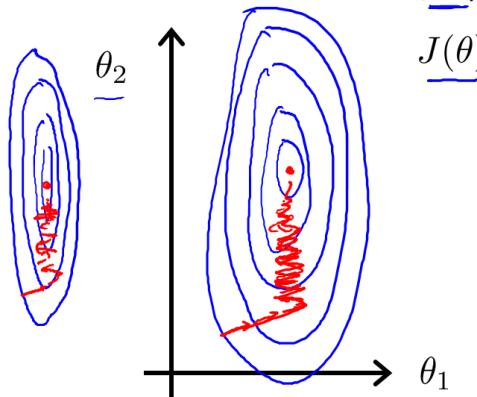
Idea: Make sure features are on a similar scale.

E.g. $x_1 = \text{size (0-2000 feet}^2)$ ←

$$\rightarrow x_1 = \frac{\text{size (feet}^2)}{2000} \quad \checkmark$$

$x_2 = \text{number of bedrooms (1-5)}$ ←

$$\rightarrow x_2 = \frac{\text{number of bedrooms}}{5} \quad \checkmark$$



最简单的方法是令：

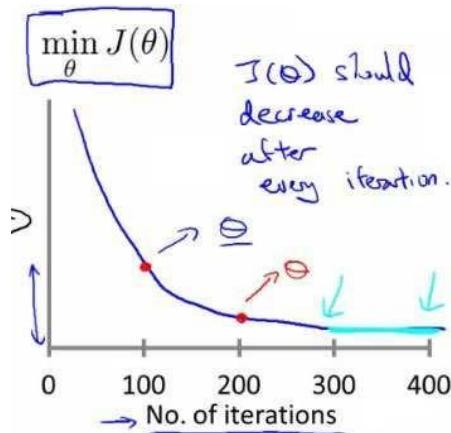
$$x_n = \frac{x_n - \mu_n}{s_n}$$

其中 μ_n 是平均值, s_n 是标准差。

4.4 梯度下降法实践 2-学习率

参考视频: [4 - 4 - Gradient Descent in Practice II - Learning Rate \(9 min\).mkv](#)

梯度下降算法收敛所需要的迭代次数根据模型的不同而不同，我们不能提前预知，我们可以绘制迭代次数和代价函数的图表来观测算法在何时趋于收敛。



也有一些自动测试是否收敛的方法，例如将代价函数的变化值与某个阀值（例如 0.001）进行比较，但通常看上面这样的图表更好。

梯度下降算法的每次迭代受到学习率的影响，如果学习率 α 过小，则达到收敛所需的迭代次数会非常高；如果学习率 α 过大，每次迭代可能不会减小代价函数，可能会越过局部最小值导致无法收敛。

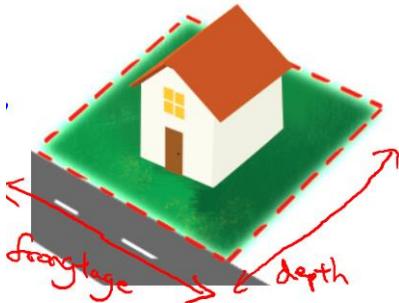
通常可以考虑尝试些学习率：

$$\alpha = 0.01, 0.03, 0.1, 0.3, 1, 3, 10$$

4.5 特征和多项式回归

参考视频: [4 - 5 - Features and Polynomial Regression \(8 min\).mkv](#)

如房价预测问题,



$$h_{\theta}(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$$

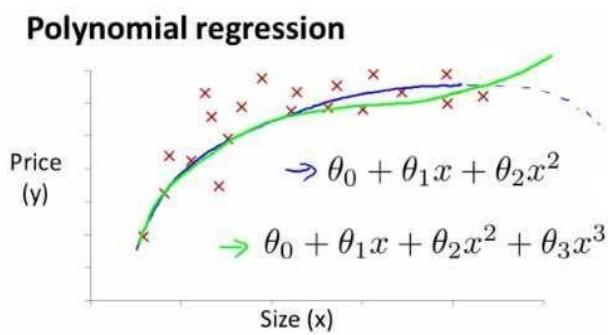
$x_1 = \text{frontage}$ (临街宽度), $x_2 = \text{depth}$ (纵向深度), $x = \text{frontage} * \text{depth} = \text{area}$ (面积), 则

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

线性回归并不适用于所有数据, 有时我们需要曲线来适应我们的数据, 比如一个二次方模型:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$$

或者三次方模型: $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$



通常我们需要先观察数据然后再决定准备尝试怎样的模型。另外, 我们可以令:

$$x_2 = x_2^2$$

$$x_3 = x_3^3$$

从而将模型转化为线性回归模型。

根据函数图形特性, 我们还可以使:

$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$$

或者：

$$h_{\theta}(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

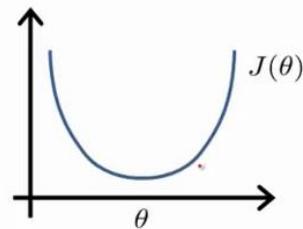
注：如果我们采用多项式回归模型，在运行梯度下降算法前，特征缩放非常有必要。

4.6 正规方程

参考视频: [4 - 6 - Normal Equation \(16 min\).mkv](#)

到目前为止, 我们都在使用梯度下降算法, 但是对于某些线性回归问题, 正规方程方法是更好的解决方案。如:

Intuition: If 1D ($\theta \in \mathbb{R}$)
 $\rightarrow J(\theta) = a\theta^2 + b\theta + c$



正规方程是通过求解下面的方程来找出使得代价函数最小的参数的:

$$\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$$

假设我们的训练集特征矩阵为 X (包含了 $x_0=1$) 并且我们的训练集结果为向量 y , 则

利用正规方程解出向量 $\theta = (X^T X)^{-1} X^T y$

上标 T 代表矩阵转置, 上标-1 代表矩阵的逆。设矩阵 $A=X^T X$, 则: $(X^T X)^{-1}=A^{-1}$

以下表示数据为例:

Examples: $m = 4$.

\downarrow x_0	Size (feet ²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price (\$1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

即:

X(0)	X(1)	X(2)	X(3)	X(4)	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

运用正规方程方法求解参数：

$$\left(\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}^{-1} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \right)$$

在 Octave 中，正规方程写作：

`pinv(X'*X)*X'*y`

注：对于那些不可逆的矩阵（通常是因为特征之间不独立，如同时包含英尺为单位的尺寸和米为单位的尺寸两个特征，也有可能是特征数量大于训练集的数量），正规方程方法是不能用的。

梯度下降与正规方程的比较：

梯度下降	正规方程
需要选择学习率 α	不需要
需要多次迭代	一次运算得出
当特征数量 n 大时也能较好适用	需要计算 $(X^T X)^{-1}$ 如果特征数量 n 较大则运算代价大，因为矩阵逆的计算时间复杂度为 $O(n^3)$ ，通常来说当 n 小于 10000 时还是可以接受的
适用于各种类型的模型	只适用于线性模型，不适合逻辑回归模型等其他模型

总结一下，只要特征变量的数目并不大，标准方程是一个很好的计算参数 θ 的替代方法。

具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

随着我们要讲的学习算法越来越复杂，例如，当我们讲到分类算法，像逻辑回归算法，我们会看到，实际上对于那些算法，并不能使用标准方程法。对于那些更复杂的学习算法，

我们将不得不仍然使用梯度下降法。因此，梯度下降法是一个非常有用的算法，可以用在有大量特征变量的线性回归问题。或者我们以后在课程中，会讲到的一些其他的算法，因为标准方程法不适合或者不能用在它们上。但对于这个特定的线性回归模型，标准方程法是一个比梯度下降法更快的替代算法。所以，根据具体的问题，以及你的特征变量的数量，这两种算法都是值得学习的。

4.7 正规方程及不可逆性（可选）

参考视频: [4 - 7 - Normal Equation Noninvertibility \(Optional\) \(6 min\).mkv](#)

$X^T X$ 是不可逆的。

五、Octave 教程(Octave Tutorial)

5.1 基本操作

参考视频: [5 - 1 - Basic Operations \(14 min\).mkv](#)

第五章 Octave 教程略, 貌似国内学生喜欢用收费的 matlab, matlab 功能要比 Octave 强大的多, 网上有各种 D 版可以下载。

这次机器学习课的作业也是用 matlab 的。

5.2 移动数据

参考视频: [5 - 2 - Moving Data Around \(16 min\).mkv](#)

5.3 计算数据

参考视频:[5 - 3 - Computing on Data \(13 min\).mkv](#)

5.4 绘图数据

参考视频: [5 - 4 - Plotting Data \(10 min\).mkv](#)

5.5 控制语句: **for**, **while**, **if** 语句

参考视频: [5 - 5 - Control Statements for, while, if statements \(13 min\).mkv](#)

5.6 矢量化

参考视频: [5 - 6 - Vectorization \(14 min\).mkv](#)

5.7 工作和提交的编程练习

参考视频: [5 - 7 - Working on and Submitting Programming Exercises \(4 min\).mkv](#)

第3周

六、逻辑回归(Logistic Regression)

6.1 分类问题

参考文档: [6 - 1 - Classification \(8 min\).mkv](#)

在分类问题中，我们尝试预测的是结果是否属于某一个类（例如正确或错误）。分类问题的例子有：判断一封电子邮件是否是垃圾邮件；判断一次金融交易是否是欺诈等等。

我们从二元的分类问题开始讨论。

我们将因变量(dependant variable)可能属于的两个类分别称为负向类 (negative class) 和正向类 (positive class)，则因变量

$$y \in \{0,1\}$$

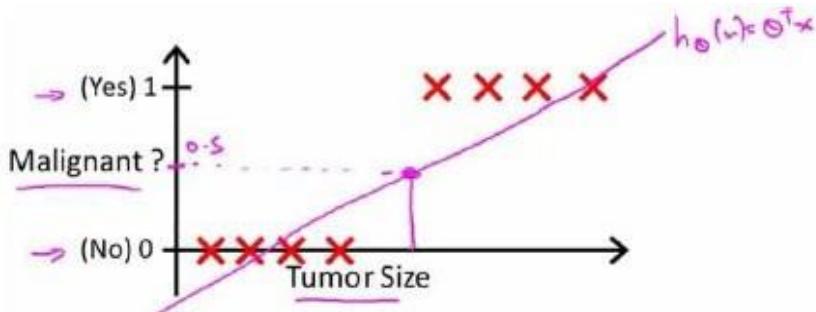
其中 0 表示负向类，1 表示正向类。

参考文档: [6 - 1 - Classification \(8 min\).mkv](#)

6.2 假说表示

参考视频: [6 - 2 - Hypothesis Representation \(7 min\).mkv](#)

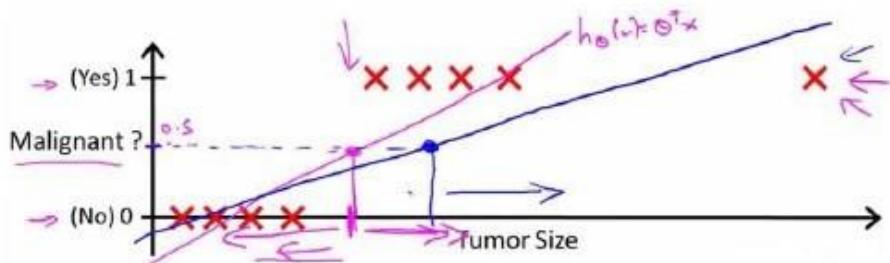
回顾一开始提到的乳腺癌分类问题, 我们可以用线性回归的方法求出适合数据的一条直线:



根据线性回归模型我们只能预测连续的值, 然而对于分类问题, 我们需要输出 0 或 1, 我们可以预测:

当 h_{θ} 大于等于 0.5 时, 预测 $y=1$

当 h_{θ} 小于 0.5 时, 预测 $y=0$ 对于上图所示的数据, 这样的一个线性模型似乎能很好地完成分类任务。假使我们又观测到一个非常大尺寸的恶性肿瘤, 将其作为实例加入到我们的训练集中来, 这将使得我们获得一条新的直线。



这时, 再使用 0.5 作为阀值来预测肿瘤是良性还是恶性便不合适了。可以看出, 线性回归模型, 因为其预测的值可以超越[0,1]的范围, 并不适合解决这样的问题。

我们引入一个新的模型, 逻辑回归, 该模型的输出变量范围始终在 0 和 1 之间。逻辑回归模型的假设是: $h_{\theta}(x) = g(\theta^T x)$

其中:

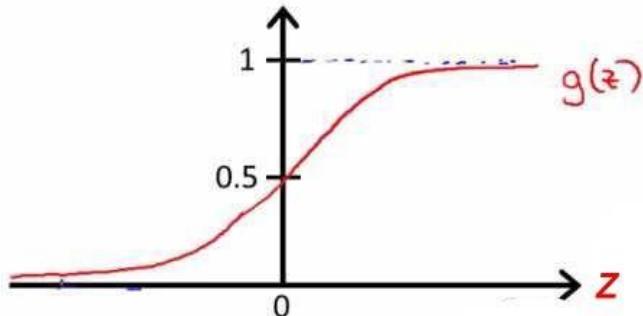
x 代表特征向量

g 代表逻辑函数(logistic function)是一个常用的逻辑函数为 S 形函数(Sigmoid function),

$$g(z) = \frac{1}{1+e^{-z}}$$

公式为：

该函数的图像为：



合起来，我们得到逻辑回归模型的假设：

对模型的理解： $h_{\theta}(x) = \frac{1}{1+e^{-\theta^T x}}$

$h_{\theta}(x)$ 的作用是，对于给定的输入变量，根据选择的参数计算输出变量=1 的可能性 (estimated probability) 即 $h_{\theta}(x) = P(y=1|x; \theta)$

例如，如果对于给定的 x ，通过已经确定的参数计算得出 $h_{\theta}(x)=0.7$ ，则表示有 70% 的几率 y 为正向类，相应地 y 为负向类的几率为 $1-0.7=0.3$ 。

6.3 判定边界

参考视频: [6 - 3 - Decision Boundary \(15 min\).mkv](#)

在逻辑回归中，我们预测：

当 h_{θ} 大于等于 0.5 时，预测 $y=1$

当 h_{θ} 小于 0.5 时，预测 $y=0$

根据上面绘制出的 S 形函数图像，我们知道当

$z=0$ 时 $g(z)=0.5$

$z>0$ 时 $g(z)>0.5$

$z<0$ 时 $g(z)<0.5$

又 $z=\theta^T X$ ，即：

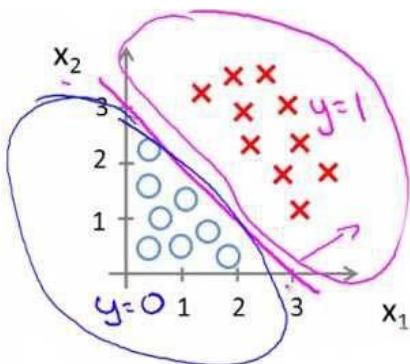
$\theta^T X$ 大于等于 0 时，预测 $y=1$

$\theta^T X$ 小于 0 时，预测 $y=0$

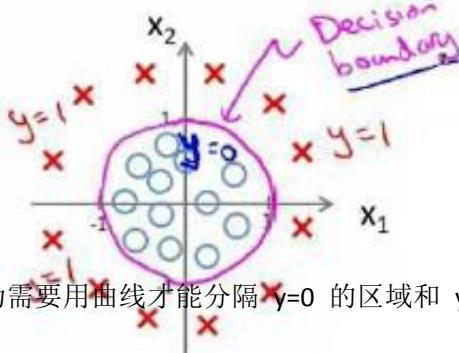
现在假设我们有一个模型： $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$ 并且参数 θ 是向量 [-3 1

1]。则当 $-3+x_1+x_2$ 大于等于 0，即 x_1+x_2 大于等于 3 时，模型将预测 $y=1$ 。

我们可以绘制直线 $x_1+x_2=3$ ，这条线便是我们模型的分界线，将预测为 1 的区域和预测为 0 的区域分隔开。



假使我们的数据呈现这样的分布情况，怎样的模型才能适合呢？



因为需要用曲线才能分隔 $y=0$ 的区域和 $y=1$ 的区域，我们需要二次方特征： 假设参

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

是[-1 0 0 1 1]，则我们得到的判定边界恰好是圆点在原点且半径为 1 的圆形。

我们可以用非常复杂的模型来适应非常复杂形状的判定边界。

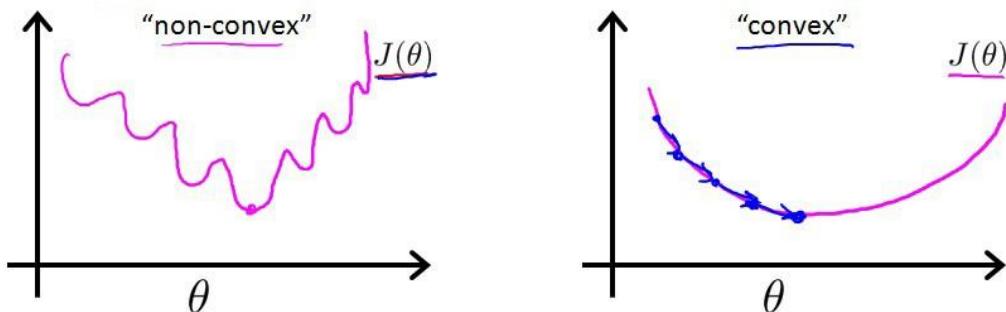
6.4 代价函数

参考视频: [6 - 4 - Cost Function \(11 min\).mkv](#)

对于线性回归模型, 我们定义的代价函数是所有模型误差的平方和。理论上来说, 我们

$$h_{\theta}(x) = \frac{1}{1+e^{-\theta^T x}}$$

也可以对逻辑回归模型沿用这个定义, 但是问题在于, 当我们将这样定义了的代价函数中时, 我们得到的代价函数将是一个非凸函数 (non-convex function)。



这意味着我们的代价函数有许多局部最小值, 这将影响梯度下降算法寻找全局最小值。

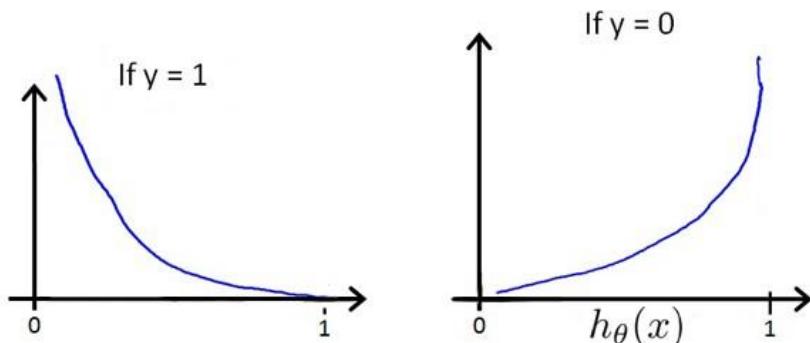
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

因此我们重新定义逻辑回归的代价函数为:

其中

$$\text{Cost}(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)) & \text{if } y = 1 \\ -\log(1 - h_{\theta}(x)) & \text{if } y = 0 \end{cases}$$

$h_{\theta}(x)$ 与 $\text{Cost}(h_{\theta}(x), y)$ 之间的关系如下图所示:



这样构建的 $\text{Cost}(h_\theta(x), y)$ 函数的特点是：当实际的 $y=1$ 且 h_θ 也为 1 时误差为 0，当 $y=1$ 但 h_θ 不为 1 时误差随着 h_θ 的变小而变大；当实际的 $y=0$ 且 h_θ 也为 0 时代价为 0，当 $y=0$ 但 h_θ 不为 0 时误差随着 h_θ 的变大而变大。

将构建的 $\text{Cost}(h_\theta(x), y)$ 简化如下：

$$\text{Cost}(h_\theta(x), y) = -y \times \log(h_\theta(x)) - (1-y) \times \log(1-h_\theta(x))$$

带入代价函数得到：

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right]$$

在得到这样一个代价函数以后，我们便可以用梯度下降算法来求得能使代价函数最小的参数了。算法为：

Repeat

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

} (simultaneously update all)

求导后得到：

Repeat

$$\theta_j := \theta_j - \alpha \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

} (simultaneously update all)

注：虽然得到的梯度下降算法表面上看上去与线性回归的梯度下降算法一样，但是这里的 $h_\theta(x)=g(\theta^T x)$ 与线性回归中不同，所以实际上是不一样的。另外，在运行梯度下降算法之前，进行特征缩放依旧是非常必要的。

一些梯度下降算法之外的选择：除了梯度下降算法以外，还有一些常被用来令代价函数最小的算法，这些算法更加复杂和优越，而且通常不需要人工选择学习率，通常比梯度下降算法要更加快速。这些算法有：共轭梯度(Conjugate Gradient)，局部优化法(Broyden fletcher goldfarb shann,BFGS)和有限内存局部优化法(LBFGS) `fminunc` 是 `matlab` 和 `octave` 中都带的一个最小值优化函数，使用时我们需要提供代价函数和每个参数的求导，下面是 `octave` 中使

用 fminunc 函数的代码示例：

```
function [jVal, gradient] = costFunction(theta)

    jVal = [...code to compute
        J(theta)...];

    gradient = [...code to compute derivative of J(theta)...];

end

options = optimset('GradObj', 'on', 'MaxIter', '100');

initialTheta = zeros(2,1);

[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
```

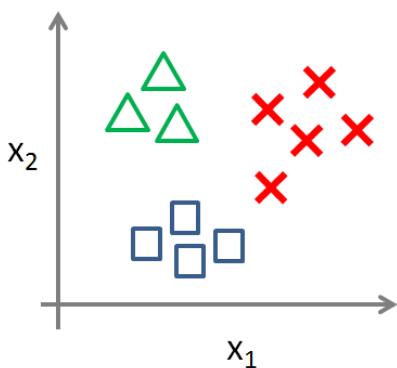
6.5 简化的成本函数和梯度下降

参考视频: [6 - 5 - Simplified Cost Function and Gradient Descent \(10 min\).mkv](#)

多类分类问题中, 我们的训练集中有多个类 (>2), 我们无法仅仅用一个二元变量 (0 或 1) 来做判断依据。例如我们要预测天气情况分四种类型: 晴天、多云、下雨或下雪。

下面是一个多类分类问题可能的情况:

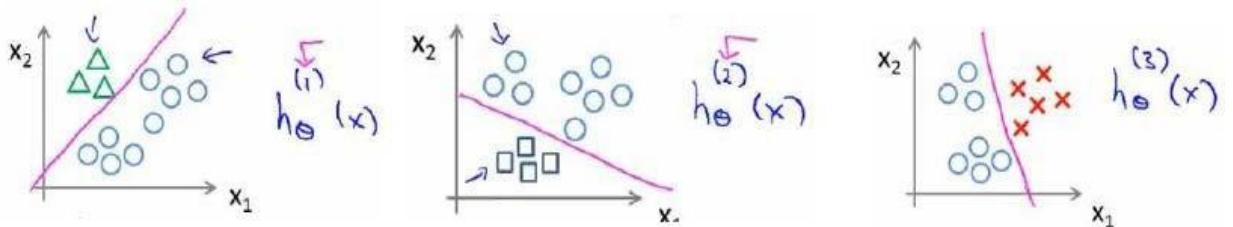
Multi-class classification:



一种解决这类问题的途径是采用一对多 (One-vs-All) 方法。在一对多方法中, 我们将多类分类问题转化成二元分类问题。

为了能实现这样的转变, 我们将多个类中的一个类标记为正向类 ($y=1$), 然后将其他所有类都标记为负向类, 这个模型记作 $h_{\theta}^{(1)}(x)$ 。接着, 类似地我们选择另一个类标记为正向类 ($y=2$), 再将其它类都标记为负向类, 将这个模型记作 $h_{\theta}^{(2)}(x)$, 依此类推。

最后我们得到一系列的模型简记为: $h_{\theta}^{(i)}(x)=p(y=i|x;\theta)$ 其中 $i=(1,2,3,\dots,k)$



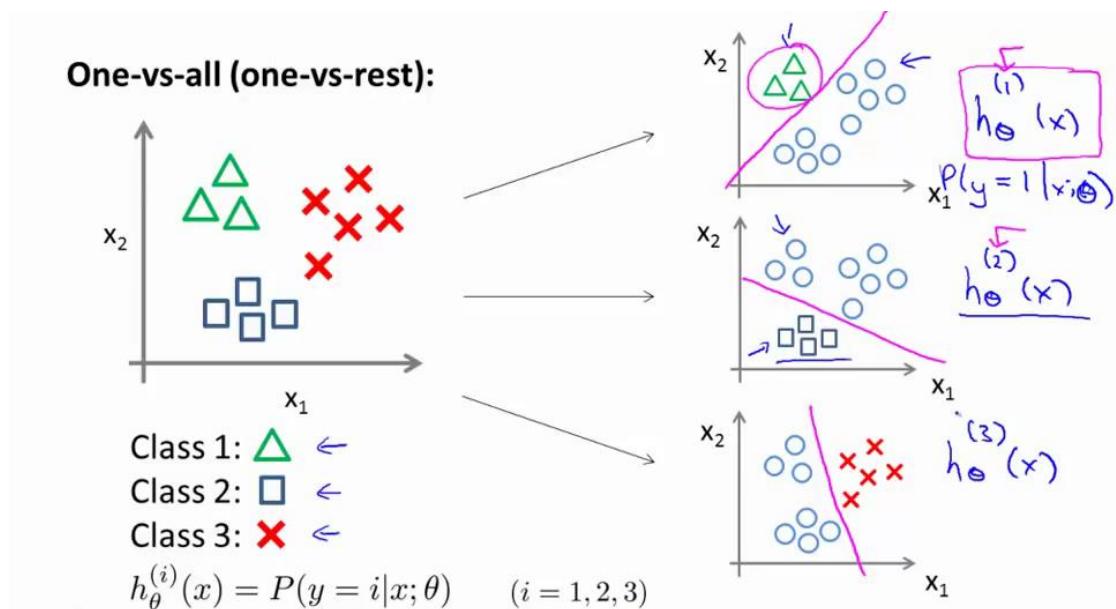
最后, 在我们需要做预测时, 我们将所有的分类机都运行一遍, 然后对每一个输入变量, 都选择最高可能性的输出变量。

6.6 高级优化

参考视频: [6 - 6 - Advanced Optimization \(14 min\).mkv](#)

6.7 多类分类：一个对所有

参考视频: [6 - 7 - Multiclass Classification One-vs-all \(6 min\).mkv](#)



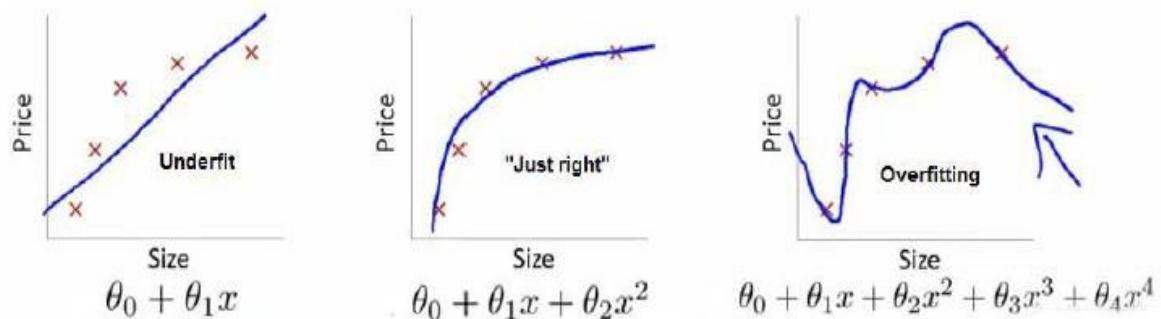
七、正则化(Regularization)

7.1 过拟合的问题

参考视频: [7 - 1 - The Problem of Overfitting \(10 min\).mkv](#)

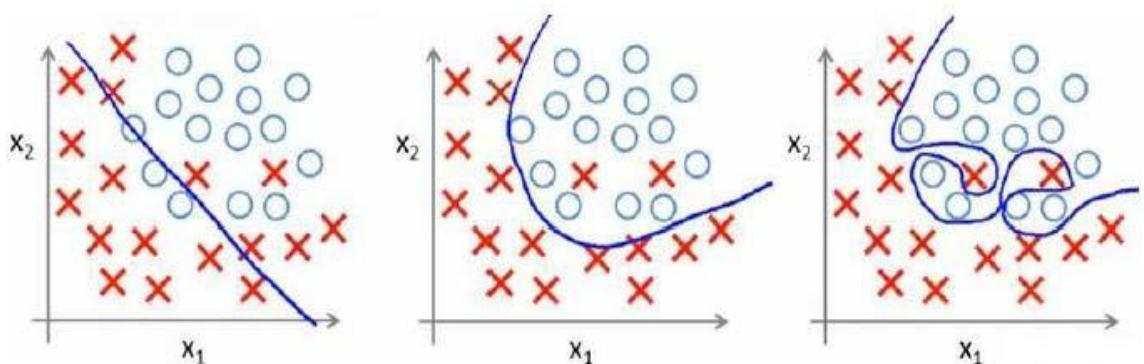
如果我们有非常多的特征, 我们通过学习得到的假设可能能够非常好地适应训练集 (代价函数可能几乎为 0), 但是可能会不能推广到新的数据。

下图是一个回归问题的例子:



第一个模型是一个线性模型, 欠拟合, 不能很好地适应我们的训练集; 第三个模型是一个四次方的模型, 过于强调拟合原始数据, 而丢失了算法的本质: 预测新数据。我们可以看出, 若给出一个新的值使之预测, 它将表现的很差, 是过拟合, 虽然能非常好地适应我们的训练集但在新输入变量进行预测时可能会效果不好; 而中间的模型似乎最合适。

分类问题中也存在这样的问题:



就以多项式理解, x 的次数越高, 拟合的越好, 但相应的预测的能力就可能变差。

问题是, 如果我们发现了过拟合问题, 应该如何处理?

