吴恩达新书 - 机器学习年鉴中文版

给机器学习从业者的 52 条建议

为什么要翻译这本书?

这本书是吴恩达新书的草稿。在书中，吴恩达总结了机器学习过程中，可能会发生的问题，并给出了解决问题的思路和建议。 这本书对于机器学习从业者来说，具有很高的参考价值。

全书包含了 9 章 52 节，每章都很短，方便大家阅读和传阅。书中的内容大部分都是跟实践相关的，数学知识很少，读起来相对简单，非常适合机器学习实践人员。

我翻译这本书的目的：一来我自己可以了解更多的机器学习知识，二来让英语相对差的同学能快速的浏览本书。

如果你对书中的内容有任何疑问或翻译有误，请加我微信:wxminapp，共同讨论。

更多资源参看 www.insideai.cn 站内其他机器学习视频教程：

1. 查看红城保卫战游戏(感知器算法)视频教程

2. 查看陨石坠落游戏(线性回归算法)视频教程

3. 查看动画版机器学习入门视频教程

目录

第一章：概述

1.为什么要说机器学习策略

机器学习是无数重要应用的基础，包括网页搜索、垃圾邮件判断、语音识别、商品推荐等等。假设你或你的团队，正在开发一款机器学习的应用，如果你想快速提升进度，那这本书将是一个不错的选择。

举个例子：建立一个猫类图片展示的创业公司，为广大的猫类爱好者，提供各种各样猫类图片。

你要使用神经网络构建一个自动检测猫位置的计算机视觉系统。 但不幸的是，你的算法的精度还不够好。 你正面临改进猫咪检测器的巨大压力，那你会怎么做呢？

你的团队可能会有很多想法，比如：

- 获取更多数据: 采集更多猫咪的图片
- 收集一个更加多样化的训练集. 比如，猫在其他位置或其他颜色猫的图片 ....
- 增加训练时长，让梯度下降法多执行更多次.
- 试试更加大型的神经网络，加深神经网络的层数、增加每层的神经元数量、使用更多 的参数值.

- 使用一个小一些的神经网络
- 试试正则化选项，比如 L2 正则化
- 改变神经网络的结构，比如激活函数、神经元数量等等
- ...

这些建议，如果你选择正确，你将会构建一个牛 X 的猫咪图片平台，并带领公司走向成功。但是如果你选择错误，则可能会浪费公司好几个月的时间。

那如何着手考虑这个问题呢?

这本书将给你指明一条道路。我们遇到的大多数机器学习问题都留下一些线索， 通过这些线索，我们可以知道哪些尝试会有用，哪些尝试会徒劳无功。 学会去看这些线索，会让你节省数月甚至数年的开发时间。

2.本书对你和你的团队有哪些帮助

读完这本书后，可以让你对机器学习项目中涉及的相关技术有更加深刻的理解。

有时你的队友可能并不理解你所建议的方向，比如你想用单一指标来评估模型，但是你的队友认为这样不好，那你怎么说服他们呢？ 为了方便让你的队友看到你建议的好处，我特地把这本书的每个章节都写的特别短， 你可以打印在一张两纸上。这样你让你的队友看这一两页纸，他就能明白这么做的好处了。

有些优先级的改变，可以让团队的效率产生巨大的提升。 通过本书，可以了解哪些事情是要优先要考虑的。让我们通过这本书，让你成为团队的超级英雄吧~

3.阅读本书的先决条件

如果你已经学习过机器学习课程或你有监督类机器学习的从业经验，那你将很容易读懂本书所讲的内容。

本书假设你已经熟悉了监督类机器学习(虽然当前有各种各样的机器学习，但是监督类机器学习仍然是应用最多的方法)， 比如线性回归、逻辑回归、神经网络等等。

我会经常提到神经网络(也称作深度学习)，因此你至少需要知道它们的基本概念。 如果你对神经网络等相关概念不是特别了解， 你可以看下 Coursera 上前三周的机器学习课程， 课程地址是：http://ml-class.org。 你也可以看我之前录制的零基础入门机器学习课程：**点击进入**。

4.驱动机器学习长足进步的因素

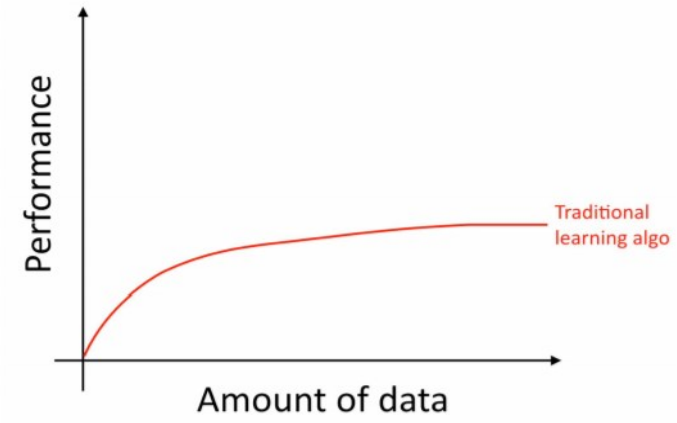
很多机器学习想法(如神经网络)，几十年前就已经提出来了，为什么这些想法，现在突然就火了呢?

促进这一现象的最大两个驱动因素是:

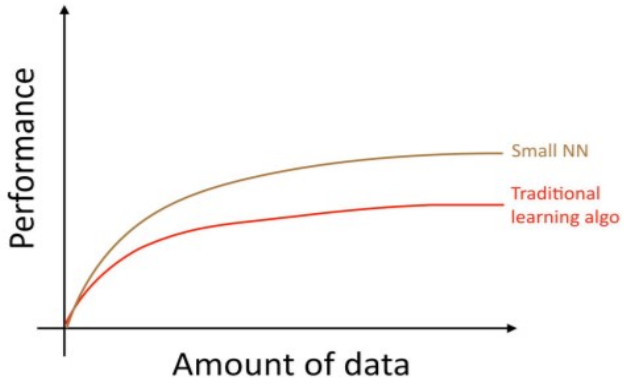
1.数据规模 现在人们大量使用诸如手机，电脑等电子设备。使用的过程中， 产生了大量数据，这些数据正是机器学习算法所必须的。

2.计算能力 因为计算能力的提升，我们近些年才可能训练出足够大的神经网络，来将 这些数据利用起来。

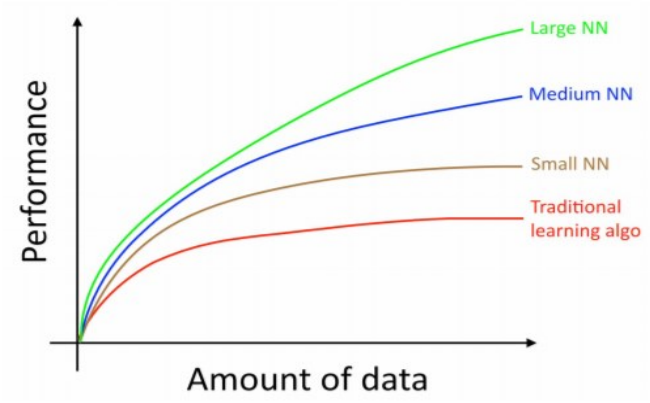
具体来说，即使你积累了大量的数据，旧的算法比如逻辑回归，也不可能会有大的提升。 当你提供再多的数据时，你也会发现它的学习曲线非常平坦。



就好像旧的算法并不知道该如何利用现有的数据一样。 但如果你使用神经网络，效果就会不一样了。 当你使用一个小型的神经网络时，你会发现模型性能略有提升。



这里的小型神经网络是指，层数、神经元和参数较少的神经网络。 接着，你再使用一个大型的神经网络试试，你会发现模型性能有了大幅度的提升。



从上图可以看成，想获取最好的性能，就需要

1.使用更大型的神经网络

2.使用更大量的数据集

还有很多其他的细节，比如神经网络结构也非常重要，这方面近些年来有很多的创新。 但当前最可靠的改进算法性能的方法仍是：

1.训练更大型的神经网络

2.使用更大量的数据集

实现上面这两个步骤，其实异常的复杂。本书将会详细讨论里面的细节。 我们先从对传统算法和神经网络都有用的策略开始说起，然后说下，如何在构建深度机器学习系统时，使用最先进的策略。

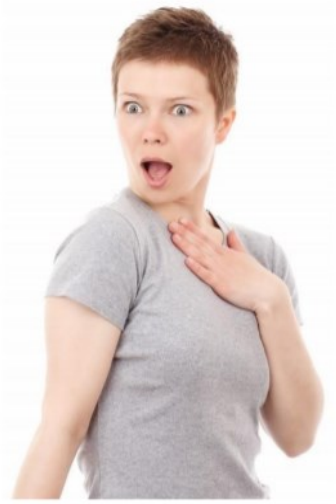
第二章：构建开发和测试数据集

5.开发和测试数据集的重要性

让我们回到之前猫咪图片的例子中，你开发了一款移动端 APP， 用户上传了各种各样的照片，你想让计算机自动区分出哪些照片中含有猫咪。

同时，为了训练模型，你的队友们从互联网上，找到了大量的猫咪图片(正例)和非猫咪图片(负例)。 他们按 7:3 的比例，把数据分成了训练样本集和测试样本集。通过这些数据， 他们构建了一个能在训练样本集和测试样本集上表现都很好猫咪检测器模型。

但是，当你们把这个模型应用到移动端的 APP 上式，你发现这个模型不管用了，检测准确率极差。



咋回事呢?

你发现，用户上传的图片跟你们从网上采集来的图片大不一样。 用户上传的图片大部分都是用手机拍摄的，因此像素较低，比较模糊，而且光线也不是很好。 因为你们之前用的训练数据全部来自与网上。你们的算法模型并不能适配真实的数据，即来自手机拍摄的照片数据。

在大数据时代之前，把数据按 7:3 分割成训练样本集和测试样本集，是一种非常通用的做法。实践证明这样的确有用，但是对于越来越多应用来说，它们的训练数据跟最终的实际数据分布不太一样， 那再这样做，可就不是一个很好的主意了。

那怎么做呢？通常我们会定义三个数据集：

1.训练数据集 — 用来训练你的算法模型

2.开发数据集 — 用于调整学习模型，比如调整参数、选择特征等。这部分数据也称 作预留交叉验证数据集。

3.测试数据集 — 用于评估模型性能，这部分数据集不参与模型调整和参数更新。

当你定义好开发和测试数据集后，你的团队就可以做出很多其他的尝试(比如再学习算法上试试其他的参数)，来找到表现最好的想法。开发和测试数据集可以让你的团队，快速的看到算法是否可行，表现如何。

换句话说，使用开发和测试数据集，可以引导你的团队，针对你们的机器学习系统，直接做出最重要的决定和改变。

所以，你应该选择最终实际要用的数据，作为开发和测试样本集。

因此，当你的最终数据，跟你预先采集的训练数据，分布不一致时，你最好不要简单的将可用数据的 30%作为测试数据集。

如果你的 APP 还没有上线，或者你还没有的用户，此时你可能无法采集来自用户的实际数据。但是我仍旧建议你想办法， 找到真实数据相似的数据。比如让你的朋友帮忙，用手机拍一下猫咪的照片发给你。当你的 APP 上线后，你就可以用真实的数据， 来替换之前开发和测试样本集中的近似数据了。

如果你实在找不到能近似实际情况的数据，你可以使用从网上采集来的数据，来训练你的模型。 但一定要注意由此可能导致的泛化风险，即你训练时，模型表现的很好，但是在实际使用时，却表现得很差。

你需要考虑用多大得成本，来构建更好的开发和测试数据集。但是永远不要假设训练数据跟实际数据的分布相同。 测试数据集尽量选择与最终数据相似的数据，而不是像训练一样，啥数据都可以选。

6.使用同一分布的开发和测试数据集

你依据市场大小，将猫咪数据分成了四个部分：美国、中国、印度和其他。

建立开发和测试数据集时，我们将美国和印度的数据放入开发测试集中， 将中国和其他地区的数据放大测试数据集中。换句话说，我们可以随机将两部分数据放到开发数据集中， 将另外两部分数据放到测试数据集中，但这样做真的好吗?

一旦定义了开发和测试数据集，你的团队就会聚焦在开发数据集上的性能提升。 因此，开发数据集应该反应出你们最想解决的迫切问题，即机器学习模型需要在四个地区都表现很好，而不是其中的两个。

还有一个问题时，因为开发和测试数据的分布不同，有可能在开发数据集上表现很好的模型，却在测试数据集上表现得很差。 发生这样得问题时，会非常打击人，大家得努利都白费了。希望这件事情不要发生在你们团队身上。

举个例子：假如你的团队开发得系统，在开发测试集上表现很好，但是在测试集上表现得并不理想。 如果你的开发数据集和测试数据集的分布是一致时，这时你会有非常明确得错误诊断方法， 即：你的模型在开发数据集上发生了过拟合。这时解决方案也很清楚，增加开发数据集中得样本数量即可。

但是如果开发数据集和测试数据集的分布不一致时，解决方案就会不那么清晰了。产生这种错误的原因可能有以下几种：

1. 模型在开发数据集上发生了过拟合

2. 测试数据更难处理和预测，你的算法可能和预期是一致的，此时没有显著提升性能的 可能性了。

3. 测试数据也肯能不是更难，只是与开发数据集中的数据不同。所以在开发数据集上表　现很好的模型，　在测试数据集上却并不管用，此时你在开发数据集上，所做的任何努力都白费了。

机器学习应用相关的工作是非常困难的。如果开发数据集和测试数据集不一致，会带来很多不确定性，　无法确定开发数据集上的改善，对测试数据集一定管用。因此如果开发数据集和测试数据集数据不一致，　就很难判断哪些工作是有效的，哪些工作是无效的，从而无法确定工作的优先级了。

如果你正在处理基于第三方基准的问题，第三方可能已经指明来来着不同分布的开发和测试数据集。　这时相对于来自同一分布的开发和测试数据集，运气对你来说就显得尤为重要了。　将一种分布数据上训练出来的模型，推广到另外一个分布的数据集上，是当前非常重要的研究课题。　但是如果你的目的是，解决某个特定机器学习应用的问题，我建议你使用来自同一分布的开发和测试数据集。　这会让你的团队工作更加高效。

7.开发和测试数据集多大合适

为了检测出不同算法之间的差异，开发数据集应该足够大。 比如，分类算法 A 的准确率为 90.0%，分类算法 B 的准确率为 90.1%， 如果开发数据集中只有 100 个样本，那你无法检测出这 0.1%差在哪儿。 100 个样本的开发数据集实在时太小了。 通常，我们会在开发数据集通中，放入 1000 到 10,000 个样本。 对于 10,000 个样本来说，找到这 0.1%并改进它，就相对容易多了。

别小看这 0.1%，对于一些成熟和重要的应用来说， 比如广告推荐、网页搜索、产品推荐，这 0.1%就直接影响着公司的利润， 因此这些团队都非常积极的改进模型，哪怕是 0.1%。 这种情况下，开发数据集中的样本数量会远大于 10,000 个，目的就是能让模型获得改善，哪怕是微小的改善。

那对于测试样本集呢？它多大合适呢？它也应该要足够大，这样他给出的评估指标才相对靠谱。 通常人们用所有数据的 30%作为测试数据集。 如果你的数据量相对适中(比如有 100 到 10,000 个样本)，这么做会非常有效。 但是在大数据时代，我们的机器学习模型，往往会处理数 10 亿个样本， 即使少分配一些给开发和测试样本集，但是相对数量也会增加不少。 此时，对于开发和测试数据集来说，太大的数据集并没有必要，能评估出你的算法性能即可。

8.使用单值衡量指标

举个例子，分类准确率就是一个单值衡量指标(在开发数据集或测试数据集上，运行你的分类器， 分类器会返回一个单一数值，即分类正确的样本比例)。 依据这个衡量标准，分类算法 A 的准确率为 97%，分类算法 B 的准确率为 90%，此时，我们任务算法 A 更好一些。

相反，查准率(Precision)和查全率(Recall)就不是一个单一的衡量指标，它返回了两个用于评估模型的数值。 当衡量指标有多个数值时，就比较难区分出哪个算法更好了，比如：

