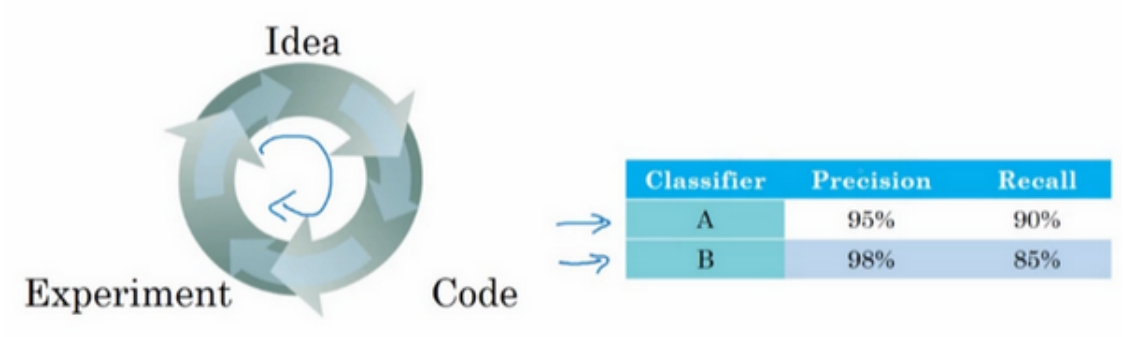
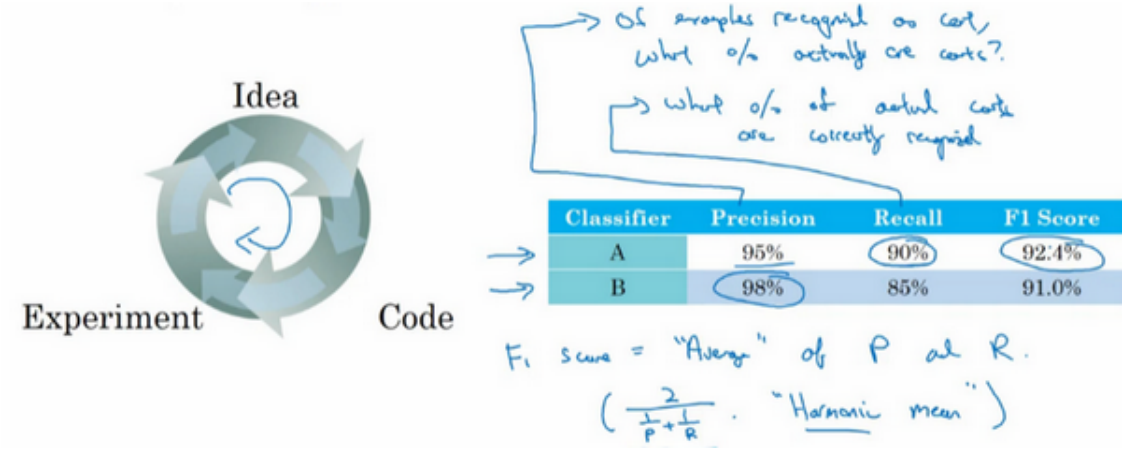
1.3 单一数字评估指标（Single number evaluation metric）

无论你是调整超参数，或者是尝试不同的学习算法，或者在搭建机器学习系统时尝试不同手段，你会发现，如果你有一个单实数评估指标，你的进展会快得多，它可以快速告诉你，新尝试的手段比之前的手段好还是差。所以当团队开始进行机器学习项目时，我经常推荐他们为问题设置一个单实数评估指标。



我们来看一个例子，你之前听过我说过，应用机器学习是一个非常经验性的过程， 我们通常有一个想法，编程序，跑实验，看看效果如何，然后使用这些实验结果来改善你的想法，然后继续走这个循环，不断改进你的算法。

比如说对于你的猫分类器，之前你搭建了某个分类器A，通过改变超参数，还有改变训练集等手段，你现在训练出来了一个新的分类器B，所以评估你的分类器的一个合理方式是观察它的查准率（precision）和查全率（recall）。