1. 丢弃一些不能帮助我们正确预测的特征。可以是手工选择保留哪些特征 或者使用一些模型选择的算法来帮忙（例如 PCA）
2. 正则化。 保留所有的特征，但是减少参数的大小（magnitude）。

7.2 代价函数

参考视频: [7 - 2 - Cost Function \(10 min\).mkv](#)

上面的回归问题中如果我们的模型是:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

我们可以从之前的事例中看出, 正是那些高次项导致了过拟合的产生, 所以如果我们能让这些高次项的系数接近于 0 的话, 我们就能很好的拟合了。

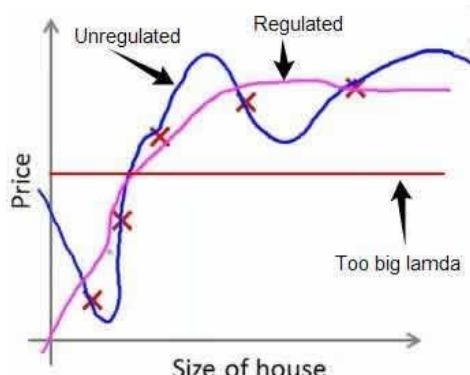
所以我们要做的就是在一定程度上减小这些参数 θ 的值, 这就是正则化的基本方法。我们决定要减少 θ_3 和 θ_4 的大小, 我们要做的便是修改代价函数, 在其中 θ_3 和 θ_4 设置一点惩罚。这样做的话, 我们在尝试最小化代价时也需要将这个惩罚纳入考虑中, 并最终导致选择较小一些的 θ_3 和 θ_4 。修改后的代价函数如下:

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 10000\theta_4^2)$$

通过这样的代价函数选择出的 θ_3 和 θ_4 对预测结果的影响就比之前要小许多。假如我们有非常多的特征, 我们并不知道其中哪些特征我们要惩罚, 我们将对所有的特征进行惩罚, 并且让代价函数最优化的软件来选择这些惩罚的程度。这样的结果是得到了一个较为简单的能防止过拟合问题的假设:

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2) \right]$$

其中 λ 又称为正则化参数 (Regularization Parameter)。注: 根据惯例, 我们不对 θ_0 进行惩罚。经过正则化处理的模型与原模型的可能对比如下图所示:



如果选择的正则化参数 λ 过大, 则会把所有的参数都最小化了, 导致模型变成 $h_{\theta}(x) = \theta_0$

也就是上图中红色直线所示的情况，造成欠拟合。

$$\lambda \sum_{j=1}^n \theta_j^2$$

那为什么增加的一项 $\lambda \sum_{j=1}^n \theta_j^2$ 可以使 θ 的值减小呢？

因为如果我们令 λ 的值很大的话，为了使 Cost Function 尽可能的小，所有的 θ 的值（不包括 θ_0 ）都会在一定程度上减小。

但若 λ 的值太大了，那么 θ （不包括 θ_0 ）都会趋近于 0，这样我们所得到的只能是一条平行于 x 轴的直线。

所以对于正则化，我们要取一个合理的 λ 的值，这样才能更好的应用正则化。

7.3 正则化线性回归

参考视频: [7 - 3 - Regularized Linear Regression \(11 min\).mkv](#)

正则化线性回归的代价函数为:

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right]$$

如果我们要使用梯度下降发令这个代价函数最小化, 因为我们未对 θ_0 进行正则化, 所以梯度下降算法将分两种情形:

$$\begin{aligned} & \text{Repeat until convergence}\{ \\ & \quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}) \\ & \quad \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \frac{\lambda}{m} \theta_j) \\ & \quad \text{for } j=1,2,\dots,n \\ & \} \end{aligned}$$

对上面的算法中 $j=1,2,\dots,n$ 时的更新式子进行调整可得:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

可以看出, 正则化线性回归的梯度下降算法的变化在于, 每次都在原有算法更新规则的基础上令 θ 值减少了一个额外的值。

我们同样也可以利用正规方程来求解正则化线性回归模型, 方法如下所示:

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

图中的矩阵尺寸为 $(n+1) \times (n+1)$ 。

7.4 正则化的逻辑回归模型

参考视频: [7 - 4 - Regularized Logistic Regression \(9 min\).mkv](#)

同样对于逻辑回归，我们也给代价函数增加一个正则化的表达式，得到：

$$J(\theta) = - \left[\frac{1}{m} \sum_{i=1}^m (y^{(i)} \times \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \times \log(1-h_\theta(x^{(i)}))) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

要最小化该代价函数，通过求导，得出梯度下降算法为：

Repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)})$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \frac{\lambda}{m} \theta_j)$$

for $j=1, 2, \dots, n$

}

注：看上去同线性回归一样，但是知道 $h_\theta(x) = g(\theta^T x)$ ，所以与线性回归不同。

Octave 中，我们依旧可以用 fminuc 函数来求解代价函数最小化的参数，值得注意的是参数 θ_0 的更新规则与其他情况不同。

注意：

1. 虽然正则化的逻辑回归中的梯度下降和正则化的线性回归中的表达式看起来一样，但由于两者的 $h(x)$ 不同所以还是有很大差别。

2. θ_0 不参与其中的任何一个正则化。

第4周

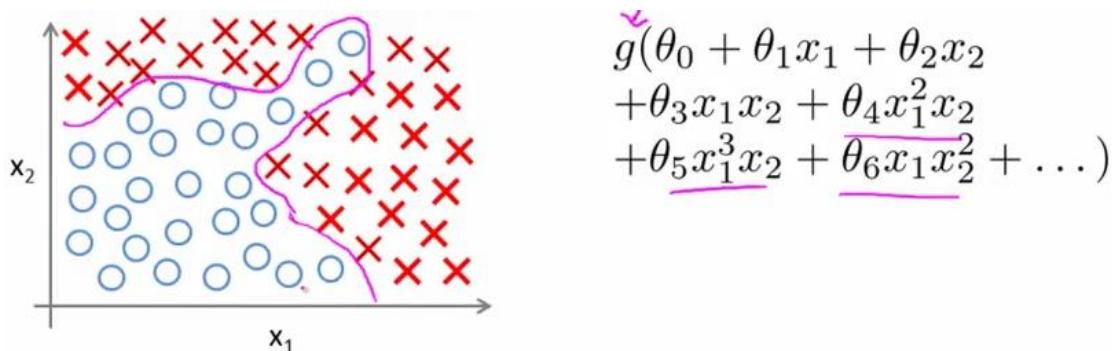
第八、神经网络：表述(Neural Networks: Representation)

8.1 非线性假设

参考视频: [8 - 1 - Non-linear Hypotheses \(10 min\).mkv](#)

我们之前学的，无论是线性回归还是逻辑回归都有这样一个缺点，即当特征太多时，计算的负荷会非常大。

下面是一个例子：

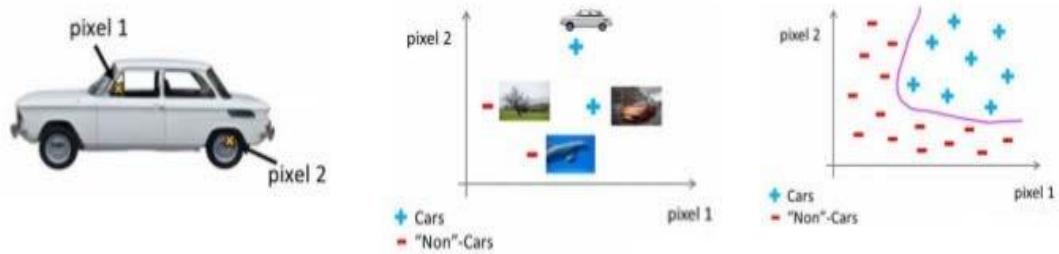


当我们使用 x_1, x_2 的多次项进行预测时，我们可以应用的很好。

之前我们已经看到过，使用非线性的多项式项，能够帮助我们建立更好的分类模型。假设我们有非常多的特征，例如大于 100 个变量，我们希望用这 100 个特征来构建一个非线性的多项式模型，结果将是数量非常惊人的特征组合，即便我们只采用两两特征的组合 ($x_1x_2 + x_1x_3 + x_1x_4 + \dots + x_2x_3 + x_2x_4 + \dots + x_{99}x_{100}$)，我们也会有接近 5000 个组合而成的特征。这对于一般的逻辑回归来说需要计算的特征太多了。

假设我们希望训练一个模型来识别视觉对象（例如识别一张图片上是否是一辆汽车），我们怎样才能这么做呢？一种方法是我们利用很多汽车的图片和很多非汽车的图片，然后利用这些图片上一个个像素的值（饱和度或亮度）来作为特征。

假如我们只选用灰度图片，每个像素则只有一个值（而非 RGB 值），我们可以选取图片上的两个不同位置上的两个像素，然后训练一个逻辑回归算法利用这两个像素的值来判断图片上是否是汽车：



假使我们采用的都是 50×50 像素的小图片，并且我们将所有的像素视为特征，则会有 2500 个特征，如果我们要进一步将两两特征组合构成一个多项式模型，则会有约 $2500^2/2$ 个（接近 3 百万个）特征。普通的逻辑回归模型，不能有效地处理这么多的特征，这时候我们需要神经网络。

8.2 神经元和大脑

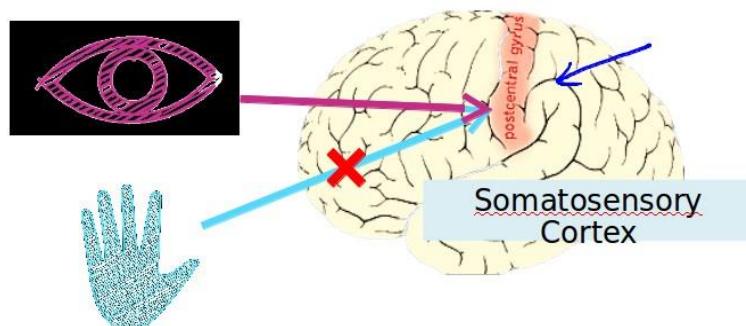
参考视频: [8 - 2 - Neurons and the Brain \(8 min\).mkv](#)

神经网络是一种很古老的算法，它最初产生的目的是制造能模拟大脑的机器。

在这门课中，我将向你们介绍神经网络。因为它能很好地解决不同的机器学习问题。而不只因为它们在逻辑上行得通，在这段视频中，我想告诉你们一些神经网络的背景知识，由此我们能知道可以用它们来做什么。不管是将其应用到现代的机器学习问题上，还是应用到那些你可能会感兴趣的问题中。也许，这一伟大的人工智能梦想在未来能制造出真正的智能机器。另外，我们还将讲解神经网络是怎么涉及这些问题的神经网络产生的原因是人们想尝试设计出模仿大脑的算法，从某种意义上说如果我们想要建立学习系统，那为什么不去模仿我们所认识的最神奇的学习机器——人类的大脑呢？

神经网络逐渐兴起于二十世纪八九十年代，应用得非常广泛。但由于各种原因，在 90 年代的后期应用减少了。但是最近，神经网络又东山再起了。其中一个原因是：神经网络是计算量有些偏大的算法。然而大概由于近些年计算机的运行速度变快，才足以真正运行起大规模的神经网络。正是由于这个原因和其他一些我们后面会讨论到的技术因素，如今的神经网络对于许多应用来说是最先进的技术。当你想模拟大脑时，是指想制造出与人类大脑作用效果相同的机器。大脑可以学会去以看而不是听的方式处理图像，学会处理我们的触觉

我们能学习数学，学着做微积分，而且大脑能处理各种不同的令人惊奇的事情。似乎如果你想要模仿它，你得写很多不同的软件来模拟所有这些五花八门的奇妙的事情。不过能不能假设大脑做所有这些，不同事情的方法，不需要用上千个不同的程序去实现。相反的，大脑处理的方法，只需要一个单一的学习算法就可以了？尽管这只是一个假设，不过让我和你分享，一些这方面的证据。



大脑的这一部分这一小片红色区域是你的听觉皮层，你现在正在理解我的话，这靠的是

耳朵。耳朵接收到声音信号，并把声音信号传递给你的听觉皮层，正因如此，你才能明白我这话。

神经系统科学家做了下面这个有趣的实验，把耳朵到听觉皮层的神经切断。在这种情况下，将其重新接到一个动物的大脑上，这样从眼睛到视神经的信号最终将传到听觉皮层。如果这样做了。那么结果表明听觉皮层将会学会“看”。这里的“看”代表了我们所知道的每层含义。所以，如果你对动物这样做，那么动物就可以完成视觉辨别任务，它们可以看图像，并根据图像做出适当的决定。它们正是通过脑组织中的这个部分完成的。下面再举另一个例子，这块红色的脑组织是你的躯体感觉皮层，这是你用来处理触觉的，如果你做一个和刚才类似的重接实验，那么躯体感觉皮层也能学会“看”。这个实验和其它一些类似的实验，被称为神经重接实验，从这个意义上说，如果人体有同一块脑组织可以处理光、声或触觉信号，那么也许存在一种学习算法，可以同时处理视觉、听觉和触觉，而不是需要运行上千个不同的程序，或者上千个不同的算法来做这些大脑所完成的成千上万的美好事情。也许我们需要做的就是找出一些近似的或实际的大脑学习算法，然后实现它大脑通过自学掌握如何处理这些不同类型的数据。在很大的程度上，可以猜想如果我们把几乎任何一种传感器接入到大脑的任何一个部位的话，大脑就会学会处理它。

下面再举几个例子：



这张图是用舌头学会“看”的一个例子。它的原理是：这实际上是一个名为 BrainPort 的系统，它现在正在 FDA (美国食品和药物管理局) 的临床试验阶段，它能帮助失明人士看见事物。它的原理是，你在前额上带一个灰度摄像头，面朝前，它就能获取你面前事物的低分辨率的灰度图像。你连一根线到舌头上安装的电极阵列上，那么每个像素都被映射到你舌头的某个位置上，可能电压值高的点对应一个暗像素电压值低的点。对于亮像素，即使依靠它现在的功能，使用这种系统就能让你我在几十分钟里就学会用我们的舌头“看”东西。



Human echolocation (sonar)

这是第二个例子，关于人体回声定位或者说人体声纳。你有两种方法可以实现：你可以弹响指，或者咂舌头。不过现在有失明人士，确实在学校里接受这样的培训，并学会解读从环境反弹回来的声波模式—这就是声纳。如果你搜索 YouTube 之后，就会发现有些视频讲述了一个令人称奇的孩子，他因为癌症眼球惨遭移除，虽然失去了眼球，但是通过打响指，他可以四处走动而不撞到任何东西，他能滑滑板，他可以将篮球投入篮框中。注意这是一个没有眼球的孩子。



Haptic belt: Direction sense

第三个例子是触觉皮带，如果你把它戴在腰上，蜂鸣器会响，而且总是朝向北时发出嗡嗡声。它可以使人们拥有方向感，用类似于鸟类感知方向的方式。

还有一些离奇的例子：



Implanting a 3rd eye

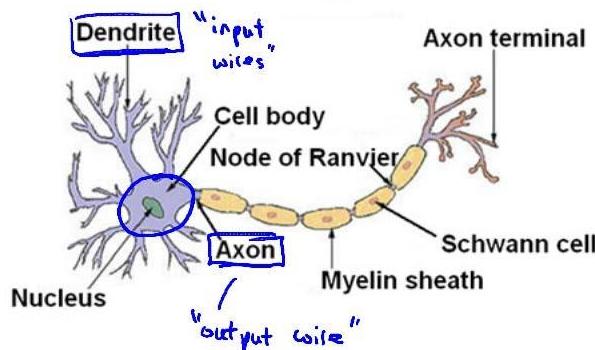
如果你在青蛙身上插入第三只眼，青蛙也能学会使用那只眼睛。因此，这将会非常令人惊奇。如果你能把几乎任何传感器接入到大脑中，大脑的学习算法就能找出学习数据的方法，并处理这些数据。从某种意义上来说，如果我们能找出大脑的学习算法，然后在计算机上执行大脑学习算法或与之相似的算法，也许这将是我们向人工智能迈进做出的最好的尝试。人工智能的梦想就是：有一天能制造出真正的智能机器。

神经网络可能为我们打开一扇进入遥远的人工智能梦的窗户，但我在这节课中讲授神经网络的原因，主要是对于现代机器学习应用。它是最有效的技术方法。因此在接下来的一些课程中，我们将开始深入到神经网络的技术细节。

8.3 模型表示 1

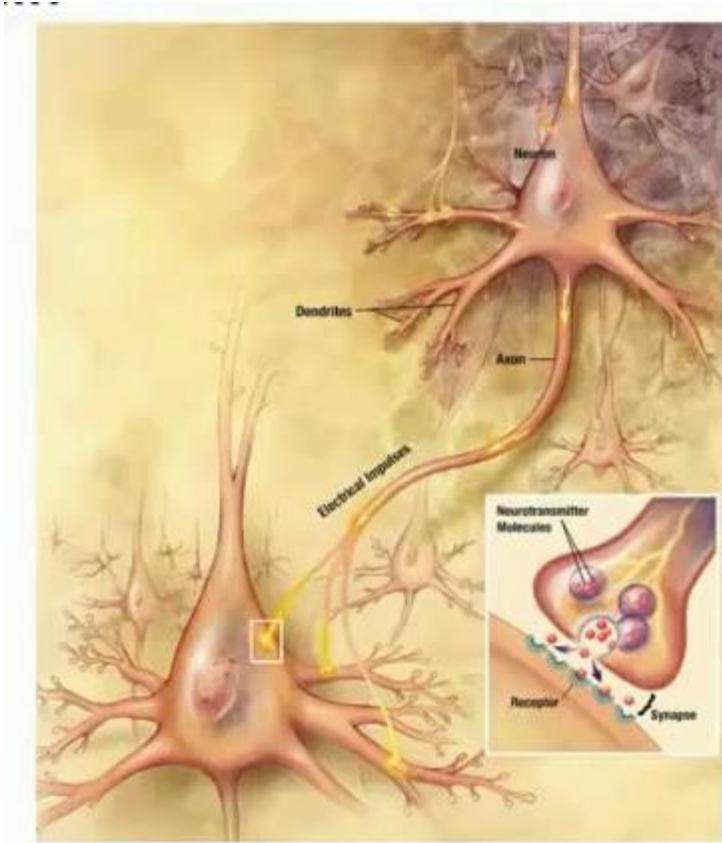
参考视频: [8 - 3 - Model Representation I \(12 min\).mkv](#)

为了构建神经网络模型, 我们需要首先思考大脑中的神经网络是怎样的? 每一个神经元都可以被认为是一个处理单元/神经核 (processing unit/ Nucleus), 它含有许多输入/树突 (input/Dendrite), 并且有一个输出/轴突 (output/Axon)。神经网络是大量神经元相互链接并通过电脉冲来交流的一个网络。

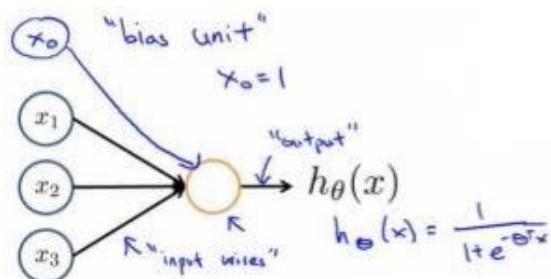


下面是一组神经元的示意图, 神经元利用微弱的电流进行沟通。这些弱电流也称作动作电位, 其实就是一些微弱的电流。所以如果神经元想要传递一个消息, 它就会就通过它的轴突, 发送一段微弱电流给其他神经元, 这就是轴突。

这里是一条连接到输入神经, 或者连接另一个神经元树突的神经, 接下来这个神经元接收这条消息, 做一些计算, 它有可能会反过来将在轴突上的自己的消息传给其他神经元。这就是所有人类思考的模型: 我们的神经元把自己的收到的消息进行计算, 并向其他神经元传递消息。这也是我们的感觉和肌肉运转的原理。如果你想活动一块肌肉, 就会触发一个神经元给你的肌肉发送脉冲, 并引起你的肌肉收缩。如果一些感官: 比如说眼睛想要给大脑传递一个消息, 那么它就像这样发送电脉冲给大脑的。

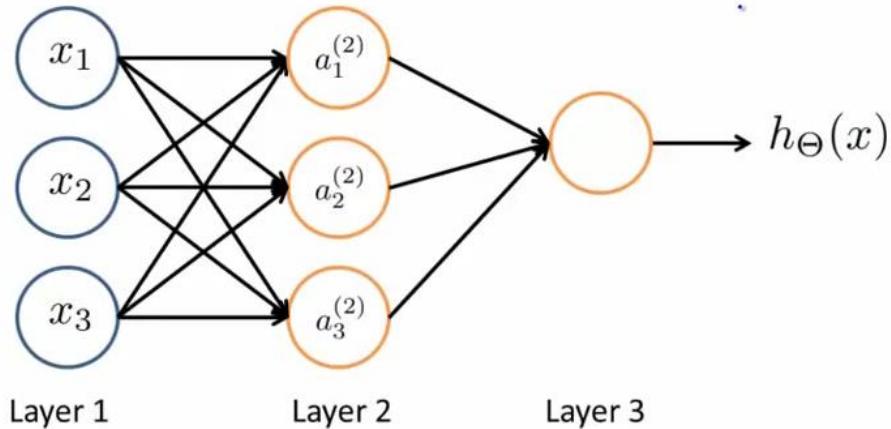


神经网络模型建立在很多神经元之上，每一个神经元又是一个个学习模型。这些神经元（也叫激活单元， activation unit）采纳一些特征作为输出，并且根据本身的模型提供一个输出。下图是一个以逻辑回归模型作为自身学习模型的神经元示例，在神经网络中，参数又可被成为权重（weight）。



Sigmoid (logistic) activation function.

我们设计出了类似于神经元的神经网络，效果如下：

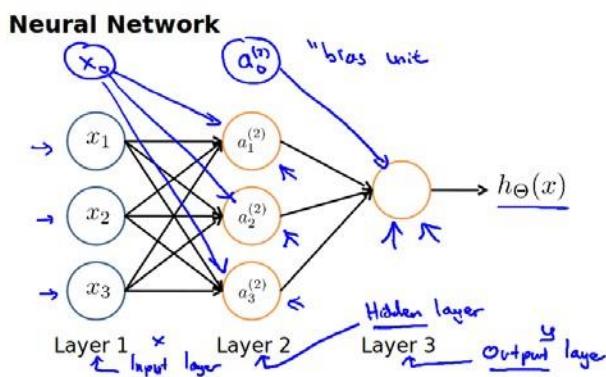


其中 x_1, x_2, x_3 是输入单元 (input units)，我们将原始数据输入给它们。

a_1, a_2, a_3 是中间单元，它们负责将数据进行处理，然后呈递到下一层。

最后是输出单元，它负责计算 $h(x)$ 。

神经网络模型是许多逻辑单元按照不同层级组织起来的网络，每一层的输出变量都是下一层的输入变量。下图是一个 3 层的神经网络，第一层成为输入层 (Input Layer)，最后一层称为输出层 (Output Layer)，中间一层成为隐藏层 (Hidden Layers)。我们为每一层都增加一个偏差单位 (bias unit)：



下面引入一些标记法来帮助描述模型：

$a_i^{(j)}$ 代表第 j 层的第 i 个激活单元。 $\theta^{(j)}$ 代表从第 j 层映射到第 $j+1$ 层时的权重的矩阵，例如 $\theta^{(1)}$ 代表从第一层映射到第二层的权重的矩阵。其尺寸为：以第 j 层的激活单元数量为行数，以第 $j+1$ 层的激活单元数为列数的矩阵。例如：上图所示的神经网络中 $\theta^{(1)}$ 的尺寸为 4×3 。

对于上图所示的模型，激活单元和输出分别表达为：

$$\begin{aligned}
 a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\
 a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\
 a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\
 h_{\Theta}(x) &= g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})
 \end{aligned}$$

上面进行的讨论中只是将特征矩阵中的一行（一个训练实例）喂给了神经网络，我们需要将整个训练集都喂给我们的神经网络算法来学习模型。

我们可以知道：每一个 a 都是由上一层所有的 x 和每一个 x 所对应的 θ 决定的。

（我们把这样从左到右的算法称为前向传播算法(FORWARD PROPAGATION)）

把 x, θ, a 分别用矩阵表示： $X = \begin{matrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{matrix}$, $\theta = \begin{matrix} \theta_{10} & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \theta_{33} \end{matrix}$, $a = \begin{matrix} a_1 \\ a_2 \\ a_3 \end{matrix}$

我们可以得到 $\theta * X = a$

8.4 模型表示 2

参考视频: [8 - 4 - Model Representation II \(12 min\).mkv](#)

(FORWARD PROPAGATION) 相对于使用循环来编码，利用向量化的方法会使得计算更为简便。以上面的神经网络为例，试着计算第二层的值：

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$g\left(\begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} \theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3 \\ \theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3 \\ \theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3 \end{bmatrix}\right) = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

我们令 $z^{(2)} = \Theta^{(1)}x$ ，则 $a^{(2)} = g(z^{(2)})$ ，计算后添加 $a_0^{(2)} = I$ 。计算输出的值为：

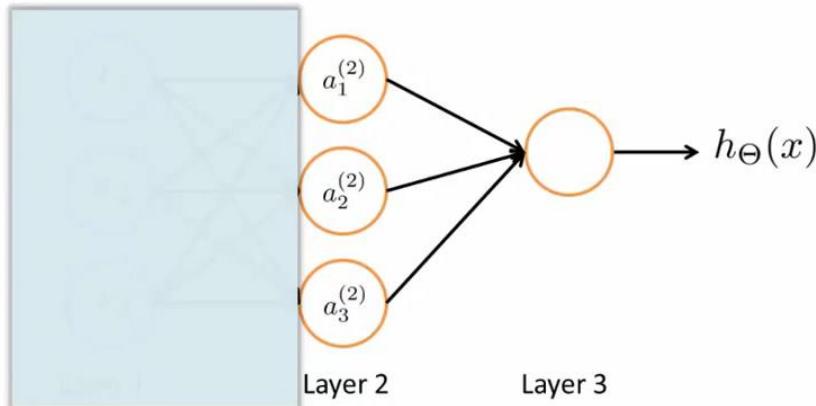
$$g\left(\begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \\ \theta_{20}^{(2)} & \theta_{21}^{(2)} & \theta_{22}^{(2)} & \theta_{23}^{(2)} \\ \theta_{30}^{(2)} & \theta_{31}^{(2)} & \theta_{32}^{(2)} & \theta_{33}^{(2)} \end{bmatrix} \times \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}\right) = g(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}) = h_\theta(x)$$

我们令 $z^{(3)} = \Theta^{(2)}a^{(2)}$ ，则 $h_\theta(x) = a^{(3)} = g(z^{(3)})$ 。

这只是针对训练集中一个训练实例所进行的计算。如果我们要对整个训练集进行计算，我们需要将训练集特征矩阵进行转置，使得同一个实例的特征都在同一列里。即：

$$\begin{aligned} z^{(2)} &= \Theta^{(1)} \times X^T \\ a^{(2)} &= g(z^{(2)}) \end{aligned}$$

为了更好地了解 Neuron Networks 的工作原理，我们先把左半部分遮住：



右半部分其实是以 a_0, a_1, a_2, a_3 按照 Logistic Regression 的方式输出 $h(x)$:

$$h_{\Theta}(x) = g(\Theta_0^{(2)} a_0^{(2)} + \Theta_1^{(2)} a_1^{(2)} + \Theta_2^{(2)} a_2^{(2)} + \Theta_3^{(2)} a_3^{(2)})$$

其实神经网络就像是 logistic regression，只不过我们把 logistic regression 中的输入向量 $[x_1 \sim x_3]$ 变成了中间层的 $[a(2)_1 \sim a(2)_3]$ ，即

$$h(x) = g(\Theta^{(2)}_0 a^{(2)}_0 + \Theta^{(2)}_1 a^{(2)}_1 + \Theta^{(2)}_2 a^{(2)}_2 + \Theta^{(2)}_3 a^{(2)}_3)$$

我们可以把 a_0, a_1, a_2, a_3 看成更为高级的特征值，也就是 x_0, x_1, x_2, x_3 的进化体，并且它们是由 x 与 θ 决定的，因为 θ 是梯度下降的，所以 a 是变化的，并且变得越来越厉害，所以这些更高级的特征值远比仅仅将 x 次方厉害，也能更好的预测新数据。

这就是神经网络相比于逻辑回归和线性回归的优势。

8.5 特征和直观理解 1

参考视频: [8 - 5 - Examples and Intuitions I \(7 min\).mkv](#)

从本质上讲，神经网络能够通过学习得出其自身的一系列特征。在普通的逻辑回归中，我们被限制为使用数据中的原始特征 x_1, x_2, \dots, x_n ，我们虽然可以使用一些二项式项来组合这些特征，但是我们仍然受到这些原始特征的限制。在神经网络中，原始特征只是输入层，在我们上面三层的神经网络例子中，第三层也就是输出层做出的预测利用的是第二层的特征，而非输入层中的原始特征，我们可以认为第二层中的特征是神经网络通过学习后自己得出的一系列用于预测输出变量的新特征。

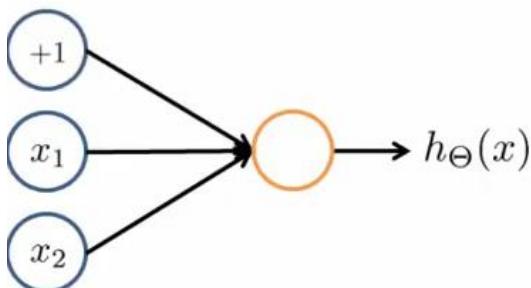
神经网络中，单层神经元（无中间层）的计算可用来表示逻辑运算，比如逻辑 AND、逻辑或 OR。

举例说明：逻辑与 AND；下图中左半部分是神经网络的设计与 output 层表达式，右边上部分是 sigmoid 函数，下半部分是真值表。

我们可以用这样的一个神经网络表示 AND 函数：

$$x_1, x_2 \in \{0, 1\}$$

$$y = x_1 \text{ AND } x_2$$

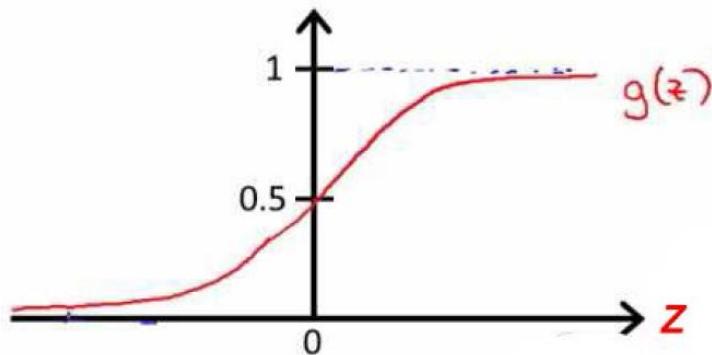


其中 $\theta_0 = -30$, $\theta_1 = -20$, $\theta_2 = 20$

我们的输出函数 $h(x)$ 即为：

$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

我们知道 $g(x)$ 的图像是：



x_1	x_2	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
→ 0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

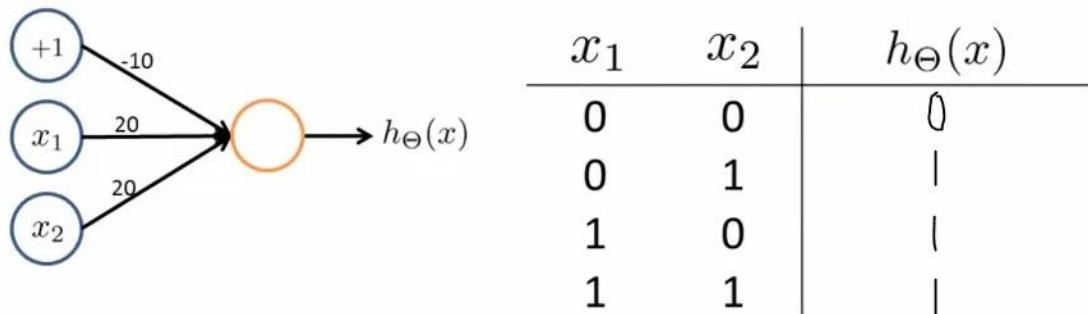
所以我们有：

$$h_{\Theta}(x) \approx x_1 \text{ AND } x_2$$

所以我们的：

这就是 AND 函数。

接下来再介绍一个 OR 函数：



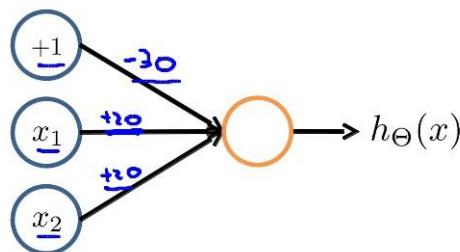
OR 与 AND 整体一样，区别只在于 θ 的取值不同。

8.6 样本和直观理解 II

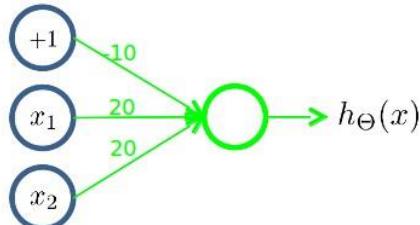
参考视频: [8 - 6 - Examples and Intuitions II \(10 min\).mkv](#)

二元逻辑运算符 (BINARY LOGICAL OPERATORS) 当输入特征为布尔值 (0 或 1) 时，我们可以用一个单一的激活层可以作为二元逻辑运算符，为了表示不同的运算符，我们之需要选择不同的权重即可。

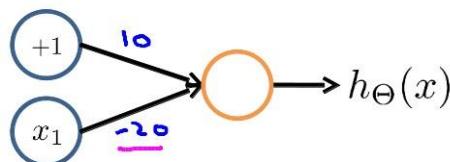
下图的神经元 (三个权重分别为-30, 20, 20) 可以被视为作用同于逻辑与 (AND):



下图的神经元 (三个权重分别为-10, 20, 20) 可以被视为作用等同于逻辑或 (OR):

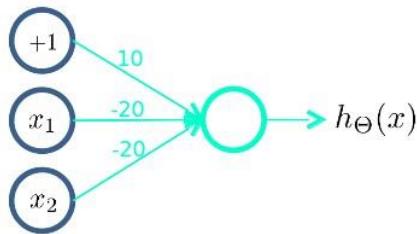


下图的神经元 (两个权重分别为 10, -20) 可以被视为作用等同于逻辑非 (NOT):

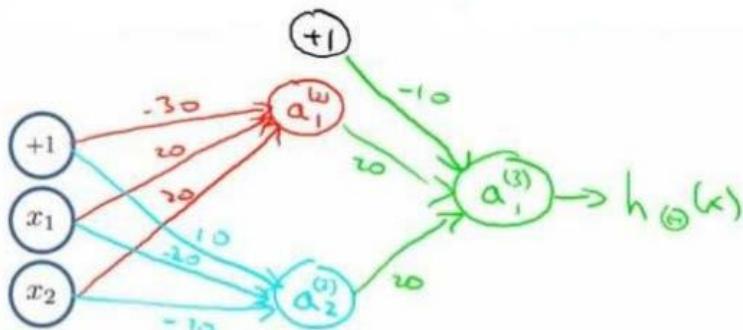


我们可以利用神经元来组合成更为复杂的神经网络以实现更复杂的运算。例如我们要实现 XNOR 功能 (输入的两个值必须一样，均为 1 或均为 0)，即 $XNOR = (x_1 \text{AND} x_2) \text{OR}((\text{NOT}x_1) \text{AND} (\text{NOT}x_2))$

首先构造一个能表达 $(\text{NOT}x_1) \text{AND} (\text{NOT}x_2)$ 部分的神经元：



然后将表示 AND 的神经元和表示(NOT x_1)AND(NOT x_2)的神经元以及表示 OR 的神经元进行组合：



我们就得到了一个能实现 XNOR 运算符功能的神经网络。

按这种方法我们可以逐渐构造出越来越复杂的函数，也能得到更加厉害的特征值。

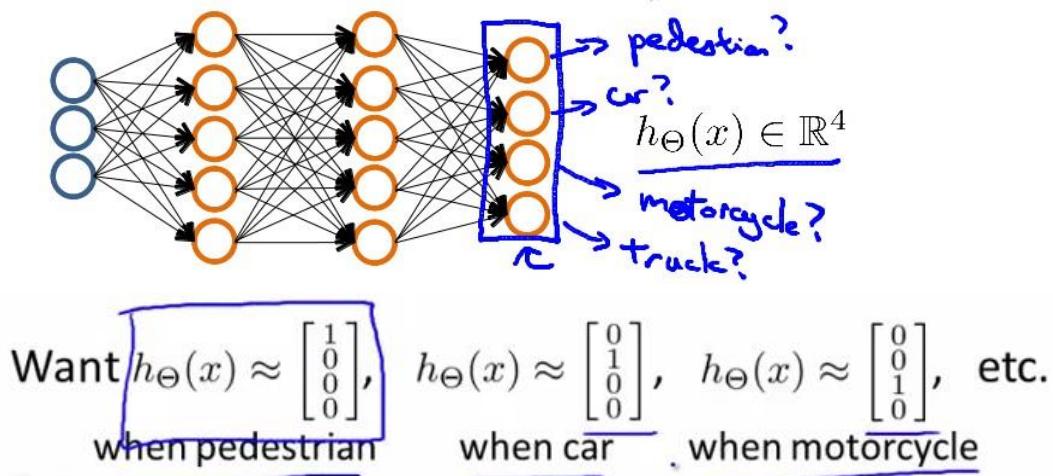
这就是神经网络的厉害之处。

8.7 多类分类

参考视频: [8 - 7 - Multiclass Classification \(4 min\).mkv](#)

当我们有不止两种分类时（也就是 $y=1,2,3\cdots$ ），比如以下这种情况，该怎么办？如果我们要训练一个神经网络算法来识别路人、汽车、摩托车和卡车，在输出层我们应该有 4 个值。例如，第一个值为 1 或 0 用于预测是否是行人，第二个值用于判断是否为汽车。

输入向量 x 有三个维度，两个中间层，输出层 4 个神经元分别用来表示 4 类，也就是每一个数据在输出层都会出现 $[a \ b \ c \ d]^T$ ，且 a,b,c,d 中仅有一个为 1，表示当前类。下面是该神经网络的可能结构示例：



神经网络算法的输出结果为四种可能情形之一：

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

第 5 周

九、神经网络的学习(Neural Networks: Learning)

9.1 代价函数

参考视频: [9 - 1 - Cost Function \(7 min\).mkv](#)

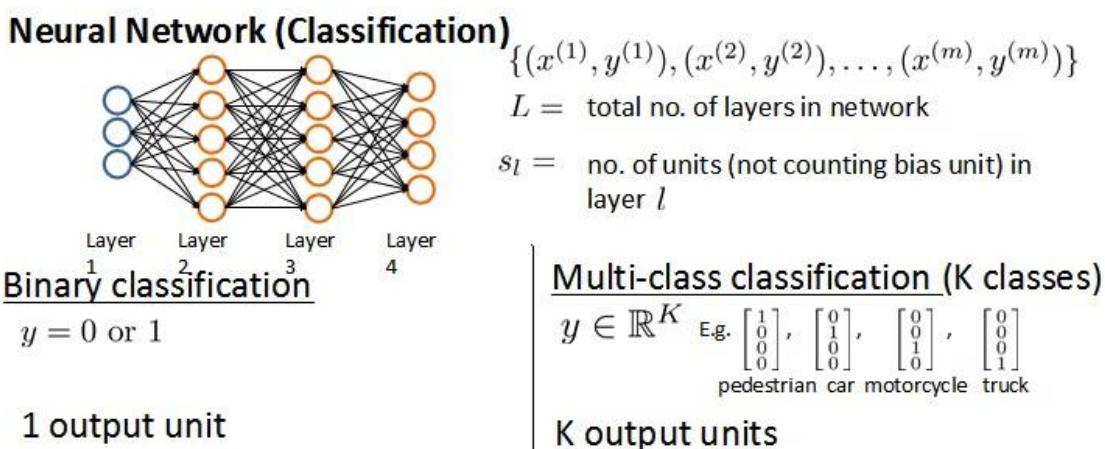
首先引入一些便于稍后讨论的新标记方法:

假设神经网络的训练样本有 m 个, 每个包含一组输入 x 和一组输出信号 y , L 表示神经网络层数, s_l 表示每层的 neuron 个数(s_L 表示输出层神经元个数), s_L 代表最后一层中处理单元的个数。

将神经网络的分类定义为两种情况: 二类分类和多类分类,

二类分类: $s_L=1$, $y=0$ or 1 表示哪一类;

K 类分类: $s_L=K$, $y_i = 1$ 表示分到第 i 类; ($K>2$)



我们回顾逻辑回归问题中我们的代价函数为:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

在逻辑回归中, 我们只有一个输出变量, 又称标量 (scalar), 也只有一个因变量 y , 但是在神经网络中, 我们可以有很多输出变量, 我们的 $h_\theta(x)$ 是一个维度为 K 的向量, 并且我们训练集中的因变量也是同样维度的一个向量, 因此我们的代价函数会比逻辑回归更加复杂一些, 为:

Neural network:

$$h_{\Theta}(x) \in \mathbb{R}^K \quad (h_{\Theta}(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\Theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\Theta}(x^{(i)}))_k) \right]$$

$$+ \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

这个看起来复杂很多的代价函数背后的思想还是一样的，我们希望通过代价函数来观察算法预测的结果与真实情况的误差有多大，唯一不同的是，对于每一行特征，我们都会给出 K 个预测，基本上我们可以利用循环，对每一行特征都预测 K 个不同结果，然后在利用循环在 K 个预测中选择可能性最高的一个，将其与 y 中的实际数据进行比较。

归一化的那一项只是排除了每一层 θ_0 后，每一层的 Θ 矩阵的和。最里层的循环 j 循环所有的行（由 s_{l+1} 层的激活单元数决定），循环 i 则循环所有的列，由该层 (s_l 层) 的激活单元数所决定。即： $h_{\theta}(x)$ 与真实值之间的距离为每个样本-每个类输出的加和，对参数进行 regularization 的 bias 项处理所有参数的平方和。

9.2 反向传播算法

参考视频: [9 - 2 - Backpropagation Algorithm \(12 min\).mkv](#)

之前我们在计算神经网络预测结果的时候我们采用了一种正向传播方法, 我们从第一层开始正向一层一层进行计算, 直到最后一层的 $h_\theta(x)$ 。

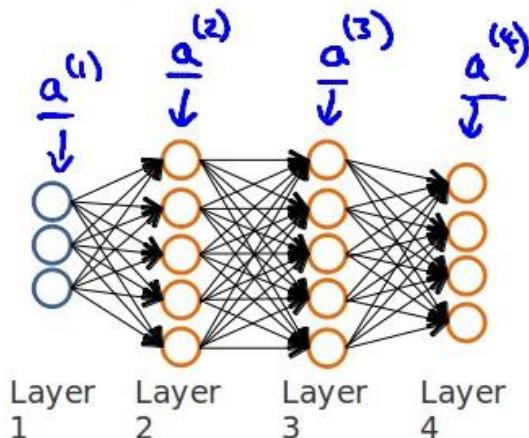
现在, 为了计算代价函数的偏导数 $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$, 我们需要采用一种反向传播算法, 也

就是首先计算最后一层的误差, 然后再一层一层反向求出各层的误差, 直到倒数第二层。以一个例子来说明反向传播算法。

假设我们的训练集只有一个实例 $(x^{(1)}, y^{(1)})$, 我们的神经网络是一个四层的神经网络, 其中 $K=4$, $S_L=4$, $L=4$:

前向传播算法:

$$\begin{aligned} a^{(1)} &= x \\ z^{(2)} &= \Theta^{(1)} a^{(1)} \\ a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\ z^{(3)} &= \Theta^{(2)} a^{(2)} \\ a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\ z^{(4)} &= \Theta^{(3)} a^{(3)} \\ a^{(4)} &= h_\Theta(x) = g(z^{(4)}) \end{aligned}$$



我们从最后一层的误差开始计算, 误差是激活单元的预测 ($a_k^{(4)}$) 与实际值 (y^k) 之间的误差, ($k=1:K$)。

我们用 δ 来表示误差, 则: $\delta^{(4)} = a^{(4)} - y$

我们利用这个误差值来计算前一层的误差: $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} * g'(z^{(3)})$

其中 $g'(z^{(3)})$ 是 S 形函数的导数, $g'(z^{(3)}) = a^{(3)} * (1 - a^{(3)})$ 。而 $(\Theta^{(3)})^T \delta^{(4)}$ 则是权重导致的误差的和。下一步是继续计算第二层的误差:

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} * g'(z^{(2)})$$

因为第一层是输入变量, 不存在误差。我们有了所有的误差的表达式后, 便可以计算代价函数的偏导数了, 假设 $\lambda=0$, 即我们不做任何归一化处理时有:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{l+1}$$

重要的是清楚地知道上面式子中上下标的含义:

l 代表目前所计算的是第几层

j 代表目前计算层中的激活单元的下标, 也将是下一层的第 j 个输入变量的下标。

i 代表下一层中误差单元的下标, 是受到权重矩阵中第 i 行影响的下一层中的误差单元的下标。

如果我们考虑归一化处理, 并且我们的训练集是一个特征矩阵而非向量。在上面的特殊情况下, 我们需要计算每一层的误差单元来计算代价函数的偏导数。在更为一般的情况下, 我们同样需要计算每一层的误差单元, 但是我们需要为整个训练集计算误差单元, 此时的误差单元也是一个矩阵, 我们用 $\Delta_{ij}^{(l)}$ 来表示这个误差矩阵。第 l 层的第 i 个激活单元受到第 j 个参数影响而导致的误差。

我们的算法表示为:

```

for i=1:m  {
    set a(i)=x(i)
    perform foward propagation to compute a(l) for l=1,2,3...L
    Using δ(L)=a(L)-yi
    perform back propagation to compute all previous layer error vector
    Δij(l):=Δij(l)+aj(l)δil+1
}

```

即首先用正向传播方法计算出每一层的激活单元, 利用训练集的结果与神经网络预测的结果求出最后一层的误差, 然后利用该误差运用反向传播法计算出直至第二层的所有误差。

在求出了 $\Delta_{ij}^{(l)}$ 之后, 我们便可以计算代价函数的偏导数了, 计算方法如下:

$$\begin{aligned} D_{ij}^{(l)} &:= \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0 \\ D_{ij}^{(l)} &:= \frac{1}{m} \Delta_{ij}^{(l)} \quad \text{if } j = 0 \end{aligned}$$

在 Octave 中，如果我们要使用 `fminuc` 这样的优化算法来求解求出权重矩阵，我们需要将矩阵首先展开成为向量，在利用算法求出最优解后再重新转换回矩阵。

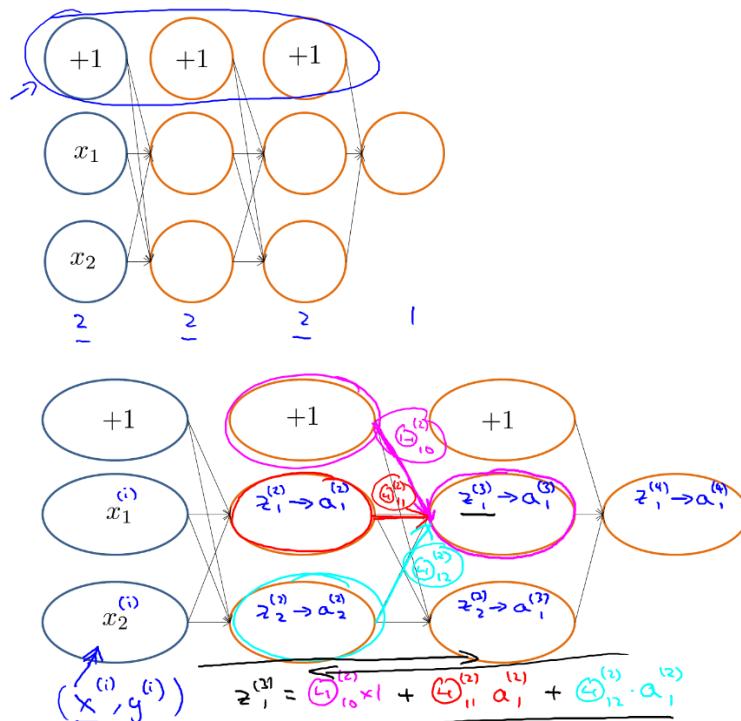
假设我们有三个权重矩阵，`Theta1`, `Theta2` 和 `Theta3`, 尺寸分别为 `10*11`, `10*11` 和 `1*11`, 下面的代码可以实现这样的转换:

```
thetaVec = [Theta1(:) ; Theta2(:) ; Theta3(:)]  
...optimization using functions like fminuc...  
Theta1 = reshape(thetaVec(1:110, 10, 11);  
Theta2 = reshape(thetaVec(111:220, 10, 11);  
Theta3 = reshape(thetaVec(221:231, 1, 11);
```

9.3 反向传播算法的直观理解

参考视频: [9 - 3 - Backpropagation Intuition \(13 min\).mkv](#)

前向传播算法:



后向传播算法做的是:

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

$(x^{(i)}, y^{(i)})$

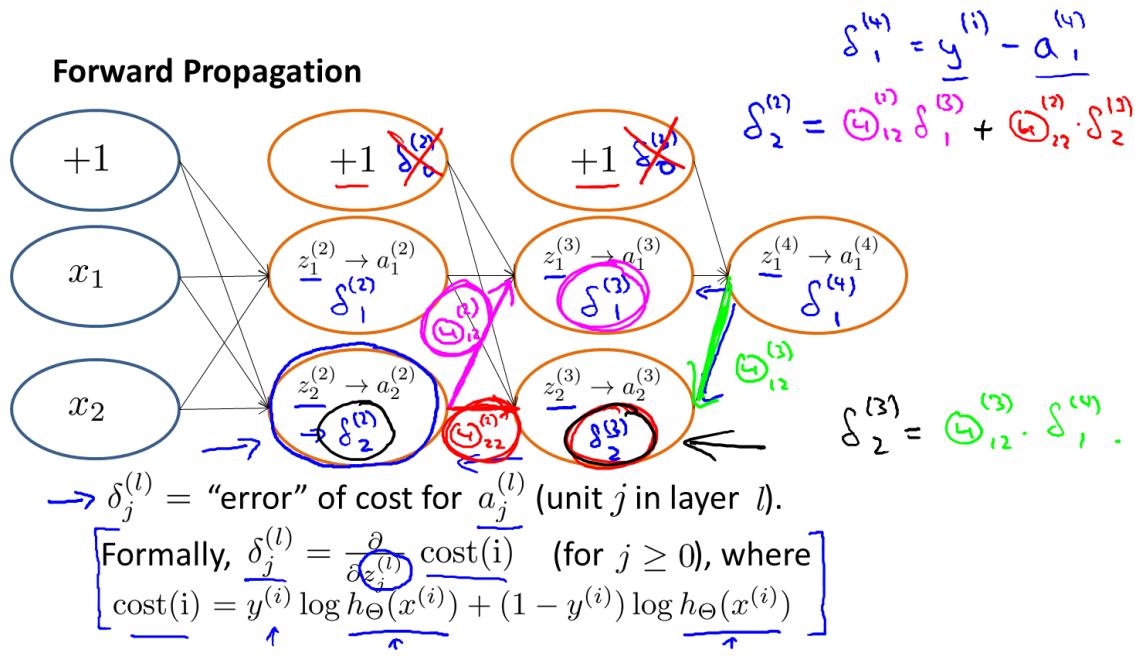
Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost}(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

I.e. how well is the network doing on example i?

$y^{(i)}$



9.4 实现注意：展开参数

参考视频: [9 - 4 - Implementation Note Unrolling Parameters \(8 min\).mkv](#)

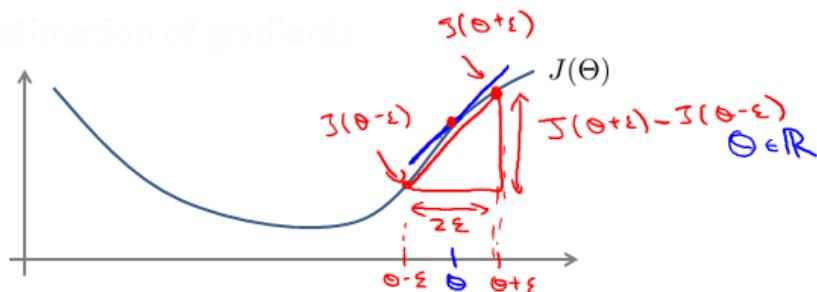
9.5 梯度检验

参考视频: [9 - 5 - Gradient Checking \(12 min\).mkv](#)

当我们对一个较为复杂的模型(例如神经网络)使用梯度下降算法时,可能会存在一些不容易察觉的错误,意味着,虽然代价看上去在不断减小,但最终的结果可能并不是最优解。

为了避免这样的问题,我们采取一种叫做梯度的数值检验(Numerical Gradient Checking)方法。这种方法的思想是通过估计梯度值来检验我们计算的导数值是否真的是我们要求的。

对梯度的估计采用的方法是在代价函数上沿着切线的方向选择离两个非常近的点然后计算两个点的平均值用以估计梯度。即对于某个特定的 θ , 我们计算出在 $\theta-\epsilon$ 处和 $\theta+\epsilon$ 的代价值(ϵ 是一个非常小的值, 通常选取 0.001), 然后求两个代价的平均, 用以估计在 θ 处的代价值。



Octave 中代码如下:

```
gradApprox = (J(theta + eps) - J(theta - eps)) / (2*eps)
```

当 θ 是一个向量时, 我们则需要对偏导数进行检验。因为代价函数的偏导数检验只针对一个参数的改变进行检验, 下面是一个只针对 θ_1 进行检验的示例:

$$\frac{\partial}{\partial \theta_1} = \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

最后我们还需要对通过反向传播方法计算出的偏导数进行检验。

根据上面的算法, 计算出的偏导数存储在矩阵 $D_{ij}^{(l)}$ 中。检验时, 我们要将该矩阵展开成为向量, 同时我们也将 θ 矩阵展开为向量, 我们针对每一个 θ 都计算一个近似的梯度值, 将这些值存储于一个近似梯度矩阵中, 最终将得出的这个矩阵同 $D_{ij}^{(l)}$ 进行比较。

```
for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                    /(2*EPSILON);
end;

Check that gradApprox ≈ DVec
```

9.6 随机初始化

参考视频: [9 - 6 - Random Initialization \(7 min\).mkv](#)

任何优化算法都需要一些初始的参数。到目前为止我们都是初始所有参数为 0，这样的初始方法对于逻辑回归来说是可行的，但是对于神经网络来说是不可行的。如果我们令所有的初始参数都为 0，这将意味着我们第二层的所有激活单元都会有相同的值。同理，如果我们初始所有的参数都为一个非 0 的数，结果也是一样的。

我们通常初始参数为正负 ϵ 之间的随机值，假设我们要随机初始一个尺寸为 10×11 的参数矩阵，代码如下：

```
Theta1 = rand(10, 11) * (2*eps) - eps
```

9.7 综合起来

参考视频: [9 - 7 - Putting It Together \(14 min\).mkv](#)

小结一下使用神经网络时的步骤:

网络结构: 第一件要做的事是选择网络结构, 即决定选择多少层以及决定每层分别有多少个单元。

第一层的单元数即我们训练集的特征数量。

最后一层的单元数是我们训练集的结果的类的数量。

如果隐藏层数大于 1, 确保每个隐藏层的单元个数相同, 通常情况下隐藏层单元的个数越多越好。

我们真正要决定的是隐藏层的层数和每个中间层的单元数。

训练神经网络:

1. 参数的随机初始化
2. 利用正向传播方法计算所有的 $h_\theta(x)$
3. 编写计算代价函数 J 的代码
4. 利用反向传播方法计算所有偏导数
5. 利用数值检验方法检验这些偏导数
6. 使用优化算法来最小化代价函数

9.8 自主驾驶

参考视频: [9 - 8 - Autonomous Driving \(7 min\).mkv](#)

在这段视频中，我想向你介绍一个具有历史意义的神经网络学习的重要例子。那就是使用神经网络来实现自动驾驶，也就是说使汽车通过学习来自己驾驶。接下来我将演示的这段视频是我从 Dean Pomerleau 那里拿到的，他是我的同事，任职于美国东海岸的卡耐基梅隆大学。在这部分视频中，你就会明白可视化技术到底是什么？在看这段视频之前，我会告诉你可视化技术是什么。

在下面也就是左下方，就是汽车所看到的前方的路况图像。



在图中你依稀能看出一条道路，朝左延伸了一点，又向右了一点，然后上面的这幅图，你可以看到一条水平的菜单栏显示的是驾驶操作人选择的方向。就是这里的这条白亮的区段显示的就是人类驾驶者选择的方向。比如：最左边的区段，对应的操作就是向左急转，而最右端则对应向右急转的操作。因此，稍微靠左的区段，也就是中心稍微向左一点的位置，则表示在这一点上人类驾驶者的操作是慢慢的向左拐。

这幅图的第二部分对应的就是学习算法选出的行驶方向。并且，类似的，这一条白亮的区段显示的就是神经网络在这里选择的行驶方向，是稍微的左转，并且实际上在神经网络开始学习之前，你会看到网络的输出是一条灰色的区段，就像这样的一条灰色区段覆盖着整个区域这些均称的灰色区域，显示出神经网络已经随机初始化了，并且初始化时，我们并不知道汽车如何行驶，或者说我们并不知道所选行驶方向。只有在学习算法运行了足够长的时间之后，才会有这条白色的区段出现在整条灰色区域之中。显示出一个具体的行驶方向这就表

示神经网络算法，在这时候已经选出了一个明确的行驶方向，不像刚开始的时候，输出一段模糊的浅灰色区域，而是输出一条白亮的区段，表示已经选出了明确的行驶方向。



ALVINN (Autonomous Land Vehicle In a Neural Network)是一个基于神经网络的智能系统，通过观察人类的驾驶来学习驾驶，ALVINN 能够控制 NavLab，装在一一辆改装版军用悍马，这辆悍马装载了传感器、计算机和驱动器用来进行自动驾驶的导航试验。实现 ALVINN 功能的第一步，是对它进行训练，也就是训练一个人驾驶汽车。



然后让 ALVINN 观看，ALVINN 每两秒将前方的路况图生成一张数字化图片，并且记录驾驶者的驾驶方向，得到的训练集图片被压缩为 30x32 像素，并且作为输入提供给 ALVINN 的三层神经网络，通过使用反向传播学习算法，ALVINN 会训练得到一个与人类驾驶员操纵方向基本相近的结果。一开始，我们的网络选择出的方向是随机的，大约经过两分钟的训练后，

我们的神经网络便能够准确地模拟人类驾驶者的驾驶方向，对其他道路类型，也重复进行这个训练过程，当网络被训练完成后，操作者就可按下运行按钮，车辆便开始行驶了。



每秒钟 ALVINN 生成 12 次数字化图片，并且将图像传送给神经网络进行训练，多个神经网络同时工作，每一个网络都生成一个行驶方向，以及一个预测自信度的参数，预测自信度最高的那个神经网络得到的行驶方向。比如这里，在这条单行道上训练出的网络将被最终用于控制车辆方向，车辆前方突然出现了一个交叉十字路口，当车辆到达这个十字路口时，我们单行道网络对应的自信度骤减，当它穿过这个十字路口时，前方的双车道将进入其视线，双车道网络的自信度便开始上升，当它的自信度上升时 双车道的网络，将被选择来控制行驶方向，车辆将被安全地引导进入双车道路。

这就是基于神经网络的自动驾驶技术。当然，我们还有很多更加先进的试验来实现自动驾驶技术。在美国，欧洲等一些国家和地区，他们提供了一些比这个方法更加稳定的驾驶控制技术。但我认为，使用这样一个简单的基于反向传播的神经网络，训练出如此强大的自动驾驶汽车，的确是一次令人惊讶的成就。

第 6 周

十、应用机器学习的建议(Advice for Applying Machine Learning)

10.1 决定下一步做什么

参考视频: [10 - 1 - Deciding What to Try Next \(6 min\).mkv](#)

假设我们需要用一个线性回归模型来预测房价, 当我们运用训练好了的模型来预测未知数据的时候发现有较大的误差, 我们下一步可以做什么?

1. 获得更多的训练实例——通常是有效的, 但代价较大, 下面的方法也可能有效, 可考虑先采用下面的几种方法。
2. 尝试减少特征的数量
3. 尝试获得更多的特征
4. 尝试增加多项式特征
5. 尝试减少归一化程度 λ
6. 尝试增加归一化程度 λ

我们不应该随机选择上面的某种方法来改进我们的算法, 而是运用一些机器学习诊断法来帮助 我们知道上面哪些方法对我们的算法是有效的。

10.2 评估一个假设

参考视频: [10 - 2 - Evaluating a Hypothesis \(8 min\).mkv](#)

过拟合检验

为了检验算法是否过拟合，我们将数据分成训练集和测试集，通常用 70% 的数据作为训练集，用剩下 30% 的数据作为测试集。很重要的一点是训练集和测试集均要含有各种类型的数据，通常我们要对数据进行“洗牌”，然后再分成训练集和测试集。

测试集评估在通过训练集让我们的模型学习得出其参数后，对测试集运用该模型，我们有两种方式计算误差：

1. 对于线性回归模型，我们利用测试集数据计算代价函数 J
2. 对于逻辑回归模型，我们除了可以利用测试数据集来计算代价函数外：

$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_{\theta}(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log (1 - h_{\theta}(x_{test}^{(i)}))$$

误分类的比率，对于每一个测试集实例，计算：

$$err(h_{\theta}(x), y) = \begin{cases} 1 & \text{if } h(x) \geq 0.5 \text{ and } y = 0, \text{ or if } h(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{Otherwise} \end{cases}$$

然后对计算结果求平均。

10.3 模型选择和交叉验证集

参考视频: [10 - 3 - Model Selection and Train Validation Test Sets \(12 min\).mkv](#)

假设我们要在 10 个不同次数的二项式模型之间进行选择:

1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3$
- \vdots
10. $h_\theta(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10}$

显然越高次数的多项式模型越能够适应我们的训练数据集,但是适应训练数据集并不代表着能推广至一般情况,我们应该选择一个更能适应一般情况的模型。我们需要使用交叉验证集来帮助选择模型。

即: 使用 60%的数据作为训练集, 使用 20%的数据作为交叉验证集, 使用 20%的数据作为测试集

Evaluating your hypothesis

Dataset:

Size	Price
2104	400
1600	330
60%	2400
1416	369
3000	232
1985	540
	300
20%	1534
1427	315
	199
20%	1380
1494	212
	243

Handwritten annotations:

- A blue bracket on the left side of the table is labeled "60%" above it.
- A blue bracket on the right side of the first six rows is labeled "Training set".
- A blue bracket on the right side of the next two rows is labeled "Cross validation set (CV)".
- A blue bracket on the right side of the last two rows is labeled "test set".

模型选择的方法为:

1. 使用训练集训练出 10 个模型
2. 用 10 个模型分别对交叉验证集计算得出交叉验证误差 (代价函数的值)
3. 选取代价函数值最小的模型
4. 用步骤 3 中选出的模型对测试集计算得出推广误差 (代价函数的值)

Train/validation/test error

Training error:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

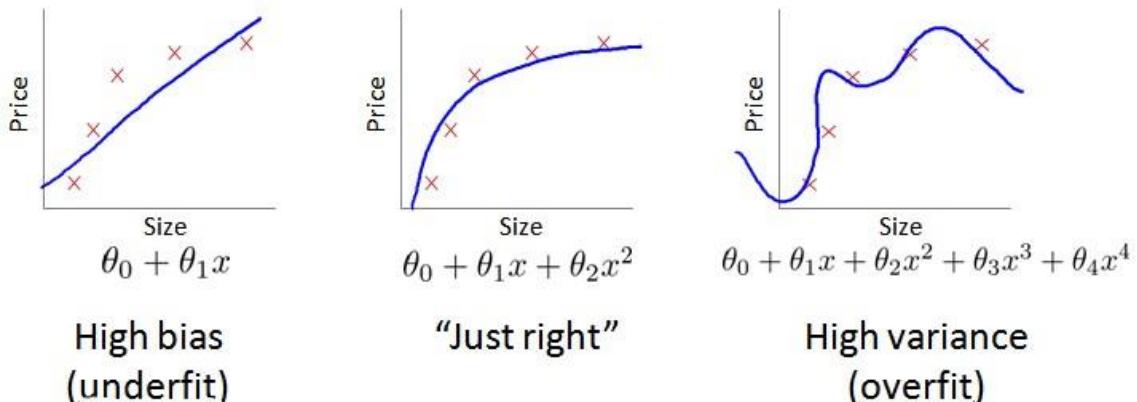
Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

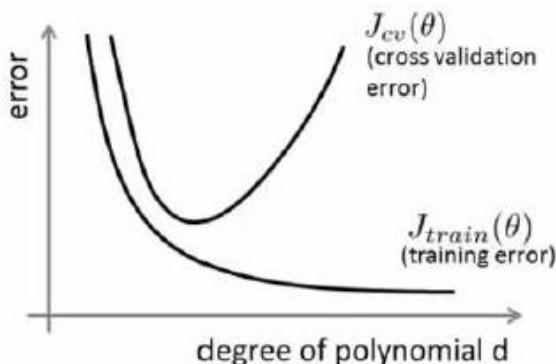
10.4 诊断偏差和方差

参考视频: [10 - 4 - Diagnosing Bias vs. Variance \(8 min\).mkv](#)

高方差和高偏差的问题基本上来说是低拟合和过拟合的问题。



我们通常会通过将训练集和交叉验证集的代价函数误差与多项式的次数绘制在同一张图表上来帮助分析:



Bias/variance

$$\text{Training error: } J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\text{Cross validation error: } J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

对于训练集, 当 d 较小时, 模型拟合程度更低, 误差较大; 随着 d 的增长, 拟合程度提高, 误差减小。

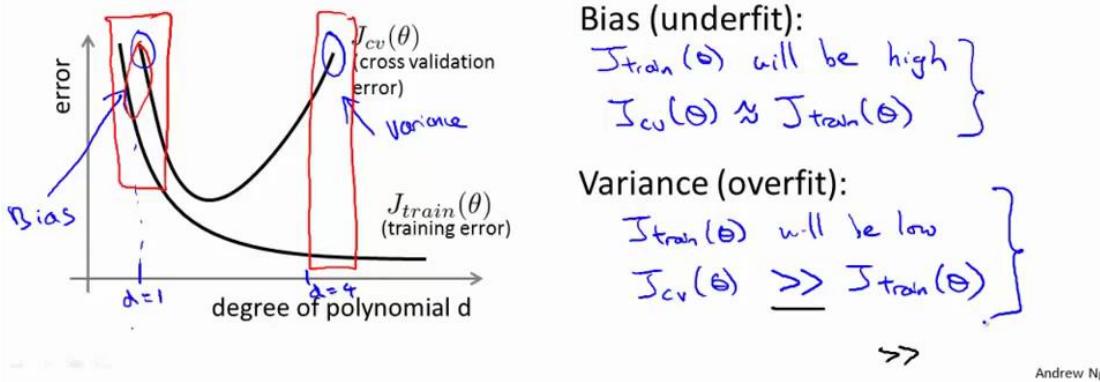
对于交叉验证集, 当 d 较小时, 模型拟合程度低, 误差较大; 但是随着 d 的增长, 误差呈现先减小后增大的趋势, 转折点是我们的模型开始过拟合训练数据集的时候。

如果我们的交叉验证集误差较大, 我们如何判断是方差还是偏差呢? 根据上面的图表,

我们知道：

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



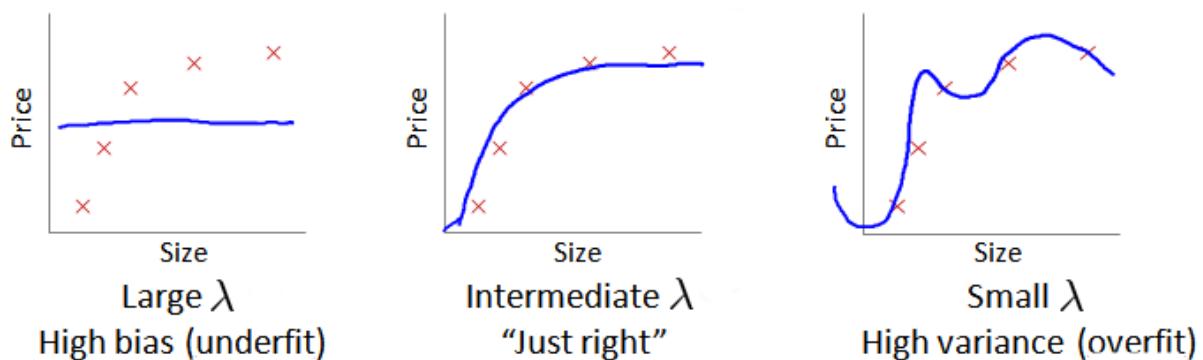
训练集误差和交叉验证集误差近似时：方差/欠拟合

交叉验证集误差远大于训练集误差时：偏差/过拟合

10.5 归一化和偏差/方差

参考视频: [10 - 5 - Regularization and Bias Variance \(11 min\).mkv](#)

在我们在训练模型的过程中,一般会使用一些归一化方法来防止过拟合。但是我们可能会归一化的程度太高或太小了,即我们在选择 λ 的值时也需要思考与刚才选择多项式模型次数类似的问题。



我们选择一系列的想要测试的 λ 值,通常是 0-10 之间的呈现 2 倍关系的值(如: 0,0.01,0.02,0.04,0.08,0.15,0.32,0.64,1.28,2.56,5.12,10 共 12 个)。我们同样把数据分为训练集、交叉验证集和测试集。

Choosing the regularization parameter λ

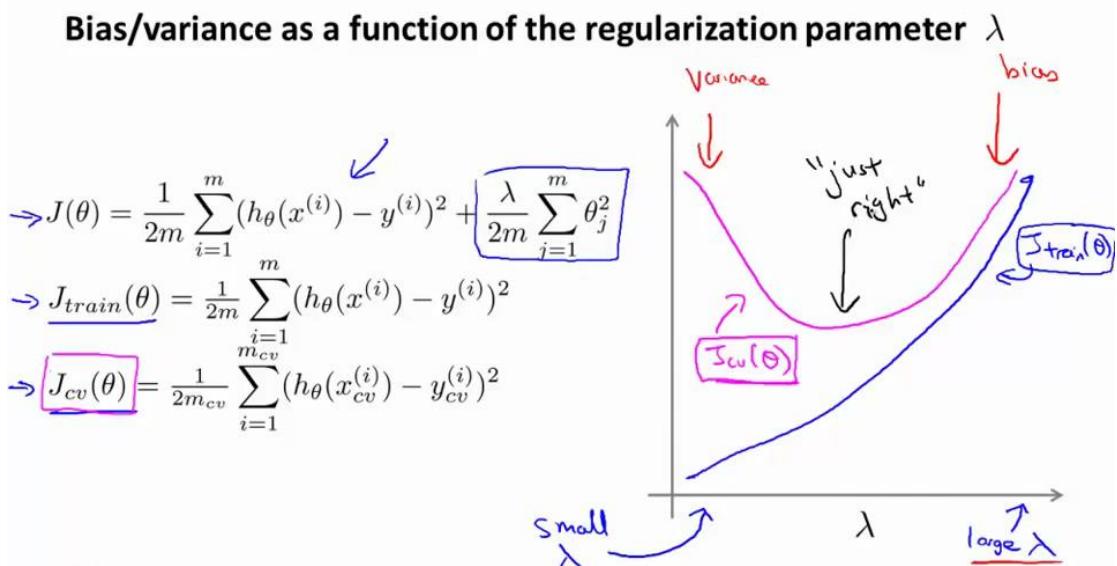
Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
 $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^m \theta_j^2$

1. Try $\lambda = 0$
2. Try $\lambda = 0.01$
3. Try $\lambda = 0.02$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$
- ⋮
12. Try $\lambda = 10$

选择 λ 的方法为:

1. 使用训练集训练出 12 个不同程度归一化的模型
2. 用 12 模型分别对交叉验证集计算的出交叉验证误差
3. 选择得出交叉验证误差**最小**的模型

4. 运用步骤 3 中选出模型对测试集计算得出推广误差，我们也可以同时将训练集和交叉验证集模型的代价函数误差与 λ 的值绘制在一张图表上：



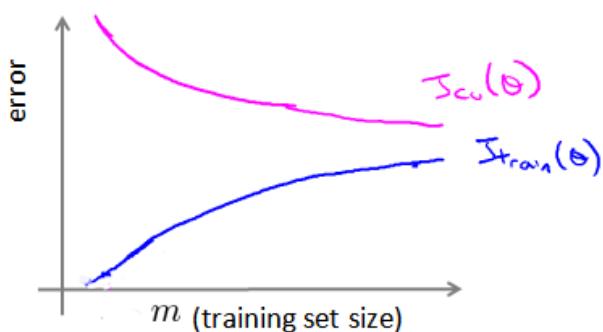
- 当 λ 较小时，训练集误差较小（过拟合）而交叉验证集误差较大
- 随着 λ 的增加，训练集误差不断增加（低拟合），而交叉验证集误差则是先减小后增加

10.6 学习曲线

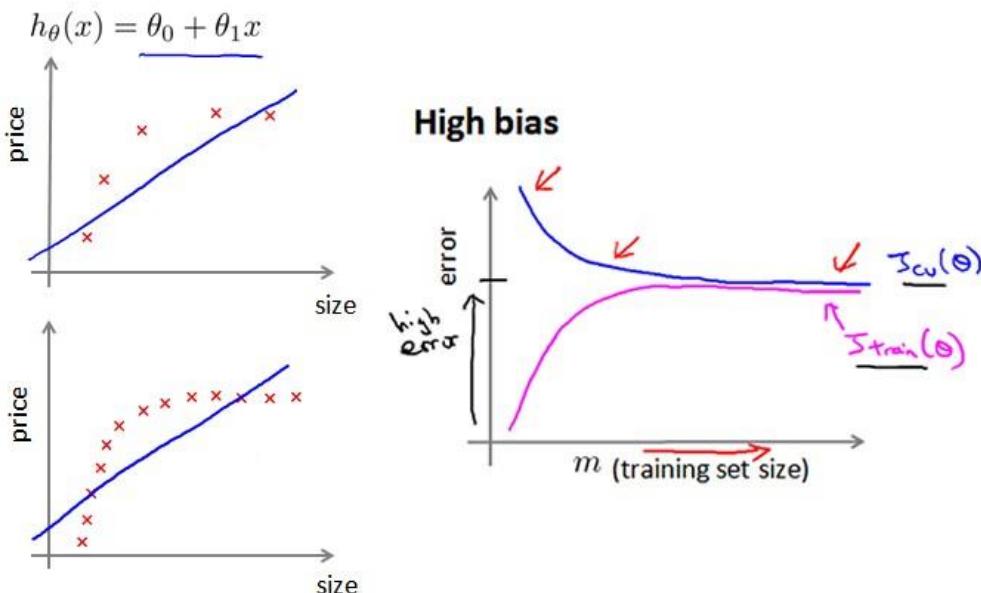
参考视频: [10 - 6 - Learning Curves \(12 min\).mkv](#)

