

斯坦福大学 2023 机器学习教程中文笔记

课程概述

课程地址: <https://www.coursera.org/course/ml>

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) **supervised** 监督学习（参数/非参数算法，支持向量机，核函数，神经网络）。
- (二) **unsupervised** 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。

目录

第 1 周	1
1、 引言 Introduction.....	1
1.1 欢迎 Welcome	1
1.2 机器学习是什么？ What is Machine Learning	3
1.3 监督学习 Supervised Learning	4
1.4 无监督学习 Unsupervised Learning.....	7
2、 单变量线性回归 Linear Regression with One Variable	10
2.1 模型表示 Model Representation	10
2.2 代价函数 Cost Function	13
2.3 代价函数的直观理解 : Cost Function - Intuition.....	15
2.4 代价函数的直观理解 Cost Function - Intuition	16
2.5 梯度下降 Gradient Descent.....	17
2.6 梯度下降的直观理解 Gradient Descent Intuition.....	18
2.7 梯度下降的线性回归 Gradient Descent For Linear Regression	21
第 2 周	23
3、 多变量线性回归 Linear Regression with Multiple Variables	23
3.1 多维特征 Multiple Features.....	23
3.2 多变量梯度下降 Gradient Descent for Multiple Varaibles.....	24
3.2. 1 Vectorization	25
3.3 梯度下降法实践-特征缩放 Gradient Descent in Practice I - Feature Scaling	27
3.4 梯度下降法实践-学习率 Gradient Descent in Practice II - Learning Rate ..	29
3.5 特征和多项式回归 Feature Engineering: Feature and Polynomial Regression	30
第 3 周	31
4、 逻辑回归 Logistic Regression	31
4.1 分类问题 Classifictation	31
4.2 假说表示 Sigmoid Function (Hypothesis Representation).....	33
4.3 判定边界 Decisoin Boundary.....	35
4.4 代价函数 Cost Function	37
4.5 简化成本函数的梯度下降 Gradient Descent Implementation for logistic regressoin	39
5、 正则化 Regularization.....	43
5.1 过拟合的问题 The Problem of Overfitting	43
5.2 Address Overfitting	45
5.3 正規化的代價函數 Cost Function with Regularization	47
5.4 正则化线性回归 Regularizaed Linear Regression	51
5.5 正则化的逻辑回归模型 Regularized Logistic Regression	53
第 4 周	55
6、 神经网络 Neural Networks	55
6.1 非线性假设 Non-linear Hypotheses	55
6.2 神经元和大脑 Neurons and the Brain	57

6.3 模型表示 Model Representation	58
6.4 模型表示 Inference: amking predictions (forward propogation)	67
6.4.1 Forward propagation from scratch	75
6.4.2 Forward propogation in Numpy	81
6.4.3 Predictions.....	82
6.5 Network function	83
6.5.1 Path to Artifical general intelligence (AGI).....	84
6.5.2 How Neural Networks Implemented Efficiently.....	84
6.5.3 Vectorization: Matrix multiplication.....	85
6.5.4 TensorFlow Implementations	88
6.5.5 TensorFlow Neural Netwrks Training Details	89
6.5.6 Activation Functions	91
6.5.7 Choosing activation functions	92
6.5.8 Why do we need activation functions?.....	94
6.6 特征和直观理解 Examples and Intuitions	95
7、 多类分类 Multiclass Classification	99
7.1 Cost functions using softmax regression.	100
7.1.1 Cost Function.....	100
7.2 Neural network with softmax output	101
7.2.1 Improved implementation of softmax.....	102
7.3 The same applies to logistic regression.	104
7.4 Classificatoin with Multiple Outputs (Multiple labels)	104
7.5 Advanced Optimization	105
7.5.1 Adam Algorithm Intuition.....	105
7.5.2 ADAM implementation	106
7.6 Additional Layer Types	106
7.6.1 Dense layer	106
7.6.2 Convolution layer:.....	107
第5周	108
8、 反向传播算法 & 代价函数 BackPropagation and Cost Function.....	108
8.1 反向传播算法 Backpropagation Algorithm	108
8.3 随机初始化 Random Initialization.....	111
8.4 综合起来 Putting It Together.....	113
8.5 自主驾驶 Autonomous Driving	114
第6周	116
9、 应用机器学习的建议 Advice for Applying Machine Learning	116
9.1 决定下一步做什么 Deciding What to Try Next.....	116
9.2 评估一个假设 Evaluating a Model	119
9.3 模型选择和交叉验证集 Model Selection and Training/Cross Validation/Test Sets	123
9.4 诊断偏差和方差 Diagnosing Bias vs. Variance	126
9.5 正则化和偏差/方差 Regularization and Bias_Variance	128
9.6 学习曲线 Learning Curves	130
9.7 决定下一步做什么 Deciding What to Do Next Revisited	132

10.	机器学习系统的设计 Machine Learning Development Process	135
10.1	Developing a Machine Learning System and the Iterative Loop	135
10.2	误差分析 Error Analysis.....	136
10.3	Adding data.....	137
10.4	Transfer learning.....	141
10.5	Full cycle of a machine learning project	143
10.6	类偏斜的误差度量 Error Metrics for Skewed Classes.....	145
10.7	Precision and Recall Metrics	145
10.8	查准率和查全率之间的权衡 Trade-Offs between Precision and Recall ..	148
11.	Decision tree model	151
11.1	Basic Concept of Decision Trees:.....	151
11.2	Learning Process	152
11.3	Entropy as a Measure of Impurity	154
11.3.1	Entropy as Impurity Function $H(\mathbf{p1})$	155
11.4	Choosing a split: information gain (reduction in Entropy)	155
11.4.1	General formula of information gain.....	156
11.5	Decision Tree Process	157
11.6	One-hot encoding for categorical features that are no longer binary classification.	157
11.7	Continuous valued features.....	158
11.8	Regression Trees	159
11.8.1	Choosing a split: reduction in Variance.....	160
12.	Tree ensembles (multiple decision tree)	160
12.1	Sampling with replacement: key technique in tree ensemble	161
12.2	Random forest algorithm: sample with replacement implementation in tree ensemble.....	161
12.2.1	Randomizing the feature choice when $n \geq 1000$	161
12.3	XGBoost (eXtreme Gradient Boosting): an improved version of random forest	162
12.3.1	Boosted trees intuition: correct trained tree faults	162
12.3.2	Using XGBoost	162
12.4	When to use decision trees.	163
12.4.1	Decision Trees and Tree ensembles.....	163
12.4.2	Neural Networks.....	163

第 1 周

1、 引言 Introduction

1.1 欢迎 Welcome

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，我希望能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？What is Machine Learning

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** 的定义可以回溯到 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。

1.3 监督学习 Supervised Learning

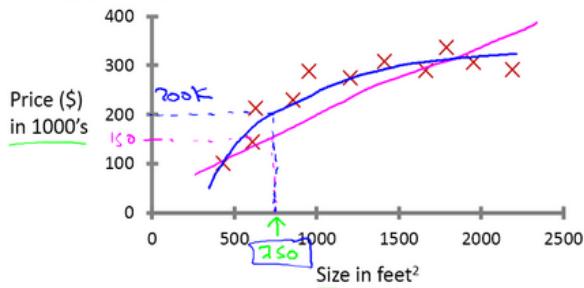
在这段视频中，我要定义可能是最常见一种机器学习问题：那就是监督学习。我将在后面正式定义监督学习。

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。

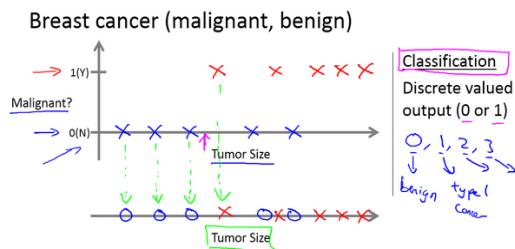
regression 回归这个词的意思是，我们在试着推测出这一系列连续值属性。

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。

Classification: output class and output category

Classification predicts categories (categories can be non-numbers)

Classification bpredicts small number nad finite of possible categories



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。

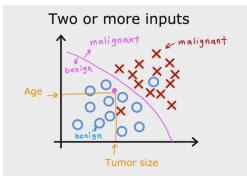
分类指的是，我们试着推测出离散的输出值：0 或 1 良性或恶性，而事实上在分类问题中，输出可能不止两个值。比如说可能有三种乳腺癌，所以你希望预测离散输出 0、1、2、3。0 代表良性，1 表示第 1 类乳腺癌，2 表示第 2 类癌症，3 表示第 3 类，但这也是分类问题。

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 classifications 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。



boundary line to assist decision making.

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。

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。

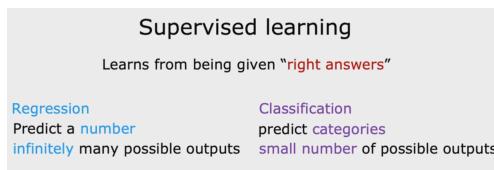
现在来个小测验：假设你经营着一家公司，你想开发学习算法来处理这两个问题：

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。

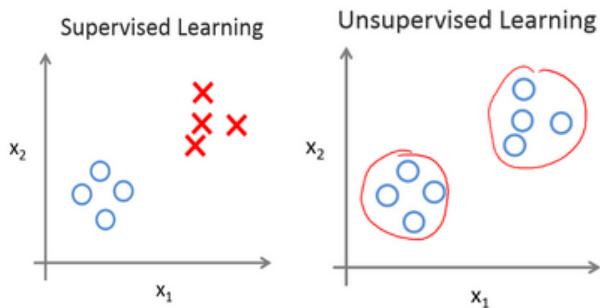
问题二是一个分类问题，因为我会把预测的值，用 0 来表示账户未被盗，用 1 表示账户曾经被盗过。所以我们根据账号是否被盗过，把它们定为 0 或 1，然后用算法推测一个账号是 0 还是 1，因为只有少数的离散值，所以我把它归为分类问题。



以上就是监督学习的内容。

1.4 无监督学习 Unsupervised Learning

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。



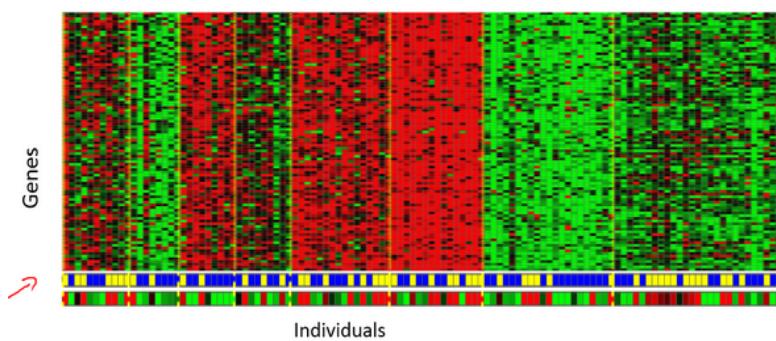
上个视频中，已经介绍了监督学习。回想当时的数据集，如图表所示，这个数据集中每条数据都已经标明是阴性或阳性，即是良性或恶性肿瘤。所以，对于监督学习里的每条数据，我们已经清楚地知道，训练集对应的正确答案，是良性或恶性了。

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 **label** 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。

Clustering 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。

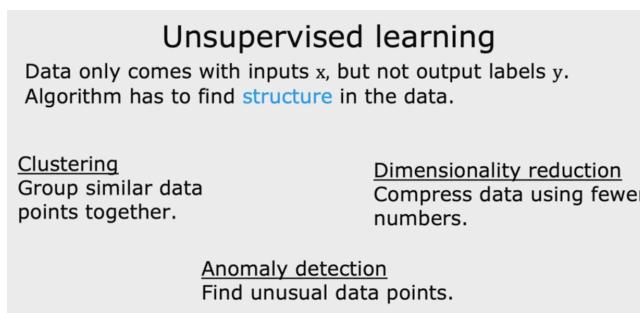
Clustering algorithm is a type of unsupervised algorithm. No right answer but only possible grouping.

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。



Supervise learning: labeling or filtering.

Unsupervised learning: no correct answer but only clustering/grouping to find the structure

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：

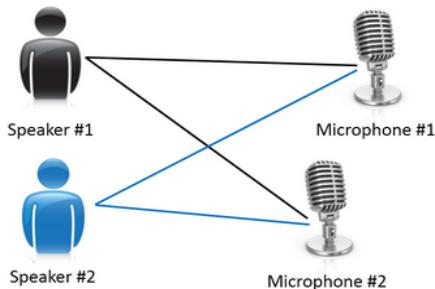
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,

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。（编者注：这个是当时的情况，现在 **Python** 变主流了）

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**。

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。

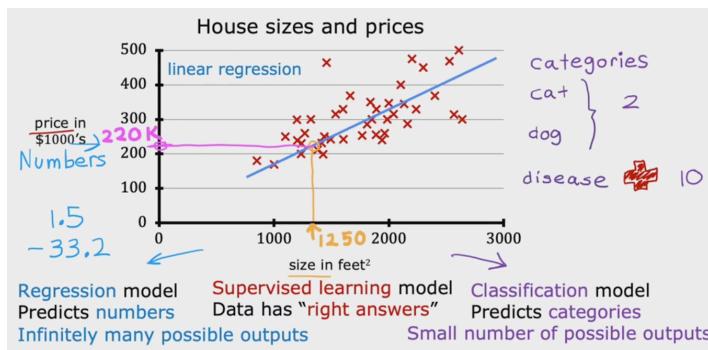
2. 单变量线性回归 Linear Regression with One Variable

2.1 模型表示 Model Representation

我们的第一个学习算法是线性回归算法。在这段视频中，你会看到这个算法的概况，更重要的是你将会了解监督学习过程完整的流程。

让我们通过一个例子来开始：这个例子是预测住房价格的，我们要使用一个数据集，数据集包含俄勒冈

州波特兰市的住房价格。在这里，我要根据不同房屋尺寸所售出的价格，画出我的数据集。比方说，如果你朋友的房子是 1250 平方尺大小，你要告诉他们这房子能卖多少钱。那么，你可以做的一件事就是构建一个模型，也许是条直线，从这个数据模型上来看，也许你可以告诉你的朋友，他能以大约 220000(美元)左右的价格卖掉这个房子。这就是监督学习算法的一个例子。



Regression model for predicting numbers where there are infinitely many possible outputs.