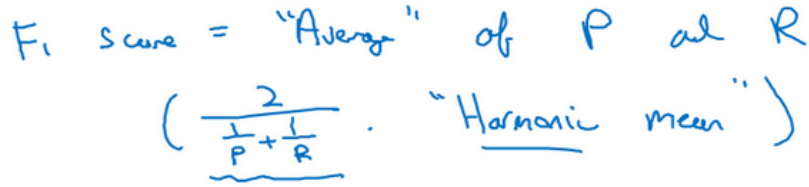
查准率和查全率的确切细节对于这个例子来说不太重要。但简而言之，查准率的定义是在你的分类器标记为猫的例子中，有多少真的是猫。

所以如果分类器A有 95%的**查准率**，这意味着你的分类器说这图有猫的时候，有 95%的机会真的是猫。

查全率就是，对于所有真猫的图片，你的分类器正确识别出了多少百分比。实际为猫的图片中，有多少被系统识别出来？如果分类器A查全率是 90%，这意味着对于所有的图像，比如说你的开发集都是真的猫图，分类器A准确地分辨出了其中的 90%。

所以关于查准率和查全率的定义，不用想太多。事实证明，查准率和查全率之间往往需要折衷，两个指标都要顾及到。你希望得到的效果是，当你的分类器说某个东西是猫的时候，有很大的机会它真的是一只猫，但对于所有是猫的图片，你也希望系统能够将大部分分类为猫，所以用查准率和查全率来评估分类器是比较合理的。

但使用查准率和查全率作为评估指标的时候，有个问题，如果分类器A在查全率上表现更好，分类器B在查准率上表现更好，你就无法判断哪个分类器更好。如果你尝试了很多不同想法，很多不同的超参数，你希望能够快速试验不仅仅是两个分类器，也许是十几个分类器，快速选出“最好的”那个，这样你可以从那里出发再迭代。如果有两个评估指标，就很难去快速地二中选一或者十中选一，所以我并不推荐使用两个评估指标，查准率和查全率来选择一个分类器。你只需要找到一个新的评估指标，能够结合查准率和查全率。



在机器学习文献中，结合查准率和查全率的标准方法是所谓的F_1分数，F_1分数的细节并不重要。但非正式的，你可以认为这是查准率P和查全率R的平均值。

F_1分数的定义是这个公式：$\frac{2}{\frac{1}{P}+\frac{1}{R}}$

在数学中，这个函数叫做查准率P和查全率R的调和平均数。但非正式来说，你可以将它看成是某种查准率和查全率的平均值，只不过你算的不是直接的算术平均，而是用这个公式定义的调和平均。这个指标在权衡查准率和查全率时有一些优势。

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

但在这个例子中，你可以马上看出，分类器A的F_1分数更高。假设F_1分数是结合查准率和查全率的合理方式，你可以快速选出分类器A，淘汰分类器B。

Dev set + Single number evaluation metric
real speed up iterating

我发现很多机器学习团队就是这样，有一个定义明确的开发集用来测量查准率和查全率，再加上这样一个单一数值评估指标，有时我叫单实数评估指标，能让你快速判断分类器A或者分类器B更好。所以有这样一个开发集，加上单实数评估指标，你的迭代速度肯定会很快，它可以加速改进您的机器学习算法的迭代过程。

Algorithm	US	China	India	Other
A	3%	7%	5%	9%
B	5%	6%	5%	10%
C	2%	3%	4%	5%
D	5%	8%	7%	2%
E	4%	5%	2%	4%
F	7%	11%	8%	12%

我们来看另一个例子，假设你在开发一个猫应用来服务四个地理大区的爱猫人士，美国、中国、印度还有世界其他地区。我们假设你的两个分类器在来自四个地理大区的数据中得到了不同的错误率，比如算法A在美国用户上传的图片中达到了3%错误率，等等。