Classifier	Precision	Recall
A	95%	90%
B	98%	85%

没法看出哪个分类算法更好，因此你也就没有办法立即选择其中的一个了。

Classifier	Precision	Recall	F1 score
A	95%	90%	92.4%

在开发阶段，你的团队会尝试很多想法，比如改变算法结构、更新模型参数、选择其他特征等等。 当建立一个单值衡量标准(比如使用准确率作为衡量标准)后，可以依据这个衡量标准，快速找出表现最好的模型。

如果你非常关心查准率和查全率，我建议你将这两个数值合并成一个数值。 比如可以使用查准率和产权率的平均值，作为最终的衡量指标。或者你可以计算“F1 得分”(查准率和查全率的加权平均值)。

Classifier	Precision	Recall	F1 score
A	95%	90%	**92.4%**
B	98%	85%	**91.0%**

使用单一衡量指标可以让你从大量分类模型中，迅速的找到合适的一个。它可以给出一个非常清晰的排序，从而加快你的开发进程。

最后，假如你在跟踪分类器四个关键市场的准确率：美国、中国、印度和其他。 这会产生四个准确率，通过计算平均值或加权平均值，你可以将这四个数值合并成一个数值。 平均或加权平均的方法，是将多个指标合并成一个指标最常用的方法。

9.优化性指标和约束性指标

还有一种合并多个衡量指标的方法。 假设你除了关心准确率外，还关心算法执行耗时。你需要从下面三个算法中选择出合适的算法：

Classifier	Accuracy	Running time
A	90%	80ms
B	92%	95ms
C	95%	1,500ms

把两个指标放到一个算式中计算好像有点怪怪的，比如： **Accuracy - 0.5*RunningTime** 。

这儿我们可以这么做：首先定义一下，我们可以接受的执行时间，比如我们可以接受算法执行 100 毫秒。然后，在可接受的时间范围内，最大化算法的准确率。这儿执行时间就是约束性指标，分类器必须满足这个指标，即最多执行 100 毫秒。分类准确率就是我们这里所说的优化性指标。

如果你需要权衡 N 个条件，比如模型文件大小(对于 APP 来说，文件大小非常重要，因为用户通常不会下载太大的 APP)、执行时间和准确率。你可以将其中的 N-1 个条件作为约束性指标，让它们满足指定的值即可。然后指明其中一个作为优化性指标。比如，为文件大小和执行时间设置一个阈值，然后在此基础上，尽量优化准确率。

最后，假如你正在设计一款硬件设备，这个设备使用麦克风来检测用户是否说了某个特定的唤醒词，如果用户说了唤醒词，则启动设备。这个功能在很多大公司的产品中都有所体现，比如唤醒亚马逊的 Echo 可以说“Alexa”，唤醒苹果的 Siri 可以说说“Hey Siri”，唤醒 Android 系统可以说“Okay，Google”，唤醒百度 APP 可以说“Hello Baidu”。设计这个功能时，你既要考虑系统误被唤醒的比例(用户没说这些唤醒词，系统却被唤醒了）又要考虑系统没被正确唤醒的比例(用户说了唤醒词，系统却没被唤醒)。这对这种系统，你可以将系统没被正确唤醒的比例作为优化性指标，将误被唤醒的比例作为约束性指标，满足 24 小时内误被唤醒的次数小于等于 1 次即可。

一旦你的团对优化指标达成了一致，那你们的进展速度就会很快了。

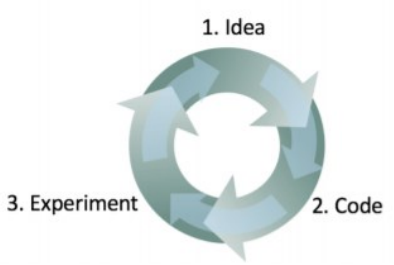
10.在开发/测试集和单值指标基础上，加速模型迭代

很难事先知道什么方法对新的问题有效。 即使是经验丰富的机器学习研究人员，在找到合适的方法之前，也会尝试大量的想法。 当构建一个机器学习系统时，我通常的做法是：

1.首先提出一些构建系统想法

2.然后用代码实现出来

3.通过实验来证实这个想法到达好不好(通常开始的一些想法并不凑效)，根据学习效 果，再提出其他想法，然后不断的迭代。



这是一个迭代过程，迭代的速度有多快，你的进展就会有多快。 这就是开发和测试数据集以及单值衡量指标重要的原因： 在开发和测试数据集上，使用单值衡量指标，可以让你你朝着正确的方向不断的前进。

相反，如果你没有指明开发数据集和单值衡量指标，每次你的团队开发一个新的猫咪分类器，你都需要将分类器打包进 APP，然后玩上数个小时，来观察模型是否有所改进。这会让你们的速度， 难以置信的慢。而且，如果你的团队将分类准确率从 95.0%,提升到了 95.1%，通过玩 APP 你可能无法感觉到这细微的改善。 然而逐步积累的 0.1%，就可以让你的 APP 产生很大的进步。使用开发数据集和单值衡量指标后，你可以快速的检测出，哪些想法可以带来细微的改善。 这时你就可以快速的做出决策，哪些想法继续改进，哪些想法应当放弃。

11.何时需要改变数据集和衡量指标

当开始一个新的项目时，我会快速的选择开发和测试数据集，这样可以给团队定出明确的目标。

我一般会要求我的团队在一周内，提出初始的开发/测试数据集和初始的衡量指标。 一开始的想法不完美并没关系，但是要快，千万不能过度思考。 但是这一做法不适合成熟的机器学习应用，比如反垃圾有效系统， 据我所知，真的这些成熟的系统，工作人员会花费数月的时间来优化开发/测试数据集。

如果后来你发现，初始的开发/测试集或衡量指标无法到达目标， 那你就需要尽可能快的改变它们了。比如，在开发数据集和衡量指标上， 分数算法 A 好于分类算法 B，但是在实际的产品中，分类算法 B 表现得更好， 这就意味着，你需要改变你的开发/测试数据集或衡量指标了。

开发数据集和度量指标错误得评估算法 A，可能有以下三个原因：

1.实际的数据分布与开发/测试集中的数据不同

比如初始的开发/测试集中都是成年的猫咪图片，而你的应用中，用户上传的都是小猫咪的图片， 这时开发和测试集中的数据分布不同于真实的数据分布， 在这种情况下，你需要更新你的开发/测试样本集，让其于实际的数据分布一致。

2. 在开发数据集上发生了过拟合

重复的开发数据集上评估你的想法，可能会逐渐让算法产生过拟合现象。 当你完成开发时，你应该在测试数据集上评估你的系统优劣。 如果你发现算法在开发数据集上的编写远好于测试样

本集，这就意味着你的算法在开发数据集上产生了过拟合。 这时，你需要获取一个新的开发数据集。如果你需要跟进你团队的进展，你可以定期的在测试数据集上评估你的系统， 比如一周一次或一个月一次。 但是对算法做出决策时，不要用测试数据集上的结果， 比如不能用测试结果的好坏，来决定是否要回滚到上一周的系统上。 如果你这么做了，很可能会让你的算法在测试数据集上，发生过拟合，之后你就不要指望测试数据集能给你提供无偏的估计了。

3. 度量指标跟项目需要优化的目标不一致

假设你是用分类准确率作为猫咪应用的衡量指标。 在此指标下，分类算法 A 优于分类算法 B。 但是当你尝试使用这两个算法时，发现算法 A 偶尔会让色情图片也通过筛选。

算法 A 虽然准确率更高，但是偶尔让色情图片通过，这却是不可接受的。 这时度量指标无法识别出，其实算法 B 更适合于我们的产品。 这时度量指标就不能帮助我们找出更好的算法了。此时我们就需要改变我们的度量方法了。 比如算法让色情图片通过的时，我们就加入很重的惩罚项。我强烈建议使用新的度量指标， 来给团队定义一个新的优化目标，而不是在没有可信度量指标时，通过手动的方式选择算法。

在项目开发时，变更开发/测试集和评估指标是很平常的事。拥有一个初始开发/测试集和衡量指标， 可以帮助你快速的迭代。如果你发现开发/测试集和度量指标，不能再给你们提供正确的前进方向了， 这也没什么大不了的，只需改变它们，让你的团队知道新的方向即可。

12.总结：构建开发和测试数据集

1.选择与实际数据(或你期望的数据)一致的开发和测试数据集，这些数据可能与你的训练数据不同 。

2.开发数据集和测试数据集中的数据分布应当一致 。

3.选择单值衡量指标作为你的团队优化目标，如果要考虑多个目标，可以通过一个表达式(比如求平均)将多个指标合并成一个指标，也可以把指标分为约束性指标和优化性指标。

4.机器学习是一个高速迭代的过程，在目标满足前，你可能需要尝试数十个想法 。

5.拥有开发/测试集和单值衡量指标后，可以让你快速评估你的算法，从而加速迭代 。

6.当开始一个全新的应用时，尽量在一周内，快速的建立开发/测试集和衡量指标。如果时成熟的应用，花长一点时间也没关系。

7.当你有非常多的数据时，将数据按 7:3 的比例分成训练和测试样本集并不合适，此时开发和测试样本集中的数量要远低于 30% 。

8.你需要有足够大的开发数据集(但也不要太大)，让你能看出算法精度上有意义的改变。测试样本集中的数据要尽可能的多，这样评估出的结论才能让人信服。

9.如果你的开发数据集和评估指标，已经不能再给你的团队指明正确的方向了，那你需要迅速的改变它们：

1).当开发数据集上发生过拟合时，此时添加更多的数据到开发数据集中

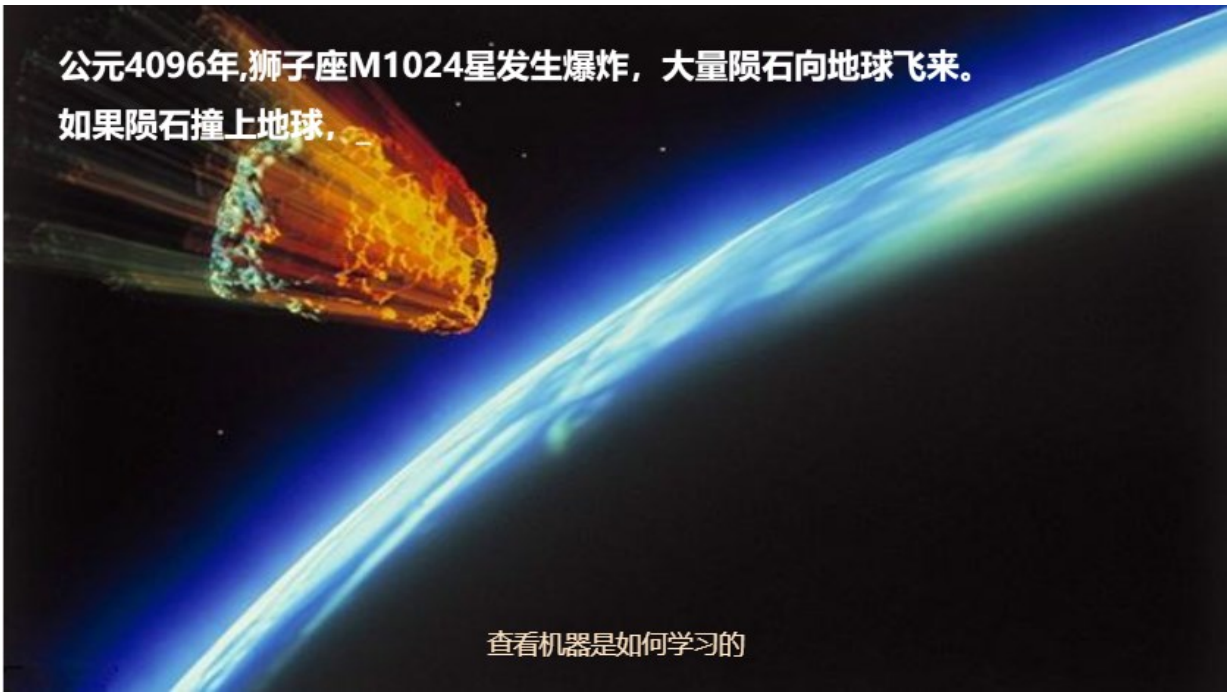
2).如果真实的数据与开发/测试集中的数据不一致时，更新开发/测试数据集即可

3).如果衡量指标不能衡量你所关心的最重要事情，则需要更换衡量指标

轻松一刻

看书累了没，看看机器是如何拯救地球的：让机器学会打游戏之陨石坠落

http://www.insideai.cn/game1_fall.html (点击进入)



第三章：基本的误差分析

13.快速构建你的系统，然后迭代

您想要构建一个新的反垃圾邮件系统。你的团队有几个想法：

1.收集大量的垃圾邮件。例如，设置一个“蜜罐”：故意的向已知的垃圾邮件发送人发送虚假的电子邮件地址， 这样你就可以自动获取由他们发出的垃圾邮件。

2.实现电子邮件内容理解功能。

3.实现邮件信封和邮件头特征理解功能，以便了解邮件都通过了哪些互联网服务器。

4......

尽管我在反垃圾邮件方面做了大量的工作，但我仍然很难从上述的方向中做出选择。如果你不是这个应用领域的专家，进行选择就更困难了。

所以，不要一开始就尝试设计和构建完美的系统。相反，你可以先用较短的时间(或许也就花几天时间)，快速建造和训练一个基本系统， 即使这个基本系统距离“最好”的标准相差甚远，建立一个基本系统是很有价值的：你可以很快找到一些线索，发现最值得投入时间的方向。 接下来的几章我们将会展示如何找到这些线索。

14.误差分析的重要性和大致过程

当你在玩你的关于猫咪的应用时，你会注意到几个误把狗狗识别为猫咪的例子。有些狗狗看起来真的太像猫了！

你的成员提议整合第三方软件，使系统能够在识别狗的图像上表现的更好。 这些改动需要花费一个月的时间，而团队成员却热情高涨。 你应该让他们继续吗?

在投资一个月的时间之前，我建议你首先评估一下，这项工作实际上能让你的系统准确率提升多少。 然后你可以更理性地决定，是用这一个月来完成这项改动还是做其他更重要的事情。

如何进行评估呢？具体的做法如下：

1.收集错误分类的 100 个示例样本。

2.手动查看这些例子，并计算它们中有多少是狗的图像。

查看错误分类的过程叫做误差分析。 在这个例子中，如果把图片错误分类为狗的情况，只占错误分类的 5%， 那么，不管针对狗的识别做多少改进，你最多只能减少总错误的 5%。 假如你的系统有 90%的准确度，你对系统所做的更改， 很可能最多让系统的准确度达到 90.5%（错误率从 10%降到了 9.5%，10%*5% = 0.5%，在原来 10%的错误率基础上减少了 5%）。

相反，如果你发现所有识别错误的情况中，有一半出现在狗的图片识别上， 那么你就可以确信，队友所提议的项目，会对系统会产生巨大的影响。 它可以将准确度从 90%提高到 95%（50%的误差相对减少，系统出错率从 10%下降到 5%）。

这个简单的误差分析过程为您提供了一种快速评估方法， 决定是否需要整合第三方软件用于狗狗图像的识别。它提供了一个定量判断的依据。

误差分析通常可以帮助你弄清楚不同方向有多大的潜力。 我看过许多工程师不愿意进行误差分析。 大刀阔斧地实施一些想法总是感觉起来更令人兴奋，很多工程师会不假思索地执行一个想法却不质疑这个想法是否值得花时间投入。 这是一个常见的误区：这么做可能导致你的团队花一个月的时间来推动方案后，却发现这个方案几乎没有带来什么好处。

手工检查 100 个例子并不需要很长时间。即使每张图片花了一分钟，你也会在两小时内完成。这两个小时可以帮你省下一个月白费的努力。

误差分析指的是对错误分类样本的分析过程，这样您就可以理解出现错误的根本原因。 这可以帮助设立项目的优先级——就像在这个例子中一样——也有助于你找到新的方向。 我们接下来会详细讨论这个问题，接下来的几章将会介绍进行误差分析的最佳实践。

15.并行评估多个想法的可行性

你的团队有几个关于改进猫咪检测器的想法：

1.解决你的算法中，把狗识别为猫的错误。

2.解决你的算法中，把大型猫科动物（比如狮子，美洲豹等）识别为宠物猫咪的错误。

3.改进系统对模糊图像识别的性能。

4....