Classification model is to predict categories where there are only small possible outputs.

它被称作监督学习是因为对于每个数据来说，我们给出了“正确的答案”，即告诉我们：根据我们的数据来说，房子实际的价格是多少，而且，更具体来说，这是一个回归问题。回归一词指的是，我们根据之前的数据预测出一个准确的输出值，对于这个例子就是价格，同时，还有另一种最常见的监督学习方式，叫做分类问题，当我们想要预测离散的输出值，例如，我们正在寻找癌症肿瘤，并想要确定肿瘤是良性的还是恶性的，这就是 0/1 离散输出的问题。更进一步来说，在监督学习中我们有一个数据集，这个数据集被称训练集。

我将在整个课程中用小写的 m 来表示训练样本的数目。

Each cross corresponds to one row in the data table.

Terminology:

以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 total number of training sample

x 代表特征 feature / 输入变量 “input” variable

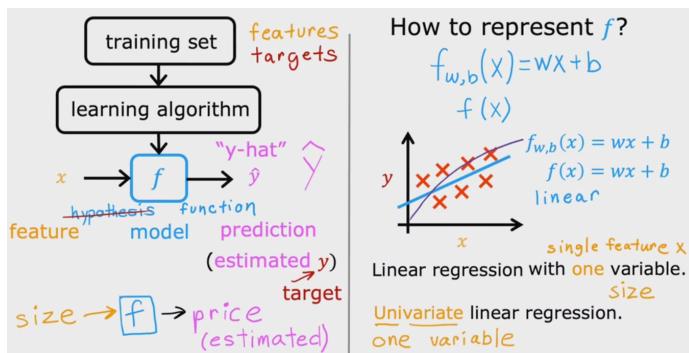
y 代表目标变量 target /输出变量 “output” variable

(x, y) 代表训练集中的实例 single training example

$(x^{(i)}, y^{(i)})$ 代表第 i 个观察实例 i^{th} training sample (not exponent)

h 代表学习算法的解决方案或函数也称为假设 (hypothesis)

\hat{y} means the estimation of target (actual) y



这就是一个监督学习算法的工作方式，我们可以看到这里有我们的训练集里房屋价格 我们把它喂给我们的学习算法，学习算法的工作了，然后输出一个函数，通常表示为小写 h 表示。 h 代表 hypothesis(假设)， h 表示一个函数，输入是房屋尺寸大小，就像你朋友想出售的房屋，因此 h 根据输入的 x 值来得出 y 值， y 值对应房子的价格 因此， h 是一个从 x 到 y 的函数映射。

我将选择最初的使用规则 h 代表 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 ？

How to represent h ?

一种可能的表达方式为: $h_{\theta}(x) = \theta_0 + \theta_1 x$ ，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。

General		Description	Python (if applicable)
Notation			
a		scalar, non bold	
\mathbf{a}		vector, bold	
Regression			
\mathbf{x}	Training Example feature values (in this lab - Size (1000 sqft))		$\mathbf{x_train}$
y	Training Example targets (in this lab Price (1000s of dollars))		y_train
$x^{(i)}, y^{(i)}$	i_{th} Training Example		x_i, y_i
m	Number of training examples		m
w	parameter: weight		w
b	parameter: bias		b
$f_{w,b}(x^{(i)})$	The result of the model evaluation at $x^{(i)}$ parameterized by w, b : $f_{w,b}(x^{(i)}) = wx^{(i)} + b$		f_wb

2.2 代价函数 Cost Function

在这段视频中我们将定义代价函数的概念，这有助于我们弄清楚如何把最有可能的直线与我们的数据相拟合。如图：

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

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

θ_i 's: Parameters

How to choose θ_i 's?

在线性回归中我们有一个像这样的训练集， m 代表了训练样本的数量，比如 $m = 47$ 。而我们的假设函数，也就是用来进行预测的函数，是这样的线性函数形式： $h_{\theta}(x) = \theta_0 + \theta_1 x$ 。

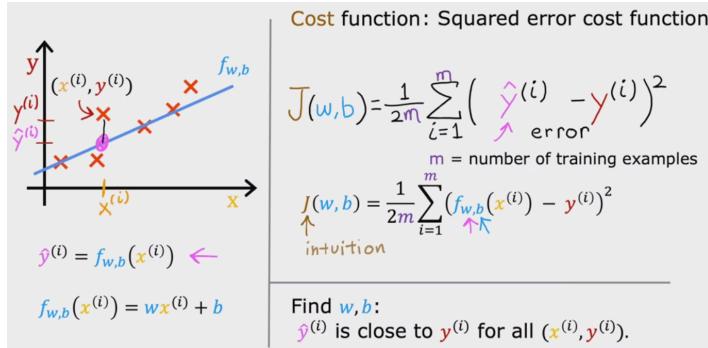
接下来我们会引入一些术语我们现在要做的便是为我们的模型选择合适的参数（parameters, or weights） θ_0 和 θ_1 ，在房价问题这个例子中便是直线的斜率和在y轴上的截距。

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）。

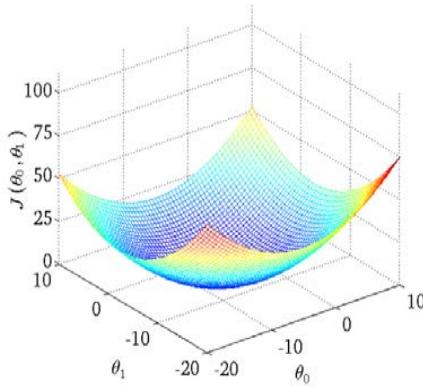
我们的目标便是选择出可以使得建模误差的平方和能够最小的模型参数。即使得代价函数 $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$ 最小。

Include the division by two since the error is the maximum error square, where to calculate the rot

mean square average error, the error should also be divided by $(\sqrt{2})^2$



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。

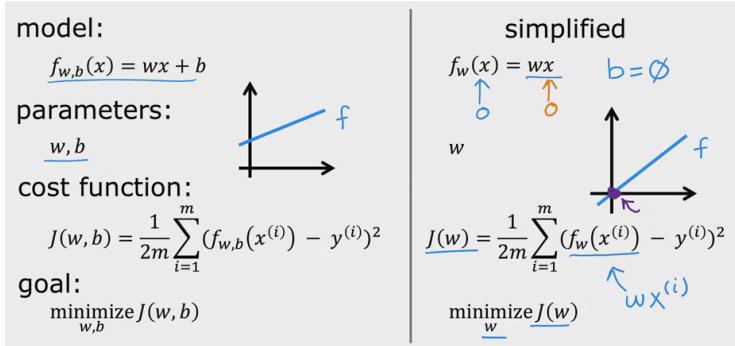
代价函数也被称作平方误差函数，有时也被称为平方误差代价函数。我们之所以要求出误差的平方和，是因为误差平方代价函数，对于大多数问题，特别是回归问题，都是一个合理的选择。还有其他的代价函数也能很好地发挥作用，但是平方误差代价函数可能是解决回归问题最常用的手段了。

在后续课程中，我们还会谈论其他的代价函数，但我们刚刚讲的选择是对于大多数线性回归问题非常合理的。

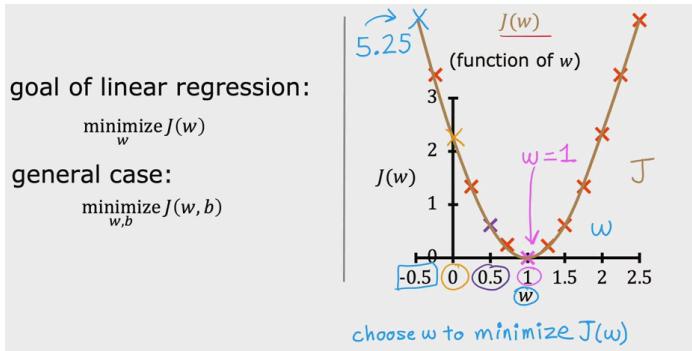
也许这个函数 $J(\theta_0, \theta_1)$ 有点抽象，可能你仍然不知道它的内涵，在接下来的几个视频里，我们要更进一步解释代价函数 J 的工作原理，并尝试更直观地解释它在计算什么，以及我们使用它的目的。

2.3 代价函数的直观理解 : Cost Function - Intuition

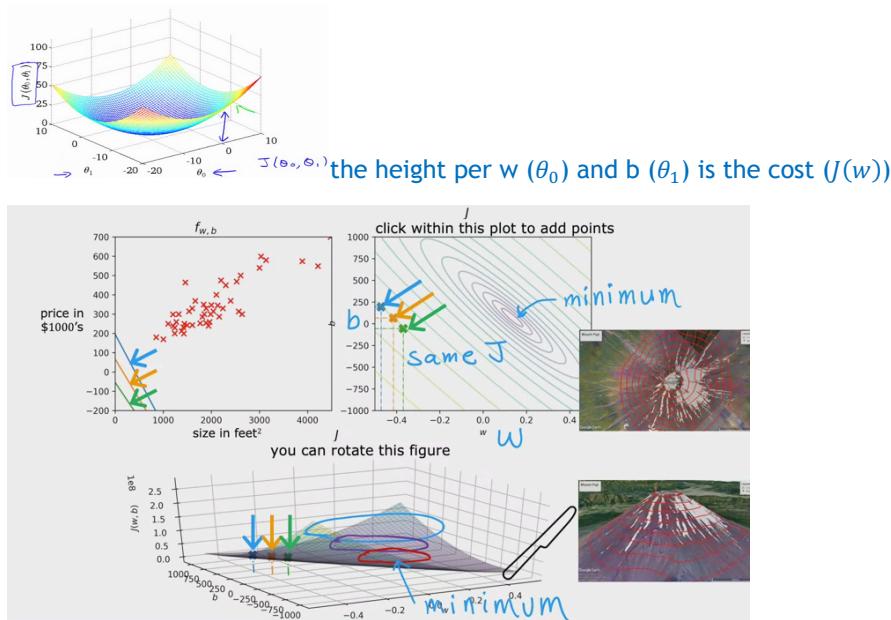
在上一个视频中，我们给了代价函数一个数学上的定义。在这个视频里，让我们通过一些例子来获取一些直观的感受，看看代价函数到底是在干什么。



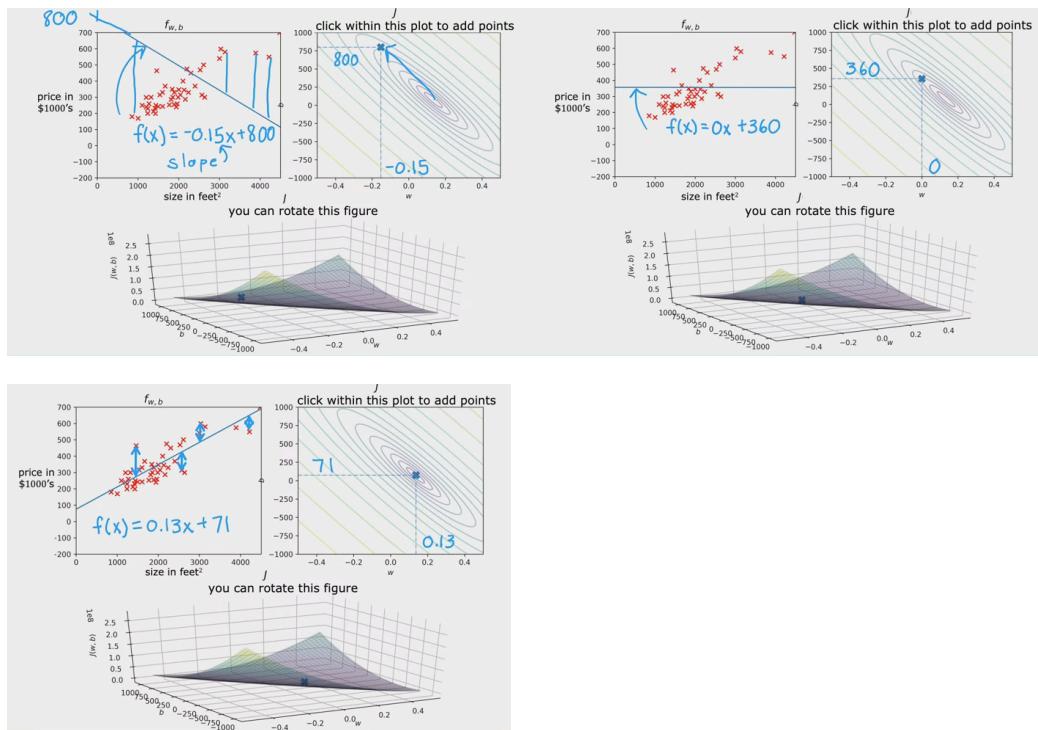
the computed error for each x given w , plot the cost function per w . Repeat the step for each a range of value w , hence for each value of w , there is a cost $J(w)$. choosing a w to minimize $J(w)$ will hence be the best fitting line for $f_{w,b}(x)$.



2.4 代价函数的直观理解 Cost Function - Intuition



代价函数的样子，等高线图 contour plot，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。



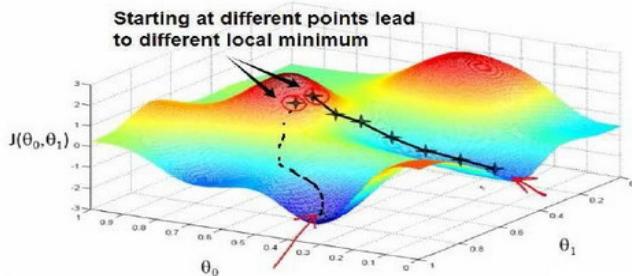
通过这些图形，我希望你能更好地理解这些代价函数 J 所表达的值是什么样的，它们对应的假设是什么样的，以及什么样的假设对应的点，更接近于代价函数 J 的最小值。

我们真正需要的是编写程序来找出这些最小化代价函数的 θ_0 和 θ_1 的值，在下一节视频中，我们将介绍一种算法，能够自动地找出能使代价函数 J 最小化的参数 θ_0 和 θ_1 的值。

2.5 梯度下降 Gradient Descent

Gradient descent is applicable for general linear regression or any function to minimize or maximize a function.