所以跟踪一下，你的分类器在不同市场和地理大区中的表现应该是有用的，但是通过跟踪四个数字，很难扫一眼这些数值就快速判断算法A或算法B哪个更好。如果你测试很多不同的分类器，那么看着那么多数字，然后快速选一个最优是很难的。所以在这个例子中，我建议，除了跟踪分类器在四个不同的地理大区的表现，也要算算平均值。假设平均表现是一个合理的单实数评估指标，通过计算平均值，你就可以快速判断。

Algorithm	US	China	India	Other	Average
A	3%	7%	5%	9%	6%
B	5%	6%	5%	10%	6.5%
C	2%	3%	4%	5%	3.5%
D	5%	8%	7%	2%	5.25%
E	4%	5%	2%	4%	3.75%
F	7%	11%	8%	12%	9.5%

看起来算法C的平均错误率最低，然后你可以继续用那个算法。你必须选择一个算法，然后不断迭代，所以你的机器学习的工作流程往往是你有一个想法，你尝试实现它，看看这

个想法好不好。

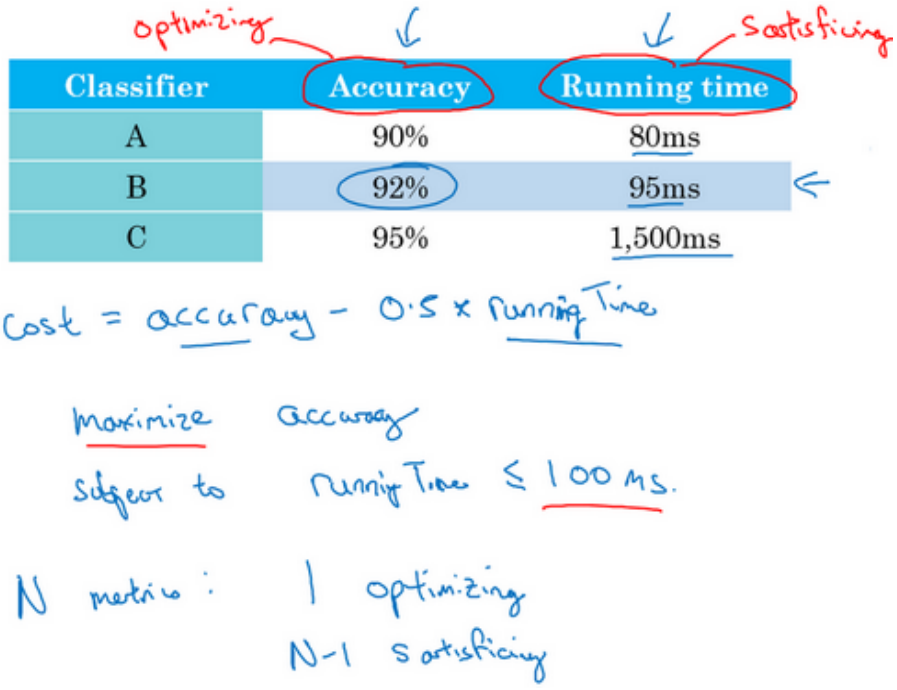
所以本视频介绍的是，有一个单实数评估指标真的可以提高你的效率，或者提高你的团队做出这些决策的效率。现在我们还没有完整讨论如何有效地建立评估指标。在下一个视频中，我会教你们如何设置优化以及满足指标，我们来看下一段视频。

1.4 满足和优化指标（Satisficing and optimizing metrics）

要把你顾及到的所有事情组合成单实数评估指标有时并不容易，在那些情况里，我发现有时候**设立满足和优化指标**是很重要的，让我告诉你是什么意思吧。

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

假设你已经决定你很看重猫分类器的分类准确度，这可以是F_1分数或者用其他衡量准确度的指标。但除了准确度之外，我们还需要**考虑运行时间**，就是需要多长时间来分类一张图。分类器A需要 80 毫秒，B需要 95 毫秒，C需要 1500 毫秒，就是说需要 1.5 秒来分类图像。