你可以有效地并行评估所有这些想法。 我通常会在浏览 100 个错误样本的时候，创建一个表格。 我也记下了一些评注帮助我记住具体的例子。 为了说明这个过程，让我们看一看在这个应用中可能会用到的一个表格：

Image	Dog	Great cat	Blurry	Comments
1	✔			Unusual pitbull color
2			✔	
3		✔	✔	Lion; picture taken at zoo on rainy day
4		✔		Panther behind tree
% of total	25%	50%	50%	

上图，同时进行大型猫科动物和模糊图片检测。此外，一张图片样本有可能与多个类别相关联。

你可以先定义不同的错误类别（狗，大猫，模糊）然后对所有的样本进行手动分类。 实际上，一旦你开始浏览这些样本，你很可能会发现，其他错误分类方法可能更好。 比如说，你在浏览你的 Instagram（照片分享软件）中的照片时，发现照片的分组有一些错误。 你可能会添加一个新的分组，并整理之前的照片。手动查看算法的错误分类样本， 并且问问想想，人类是否可以正确地给图片贴上标签，这么做通常可以让你提出新的错误分类方案。

最有帮助的错误分类方案是可以让你算法性能获得提升方案。 例如，如果你想到使用“撤销”操作来恢复原始图片，那么对你的 Instagram 分类将会是最有帮助的。 但是，你不需要把自己局限于你所知道的错误分类。这个过程的目标是建立你对最有潜力的领域的直觉，并专注于此。

误差分析是一个迭代过程。如果你一开始对分类一无所知，不要担心。 在查看了一些图片之后，您可能就会对错误类别有一些想法了。 在动手对一些图像进行分类之后，您可能会想到新的类别并重新检查这些图片，加入新的分类方案。 在再次检查期间，你可能又会有新的分类想法，以此类推。

假设您完成了 100 个出错样本的误差分析，并得到以下几点:

Image	Dog	Great cat	Blurry	Comments
1	✔			Usual pitbull color
2			✔	
3		✔	✔	Lion; picture taken at zoo on rainy day
4		✔		Panther behind tree
...	...	...	...	...
% of total	8%	43%	61%	

你现在知道，在一个项目中解决关于狗的识别的问题，最多可以消除 8%的错误。 针对大型猫科动物或模糊图像上的工作可以帮助消除更多的错误。 因此，您可以在后两种类型中首先选择一个，并专注于这个方向。 如果你的团队中有足够多的人可以并行执行多个方向，你也可以两个方向都做， 让一部分工程师处理关于大型猫科动物的工作，另一些工程师处理模糊图像的工作。

误差分析并没有产生一个严格的数学公式来告诉你什么是具有最高优先级的任务。此外，你还须平衡性能提升和所需的工作量。

16.修正标记错误的样本数据

在误差分析过程中，你可能会发现，开发样本集中有的开样本被标记错误了。 当我说“标记错误”的时候，我的意思是这些图片被人为的标记错了。 即，在样本分类（x，y）中，y 是一个不正确的值。 例如，有一些本来不是猫的图片被错误标记成了猫，反之亦然。 如果你怀疑一部分样本被标记错误，就在下图的表格中再增加一列分组，来跟踪这些被错误标记了的样本：

Image	Dog	Great cat	Blurry	Mislabeled	Comments
...					
98				✔	Labeler missed cat in background
99		✔			
100				✔	Drawing of a cat; not a real cat.
% of total	8%	43%	61%	6%	

你应该纠正开发样本集中，这些被错误标记的样本吗？ 请记住，开发样本集的目标是帮助你快速的评估算法， 这样你可迅速区分算法 A 和算法 B 哪个更好。 如果样本集中的这些被错误标记的样本影响了你对算法优劣的判断，那么就值得花费一些时间来纠正错误标签了。

例如，假设你的分类器的性能如下：

1.在开发样本集上的整体准确率为 90%（总体错误率为 10%）

2.标记错误的样本带来的错误率为 0.6%（占总错误率的 6%）

3.其他原因导致的错误率为 9.4%（占总错误率的 94%）

这里，由于错误标记的样本带来的 0.6%错误率相对于其他原因带来的 9.4%错误率，显得不那么重要。 手动纠正出错的标签当然也可以，但是通常也没有必要这么做：因为系统是 10%错误率还是 9.4%错误率，相差并不大。

假设你一直在改进猫咪分类器并且达到了以下性能：

1.整体准确率为 98.0%（整体错误率为 2.0%）

2.标记错误的样本带来的错误率为 0.6%（占总错误率的 30%）

3.其他原因导致的错误率为 1.4%（占总错误率的 70%）

总错误的 30%因为样本标记错误导致的，对系统的准确度带来了重要的影响。 这个时候，值得花时间来提升样本标签的正确性。 处理被错误标记的样本将会帮助你判断一个分类器的错误率更接近于 1.4%还是 2%——这是一个相对显著地区别。

一开始的时候，在开发/测试样本集中，存在一些错误标签的样本通常是可以被容忍的， 直到错误标记的样本数量增加到足够影响样本整体的准确性的时候，才需要关注这一问题，并对错误的标签样本进行处理。

最后一章解释了如何通过算法的提升改进错误分类，比如对狗，大型猫科动物和模糊情况的处理。 你在这一章中已经了解到可以通过优化数据标签的方法对标记错误的分类进行处理。

不管你使用什么样的方法和规则来处理开发样本集中的错误标签，要记得在测试样本集中也使用同样的方法， 以保证开发样本集和测试样本集满足同样的数据分布。 同时修复开发样本集和测试样本集会防止出现第 6 章中讨论的问题–只优化了开发样本集却没有优化测试样本集， 随后面对测试结果你的团队可能会备受打击。

如果你决定提高标签的质量，可以考虑双重检查的方法，不仅检查你的系统中出现的错误分类的标签，还要检查系统正确分类的标签。 很有可能，原始的标签和你的学习算法在同一个样本中都出错了。如果只修复你的系统产生的分类错误的标签，会在你的系统评估中引入偏见。 如果你有 1000 个开发样本集，你的分类器的准确率是 98.0%，检查它出错的 20 个样本比检查正确的 980 个样本要容易得多。 在实际的开发中，开发者很容易只检查出错的样本集，这样的确会令偏差蔓延到一些开发样本集。 如果你只是对开发产品或应用感兴趣，这样的偏差是可以接受的。 但是如果你想在学术论文中引用测试结果或是需要严格的无偏差保证的测试样本集中使用这些测试数据会带来很大的麻烦。

17.人工观测数据集和黑盒开发数据集

假设你有一个很大的开发样本集，这个样本集有 5000 个样本，误差率是 20%。 这样你的算法产生了 1000 个错误分类。 要手动检查 1000 个图片样本需要花费太长时间，所以我们在误差分析的时候，不使用所有的这些分类错误的样本。

在这种情况下，我将显式地将开发样本集分成两个子集，其中一个是你会人工查看的，其中一个是你不会看的。 这部分人工查看的数据会被处理的更快。你可以把另一部分你不会人工查看的数据交给模型进行参数调优。

让我们继续上面的例子，在这个例子中，算法从开发样本集的 5000 个样本中产生了约 1000 个错误样本。 假设我们想要手动检查 100 个错误样本进行误差分析(抽取 10%的错误样本)。 你应该从开发样本集中随机选择 10%的样本，把它们放在我们称之为 人工观测数据集(Eyeball dev set)中， 来提醒我们使用自己的眼睛人工的查看这批数据。 (对于一个语音识别系统，你用耳朵来听的音频片段组成的集合，你也可以它为 Ear dev set)。 这个人工观测数据集有 500 个样本，其中约有 100 个在算法中被错误的标记。

开发样本集的第二个子集，称为黑盒开发数据集(Blackbox dev set)，拥有 4500 个样本。 你可以使用黑盒开发数据集来自动评估分类器的错误率。 你也可以用它来选择算法或超参数调优。然而，你应该避免用自己的眼睛去看它。 我们用这个词“黑盒（Block box）"是因为在这子集我们会用“黑盒”方式评估分类器。

为什么我们要显式地将开发样本集分隔成人工观测数据集和黑盒开发数据集？ 因为你会对人工观测数据集中的样本有直观的认识，你就更快的发现过拟合现象。 如果你看到人工观测数据集的性能提升远高于黑盒开发数据集，那么当前的这个人工观测数据集很可能发生了过拟合。 在这种情况下，你可能需要丢弃当前这个人工观测数据集， 可以从黑盒开发数据集移过来更多的样本或者获取新的已标记数据，来组成新的人工观测数据集。

显式地将你的开发样本集集分割成人工观测数据集和黑盒开发数据集， 可以让你及时发现在人工观测数据集上，手动的误差分析所产生的过拟合问题。

18.人工观测数据集和黑盒开发数据集多大合适

人工观测数据集应该设置的足够大，这样方便你找到主要的错误类别。 如果你在做一项人类可以完成的很好工作(比如识别图片中的猫)，那人工观测数据集要设置多大呢？下面是些粗略的指导准则：

1.如果在人类观测数据集上只产生了 10 个错误，则这个错误的数量太少了， 用这 10 个错误，很难精确评估出不同类别错误的影响。 但是如果数据量本身就比较少，而且无法投入更多的成本在人工观测数据集上， 那么有总比没有好，这 10 个错误，也可以帮助你划分工作的优先级。

2.如果你的分类器在人工观测数据集上产生了 20 个错误，那么你就可以粗略的估计错误的来源了。

3.如果产生了 50 个错误，那么你就可以很好的找到错误的来源了。

4.如果产生了 100 个错误，那么你就可以更好的理解错误的来源。 据我所知，人们往往会手工分析更多的错误，有时会多达 500 个。当你有足够多的数据时，这是一个不错的选择，这么做只有好处没有坏处。

如果说你的分类器有 5%的错误率，为了在人工观测样本集上找到 100 个错误样本， 你需要在人工观测样本集中放置 2000 个样本(因为 0.05*2000=10)。 为了得到足够多的错误样本，分类器的错误率越低，人工观测的数据集就要越大。

如果你在做一项人类无法很好完成的工作，这时在人工观测样本集上做实验作用就没那么大了。 因为人类很难分辨错误的原因。这种情况下，你可以忽略人工观测样本集。

我们将在后面的章节，讨论这些问题的处理准则。

针对黑盒开发数据集，我们之前说过开发数据集中通常包含 1000 到 10,000 个样本。 尽管更多的数据也没什么坏处，但 1000 到 10,000 个样本的黑盒开发数据集完全足够我们进行参数调优和模型选择了。 用于 100 个样本的黑盒开发数据集可能有些小，但是以然可用。

如果你的开发数据集比较小，可能无法将其分割成两个数据集，以满足手工调测数据集和黑盒开发数据集的需求。 此时，你可以将所有的数据都用做人工调测数据集，这时你需要手工处理所有的开发数据集样本。

在人工调测数据集和黑盒开发数据集中， 我认为人工观测数据集更加重要一些(假设你在处理一个人类能很好完成的工作，通过实验可以让你更加清楚的看成问题所在)， 如果你只有人工观测数据集，那么你可以在这个数据集上进行误差分析、模型选择、参数调优。 只有人工观测数据集的缺点是更容易产生过拟合的现象。

如果你有足够多的数据，那么人工观测数据集的大小取决于你有多少时间来进行手工分析。 我很少看到有人手工分析 1000 个以上的错误数据。

19.总结：基本误差分析

1.当你开始一个新项目时，特别是如果你在一个你不是专家的领域，很难正确地预测最有效的工作方向。

2.因此，不要一开始就尝试设计和构建完美的系统。相反，尽可能快地建立基本的系统(也许在几天之内)。 然后使用误差分析帮助你识别最有效的方向并迭代地改进你的算法。

3.从开发样本集中，选择分类错误的 100 个样本进行进行误差分析，评估出主要的错误类别。利用这些信息确定不同类别错误的修复优先级。

4.将开发数据集分成人工观测数据集和黑盒开发数据集。即用于人工处理的人工观测数据集和机器处理的黑盒开发数据集。 如果人工观测数据集上的性能远远优于黑盒开发数据集，说明你在人工观测数据集上已经发生了过拟合，此时应该加入更多数据到人工观测数据集中。

5.人工观测数据集应该足够大，这样你的算法就会有足够多的错误样本供你进行误差分析。 对于很多实际的应用来说，1000-10000 个样本是黑盒开发数据集理想的数量。

6.如果你的开发样本集数量不够大，不足使用这种分割的方式， 那么只需使用一个人工观测数据集来手动进行误差分析、模型选择和超参数调优。

第四章：偏差和方差

20.偏差和方差的概念及用途

假设你的开发、测试、训练样本集服从同一分布，那么获取更多的训练数据，可以让你的算法性能获得巨大的提升吗?

尽管获取更多的数据没啥坏处，但可能无法像你预期的那样，有很大提升。 而且采集数据本身会耗费大量的时间，那如何判断，什么时候需要添加数据，什么时候不需要添加数据呢?

机器学习中的误差主要来自两个方面：偏差和方差。 理解偏差和方差，可以帮助你决定是通过添加数据来提升算法性能，还是通过其他的策略来提升算法性能。

假如你希望你的猫咪检测器错误率在 5%以下，当前你的算法在训练样本集上的错误率为 15%，在开发样本集上的错误率为 16%。 此时加入更多的训练数据，可能并没有太大帮助。加入更多的训练数据，只会让你的模型更难训练， 你应该改变你的算法(之后的章节，我们将介绍为什么要这么做)。

如果算法在训练样本集上的错误率为 15%,而你希望准确率在 5%以下， 那么首要的问题是如何提升算法在训练样本集上的性能。 算法在开发/测试测试集上的表现，一般会差于在训练样本集上的表现。 因此如果你的算法在训练样本集上的准确率为 85%，则想都不用想，算法在开发和测试样本集上的准确率不可能达到 95%。

继续上面的例子，算法在开发样本集上的错误率为 16%，我们可以把这 16%分成两部分：

1.算法在训练样本集上的错误率，本例中为 15%，这通常称作算法的 **偏差** 。

2.算法在开发/测试样本集上相对训练样本集上高处的错误率部分， 本例中，算法那在开发样本集上的错误率比在训练样本集上的错误率高 1%,这通常称作算法的 **方差** 。

有些改进可以解决第一部分的问题，即偏差问题，从而可以提升算法在训练样本集上的性能；而有些改进可以解决第二部分问题，即方差问题，让模型具有更好的泛化效果。 为了找到更合理的改进方案，想想哪一部分更需要迫切解决，是非常重要的。

对偏差和方差有更直观的理解，可以帮助你更加有些的改进你的算法。

21.举例说明偏差和方差

对于猫咪分类器来说，理想的分类器能达到近乎完美的效果(比如人来分类)。

假设你的算法性能如下：

1.在训练样本集上的错误率为 1%

2.在开发样本集上的错误率为 11%

那模型的问题出现在哪儿呢？根据上一章的结论，我们算出偏差为 1%，方差为 10%（11% - 1%）。 方差较大，因此它的问题是 **高方差**。模型在训练样本集上的错误率很低，但是无法泛化到开发样本集上。 这种问题也称作 **过拟合**。

再考虑下下面这类问题：

1.在训练样本集上的错误率为 15%

2.在开发样本集上的错误率为 16%

我们算出此时的偏差为 15%，方差为 1%。 模型在训练样本集上表现得很差，错误率达到了 15%， 但是模型在开发样本集上的错误率只比在训练样本集上的错误率高一点点。 这时分类器的问题是 **高偏差**，但是低方差。此时，我们称算法发生了 **欠拟合**。

再考虑一种情况：

1.模型在训练样本集上的错误率为 15%

2.模型在开发样本集上的错误率为 30%

我们算出此时的偏差为 15%，方差也为 15%（30% - 15%）。 这时分类器既是高偏差又是高方差。 模型在训练样本集上表现很差，因此它有高偏差的问题，同时模型在开发样本集上表现的更差，因此它还有高方差的问题。 这时我们无法称模型是过拟合还是欠拟合，因为它既有高偏差的问题，又有高方差的问题。

最后，再考虑一种情况：

1.模型在训练样本集上的错误率为 0.5%

2.模型在开发样本集上的错误率为 1%

这时模型表现就很棒了，它既是低偏差又是低方差。如果模型达到这种水平，那么就恭喜你了，获得了非常好的分类器。

22.向最优的错误率看齐

在我们的猫咪识别器的例子中，最理想的情况是实现一个最优的识别器，提供接近于 0 的错误率。如果图片中有猫，人类几乎可以 100%识别出来；因此，我们也期望机器可以达到同样的水平。