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。



想象一下你正站立在山的这一点上，站立在你想象的公园这座红色山上，在梯度下降算法中，我们要做的就是旋转 360 度，看看我们的周围，并问自己要在某个方向上，用小碎步尽快下山。这些小碎步需要朝什么方向？如果我们站在山坡上的这一点，你看一下周围，你会发现最佳的下山方向，你再看看周围，然后再一次想想，我应该从什么方向迈着小碎步下山？然后你按照自己的判断又迈出一步，重复上面的步骤，从这个新的点，你环顾四周，并决定从什么方向将会最快下山，然后又迈进了一小步，并依此类推，直到你接近局部最低点的位置。

Implementing gradient descent

批量梯度下降（batch gradient descent）算法的公式为：

Gradient descent algorithm:

$$\mathbf{w} = \mathbf{w} - \alpha \frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}, \mathbf{b})$$

$$\mathbf{b} = \mathbf{b} - \alpha \frac{\partial}{\partial \mathbf{b}} J(\mathbf{w}, \mathbf{b})$$

Repeat the simultaneously updating of the two coefficients (w and b, **simultaneously**) updates until reaching convergence.

其中 α 是学习率 (learning rate), which is always a positive number, 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。

Repeat until convergence	Learning rate Derivative	Code	Math
$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$ $b = b - \alpha \frac{\partial}{\partial b} J(w, b)$	Simultaneously update w and b	$\alpha = c$ $\alpha = \alpha + 1$	$a = c$ $a = a + 1$ a==c
Correct: Simultaneous update			Incorrect
$tmp_w = w - \alpha \frac{\partial}{\partial w} J(w, b)$ $tmp_b = b - \alpha \frac{\partial}{\partial b} J(w, b)$ $w = tmp_w$ $b = tmp_b$			$tmp_w = w - \alpha \frac{\partial}{\partial w} J(w, b)$ $w = tmp_w$ $tmp_b = b - \alpha \frac{\partial}{\partial b} J(w, b)$ $b = tmp_b$

2.6 梯度下降的直观理解 Gradient Descent Intuition

在之前的视频中, 我们给出了一个数学上关于梯度下降的定义, 本次视频我们更深入研究一下, 更直观地感受一下这个算法是做什么的, 以及梯度下降算法的更新过程有什么意义。梯度下降算法如下:

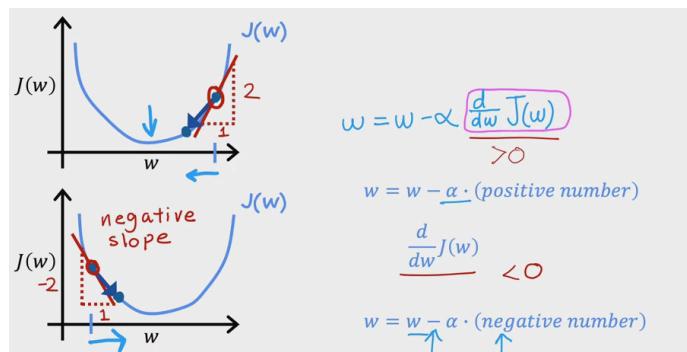
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \rightarrow w = w - \alpha \cdot \frac{\partial}{\partial w} J(w)$$

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。

$$\frac{\partial}{\partial w} J(w) \text{ is the slope}$$

求导的目的, 基本上可以说取这个红点的切线, 就是这样一条红色的直线, 刚好与函数相切于这一点, 让我们看看这条红色直线的斜率, 就是这条刚好与函数曲线相切的这条直线, 这条直线的斜率正好是这个三角形的高度除以这个水平长度, 现在, 这条线有一个正斜率, 也就是说它有正导数, 因此, 我得到的新的 θ_1 , θ_1 更新后等于 θ_1 减去一个正数乘以 α 。

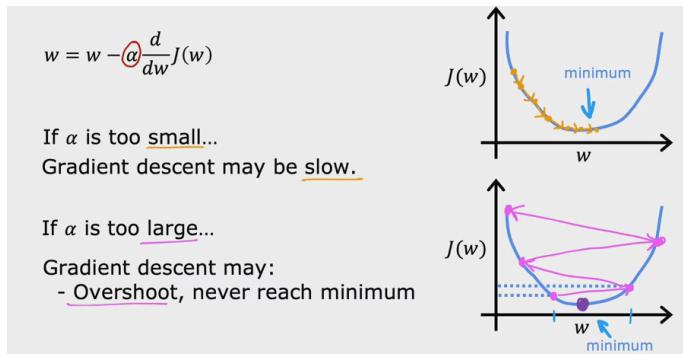
这就是我梯度下降法的更新规则: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$



让我们来看看如果 α 太小或 α 太大会出现什么情况：

如果 α 太小了，即我的学习速率太小，结果就是只能这样像小宝宝一样一点点地挪动，去努力接近最低点，这样就需要很多步才能到达最低点，所以如果 α 太小的话，可能会很慢，因为它会一点点挪动，它会需要很多步才能到达全局最低点。

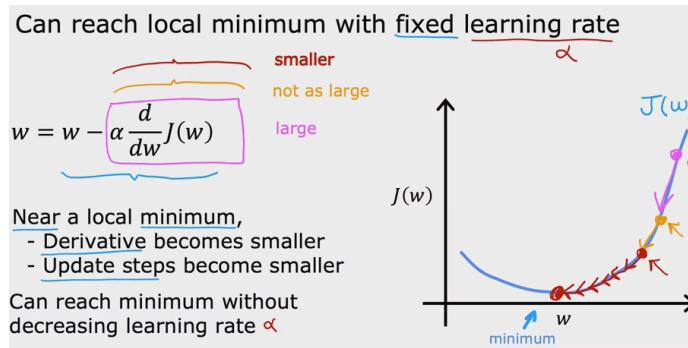
如果 α 太大，那么梯度下降法可能会越过最低点，甚至可能无法收敛，下一次迭代又移动了一大步，越过一次，又越过一次，一次次越过最低点，直到你发现实际上离最低点越来越远，所以，如果 α 太大，它会导致无法收敛，甚至发散。



如果我们预先把 θ_1 放在一个局部的最低点，你认为下一步梯度下降法会怎样工作？

假设你将 θ_1 初始化在局部最低点，在这儿，它已经在一个局部的最优处或局部最低点。结果是局部最优点的导数 $\frac{\partial}{\partial \omega} J(\omega)$ 将等于零，因为它是那条切线的斜率。这意味着你已经在局部最优点，它使得 θ_1 不再改变，也就是新的 θ_1 等于原来的 θ_1 ，因此，如果你的参数已经处于局部最低点，那么梯度下降法更新其实什么都没做，它不会改变参数的值。

Can reach local minimum with fixed learning rate? Yes.



我想找到它的最小值，首先初始化我的梯度下降算法，在那个品红色的点初始化，如果我更新一步梯度下降，也许它会带我到这个点，因为这个点的导数是相当陡的。现在，在这个绿色的点，如果我再更新一步，你会发现我的导数 (derivative)，也即斜率，是没那么陡的。随着我接近最低点，我的导数越来越接近零，

所以，梯度下降一步后，新的导数会变小一点点。然后我想再梯度下降一步，在这个绿点，我自然会用一个稍微跟刚才在那个品红点时比，再小一点的一步，到了新的红色点，更接近全局最低点了，因此这点的导数会比在绿点时更小。所以，我再进行一步梯度下降时，我的导数项是更小的， θ_1 更新的幅度就会更小。所以随着梯度下降法的运行，你移动的幅度会自动变得越来越小，直到最终移动幅度非常小，你会发现，已经收敛到局部极小值。

回顾一下，在梯度下降法中，**当我们接近局部最低点时，梯度下降法会自动采取更小的幅度**，这是因为当我们接近局部最低点时，很显然在局部最低时导数等于零，所以当我们接近局部最低时，导数值会自动变得越来越小，所以梯度下降将自动采取较小的幅度，这就是梯度下降的做法。所以实际上没有必要再另外减小 a 。

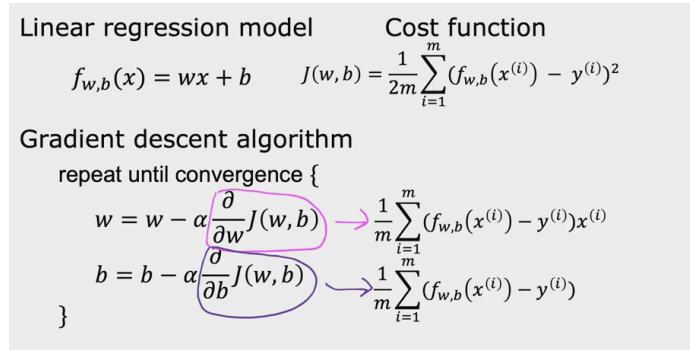
这就是梯度下降算法，你可以用它来最小化任何代价函数 J ，不只是线性回归中的代价函数 J 。

在接下来的视频中，我们要用代价函数 J ，回到它的本质，线性回归中的代价函数。也就是我们前面得出的平方误差函数，结合梯度下降法，以及平方代价函数，我们会得出第一个机器学习算法，即**线性回归算法**。

2.7 梯度下降的线性回归 Gradient Descent For Linear Regression

在以前的视频中我们谈到关于梯度下降算法，梯度下降是很常用的算法，它不仅被用在线性回归上和线性回归模型、平方误差代价函数。在这段视频中，我们要将梯度下降和代价函数结合。我们将用到此算法，并将其应用于具体的拟合直线的线性回归算法里。

梯度下降算法和线性回归算法比较如图：



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：

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

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

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

则算法改写成：

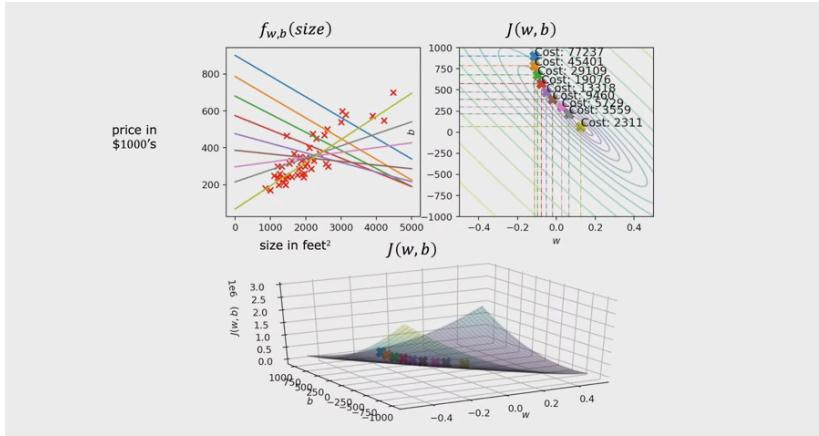
Repeat {

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

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

}

When using a squared error cost function, there will never be multiple local minimum given that the squared error cost function gives a bowl shape and hence a convex function where the deduced local minimum is the global minimum.



我们刚刚使用的算法，有时也称为批量梯度下降。实际上，在机器学习中，通常不太会给算法起名字，但这个名字”**批量梯度下降 batch gradient descent**”，指的是在梯度下降的每一步中，**我们都用到了所有的训练样本 (use all training samples)**，在梯度下降中，在计算微分求导项时，我们需要进行求和运算，所以，在每一个单独的梯度下降中，我们最终都要计算这样一个东西，这个项需要对所有 m 个训练样本求和。因此，批量梯度下降法这个名字说明了我们需要考虑所有这一“批”训练样本，而事实上，有时也有其他类型的梯度下降法，不是这种“批量”型的，不考虑整个的训练集，而是每次只关注训练集中的一些小的子集。在后面的课程中，我们也将介绍这些方法。

如果你之前学过线性代数，有些同学之前可能已经学过高等线性代数，你应该知道有一种计算代价函数 J 最小值的数值解法，不需要梯度下降这种迭代算法。在后面的课程中，我们也会谈到这个方法，它可以在不需要多步梯度下降的情况下，也能解出代价函数 J 的最小值，这是另一种称为正规方程(**normal equations**)的方法。实际上在数据量较大的情况下，梯度下降法比正规方程要更适用一些。

现在我们已经掌握了梯度下降，我们可以在不同的环境中使用梯度下降法，我们还将在不同的机器学习问题中大量地使用它。所以，祝贺大家成功学会你的第一个机器学习算法。

在下一段视频中，告诉你泛化的梯度下降算法，这将使梯度下降更加强大。

第 2 周

3、 多变量线性回归 Linear Regression with Multiple Variables

3.1 多维特征 Multiple Features

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 **total number of features**

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

比方说，上图的

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix},$$

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

如上图的 $x_2^{(2)} = 3, x_3^{(2)} = 2, f_{w,b}(x) = w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b$.

$$\begin{aligned} f_{\vec{w}, b}(\vec{x}) &= w_1x_1 + w_2x_2 + \dots + w_nx_n + b \\ \vec{w} &= [w_1 \ w_2 \ w_3 \ \dots \ w_n] \quad \text{parameters of the model} \\ b &\text{ is a number} \\ \text{vector } \vec{x} &= [x_1 \ x_2 \ x_3 \ \dots \ x_n] \\ f_{\vec{w}, b}(\vec{x}) &= \vec{w} \cdot \vec{x} + b = w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n + b \\ \text{dot product} &\quad \text{multiple linear regression} \end{aligned}$$

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 $m * (n + 1)$ 。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。

3.2 多变量梯度下降 Gradient Descent for Multiple Variables

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

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 θ_j
for $j=0, 1, \dots, n$)

}

当 $n >= 1$ 时，

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

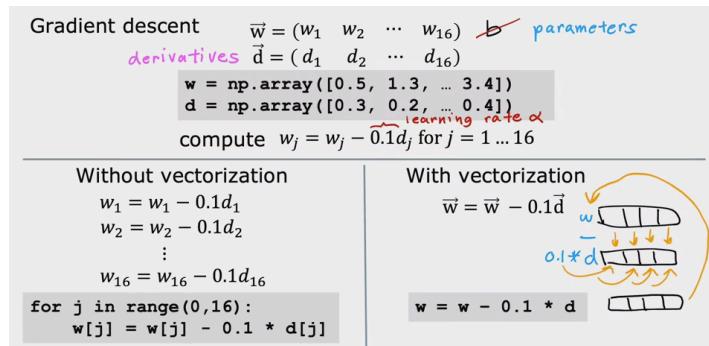
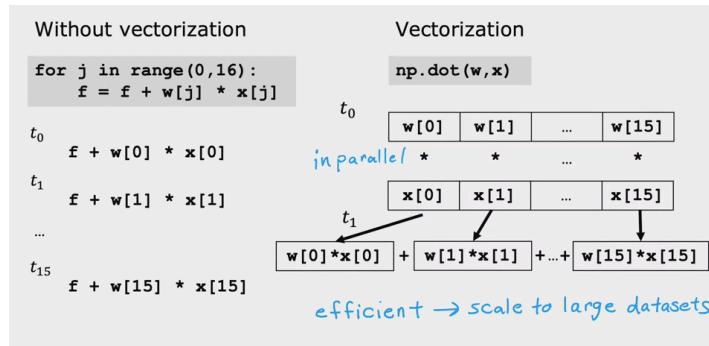
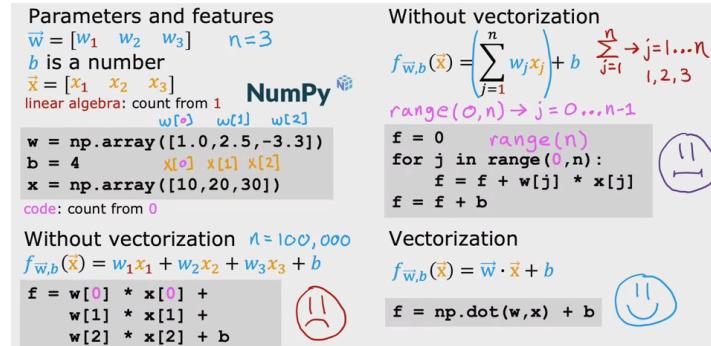
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

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

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。

3.2. 1 Vectorization

Matrix dot product (np.dot in python) is more efficient due to parallelism than serial multiplications.