学习曲线是学习算法的一个很好的合理检验 (sanity check)。学习曲线是将训练集误差和交叉验证集误差作为训练集实例数量 (m) 的函数绘制的图表。

即, 如果我们有 100 行数据, 我们从 1 行数据开始, 逐渐学习更多行的数据。思想是: 当训练较少行数据的时候, 训练的模型将能够非常完美地适应较少的训练数据, 但是训练出来的模型却不能很好地适应交叉验证集数据或测试集数据。



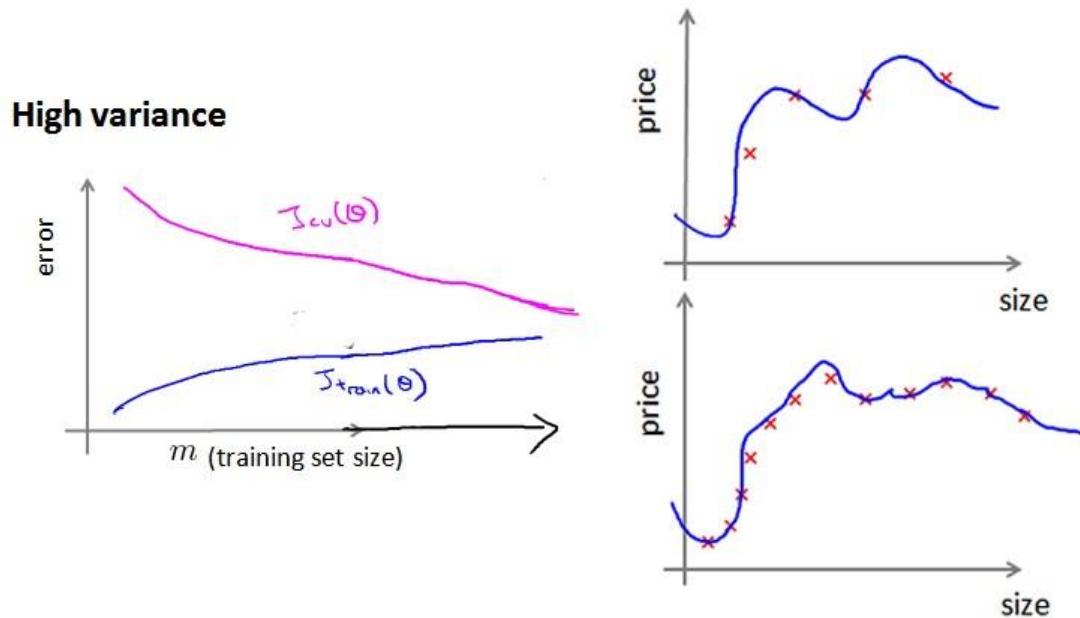
如何利用学习曲线识别高方差/欠拟合: 作为例子, 我们尝试用一条直线来适应下面的数据, 可以看出, 无论训练集有多么大误差都不会有太大改观:



也就是说在高方差/欠拟合的情况下, 增加数据到训练集不一定能有帮助。

如何利用学习曲线识别高方差/过拟合: 假设我们使用一个非常高次的多项式模型, 并且归一化非常小, 可以看出, 当交叉验证集误差远大于训练集误差时, 往训练集增加更多数

据可以提高模型的效果。



也就是说在高偏差/过拟合的情况下，增加更多数据到训练集可能可以提高算法效果。

10.7 决定下一步做什么

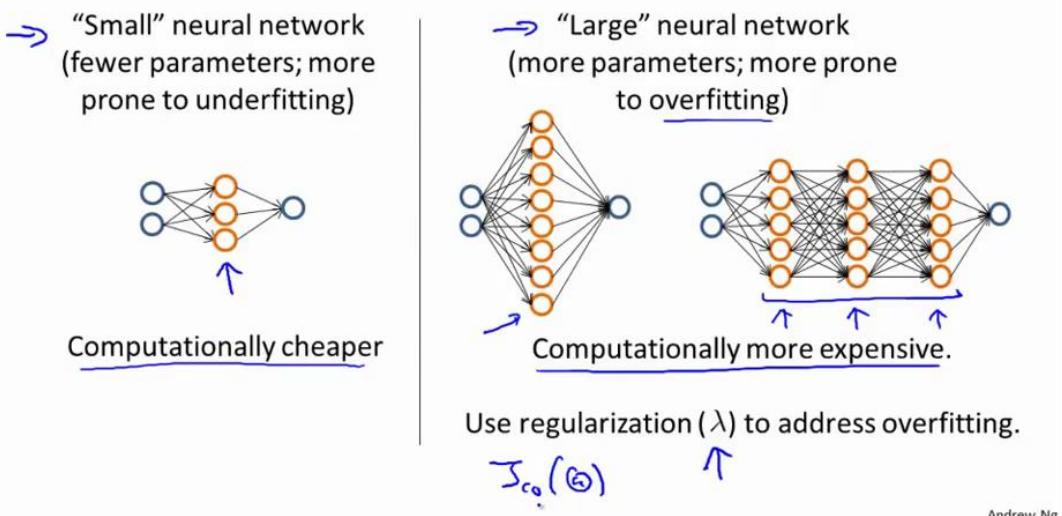
参考视频: [10 - 7 - Deciding What to Do Next Revisited \(7 min\).mkv](#)

回顾 1.1 中提出的六种可选的下一步, 让我们来看一看我们在什么情况下应该怎样选择:

1. 获得更多的训练实例——解决高偏差
2. 尝试减少特征的数量——解决高偏差
3. 尝试获得更多的特征——解决高方差
4. 尝试增加多项式特征——解决高方差
5. 尝试减少归一化程度 λ ——解决高方差
6. 尝试增加归一化程度 λ ——解决高偏差

神经网络的方差和偏差:

Neural networks and overfitting



使用较小的神经网络, 类似于参数较少的情况, 容易导致高方差和低拟合, 但计算代价较小。使用较大的神经网络, 类似于参数较多的情况, 容易导致高偏差和过拟合, 虽然计算代价比较大, 但是可以通过归一化手段来调整而更加适应数据。

通常选择较大的神经网络并采用归一化处理会比采用较小的神经网络效果要好。

对于神经网络中的隐藏层的层数的选择, 通常从一层开始逐渐增加层数, 为了更好地作选择, 可以把数据分为训练集、交叉验证集和测试集, 针对不同隐藏层层数的神经网络训

练神经网络， 然后选择交叉验证集代价最小的神经网络。

十一、机器学习系统的设计(Machine Learning System Design)

11.1 首先要做什么

参考视频: [11 - 1 - Prioritizing What to Work On \(10 min\).mkv](#)

本周以一个垃圾邮件分类器算法为例进行讨论。

为了解决这样一个问题，我们首先要做的决定是如何选择并表达特征向量 x 。我们可以选择一个由 100 个最常出现在垃圾邮件中的词所构成的列表，根据这些词是否有在邮件中出现，来获得我们的特征向量（出现为 1，不出现为 0），尺寸为 100×1 。

为了构建这个分类器算法，我们可以做很多事，例如：

1. 收集更多的数据，让我们有更多的垃圾邮件和非垃圾邮件的样本
2. 基于邮件的路由信息开发一系列复杂的特征
3. 基于邮件的正文信息开发一系列复杂的特征，包括考虑截词的处理
4. 为探测刻意的拼写错误（把 `watch` 写成 `w4tch`）开发复杂的算法

在上面这些选项中，非常难决定应该在哪一项上花费时间和精力，作出明智的选择比随着感觉走要更好。

11.2 误差分析

参考视频: [11 - 2 - Error Analysis \(13 min\).mkv](#)

误差分析可以帮助我们系统化地选择该做什么。

构建一个学习算法的推荐方法为:

1. 从一个简单的能快速实现的算法开始, 实现该算法并用交叉验证集数据测试这个算法
2. 绘制学习曲线, 决定是增加更多数据, 或者添加更多特征, 还是其他选择
3. 进行误差分析: 人工检查交叉验证集中我们算法中产生预测误差的实例, 看看这些实例是否有某种系统化的趋势

以我们的垃圾邮件过滤器为例, 误差分析要做的既是检验交叉验证集中我们的算法产生错误预测的所有邮件, 看: 是否能将这些邮件按照类分组。例如医药品垃圾邮件, 仿冒品垃圾邮件或者密码窃取 邮件等。然后看分类器对哪一组邮件的预测误差最大, 并着手优化。

思考怎样能改进分类器。例如, 发现是否缺少某些特征, 记下这些特征出现的次数。

例如记录下错误拼写出现了多少次, 异常的邮件路由情况出现了多少次等等, 然后从 出现次数最多的情况开始着手优化。

误差分析并不总能帮助我们判断应该采取怎样的行动。有时我们需要尝试不同的模型, 然后进行比较, 在模型比较时, 用数值来判断哪一个模型更好更有效, 通常我们是看交叉验证集的误差。

在 我 们 的 垃 圾 邮 件 分 类 器 例 子 中 , 对 于 “ 我 们 是 否 应 该 将 discount/discounts/discounted/discounting 处理成同一个词 ? ”如果这样做可以改善我们算法, 我们会采用一些截词软件。误差分析不能帮助我们做出这类判断, 我们只能尝试采用和不采用截词软件这两种不同方案, 然后根据数值检验的结果来判断哪一种更好。

11.3 类偏斜的误差度量

参考视频: [11 - 3 - Error Metrics for Skewed Classes \(12 min\).mkv](#)

类偏斜情况表现为我们的训练集中有非常多的同一种类的实例，只有很少或没有其他类的实例。

例如我们希望用算法来预测癌症是否是恶性的，在我们的训练集中，只有 0.5% 的实例是恶性肿瘤。假设我们编写一个非学习而来的算法，在所有情况下都预测肿瘤是良性的，那么误差只有 0.5%。然而我们通过训练而得到的神经网络算法却有 1% 的误差。这时，误差的大小是不能视为评判算法效果的依据的。

查准率（Precision）和查全率（Recall） 我们将算法预测的结果分成四种情况：

1. 正确肯定 (True Positive, TP): 预测为真，实际为真
2. 正确否定 (True Negative, TN): 预测为假，实际为真
3. 错误肯定 (False Positive, FP): 预测为真，实际为假
4. 错误否定 (False Negative, FN): 预测为假，实际为假

则：

查准率=TP/(TP+FP) 例，在所有我们预测有恶性肿瘤的病人中，实际上有恶性肿瘤的病人的百分比，越高越好。

查全率=TP/(TP+FN) 例，在所有实际上有恶性肿瘤的病人中，成功预测有恶性肿瘤的病人的百分比，越高越好。

这样，对于我们刚才那个总是预测病人肿瘤为良性的算法，其查全率是 0。

11.4 查全率和查准率之间的权衡

参考视频: [11 - 4 - Trading Off Precision and Recall \(14 min\).mkv](#)

继续沿用刚才预测肿瘤性质的例子。假使，我们的算法输出的结果在 0-1 之间，我们使用阀值 0.5 来预测真和假。

Trading off precision and recall

→ Logistic regression: $0 \leq h_\theta(x) \leq 1$

Predict 1 if $h_\theta(x) \geq 0.5$

Predict 0 if $h_\theta(x) < 0.5$

Suppose we want to predict $y = 1$ (cancer)
only if very confident.

$$\rightarrow \text{precision} = \frac{\text{true positives}}{\text{no. of predicted positive}}$$

$$\rightarrow \text{recall} = \frac{\text{true positives}}{\text{no. of actual positive}}$$

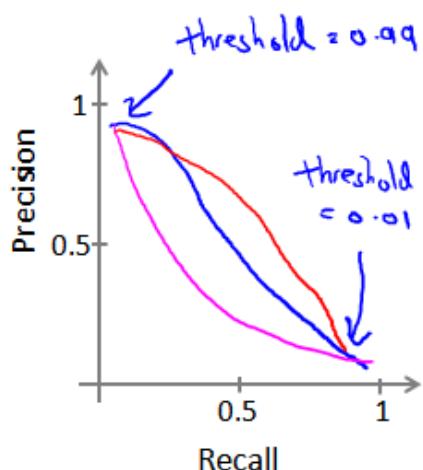
查准率 (Precision) =TP/ (TP+FP) 例，在所有我们预测有恶性肿瘤的病人中，实际上有恶性肿瘤的病人的百分比，越高越好。

查全率 (Recall) =TP/ (TP+FN) 例，在所有实际上有恶性肿瘤的病人中，成功预测有恶性肿瘤的病人的百分比，越高越好。

如果我们希望只在非常确信的情况下预测为真（肿瘤为恶性），即我们希望更高的查准率，我们可以使用比 0.5 更大的阀值，如 0.7, 0.9。这样做我们会减少错误预测病人为恶性肿瘤的情况，同时却会增加未能成功预测肿瘤为恶性的情况。

如果我们希望提高查全率，尽可能地让所有有可能是恶性肿瘤的病人都得到进一步地检查、诊断，我们可以使用比 0.5 更小的阀值，如 0.3。

我们可以将不同阀值情况下，查全率与查准率的关系绘制成图表，曲线的形状根据数据的不同而不同：



我们希望有一个帮助我们选择这个阈值的方法。一种方法是计算 **F₁** 值 (F1 Score)，其计算公式为：

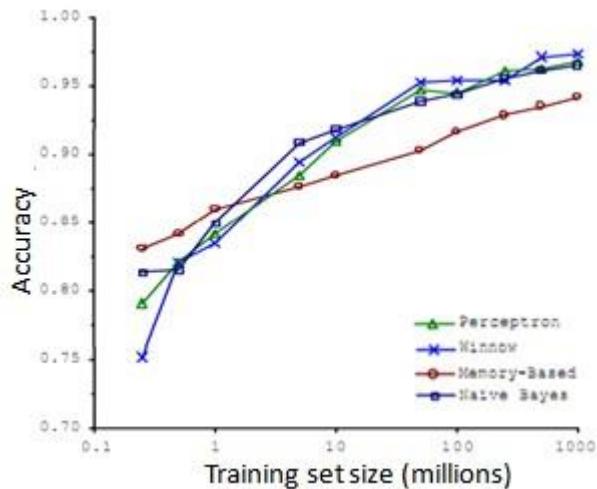
$$F_1 \text{ Score: } 2 \frac{P R}{P+R}$$

我们选择使得 F1 值最高的阈值。

11.5 机器学习的数据

参考视频: [11 - 5 - Data For Machine Learning \(11 min\).mkv](#)

Bank 和 Brill 的尝试通过机器学习算法来区分常见的易混淆的单词，他们尝试了许多种不同的算法，并发现数据量非常大时，这些不同类型的算法效果都很好。我们下面希望探讨，什么时候我们会希望获得更多数据，而非修改算法。



通常情况下，首先思考这样一个问题，“在这些特征面前，一个真人专家是否能有信心地预测结果？”如果回答是肯定的，我们需要再思考我们的模型是怎样的。如果算法是高偏差的，且代价函数很小，那么增加训练集的数据量不太可能导致过拟合，可使得交叉验证误差和训练误差之间差距更小。这种情况下，考虑获得更多数据。

也可以这样来认识，我们希望我们的算法低方差，低偏差，我们通过选择更多的特征来降低方差，再通过增加数据量来降低偏差。

第 7 周

十二、支持向量机(Support Vector Machines)

12.1 优化目标

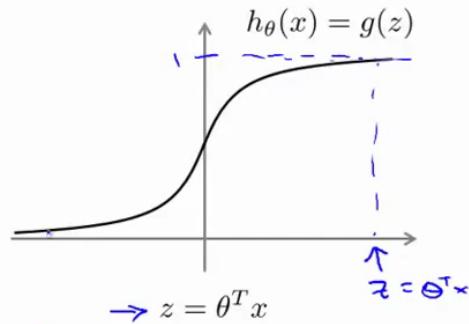
参考视频: [12 - 1 - Optimization Objective \(15 min\).mkv](#)

到目前为止,你已经见过一系列不同的学习算法。在监督学习中,许多学习算法的性能都非常类似,因此,重要的不是你该选择使用学习算法 A 还是学习算法 B,而更重要的是,应用这些算法时,所创建的大量数据在应用这些算法时,表现情况通常依赖于你的水平。比如:你为学习算法所设计的特征量的选择,以及如何选择正则化参数,诸如此类的事。还有一个更加强大的算法广泛的应用于工业界和学术界,它被称为支持向量机(Support Vector Machine)。与逻辑回归和神经网络相比,支持向量机,或者简称 SVM。在学习复杂的非线性方程时提供了一种更为清晰,更加强大的方式。因此,在接下来的视频中,我会探讨这一算法。在稍后的课程中,我也会对监督学习算法进行简要的总结。当然,仅仅是作简要描述。但对于支持向量机,鉴于该算法的强大和受欢迎度,在本课中,我会花许多时间来讲解它。它也是我们所介绍的最后一个监督学习算法。

正如我们之前开发的学习算法,我们从优化目标开始。那么,我们开始学习这个算法。为了描述支持向量机,事实上,我将会从逻辑回归开始展示我们如何一点一点修改来得到本质上支持向量机。

Alternative view of logistic regression

$$\rightarrow h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If $y = 1$, we want $h_{\theta}(x) \approx 1$, $\theta^T x \gg 0$
 If $y = 0$, we want $h_{\theta}(x) \approx 0$, $\theta^T x \ll 0$

Andrew Ng

那么，在逻辑回归中我们已经熟悉了这里的假设函数形式，和右边的 S 型激励函数。然而，为了解释一些数学知识，我将用 z 表示 $\theta^T x$ 。

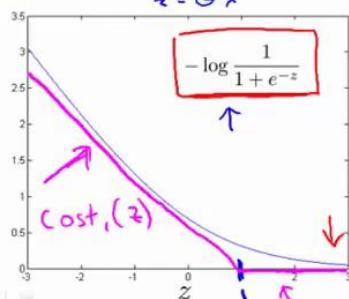
现在考虑下我们想要逻辑回归做什么：如果有一个 $y=1$ 的样本，我的意思是不管是在训练集中或是在测试集中，又或者在交叉验证集中，总之是 $y=1$ ，现在我们希望 $h(x)$ 趋近 1。因为我们想要正确地将此样本分类，这就意味着当 $h(x)$ 趋近于 1 时， $\theta^T x$ 应当远大于 0，这里的“>>”意思是远远大于 0。这是因为由于 z 表示 $\theta^T x$ ，当 z 远大于 0 时，即到了该图的右边，你不难发现此时逻辑回归的输出将趋近于 1。相反地，如果我们有另一个样本，即 $y=0$ 。我们希望假设函数的输出值将趋近于 0，这对应于 $\theta^T x$ ，或者就是 z 会远小于 0，因为对应的假设函数的输出值趋近于 0。

Alternative view of logistic regression

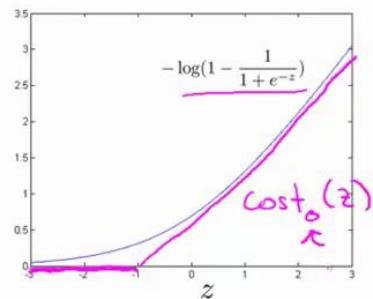
Cost of example: $-(y \log h_\theta(x) + (1-y) \log(1 - h_\theta(x))) \leftarrow$

$$= \boxed{-y \log \frac{1}{1 + e^{-\theta^T x}}} - \boxed{(1-y) \log(1 - \frac{1}{1 + e^{-\theta^T x}})} \leftarrow$$

If $y = 1$ (want $\theta^T x \gg 0$):
 $z = \theta^T x$



If $y = 0$ (want $\theta^T x \ll 0$):



如果你进一步观察逻辑回归的代价函数，你会发现每个样本 (x, y) 都会为总代价函数增加这里的一项，因此，对于总代价函数通常会有对所有的训练样本求和，并且这里还有一个 $1/m$ 项，但是，在逻辑回归中，这里的这一项就是表示一个训练样本所对应的表达式。现在，如果我将完整定义的假设函数代入这里。那么，我们就会得到每一个训练样本都影响这一项。

现在，先忽略 $1/m$ 这一项，但是这一项是影响整个总代价函数中的这一项的。现在，一起来考虑两种情况：一种是 y 等于 1 的情况；另一种是 y 等于 0 的情况。在第一种情况中，假设 y 等于 1，此时在目标函数中只需有第一项起作用，因为 y 等于 1 时， $(1-y)$ 项将等于 0。因此，当在 y 等于 1 的样本中时，即在 (x, y) 中 y 等于 1，我们得到 $-\log(1 - \frac{1}{1 + e^{-z}})$

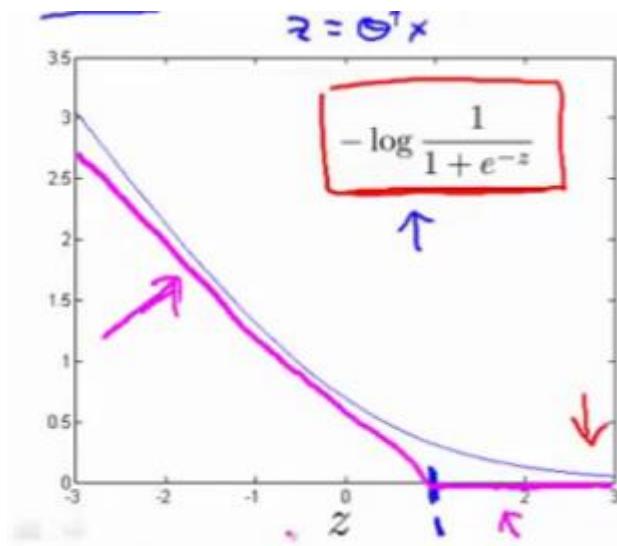
这样一项，这里同上一张幻灯片一致。

我用 z 表示 $\theta^T x$ 。当然，在代价函数中， y 前面有负号。我们只是这样表示，如果 y 等于 1 代价函数中，这一项也等于 1。这样做是为了简化此处的表达式。如果画出关于 z 的函数，你会看到左下角的这条曲线，我们同样可以看到，当 z 增大时，也就是相当于 $\theta^T x$ 增大时， z 对应的值会变的非常小。对整个代价函数而言，影响也非常小。这也就解释了，为什么逻辑回归在观察到正样本 $y=1$ 时，试图将 $\theta^T x$ 设置得非常大。因为在代价函数中的这一项会变的非常小。

现在开始建立支持向量机，我们从这里开始：

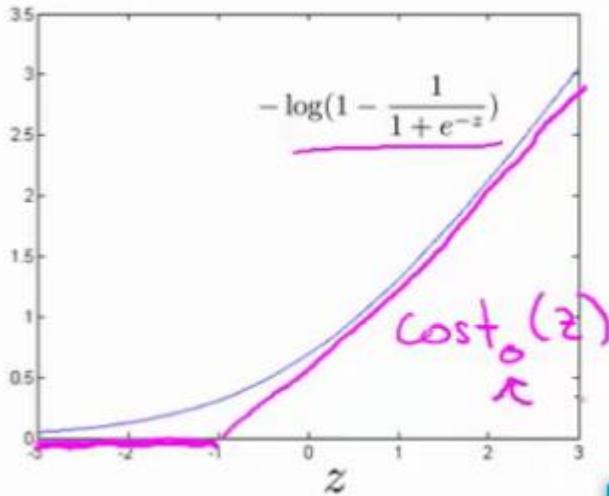
我们会从这个代价函数开始，也就是 $-\log\left(1 - \frac{1}{1+e^{-z}}\right)$ 一点一点修改，让我取这里的

$z=1$ 点，我先画出将要用的代价函数。



新的代价函数将会水平的从这里到右边（图外），然后我再画一条同逻辑回归非常相似的直线，但是，在这里是一条直线，也就是我用紫红色画的曲线，就是这条紫红色的曲线。那么，到了这里已经非常接近逻辑回归中使用的代价函数了。只是这里是由两条线段组成，即位于右边的水平部分和位于左边的直线部分，先别过多的考虑左边直线部分的斜率，这并不是很重要。但是，这里我们将使用的新的代价函数，是在 $y=1$ 的前提下的。你也许能想到，这应该能做同逻辑回归中类似的事情，但事实上，在之后的优化问题中，这会变得更坚定，并且为支持向量机，带来计算上的优势。例如，更容易计算股票交易的问题等等。

目前，我们只是讨论了 $y=1$ 的情况，另外一种情况是当 $y=0$ 时，此时如果你仔细观察代价函数只留下了第二项，因为第一项被消除了。如果当 $y=0$ 时，那么这一项也就是 0 了。所以上述表达式只留下了第二项。因此，这个样本的代价或是代价函数的贡献。将会由这一项表示。并且，如果你将这一项作为 z 的函数，那么，这里就会得到横轴 z 。现在，你完成了支持向量机中的部分内容，同样地，我们要替代这一条蓝色的线，用相似的方法。



如果我们用一个新的代价函数来代替，即这条从 0 点开始的水平直线，然后是一条斜线，像上图。那么，现在让我给这两个方程命名，左边的函数，我称之为 $\text{cost}_1(\theta^T x^{(i)})$ ，同时，右边函数我称它为 $\text{cost}_0(\theta^T x^{(i)})$ 。这里的下标是指在代价函数中，对应的 $y=1$ 和 $y=0$ 的情况，拥有了这些定义后，现在，我们就开始构建支持向量机。

Support vector machine

Logistic regression:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \underbrace{\left(-\log h_{\theta}(x^{(i)}) \right)}_{\text{cost}_1(\theta^T x^{(i)})} + (1-y^{(i)}) \underbrace{\left(-\log(1-h_{\theta}(x^{(i)})) \right)}_{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Support vector machine:

$$\begin{aligned} \min_{\theta} & C \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^n \theta_j^2 \\ & \min_u \frac{(u-s)^2 + 1}{2} \rightarrow u=5 \\ & \min_u \log(u-s)^2 + 1 \rightarrow u=5 \end{aligned}$$

$$\begin{aligned} & \xrightarrow{\text{A} + \lambda \text{B} \leftarrow} \\ & \xrightarrow{\text{C} = \frac{1}{\lambda}} \end{aligned}$$

$$\Rightarrow \min_{\theta} C \sum_{i=1}^m [y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

这是我们在逻辑回归中使用代价函数 $J(\theta)$ 。也许这个方程看起来不是非常熟悉。这是因为之前有个负号在方程外面，但是，这里我所做的是，将负号移到了表达式的里面，这样做使得方程看起来有些不同。对于支持向量机而言，实质上我们要将这替换为 $\text{cost}_1(\theta^T x)$ ，也就是 $\text{cost}_1(\theta^T x)$ ，同样地，我也将这一项替换为 $\text{cost}_0(\theta^T x)$ ，也就是代价 $\text{cost}_0(\theta^T x)$ 。这里的代价函数 cost_1 ，就是之前所提到的那条线。此外，代价函数 cost_0 ，也是上面所介绍

过的那条线。因此，对于支持向量机，我们得到了这里的最小化问题，即：

$$\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \underbrace{\left(-\log h_\theta(x^{(i)}) \right)}_{\text{Cost}_1(\theta^\top x^{(i)})} + (1-y^{(i)}) \underbrace{\left(-\log(1-h_\theta(x^{(i)})) \right)}_{\text{Cost}_0(\theta^\top x^{(i)})} \right]$$

然后，再加上正则化参数。现在，按照支持向量机的惯例，事实上，我们的书写会稍微有些不同，代价函数的参数表示也会稍微有些不同。

首先，我们要除去 $1/m$ 这一项，当然，这仅仅是由于人们使用支持向量机时，对比于逻辑回归而言，不同的习惯所致，但这里我所说的意思是：你知道，我将要做的是仅仅除去 $1/m$ 这一项，但是，这也会得出同样的 θ 最优值，好的，因为 $1/m$ 仅是个常量，因此，你知道在这个最小化问题中，无论前面是否有 $1/m$ 这一项，最终我所得到的最优值 θ 都是一样的。这里我的意思是，先给你举一个实例，假定有一最小化问题：即要求当 $(u-5)^2+1$ 取得最小值时的 u 值，这时最小值为：当 $u=5$ 时取得最小值。

现在，如果我们想要将这个目标函数乘上常数 10，这里我的最小化问题就变成了：求使得 $10 \times (u-5)^2+10$ 最小的值 u ，然而，使得这里最小的 u 值仍为 5。因此将一些常数乘以你的最小化项，这并不会改变最小化该方程时得到 u 值。因此，这里我所做的就是删去常量 m 。也相同的，我将目标函数乘上一个常量 m ，并不会改变取得最小值时的 θ 值。

第二点概念上的变化，我们只是指在使用，支持向量机时，一些如下的标准惯例，而不是逻辑回归。因此，对于逻辑回归，在目标函数中，我们有两项：第一个是训练样本的代价，第二个是我们的正则化项，我们不得不去用这一项来平衡。这就相当于我们想要最小化 A 加上正则化参数 λ ，然后乘以其他项 B 对吧？这里的 A 表示这里的第一项，同时我用 B 表示第二项，但不包括 λ ，我们不是优化这里的 $A + \lambda \times B$ 。我们所做的就是通过设置不同正则参数 λ 达到优化目的。这样，我们就能够权衡对应的项，是使得训练样本拟合的更好。即最小化 A 。还是保证正则参数足够小，也即是对于 B 项而言，但对于支持向量机，按照惯例，我们将使用一个不同的参数替换这里使用的 λ 来权衡这两项。你知道，就是第一项和第二项我们依照惯例使用一个不同的参数称为 C ，同时改为优化目标， $C \times A + B$ 因此，在逻辑回归中，如果给定 λ ，一个非常大的值，意味着给予 B 更大的权重。而这里，就对应于将 C 设定为非常小的值，那么，相应的将会给 B 比给 A 更大的权重。因此，这只不过是一种不同的方式来控制这种权衡或者一种不同的方法，即用参数来决定是更关心第一项的优化，还是更关心第二项的优化。当然你也可以把这里的参数 C 考虑成 $1/\lambda$ ，同 $1/\lambda$ 所扮演的角色相同，并且这两个方程或这两个表达式并不相同，因为 C 等于 $1/\lambda$ ，但是也并不全是这样，如果当 C 等

于 $1/\lambda$ 时，这两个优化目标应当得到相同的值，相同的最优值 θ 。因此，就用它们来代替。那么，我现在删掉这里的 λ ，并且用常数 C 来代替。因此，这就得到了在支持向量机中我们的整个优化目标函数。然后最小化这个目标函数，得到 SVM 学习到的参数 C 。

SVM hypothesis

$$\Rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Hypothesis:

$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

最后有别于逻辑回归输出的概率。在这里，我们的代价函数，当最小化代价函数，获得参数 θ 时，支持向量机所做的是它来直接预测 y 的值等于 1，还是等于 0。因此，这个假设函数会预测 1。当 $\theta^T x$ 大于或者等于 0 时，或者等于 0 时，所以学习参数 θ 就是支持向量机假设函数的形式。那么，这就是支持向量机数学上的定义。

在接下来的视频中，让我们再回去从直观的角度看看优化目标，实际上是在做什么，以及 SVM 的假设函数将会学习什么，同时也会谈谈如何做些许修改，学习更加复杂、非线性的函数。

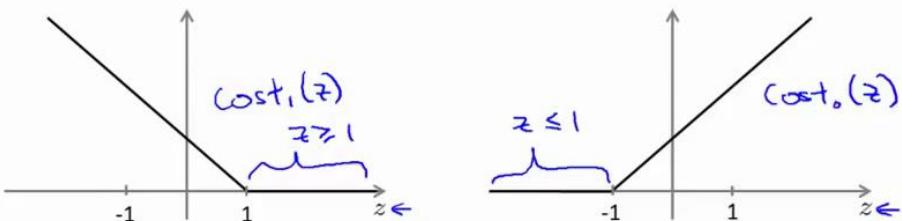
12.2 大边界的直观理解

参考视频: [12 - 2 - Large Margin Intuition \(11 min\).mkv](#)

人们有时将支持向量机看做是大间距分类器。在这一部分, 我将介绍其中的含义, 这有助于我们直观理解 SVM 模型的假设是什么样的。

Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \underline{\text{cost}_1(\theta^T x^{(i)})} + (1 - y^{(i)}) \underline{\text{cost}_0(\theta^T x^{(i)})} \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



- If $y = 1$, we want $\underline{\theta^T x \geq 1}$ (not just ≥ 0)
- If $y = 0$, we want $\underline{\theta^T x \leq -1}$ (not just < 0)

这是我的支持向量机模型的代价函数, 在左边这里我画出了关于 z 的代价函数 $\text{cost}_1(z)$, 此函数用于正样本, 而在右边这里我画出了关于 z 的代价函数 $\text{cost}_0(z)$, 横轴表示 z , 现在让我们考虑一下, 最小化这些代价函数的必要条件是什么。如果你有一个正样本, y 等于 1, 则只有在 z 大于等于 1 时, 代价函数 $\text{cost}_1(z)$ 才等于 0。换句话说, 如果你有一个正样本, 我们会希望 $\underline{\theta^T x \geq 1}$, 反之, 如果 y 是等于 0 的, 我们观察一下, 函数 $\text{cost}_0(z)$, 它只有在 $z \leq 1$ 的区间里函数值为 0。这是支持向量机的一个有趣性质。事实上, 如果你有一个正样本 y 等于 1, 则其实我们仅仅要求 $\underline{\theta^T x \geq 0}$, 就能将该样本恰当分出, 这是因为如果 $\underline{\theta^T x > 0}$

大的话, 我们的模型代价函数值为 0, 类似地, 如果你有一个负样本, 则仅需要 $\underline{\theta^T x \leq 0}$ 就会将负例正确分离, 但是, 支持向量机的要求更高, 不仅仅要能正确分开输入的样本, 即不仅仅要求 $\underline{\theta^T x > 0}$, 我们需要的是比 0 值大很多, 比如大于等于 1, 我也想这个比 0 小很多, 比如我希望它小于等于 -1, 这就相当于在支持向量机中嵌入了一个额外的安全因子。或者说

安全的间距因子。

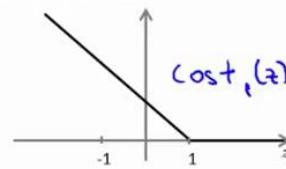
当然，逻辑回归做了类似的事情。但是让我们看一下，在支持向量机中，这个因子会导至什么结果。具体而言，我接下来会考虑一个特例。我们将这个常数 C 设置成一个非常大的值。比如我们假设 C 的值为 100000 或者其它非常大的数，然后来观察支持向量机会给出什么结果？

SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

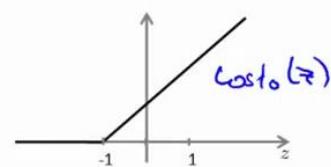
Whenever $y^{(i)} = 1$:

$$\theta^T x^{(i)} \geq 1$$



Whenever $y^{(i)} = 0$:

$$\theta^T x^{(i)} \leq -1$$



如果 C 非常大，则最小化代价函数的时候，我们将会很希望找到一个使第一项为 0 的最优解。因此，让我们尝试在代价项的第一项为 0 的情形下理解该优化问题。比如我们可以把 C 设置成了非常大的常数，这将给我们一些关于支持向量机模型的直观感受。

$$\min_{\theta} C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

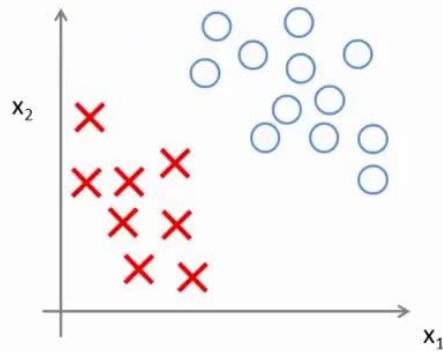
我们已经看到输入一个训练样本标签为 $y=1$ ，你想令第一项为 0，你需要做的是找到一个 θ ，使得 $\theta^T x >= 1$ ，类似地，对于一个训练样本，标签为 $y=0$ ，为了使 $\text{cost}_0(z)$ 函数的值

为 0，我们需要 $\theta^T x <=-1$ 。因此，现在考虑我们的优化问题。选择参数，使得第一项等于 0，

就会导致下面的优化问题，因为我们将选择参数使第一项为 0，因此这个函数的第一项为 0，因此是 C 乘以 0 加上二分之一乘以第二项。这里第一项是 C 乘以 0，因此可以将其删去，因为我知道它是 0。

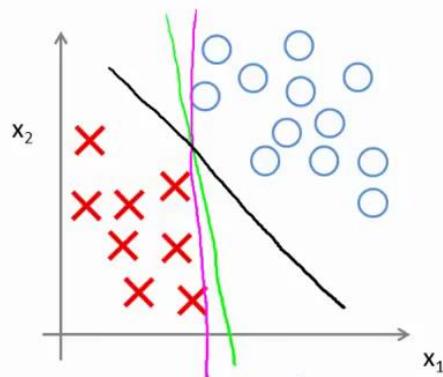
这将遵从以下的约束： $\theta^T x^{(i)} >= 1$ ，如果 $y^{(i)}$ 是等于 1 的， $\theta^T x^{(i)} <=-1$ ，如果样本 i 是一个负样本，这样当你求解这个优化问题的时候，当你最小化这个关于变量 θ 的函数的时候，你会得到一个非常有趣的决策边界。

SVM Decision Boundary: Linearly separable case



具体而言，如果你考察这样一个数据集，其中有正样本，也有负样本，可以看到这个数据集是线性可分的。我的意思是，存在一条直线把正负样本分开。当然有多条不同的直线，可以把正样本和负样本完全分开。

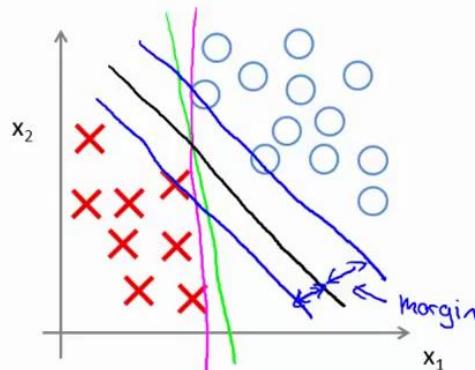
SVM Decision Boundary: Linearly separable case



比如，这就是一个决策边界可以把正样本和负样本分开。但是多多少少这个看起来并不是非常自然是么？

或者我们可以画一条更差的决策界，这是另一条决策边界，可以将正样本和负样本分开，但仅仅是勉强分开，这些决策边界看起来都不是特别好的选择，支持向量机将会选择这个黑色的决策边界，相较于之前我用粉色或者绿色画的决策界。这条黑色的看起来好得多，黑线看起来是更稳健的决策界。在分离正样本和负样本上它显得的更好。数学上来讲，这是什么意思呢？这条黑线有更大的距离，这个距离叫做间距 (margin)。

SVM Decision Boundary: Linearly separable case



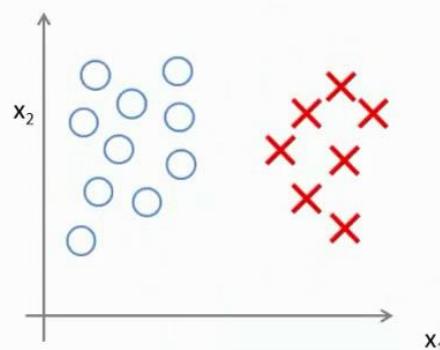
Large margin classifier

当画出这两条额外的蓝线，我们看到黑色的决策界和训练样本之间有更大的最短距离。然而粉线和蓝线离训练样本就非常近，在分离样本的时候就会比黑线表现差。因此，这个距离叫做支持向量机的间距，而这是支持向量机具有鲁棒性的原因，因为它努力用一个最大间距来分离样本。因此支持向量机有时被称为**大间距分类器**，而这其实是求解上一页幻灯片上优化问题的结果。

我知道你也许想知道求解上一页幻灯片中的优化问题为什么会产生这个结果？它是如何产生这个大间距分类器的呢？我知道我还没有解释这一点。

我将会从直观上略述为什么这个优化问题会产生大间距分类器。总之这个图示有助于你理解支持向量机模型的做法，即努力将正样本和负样本用最大的间距分开。

Large margin classifier in presence of outliers



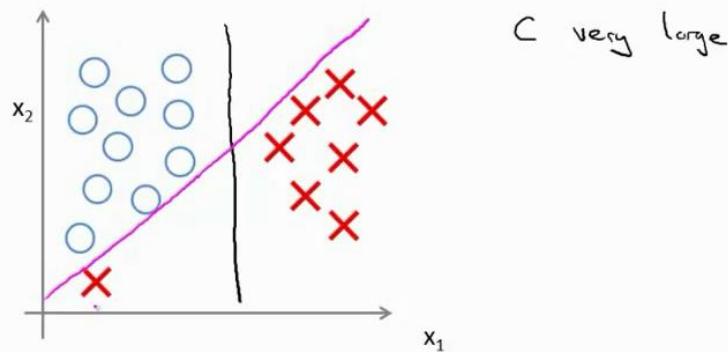
在本节课中关于大间距分类器，我想讲最后一点：我们将这个大间距分类器中的正则化因子常数 C 设置的非常大，我记得我将其设置为了 100000，因此对这样的一个数据集，也许我们将选择这样的决策界，从而最大间距地分离开正样本和负样本。那么在让代价函数最小化的过程中，我们希望找出在 $y=1$ 和 $y=0$ 两种情况下都使得代价函数中左边的这一项尽

量为零的参数。如果我们找到了这样的参数，则我们的最小化问题便转变成：

$$\min \frac{1}{2} \sum_{j=1}^n \theta_j^2 \quad s.t. \begin{cases} \theta^T x^{(i)} \geq 1 & \text{if } y^{(i)} = 1 \\ \theta^T x^{(i)} \leq -1 & \text{if } y^{(i)} = 0 \end{cases}$$

事实上，支持向量机现在要比这个大间距分类器所体现得更成熟，尤其是当你使用大间距分类器的时候，你的学习算法会受异常点 (outlier) 的影响。比如我们加入一个额外的正样本。

Large margin classifier in presence of outliers



在这里，如果你加了这个样本，为了将样本用最大间距分开，也许我最终会得到一条类似这样的决策界，对么？就是这条粉色的线，仅仅基于一个异常值，仅仅基于一个样本，就将我的决策界从这条黑线变到这条粉线，这实在是不明智的。而如果正则化参数 C ，设置的非常大，这事实上正是支持向量机将会做的。它将决策界，从黑线变到了粉线，但是如果 C 设置的小一点，**如果你将 C 设置的不要太大，则你最终会得到这条黑线**，当然数据如果不是线性可分的，如果你在这里有一些正样本或者你在这里有一些负样本，则支持向量机也会将它们恰当分开。因此，大间距分类器的描述，仅仅是从直观上给出了正则化参数 C 非常大的情形，同时，要提醒你 C 的作用类似于 $1/\lambda$ ， λ 是我们之前使用过的正则化参数。这只是 C 非常大的情形，或者等价地 λ 非常小的情形。你最终会得到类似粉线这样的决策界，但是实际上应用支持向量机的时候，**当 C 不是非常非常大的时候，它可以忽略掉一些异常点的影响，得到更好的决策界**。甚至当你的数据不是线性可分的时候，支持向量机也可以给出好的结果。

回顾 $C=1/\lambda$ ，因此：

C 较大时，相当于 λ 较小，可能会导致过拟合，高方差。

C 较小时，相当于 λ 较大，可能会导致低拟合，高偏差。

我们稍后会介绍支持向量机的偏差和方差，希望在那时候关于如何处理参数的这种平衡

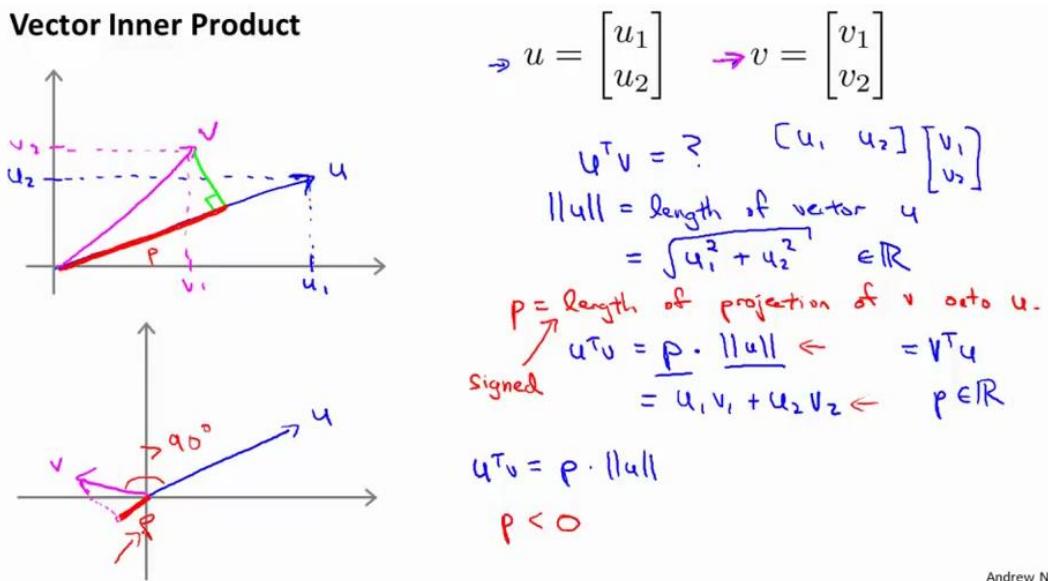
会变得更加清晰。我希望，这节课给出了一些关于为什么支持向量机被看做大间距分类器的直观理解。它用最大间距将样本区分开，尽管从技术上讲，这只有当参数 C 是非常大的时候是真的，但是它对于理解支持向量机是有益的。

本节课中我们略去了一步，那就是我们在幻灯片中给出的优化问题。为什么会是这样的？它是如何得出大间距分类器的？我在本节中没有讲解，在下一节课中，我将略述这些问题背后的数学原理，来解释这个优化问题是如何得到一个大间距分类器的。

12.3 数学背后的大边界分类（可选）

参考视频: [12 - 3 - Mathematics Behind Large Margin Classification \(Optional\) \(20 min\).mkv](#)

在本节课中，我将介绍一些大间隔分类背后的数学原理。本节为选学部分，你完全可以跳过它，但是听听这节课可能让你对支持向量机中的优化问题，以及如何得到大间距分类器，产生更好的直观理解。

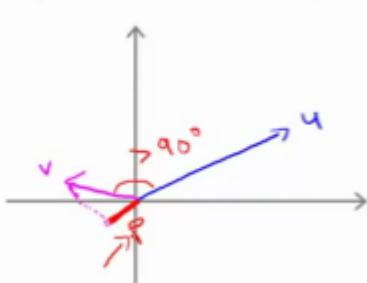


Andrew Ng

首先，让我来给大家复习一下关于向量内积的知识。假设我有两个向量， u 和 v 我将它们写在这里。两个都是二维向量，我们看一下， $u^T v$ 的结果。 $u^T v$ 也叫做向量 u 和 v 之间的内积。由于是二维向量，我可以将它们画在这个图上。我们说，这就是向量 u 即在横轴上，取值为某个 u_1 ，而在纵轴上，高度是某个 u_2 作为 u 的第二个分量。现在，很容易计算的一个量就是向量 u 的范数。 $\|u\|$ 表示 u 的范数，即 u 的长度，即向量 u 的欧几里得长度。根据毕达哥拉斯定理， $\|u\| = \sqrt{u_1^2 + u_2^2}$ ，这是向量 u 的长度，它是一个实数。现在你知道了这个的长度是多少了。我刚刚画的这个向量的长度就知道了。

现在让我们回头来看向量 v ，因为我们想计算内积。 v 是另一个向量，它的两个分量 v_1 和 v_2 是已知的。向量 v 可以画在这里，现在让我们来看看如何计算 u 和 v 之间的内积。这就是具体做法，我们将向量 v 投影到向量 u 上，我们做一个直角投影，或者说一个 90 度投影将其投影到 u 上，接下来我度量这条红线的长度。我称这条红线的长度为 p ，因此 p 就

是长度，或者说是向量 v 投影到向量 u 上的量，我将它写下来， p 是 v 投影到向量 u 上的长度，因此可以将 $u^T v = p \cdot \|u\|$ ，或者说 u 的长度。这是计算内积的一种方法。如果你从几何上画出 p 的值，同时画出 u 的范数，你也会同样地计算出内积，答案是一样的。另一个计算公式是： $u^T v$ 就是 $[u_1 \ u_2]$ 这个一行两列的矩阵乘以 v 。因此可以得到 $u_1 \times v_1 + u_2 \times v_2$ 。根据线性代数的知识，这两个公式会给出同样的结果。顺便说一句， $u^T v = v^T u$ 。因此如果你将 u 和 v 交换位置，将 u 投影到 v 上，而不是将 v 投影到 u 上，然后做同样地计算，只是把 u 和 v 的位置交换一下，你事实上可以得到同样的结果。申明一点，在这个等式中 u 的范数是一个实数， p 也是一个实数，因此 $u^T v$ 就是两个实数正常相乘。

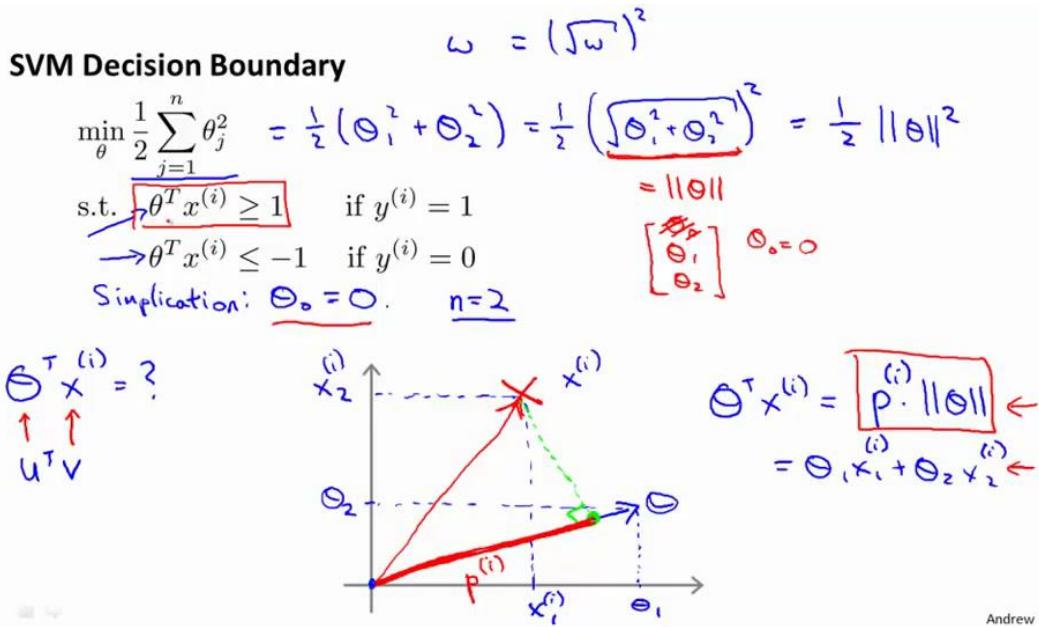


最后一点，需要注意的就是 p 值， p 事实上是有符号的，即它可能是正值，也可能是负值。我的意思是说，如果 u 是一个类似这样的向量， v 是一个类似这样的向量， u 和 v 之间的夹角大于 90 度，则如果将 v 投影到 u 上，会得到这样的一个投影，这是 p 的长度，在这个情形下我们仍然有 $u^T v$ 等于 p 乘以 u 的范数。唯一一点不同的是 p 在这里是负的。在内积计算中，如果 u 和 v 之间的夹角小于 90 度，那么那条红线的长度 p 是正值。然而如果这个夹角大于 90 度，则 p 将会是负的。就是这个小线段的长度是负的。如果它们之间的夹角大于 90 度，两个向量之间的内积也是负的。这就是关于向量内积的知识。我们接下来将会使用这些关于向量内积的性质试图来理解支持向量机中的目标函数。

SVM Decision Boundary

$$\begin{aligned} & \min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 \\ \text{s.t. } & \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1 \\ & \theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0 \end{aligned}$$

这就是我们先前给出的支持向量机模型中的目标函数。为了讲解方便，我做一点简化，仅仅是为了让目标函数更容易被分析。



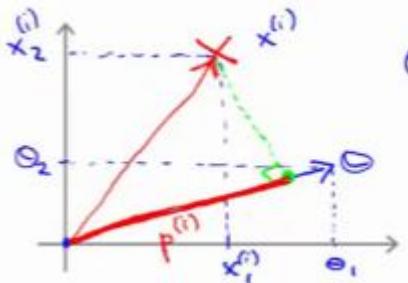
我接下来忽略掉截距，令 $\theta_0 = 0$ ，这样更容易画示意图。我将特征数 n 置为 2，因此我们仅有两个特征 x_1 和 x_2 ，现在 我们来看一下目标函数，支持向量机的优化目标函数。当我们仅有两个特征，即 $n=2$ 时，这个式子可以写作：

$$\frac{1}{2} (\Theta_1^2 + \Theta_2^2) = \frac{1}{2} (\sqrt{\Theta_1^2 + \Theta_2^2})^2 = \frac{1}{2} \|\theta\|^2, \text{ 我们只有两个参数 } \theta_1 \text{ 和 } \theta_2。你可能注意到$$

括号里面的这一项是向量 θ 的范数，或者说是向量 θ 的长度。我的意思是如果我们将向量 θ 写出来，那么我刚刚画红线的这一项就是向量 θ 的长度或范数。这里我们用的是之前学过的向量范数的定义事实上这就等于向量 θ 的长度。

当然你可以将其写作 $\theta_0, \theta_1, \theta_2$ ，如果 θ_0 等于 0，那就是 θ_1, θ_2 的长度。在这里我将忽略 θ_0 ，这样来写 θ 的范数，它仅仅和 θ_1, θ_2 有关。但是，数学上不管你是否包含 θ_0 ，其实并没有差别，因此在我们接下来的推导中去掉 θ_0 不会有影响这意味着我们的目标函数是等于 $\frac{1}{2} \|\theta\|^2$ 。因此支持向量机做的全部事情就是极小化参数向量 θ 范数的平方或者说长度的平方。

现在我将要看看这些项： $\theta^T x$ 更深入地理解它们的含义。给定参数向量 θ 给定一个样本 x ，这等于什么呢？在前一页幻灯片上，我们画出了在不同情形下， $u^T v$ 的示意图，我们将会使用这些概念， θ 和 $x^{(i)}$ 就类似于 u 和 v 。



让我们看一下示意图：我们考察一个单一的训练样本，我有一个正样本在这里，用一个叉来表示这个样本 $x^{(i)}$ ，意思是在水平轴上取值为 $x_1^{(i)}$ ，在竖直轴上取值为 $x_2^{(i)}$ 。这就是我画出的训练样本。尽管我没有将其真的看做向量。它事实上就是一个始于原点，终点位置在这个训练样本点的向量。现在，我们有一个参数向量我会将它也画成向量。我将 θ_1 画在横轴这里，将 θ_2 画在纵轴这里，那么内积 $\theta^\top x^{(i)}$ 将会是什么呢？

使用我们之前的方法，我们计算的方式就是我将训练样本投影到参数向量 θ ，然后我来看一看这个线段的长度，我将它画成红色。我将它称为 $p^{(i)}$ 来表示这是第 i 个训练样本在参数向量 θ 上的投影。根据我们之前幻灯片的内容，我们知道的是 $\theta^\top x^{(i)}$ 将会等于 p 乘以向量 θ 的长度或范数。这就等于 $\theta_1 \cdot x_1^{(i)} + \theta_2 \cdot x_2^{(i)}$ 。这两种方式是等价的，都可以用来计算 θ 和 $x^{(i)}$ 之间的内积。

这告诉了我们什么呢？这里表达的意思是：这个 $\theta^\top x^{(i)} \geq 1$ 或者 $\theta^\top x^{(i)} \leq -1$ 的约束是可以被 $p^{(i)} \cdot \theta \geq 1$ 这个约束所代替的。因为 $\theta^\top x^{(i)} = p^{(i)} \cdot \|\theta\|$ ，将其写入我们的优化目标。我们将会得到没有了约束， $\theta^\top x^{(i)}$ 而变成了 $p^{(i)} \cdot \|\theta\|$ 。

SVM Decision Boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} \|\theta\|^2$$

$$\text{s.t. } p^{(i)} \cdot \|\theta\| \geq 1 \quad \text{if } y^{(i)} = 1$$

$$p^{(i)} \cdot \|\theta\| \leq -1 \quad \text{if } y^{(i)} = -1$$

where $p^{(i)}$ is the projection of $x^{(i)}$ onto the vector θ .

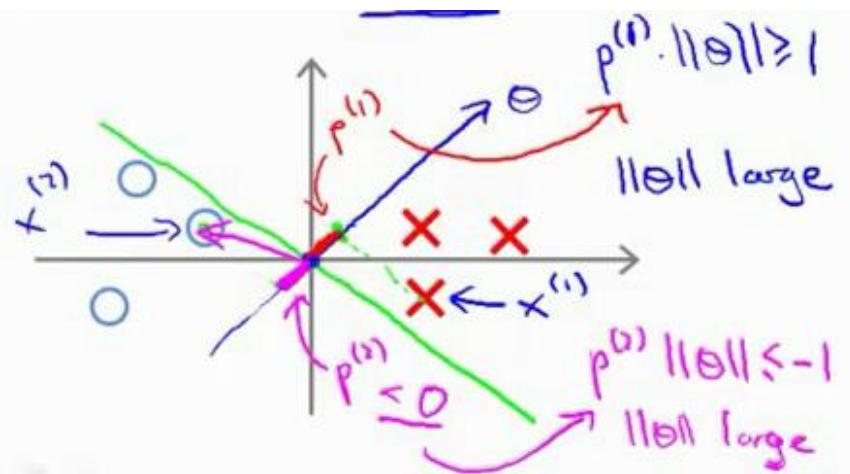
Simplification: $\underline{\theta_0 = 0}$



需要提醒一点，我们之前曾讲过这个优化目标函数可以被写成等于 $\frac{1}{2} \|\theta\|^2$ 。

现在让我们考虑下面这里的训练样本。现在，继续使用之前的简化，即 $\theta_0=0$ ，我们来看一下支持向量机会选择什么样的决策界。这是一种选择，我们假设支持向量机会选择这个决策边界。这不是一个非常好的选择，因为它的间距很小。这个决策界离训练样本的距离很近。我们来看一下为什么支持向量机不会选择它。

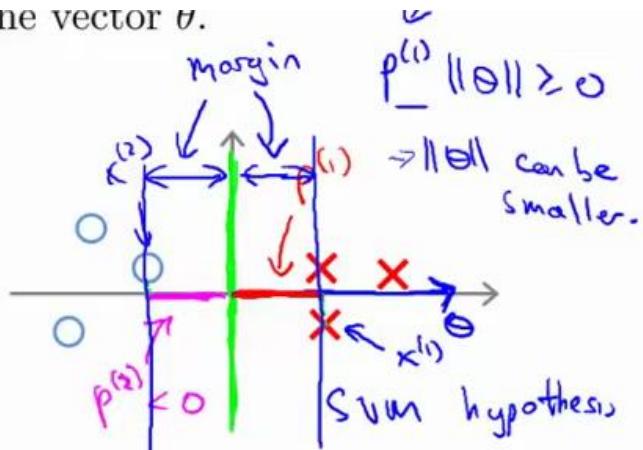
对于这样选择的参数 θ ，可以看到参数向量 θ 事实上是和决策界是 90 度正交的，因此这个绿色的决策界对应着一个参数向量 θ 指向这个方向，顺便提一句 $\theta_0=0$ 的简化仅仅意味着决策界必须通过原点 $(0,0)$ 。现在让我们看一下这对于优化目标函数意味着什么。



比如这个样本，我们假设它是我的第一个样本 $x^{(1)}$ ，如果我考察这个样本到参数 θ 的投影，投影是这个短的红线段，就等于 $p^{(1)}$ ，它非常短。类似地，这个样本如果它恰好是 $x^{(2)}$ ，我的第二个训练样本，则它到 θ 的投影在这里。我将它画成粉色，这个短的粉色线段是 $p^{(2)}$ ，即第二个样本到我的参数向量 θ 的投影。因此，这个投影非常短。 $p^{(2)}$ 事实上是一个负值， $p^{(2)}$ 是在相反的方向，这个向量和参数向量 θ 的夹角大于 90 度， $p^{(2)}$ 的值小于 0。

我们会发现这些 $p^{(i)}$ 将会是非常小的数，因此当我们考察优化目标函数的时候，对于正样本而言，我们需要 $p^{(i)} \cdot \|\theta\| >= 1$ ，但是如果 $p^{(i)}$ 在这里非常小，那就意味着我们需要 θ 的范数非常大。因为如果 $p^{(1)}$ 很小，而我们希望 $p^{(1)} \cdot \|\theta\| >= 1$ ，令其实现的唯一的办法就是这两个数较大。如果 $p^{(1)}$ 小，我们就希望 θ 的范数大。类似地，对于负样本而言我们需要 $p^{(2)} \cdot \|\theta\| <= -1$ 。我们已经在这个样本中看到 $p^{(2)}$ 会是一个非常小的数，因此唯一的办法就是 θ 的范数变大。但是我们的目标函数是希望找到一个参数 θ ，它的范数是小的。因此，这看起来不像一个好的参数向量 θ 的选择。

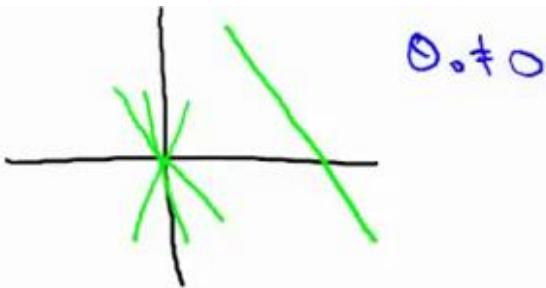
ne vector θ .



相反的，来看一个不同的决策边界。比如说，支持向量机选择了这个决策界，现在状况会有很大不同。如果这是决策界，这就是相对应的参数 θ 的方向，因此，在这个决策界之下，垂直线是决策界。使用线性代数的知识，可以说明，这个绿色的决策界有一个垂直于它的向量 θ 。现在如果你考察你的数据在横轴 x 上的投影，比如这个我之前提到的样本，我的样本 $x^{(1)}$ ，当我将它投影到横轴 x 上，或说投影到 θ 上，就会得到这样的 $p^{(1)}$ 。它的长度是 $p^{(1)}$ ，另一个样本，那个样本是 $x^{(2)}$ 。我做同样的投影，我会发现， $p^{(2)}$ 的长度是负值。你会注意到现在 $p^{(1)}$ 和 $p^{(2)}$ 这些投影长度是长多了。如果我们仍然要满足这些约束， $p^{(1)} \cdot \|\theta\| > 1$ ，则因为 $p^{(1)}$ 变大了， θ 的范数就可以变小了。因此这意味着通过选择右边的决策界，而不是左边的那个，支持向量机可以使参数 θ 的范数变小很多。因此，如果我们想令 θ 的范数变小，从而令 θ 范数的平方变小，就能让支持向量机选择右边的决策界。这就是支持向量机如何能有效地产生大间距分类的原因。

看这条绿线，这个绿色的决策界。我们希望正样本和负样本投影到 θ 的值大。要做到这一点的唯一方式就是选择这条绿线做决策界。这是大间距决策界来区分开正样本和负样本这个间距的值。这个间距的值就是 $p^{(1)} p^{(2)} p^{(3)}$ 等等的值。通过让间距变大，即通过这些 $p^{(1)} p^{(2)} p^{(3)}$ 等等的值，支持向量机最终可以找到一个较小的 θ 范数。这正是支持向量机中最小化目标函数的目的。

以上就是为什么支持向量机最终会找到大间距分类器的原因。因为它试图极大化这些 $p^{(i)}$ 的范数，它们是训练样本到决策边界的距离。最后一点，我们的推导自始至终使用了这个简化假设，就是参数 $\theta_0 = 0$ 。



就像我之前提到的。这个的作用是： $\theta_0=0$ 的意思是让决策界通过原点。如果你令 θ_0 不是 0 的话，含义就是你希望决策界不通过原点。我将不会做全部的推导。实际上，支持向量机产生大间距分类器的结论，会被证明同样成立，证明方式是非常类似的，是我们刚刚做的证明的推广。

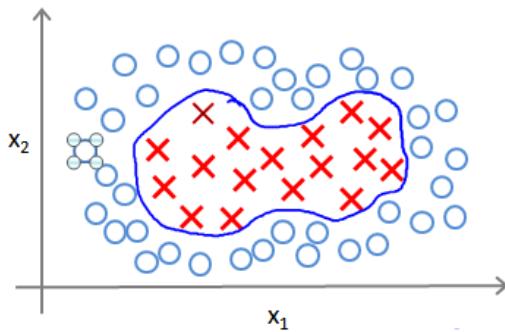
之前视频中说过，即便 θ_0 不等于 0，支持向量机要做的事情都是优化这个目标函数对应着 C 值非常大的情况，但是可以说明的是，即便 θ_0 不等于 0，支持向量机仍然会找到正样本和负样本之间的大间距分隔。

总之，我们解释了为什么支持向量机是一个大间距分类器。在下一节我们，将开始讨论如何利用支持向量机的原理，应用它们建立一个复杂的非线性分类器。

12.4 核函数 1

参考视频: [12 - 4 - Kernels I \(16 min\).mkv](#)

回顾我们之前讨论过可以使用高级数的多项式模型来解决无法用直线进行分隔的分类问题:



为了获得上图所示的判定边界，我们的模型可能是

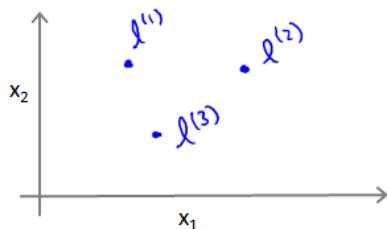
$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots$$

我们可以用一系列的新的特征 f 来替换模型中的每一项。例如令:

$$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2$$

...得到 $h\theta(x) = f_1 + f_2 + \dots + f_n$ 。然而，除了对原有的特征进行组合以外，有没有更好的方法来构造 f_1, f_2, f_3 ? 我们可以利用核函数来计算出新的特征。

给定一个训练实例 x ，我们利用 x 的各个特征与我们预先选定的地标(landmarks) $l^{(1)}, l^{(2)}, l^{(3)}$ 的近似程度来选取新的特征 f_1, f_2, f_3 。



例如：

$$f_1 = similarity(x, l^{(1)}) = e^{-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}}$$

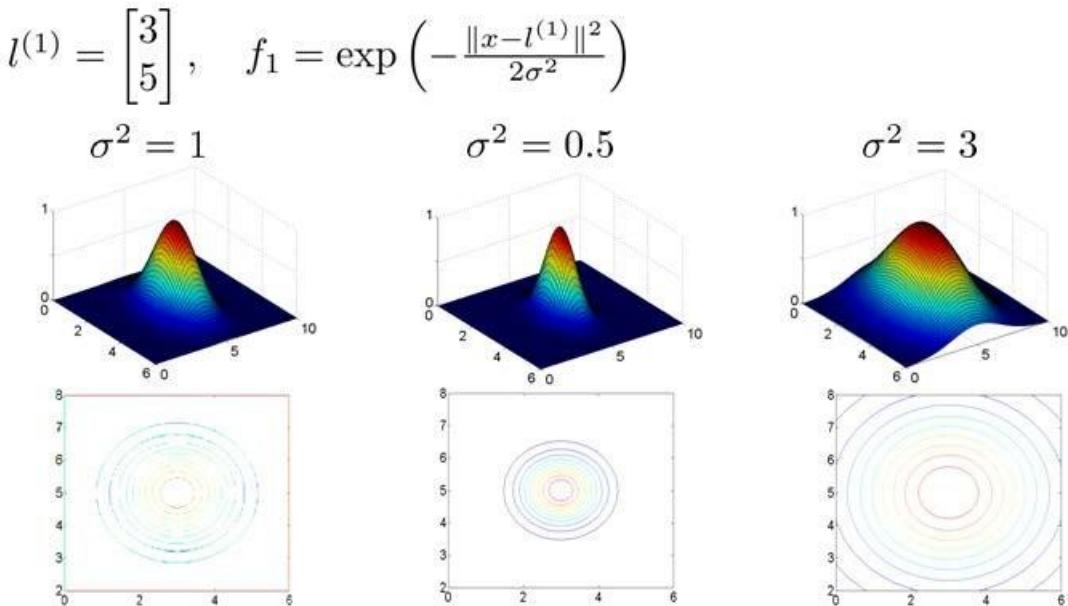
其中: $\|x - l^{(1)}\|^2 = \sum_{j=1}^n (x_j - l_j^{(1)})^2$ ，为实例 x 中所有特征与地标 $l^{(1)}$ 之间的距离的和。

上例中的 $similarity(x, l^{(1)})$ 就是核函数，具体而言，这里是一个高斯核函数 (Gaussian

Kernel)。注：这个函数与正态分布没什么实际上的关系，只是看上去像而已。

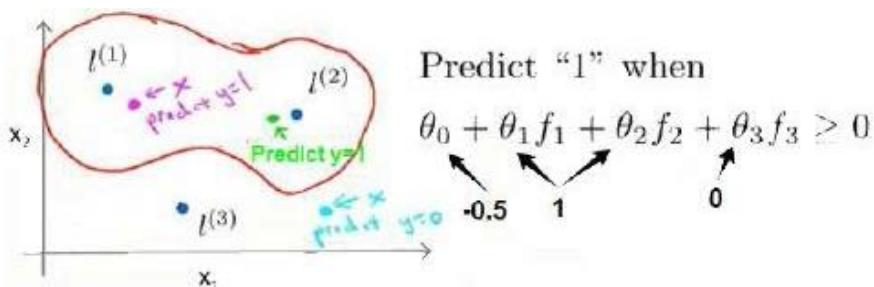
这些地标的作用是什么？如果一个训练实例 x 与地标 l 之间的距离近似于 0，则新特征 f 近似于 $e^{-0}=1$ ，如果训练实例 x 与地标 l 之间距离较远，则 f 近似于 $e^{(-\text{一个较大的数})}=0$ 。

假设我们的训练实例含有两个特征 $[x_1 \ x_2]$ ，给定地标 $l^{(1)}$ 与不同的 σ 值，见下图：



图中水平面的坐标为 x_1, x_2 而垂直坐标轴代表 f 。可以看出，只有当 x 与 $l^{(1)}$ 重合时 f 才具有最大值。随着 x 的改变 f 值改变的速率受到 σ^2 的控制。

在下图中，当实例处于洋红色的点位置处，因为其离 $l^{(1)}$ 更近，但是离 $l^{(2)}$ 和 $l^{(3)}$ 较远，因此 f_1 接近 1，而 f_2, f_3 接近 0。因此 $h_\theta(x)=\theta_0+\theta_1f_1+\theta_2f_2+\theta_3f_3>0$ ，因此预测 $y=1$ 。同理可以求出，对于离 $l^{(2)}$ 较近的绿色点，也预测 $y=1$ ，但是对于蓝绿色的点，因为其离三个地标都较远，预测 $y=0$ 。



这样，图中红色的封闭曲线所表示的范围，便是我们依据一个单一的训练实例和我们选取的地标所得出的判定边界，在预测时，我们采用的特征不是训练实例本身的特征，而是通过核函数计算出的新特征 f_1, f_2, f_3 。

12.5 核函数 2

参考视频: [12 - 5 - Kernels II \(16 min\).mkv](#)

在上一节视频里, 我们讨论了核函数这个想法, 以及怎样利用它去实现支持向量机的一些新特性。在这一节视频中, 我将补充一些缺失的细节, 并简单的介绍一下怎么在实际中使用应用这些想法。

如何选择地标?