和猫咪的例子相比，其他的问题相对复杂得多。例设，你正在开发一款语音设别系统，但你发现 14%的语音片段有太多的背景噪音，还有些连人类都无法识别的无序信息。在这种情况下，即使是最优秀的语音识别系统也会产生 14%左右的错误率。

假设你的算法在语音识别任务中的效果如下：

1.在训练样本集上的错误率为 15%

2.在开发样本集上的错误率为 30%

算法在训练样本集上的表现非常接近于最优的错误率 14%。因此在算法在训练样本集上的已经没有多大的提升空间了。但是，算法在开发样本集上的泛化效果并不理想；因此方差有广阔的提升空间。

这个例子和上一章提到的第三个例子很相似，在训练样本集上有 15%的错误率，在开发样本集上有 30%的错误率。如果最优的错误率接近 0%，那么训练样本集上的 15%错误率还是有很大的提升空间的。这就意味着偏差方面的优化将会非常有果效。但是如果最优的错误率是 14%，那么在训练样本集上，留给我们的分类器的提升空间就非常少了。

对于那些 z 最优错误率远大于 0 的问题，需要对算法的错误率做更加详细的分解。让我们继续上面提到的语音识别的例子，开发样本集上的总错误率为 30%，可以按照如下方法进行分解(同样的方法也可以应用于测试样本集):

1.最优错误率(不可避免的偏差)：14%。此时即使是世界上最最优秀的语音识别系统，

2.可避免的偏差：1%。通过计算训练集错误率与 z 最优错误率的差值得来。

3.方差：15%。开发样本集的错误率与训练样本集上错误率的差值。为了和之前讲到的概念关联，偏差和可避免偏差之间的关系如下：

偏差 = 最优错误率(不可避免的偏差) + 可避免的偏差

可避免的偏差反映出，你的算法距离最优的分类器还有多远的距离。

方差的概念和之前描述的相同。从理论上说，我们可以通过使用一个庞大的训练样本集，来使方差趋于 0。因此，当数据集足够大时，所有的方差都是可以被避免的，也就没有了不可避免方差的概念。

再考虑一个例子，最优的错误率是 14%，我们已知：

1.训练样本集错误率=15%

2.开发样本集错误率=16%

依据前几章的定义，我们称这个分类器为高偏差分类器，但是如果最优的错误率为 14%，则可避免的偏差是 1%，方差也是 1%，这个算法已经做的很好，需要改进的空间很少，只比最优错误率多 2%。

我们从这些例子可以看出，知道最优错误率，对我们之后的步骤，会有很大的帮助。在统计学中，最优错误率也被称为贝叶斯错误率或是贝叶斯率。

我们如何知道最优的错误率是多少呢？对于那些人类擅长的任务，比如图片识别或语音片段转录，你可以让人在训练样本集上对样本进行标记，然后计算人类的准确率。 这样你就可以得到大致的最优错误率了。但是，如果你要处理连人类都很难解决的问题(比如，把哪些电影或是广告推荐给用户)，估计最优错误率就变得困难的多了。

在 33 章-35 章（和人类水平进行对比的章节）中，我们会详细讨论，如何对比机器学习算法性能和人类水平。

在之前的几章中，你学习了如何通过观察训练集和开发集上的错误率，来估计可避免/不可避免偏差和方差。下一章将会讨论如何使用这些错误率，通过对错误率的分析，来设立优先级，决定要首先减少偏差还是方差。你的项目是高偏差(可避免的)还是高方差，将决定了你会采用什么样的技术，来解决问题。

23.方差和偏差的处理方法

处理偏差和方差的时候有几个最简单的准则：

1.如果可避免的偏差很高，则增加你的模型的规模(比如，在神经网络中增加更多的隐藏层或神经元)。

2.如果方差很高，就在训练样本集中增加更多的数据。

如果可以不受任何约束地扩大神经网络规模和训练数据量，那任何机器学习问题都会不在话下。

事实上，扩大学习模型的规模会逐渐带来计算复杂度的问题，因为训练一个巨大的模型是很慢的。获取更多的训练数据同样也会让你耗费巨大精力(即使是在互联网上，猫咪图片的数量也是有限的!)。

不同的模型架构，如不同的神经网络架构，会在你要解决的问题上，产生不同的偏差或方差。最近很多深度学习领域，出现了很多创新的模型架构。如果你正在使用神经网络，查看相关的学术文献会给你很多的启发。在 github 上也有很多开源的资源。但是尝试新的架构相对于简单的增加模型和数据的规模，不会带来那样直接和可预测的收益。

增加模型的大小可以逐渐减少偏差，但是也可能会同时增加方差和过拟合的风险。然而，过拟合的问题通常只会在你没有使用正则化时出现。 如果你使用了正则化的模型，你通常可以安全放心的增加模型的规模而不必担心会带来过拟合。

假设你正在使用深度学习，用到了 L2 正则化或是 dropout，在使用正则化的情况下，模型在开发样本集上达到最优的性能。如果你增加模型的规模，通常性能会保持不变或是有所提升；但是不太可能会明显变糟。避免使用更大的模型的唯一的原因是，随着模型规模的增加，计算量也会逐渐增加。

24.权衡模型的方差和偏差

你可能以前听过“权衡偏差和方差”。大多数机器学习改进方法中，有一些可以降低偏差但是会导致方差的上升，反之亦然。这个时候就需要在偏差和方差中进行权衡了。

举例来说，增加你的模型的规模，不管是在神经网络中增加神经元/隐藏层，还是增加输入特征，可以普遍减少偏差但是会增加方差。另一方便，增加正则化一般会增加偏差，但是可以减少方差。

现在，因为可以使用大量的数据并且可以训练大型的神经网络(深度学习)。我们有了更多的选择，我们可以在不降低方差的同时，来让偏差降低，反之亦然。

比如，可以使用扩大神经网络规模和应用正则化的方法，来减少偏差的同时不会明显的影响偏差。通过增加训练的数据量，你可以在不影响偏差的前提下，减少方差。

一旦你选定了一个适用与你当前任务的模型架构，你可能需要同时减少偏差和方差。当然选择这样的架构并不容易。

在接下来章节中，我们会讨论一些特定的方法，来降低偏差和方差。

25.减少可避免的偏差方法

如果你的学习算法中的可避免偏差很高，你可以尝试使用下面的方法：

1. 增加模型规模(比如神经元/层的数量)：这个方法可以减少偏差，因为这样可以使得你的模型可以更好的拟合当前的训练集。如果你发现使用这个方法时方差增加了，那就使用正则化，这一做法通常会抑制方差的上升。

2. 基于误差分析修改输入特征：如果说你的误差分析告诉你要创建更多的特征来帮助算法减少某一特定种类的错误（我们会在下一章讨论）。这些新的特征应该在偏差和方差方面都可以帮助你。理论上来说，增加更多的特征会相应增加方差；如果你遇到这种情况，就使用正则化，来抑制方差的增长。

3. 减少或去除正则化（L2 正则化，L1 正则化，dropout)：这会减少可避免偏差，但是会增加方差。

4. 修改模型架构(比如神经网络架构)：使得你的模型更加适合于你的问题，这个方法会对偏差和方差都造成影响。

关于降低偏差可能有个误区：

加入更多训练数据：这种方法对方差问题有帮助，但是通常无法解决偏差的问题。

26.训练数据集上的误差分析

在你期待你的算法可以在开发/测试集上表现良好之前，它首先必须能够在训练集上表现出众。

在之前章节介绍的那些可以处理高偏差的方法中，我有时会采用， 在训练数据集上进行误差分析，这种分析方法跟在人工观察数据集上的错误分析方法类似。 当你的算法偏差很高时-也就是这个算法不适合训练集时，这样的误差分析会很有帮助。

举例来说，假设你正在开发一款语音识别系统，并且已经采集了一个训练样本集。 当你的系统在这个训练集上表现的不好时，你可能会从算法表现很差的语音片段中选择约 100 个出来，自己用耳朵听一下， 来找到算法主要出现了哪几类错误。跟之前在开发数据集上的错误分析类似，你可以把错误分成以下几个类别：

Audio clip	Loud background noise	User spoke quickly	Far from microphone	Comments
1	✔			Car noise
2	✔		✔	Restaurant noise
3		✔	✔	User shouting across living room?
4	✔			Coffeeshop
% of total	75%	25%	50%	

在这个例子中，你可能会发现，你的算法对有背景噪音的训练样本很难处理。这样，你就可以专注于那些擅长处理背景噪音的方法了。

你也可以使用双重的检查，把同样的音频数据(训练集)交给人来进行标记。如果这些包含背景噪音的音频连人类也很难识别他们到底在说什么，那么要求任何的算法来正确的识别这些音频，就实在是太强算法所难了。我们会在之后的章节讨论，把算法性能和人类水平进行比较的好处。

27.减少方差的方法

如果你的学习算法的方差很高，你可能需要尝试下面的方法：、

1. **增加更多的训练数据**：只要你可以拿到更多数据，并且有足够强计算能力，这是最简单可靠的处理方差的方法。

2. **使用正则化(L2 正则化，L1 正则化，dropout)**：这个方法可以减少方差，但是会增加偏差。

3. **提早停止(基于开发集的错误率，提前停止梯度下降)**：这个方法可以降低方差但是会增加偏差。提早停止的做法和正则化方法行为上有所类似，所以一些人也会把这种方法归入正则化方法。

4. **特征选择时减少输入特征的数量/类型**：这种方法可以会对方差问题有帮助，但是同时可能增加偏差。轻微的减少特征的数量（比如，从 1000 个减到 900 个）可能不会对偏差造成较大的影响。但是大幅度的减少特征数量（比如从 1000 个减到 100 个，10 倍的缩减）很可能对偏差造成严重的影响，因为这样会导致有用的特征大大减少。在现代的深度学习中，当数据很丰富时，一般不会采用特征选择的方法，而是尽可能地让算法获取所有的特征，并且让算法自己选出一些来用。但是当你的训练样本集很小时，特征选择会非常有用。

5. **减少模型规模(比如神经元/层的数量)**：要小心使用这个方法。这个方法可以降低方差但是很大可能增加偏差。因此，我不推荐使用这个方法来处理方差问题。
增加正则化通常会提升分类器的性能。减少模型的规模的优点在于可以降低计算开销，进而提升训练模型的速度。如果加速模型训练很有用，那么你当然可以考虑降低模型规模。但是如果你的目标是减少方差，而不关心计算开销，那就考虑使用正则化的方法。

这里还有两个附加的策略，与 25 章处理偏差问题的方法相同：

1. **基于误差分析修改输入特征**：如果说你的误差分析告诉你要创建更多的特征来帮助算法减少某一特定种类的错误。这些新的特征应该在偏差和方差方面都可以帮助你。理论上来说，增加更多的特征会相应增加方差；如果你遇到这种情况，就使用正则化，来抑制方差的增加即可。

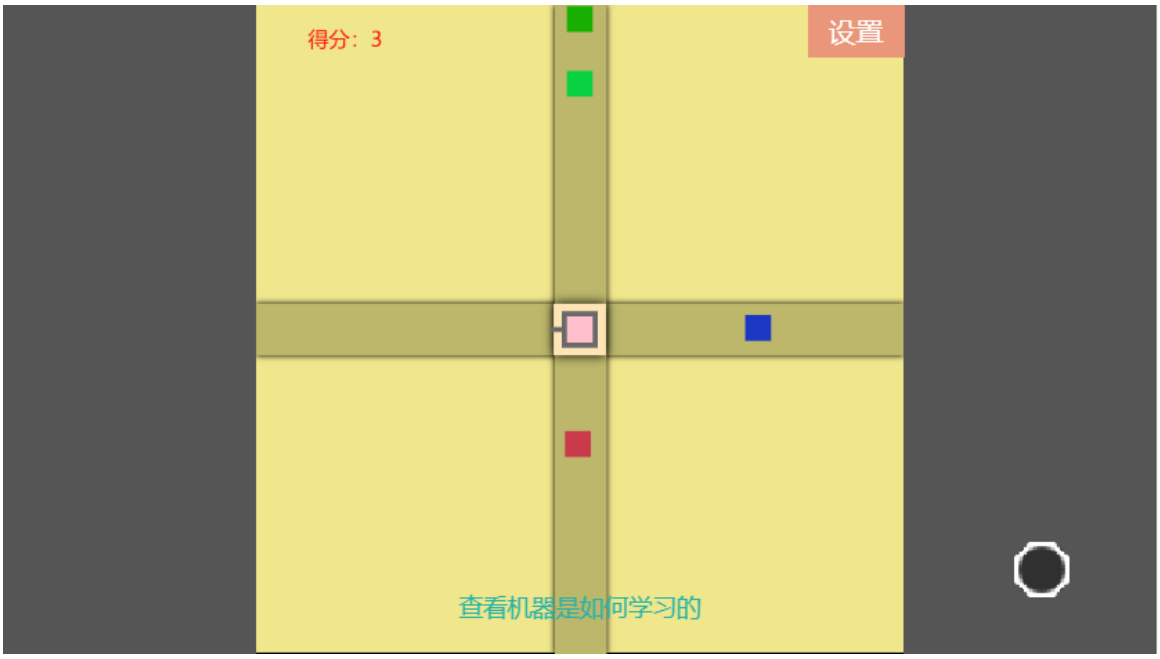
2. **修改模型架构(比如神经网络架构)**：使得你的模型更加适合于你的问题，这个方法会对偏差和方差都造成影响。

轻松一刻

看书累了没

看看面对黑压压的敌军，机器能否守卫住红城：让机器学会打游戏之红城保卫战

http://www.insideai.cn/game2_save.html(Ctrl+鼠标 点击进入)

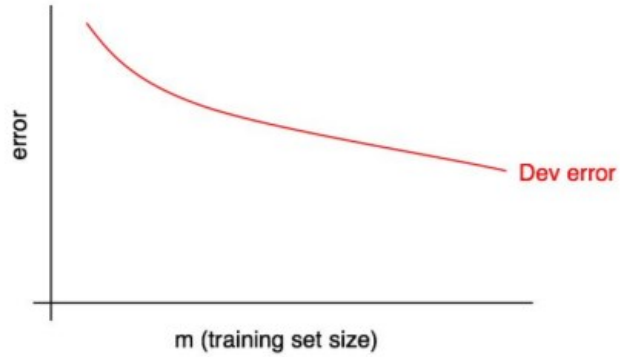


第五章：学习曲线

28.通过学习曲线诊断偏差和方差

我们已经了解了一些方法，可以算出有多少错误是来自于可避免得方差和偏差了。这些方法包括评估最优错误率、计算模型在训练样本集和开发样本集上的错误率。下面我们讨论两外一项可获得更多信息得方法：绘制学习曲线。

学习曲线显示出模型在开发数据集上的错误率与训练样本数量的关系。绘制这个曲线时，你需要设置不同的训练样本集大小。比如你有 1000 个样本，你可以分别用 100、200、300、……、1000 个样本来训练模型。然后就可以绘制出模型在开发样本集上的错误率与训练样本数量的对应关系了，比如下面这个例子：



从上图可以看出，随着训练样本数量的增多，模型在开发样本集上的误差在逐渐下降。

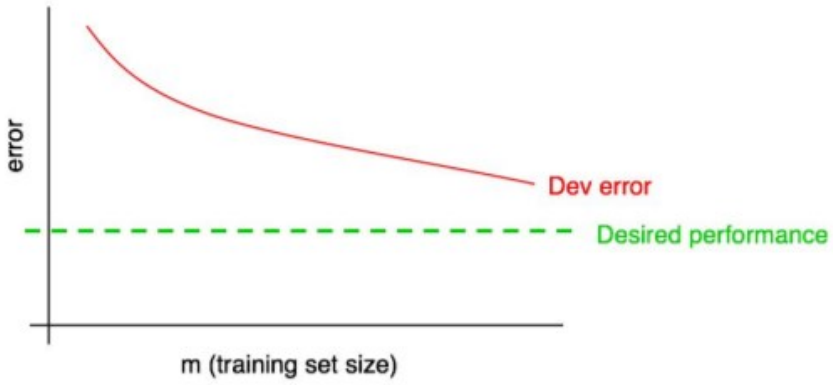
对于我们的模型来说，我们通常会有些“期望的错误率”，我们希望我们的模型最终能达到这一效果，比如：