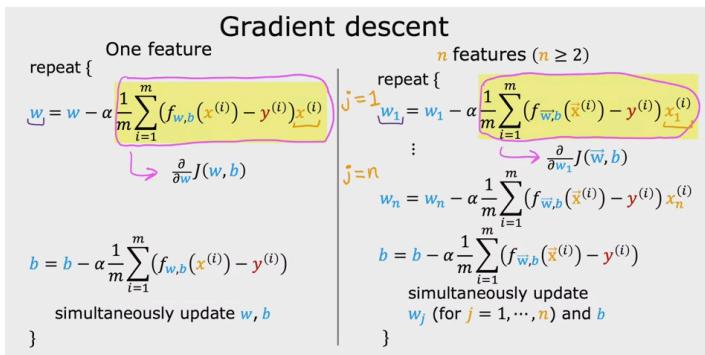
代码示例：

计算代价函数 $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$ 其中： $h_\theta(x) = \theta^T X = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$

Python 代码：

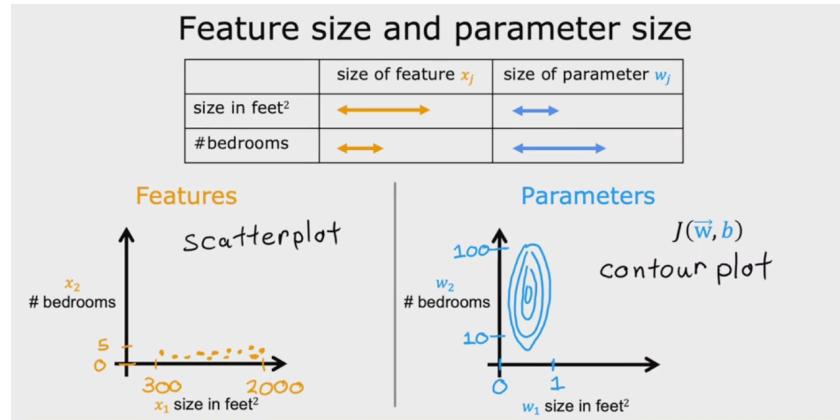
```
def computeCost(X, y, theta):
    inner = np.power(((X * theta.T) - y), 2)
    return np.sum(inner) / (2 * len(X))
```

	Previous notation	Vector notation
Parameters	w_1, \dots, w_n b	$\vec{w} = [w_1 \dots w_n]$ b still a number
Model	$f_{\vec{w}, b}(\vec{x}) = w_1 x_1 + \dots + w_n x_n + b$	$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$
Cost function	$J(w_1, \dots, w_n, b)$	$J(\vec{w}, b)$ dot product
Gradient descent	<pre>repeat { $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w_1, \dots, w_n, b)$ $b = b - \alpha \frac{\partial}{\partial b} J(w_1, \dots, w_n, b)$ }</pre>	<pre>repeat { $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$ $b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$ }</pre>

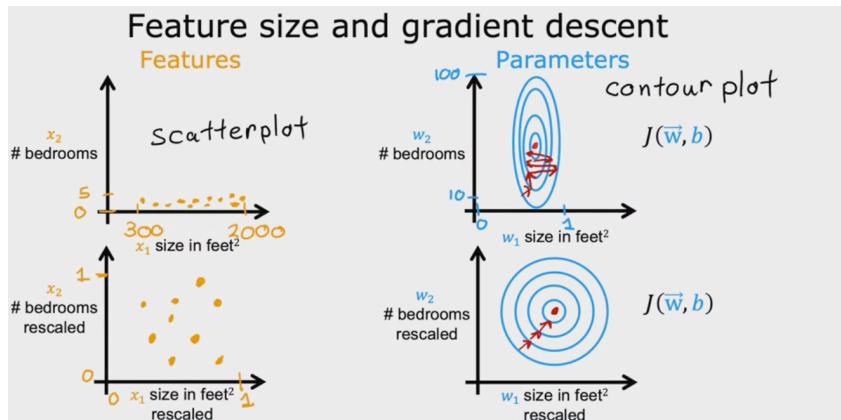


3.3 梯度下降法实践-特征缩放 Gradient Descent in Practice I - Feature Scaling

Feature scaling is useful in situations where the size of feature varies greatly such that a small change in the large size of feature, e.g. x_1 , would result in a significant change in the estimation output \hat{y} , hence comes the feature scaling for features rescaling such that the larger feature is rescaled smaller while smaller feature is rescaled larger, avoiding a diverged gradient descent result.

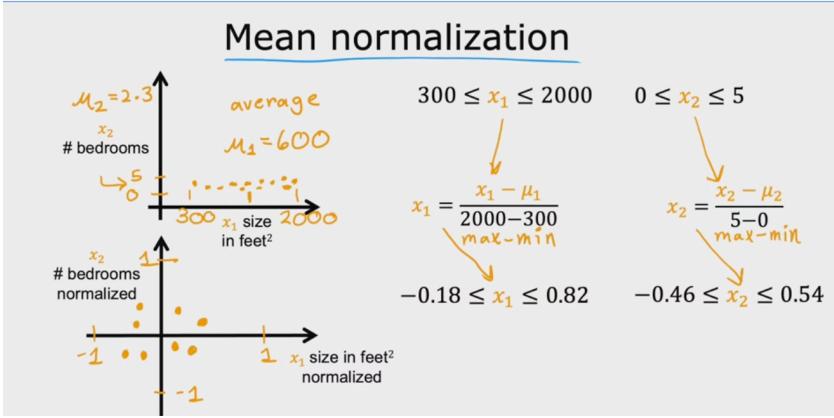


在我们面对多维特征问题的时候，我们要保证这些特征都具有相近的尺度，这将帮助梯度下降算法更快地收敛。以两个参数分别为横纵坐标，绘制代价函数的等高线图能，看出图像会显得很扁，梯度下降算法需要非常多次的迭代才能收敛。

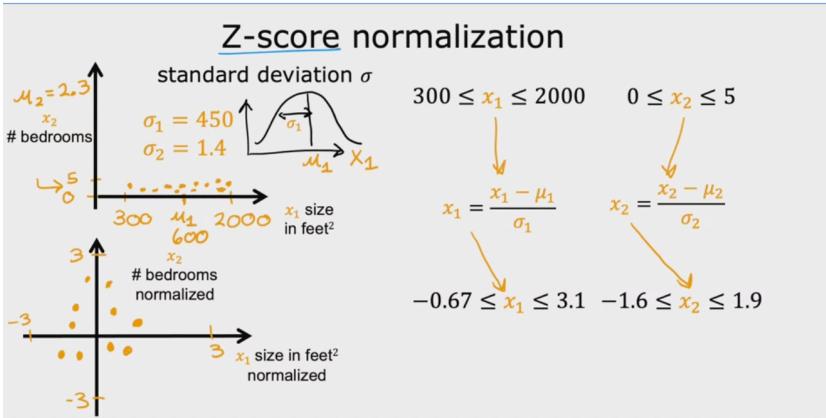


How to scale? Normalized

Mean normalization: $x_n = \frac{x_n - \mu_n}{x_{max} - x_{min}}$, 其中 μ_n 是平均值



Zscore normalization: $x_n = \frac{x_n - \mu_n}{\sigma_n}$, 其中 μ_n 是平均值, σ_n 是标准差。



z-score normalization

After z-score normalization, all features will have a mean of 0 and a standard deviation of 1.

To implement z-score normalization, adjust your input values as shown in this formula:

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j} \quad (4)$$

where j selects a feature or a column in the \mathbf{X} matrix. μ_j is the mean of all the values for feature (j) and σ_j is the standard deviation of feature (j) .

$$\mu_j = \frac{1}{m} \sum_{i=0}^{m-1} x_j^{(i)} \quad (5)$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=0}^{m-1} (x_j^{(i)} - \mu_j)^2 \quad (6)$$

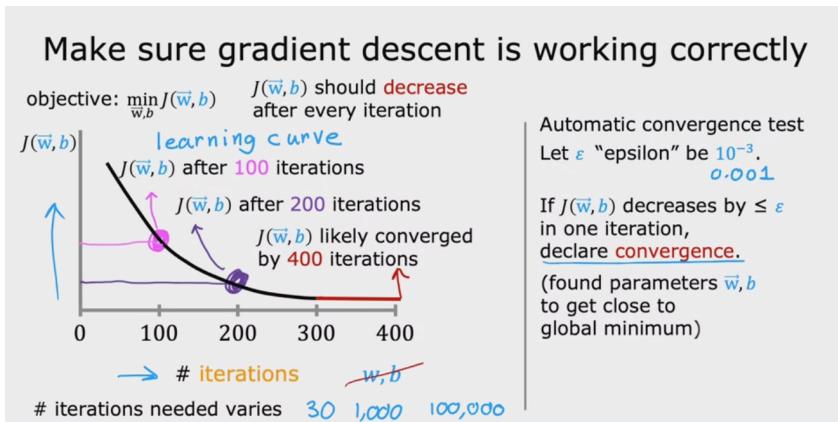
Rule of thumb of feature scaling:

Feature scaling	
aim for about $-1 \leq x_j \leq 1$ for each feature x_j	
$-3 \leq x_j \leq 3$	acceptable ranges
$-0.3 \leq x_j \leq 0.3$	
$0 \leq x_1 \leq 3$	okay, no rescaling
$-2 \leq x_2 \leq 0.5$	okay, no rescaling
$-100 \leq x_3 \leq 100$	too large → rescale
$-0.001 \leq x_4 \leq 0.001$	too small → rescale
$98.6 \leq x_5 \leq 105$	too large → rescale

3.4 梯度下降法实践-学习率 Gradient Descent in Practice II - Learning Rate

Checking gradient descent for convergence: by checking the gradient descent learning curve

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。



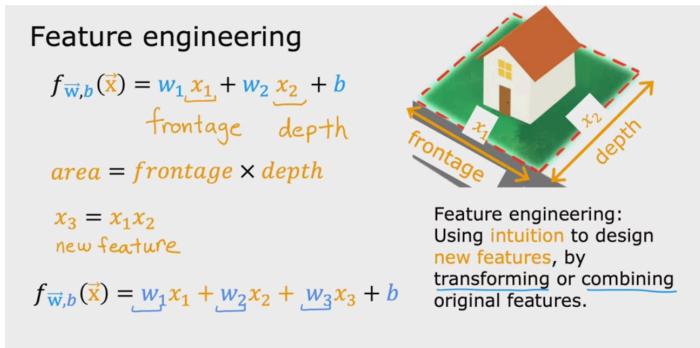
Automatic convergence test (ϵ , epsilon): 也有一些自动测试是否收敛的方法，例如将代价函数的变化值与某个阈值（例如 0.001）进行比较，if $J(\vec{w}, b)$ decreases by $\leq \epsilon$ in one iteration, convergence can be declared, 但通常看上面这样的图表更好。

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。

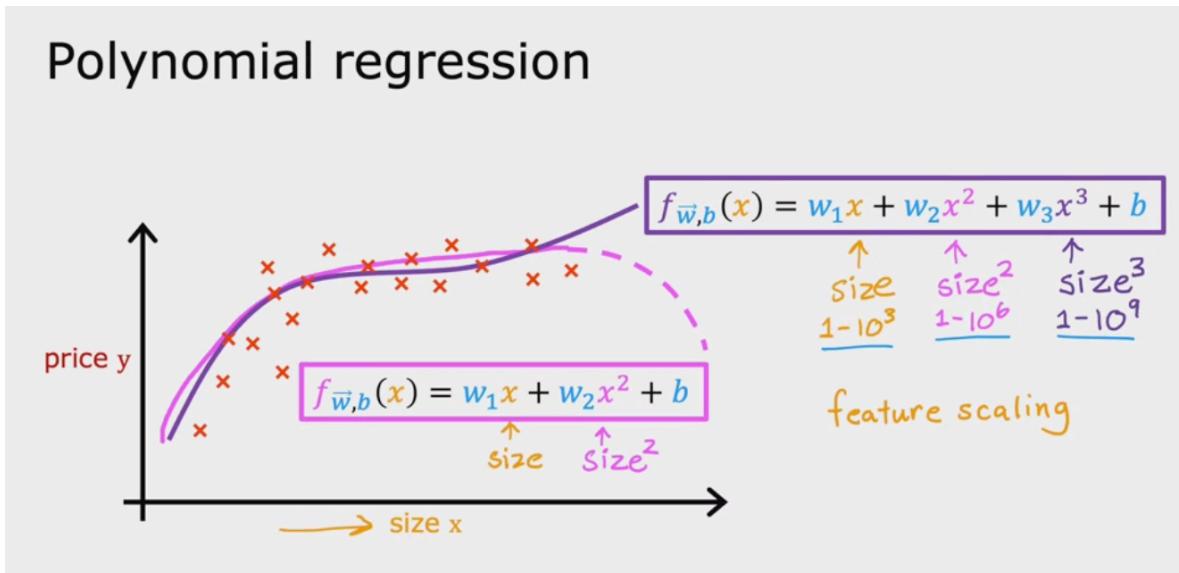
Choosing the learning rate, 通常可以考虑尝试些学习率：