我们通常是根据训练集的数量选择地标的数量, 即如果训练集中有 m 个实例, 则我们选取 m 个地标, 并且令: $|^{(1)}=x^{(1)}, |^{(2)}=x^{(2)}, \dots, |^{(m)}=x^{(m)}$ 。这样做的好处在于: 现在我们得到的新特征是建立在原有特征与训练集中所有其他特征之间距离的基础之上的, 即:

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} = 1 \\ f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)}) \\ f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = e^0 = 1 \\ \vdots \\ f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)}) \end{bmatrix}$$

SVM with Kernels

- Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$,
- choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$.

Given example x :

$$\begin{aligned} &\rightarrow f_1 = \text{similarity}(x, l^{(1)}) \\ &\rightarrow f_2 = \text{similarity}(x, l^{(2)}) \\ &\dots \end{aligned}$$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \quad f_0 = 1$$

For training example $(x^{(i)}, y^{(i)})$:

$$\begin{aligned} x^{(i)} \rightarrow & \begin{bmatrix} f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)}) \\ \vdots \\ f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)}) \end{bmatrix} \\ & f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp(-\frac{\|x^{(i)} - l^{(i)}\|^2}{2\sigma^2}) = 1 \end{aligned}$$

$$\begin{aligned} x^{(i)} \in \mathbb{R}^{n+1} \quad & \text{(or } \mathbb{R}^n \text{)} \\ f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \\ f_0^{(i)} = 1 \end{aligned}$$

Andrew Ng

下面我们将核函数运用到支持向量机中, 修改我们的支持向量机假设为:

- 给定 x , 计算新特征 f , 当 $\theta^T f \geq 0$ 时, 预测 $y=1$, 否则反之。相应地修改代价函数为:

$$\min C \sum_{i=1}^m [y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^{n=m} \theta_j^2$$

在具体实施过程中，我们还需要对最后的归一化项进行些微调整，在计算 $\sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta$ 时，我们用 $\theta^T M \theta$ 代替 $\theta^T \theta$ ，其中 M 是根据我们选择的核函数而不同的一个矩阵。这样做的原因是为了简化计算。

理论上讲，我们也可以在逻辑回归中使用核函数，但是上面使用 M 来简化计算的方法不适用与逻辑回归，因此计算将非常耗费时间。

在此，我们不介绍最小化支持向量机的代价函数的方法，你可以使用现有的软件包（如 liblinear, libsvm 等）。在使用这些软件包最小化我们的代价函数之前，我们通常需要编写核函数，并且如果我们使用高斯核函数，那么在使用之前进行特征缩放是非常必要的。

另外，支持向量机也可以不使用核函数，不使用核函数又称为线性核函数 (linear kernel)，当我们不采用非常复杂的函数，或者我们的训练集特征非常多而实例非常少的时候，可以采用这种不带核函数的支持向量机。

下面是支持向量机的两个参数 C 和 σ 的影响：

C 较大时，相当于 λ 较小，可能会导致过拟合，高方差

C 较小时，相当于 λ 较大，可能会导致低拟合，高偏差

σ 较大时，导致高方差

σ 较小时，导致高偏差

如果你看了本周的编程作业，你就能亲自实现这些想法，并亲眼看到这些效果。这就是利用核函数的支持向量机算法，希望这些关于偏差和方差的讨论，能给你一些对于算法结果预期的直观印象。

12.6 使用支持向量机

参考视频: [12 - 6 - Using An SVM \(21 min\).mkv](#)

目前为止，我们已经讨论了 **SVM** 比较抽象的层面，在这个视频中我将要讨论到为了运行或者运用 **SVM**。你实际上所需要的一些东西：支持向量机算法，提出了一个特别优化的问题。但是就如在之前的视频中我简单提到的，我真的不建议你自己写软件来求解参数 θ ，因此由于今天我们中的很少人，或者其实没有人考虑过自己写代码来转换矩阵，或求一个数的平方根等我们只是知道如何去调用库函数来实现这些功能。同样的，用以解决 **SVM** 最优化问题的软件很复杂，且已经有研究者做了很多年数值优化了。因此你提出好的软件库和好的软件包来做这样一些事儿。然后强烈建议使用高优化软件库中的一个，而不是尝试自己落实一些数据。有许多好的软件库，我正好用得最多的两个是 **liblinear** 和 **libsrm**，但是真的有很多软件库可以用来做这件事儿。你可以连接许多你可能会用来编写学习算法的主要编程语言。

在高斯核函数之外我们还有其他一些选择，如：

多项式核函数（Polynomial Kernel）

字符串核函数（String kernel）

卡方核函数（chi-square kernel）

直方图交集核函数（histogram intersection kernel）

等等...

这些核函数的目标也都是根据训练集和地标之间的距离来构建新特征，这些核函数需要满足 **Mercer's** 定理，才能被支持向量机的优化软件正确处理。

多类分类问题

假设我们利用之前介绍的一对多方法来解决一个多类分类问题。如果一共有 k 个类，则我们需要 k 个模型，以及 k 个参数向量 θ 。我们同样也可以训练 k 个支持向量机来解决多类分类问题。但是大多数支持向量机软件包都有内置的多类分类功能，我们只要直接使用即可。

尽管你不去写你自己的 **SVM**（支持向量机）的优化软件，但是你也需要做几件事：

- 1、是提出参数 C 的选择。我们在之前的视频中讨论过误差/方差在这方面的性质。
- 2、你也需要选择内核参数或你想要使用的相似函数，其中一个选择是：我们选择不需要任何内核参数，没有内核参数的理念，也叫线性核函数。因此，如果说有人说他使用了线性核的 **SVM**（支持向量机），这就意味这他使用了不带有核函数的 **SVM**（支持向量机）。

从逻辑回归模型，我们得到了支持向量机模型，在两者之间，我们应该如何选择呢？

下面是一些普遍使用的准则：

n 为特征数， m 为训练样本数。

如果相较于 m 而言， n 要大许多，即训练集数据量不够支持我们训练一个复杂的非线性模型，我们选用逻辑回归模型或者不带核函数的支持向量机。

如果 n 较小，而且 m 大小中等，例如 n 在 1-1000 之间，而 m 在 10-10000 之间，使用高斯核函数的支持向量机。

如果 n 较小，而 m 较大，例如 n 在 1-1000 之间，而 m 大于 50000，则使用支持向量机会非常慢，解决方案是创造、增加更多的特征，然后使用逻辑回归或不带核函数的支持向量机。

值得一提的是，神经网络在以上三种情况下都可能会有较好的表现，但是训练神经网络可能非常慢，选择支持向量机的原因主要在于它的代价函数是凸函数，不存在局部最小值。

今天的 SVM 包会工作得很好，但是它们仍然会有一些慢。当你有非常非常大的训练集，且用高斯核函数是在这种情况下，我经常会做的是尝试手动地创建，拥有更多的特征变量，然后用逻辑回归或者不带核函数的支持向量机。如果你看到这个幻灯片，看到了逻辑回归，或者不带核函数的支持向量机。在这个两个地方，我把它们放在一起是有原因的。原因是：逻辑回归和不带核函数的支持向量机它们都是非常相似的算法，不管是逻辑回归还是不带核函数的 SVM，通常都会做相似的事情，并给出相似的结果。但是根据你实现的情况，其中一个可能会比另一个更加有效。但是在其中一个算法应用的地方，逻辑回归或不带核函数的 SVM 另一个也很有可能很有效。但是随着 SVM 的复杂度增加，当你使用不同的内核函数来学习复杂的非线性函数时，这个体系，你知道的，当你有多达 1 万 (10,000) 的样本时，也可能是 5 万 (50,000)，你的特征变量的数量这是相当大的。那是一个非常常见的体系，也许在这个体系里，不带核函数的支持向量机就会表现得相当突出。你可以做比这困难得多需要逻辑回归的事情。

最后，神经网络使用于什么时候呢？对于所有的这些问题，对于所有的这些不同体系一个设计得很好的神经网络也很有可能会非常有效。有一个缺点是，或者说是有时可能不会使用神经网络的原因是：对于许多这样的问题，神经网络训练起来可能会特别慢，但是如果你有一个非常好的 SVM 实现包，它可能会运行得比较快比神经网络快很多，尽管我们在此之前没有展示，但是事实证明，SVM 具有的优化问题，是一种凸优化问题。因此，好的 SVM 优化软件包总是会找到全局最小值，或者接近它的值。对于 SVM 你不需要担心局部最优。

在实际应用中，局部最优不是神经网络所需要解决的一个重大问题，所以这是你在使用 SVM 的时候不需要太去担心的一个问题。根据你的问题，神经网络可能会比 SVM 慢，尤其是在这样一个体系中，至于这里给出的参考，看上去有些模糊，如果你在考虑一些问题，这些参考会有一些模糊，但是我仍然不能完全确定，我是该用这个算法还是改用那个算法，这个没有太大关系，当我遇到机器学习问题的时候，有时它确实不清楚这是否是最好的算法，但是就如在之前的视频中看到的算法确实很重要。但是通常更加重要的是：你有多少数据，你有多熟练是否擅长做误差分析和排除学习算法，指出如何设定新的特征变量和找出其他能决定你学习算法的变量等方面，通常这些方面会比你使用逻辑回归还是 SVM 这方面更加重要。但是，已经说过了，SVM 仍然被广泛认为是一种最强大的学习算法，这是一个体系，包含了什么时候一个有效的方法去学习复杂的非线性函数。因此，实际上与逻辑回归神经网络 SVM 一起使用这些方法来提高学习算法，我认为你会很好地建立很有技术的状态。

机器学习系统对于一个宽泛的应用领域来说，这是另一个在你军械库里非常强大的工具，你可以把它应用到很多地方，如硅谷、在工业、学术等领域建立许多高性能的机器学习系统。

第 8 周

十三、聚类(Clustering)

13.1 无监督学习：简介

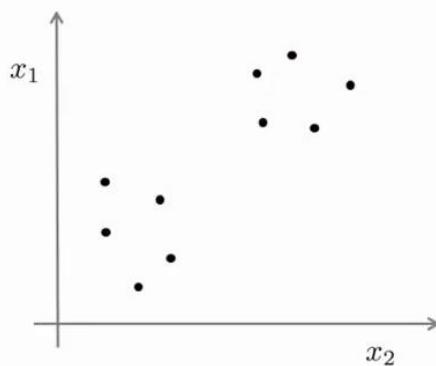
参考视频: [13 - 1 - Unsupervised Learning Introduction \(3 min\).mkv](#)

在这个视频中，我将开始介绍聚类算法。这将是一个激动人心的时刻，因为这是我们学习的第一个非监督学习算法。我们将要让计算机学习无标签数据，而不是此前的标签数据。

那么，什么是非监督学习呢？在课程的一开始，我曾简单的介绍过非监督学习，然而，我们还是有必要将其与监督学习做一下比较。

在一个典型的监督学习中，我们有一个有标签的训练集，我们的目标是找到能够区分正样本和负样本的决策边界，在这里的监督学习中，我们有一系列标签，我们需要据此拟合一个假设函数。与此不同的是，在非监督学习中，我们的数据没有附带任何标签，我们拿到的数据就是这样的：

Unsupervised learning

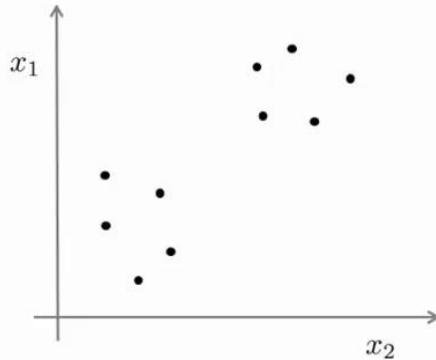


Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$ ←

在这里我们有一系列点，却没有标签。因此，我们的训练集可以写成只有 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ 。我们没有任何标签 y 。因此，图上画的这些点没有标签信息。也就是说，在非监督学习中，我们需要将一系列无标签的训练数据，输入到一个算法中，然后我们告诉这个算法，快去为我们找找这个数据的内在结构给定数据。我们可能需要某种算法帮助我们寻找一种结构。图上的数据看起来可以分成两个分开的点集（称为簇），一个能够找到我圈出的这些点

集的算法，就被称为聚类算法。

Unsupervised learning

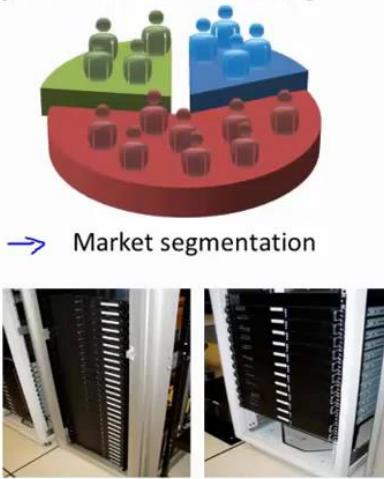


Training set: $\{x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(m)}\}$ ←

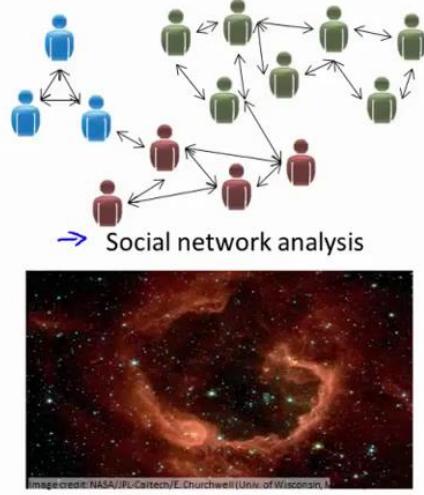
这将是我们介绍的第一个非监督学习算法。当然，此后我们还将提到其他类型的非监督学习算法，它们可以为我们找到其他类型的结构或者其他的一些模式，而不只是簇。

我们将先介绍聚类算法。此后，我们将陆续介绍其他算法。那么聚类算法一般用来做什么呢？

Applications of clustering



→ Market segmentation



→ Social network analysis



Organize computing clusters

Astronomical data analysis

在这门课程的早些时候，我曾经列举过一些应用：比如市场分割。也许你在数据库中存储了许多客户的信息，而你希望将他们分成不同的客户群，这样你可以对不同类型的客户分别销售产品或者分别提供更适合的服务。社交网络分析：事实上有许多研究人员正在研究这样一些内容，他们关注一群人，关注社交网络，例如 Facebook, Google+, 或者是其他的一些信息，比如说：你经常跟哪些人联系，而这些人又经常给哪些人发邮件，由此找到关系密切的人群。因此，这可能需要另一个聚类算法，你希望用它发现社交网络中关系密切的朋友。

我有一个朋友正在研究这个问题，他希望使用聚类算法来更好的组织计算机集群，或者更好的管理数据中心。因为如果你知道数据中心中，那些计算机经常协作工作。那么，你可以重新分配资源，重新布局网络。由此优化数据中心，优化数据通信。

最后，我实际上还在研究如何利用聚类算法了解星系的形成。然后用这个知识，了解一些天文学上的细节问题。好的，这就是聚类算法。这将是我们介绍的第一个非监督学习算法。在下一个视频中，我们将开始介绍一个具体的聚类算法。

13.2 K-均值算法

参考视频: [13 - 2 - K-Means Algorithm \(13 min\).mkv](#)

K-均值是最普及的聚类算法，算法接受一个未标记的数据集，然后将数据聚类成不同的组。

K-均值是一个迭代算法，假设我们想要将数据聚类成 n 个组，其方法为：

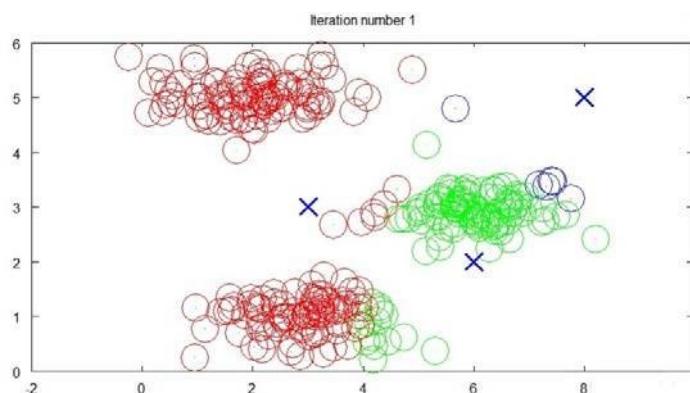
首先选择 K 个随机的点，称为聚类中心（cluster centroids）；

对于数据集中的每一个数据，按照距离 K 个中心点的距离，将其与距离最近的中心点关联起来，与同一个中心点关联的所有点聚成一类。

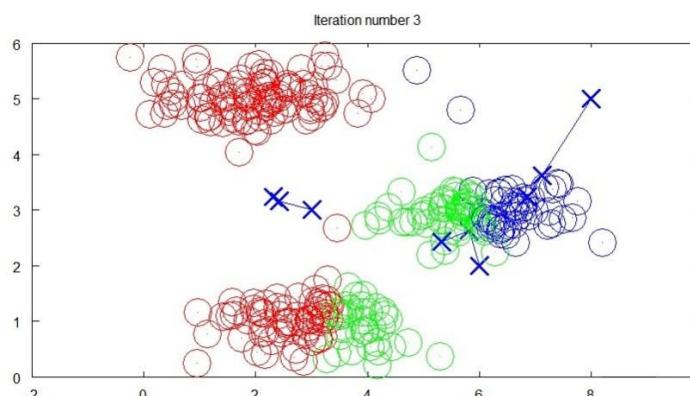
计算每一个组的平均值，将该组所关联的中心点移动到平均值的位置。

重复步骤 2-4 直至中心点不再变化。

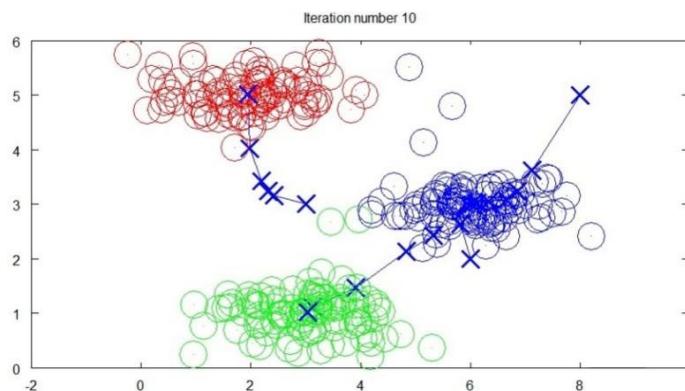
下面是一个聚类示例：



迭代 1 次



迭代 3 次



迭代 10 次

用 $\mu^1, \mu^2, \dots, \mu^K$ 来表示聚类中心，用 $c^{(1)}, c^{(2)}, \dots, c^{(m)}$ 来存储与第 i 个实例数据最近的聚类中心的索引，K-均值算法的伪代码如下：

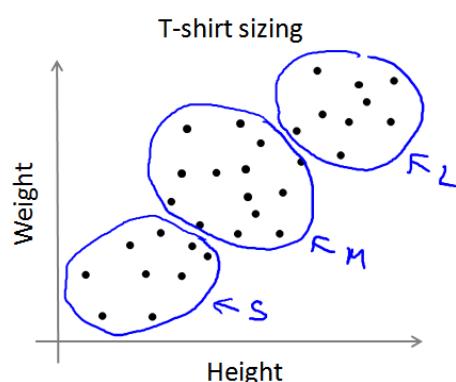
```

Repeat {
    for i = 1 to m
         $c^{(i)} := \text{index (from 1 to } K \text{) of cluster centroid closest to } x^{(i)}$ 
    for k = 1 to K
         $\mu_k := \text{average (mean) of points assigned to cluster } k$ 
}

```

算法分为两个步骤，第一个 **for** 循环是赋值步骤，即：对于每一个样例 i ，计算其应该属于的类。第二个 **for** 循环是聚类中心的移动，即：对于每一个类 k ，重新计算该类的质心。

K-均值算法也可以很便利地用于将数据分为许多不同组，即使在没有非常明显区分的组群的情况下也可以。下图所示的数据集包含身高和体重两项特征构成的，利用 K-均值算法将数据分为三类，用于帮助确定将要生产的 T-恤衫的三种尺寸。



13.3 优化目标

参考视频: [13 - 3 - Optimization Objective \(7 min\).mkv](#)

K-均值最小化问题，是要最小化所有的数据点与其所关联的聚类中心点之间的距离之和，因此 K-均值的代价函数（又称畸变函数 Distortion function）为：

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

其中 $\mu_{c(i)}$ 代表与 $x^{(i)}$ 最近的聚类中心点。我们的优化目标便是找出使得代价函数最小的 $c^{(1)}, c^{(2)}, \dots, c^{(m)}$ 和 $\mu^1, \mu^2, \dots, \mu^K$ ：

$$\min_{\substack{c^{(1)}, \dots, c^{(m)}, \\ \mu_1, \dots, \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

回顾刚才给出的 K-均值迭代算法，我们知道，第一个循环是用于减小 $c^{(i)}$ 引起的代价，而第二个循环则是用于减小 μ_i 引起的代价。迭代的过程一定会是每一次迭代都在减小代价函数，不然便是出现了错误。

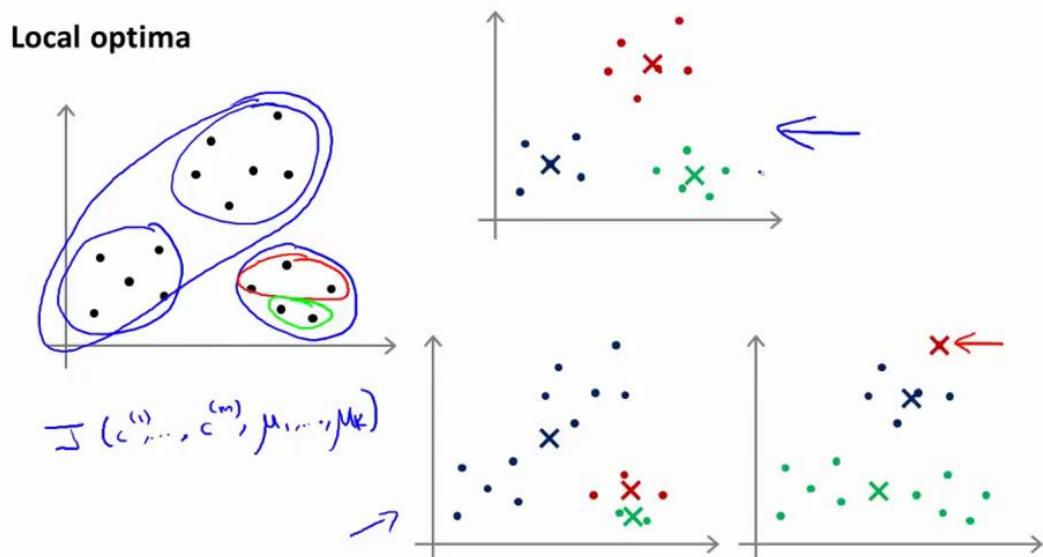
13.4 随机初始化

参考视频: [13 - 4 - Random Initialization \(8 min\).mkv](#)

在运行 K -均值算法的之前，我们首先要随机初始化所有的聚类中心点，下面介绍怎样做：

1. 我们应该选择 $K < m$ ，即聚类中心点的个数要小于所有训练集实例的数量
2. 随机选择 K 个训练实例，然后令 K 个聚类中心分别与这 K 个训练实例相等

K -均值的一个问题在于，它有可能会停留在一个局部最小值处，而这取决于初始化的情况。



为了解决这个问题，我们通常需要多次运行 K -均值算法，每一次都重新进行随机初始化，最后再比较多次运行 K -均值的结果，选择代价函数最小的结果。这种方法在 K 较小的时候（2-10）还是可行的，但是如果 K 较大，这么做也可能不会有明显地改善。

13.5 选择聚类数

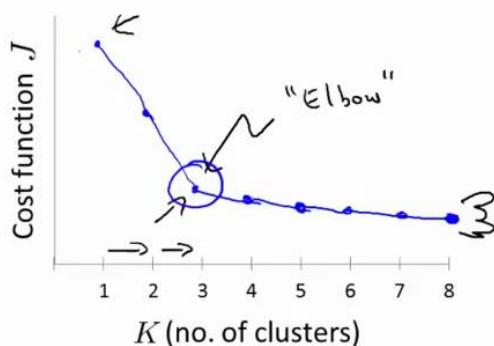
参考视频: [13 - 5 - Choosing the Number of Clusters \(8 min\).mkv](#)

没有所谓最好的选择聚类数的方法,通常是需要根据不同的问题,人工进行选择的。选择的时候思考我们运用 K-均值算法聚类的动机是什么,然后选择能最好服务于该目的标聚类数。

当人们在讨论,选择聚类数目的方法时,有一个可能会谈及的方法叫作“肘部法则”。关于“肘部法则”,我们所需要做的是改变 K 值,也就是聚类类别数目的总数。我们用一个聚类来运行 K 均值聚类方法。这就意味着,所有的数据都会分到一个聚类里,然后计算成本函数或者计算畸变函数 J 。 K 代表聚类数字。

Choosing the value of K

Elbow method:



我们可能会得到一条类似于这样的曲线。像一个人的肘部。这就是“肘部法则”所做的,让我们来看这样一个图,看起来就好像有一个很清楚的肘在那儿。好像人的手臂,如果你伸出你的胳膊,那么这就是你的肩关节、肘关节、手。这就是“肘部法则”。你会发现这种模式,它的畸变值会迅速下降,从 1 到 2, 从 2 到 3 之后,你会在 3 的时候达到一个肘点。在此之后,畸变值就下降的非常慢,看起来就像使用 3 个聚类来进行聚类是正确的,这是因为那个点是曲线的肘点,畸变值下降得很快, K 等于 3 之后就下降得很慢,那么我们就选 K 等于 3。当你应用“肘部法则”的时候,如果你得到了一个像上面这样的图,那么这将是一种用来选择聚类个数的合理方法。

例如,我们的 T-恤制造例子中,我们要将用户按照身材聚类,我们可以分成 3 个尺寸 S,M,L 也可以分成 5 个尺寸 XS,S,M,L,XL, 这样的选择是建立在回答“聚类后我们制造的 T-恤是否能较好地适合我们的客户”这个问题的基础上作出的。

十四、降维(Dimensionality Reduction)

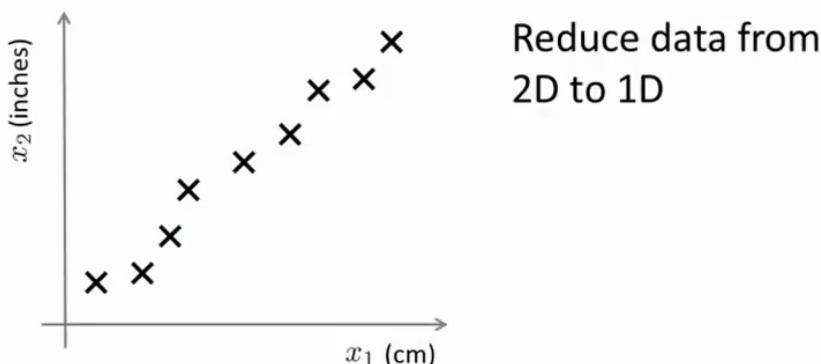
14.1 动机一：数据压缩

参考视频: [14 - 1 - Motivation I Data Compression \(10 min\).mkv](#)

这个视频，我想开始谈论第二种类型的无监督学习问题，称为降维。有几个不同的的原因使你可能想要做降维。一是数据压缩，后面我们会看了一些视频后，数据压缩不仅允许我们压缩数据，因而使用较少的计算机内存或磁盘空间，但它也让我们加快我们的学习算法。

但首先，让我们谈论降维是什么。作为一种生动的例子，我们收集的数据集，有许多，许多特征，我绘制两个在这里。

Data Compression



假设我们未知两个的特征 x_1 :长度: 用厘米表示; x_2 , 是用英寸表示同一物体的长度。

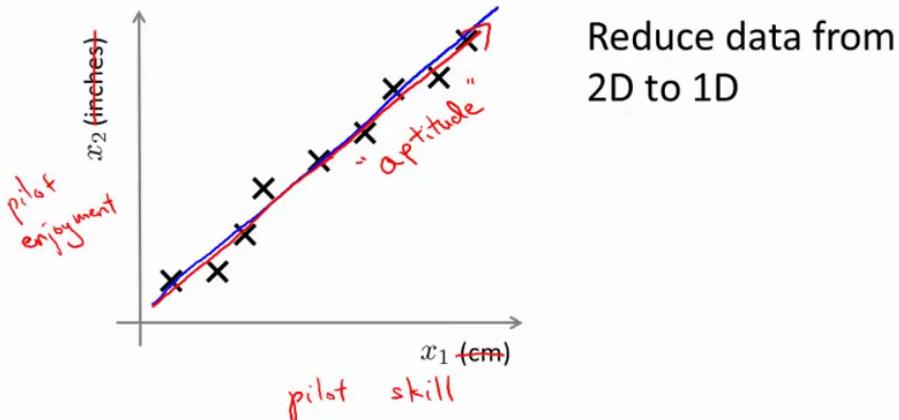
所以，这给了我们高度冗余表示，也许不是两个分开的特征 x_1 和 x_2 ，这两个基本的长度度量，也许我们想要做的是减少数据到一维，只有一个数测量这个长度。这个例子似乎有点做作，这里厘米英寸的例子实际上不是那么不切实际的，两者并没有什么不同。

将数据从二维降至一维: 假使我们要采用两种不同的仪器来测量一些东西的尺寸，其中一个仪器测量结果的单位是英寸，另一个仪器测量的结果是厘米，我们希望将测量的结果作为我们机器学习的特征。现在的问题的是，两种仪器对同一个东西测量的结果不完全相等（由于误差、精度等），而将两者都作为 特征有些重复，因而，我们希望将这个二维的数据降至一维。

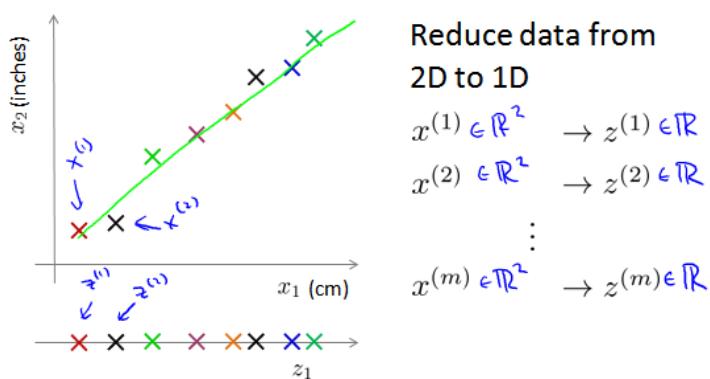
从这件事情我看到的东西发生在工业上的事。如果你有几百个或成千上万的特征，它是它这往往容易失去你需要的特征。有时可能有几个不同的工程团队，也许一个工程队给你二百个特征，第二工程队给你另外三百个的特征，第三工程队给你五百个特征，一千多个特征

都在一起，它实际上会变得非常困难，去跟踪你知道的那些特征，你从那些工程队得到的。其实不想有高度冗余的特征一样。

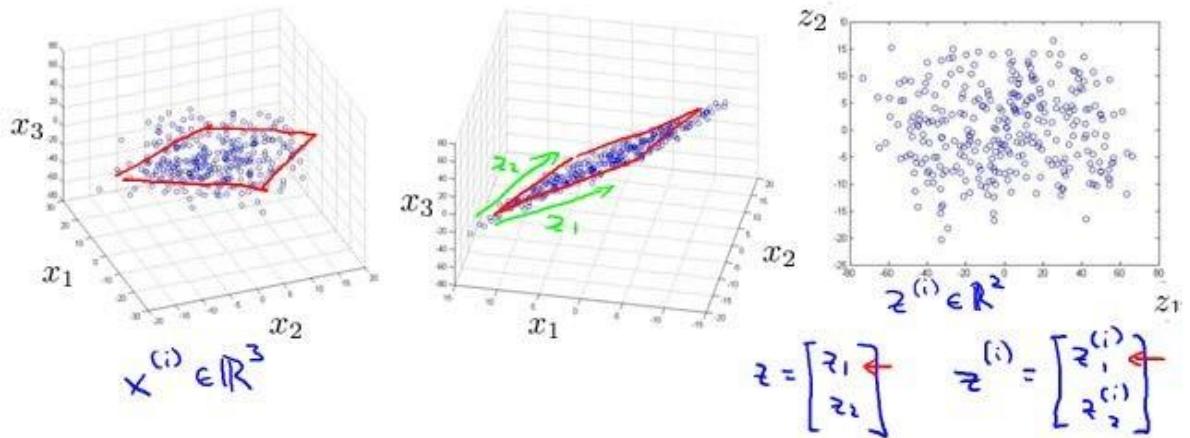
Data Compression



多年我一直在研究直升飞机自动驾驶。诸如此类。如果你想测量——如果你想做，你知道，做一个调查或做这些不同飞行员的测试——你可能有一个特征： x_1 ，这也许是他们的技能（直升飞机飞行员），也许“ x_2 ”可能是飞行员的爱好。这是表示他们是否喜欢飞行，也许这两个特征将高度相关。你真正关心的可能是这条红线的方向，不同的特征，决定飞行员的能力。



将数据从三维降至二维：这个例子中我们要将一个三维的特征向量降至一个二维的特征向量。过程是与上面类似的，我们将三维向量投射到一个二维的平面上，强迫使得所有的数据都在同一个平面上，降至二维的特征向量。



这样的处理过程可以被用于把任何维度的数据降到任何想要的维度，例如将 1000 维的特征降至 100 维。

正如我们所看到的，最后，这将使我们能够使我们的一些学习算法运行也较晚，但我们在以后的视频提到它。

14.2 动机二：数据可视化

参考视频: [14 - 2 - Motivation II Visualization \(6 min\).mkv](#)

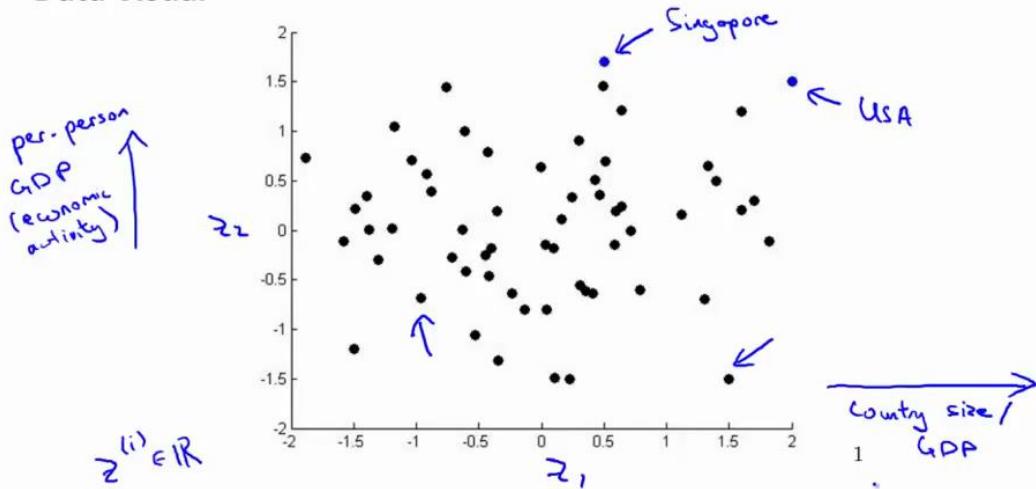
在许多及其学习问题中，如果我们能将数据可视化，我们便能寻找到一个更好的解决方案，降维可以帮助我们。

Data Visualization

Country	GDP (trillions of US\$)	Per capita GDP (thousands of intl. \$)	Human Development Index	Life expectancy	Poverty Index (Gini as percentage)	Mean household income (thousands of US\$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...