1. 我们期望模型达到人类的水平，则人类能达到的错误率就是我们期望的错误率。

2. 如果我们的算法服务于某些产品(比如猫咪分类)，我们会有个大概的感觉，当模型达到什么样的水平时，用户会有比较好的体验。

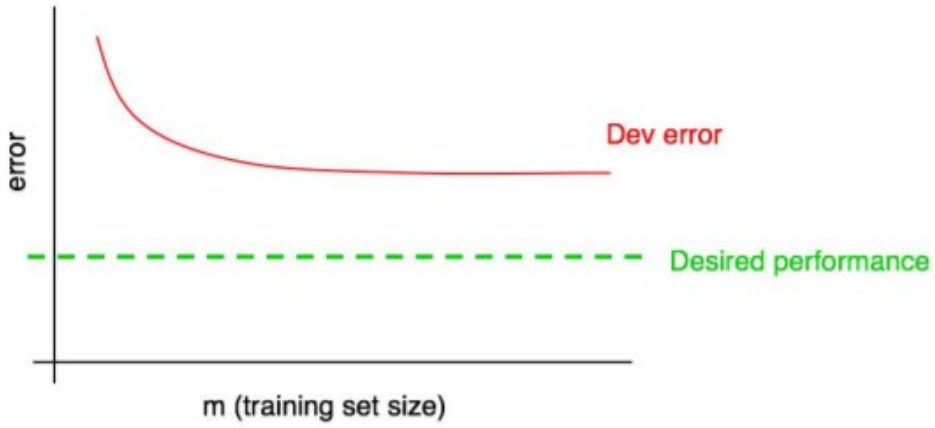
3. 如果我们已经在某个应用上工作了很长时间，你应该能感觉出，下一季度/年度模型能达到的合理水平。

我们可以把期望水平加入到学习曲线中：



你可以直接看出加入数据后，"开发错误率"离你期望的水平还差多远。从上面的例子中可以看出，似乎增加训练样本集中样本的数量，就可以让算法达到我们预期的水平了。

但是如果开发错误率变平了(不再下降了)，你可以清晰的看出添加数据无助于你达到目标：



看看学习曲线就能帮助你发现收集更多的数据并无效果。 而不是花费数月收集了两倍数据后，才发现更多的数据并没有用。 学习曲线有个缺点是，它没有办法帮你准确的预测模型会达到什么水平，它只能反映出一个趋势。

还有个图形也可以帮你评估数据量对模型性能的影响，即：训练错误率。

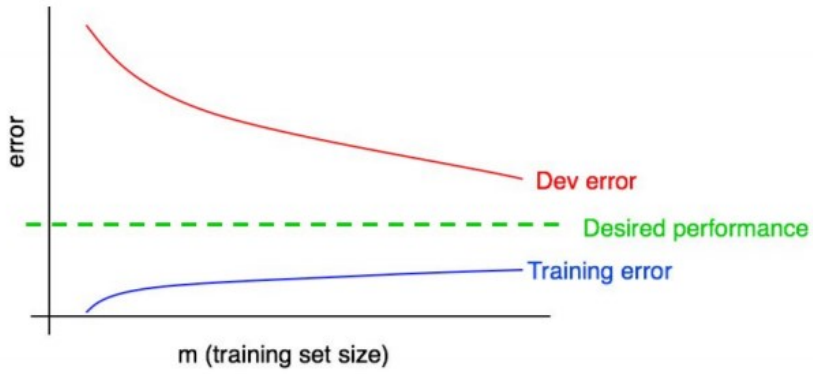
29.将训练错误率用图形绘制出来

你的开发/测试错误率应该会随着训练样本数量的增加而减少。但是训练错误率通常会随着样本数量的增加而增加。假设你的训练样本集中有两个样本：一张猫咪图片和一张非猫咪图片。这时算法很容易就会记住这两个样本，从而得到 0%的训练错误率。即使样本集中一个甚至两个都标记错误了，算法也很容易就能记住它们的标签。

现在假设你的训练样本集中有 100 个样本。可能还有些样本标记错误了，或者非常模糊，连人都分不清图片上是不是有猫。此时或许模型还是能记住每个样本对应的标签，但是此时很难到达 100%的准确率了。样本数量从 2 个上升到 100 个，你就会发现训练准确率在下降了。

最后，假如你的训练样本集中有 10,000 个样本。这时，算法就很难拟合这 10,000 个样本了，如果样本集合中，还有些是模棱两可的或标记错误的，那就更难拟合了。因此的算法会在这个训练样本集上表现的更差一些。

让我们把训练样本集也加到之前的图形中：

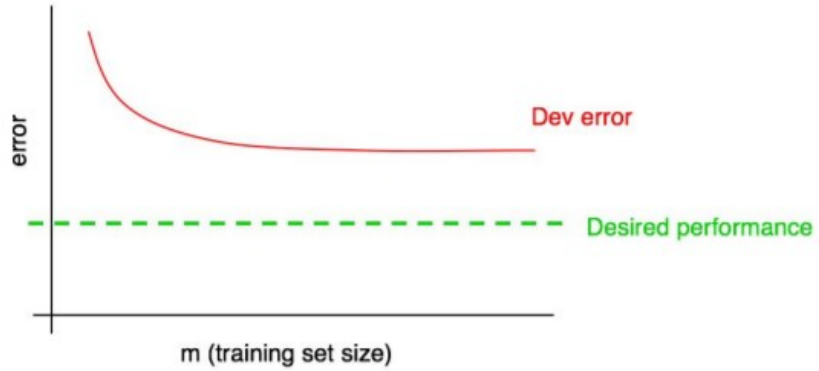


你可以看出随着训练样本的增加，蓝色曲线代表的训练错误率一直在增加。而且，可以看出，算法通常在训练样本集比在开发样本集上的表现更好一些：因此红色曲线代表的开发错误率始终在蓝色曲线代表的训练错误率上方。

下面一章我们将详细解释这个图形。

30.高偏差时的学习曲线

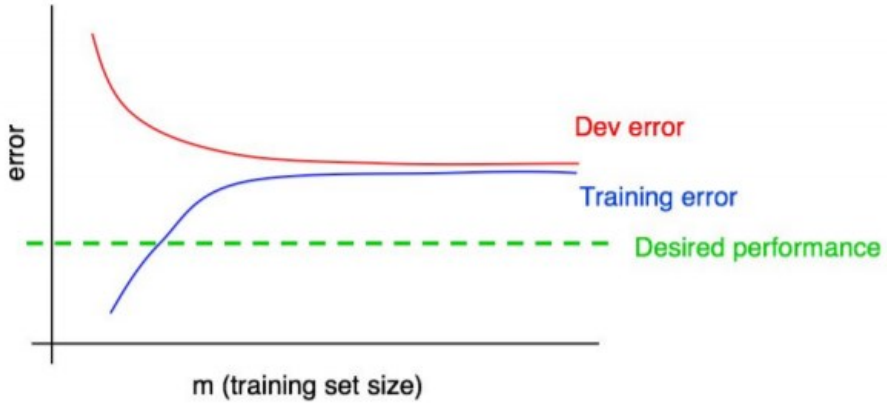
假设你的模型开发错误率图形如下图所示：



我们之前说过，如果开发错误率的曲线变平了，则仅仅添加数据，可能无法让你的算法到达预期的水平。

但是很难知道红色的曲线接下来的走势会是什么样子。如果开发样本集中的样本数量特别少，则就更加无法肯定曲线的走势了，因为开发样本集可能会有些干扰数据。

加入我们把训练误差率也绘制出来：



现在，你可以十分确信，添加数据并不会起到效果了。为什么呢？记住我们的两项观察结论：

1. 由于添加了更多的训练数据，训练错误率应该会有所增加，因此蓝色的曲线的高度应该不变或者变得更高。因此它会远离我们期望的水平(绿色的曲线)。

2. 红色的开发错误率通常会高于蓝色的训练错误率。因此当训练错误率高于期望水平时，在怎么增加数据，也无法让开发错误率降低到期望的水平以下。

通过查看绘制在同一图形中的开发错误率曲线和训练错误率曲线，可以让我们更加确信开发错误率的图形走势。

为方便讨论，假设期望的水平就是我们预估的最优错误率。上面的例子是个标准的“教科书式”高偏差的例子：在最大的训练样本集下，训练错误率和预期水平还有很大差距，这预示着算法有很大的可避免偏差。而且，训练错误率曲线和开发错误率曲线很接近，说明偏差很小。

之前，我们计算的训练错误率和开发错误率，只是这个图形最后端的部分，也就是我们使用所有数据的情况。绘制完整的学习曲线，可以让我们更全面地了解，算法在不同大小训练集下的性能表现。

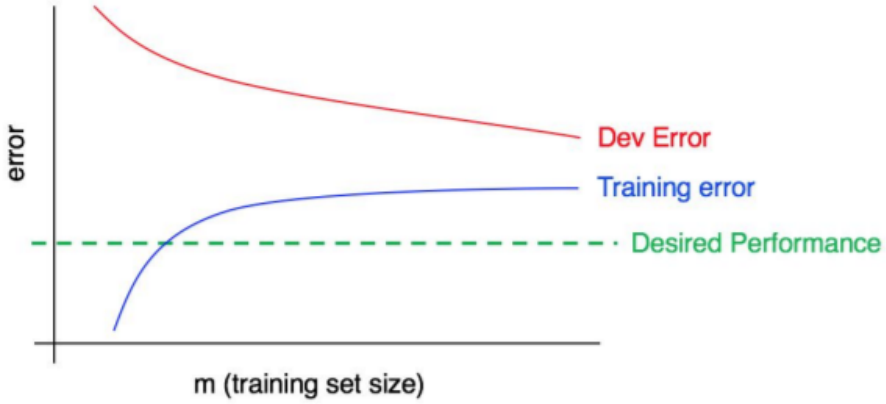
31.其他情况下的学习曲线

看看下面的学习曲线:

上面这幅图暗示模型是高偏差、高方差还是全都有呢?

蓝色的训练错误率曲线相对较低，红色的开发错误率曲线相对蓝色的开发错误率高出不少。因此，此时偏差很小，但是方差很大。添加更多的数据，可能会让开发错误率与训练错误率更加接近。

再看了下面这种情况:



训练错误率远高于预期的水平，开发错误率又远高于训练错误率。此时模型有着明显的偏差和方差。因此，需要找到一个方法，让你的算法偏差和方差同时下降。

32.学习曲线绘制技巧

假如你的训练样本集非常小，只有 100 个样本。你依次随机抽取 10 个样本、20 个样本、30 个样本，每次增加 10 个样本依次类推，一直到 100 个样本，进行模型训练，然后把学习曲线绘制出来，你可能会发现，当训练样本集很少时，曲线看起来有很多噪音。

当你只随机选择 10 个样本进行训练时，你可能会非常不幸的选中一个非常差的训练样本集，比如样本集中包含很多模糊不清或标记错误的样本，或者你非常幸运，选到了一个非常好的训练样本集，使用小的训练样本集意味着开发错误率和训练错误率会产生很大的随机波动。

如果你的机器学习应用严重偏向某一类(比如猫咪分类器任务重，负例的数量远大于正例的数量)，或者你的应用含有非常多的类别(比如识别 100 种不同的动物)，此时选出不具有代表性或糟糕数据集的可能性就会很大。比如，你的样本集中，负例(y=0)的数量达到了 80%,正例(y=1)的数量只有 20%，这是如果训练样本集中只有 10 个样本，那么这些样本很可能全是负例，这样你的算法就学不到任何有意义的东西了。

如果因为噪音，学习曲线无法看出真实走势，我们可以使用以下两个解决方案：

1.相对于在一个模型上训练 10 个样本，我们可以随机选择多个包含 10 个样本的样本集，然后训练不同的模型，之后计算每个模型的训练准确率和测试准确率，然后计算平均值，最后再把平均值绘制出来。

2.如果你的训练样本集严重偏向某一类别，或者模型要区分很多类别，你可以选择一个相对均衡的训练子集，而不是随机选择 10 个样本，比如确保训练子集中，正例占 2/10,负例占 8/10。一般情况下，我们要让子集中各个类别的比例与原始数据相同。

除非你已经尝试过绘制学习曲线，并得出结论说曲线太嘈杂，无法看到潜在的趋势，否则我建议你使用这些技术。如果你的训练样本集很大(比如超过 10,000 个样本)，而且各个类别分布相对均匀，则你可能用不到这些技术。

最后，绘制学习曲线是比较费计算资源的。比如你可能需要训练 10 个模型，那么这些模型可能需要在 1000 个样本、2000 个样本、……、10,000 个样本上做训练。在小的训练集上训练会比大的训练集快得多。当然你也可以不像上面那样，线性均匀的设置每个训练样本集大小，你可以在

1000、2000、4000、6000、10,000 个样本上做训练，这同样可以让你看清算法的走势。当然只有附加的训练成本(计算资源)可以接受时，这项技术才是有意义的。

第六章：与人类的水平进行比较

33.为什么要与人类的水平进行比较

机器学习系统的目标是让机器自动化的完成人类能做的很好的事情。比如图片识别、语音识别、垃圾邮件分类。学习算法已获得的长足进步，在很多任务上表现出的性能超过了人类。

在人类能做的很好的任务中，构建机器学系统相对容易一些，原因有以下几点：

1. 通过人工标注很容易获取数据。比如，因为人可以很好的识别猫类图片，你的算法可以获得，人类提供的高精度标记数据。

2. 人类可以凭直觉分析误差。比如语音识别算法的性能低于人类识别水平，比如一段音频被错误的识别成“This recipe calls for a pear of apples"，错误把"pair"识别成"pear"了。你可以凭直觉想想，人类怎么来修正这些错误，并把这些知识来修正你的算法。

3. 可以用人类水平来评估算法的最优错误率，并设置期望的错误率。比如，你的算法在某项任务上的错误率为 10%，而在这项任务上，人类能达到 2%的错误率，那我们就知道最优的错误率为 2%或者更低，因此可以避免的偏差最少为 8%。此时你可以试试减少偏差的相关技术。

即使第三项看上去，可能并不那么重要，但是设置一个合理的且能达到的目标，的确可以帮助一个团队，加速他们的进度。知道你的算法有很高的偏差是非常价值的，这时我们就可以有方向的选择一些工作去做了。

有些工作可能人类也不擅长。比如向你推荐一本书、或向网站用户推荐一条广告、或预测股票行情。计算机在这些工作上，已经超出了大部分人的表现。在这些应用中，我们会遇到以下问题：

1. 很难获取样本标签。人类很难标记出给用户推荐那些书最好。如果你运营一个售书的网站或 APP，你可以通过展示书籍或查看用户以前的购买书籍，来获取这些数据。如果没有运营这样的网站，那么你就得想着其他法子，来获取这些数据。

2. 人类的直觉很难靠得住，比如没人能准确的预测股票的行情，因此如果股票行情预测算法还没瞎猜的好，我们也很难找到解决的办法。

3. 很难知道最优的错误率和合理的期望值。比如你的图书推荐系统已经做得相当好了，但是你怎么知道相对人来说，还有多大的提升空间呢？

34.如何定义人类处于什么样的水平

假设你正在开发一个医学成像应用，它能自动地从 X 射线图像中做出诊断。一个没有医学背景人，通过简单训练后，能达到 15%的错误率，初级医生能达到 10%的错误率，一个老练医生可以达到 5%的错误率。一个小型的医生小组通过讨论和辩论后，能达到 2%的错误率。那我们要把哪一个定义为人类水平呢?