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

3.5 特征和多项式回归 Feature Engineering: Feature and Polynomial Regression



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



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：

$x_2 = x_1^2, x_3 = x_1^3$ ，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：

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

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。

第3周

4、逻辑回归 Logistic Regression

4.1 分类问题 Classification

在这个以及接下来的几个视频中，开始介绍分类 **classification** 问题。

在分类问题中，你要预测的变量 y 是离散的值，我们将学习一种叫做逻辑回归 (**Logistic Regression**) 的算法，这是目前最流行使用最广泛的一种学习算法。

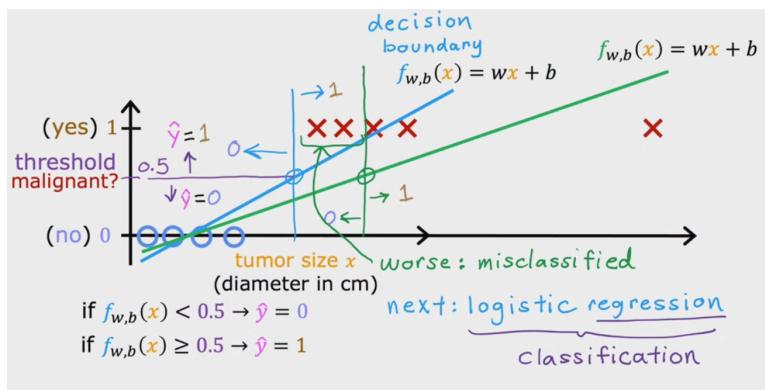
在分类问题中，我们尝试预测的是结果是否属于某一个类（例如正确或错误）。分类问题的例子有：判断一封电子邮件是否是垃圾邮件；判断一次金融交易是否(**binary classification: y can only be two values**)是欺诈；之前我们也谈到了肿瘤分类问题的例子，区别一个肿瘤是恶性的还是良性的。

Classification

- Email: Spam / Not Spam?
- Online Transactions: Fraudulent (Yes / No)?
- Tumor: Malignant / Benign ?

我们从二元的分类问题开始讨论。

我们将因变量(**dependent 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。



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。

如果我们要用线性回归算法来解决一个分类问题，对于分类， y 取值为 0 或者 1，但如果你使用的是线性回归，那么假设函数的输出值可能远大于 1，或者远小于 0，即使所有训练样本的标签 y 都等于 0 或 1。尽管我们知道标签应该取值 0 或者 1，但是如果算法得到的值远大于 1 或者远小于 0 的话，就会感觉很奇怪。所以我们在接下来的要研究的算法就叫做逻辑回归算法，这个算法的性质是：它的输出值永远在 0 到 1 之间。

顺便说一下，逻辑回归算法是分类算法，我们将它作为分类算法使用。有时候可能因为这个算法的名字中出现了“回归”使你感到困惑，但逻辑回归算法实际上是一种分类算法，它适用于标签 y 取值离散的情况，如：1 0 0 1。

在接下来的视频中，我们将开始学习逻辑回归算法的细节。

4.2 假说表示 Sigmoid Function (Hypothesis Representation)

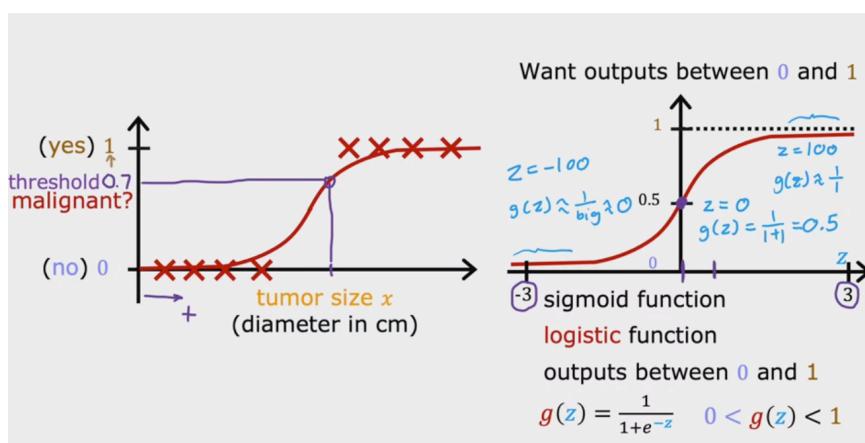
在这段视频中，我要给你展示假设函数的表达式，也就是说，在分类问题中，要用什么样的函数来表示我们的假设。此前我们说过，希望我们的分类器的输出值在 0 和 1 之间，因此，我们希望想出一个满足某个性质的假设函数，这个性质是它的预测值要在 0 和 1 之间。

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 $h_\theta(x) \geq 0.5$ 时，预测 $y = 1$ 。

当 $h_\theta(x) < 0.5$ 时，预测 $y = 0$ 。

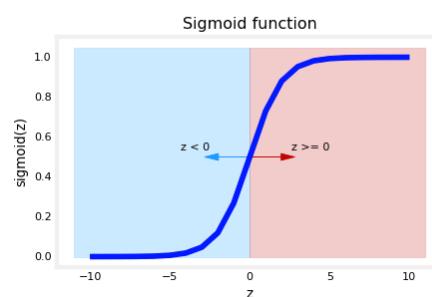
我们引入一个新的模型，逻辑回归，该模型的输出变量范围始终在 0 和 1 之间。逻辑回归模型的假设是： $h_\theta(x) = g(\theta^T x)$ 其中： X 代表特征向量 g 代表逻辑函数（**logistic function**）是一个常用的逻辑函数为 S 形函数（**Sigmoid function**），公式为： $g(z) = \frac{1}{1+e^{-z}}$

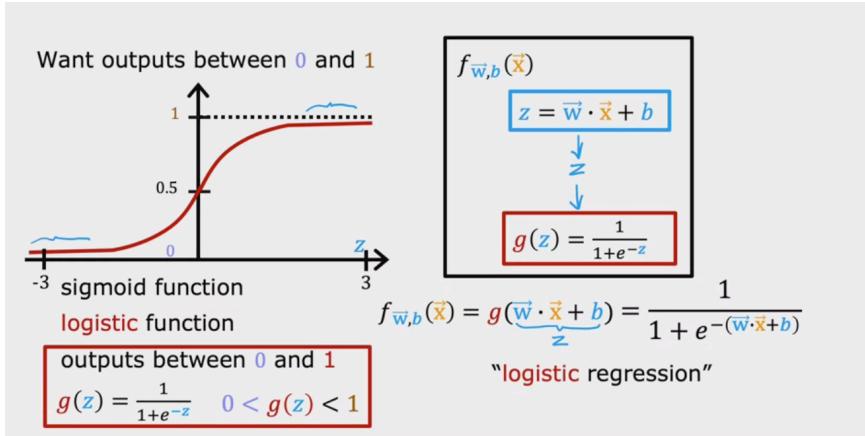


python 代码实现：

```
import numpy as np
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
```

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： $g(z) = \frac{1}{1+e^{-z}}$ 。

Interpretation of logistic regression output

$h_\theta(x)$ 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 $1-0.7=0.3$ 。

Interpretation of logistic regression output

$$f_{\vec{w},b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$$

"probability" that class is 1

$$f_{\vec{w},b}(\vec{x}) = P(y = 1|\vec{x}; \vec{w}, b)$$

Probability that y is 1, given input \vec{x} , parameters \vec{w}, b

Example:

x is "tumor size"

y is 0 (not malignant)
or 1 (malignant)

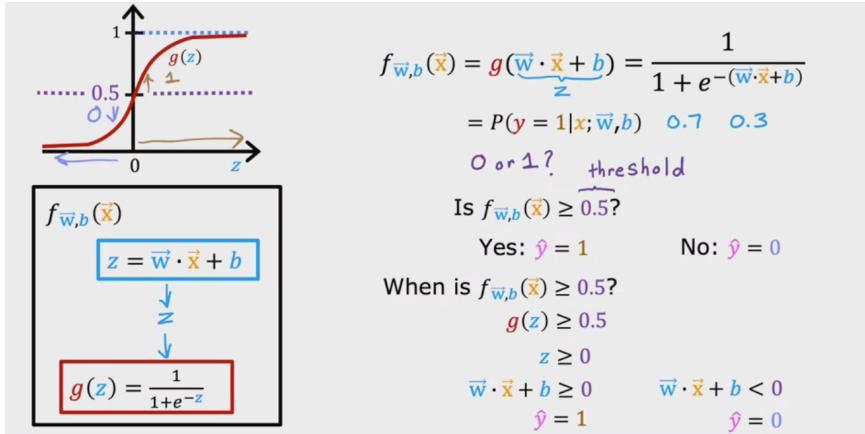
$$P(y = 0) + P(y = 1) = 1$$

$$f_{\vec{w},b}(\vec{x}) = 0.7$$

70% chance that y is 1

4.3 判定边界 Decision Boundary

现在讲下决策边界(**decision boundary**)的概念。这个概念能更好地帮助我们理解逻辑回归的假设函数在计算什么。



在逻辑回归中，我们预测：

当 $f_{w,b}(x) \geq 0.5$ 时，预测 $y = 1$ ；当 $f_{w,b}(x) < 0.5$ 时，预测 $y = 0$ 。

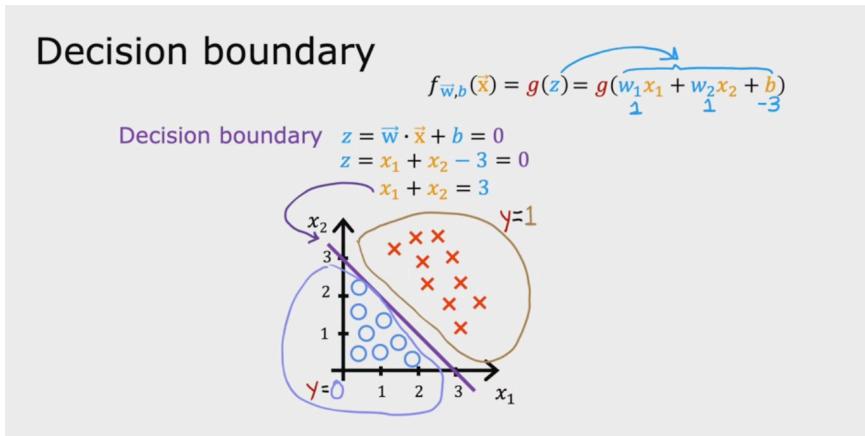
根据上面绘制出的 S 形函数图像，我们知道当

$z = 0$ 时 $g(z) = 0.5$ ； $z > 0$ 时 $g(z) > 0.5$ ； $z < 0$ 时 $g(z) < 0.5$

又 $z = \vec{w} \cdot \vec{x} + b$ ，即：

$\vec{w} \cdot \vec{x} + b \geq 0$ 时，预测 $y = 1$ ； $\vec{w} \cdot \vec{x} + b \leq 0$ 时，预测 $y = 0$

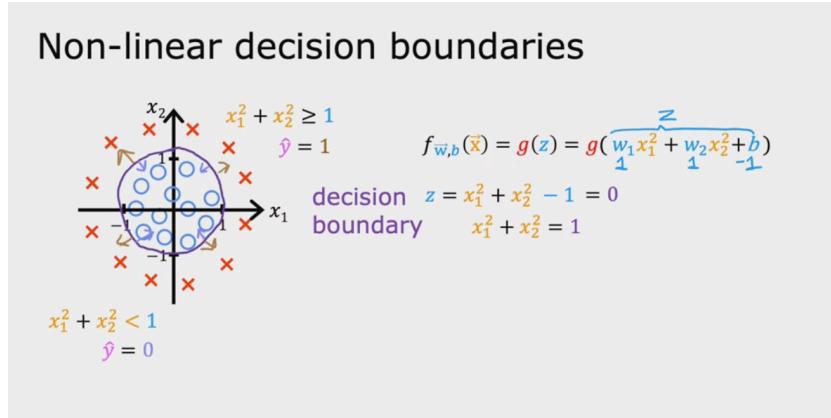
现在假设我们有一个模型：



并且参数 θ 是向量 $[-3 \ 1 \ 1]$ 。则当 $-3 + x_1 + x_2 \geq 0$ ，即 $x_1 + x_2 \geq 3$ 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。

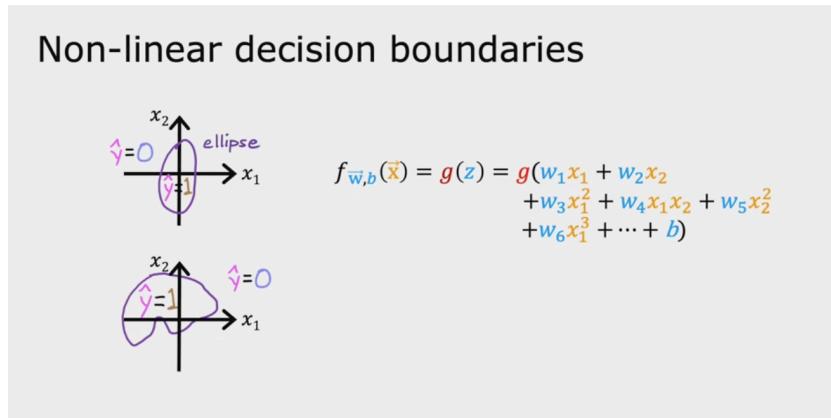
Non-linear decision boundaries

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： $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$ 是 $[-1 \ 0 \ 0 \ 1 \ 1]$ ，则我们得到的判定边界恰好是圆心在原点且半径为 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。

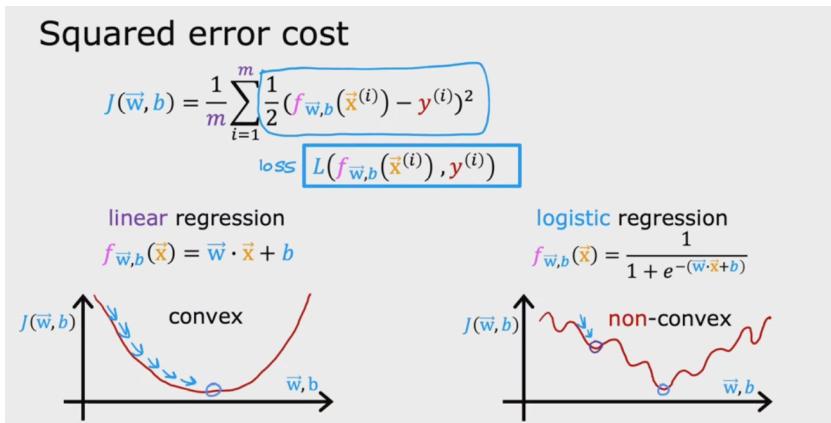


4.4 代价函数 Cost Function

在这段视频中，我们要介绍如何拟合逻辑回归模型的参数 θ 。具体来说，我要定义用来拟合参数的优化目标或者叫代价函数，这便是监督学习问题中的逻辑回归模型的拟合问题。

How to choose the parameters w and b ? Logistic loss function

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，当我们把 $f_{w,b}(x) = \frac{1}{1+e^{-\theta^T x}}$ 带入到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 lots of local minima that might have the cost function stuck，这将影响梯度下降算法寻找全局最小值 pose an issue to finding the optimal global minima。

What should be the more suitable prediction algorithm for situation when y can only take on values of 1 or 0 (Classification)?