假使我们有有关于许多不同国家的数据，每一个特征向量都有 50 个特征（如，GDP，人均 GDP，平均寿命等）。如果要将这个 50 维的数据可视化是不可能的。使用降维的方法将其降至 2 维，我们便可以将其可视化了。

Data Visualization



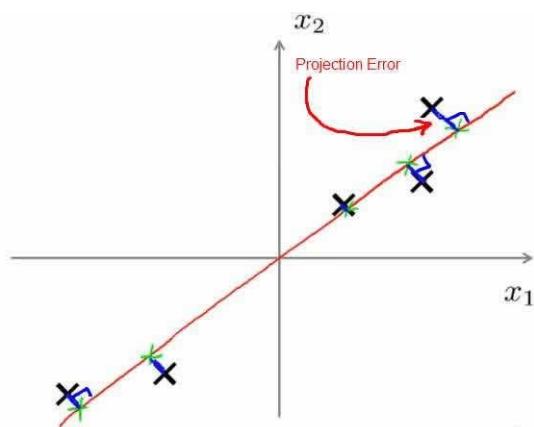
这样做的问题在于，降维的算法只负责减少维数，新产生的特征的意义就必须由我们自己去发现了。

14.3 主成分分析问题

参考视频: [14 - 3 - Principal Component Analysis Problem Formulation \(9 min\).mkv](#)

主成分分析 (PCA) 是最常见的降维算法。

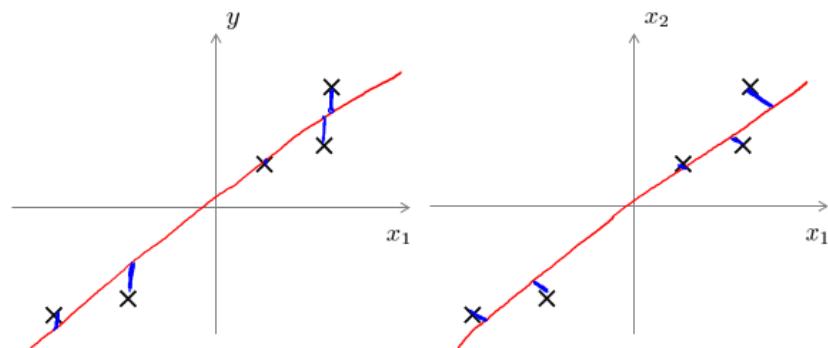
在 PCA 中, 我们要做的是找到一个方向向量 (Vector direction), 当我们把所有的数据都投射到该向量上时, 我们希望投射平均均方误差能尽可能地小。方向向量是一个经过原点的向量, 而投射误差是从特征向量向该方向向量作垂线的长度。



下面给出主成分分析问题的描述:

问题是要将 n 维数据降至 k 维, 目标是找到向量 $u(1), u(2), \dots, u(k)$ 使得总的投射误差最小。主成分分析与线性回顾的比较:

主成分分析与线性回归是两种不同的算法。主成分分析最小化的是投射误差 (Projected Error), 而线性回归尝试的是最小化预测误差。线性回归的目的是预测结果, 而主成分分析不作任何预测。



上图中, 左边的是线性回归的误差 (垂直于横轴投影), 右边则是主要成分分析的误差 (垂直于红线投影)。

PCA 将 n 个特征降维到 k 个，可以用来进行数据压缩，如果 100 维的向量最后可以用 10 维来表示，那么压缩率为 90%。同样图像处理领域的 KL 变换使用 PCA 做图像压缩。但 PCA 要保证降维后，还要保证数据的特性损失最小。

PCA 技术的一大好处是对数据进行降维的处理。我们可以对新求出的“主元”向量的重要性进行排序，根据需要取前面最重要的部分，将后面的维数省去，可以达到降维从而简化模型或是对数据进行压缩的效果。同时最大程度的保持了原有数据的信息。

PCA 技术的一个很大的优点是，它是完全无参数限制的。在 PCA 的计算过程中完全不需要人为的设定参数或是根据任何经验模型对计算进行干预，最后的结果只与数据相关，与用户是独立的。

但是，这一点同时也可看作是缺点。如果用户对观测对象有一定的先验知识，掌握了数据的一些特征，却无法通过参数化等方法对处理过程进行干预，可能会得不到预期的效果，效率也不高。

14.4 主成分分析算法

参考视频: [14 - 4 - Principal Component Analysis Algorithm \(15 min\).mkv](#)

PCA 减少 n 维到 k 维:

第一步是均值归一化。我们需要计算出所有特征的均值, 然后令 $x_j = x_j - \mu_j$ 。如果特征是在不同的数量级上, 我们还需要将其除以标准差 σ^2 。

第二步是计算协方差矩阵 (covariance matrix) Σ :

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)})(x^{(i)})^T$$

第三步是计算协方差矩阵 Σ 的特征向量 (eigenvectors) :

在 Octave 里我们可以利用奇异值分解 (singular value decomposition) 来求解, $[U, S, V] = svd(\Sigma)$ 。

$$U = \begin{bmatrix} | & | & | & | \\ u^{(1)} & u^{(2)} & \dots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)})(x^{(i)})^T$$

对于一个 $n \times n$ 维度的矩阵, 上式中的 U 是一个具有与数据之间最小投射误差的方向向量构成的矩阵。如果我们希望将数据从 n 维降至 k 维, 我们只需要从 U 中选取前 k 个向量, 获得一个 $n \times k$ 维度的矩阵, 我们用 U_{reduce} 表示, 然后通过如下计算获得要求的新特征向量 $z^{(i)}$:

$$z^{(i)} = U_{reduce}^T \times x^{(i)}$$

其中 x 是 $n \times 1$ 维的, 因此结果为 $k \times 1$ 维度。注, 我们不对方差特征进行处理。

14.5 选择主成分的数量

参考视频: [14 - 5 - Choosing The Number Of Principal Components \(13 min\).mkv](#)

主要成分分析是减少投射的平均均方误差:

训练集的方差为: $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

我们希望在平均均方误差与训练集方差的比例尽可能小的情况下选择尽可能小的 K 值。

如果我们希望这个比例小于 1%，就意味着原本数据的偏差有 99% 都保留下来了，如果我们选择保留 95% 的偏差，便能非常显著地降低模型中特征的维度了。

我们可以先令 $K=1$ ，然后进行主要成分分析，获得 U_{reduce} 和 z ，然后计算比例是否小于 1%。如果不是的话再令 $K=2$ ，如此类推，直到找到可以使得比例小于 1% 的最小 K 值（原因是各个特征之间通常情况存在某种相关性）。

还有一些更好的方式来选择 K ，当我们在 Octave 中调用“svd”函数的时候，我们获得三个参数: $[U, S, V] = svd(\sigma)$ 。

$$S = \begin{bmatrix} s_{11} & & & \\ & s_{22} & & \\ & & s_{33} & \\ & & & \ddots \\ & & & s_{nn} \end{bmatrix}$$

其中的 S 是一个 $n \times n$ 的矩阵，只有对角线上有值，而其它单元都是 0，我们可以使用这个矩阵来计算平均均方误差与训练集方差的比例:

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} = I - \frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^m s_{ii}} \leq 1\%$$

也就是:

$$\frac{\sum_{i=1}^k s_{ii}}{\sum_{i=1}^n s_{ii}} \geq 0.99$$

在压缩过数据后，我们可以采用如下方法来近似地获得原有的特征:

$$x_{approx}^{(i)} = U_{reduce} z^{(i)}$$

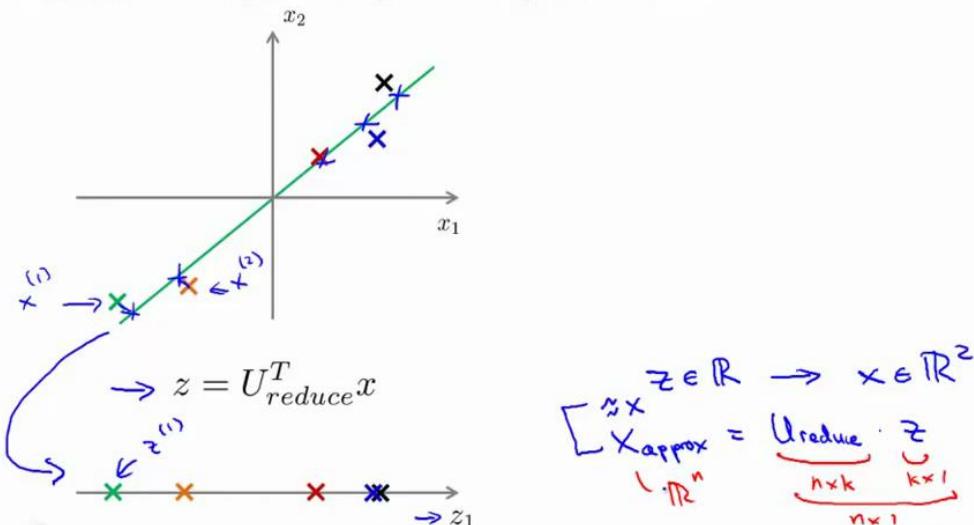
14.6 重建的压缩表示

参考视频: [14 - 6 - Reconstruction from Compressed Representation \(4 min\).mkv](#)

在以前的视频中, 我谈论 PCA 作为压缩算法。在那里你可能需要把 1000 维的数据压缩一百维特征, 或具有三维数据压缩到一二维表示。所以, 如果这是一个压缩算法, 应该能回到这个压缩表示, 回到你原有的高维数据的一种近似。

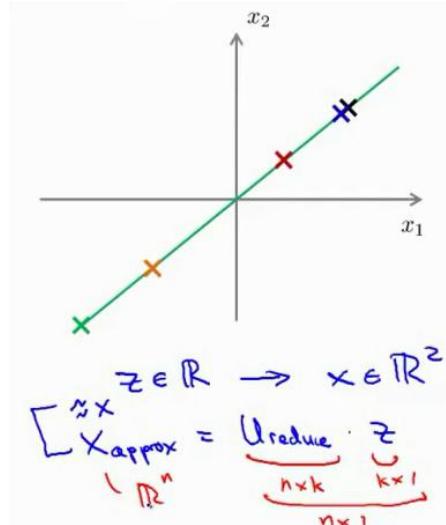
所以, 给定的 $z^{(1)}$, 这可能一百维, 怎么回到你原来的表示 $x^{(1)}$, 这可能是一千维的数据组?

Reconstruction from compressed representation



PCA 算法, 我们可能有一个这样的样本。如图中样本 $x^{(1)}$, $x^{(2)}$ 。我们做的是, 我们把这些样本投射到图中这个一维平面。然后现在我们需要只使用一个实数, 比如 $z^{(1)}$, 指定这些点的位置后他们被投射到这一个三维曲面。给定一个点 $z^{(1)}$, 我们怎么能回去这个原始的二维空间呢? x 为 2 维, z 为 1 维, $z = U_{reduce}^T x$, 相反的方程为: $x_{approx} = U_{reduce} \cdot z$ 。

$x_{approx} \approx x$ 。如图:



如你所知，这是一个漂亮的与原始数据相当相似。所以，这就是你从低维表示 z 回到未压缩的表示。我们得到的数据的一个之间你的原始数据 x ，我们把这个过程称为重建原始数据。

当我们认为试图重建从压缩表示 x 的初始值。所以，给定未标记的数据集，您现在知道如何应用 PCA，你的带高维特征 x 和映射到这的低维表示 z 。这个视频，希望你现在也知道如何采取这些低维表示 z ，映射到备份到一个近似你原有的高维数据。

现在你知道如何实施应用 PCA，我们将要做的事是谈论一些技术在实际使用 PCA 很好，特别是，在接下来的视频中，我想谈一谈关于如何选择 K 。

14.7 主成分分析法的应用建议

参考视频: [14 - 7 - Advice for Applying PCA \(13 min\).mkv](#)

假使我们正在针对一张 100×100 像素的图片进行某个计算机视觉的机器学习，即总共有 10000 个特征。

1. 第一步是运用主要成分分析将数据压缩至 1000 个特征
2. 然后对训练集运行学习算法
3. 在预测时，采用之前学习而来的 U_{reduce} 将输入的特征 x 转换成特征向量 z ，然后再进行预测

注：如果我们有交叉验证集合测试集，也采用对训练集学习而来的 U_{reduce} 。

错误的主要成分分析情况：一个常见错误使用主要成分分析的情况是，将其用于减少过拟合（减少了特征的数量）。这样做非常不好，不如尝试归一化处理。原因在于主要成分分析只是近似地丢弃掉一些特征，它并不考虑任何与结果变量有关的信息，因此可能会丢失非常重要的特征。然而当我们进行归一化处理时，会考虑到结果变量，不会丢掉重要的数据。

另一个常见的错误是，默认地将主要成分分析作为学习过程中的一部分，这虽然很多时候有效果，最好还是从所有原始特征开始，只在有必要的时候（算法运行太慢或者占用太多内存）才考虑采用主要成分分析。

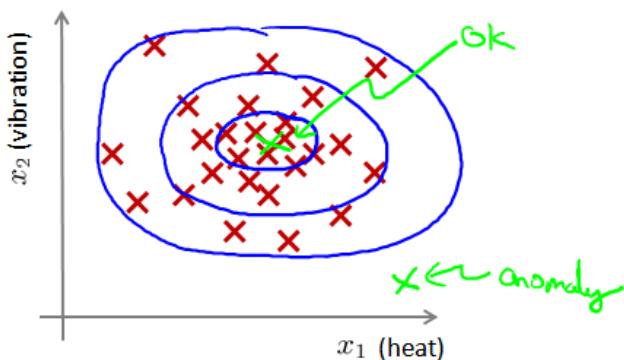
第9周

十五、异常检测(Anomaly Detection)

15.1 问题的动机

参考文档: [15 - 1 - Problem Motivation \(8 min\).mkv](#)

给定数据集 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$, 我们假使数据集是正常的, 我们希望知道新的数据 x_{test} 是不是异常的, 即这个测试数据不属于该组数据的几率如何。我们所构建的模型应该能根据该测试数据的位置告诉我们其属于一组数据的可能性 $p(x)$ 。



上图中, 在蓝色圈内的数据属于该组数据的可能性较高, 而越是偏远的数据, 其属于该组数据的可能性就越低。

这种方法称为密度估计, 表达如下:

$$\text{if } p(x) \begin{cases} \leq \varepsilon & \text{anomaly} \\ > \varepsilon & \text{normal} \end{cases}$$

欺诈检测:

$x^{(i)}$ = 用户的第 i 个活动特征

模型 $p(x)$ = 我们其属于一组数据的可能性

通过 $p(x) < \varepsilon$ 检测非正常用户

异常检测主要用来识别欺骗。例如在线采集而来的有关用户的 data, 一个特征向量中

可能会包含如：用户多久登录一次，访问过的页面，在论坛发布的帖子数量，甚至是打字速度等。尝试根据这些特征构建一个模型，可以用这个模型来识别那些不符合该模式的用户。

再一个例子是检测一个数据中心，特征可能包含：内存使用情况，被访问的磁盘数量，CPU 的负载，网络的通信量等。根据这些特征可以构建一个模型，用来判断某些计算机是否有可能出错了。

15.2 高斯分布

参考视频: [15 - 2 - Gaussian Distribution \(10 min\).mkv](#)

回顾高斯分布的基本知识。

通常如果我们认为变量 x 符合高斯分布 $x \sim N(\mu, \sigma^2)$ 则其概率密度函数为:

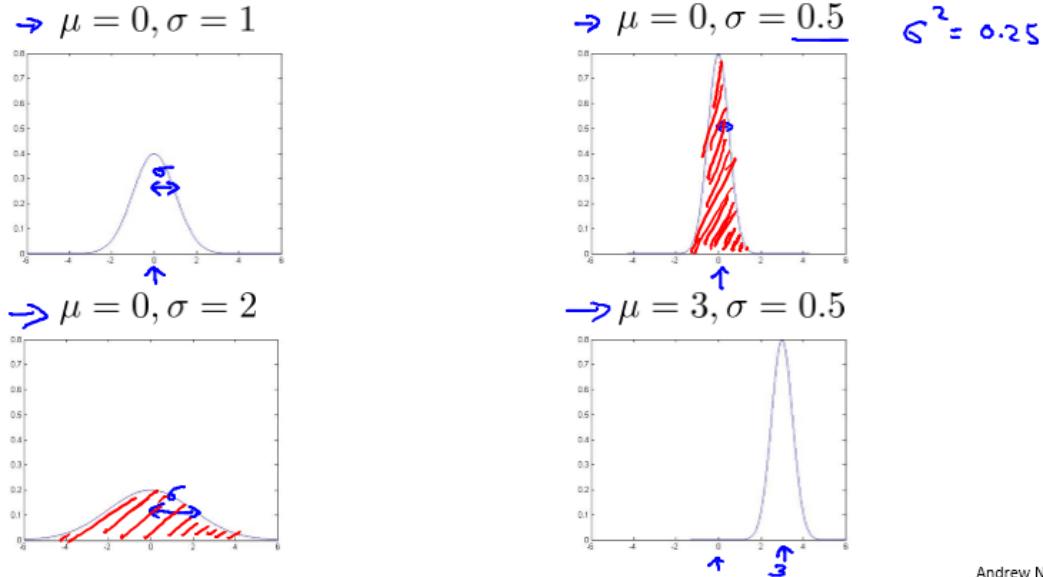
$$P(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

我们可以利用已有的数据来预测总体中的 μ 和 σ^2 的计算方法如下:

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

高斯分布样例:



注: 机器学习中对于方差我们通常只除以 m 而非统计学中的 $(m-1)$ 。

15.3 算法

参考视频: [15 - 3 - Algorithm \(12 min\).mkv](#)

异常检测算法:

对于给定的数据集 $x^{(1)}, x^{(2)}, \dots, x^{(m)}$ ，我们要针对每一个特征计算 μ 和 σ^2 的估计值。

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

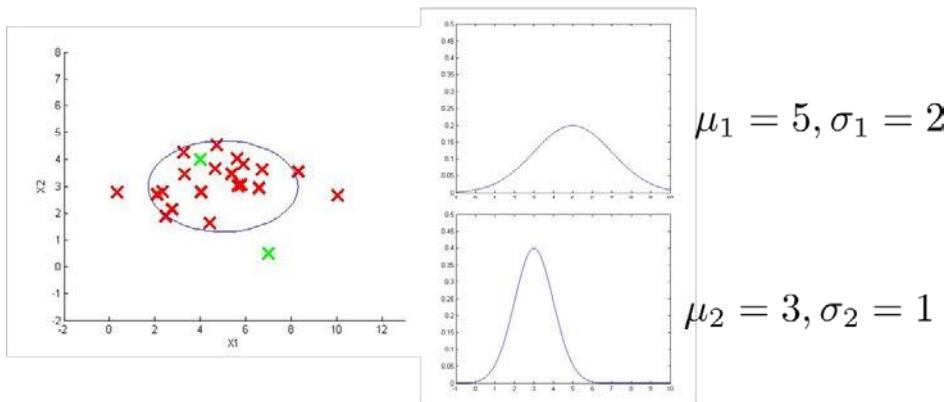
$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

一旦我们获得了平均值和方差的估计值，给定新的一个训练实例，根据模型计算 $p(x)$:

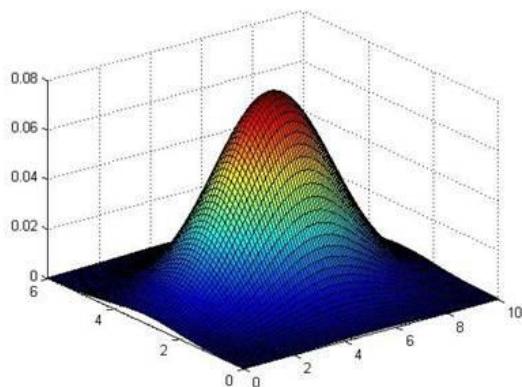
$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

当 $p(x) < \epsilon$ 时，为异常

下图是一个由两个特征的训练集，以及特征的分布情况:



下面的三维图表表示的是密度估计函数，z 轴为根据两个特征的值所估计 $p(x)$ 值:



我们选择一个 ϵ , 将 $p(x)=\epsilon$ 作为我们的判定边界, 当 $p(x)>\epsilon$ 时预测数据为正常数据, 否则则为异常。

15.4 开发和评价一个异常检测系统

参考视频: [15 - 4 - Developing and Evaluating an Anomaly Detection System \(13 min\).mkv](#)

异常检测算法是一个非监督学习算法, 意味着我们无法根据结果变量 y 的值来告诉我们数据是否真的是异常的。我们需要另一种方法来帮助检验算法是否有效。当我们开发一个异常检测系 统时, 我们从带标记 (异常或正常) 的数据着手, 我们从其中选择一部分正常数据用于构建训练集, 然后用剩下的正常数据和异常数据混合的数据构成交叉检验集和测试集。

例如: 我们有 10000 台正常引擎的数据, 有 20 台异常引擎的数据。 我们这样分配数据:

6000 台正常引擎的数据作为训练集

2000 台正常引擎和 10 台异常引擎的数据作为交叉检验集

2000 台正常引擎和 10 台异常引擎的数据作为测试集

具体的评价方法如下:

1. 根据测试集数据, 我们估计特征的平均值和方差并构建 $p(x)$ 函数
2. 对交叉检验集, 我们尝试使用不同的 ϵ 值作为阀值, 并预测数据是否异常, 根据 F_1 值或者查准率与查全率的比例来选择 ϵ
3. 选出 ϵ 后, 针对测试集进行预测, 计算异常检验系统的 F_1 值, 或者查准率与查全率之比

15.5 异常检测与监督学习对比

参考视频: [15 - 5 - Anomaly Detection vs. Supervised Learning \(8 min\).mkv](#)

之前我们构建的异常检测系统也使用了带标记的数据，与监督学习有些相似，下面的对比有助于选择采用监督学习还是异常检测：

两者比较：

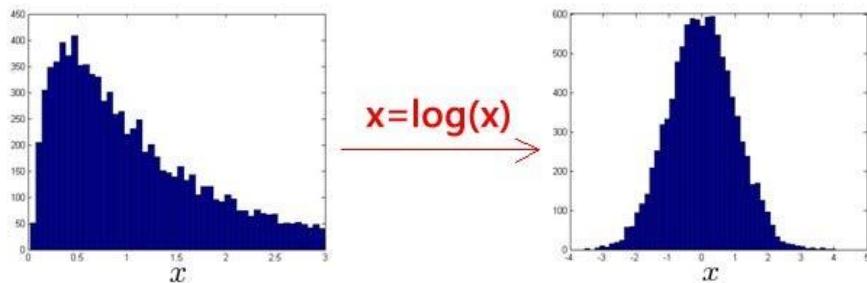
异常检测	监督学习
非常少量的正向类（异常数据 $y=1$ ），大量的负向类（ $y=0$ ）	同时有大量的正向类和负向类
许多不同种类的异常，非常难根据非常少量的正向类数据来训练算法。	有足够多的正向类实例，足够用于训练算法，未来遇到的正向类实例可能与训练集中的非常近似。
未来遇到的异常可能与已掌握的异常非常的不同。	
例如： 1. 欺诈行为检测 2. 生产（例如飞机引擎） 3. 检测数据中心的计算机运行状况	例如： 1. 邮件过滤器 2. 天气预报 3. 肿瘤分类

15.6 选择特征

参考视频: [15 - 6 - Choosing What Features to Use \(12 min\).mkv](#)

对于异常检测算法，我们使用的特征是至关重要的，下面谈谈如何选择特征：

异常检测假设特征符合高斯分布，如果数据的分布不是高斯分布，异常检测算法也能够工作，但是最好还是将数据转换成高斯分布，例如使用对数函数： $x = \log(x+c)$ ，其中 c 为非负常数；或者 $x=x^c$, c 为 0-1 之间的一个分数，等方法。



误差分析：

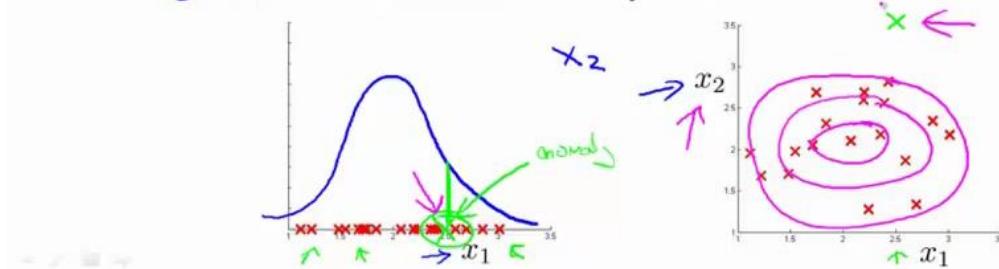
一个常见的问题是些异常的数据可能也会有较高的 $p(x)$ 值，因而被算法认为是正常的。这种情况下误差分析能够帮助我们，我们可以分析那些被算法错误预测为正常的数据，观察能否找出一些问题。我们可能能从问题中发现我们需要增加一些新的特征，增加这些新特征后获得的新算法能够帮助我们更好地进行异常检测。

异常检测误差分析：

Want $p(x)$ large for normal examples x .
 $p(x)$ small for anomalous examples x .

Most common problem:

$p(x)$ is comparable (say, both large) for normal and anomalous examples



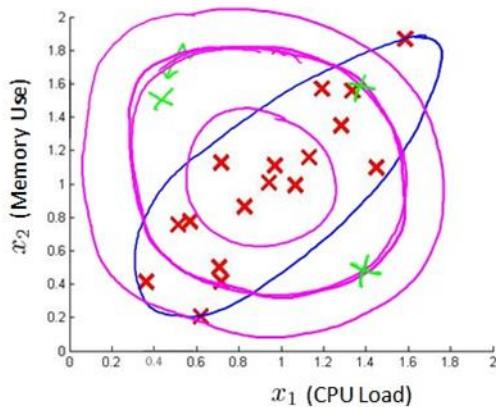
我们通常可以通过将一些相关的特征进行组合，来获得一些新的更好的特征（异常数据的该特征值异常地大或小），例如，在检测数据中心的计算机状况的例子中，我们可以用 CPU 负载与网络通信量的比例作为一个新的特征，如果该值异常地大，便有可能意味着该服务器是陷入了一些问题中。

15.7 多元高斯分布（可选）

参考视频: [15 - 7 - Multivariate Gaussian Distribution \(Optional\) \(14 min\).mkv](#)

假使我们有两个相关的特征，而且这两个特征的值域范围比较宽，这种情况下，一般的高斯分布模型可能不能很好地识别异常数据。其原因在于，一般的高斯分布模型尝试的是去同时抓住两个特征的偏差，因此创造出一个比较大的判定边界。

下图中是两个相关特征，洋红色的线（根据 ϵ 的不同其范围可大可小）是一般的高斯分布模型获得的判定边界，很明显绿色的 x 所代表的数据点很可能是异常值，但是其 $p(x)$ 值却仍然在正常范围内。多元高斯分布将创建像图中蓝色曲线所示的判定边界。



在一般的高斯分布模型中，我们计算 $p(x)$ 的方法是：通过分别计算每个特征对应的几率然后将其累乘起来，在多元高斯分布模型中，我们将构建特征的协方差矩阵，用所有的特征一起来计算 $p(x)$ 。

我们首先计算所有特征的平均值，然后再计算协方差矩阵：

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T = \frac{1}{m} (X - \mu)^T (X - \mu)$$

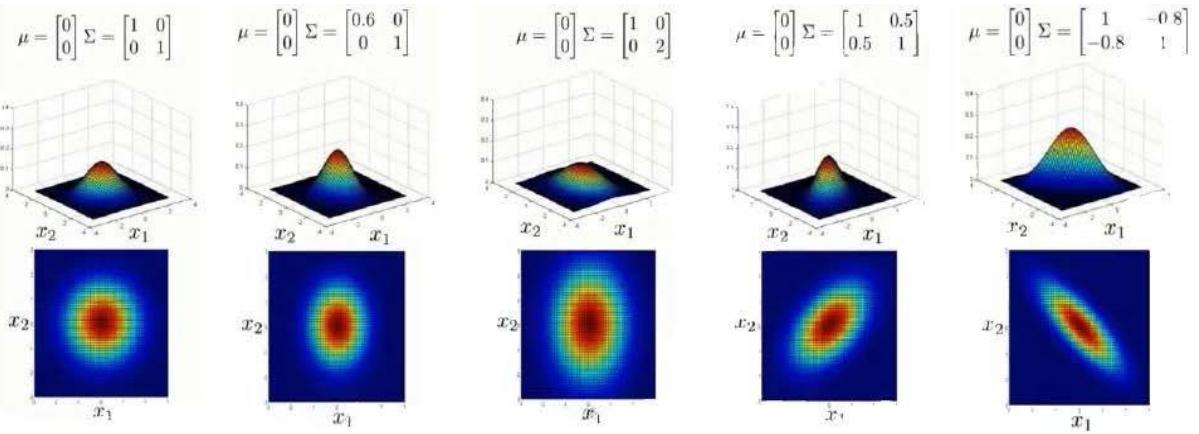
注：其中 μ 是一个向量，其每一个单元都是原特征矩阵中一行数据的均值。最后我们计算多元高斯分布的 $p(x)$ ：

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

其中：

$|\Sigma|$ 是定矩阵，在 Octave 中用 `det(sigma)` 计算

Σ^{-1} 是逆矩阵，下面我们来看看协方差矩阵是如何影响模型的：



上图是 5 个不同的模型，从左往右依次分析：

1. 是一个一般的高斯分布模型
2. 通过协方差矩阵，令特征 1 拥有较小的偏差，同时保持特征 2 的偏差
3. 通过协方差矩阵，令特征 2 拥有较大的偏差，同时保持特征 1 的偏差
4. 通过协方差矩阵，在不改变两个特征的原有偏差的基础上，增加两者之间的正相关性
5. 通过协方差矩阵，在不改变两个特征的原有偏差的基础上，增加两者之间的负相关性

多元高斯分布模型与原高斯分布模型的关系：

可以证明的是，原本的高斯分布模型是多元高斯分布模型的一个子集，即像上图中的第 1、2、3、4 个例子所示，如果协方差矩阵只在对角线的单位上有非零的值时，即为原本的高斯分布模型了。

原高斯分布模型和多元高斯分布模型的比较：

原高斯分布模型	多元高斯分布模型
不能捕捉特征之间的相关性 但可以通过将特征进行组合的方法来解决	自动捕捉特征之间的相关性
计算代价低，能适应大规模的特征	计算代价较高 训练集较小时也同样适用 必须要有 $m > n$ ，不然的话协方差矩阵

	不可逆的，通常需要 $m > 10n$ 另外特征冗余也会导致协方差矩阵不可逆
--	--

原高斯分布模型被广泛使用着，如果特征之间在某种程度上存在相互关联的情况，我们可以通过构造新特征的方法来捕捉这些相关性。

如果训练集不是太大，并且没有太多的特征，我们可以使用多元高斯分布模型。

15.8 使用多元高斯分布进行异常检测（可选）

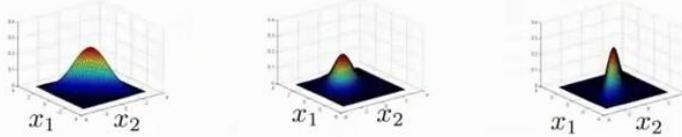
参考视频: [15 - 8 - Anomaly Detection using the Multivariate Gaussian Distribution \(Optional\) \(14 min\).mkv](#)

在我们谈到的最后一个视频，关于多元高斯分布，看到的一些建立的各种分布模型，当你改变参数， μ 和 Σ 。在这段视频中，让我们用这些想法，并应用它们制定一个不同的异常检测算法。

要回顾一下多元高斯分布和多元正态分布：

Parameters μ, Σ

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$



分布有两个参数， μ 和 Σ 。其中 μ 这一个 n 维向量和 Σ 的协方差矩阵，是一种 n 乘 n 的矩阵。而这里的公式 x 的概率，如按 μ 和参数化 Σ ，和你的变量 μ 和 Σ ，你可以得到一个范围的不同分布一样，你知道的，这些都是三个样本，那些我们在以前的视频看过了。

因此，让我们谈谈参数拟合或参数估计问题：

我有一组样本 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ 是一个 n 维向量，我想我的样本来自一个多元高斯分布。我如何尝试估计我的参数 μ 和 Σ 以及标准公式？

估计他们是你设置 μ 是你的训练样本的平均值。

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

并设置 Σ ：

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

这其实只是当我们使用 PCA 算法时候，有 Σ 时写出来。所以你只需插入上述两个公式，这会给你你估计的参数 μ 和你估计的参数 Σ 。所以，这里给出的数据集是你如何估计 μ 和 Σ 。让我们以这种方法而只需将其插入到异常检测算法。那么，我们如何把所有这一切共同开发

一个异常检测算法？

Anomaly detection with the multivariate Gaussian

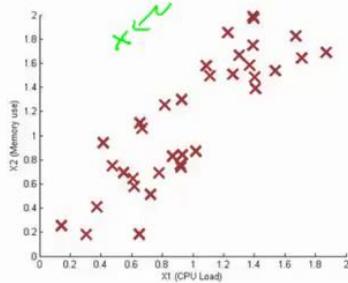
1. Fit model $p(x)$ by setting

$$\begin{cases} \mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \\ \Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T \end{cases}$$

2. Given a new example x , compute

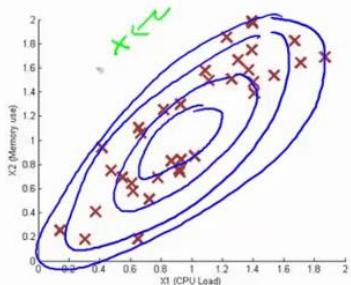
$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

Flag an anomaly if $p(x) < \varepsilon$



Andrew Ng

首先，我们把我们的训练集，和我们的拟合模型，我们计算 $P(x)$ ，要知道，设定 μ 和描述的一样 Σ 。



如图，该分布在中央最多，越到外面的圈的范围越小。

并在该点是出路这里的概率非常低。

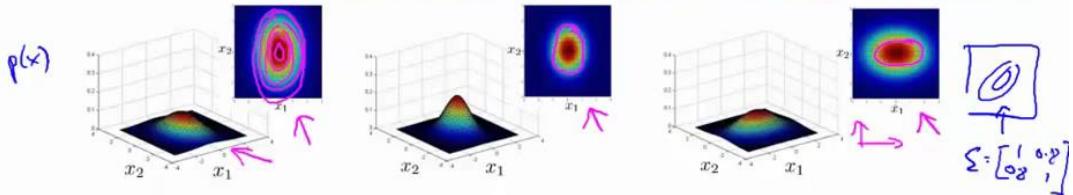
原始模型与多元高斯模型的关系如图：

其中：协方差矩阵 Σ 为：

$$\Sigma = \begin{bmatrix} \sigma_1^2 & \cdots & \cdots \\ \vdots & \ddots & \cdots \\ \cdots & \cdots & \sigma_n^2 \end{bmatrix}$$

Relationship to original model

Original model: $p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$



Corresponds to multivariate Gaussian

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

where $\Sigma = \begin{bmatrix} \sigma_1^2 & \dots & \dots \\ \dots & \ddots & \dots \\ \dots & \dots & \sigma_n^2 \end{bmatrix}$

Andrew Ng

原始模型和多元高斯分布比较如图:

→ Original model

$$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$$

Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values.

$$\rightarrow x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$$

→ Computationally cheaper (alternatively, scales better to large n) $n=10,000, m=100,000$

OK even if m (training set size) is small

vs.