在这种情况下，我们将用 2%的错误率作为我们最优的错误率。你也可以将 2%错误率作为算法的期望值，如上一章所说，这么做有三个好处：

1.容易获取人类标记的数据。你可以找一个医生团队，来标记样本标签，这样你就能获得 2%错误率的样本集了。

2.可以凭直觉进行错误分析。你可以利用医生的直觉，来帮助你改进模型。

3.用人类水平来评估最优的错误率并设置模型的期望值。用 2%错误率来作为我们的目标是合理的。最优的错误率应该低于 2%，不会更高，因为医生团队可以达到 2%的错误率。

当需要标注数据时，往往不需要医生团队针对每张图片都讨论后给出标记，因为他们的时间是非常昂贵的。或许你可以找一个初级医生标记大多数的样本，然后将较难识别的样本交给成本较高的医生或医生团队。

如果你的系统当前的错误率为 40%，那么使用初级医生或者老练医生来标记数据并没多大区别。但是，如果你的系统错误率已经降到 10%了，那么将人工水平设置到 2%，会非常有助于你改进系统。

35.超越人类的水平

假如你正在做一款语音识别产品，现在你收集了很多语音数据。假如你的语音数据中有很多噪音，因此即使是人类也会有 10%的识别错误率。假如你的系统识别错误率已经降到了 8%，那你还能用第 33 章所说的三项技术来改进你的系统吗?

如果你可以定义一个数据子集，在这个子集上人类的水平显著高于你的系统，那么你仍然可以用之前所说的技术来推动你的系统快速发展。比如，假设你的系统在含噪音的音频中，识别准确率高于人类，但人类在非常快速的口语识别中，依然更为优秀。

对于口语非常快的数据子集：

1.你可以找人帮你对快速口语音频进行标记，从而获取高精度的样本数据。

2.你可以通过人类的直觉来理解为什么人类可以正确的识别快速口语，但是你的算法去不行

3.你可以将人类在快速口语上的水平，作为你算法预期的目标。

更一般的来说，即使你的算法的平均性能已经超过了人类的水平，只要能找到人类能做对，但是机器做不对的样本集，那么之前所说的技术都可以使用。。

许多重要的机器学习应用已经超过了人类水平，比如预测电影收视率、火车的送货时长、是否要批准贷款。还有一种情况是人类分类很难，机器也不能处理的很好。因此当机器已经超过人类水平的时候，算法的进展就会变慢了，当机器还在追赶人类时，算法的进展就会快的多。

第七章：训练和测试数据集的分布不一致

36.何时使用不同分布的数据集

你的猫咪图片应用中，用户已经上传了 10,000 张图片，你手工标注出哪些图片中含有猫，那些图片中没有猫。此外你还有一个更大的从网上下载而来的数据集，这个书记包含了 200,000 张图片。你应该如果定义训练/开发/测试样本集呢?

由于用户上传的 10,000 张图片，反应了真实数据的概率分布，你可能会将其用作开发/测试样本集。如果你训练的模型需要大量的数据，你可能会使用从互联网上采集的 200,000 图片用作训练。现在，你的训练样本集和开发/测试样本集的概率分布是不同的。这会对你的工作造成什么影响呢?

在没有分割数据之前，我们总共有 210,100 张图片，我们可以将他们随机打乱，并放入到训练/开发/测试样本集中。这样所有的数据来自同一分布了。但我并不推荐这么做，因为这么有会，开发/训练样本集中会有 200,000/210,000 = 95%来自互联网的图片，这些图片并不能反映实际的数据分布。记住我们所说的选择开发/测试样本集的建议：

开发/测试样本集中尽量放入能反映将来真实数据的样本

机器学习学术文献中，大多假设训练/开发/测试样本集服从统一分布。在机器学习的早期，数据是比较缺乏的。我们通常只从一个概率分布中采集数据，因此我们随机将数据分割成训练/开发/测试样本集，这种情况，假设所有数据来自同一分布是成立的。

但是在大数据时代，我们可以访问大量的数据，比如可以从互联网上下载大量的猫类图片。尽管训练数据集相对开发/测试数据集来自不同的分布，但是我们仍然会在机器学习中使用它们，因为它可给我们提供很多的信息。

针对猫咪检测例子，我们会放置 5000 张用户上传的图片到开发/测试集中，而不是直接将用户上传的 10,000 图片到开发/测试集中。我们会把用户上传剩余的 5000 张图片放到训练样本集中，这样训练样本集中就包含了 5000 张来自用户上传的数据和 200,000 张从互联网上下载来的数据，共计 205,000 张图片。我们会在后续的章节中说明，为什么这么做有效。

我们再想想第二个例子，假如你要做一个语音识系统，将街道地址转录成文本，用于语音控制的地图或导航应用。你有 20,000 用户的所说的街道样本。但是你还有 500,000 个其他样本，这些样本中，用户再讨论其他话题。你可以将 10,000 个街道语音样本放入开发/测试集中，将剩余的 10,000 个样本和其他的 500,000 个样本放入训练样本集中。

我们继续假设开发样本集中的数据和测试样本集中的数据来自同一分布。但是理解训练样本集和开发/测试样本分布不一致非常重要，因为这会给你带来一些特殊的挑战。

37.是否要将所有数据都用作训练

假如你的猫咪检测器训练样本集中包含 10,000 张用户上传的图片。这些数据来自同一分布，可以把它们用作开发/测试样本集上。你还有额外的 20,000 张图片，这些图片是从互联网上下载的，应该要把 20,000 + 10,000 = 30,000 张图片一起放到训练样本集中，用作训练吗?

在早期的机器学习算法中(手工设计的计算机视觉特征，然后用一个简单的线性模型进行分类)，把数据合并到一起，的确可能会让你的算法变得更糟。因此有些工程会警告你，不要把这 20,000 张图片放入训练集中。

但是在当今强大，灵活的机器学习算法中(比如大型的神经网络)，这些风险已经大大的降低了。你可以用大量的神经元或隐藏层来构建神经网络，你可以放心的将 20,000 张图片放到训练样本集中。增加图片后，算法的性能更可能会获得提升。这种观察基于两种数据类型都有 x->y 映射关系的事实。也就是说存在一些系统，输入来自互联网或来自用户的图片，都能可靠的预测出样本标签，甚至不知道来源的图片，也能识别的很好。

将额外的 20,000 张图片加入训练样本集，会产生如下的效果： 它为你的神经网络提供了更多像猫或不像猫的样本，这对模型来说非常有帮助，因为互联网图片和用户上传的图片，有着一些共同点。你的神经网络可以从网络图片中学习到一些知识，然后应用于用户上传的图片中。 它迫使你的神经网络增加更多的功能，以便学习互联网图片的特殊特征(比如高分辨率、不同分布的图形)，如果这些特征与 APP 上传的图片特征差别很大，则他可能会耗尽神经网络的能力，因此用于手机 APP 图片识别的能力就更少了，这是你需要真正关心的。理论上，这并不会对你的算法性能(准确率)有何坏处。

我们可以用另外一种方式描述第二项，我们引用虚构人物福尔摩斯的话：你的大脑像个楼阁，因此它的空间是有限的，他还说： “当增加知识时，你就会忘记一些其他的事情。因此一定要记住，不要用无用的知识把有用的知识挤出大脑” 。

幸运的是，如果你有足够的计算能力，那么你就可以构建足够大的网络，建造足够大的楼阁，这样上面说的问题就不足忧虑了。你有足够的算力来让机器机器学习网络图片，及用户上传的图片，这两种图片就不会竞争算力了。你算法的大脑足够大，因此你不需要担心被挤出阁楼。

但是如果你没有足够大的神经网络(或者其他高度灵活的算法)，那你需要注意，让你的训练样本集与开发/测试样本集中数据分布相匹配了。

如果你认为你的数据没啥益处，你应该把它们排除在计算之外。比如，假设你的开发/测试样本集中主要包含人、地方、地标、动物等无效图片，比如你有大量的历史文档扫描图片。


Convention
relative à la Tour Eiffel.

这些文档图片中没有包含任何像猫的图片。他们跟开发/测试样本集的分布完全不同。将这些图片作为负例没有任何意义，因为神经网络从这部分数据中，学不到任何能应用在开发/测试样本集上的知识。将这些图片放入训练样本集，只会浪费计算资源和神经网络的表示能力。

38.是否要使用不一致的数据

假如你想让机器学会预测纽约的房价。即通过房屋面积(样本特征 x),预测房屋售价(样本标签 y)。纽约的房价非常高，假设你还有一个数据库，这个数据库中的数据为密歇根的底特律的房价，底特律的房价会低很多。你是否应该将底特律的房价数据，纳入到训练样本集中呢?

提供相同的面积特征 x,在两个城市房屋售价 y 表现出来的值有很大的差别。如果你只想预测纽约的房价，那么把底特律的房价数据放入训练集，会损害算法的性能。这时，最好不要把底特律的房价数据放入训练集中。

为什么房价数据的处理方式跟之前猫咪数据的处理方式不同呢？这时因为，在猫咪图片数据中，给出一个图片 x，就可以确切的预测出样本标签 y，也就是图片中是否包含猫，这里图片 x 和样本标签 y 有一个确定的映射关系。因此可以把从网上下载的图片和手机 APP 用户上传的图片放到一起进行训练。这时，使用所有数据的好处大于的对应的坏处(消耗更多的计算资源)。与之相反，纽约的房价数据和底特律的房价数据是不一致的。给出相同的房屋面积 x，房价 y 会因为城市的不同而不同。

39.区分不同数据的权重

假如从互联网上下载了 200,000 张图片，然后有获取了用户从手机 APP 上传的 5,000 张图片，这两种数据的比例是 40:1。理论上，只要你训练最够大的神经网络，然后在 205，000 张图片上训练足够长的时间，这样就可以让你的算法，在网络图片和手机上传的图片上，都能有很好的识别效果。

但是在实践中，相对于手机上传的图片，网络图片要多出 40 倍，这意味着，相比用 5,000 图片训练模型，我们要多耗费 40 倍(或更多)的计算资源。如果你没有足够多的计算资源，则可以妥协以下，你可以给网络图片设置非常小的权重。

比如，假设你要优化的目标是平方误差(对于分类任务来说，这个不是很好的选择，但是它非常简单，且易于解释)。因此我们的学习算法会做出如下优化：

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} \left(h_\theta\left(x\right)-y\right)^2 + \sum_{(x,y)\in \text{InternetImg}} \left(h_\theta\left(x\right)-y\right)^2$$

公式中的第一项为 5000 张手机图片的损失和，第二项为网上 200,000 张图片的损失和。你可以通过额外的参数

$$\min_{\theta} \sum_{(x,y)\in \text{MobileImg}} \left(h_\theta\left(x\right)-y\right)^2 + \beta \sum_{(x,y)\in \text{InternetImg}} \left(h_\theta\left(x\right)-y\right)^2$$

如果通过让网络图片的权重降低，你不需要一个大型的网络，来让算法在两种数据类型上都表现很好。这种重新定义权重的方法，只有在额外添加的数据(网络图片)与开发/测试样本集数据分布存在很大差异时，或额外添加的数据比开发/测试集中的数据多很多的时候，才会用到。

40.从训练数据集泛化到开发数据集

假如你正在配置一个模型，这个模型中训练样本集中的数据分布与开发/测试样本不同。

比如说，训练样本集中包含网络图片和手机上传图片，而测试/开发样本集中只包含手机上传的图片。然后，此时算法表现很差，它在开发/测试样本集上的错误率远高于你的期望。造成这种现象的原因可能有以下几个：

1.它在训练样本集上表现的也很差。这时模型在训练样本集中，有高偏差的问题。

2.他可以在训练样本集上表现的很好。但是算法不能泛化到与训练样本集分布相同，但是之前没见过的数据上，这是高方差的问题。

3.模型可以很好的泛化到相同分布的数据上，但是在不同分布的开发/测试样本集上，表现的很差。我们称这个问题叫做数据不匹配，这是因为训练样本集与开发/测试样本集匹配度很差。

比如，在人类可以取得近乎完美的猫的图片识别任务，你的算法表现如下：

1.在训练样本集上的错误率为 1%

2.在与训练数据分布相同，但是之前没用过的数据上，算法的错误率为 1.5%

3.在开发数据集上的错误率为 10%

在这个例子中，你可以很明显的看出数据不匹配问题。为了处理这个问题，你需要让训练数据更加接近开发/测试数据。我们之后将会讨论与之相关的技术。

为了诊断出算法出现了上面三个问题中的哪一个，你需要再使用一个数据集。因此，相比把所有的训练数据都用作训练，你应该把训练数据分成两部分：一部分用作算法的训练，另一部分称作训练开发集，这部分数据不参与训练。

你现在有四个数据子集：

1. 训练数据集：算法用于学习训练的数据(比如网络图片+用户上传的图片)。这部分数据与我们关心数据(开发/测试样本集)的分布不同。

2. 训练开发集：这部分数据与训练数据集的分布相同(比如网络图片+用户上传图片)，他通常会比训练样本集小，因为它只要能评估和追踪算法的进度就可以了。

3. 开发数据集：这部分数据与测试数据集分布相同，反映了我们真是关心的数据分布(比如用户上传的图片数据分布)

4. 测试数据集：这部分数据与开发数据集分布相同。

拥有这四个数据集，你可以完成以下评估：

1.通过训练样本集评估训练错误率

2.通过训练开发集，评估算法在同一分布数据上的泛化能力

3.通过开发/测试样本集，评估算法在你真正关心数据上的表现

5-7 章所说的开发测试大小的选择方法，也同样适用于训练开发集大小的选择。

41.区分偏差、方差和数据不匹配问题

假设人类可以在猫咪识别的任务中达到近乎完美的性能（约等于 0%的错误率），那么，最优的错误率就约为 0%。假如系统性能表现是：

1.训练集上的错误率为 1%

2.训练开发集上的错误率为 5%

3.开发集上的错误率为 5%

这种情况说明了什么？你可以看到，方差很高。之前讲到的降低方差的方法可以帮助你处理当前的情况。

现在，再假设你的算法性能如下：

1.训练集上的错误率为 10%

2.训练开发集上的错误率为 11%

3.开发集上的错误率为 12%

由此可以看出，你的算法在训练集上的可避免偏差很高，也就是说算法在训练集上表现很差。这时偏差处理相关的方法可以对你有帮助。

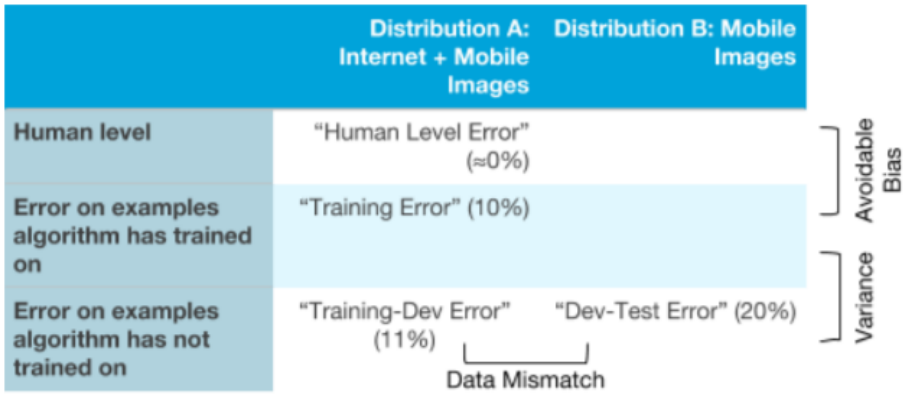
在前面的两个例子中，算法产生了很高的可避免偏差或是很高的方差。算法还有可能同时面对，高可避免偏差、高方差和数据不匹配当中的多个问题。例如：

1.训练集上的错误率为 10%

2.训练开发集上的错误率为 11%

3.开发集上的错误率为 20%

算法的可避免偏差很高，并且也有数据不匹配的问题。但是，算法在训练集上的方差并不高。为了更清晰理解不同类型的错误之间的相关性，我们把它们写在表格中：



继续之前的猫咪图像检测器例子，你可以看到，在 X 轴上有两种不同的数据分布。在 Y 轴上，有 3 种不同类型的错误率：人类错误率、算法在已经训练过的样本上的错误率、算法在从未训练过的样本上的错误率。我们可以在表格中第一列填上这三种不同类型的错误率。

如果你愿意,你也可以补充表格中的后两列的内容：你可以找些人对手机上的猫咪图片进行标记，并且评估他们的错误，然后来填写右上角的表格。

你可以按照分布 B 选取一些手机上的猫咪图片，并且把它们放在训练集中，让神经网络来学习。然后你来评估在这个数据子集上，学习模型的错误率。

填写表中的这些条目，可以给你直观的感受到算法在两种不同的数据分布（分布 A 和分布 B）上的表现。

通过了解你的算法所面临的问题，你可以更好的决策下一步应该专注于减少偏差，方差还是减少数据不匹配的问题。

42.处理数据不匹配问题

假设你开发了一款语音识别系统，在训练集和训练开发集上表现很好。然而，系统在开发集上表现很差：这是一个数据不匹配的问题。你会怎么做?