线性回归的代价函数为: $J(w, b) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (f_{w,b}(x^{(i)}) - y^{(i)})^2$ 。

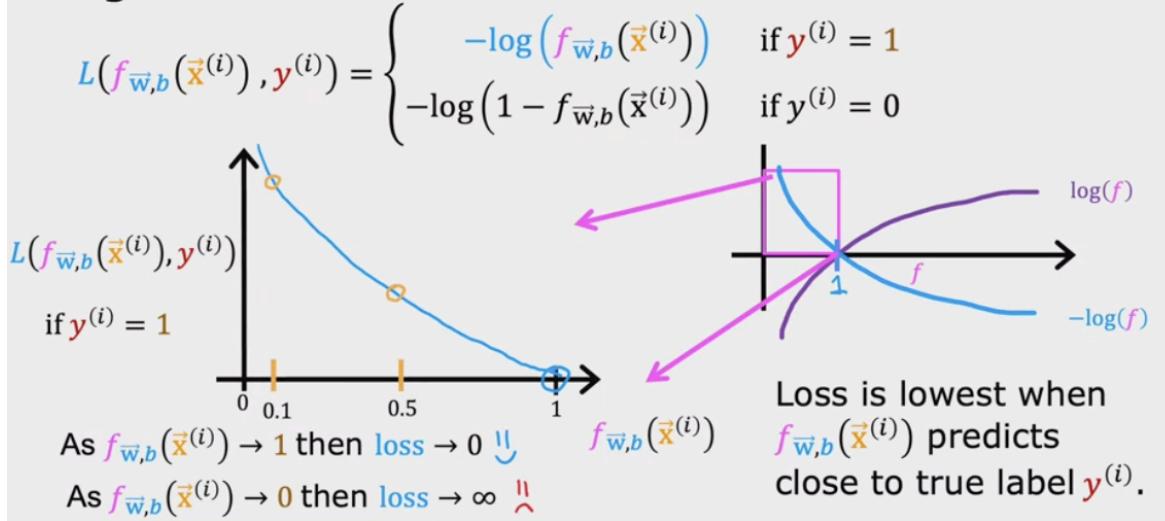
我们重新定义逻辑回归的代价函数为: $J(w, b) = \frac{1}{m} \sum_{i=1}^m L(f_{w,b}(x^{(i)}), y^{(i)})$ ，其中

Logistic loss function

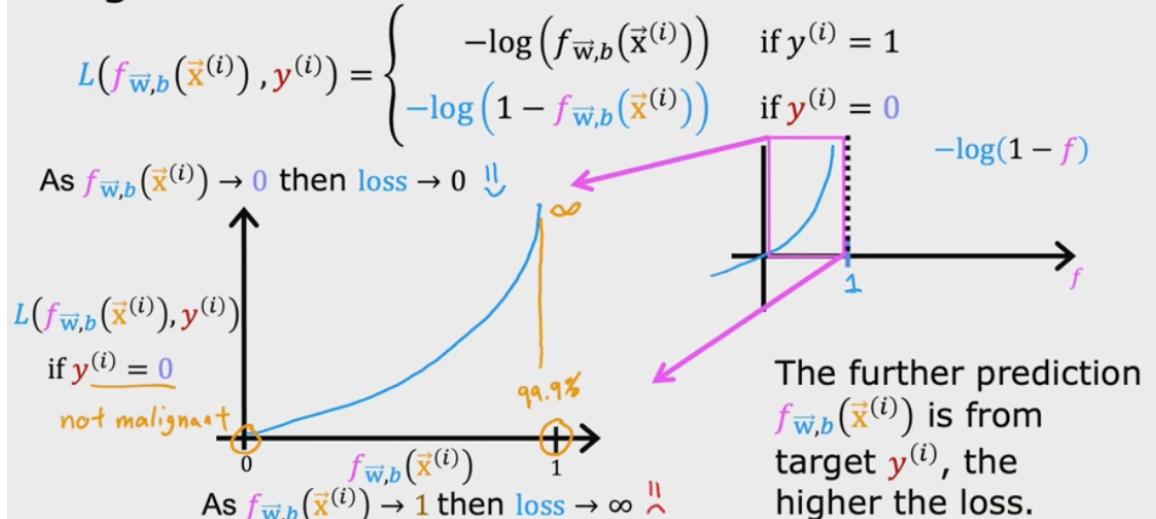
$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{if } y^{(i)} = 0 \end{cases}$$

$f_{w,b}(x)$ 与 $L(f_{w,b}(x), y)$ 之间的关系如下图所示:

Logistic loss function



Logistic loss function



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

将构建的 $L(f_{w, b}(x), y)$ 简化如下：

$$L(f_{w, b}(x), y) = -y \cdot \log(f_{w, b}(x)) - (1 - y) \cdot \log(1 - f_{w, b}(x))$$

带入代价函数得到：

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m [-y^{(i)} \log(f_{w, b}(x^{(i)})) - (1 - y^{(i)}) \log(1 - f_{w, b}(x^{(i)}))]$$

$$\text{即: } J(w, b) = \frac{1}{m} \sum_{i=1}^m [-y^{(i)} \log(f_{w,b}(x^{(i)})) - (1 - y^{(i)}) \log(1 - f_{w,b}(x^{(i)}))]$$

Simplified cost function

$$\begin{aligned}
 & \underset{\text{loss}}{L}(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) \\
 & \underset{\text{cost}}{J}(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m [L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})] \\
 & = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] \\
 & \quad \text{maximum likelihood} \\
 & \quad \text{(don't worry about it!)}
 \end{aligned}$$

Gradeint Descent Implementation: Training logistic regression

4.5 简化成本函数的梯度下降 Gradient Descent Implementation for logistic regressoin

根据这个代价函数, 为了拟合出参数, 该怎么做呢? 我们要试图找尽量让 $J(w, b)$ 取得最小值的参数 $f_{w,b}$ 。

$$\min J(w, b)$$

Find \vec{w}, b

$$\text{Given new } \vec{x}, \text{ output } f_{\vec{w}, b}(\vec{x}) = \frac{1}{1+e^{-(\vec{w} \cdot \vec{x} + b)}}$$

$$P(y = 1 | \vec{x}; \vec{w}, b)$$

所以我们想要尽量减小这一项, 这将我们将得到某个参数 (\vec{w}, b) 。

如果我们给出一个新的样本, 假如某个特征 x , 我们可以用拟合训练样本的参数 (\vec{w}, b) , 来输出对假设的预测。

另外, 我们假设的输出, 实际上就是这个概率值: $p(y = 1 | \vec{x}; (\vec{w}, b))$, 就是关于 x 以 (\vec{w}, b) 为参数, $y = 1$ 的概率, 你可以认为我们的假设就是估计 $y = 1$ 的概率, 所以, 接下来就是弄清楚如何最大限度地最小化代价函数 $J(\vec{w}, b)$, 作为一个关于 θ 的函数, 这样我们才能为训练集拟合出参数 (\vec{w}, b) 。

最小化代价函数的方法, 是使用梯度下降法(gradient descent)。这是我们的代价函数:

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(x^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(x^{(i)}))]$$

如果我们要最小化这个关于(\vec{w}, b)的函数值，这就是我们通常用的梯度下降法的模板。

n is the number of features

α is the learning rate

Gradient descent

cost

$$J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))]$$

repeat {

$j = 1 \dots n$

$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$

$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$

} simultaneous updates

$\frac{\partial}{\partial w_j} J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}$

$\frac{\partial}{\partial b} J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$

我们要反复更新每个参数，用这个式子来更新，就是用它自己减去学习率 α 乘以后面的微分项。求导后得到：

如果你计算一下的话，你会得到这个等式：

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$b := b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

所以，如果你有 n 个特征，也就是说：参数向量 θ 包括 $\theta_0 \ \theta_1 \ \theta_2 \ \dots \ \theta_n$ ，那么你就需

要用这个式子：

$$w_j = \theta_j$$

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

现在，如果你把这个更新规则和我们之前用在线性回归上的进行比较的话，你会惊讶地发现，这个式子正是我们用来做线性回归梯度下降的。

那么，线性回归和逻辑回归是同一个算法吗？要回答这个问题，我们要观察逻辑回归看看发生了哪些变化。实际上，假设的定义发生了变化。

对于线性回归假设函数 **For Linear regression:**

$$f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$$

而现在逻辑函数假设函数 **For Logistic regression:**

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

因此，即使更新参数的规则看起来基本相同，但由于假设的定义发生了变化，所以逻辑函数的梯度下降，跟线性回归的梯度下降实际上是两个完全不同的东西。

在先前的视频中，当我们在谈论线性回归的梯度下降法时，我们谈到了如何监控梯度下降法以确保其收敛，我通常也把同样的方法用在逻辑回归中，来监测梯度下降，以确保它正常收敛。

Gradient descent for logistic regression

repeat {

looks like linear regression!

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

} simultaneous updates

Same concepts:

- Monitor gradient descent (learning curve)
- Vectorized implementation
- Feature scaling

Linear regression $f_{\vec{w}, b}(\vec{x}) = \vec{w} \cdot \vec{x} + b$

Logistic regression $f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$

当使用梯度下降法来实现逻辑回归时，我们有这些不同的参数 θ ，就是 $\theta_0 \ \theta_1 \ \theta_2$ 一直到 θ_n ，我们需要用这个表达式来更新这些参数。我们还可以使用 **for 循环** 来更新这些参数值，用 **for i=1 to n**，或者 **for i=1 to n+1**。当然，不用 **for 循环** 也是可以的，理想情况下，我们更提倡使用向量化的实现，可以把所有这些 n 个参数同时更新。

最后还有一点，我们之前在谈线性回归时讲到的特征缩放，我们看到了特征缩放是如何提高梯度下降的收敛速度的，这个特征缩放的方法，也适用于逻辑回归。如果你的特征范围差距很大的话，那么应用特征缩放的方法，同样也可以让逻辑回归中，梯度下降收敛更快。

就是这样，现在你知道如何实现逻辑回归，这是一种非常强大，甚至可能世界上使用最广泛的一种分类算法。

5、正则化 Regularization

5.1 过拟合的问题 The Problem of Overfitting

Regularization is the commonly used technique helps minimize the problem of overfitting.