→ Multivariate Gaussian

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

→ Automatically captures correlations between features

$$\Sigma \in \mathbb{R}^{n \times n}$$

$$\Sigma^{-1}$$

Computationally more expensive

$$\rightarrow \Sigma \sim \frac{n^2}{2}$$

Must have $m > n$ or else Σ is non-invertible.

$$\left. \begin{array}{l} \rightarrow x_1 = x_2 \\ \times x_3 = x_4 \\ + x_5 \end{array} \right]$$

$$\rightarrow m \geq 10n$$

Andrew Ng

十六、推荐系统(Recommender Systems)

16.1 问题形式化

参考视频: [16 - 1 - Problem Formulation \(8 min\).mkv](#)

在接下来的视频中，我想讲一下推荐系统。我想讲推荐系统有两个原因：

第一、仅仅因为它是机器学习中的一个重要的应用。在过去几年，我偶尔访问硅谷不同的技术公司，我常和工作在这儿致力于机器学习应用的人们聊天，我常问他们，最重要的机器学习的应用是什么，或者，你最想改进的机器学习应用有哪些。我最常听到的答案是推荐系统。现在，在硅谷有很多团体试图建立很好的推荐系统。因此，如果你考虑网站像亚马逊，或网飞公司或易趣，或 iTunes Genius，有很多的网站或系统试图推荐新产品给用户。如，亚马逊推荐新书给你，网飞公司试图推荐新电影给你，等等。这些推荐系统，根据浏览你过去买过什么书，或过去评价过什么电影来判断。这些系统会带来很大一部分收入，比如为亚马逊和像网飞这样的公司。因此，对推荐系统性能的改善，将对这些企业的有实质性和直接的影响。

推荐系统是个有趣的问题，在学术机器学习中因此，我们可以去参加一个学术机器学习会议，推荐系统问题实际上受到很少的关注，或者，至少在学术界它占了很小的份额。但是，如果你看正在发生的事情，许多有能力构建这些系统的科技企业，他们似乎在很多企业中占据很高的优先级。这是我为什么在这节课讨论它的原因之一。

我想讨论推荐系统地第二个原因是：这个班视频的最后几集我想讨论机器学习中的一些大思想，并和大家分享。这节课我们也看到了，对机器学习来说，特征是很重要的，你所选择的特征，将对你学习算法的性能有很大的影响。因此，在机器学习中有一种大思想，它针对一些问题，可能并不是所有的问题，而是一些问题，有算法可以为你自动学习一套好的特征。因此，不要试图手动设计，而手写代码这是目前为止我们常干的。有一些设置，你可以有一个算法，仅仅学习其使用的特征，推荐系统就是类型设置的一个例子。还有很多其它的，但是通过推荐系统，我们将领略一小部分特征学习的思想，至少，你将能够了解到这方面的一个例子，我认为，机器学习中的大思想也是这样。因此，让我们开始讨论推荐系统问题形式化。

我们从一个例子开始定义推荐系统的问题。

假使我们是一个电影供应商，我们有 5 部电影和 4 个用户，我们要求用户为电影打分。

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	6
Romance forever	5	?	0	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	0	5
Swords vs. karate	0	0	?	4

前三部电影是爱情片，后两部则是动作片，我们可以看出 Alice 和 Bob 似乎更倾向与爱情片，而 Carol 和 Dave 似乎更倾向与动作片。并且没有一个用户给所有的电影都打过分。我们希望构建一个算法来预测他们每个人可能会给他们没看过的电影打多少分，并以此作为推荐的依据。

下面引入一些标记：

n_u 代表用户的数量

n_m 代表电影的数量

$r(i,j)$ 如果用户 i 给电影 j 评过分则 $r(i,j)=1$

$y^{(i,j)}$ 代表用户 i 给电影 j 的评分

m_j 代表用户 j 评过分的电影的总数

16.2 基于内容的推荐系统

参考视频: [16 - 2 - Content Based Recommendations \(15 min\).mkv](#)

在一个基于内容的推荐系统算法中，我们假设对于我们希望推荐的东西有一些数据，这些数据是有关这些东西的特征。

在我们的例子中，我们可以假设每部电影都有两个特征，如 x_1 代表电影的浪漫程度， x_2 代表电影的动作程度。

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

则每部电影都有一个特征向量，如 $x^{(1)}$ 是第一部电影的特征向量为 $[0.9 \ 0]$ 。

下面我们要基于这些特征来构建一个推荐系统算法。假设我们采用线性回归模型，我们可以针对每一个用户都训练一个线性回归模型，如 $\theta^{(1)}$ 是第一个用户的模型的参数。于是，我们有：

$\theta^{(j)}$ 用户 j 的参数向量

$x^{(i)}$ 电影 i 的特征向量

对于用户 j 和电影 i ，我们预测评分为： $(\theta^{(j)})^T x^{(i)}$

代价函数

针对用户 j ，该线性回归模型的代价为预测误差的平方和，加上归一化项：

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

其中 $i:r(i,j)$ 表示我们只计算那些用户 j 评过分的电影。在一般的线性回归模型中，误差项和归一项应该都是乘以 $1/2m$ ，在这里我们将 m 去掉。并且我们不对方差项 θ_0 进行归一化处理。

上面的代价函数只是针对一个用户的，为了学习所有用户，我们将所有用户的代价函数求和：

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

如果我们要用梯度下降法来求解最优解，我们计算代价函数的偏导数后得到梯度下降的更新公式为：

$$\begin{aligned} \theta_k^{(j)} &:= \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k = 0) \\ \theta_k^{(j)} &:= \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0) \end{aligned}$$

16.3 协同过滤

参考视频: [16 - 3 - Collaborative Filtering \(10 min\).mkv](#)

在之前的基于内容的推荐系统中,对于每一部电影,我们都掌握了可用的特征,使用这些特征训练出了每一个用户的参数。相反地,如果我们拥有用户的参数,我们可以学习得出电影的特征。

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

但是如果我们既没有用户的参数,也没有电影的特征,这两种方法都不可行了。协同过滤算法可以同时学习这两者。

我们的优化目标便改为同时针对 x 和 θ 进行。

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

对代价函数求偏导数的结果如下:

$$x_k^{(i)} := x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

注: 在协同过滤从算法中,我们通常不使用方差项,如果需要的话,算法会自动学得。

协同过滤算法使用步骤如下:

1. 初始 $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$, $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$ 为一些随机小值
2. 使用梯度下降算法最小化代价函数
3. 在训练完算法后, 我们预测 $(\theta^{(j)})^T x^{(i)}$ 为用户 j 给电影 i 的评分

通过这个学习过程获得的特征矩阵包含了有关电影的重要数据,这些数据不总是人能读懂的,但是我们可以用这些数据作为给用户推荐电影的依据。

例如,如果一位用户正在观看电影 $x^{(i)}$,我们可以寻找另一部电影 $x^{(j)}$,依据两部电影的特征向量之间的距离 $||x^{(i)} - x^{(j)}||$ 的大小。

16.4 协同过滤算法

参考视频: [16 - 4 - Collaborative Filtering Algorithm \(9 min\).mkv](#)

协同过滤优化目标:

给定 $x^{(1)}, \dots, x^{(n_m)}$, 估计 $\theta^{(1)}, \dots, \theta^{(n_u)}$:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

给定 $\theta^{(1)}, \dots, \theta^{(n_u)}$, 估计 $x^{(1)}, \dots, x^{(n_m)}$:

同时最小化 $x^{(1)}, \dots, x^{(n_m)}$ 和 $\theta^{(1)}, \dots, \theta^{(n_u)}$:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) =$$

$$\frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$

16.5 矢量化：低秩矩阵分解

参考视频: [16 - 5 - Vectorization Low Rank Matrix Factorization \(8 min\).mkv](#)

协同过滤:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

推出评分:

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

找到相关影片:

For each product i , we learn a feature vector $\underline{x}^{(i)} \in \mathbb{R}^n$.

$\rightarrow x_1 = \text{romance}$, $x_2 = \text{action}$, $x_3 = \text{comedy}$, $x_4 = \dots$

How to find $\underset{\uparrow}{\text{movies } j}$ related to $\underset{\wedge}{\text{movie } i}$?

small $\|x^{(i)} - x^{(j)}\| \rightarrow$ movie j and i are "similar"

5 most similar movies to movie i :

Find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

16.6 推荐工作上的细节：均值归一化

参考视频: [16 - 6 - Implementational Detail Mean Normalization \(9 min\).mkv](#)

让我们来看下面的用户评分数据:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	$Y =$
Love at last	5	5	0	0	?	$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \end{bmatrix}$
Romance forever	5	?	?	0	?	$\begin{bmatrix} ? & 4 & 0 & ? & ? \end{bmatrix}$
Cute puppies of love	?	4	0	?	?	$\begin{bmatrix} 0 & 0 & 5 & 4 & ? \end{bmatrix}$
Nonstop car chases	0	0	5	4	?	$\begin{bmatrix} 0 & 0 & 5 & 0 & ? \end{bmatrix}$
Swords vs. karate	0	0	5	?	?	

如果我们新增一个用户 Eve，并且 Eve 没有为任何电影评分，那么我们以什么为依据为 Eve 推荐电影呢？

我们首先需要对结果 Y 矩阵进行均值归一化处理，将每一个用户对某一部电影的评分减去所有 用户对该电影评分的平均值：

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

然后我们利用这个新的 Y 矩阵来训练算法。如果我们要用新训练出的算法来预测评分，则需要将平均值重新加回去，预测 $(\theta^{(j)})^T(x^{(i)}) + \mu_i$ 对于 Eve，我们的新模型会认为她给每部电影的评分都是该电影的平均分。

第 10 周

十七、大规模机器学习(Large Scale Machine Learning)

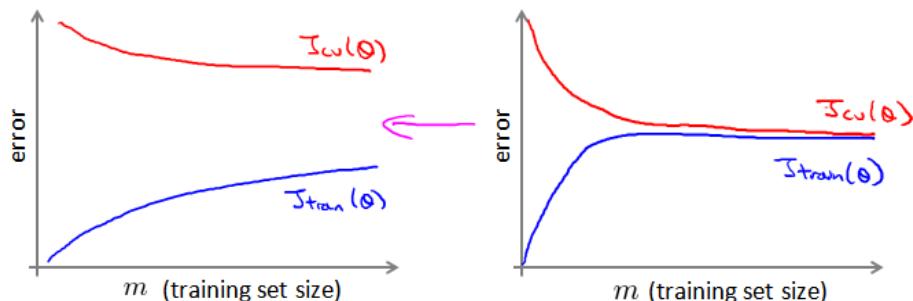
17.1 大型数据集的学习

参考视频: [17 - 1 - Learning With Large Datasets \(6 min\).mkv](#)

如果我们有一个低方差的模型，增加数据集的规模可以帮助你获得更好的结果。我们应该怎样应对一个有 100 万条记录的训练集？

以线性回归模型为例，每一次梯度下降迭代，我们都需要计算训练集的误差的平方和，如果我们的学习算法需要有 20 次迭代，这便已经是非常大的计算代价。

首先应该做的事是去检查一个这么大规模的训练集是否真的必要，也许我们只用 1000 个训练集也能获得较好的效果，我们可以绘制学习曲线来帮助判断。



17.2 随机梯度下降法

参考视频: [17 - 2 - Stochastic Gradient Descent \(13 min\).mkv](#)

如果我们一定需要一个大规模的训练集, 我们可以尝试使用随机梯度下降法来代替批量梯度下降法。

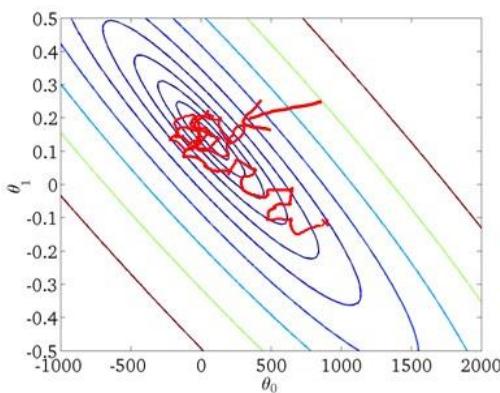
在随机梯度下降法中, 我们定义代价函数为一个单一训练实例的代价:

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_{\theta}(x^{(i)}) - y^{(i)})^2$$

随机梯度下降算法为: 首先对训练集随机“洗牌”, 然后:

```
Repeat (usually anywhere between 1-10) {
    for i=1:m {
        theta_j := theta_j - alpha(h_theta(x^(i)) - y^(i)) * x_j^(i)
        (for j=0:n)
    }
}
```

随机梯度下降算法在每一次计算之后便更新参数 θ , 而不需要首先将所有的训练集求和, 在梯度下降算法还没有完成一次迭代时, 随机梯度下降算法便已经走出了很远。但是这样的算法存在的问题是, 不是每一步都是朝着“正确”的方向迈出的。因此算法虽然会逐渐走向全局最小值的位置, 但是可能无法站到那个最小值的那一点, 而是在最小值点附近徘徊。



17.3 微型批量梯度下降

参考视频: [17 - 3 - Mini-Batch Gradient Descent \(6 min\).mkv](#)

微型批量梯度下降算法是介于批量梯度下降算法和随机梯度下降算法之间的算法, 每计算常数 b 次训练实例, 便更新一次参数 Θ 。

```
Repeat {
    for i=1:m {
         $\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$ 
        (for j=0:n)

        i += 10;
    }
}
```

通常我们会令 b 在 2-100 之间。这样做的好处在于, 我们可以用向量化的方式来循环 b 个训练实例, 如果我们用的线性代数函数库比较好, 能够支持平行处理, 那么算法的总体表现将不受影响 (与随机梯度下降相同)。

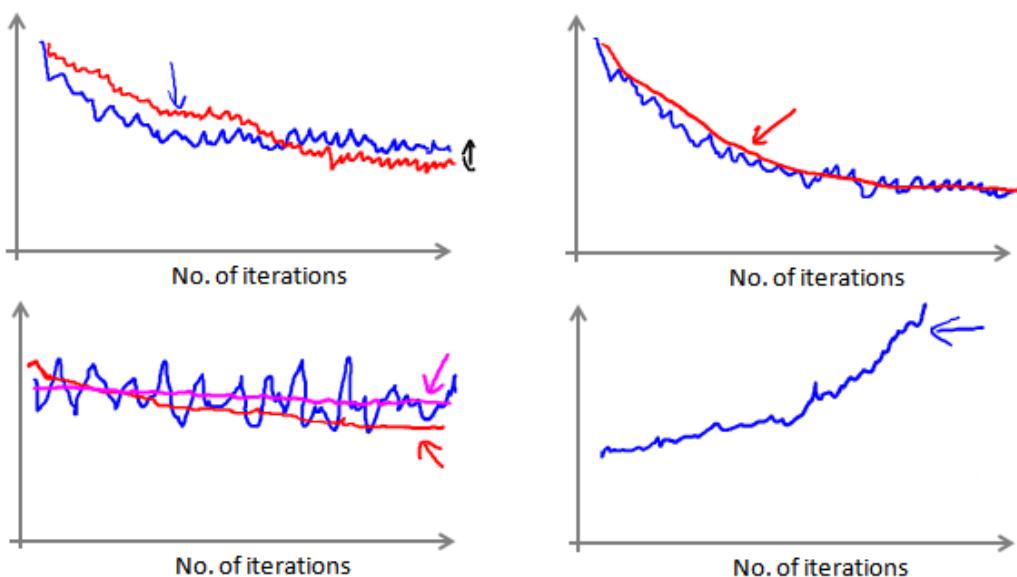
17.4 随机梯度下降收敛

参考视频: [17 - 4 - Stochastic Gradient Descent Convergence \(12 min\).mkv](#)

现在我们介绍随机梯度下降算法的调试，以及学习率 α 的选取。

在批量梯度下降中，我们可以令代价函数 J 为迭代次数的函数，绘制图表，根据图表来判断梯度下降是否收敛。但是，在大规模的训练集的情况下，这是不现实的，因为计算代价太大了。

在随机梯度下降中，我们在每一次更新 Θ 之前都计算一次代价，然后每 X 次迭代后，求出这 X 次对训练实例计算代价的平均值，然后绘制这些平均值与 X 次迭代的次数之间的函数图表。



当我们绘制这样的图表时，可能会得到一个颠簸不平但是不会明显减少的函数图像（如上面左下图中蓝线所示）。我们可以增加 X 来使得函数更加平缓，也许便能看出下降的趋势了（如上面左下图中红线所示）；或者可能函数图表仍然是颠簸不平且不下降的（如洋红色线所示），那么我们的模型本身可能存在一些错误。

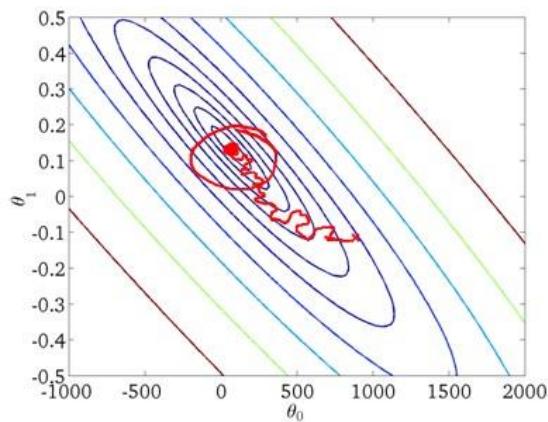
如果我们得到的曲线如上面右下方所示，不断地上升，那么我们可能会需要选择一个较小的学习率 α 。

我们也可以令学习率随着迭代次数的增加而减小，例如令：

$$\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$$

随着我们不断地靠近全局最小值，通过减小学习率，我们迫使算法收敛而非在最小值附

近徘徊。但是通常我们不需要这样做便能有非常好的效果了，对 α 进行调整所耗费的计算通常不值得



17.5 在线学习

参考视频: [17 - 5 - Online Learning \(13 min\).mkv](#)

在线学习算法指的是对数据流而非离线的静态数据集的学习。许多在线网站都有持续不断的用户流，对于每一个用户，网站希望能在不将数据存储到数据库中便顺利地进行算法学习。

假使我们正在经营一家物流公司，每当一个用户询问从地点 A 至地点 B 的快递费用时，我们给用户一个报价，该用户可能选择接受 ($y=1$) 或不接受 ($y=0$)。

现在，我们希望构建一个模型，来预测用户接受报价使用我们的物流服务的可能性。因此报价 是我们的一个特征，其他特征为距离，起始地点，目标地点以及特定的用户数据。模型的输出是 $p(y=1)$ 。

在线学习的算法与随机梯度下降算法有些类似，我们对单一的实例进行学习，而非对一个提前定义的训练集进行循环。

```
Repeat forever (as long as the website is running) {
    Get (x, y) corresponding to the current user
     $\theta_j := \theta_j - \alpha(h_{\theta}(x) - y)x_j$ 
    (for j=0:n)
}
```

一旦对一个数据的学习完成了，我们便可以丢弃该数据，不需要再存储它了。这种方式的好处在于，我们的算法可以很好的适应用户的倾向性，算法可以针对用户的当前行为不断地更新模型以适应该用户。

每次交互事件并不只产生一个数据集，例如，我们一次给用户提供 3 个物流选项，用户选择 2 项，我们实际上可以获得 3 个新的训练实例，因而我们的算法可以一次从 3 个实例中学习并更新模型。

17.6 映射化简和数据并行

参考视频: [17 - 6 - Map Reduce and Data Parallelism \(14 min\).mkv](#)

映射化简和数据并行对于大规模机器学习问题而言是非常重要的概念。之前提到, 如果我们用批量梯度下降算法来求解大规模数据集的最优解, 我们需要对整个训练集进行循环, 计算偏导数和代价, 再求和, 计算代价非常大。如果我们能够将我们的数据集分配给多台计算机, 让每一台计算机处理数据集的一个子集, 然后我们将计所的结果汇总在求和。这样的方法叫做映射简化。

具体而言, 如果任何学习算法能够表达为, 对训练集的函数的求和, 那么便能将这个任务分配给多台计算机 (或者同一台计算机的不同 CPU 核心), 以达到加速处理的目的。

例如, 我们有 400 个训练实例, 我们可以将批量梯度下降的求和任务分配给 4 台计算机进行处理:

Machine no	Samples	Computation	Result
1	1-100	$temp^{(1)} = \sum_{i=1}^{100} (h_\theta(x^i) - y^i)x_j^i$	
2	101-200	$temp^{(2)} = \sum_{i=101}^{200} (h_\theta(x^i) - y^i)x_j^i$	
3	201-300	$temp^{(3)} = \sum_{i=201}^{300} (h_\theta(x^i) - y^i)x_j^i$	
4	301-400	$temp^{(4)} = \sum_{i=301}^{400} (h_\theta(x^i) - y^i)x_j^i$	$\theta_j = \theta_j - \alpha \frac{1}{400} (temp^{(1)} + temp^{(2)} + temp^{(3)} + temp^{(4)})$

很多高级的线性代数函数库已经能够利用多核 CPU 的多个核心来并行地处理矩阵运算, 这也是算法的向量化实现如此重要的缘故 (比调用循环快)。

十八、应用实例：图片文字识别(Application Example: Photo OCR)

18.1 问题描述和流程图

参考视频: [18 - 1 - Problem Description and Pipeline \(7 min\).mkv](#)

图像文字识别应用所作的事是，从一张给定的图片中识别文字。这比从一份扫描文档中识别文字要复杂的多。



为了完成这样的工作，需要采取如下步骤：

1. 文字侦测（Text detection）——将图片上的文字与其他环境对象分离开来
2. 字符切分（Character segmentation）——将文字分割成一个个单一的字符
3. 字符分类（Character classification）——确定每一个字符是什么 可以用任务流程图来表达这个问题，每一项任务可以由一个单独的小队来负责解决：

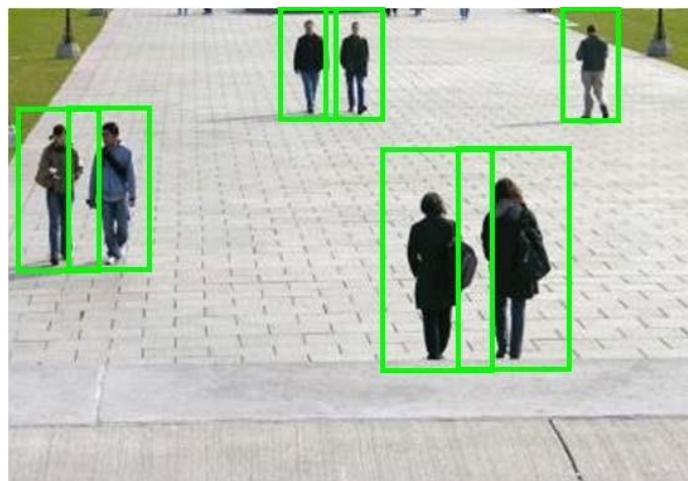


18.2 滑动窗口

参考视频: [18 - 2 - Sliding Windows \(15 min\).mkv](#)

滑动窗口是一项用来从图像中抽取对象的技术。假使我们需要在一张图片中识别行人，首先要做的是用许多固定尺寸的图片来训练一个能够准确识别行人的模型。然后我们以之前训练识别行人的模型时所采用的图片尺寸在我们要进行行人识别的图片上进行剪裁，然后将剪裁得到的切片交给模型，让模型判断是否为行人，然后在图片上滑动剪裁区域重新进行剪裁，将新剪裁的切片也交给模型进行判断，如此循环直至将图片全部检测完。

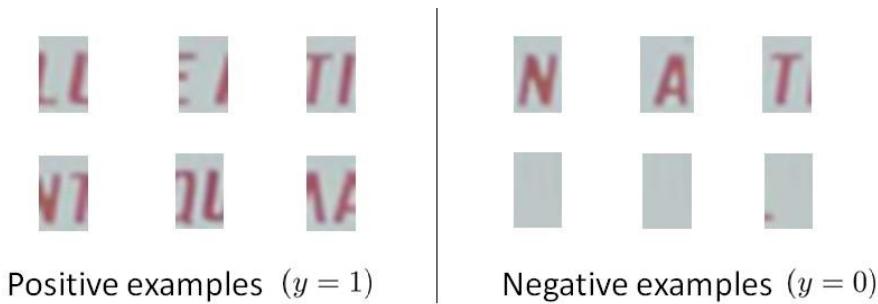
一旦完成后，我们按比例放大剪裁的区域，再以新的尺寸对图片进行剪裁，将新剪裁的切片按比例缩小至模型所采纳的尺寸，交给模型进行判断，如此循环。



滑动窗口技术也被用于文字识别，首先训练模型能够区分字符与非字符，然后，运用滑动窗口技术识别字符，一旦完成了字符的识别，我们将识别得出的区域进行一些扩展，然后将重叠的区域进行合并。接着我们以宽高比作为过滤条件，过滤掉高度比宽度更大的区域（认为单词的长度通常比高度要大）。下图中绿色的区域是经过这些步骤后被认为是文字的区域，而红色的区域是被忽略的。



以上便是文字侦测阶段。下一步是训练一个模型来完成将文字分割成一个个字符的任务，需要的训练集由单个字符的图片和两个相连字符之间的图片来训练模型。



模型训练完后，我们仍然是使用滑动窗口技术来进行字符识别。



以上便是字符切分阶段。最后一个阶段是字符分类阶段，利用神经网络、支持向量机或者逻辑回归算法训练一个分类器即可。

18.3 获取大量数据和人工数据

参考视频: [18 - 3 - Getting Lots of Data and Artificial Data \(16 min\).mkv](#)

如果我们的模型是低方差的，那么获得更多的数据用于训练模型，是能够有更好的效果的。问题在于，我们怎样获得数据，数据不总是可以直接获得的，我们有可能需要人工地创造一些数据。

以我们的文字识别应用为例，我们可以字体网站下载各种字体，然后利用这些不同的字体配上各种不同的随机背景图片创造出一些用于训练的实例，这让我们能够获得一个无限大的训练集。这是从零开始创造实例。

另一种方法是，利用已有的数据，然后对其进行修改，例如将已有的字符图片进行一些扭曲、旋转、模糊处理。只要我们认为实际数据有可能和经过这样处理后的数据类似，我们便可以用这样的方法来创造大量的数据。

有关获得更多数据的几种方法：

1. 人工数据合成
2. 手动收集、标记数据
3. 众包

18.4 上限分析：哪部分管道的接下去做

参考视频: [18 - 4 - Ceiling Analysis What Part of the Pipeline to Work on Next \(14 min\).mkv](#)

在机器学习的应用中，我们通常需要通过几个步骤才能进行最终的预测，我们如何能够知道哪一部分最值得我们花时间和精力去改善呢？这个问题可以通过上限分析来回答。

回到我们的文字识别应用中，我们的流程图如下：



流程图中每一部分的输出都是下一部分的输入，上限分析中，我们选取一部分，手工提供 100% 正确的输出结果，然后看应用的整体效果提升了多少。假使我们的例子中总体效果为 72% 的正确率。

如果我们令文字侦测部分输出的结果 100% 正确，发现系统的总体效果从 72% 提高到了 89%。这意味着我们很可能会希望投入时间精力来提高我们的文字侦测部分。

接着我们手动选择数据，让字符切分输出的结果 100% 正确，发现系统的总体效果只提升了 1%，这意味着，我们的字符切分部分可能已经足够好了。

最后我们手工选择数据，让字符分类输出的结果 100% 正确，系统的总体效果又提升了 10%，这意味着我们可能也会应该投入更多的时间和精力来提高应用的总体表现。

Component	Accuracy
Overall system	72%
Text detection	89%
Character segmentation	90%
Character recognition	100%

↓ 17%
↓ 1%
↓ 10%

十九、总结(Conclusion)

19.1 总结和致谢

参考视频: [19 - 1 - Summary and Thank You \(5 min\).mkv](#)

欢迎来到《机器学习》课的最后一段视频。我们已经一起学习很长一段时间了。在最后这段视频中，我想快速地回顾一下这门课的主要内容，然后简单说几句想说的话。

作为这门课的结束时间，那么我们学到了些什么呢？在这门课中，我们花了大量的时间介绍了诸如线性回归、逻辑回归、神经网络、支持向量机等等一些监督学习算法，这类算法具有带标签的数据和样本，比如 $x^{(i)}$ 、 $y^{(i)}$ 。

然后我们也花了很多时间介绍无监督学习。例如 K-均值聚类、用于降维的主成分分析，以及当你只有一系列无标签数据 $x^{(i)}$ 时的异常检测算法。

当然，有时带标签的数据，也可以用于异常检测算法的评估。此外，我们也花时间讨论了一些特别的应用或者特别的话题，比如说推荐系统。以及大规模机器学习系统，包括并行系统和映射化简方法，还有其他一些特别的应用。比如，用于计算机视觉技术的滑动窗口分类算法。

最后，我们还提到了很多关于构建机器学习系统的实用建议。这包括了怎样理解某个机器学习算法是否正常工作的原因，所以我们谈到了偏差和方差的问题，也谈到了解决方差问题的正则化，同时我们也讨论了怎样决定接下来怎么做的问题，也就是说当你在开发一个机器学习系统时，什么工作才是接下来应该优先考虑的问题。因此我们讨论了学习算法的评价法。介绍了评价矩阵，比如：查准率、召回率以及 F1 分数，还有评价学习算法比较实用的训练集、交叉验证集和测试集。我们也介绍了学习算法的调试，以及如何确保学习算法的正常运行，于是我们介绍了一些诊断法，比如学习曲线，同时也讨论了误差分析、上限分析等等内容。

所有这些工具都能有效地指引你决定接下来应该怎样做，让你把宝贵的时间用在刀刃上。现在你已经掌握了很多机器学习的工具，包括监督学习算法和无监督学习算法等等。

但除了这些以外，我更希望你现在不仅仅只是认识这些工具，更重要的是掌握怎样有效地利用这些工具来建立强大的机器学习系统。所以，以上就是这门课的全部内容。如果你跟着我们的课程一路走来，到现在，你应该已经感觉到自己已经成为机器学习方面的专家了吧？

我们都知道，机器学习是一门对科技、工业产生深远影响的重要学科，而现在，你已经完全具备了应用这些机器学习工具来创造伟大成就的能力。我希望你们中的很多人都能在相应的领域，应用所学的机器学习工具，构建出完美的机器学习系统，开发出无与伦比的产品和应用。并且我也希望你们通过应用机器学习，不仅仅改变自己的生活，有朝一日，还要让更多的人生活得更加美好！

我也想告诉大家，教这门课对我来讲是一种享受。所以，谢谢大家！

最后，在结束之前，我还想再多说一点：那就是，也许不久以前我也是一个学生，即使是现在，我也尽可能挤出时间听一些课，学一些新的东西。所以，我深知要坚持学完这门课是很需要花一些时间的，我知道，也许你是一个很忙的人，生活中有很多很多事情要处理。正因如此，你依然挤出时间来观看这些课程视频。我知道，很多视频的时间都长达数小时，你依然花了好多时间来做这些复习题。你们中好多人，还愿意花时间来研究那些编程练习，那些又长又复杂的编程练习。我对你们表示衷心的感谢！我知道你们很多人在这门课中都非常努力，很多人都在这门课上花了很多时间，很多人都为这门课贡献了自己的很多精力。所以，我衷心地希望你们能从这门课中有所收获！

最后我想说！再次感谢你们选修这门课程！

Andrew Ng