我的建议：

1.了解训练集和开发集上数据分布的不同点。

2.找到更多和你开发集分布一致的数据。

举例来说，假设你在语音识别的开发集上进行误差分析：你手动检查了 100 个样本，然后试着分析算法出错的原因。你发现你的算法表现不好，是因为开发集上的大部分的语音片段是在车里录制的，而训练集上的语音样本是在背景安静的环境下录制的。汽车发动机还有马路上的噪音严重影响了你的语音系统的识别性能。这种情况下，你可能需要试着获取更多在车上录制的语音片段，并加入训练集。误差分析的目的是了解训练集和开发集之间主要的区别，弄清楚导致数据不匹配的主要问题。

如果你的训练集和训练开发集包括那些在车上录制的音频样本，你也应该在这部分的数据子集上双重地检查你的系统性能。如果在训练集上，算法对车内录制的音频样本数据表现很好，但是在训练开发集上，对车内录制的样本表现很差，这就验证了一个假设，获取更多在车上录制的音频样本会很有帮助。这就是为什么我们会在之前的章节中讨论，训练集上要包含一些样本数据，和开发/测试集数据的数据分布相同。这样做还可以让你对比在训练集以及开发/测试集上，关于车内音频数据上性能的不同。

不幸的是，这个过程是无法保证的。比如，如果你怎么都找不到可以开发集数据相匹配的测试数据，那么，你想要提升性能就没有十分清晰有效的方法。

43.人工数据合成

你的语音识别系统需要更多听起来在车里录制的语音数据。相对于收集那些在开车的时候录制的音频，有一个更简单的方法获取这些数据：通过人工合成数据。

假设你已经获得足够的汽车/马路噪音的音频片段。你可以从多个网站上下载这些数据。如果你已经有很多安静环境下录制的训练样本。你可以选出其中一个音频片段，然后给它加上背景噪音，这样你就获得了一个听起来好像在嘈杂的车上录制音频。

用这种方法，你可以合成大量的数据，就好像是收集了大量在车里录制的语音。

更广泛的说，在一些应用场景中，人工数据合成让你可以创造一个庞大的数据集来充分的匹配开发样本集。用猫咪检测器作为第二个例子。你会意识到开发样本集中很多会有运动模糊，这是因为手机用户在照相的时候都会出现轻微的抖动。

你可以从网上找到没有模糊的图片组成训练集，然后加上一些模拟的运动模糊。

要记得人工数据合成也有它的挑战：有时候，合成的数据在人看来很真实，但是在机器看来就没有那么真实了。例如，假设你有 1000 个小时的语音训练数据，但是只有 1 个小时的汽车噪音数据。在合成数据的时候，如果你在训练样本集中 1000 个小时的原始数据中，总是重复的使用这 1 个小时的汽车噪音，最后你就得到一个有很多重复噪音的合成数据集。对于人来说可能并不能听出有失真的情况，毕竟所有的汽车噪音对我们来说都没有太大的区别，但是机器学习算法却可能因为这重复的 1 个小时的汽车噪音变得过拟合。那么，算法就很难泛化到其他新的有背景噪音的音频片段上，当汽车噪音和原来不同的时候，算法就会遇到识别的麻烦。

另一种相似的情况，假设你有 1000 个小时的汽车噪音，但是这些音频只来自 10 种不同的汽车。这种情况下，算法很可能会对这 10 种汽车过拟合，可能在处理其他类型车子的声音时性能表现的很差。不幸的是，这些问题很难会被注意到。

再举一个例子，假设你正在建造一个计算机视觉系统来识别车子。假设你有一个伙伴在视频游戏公司，有几种车子图像模型。为了训练你的算法，你会用这些模型来生成车子的合成图像。即使这些合成的图片看起来非常真实，这个方法很能还是不能很好的工作。所有视频游戏的场景中可能只用到 20 种左右的车子的设计，如果你在玩游戏，你很可能根本不会注意到你看到的都是重复的车子，很多只是车子颜色不同。也就是说，这些车子在你看来都很真实。但是和马路上真实在开动的车子相比，也就是在开发/测试集中的样本相比，这些 20 种型号的车子只是世界上车子类型里很小很小的一部分。如果你用 20 种型号的车子来合成了 100000 个测试样本，你的系统会对这 20 种车子过拟合，系统会无法泛化，也就无法在开发/测试样本集中识别出其他车子的设计。

在合成数据的时候，要有意识的提醒自己是不是真的合成出了有代表性的样本。尽量避免使用那些能够被学习算法识别出人工合成的痕迹，比如所有数据都是从 20 种车子合成而来，或是所有的音频样本都是仅由 1 个小时的汽车噪音而来。这个建议实际上很难遵守。当合成数据时，我们的团队有时会在产生数据之前花上好几个星期在一些重要的细节上，来保证合成的数据可以足够接近真实的数据分布。如果你想正确的获知这些细节，你必须提前就接触非常大量的训练样本集。

轻松一刻

看书累了没，敢不敢跟机器赌上一吧(赌输了要分享本站哦)：

让机器学会打游戏之决胜八点

http://www.insideai.cn/ai3_poker/game3_poker.html(Ctrl+鼠标 点击进入)

第八章：算法调试

44.优化验证测试

假设你正在建立一个语音识别系统。你的系统需要输入一个音频片段 A，并且输出每个可能句子的得分 $Score_A(S)$。例如，你可能会尝试 $Score_A(S) = P(S|A)$，其中， A 是输入的音频片段，P(S|A)是在正确出的语句中 S 的概率。

给出了一种计算 $Score_A(S)$的方法，你仍然需要找到英语句子S 并把它最大化：

$$\text{Output} = \arg\max_S \text{Score}_A(S)$$

你应该怎样计算这里提到的“最大的自变量”呢？假设英语里有 50000 个单词，长度为 N 的句子有(50,000)N 种可能，不胜枚举。所以你需要使用一种近似的搜索算法，找到分值最高的 S。

举例来说，有一种搜索算法“束搜索”，只保留搜索过程中前 K 个选择。（在本章，你不需要了解关于束搜索更多的细节。）像这样的算法不能保证可以找到分值最高的 S。

假设有一个音频片段 A 记录了某人在说“我爱机器学习”。但是系统并没有给出正确的输出，而是输出了“我爱机器人”。出现这样的错误有两种可能：

1. 搜索算法的问题。近似的搜索算法（比如束搜索）无法找到分值最高的 S。

2. 客观性问题（比如积分函数）。我们使用$Score_A(S) = P(S|A)$来计算分数的方法不准确的。特别是 $Score_A(S)$ 没有把“我爱机器学习”识别为正确的语句。

不同类型的问题对应的策略和努力的方向不同。如果当前面对的是第 1 类错误，你应该关心搜索算法的提升。如果是第 2 类问题，你应该关注评估 $Score_A(S)$ 的学习算法。

面对这种情况，有些研究者会随机的选择搜索算法；还有一些研究者会随机寻找更好的$Score_A(S)$的学习方法。但是除非你知道问题的原因到底是哪一个，不然就是白费力气。你如何才能更加系统的知道要做什么呢？

假设 S_{out} 表示实际输出的文本（“I love robots”）。S^*表示正确的文本（“I love machine learning”）。为了确定出现的是上述中哪一类问题（搜索算法的问题或客观性问题），你可以使用**最优化验证测试**：首先，计算 $Score_A(S^*)$和 $Score_A(S_{out})$。然后检查 $Score_A(S^*)$和 $Score_A(S_{out})$的大小。有两种情况：

Case 1：$Score_A(S^*) > Score_A(S_{out})$

这种情况，你的学习算法可以保证，正确的文本 S^*的分数高于实际输出文本 S_{out} 的分数。只是，我们的最优的搜索算法选择了 S_{out} 而不是 S^*。这种情况就说明，你的最优的搜索算法没有选择分数 $Score_A(S)$最高的 S。这样，最优化验证测试就告诉你，你的搜索算法有问题，你需要专注解决这个问题。比如，可以尝试增加束搜索中束的宽度。

Case 2：$Score_A(S^*) \leq Score_A(S_{out})$

这种情况下，可以知道，你用来计算分数的 $Score_A(.)$方法是错误的：它不能严格保证，正确的文本所得的分数 S^*必须高于当前输出文本的分数 S_{out}。最优化验证测试告诉你，这是一个客观性问题（计分函数的问题）。这样你就应该优化你的计分函数 $Score_A(.)$。

目前我们的讨论集中在单个样本的分析上。为了在实际中应用最优化验证测试，你应该检查开发样本集上的错误。针对每一个错误，对比 $Score_A(S^*)$ 和 $Score_A(S_{out})$的值。对于开发样本集上的错误样本，如果，$Score_A(S^*) > Score_A(S_{out})$，说明是优化算法的问题。否则（$Score_A(S^*) \leq Score_A(S_{out})$），就说明是计分函数 $Score_A(.)$的问题。

例如，假设，你发现 95%的错误都是因为打分方法 $Score_A(S)$的问题，只有 5%的错误是由于优化算法导致的。现在，你知道不管你如何改进你的优化程序。你实际上只能消除约 5%的错误。那么，你应该专注在提升计分函数 $Score_A(S)$上面。

45.优化验证测试的一般形式

提供一些输入 x，当你知道如何通过计算分值，来衡量输出 y 相对输入 x 有多好时，你就可以使用优化验证测试了。

比如，我们之前说的语音识别例子中，x 是一个语音片段，y 是输出的识别内容，y*是正确的结果。 那我们的关键测试就是判断 $Score_x(y^*) > Score_x(y)$，如果这个不等式成立，那就说明我们的最优算法有问题。 可以参考上一章，来理解这背后的逻辑。否则，我们认为得分值计算有问题。

我们再看一个例子。你在构建一个汉英翻译系统。 输入汉语句子 C 到你的系统中，会输出各种可能英文句子 E，每个句子的得分值用 $Socre_c(E)$来表示， 得分值可以用汉语句子 C 翻译成英语句子 E 的概率来计算：Socrec(E) = P(E|C)。

则通过得分值可以获得最终的输出结果：

$$\text{Output} = \arg \max_{E} \text{Score}_C(E)$$

然而，可选的句子数量太多了，因此你需要使用启发式搜索算法。假设你的算法输出了错误的翻译 E_{out}，而不是正确的结果 E*。 此时优化验证测试会让你计算并判断 $Socre_c(E^*) > Socre_c(E_{out})$， 如果不等式成立，则 $Socre_c()$的计算是正确的，因为 E* 的确优于 E_{out}， 因此这时的问题出现在近似的搜索算法上，否则问题出现在 $Socre_c()$方法上。

在人工智能中，有一个非常常见的设计模式。首先找到一个近似的得分值计算函数 $Socre_c()$，然后逐渐靠近最优算法。如果能恰当的使用这个模式，通过优化验证测试，你将很容易理解错误的根源。

46.强化学习实例

假如你正在使用机器学习教直升飞机，执行复杂的飞行任务。这儿有一张直升飞机引擎关闭时，着陆时拍摄的照片。

在引擎意外关闭的情况下，让直升飞机平稳着陆，这就是所谓的“autorotation”策略。 这种训练是人类飞行员训练的一部分。你的目标是，让机器学习算法指导直升飞机通过轨道 T，并最终安全着陆。

为了应用强化学习，你需要设计一个"奖励方法"R()，这个方法来计算每种可能的轨道 T 的得分值。 比如如果轨道 T，最终导致直升飞机坠毁，此时的奖励 R(T) = -1000，是一个非常大的负数。 如果直升飞机安全着陆，则 R(T)为正数，它的大小取决于着陆轨道是否光滑平稳。奖励函数 R()通常由人手工设计，用来计算轨道 T 有多好。 它需要平衡着陆点有多崎岖（直升飞机是否落到了期望的着陆点）以及对于乘客来说，下降过程的的舒适度，设计好的奖励函数并不简单。

设计好奖励函数 R(T)后，强化学习算法就会控制直升分级取得 $\max_T R(T)$,然而强化学习算法有很多的近似值，因此可能取不到最大值。

假设你选择了一些奖励函数 R(T),并开始运行你的学习算法。 然而，你的算法性能可能远差于人类飞行员的水平，下降过程非常颠簸，而且也没有人类操作安全。 那如何区分这个问题是因为

学习算法(通过学习算法来取得 $\max_T R(T)$) 导致的还是由于奖励函数(通过奖励函数来衡量下降舒适度和降落准确率的得分)导致的呢？ 为了使用优化验证测试，将 T_{human} 定义为人类飞行员飞行的轨道，将 T_{out} 定义为算法得出的飞行轨道。 根据我们之前的描述，T_{human} 优于 T_{out}。因此关键测试如下：

$$\mathbf{R(T_{human}) > R(T_{out})}$$

1.如果不等式成立，则奖励函数 R()的计算是正确的。T_{human} 优于 T_{out}，这时我们的枪火学习算法就是有问题的，因此改进学习算法是非常有意义的。

2.如果不等式不成立 $R(T_{human}) < R(T_{out})$，这就意味着奖励函数 R()是有问题的，你应该修改奖励函数。

很多机器学习应用使用近似搜索算法的模式，来优化近似得分函数 $Score_x()$。有时没有特定的输入 x，因此这种优化方法只适用于得分函数 Score()。 上面的例子中，得分函数就是奖励函数：Score(T) = R(T)，优化算法就是强化学习算法试着去执行的合理飞行轨道 T。

这个例子跟之前例子的一点不同是，这例子中比较的对象不是最优的输出，而是人类的水平 T_{human}。 我们假设人类的水平即使不是最优的，也是依然非常好的。 一般情况下，只要你能找到最优 y*(本例中为 T_{human})，且你的算法比最优的结果差， 那么优化验证测试就能帮你改进优化算法或得分函数。

第九章：端到端深度学习

47.端到端学习的兴起

假如你正在构建一个机器学习系统，这个系统可以检查在线的产品评论，然后告诉你评论者是否喜欢对应的产品。比如，你希望将下面的例子识别成正例：

这个拖把非常棒

或者把下面这句识别成负例：

这个拖把质量非常差，买了非常后悔。

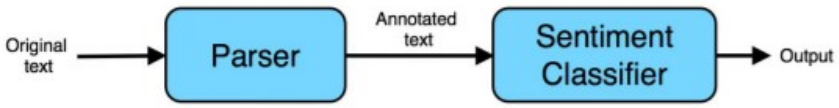
这种识别正例/负例的问题称作情感分类。

为了构建这样一个系统，你可能需要开发两个组件：

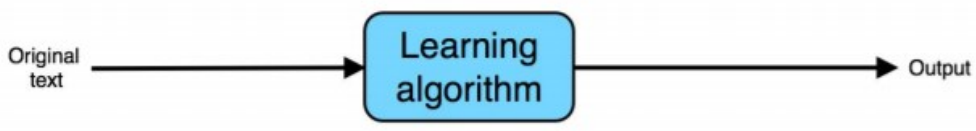
1.文本解析器： 你的系统给文本加上注释，来找到重要的单词。比如你可能会使用解析器来标记出所有的形容词和名次。因此通过解析器你可能会到的下面这个句子： 这个**拖把**名词**非常棒**形容词

2.情感分类器：通过机器学习算法，使用标记好的文本，预测出文本对应的感情。解析器注释后的文本，非常有利于算法的学习：通过给形容词加上更高的权重， 比如“非常棒”，忽略不重要的单词，比如“这个”。

我们可以把两个组件组成的流水线绘制出来：



最近有个趋势就是把流水线式的系统变成一个单一的学习算法。端到端学习算法就是用来处理这种任务的。只要提供原始的输入，比如原始文本"这个拖把非常棒！"，然后直接进行情感识别：