到现在为止，我们已经学习了几种不同的学习算法，包括线性回归和逻辑回归，它们能够有效地解决许多问题，但是当将它们应用到某些特定的机器学习应用时，会遇到过拟合(over-fitting)的问题，可能会导致它们效果很差。

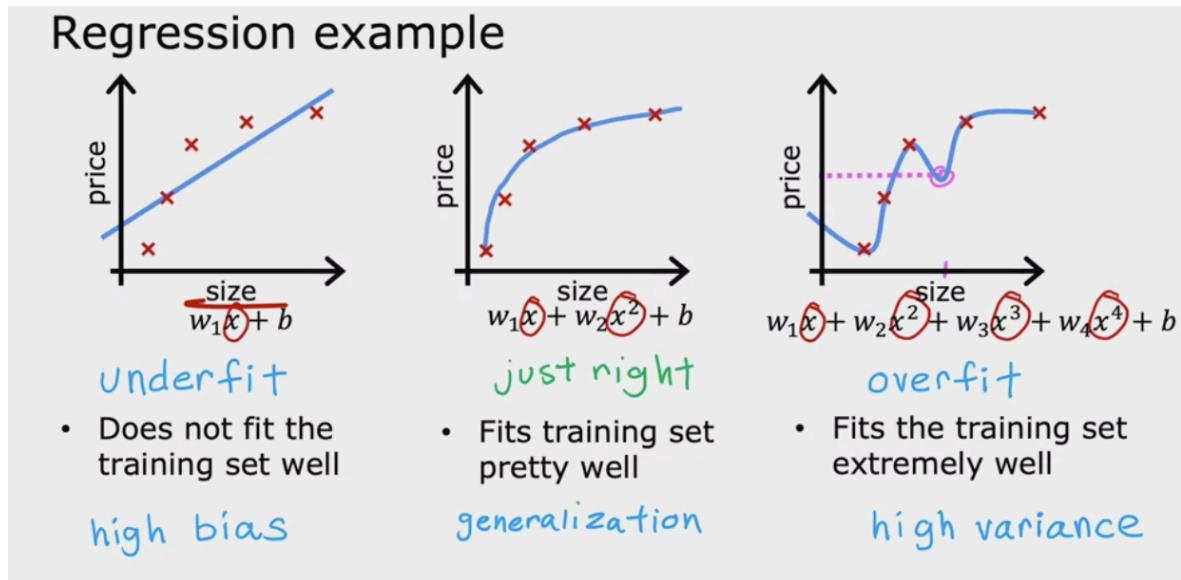
在这段视频中，我将为你解释什么是过度拟合问题，并且在此之后接下来的几个视频中，我们将谈论一种称为正则化(regularization)的技术，它可以改善或者减少过度拟合问题。

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。

Types of regression:

- Underfitting/ high bias (algorithm unable to capture and hence is highly biased)
- Just right/ generalization: fits training pretty well.
- Overfitting/ high variance fits the training set extremely well

下图是一个回归问题的例子：



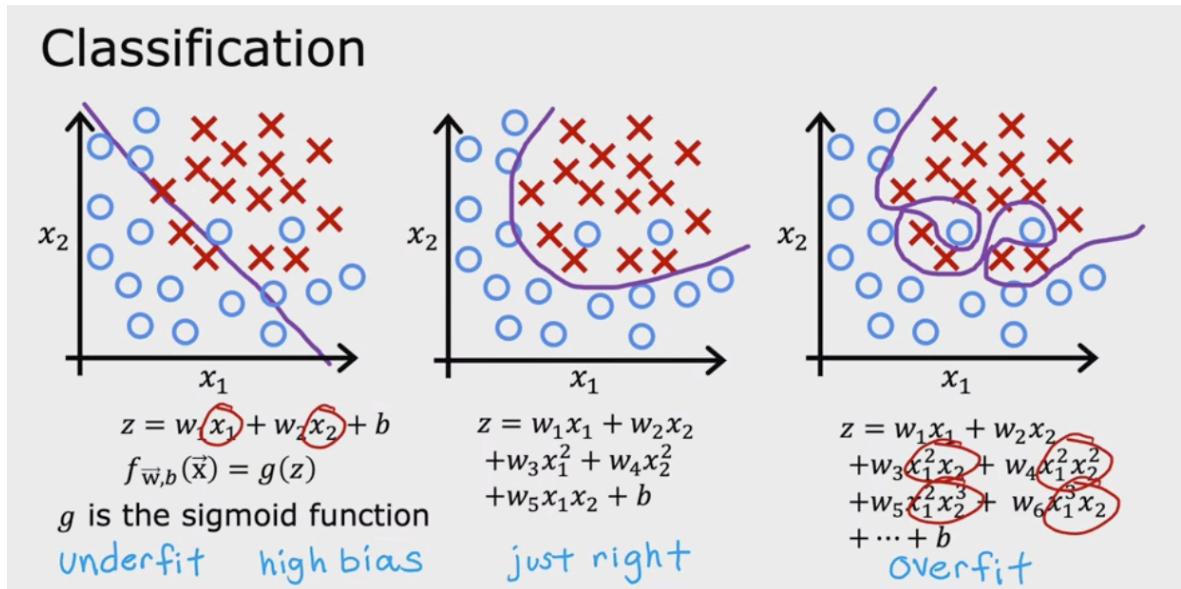
第一个模型是一个线性模型，欠拟合，不能很好地适应我们的训练集；第三个模型是一个四次方的模型，过于强调拟合原始数据，而丢失了算法的本质：预测新数据。我们可以看出，若给出一个新的值使之预测，它将表现的很差，是过拟合，虽然能非常好地适应我们的训练集但在新输入变量进行预测时可能会效果

不好；而中间的模型似乎最合适。

Our goal when creating a model is to be able to use the model to predict outcomes correctly for new examples. A model which does this is said to **generalize well**.

The same problems apply to Classification

分类问题中也存在这样的问题：



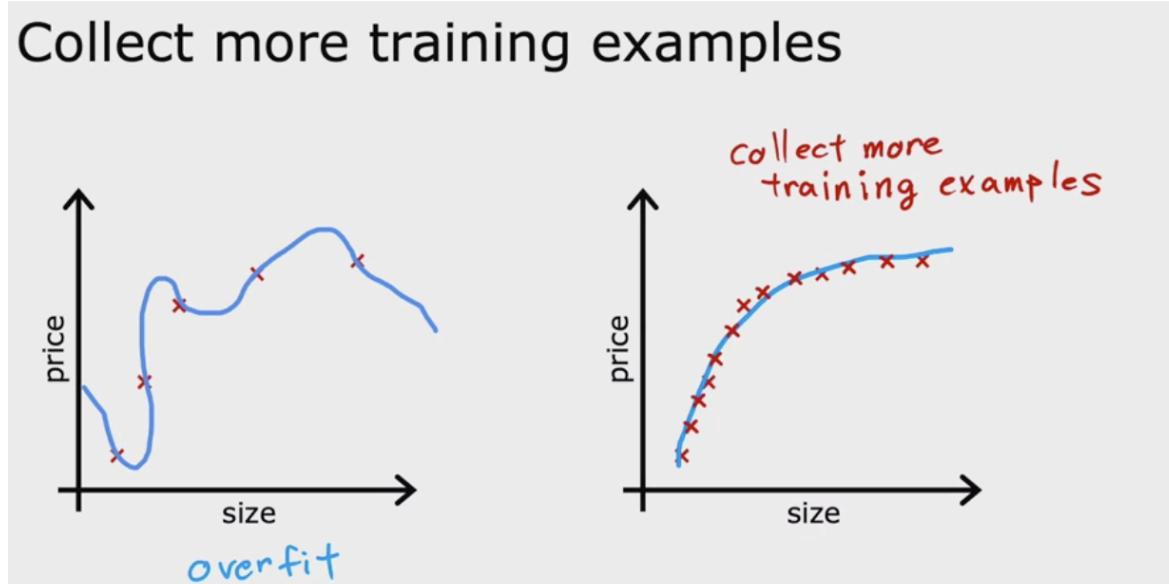
就以多项式理解， x 的次数越高，拟合的越好，但相应的预测的能力就可能变差。

问题是，如果我们发现了过拟合问题，应该如何处理？

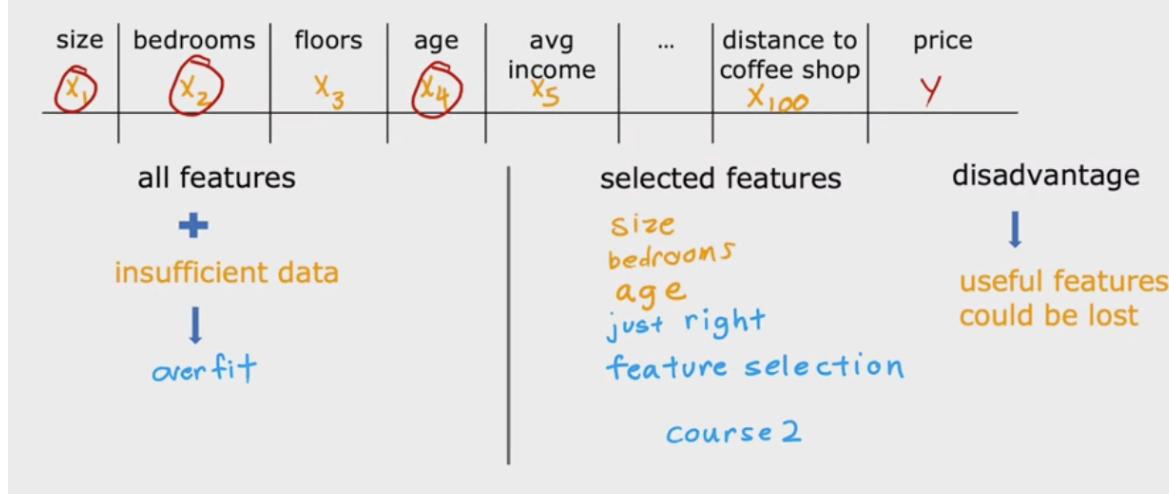
5.2 Address Overfitting

Address overfitting

- a) Collect more training data.
- b) Select features to include/exclude 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）
- c) 3. Regularization to reduce Overfitting 正则化。保留所有的特征，但是减少参数的大小 (magnitude)。



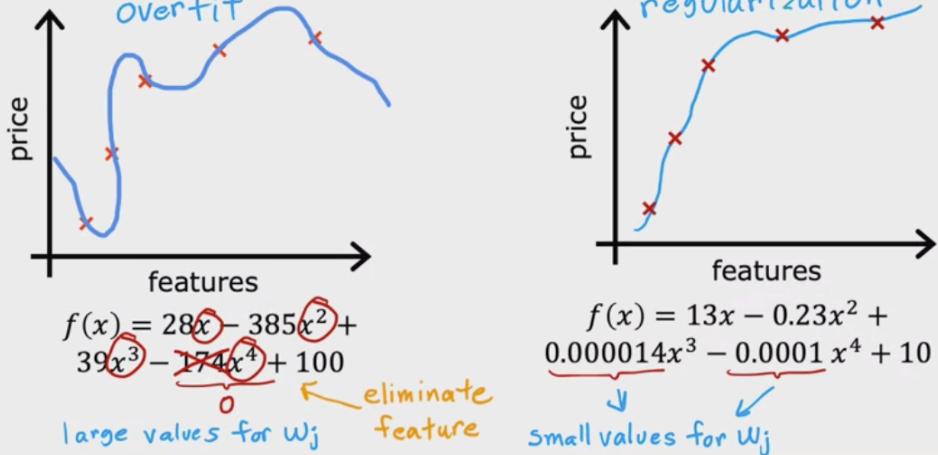
Select features to include/exclude



我们可以从之前的事例中看出，正是那些高次 **higher power polynomial terms** 项导致了过拟合 **overfitting** 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。

Regularization

Reduce the size of parameters w_j



Addressing overfitting

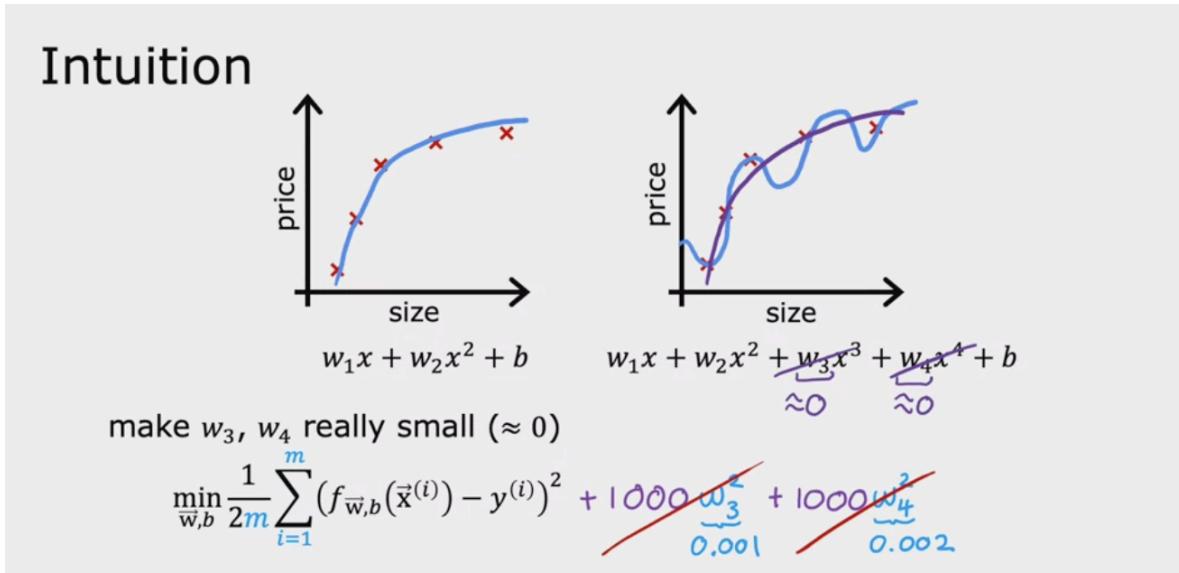
Options

1. Collect more data
2. Select features
 - Feature selection *in course 2*
3. Reduce size of parameters
 - "Regularization" *next videos!*

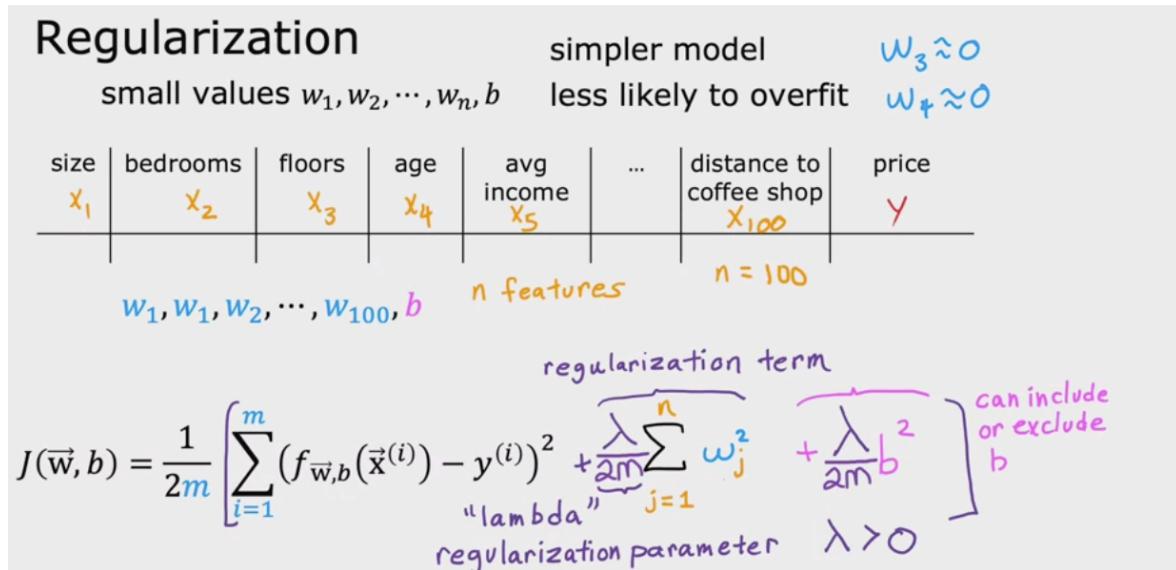
5.3 正規化的代價函數 Cost Function with Regularization

Instead of modifying the cost function for linear regression, we add terms of w_3 , and w_4 , where when minimizing the cost function, w_3 and w_4 are forced to be small and penalized, hence achieving the task of cost minimization.

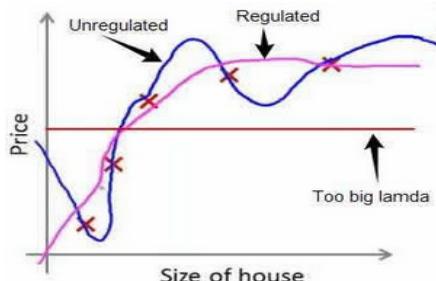
所以我们要做的就是在一定程度上减小这些参数 w 的值，这就是正则化的基本方法。我们决定要减少 w_3 和 w_4 的大小，我们要做的便是修改代价函数，在其中 w_3 和 w_4 设置一点惩罚。这样做的话，我们在尝试最小化代价时也需要将这个惩罚纳入考虑中，并最终导致选择较小一些的 w_3 和 w_4 。通过这样的代价函数选择出的 w_3 和 w_4 对预测结果的影响就比之前要小许多。



When all n features are to be used, we can add a term of λ , a regularization parameter used to penalize all of the features through weights. $\lambda > 0$. Just to note, we also devide λ by $2m$ as extra scaling if m , or number of features, is large. 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： $J(\vec{w}, b) = \frac{1}{2m} [\sum_{i=1}^m (f_{\vec{w}, b}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n w_j^2]$

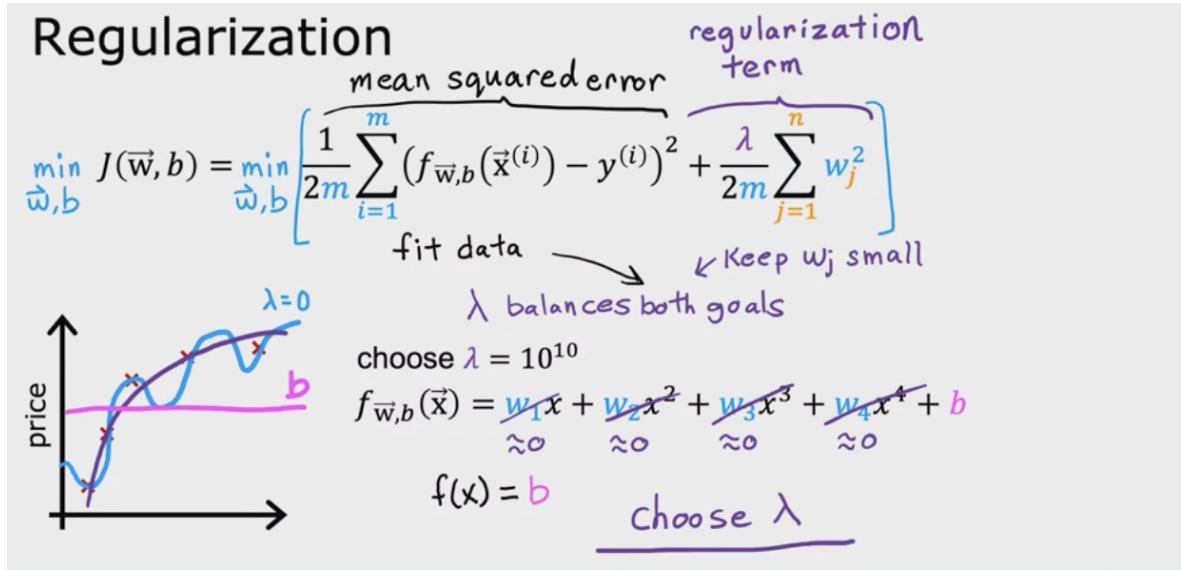


其中 λ 又称为正则化参数 (Regularization Parameter)。注：根据惯例，我们不对 b 进行惩罚。经过正则化处理的模型与原模型的可能对比如下图所示：



If λ is too small, eg. $\lambda = 0$, the model will overfit

If λ is too large, eg. $\lambda = \infty$, the model will underfit



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 $f_{\vec{w}, b}(x) = b$ ，也就是上图中红色直线所示的情况，造成欠拟合。

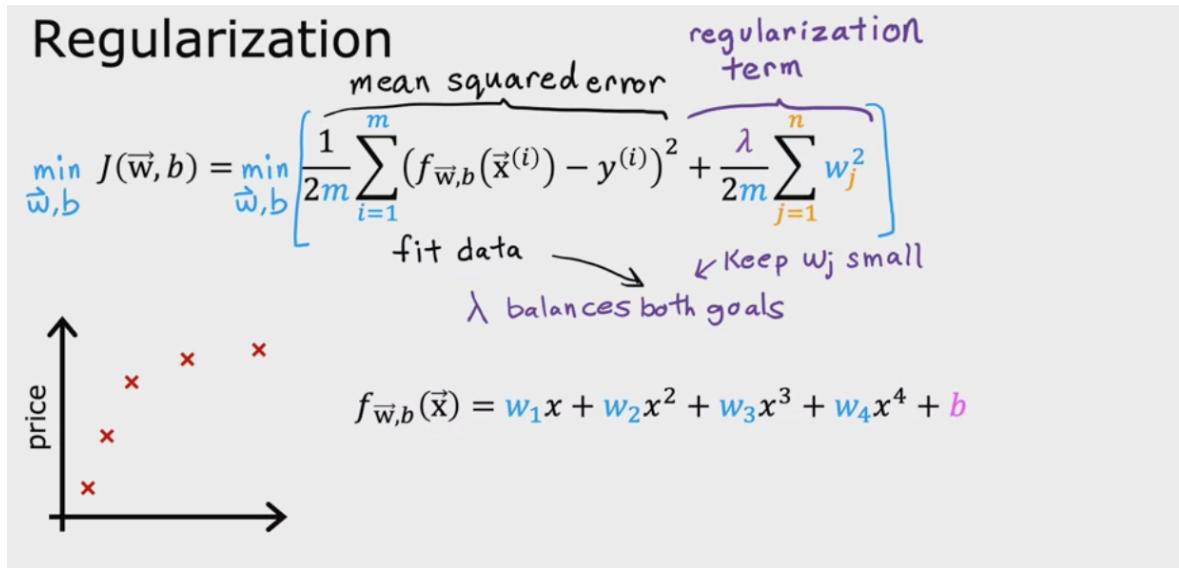
那为什么增加的一项 $\lambda = \sum_{j=1}^n w_j^2$ 可以使 w 的值减小呢？

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 w 的值都会在一定程度上减小。

但若 λ 的值太大了，那么 w 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。

回顾一下代价函数，为了使用正则化，让我们把这些概念应用到线性回归和逻辑回归中去，那么我们就可以让他们避免过度拟合了。



5.4 正则化线性回归 Regularized Linear Regression

Regularized linear regression

$$\min_{\vec{w}, b} J(\vec{w}, b) = \min_{\vec{w}, b} \left[\frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right]$$

Gradient descent

repeat {

$$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$j = 1, \dots, n$

$$b = b - \alpha \frac{\partial}{\partial b} J(\vec{w}, b)$$

} simultaneous update

$$= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j$$

$$= \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

don't have to regularize b

Implementing gradient descent

repeat {

$$w_j = w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m [(f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}] + \frac{\lambda}{m} w_j \right]$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update $j = 1, \dots, n$

$$w_j = \underbrace{w_j - \alpha \frac{\lambda}{m} w_j}_{w_j \left(1 - \alpha \frac{\lambda}{m} \right)} - \underbrace{\alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}}_{\text{usual update}}$$

$$\alpha \frac{\lambda}{m}$$

$0.01 \frac{1}{50} = 0.0002$

$$w_j \left(1 - 0.0002 \right)$$

0.9998

对于线性回归的求解，我们之前推导了两种学习算法：一种基于梯度下降，一种基于正规方程。

正则化线性回归的代价函数为：

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m [(f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n w_j^2]$$

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

Repeat until convergence{

$$\begin{aligned} b &:= b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(x^{(i)}) - y^{(i)}) \\ w_j &:= w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j \right] \\ &\quad \} \end{aligned}$$

Repeat

对上面的算法中 $j = 1, 2, \dots, n$ 时的更新式子进行调整可得：

$$w_j := w_j (1 - \alpha \frac{\lambda}{m}) - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

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 w 值减少了一个额外的值。

How we get the derivative term (optional)

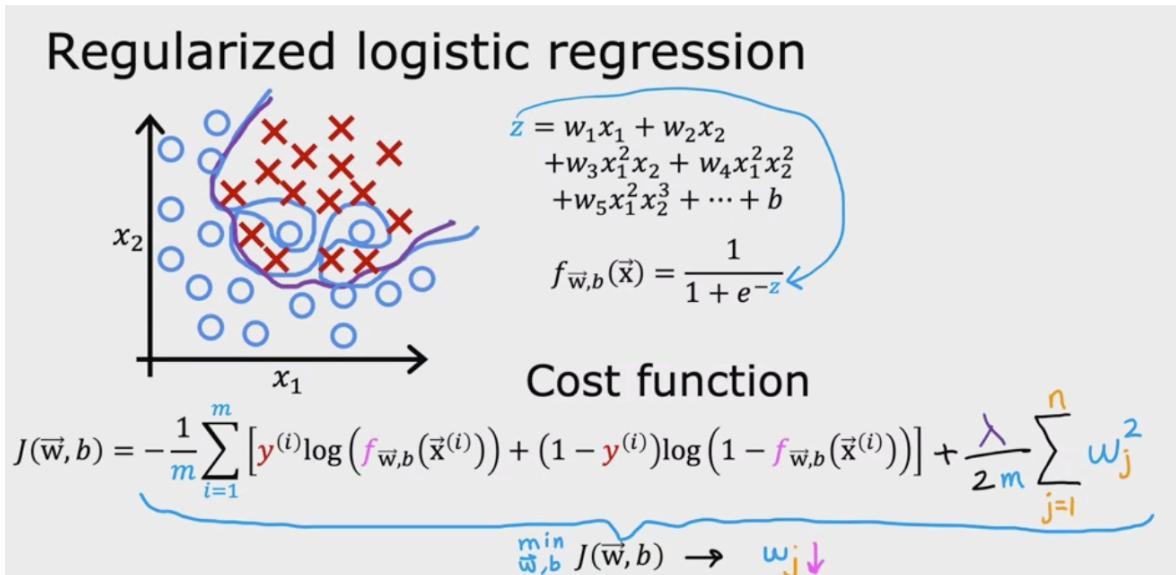
$$\begin{aligned} \frac{\partial}{\partial w_j} J(\vec{w}, b) &= \frac{\partial}{\partial w_j} \left[\frac{1}{2m} \sum_{i=1}^m \underbrace{(f(\vec{x}^{(i)}) - y^{(i)})^2}_{\vec{w} \cdot \vec{x}^{(i)} + b} + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2 \right] \\ &= \frac{1}{2m} \sum_{i=1}^m \left[(\vec{w} \cdot \vec{x}^{(i)} + b - y^{(i)}) \cancel{x_j^{(i)}} \right] + \frac{\lambda}{2m} \cancel{2w_j} \stackrel{\text{No}}{\sum_{j=1}^n} w_j \\ &= \frac{1}{m} \sum_{i=1}^m \left[\underbrace{(\vec{w} \cdot \vec{x}^{(i)} + b - y^{(i)})}_{f(\vec{x}^{(i)})} x_j^{(i)} \right] + \frac{\lambda}{m} w_j \\ &= \frac{1}{m} \sum_{i=1}^m \left[(f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} w_j \end{aligned}$$

5.5 正则化的逻辑回归模型 Regularized Logistic Regression

针对逻辑回归问题，我们在之前的课程已经学习过两种优化算法：我们首先学习了使用梯度下降法来优化代价函数 $J(\vec{w}, b)$ ，接下来学习了更高级的优化算法，这些高级优化算法需要你自己设计代价函数 $J(\vec{w}, b)$ 。

自己计算导数同样对于逻辑回归，我们也给代价函数增加一个正则化的表达式，得到代价函数：

$$J(w, b) = \frac{1}{m} \sum_{i=1}^m [-y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) - (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$



Regularized logistic regression

$$\min_{\vec{w}, b} J(\vec{w}, b) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + (1 - y^{(i)}) \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

Gradient descent

$$\text{repeat } \{$$

$$\begin{aligned} w_j &= w_j - \alpha \left(\frac{\partial}{\partial w_j} J(\vec{w}, b) \right) \\ b &= b - \alpha \left(\frac{\partial}{\partial b} J(\vec{w}, b) \right) \end{aligned}$$

= $\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)}$ + $\frac{\lambda}{m} w_j$

= $\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$ don't have to regularize

Looks same as for linear regression!

logistic regression

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{

$$\begin{aligned} \mathbf{b} &:= b - \alpha \frac{1}{m} \sum_{i=1}^m ((f_{\vec{w}, b}(x^{(i)}) - y^{(i)})) \\ w_j &:= w_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} w_j \right] \\ \text{for } j &= 1, 2, \dots, n \\ &\quad \} \end{aligned}$$

注：看上去同线性回归一样，但是知道 $f_{\vec{w}, b}(x) = g(\vec{w} \cdot X)$ ，所以与线性回归不同。

值得注意的是参数**b**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 $f_{\vec{w}, b}(x)$ 不同所以还是有很大差别。
2. **b**不参与其中的任何一个正则化。

目前大家对机器学习算法可能还只是略懂，但是一旦你精通了线性回归、高级优化算法和正则化技术，坦率地说，你对机器学习的理解可能已经比许多工程师深入了。现在，你已经有了丰富的机器学习知识，目测比那些硅谷工程师还厉害，或者用机器学习算法来做产品。

接下来的课程中，我们将学习一个非常强大的非线性分类器，无论是线性回归问题，还是逻辑回归问题，都可以构造多项式来解决。你将逐渐发现还有更强大的非线性分类器，可以用来解决多项式回归问题。我们接下来将学会，比现在解决问题的方法强大 N 倍的学习算法。