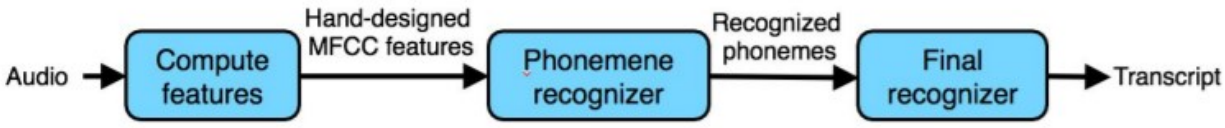


神经网络经常用于端到端的学习系统。端到端这个词语的意思是通过学习算法，直接将原始输入转化成最终输出。也就是说使用学习算法将输入端和输出端链接起来。

在数据大量的问题中，端到端系统取得了非常显著的成功。但是它们也不是在任何情况下，都是最好的选择。下一章，我们将给出更多端到端的例子，并给出一些建议，说明那些问题适合使用端到端系统，哪些问题不适合使用端到端系统。

48.更多的端到端学习实例

假如你正在构建一个语音识别系统，这个系统中，你需要开发三个组件：



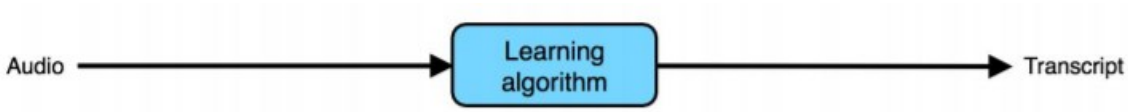
这三个组件负责的工作如下：

1.计算特征：抽取手工设计的特征，比如 MFCC(梅尔频率倒谱系数)特征，采集有用的内容，忽略不相干的属性，比如说话者的音高。

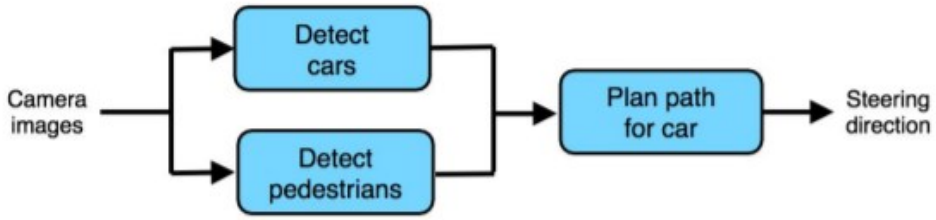
2.音素识别：一些语言学家任务，声音的基本单元是音素。比如单词"Keep"中的"K"与单词“cake”中的“c”发音相同，它们是同一个音素。

3.最终识别：将识别出的音素序列，组合起来形成最终的输出文本。

与上面这个结构相反，端到端系统输入语音片段后，可以直接输出对应的转录文本：



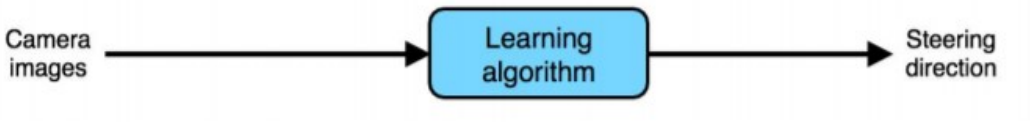
到目前为止，我们所说的机器学习算法流程都是线性线性的： 输出序列从一个阶段传递到下一阶段。算法的流程也可能会更复杂，比如，自动驾驶。 下面是自动驾驶算法的简单结构图：



它包含三个组件：一个通过照相机图片检测其他汽车，一个用于检测行人，最后一个组件用于给当前车辆规划路线，避免装上其他车辆或行人。

流程中并不是每一个组件都需要学习的。比如机器人运动规划的文献中，有很多算法都是用于路径规划的，但是但部分算法都不涉及到学习。

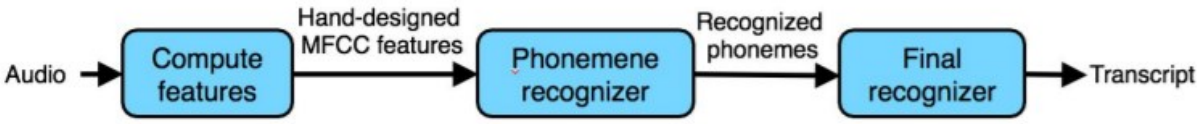
相反，端到端方法中可以依据传感器的输入，直接输出导航方向：



尽管端到端学习在很多领域取得了成功。但也不是在所有的场景中，他都是最好的方法。 比如端到端系统可以很好的完成语音识别任务，但是我对端到端能否完成自动驾驶任务，表示怀疑。下面几章，我们解释其中的原因。

49.端到端学习的利与弊

回想我们之前说的语音识别流程：



流程中很多内容都是"手工工程"：

1.MFCCs 是一系列手工设计的音频特征。这个特征提取过程会丢弃一些无用的信息，并整理出合理的音频摘要信息。

2.音素是语音学家发明的。它是音频的一种不完美表示方法，用音素来表示音频会限制语音识别系统的性能。

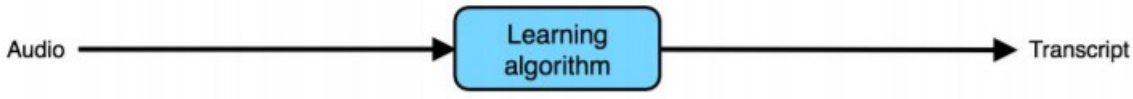
然而，手动工程组件也有它的优势：

1.MFCC 特征对那些不影响语音内容的因素，比如音高，具有很好的鲁棒性，这可以有效的简化机器学习算法要解决的问题。

2.音素也是音频的一种合理表示。可以让算法了解基本的语音单元，然后帮助改进算法。

使用更多的手工工程组件，可以减少学习算法所需要的数据量。手工工程从 MFCCs 和因素获得的知识，从而可以补充算法从数据中获得的知识。当我们没有很多数据时，这些知识就非常有用了。

现在，再考虑下端到端系统：



这个系统缺少手工工程相关的知识，因此当训练数据集很小时，他可能表现出的性能差于手工工程。

然而，当数据量比较大时，学习算法就不会受到 MFCC 或基于音素表示方法的限制了，如果学习算法是一个足够大的神经网络，而且有足够多的数据参与训练，它的潜力是非常大的，或许能达到最优的错误率。

端到端学习系统可以在两端(输入端、输出端)有大量已标记数据的情况下，表现的非常好。 在这个例子中，我们需要准备大量的音频和对应的输出文本。但当数据不可用时，端到端学习方法就有很大的风险了。

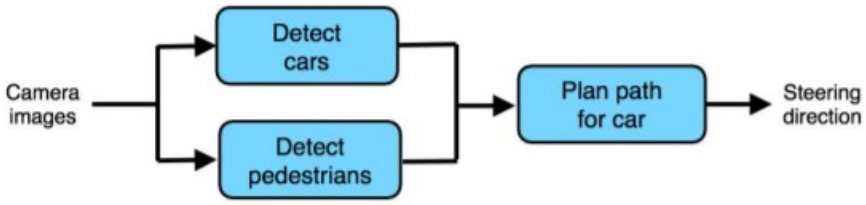
当你用机器学习算法解决问题时，如果训练样本集非常小，算法的大部分知识需要来自于人类的观察，因此需要手工工程设计的组件。

当你不使用端到端系统时，你需要考虑算法流程分成哪几步，这几步如何组装在一起，下面我们将会提供一些关于算法流程的设计建议。

50.依据数据可用性选择组件

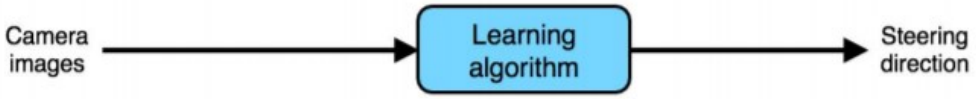
当构建一个非端到端系统时，算法流程中，应该如何选择和组装组件呢？如何设计算法流程会极大的影响系统的性能。 选择组件需要考虑的一个因素是，组件需要的数据能否轻易采集到。

比如，考虑下面这个自动驾驶的例子：



你可以使用机器学习算法检测其他车辆和行人。这种情况下采集数据并不难：有大量的计算机数据库中，包含了标记好的汽车和行人。你也可以使用众包(比如亚马逊的 Mechanical Turk)来获取更多的数据集。因此很容易获取训练数据，来构建汽车检测器和行人检测器。

相反，在纯端到端方法中：



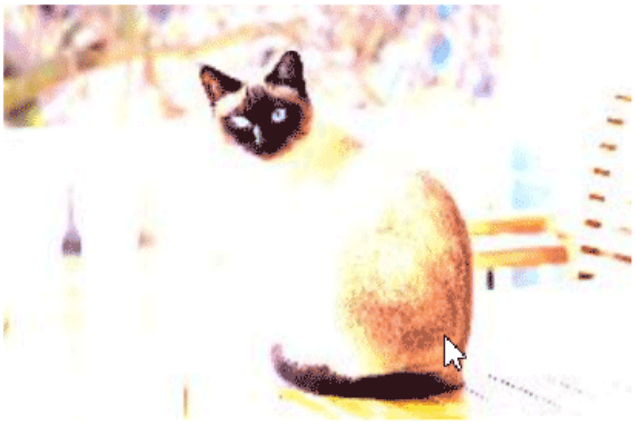
你需要大量的图片和转向方向对应的数据集。让人到处开车并记录转向数据，是非常耗时的，而且成本非常高。 你需要组建一个装有特殊装备的车队，并让他们到处驾驶，以便覆盖各种可能的场景。这使得端到端系统变得难以训练。相反，采集大量标记好的汽车或行人图片，相对简单的多。

更一般的说，如果有大量的数据用来训练流程的中间模块(比如汽车检测器、行人检测器)，那么你可以使用一个多阶段的流程。这种结构可能更好一些，这样你就能使用所有的数据来训练中间模块了。

在没有更多的端到端数据可用之前，我相信在自动驾驶问题上，非端到端的方法还是非常有前景的。它的结构与可用的数据更匹配。

51.依据难易程度选择组件

除了数据可用性以外，考虑流程中组件时，你还需要考虑一个因素：独立组件解决问题的简单程度。 你应该试着选择那些容易构建和学习的组件。那么对于组件来说，什么叫做易于学习呢?



看看下面几个机器学习任务，按难度递增的顺序排列出来：

1.判断图片是否曝光过度

2.判断图片是在室内拍摄的还是在室外拍摄的

3.判断图片中是否含有猫

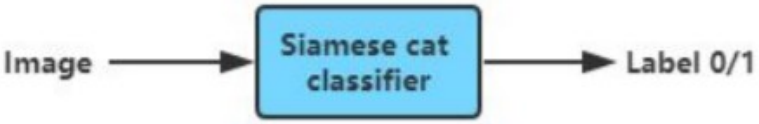
4.判断图片中的猫是否有黑色和白色的毛

5.判断图像是否包含暹罗猫（一种特殊的猫）

上面这些都是图形二分类任务：输入一幅图像，输出 0 或 1。但是列表中前几个，对于神经网络来说相对容易学习。你可以通过很少的样本集就可以完成到前几个任务。机器学习中，还没有一个很好的正式定义，什么样的任务难，什么样的任务简单。随着机器学习和神经网络的兴起，我们说，如果训练的步数很少(或神经网络很浅)，任务就可以完成，那么我们称任务简单。如果训练需要更多的步数(或更深的网络)，则我们说任务很难。但是这些都是非正式的定义。

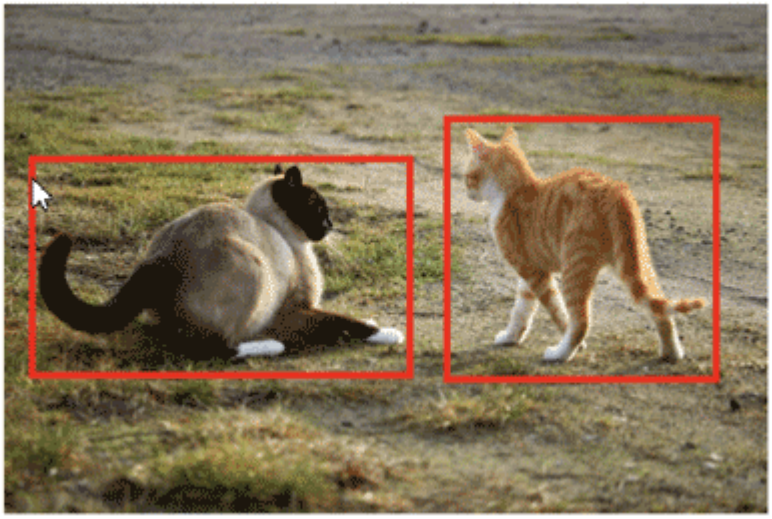
如果你能将一个复杂的任务，拆分成几个简单的子任务，然依据简单的子任务进行编码，通过给你的算法提供先验知识，可以帮助你的算法更有效的学习这个任务。

假如你正在构建一个暹罗猫检测器，下面是一个端到端的结构：

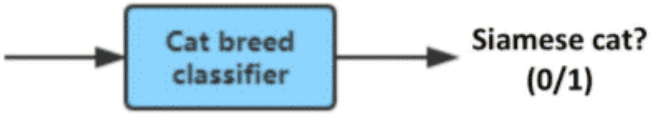


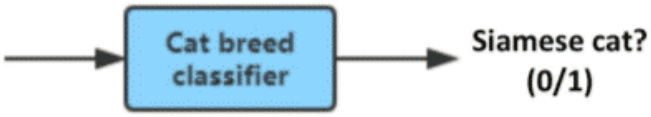
你也可以使用两步来完成这个任务：

第一步先检测图片中的所有猫。



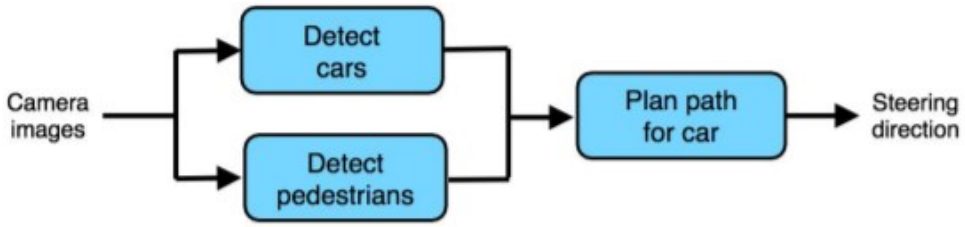
第二步，将每只猫裁剪出来，然后通过特殊猫咪检测器，判断是否有暹罗猫，如果有暹罗猫，则输出 1。





相对于纯端到端系统，只是用 0/1 标签，使用两个组件：猫咪检测器和特殊猫咪分类器，似乎更容易学习且需要的数据量更少。

最后一个例子，让我们回到自动驾驶流程中：



通过使用这个流程，你告诉算法执行三个关键步骤来实现自动驾驶：

1.检测其他汽车

2.检测路上行人

3.行车路线规划

这三个步骤中，每一步都相对简单，相对纯端到端系统，所需的数据量也更少。

总的来说，当你决定流程采用哪些组件时，尽量选择相对简单的组件，这样可以使用更少的数据，就可以学习了。

52.让机器学习输出更加丰富的内容

图片分类算法中，输入图片，然后输出一个数字，代表这张图片所属的类别。算法能输出一句完整的话来描述图片吗？ 比如：

x = 

y = "A yellow bus driving down a road with green trees and green grass in the background."

传统的监督类机器学习，需学习到的内容都是 h:X→Y，通常 y 都是一个整数或者实数。比如：

Problem	X	Y
Spam classification	Email	Spam/Not spam (0/1)
Image recognition	Image	Integer label
Housing price prediction	Features of house	Price in dollars
Product recommendation	Product & user features	Chance of purchase

端到端系统一个最令人激动的进展是，它可以输出更加复杂的信息。在上面图片描述的例子中，你可以使用一个神经网络，输入图形(x)，直接输出图形的描述信息(y)。

还有更多的例子:

Problem	X	Y	Example Citation
Image captioning	Image	Text	Mao et al., 2014
Machine translation	English text	French text	Suskever et al., 2014
Question answering	(Text,Question) pair	Answer text	Bordes et al., 2015
Speech recognition	Audio	Transcription	Hannun et al., 2015
TTS	Text features	Audio	van der Oord et al., 2016

在机器学习中，这个趋势正在加速发展：当你有上图右侧的(输入、输出)标记数据时，你可以使用端到端系统，让模型输出比数字更复杂的句子、图片或者音频或者其他内容。

完结

感谢您的阅读，关注公众号：袋马 AI，获取更多最新资料。