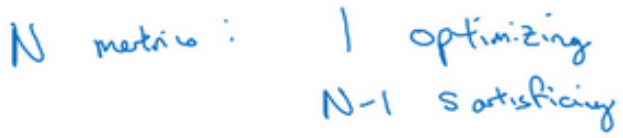
你可以这么做，**将准确度和运行时间组合成一个整体评估指标。**所以成本，比如说，总体成本是$cost = accuracy - 0.5 \times \text{runningTime}$，这种组合方式可能太刻意，只用这样的公式来组合准确度和运行时间，两个数值的线性加权求和。

你还可以做其他事情，就是你可能选择一个分类器，**能够最大限度提高准确度，但必须满足运行时间要求，**就是对图像进行分类所需的时间必须小于等于 100 毫秒。所以在这种情况下，我们就说准确度是一个优化指标，因为你想要准确度最大化，你想做的尽可能准确，但是运行时间就是我们所说的满足指标，意思是它必须足够好，它只需要小于 100 毫秒，达

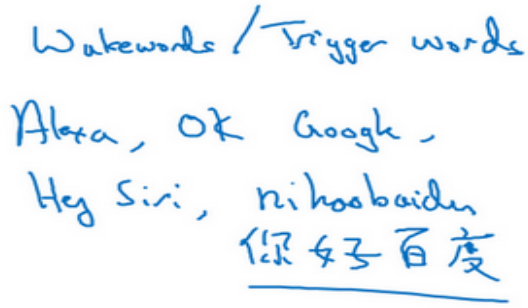
到之后，你不在乎这指标有多好，或者至少你不会那么在乎。所以这是一个相当合理的权衡方式，或者说将准确度和运行时间结合起来的方式。实际情况可能是，只要运行时间少于 100 毫秒，你的用户就不会在乎运行时间是 100 毫秒还是 50 毫秒，甚至更快。

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

通过定义优化和满足指标，就可以给你提供一个明确的方式，去选择“最好的”分类器。在这种情况下分类器 B 最好，因为在所有的运行时间都小于 100 毫秒的分类器中，它的准确度最好。



所以更一般地说，**如果你要考虑N个指标，有时候选择其中一个指标做为优化指标是合理的。所以你想尽量优化那个指标，然后剩下$N-1$个指标都是满足指标，意味着只要它们达到一定阈值，**例如运行时间快于 100 毫秒，但只要达到一定的阈值，你不在乎它超过那个门槛之后的表现，但它们必须达到这个门槛。

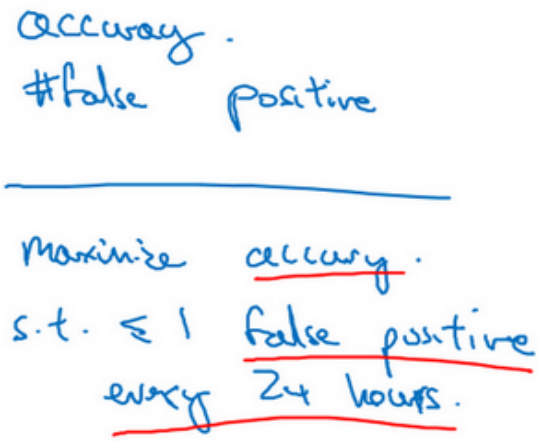


这里是另一个例子，假设你正在构建一个系统来检测唤醒语，也叫触发词，这指的是语音控制设备。比如亚马逊 **Echo**，你会说“**Alexa**”，或者用“**Okay Google**”来唤醒谷歌设备，或者对于苹果设备，你会说“**Hey Siri**”，或者对于某些百度设备，我们用“你好百度”唤醒。

对的，这些就是唤醒词，可以唤醒这些语音控制设备，然后监听你想说的话。所以你可能会在乎触发字**检测系统的准确性**，所以当有人说出其中一个触发词时，有多大概率可以唤醒你的设备。

你可能也需要顾及**假阳性（false positive）的数量**，就是没有人在说这个触发词时，它被随机唤醒的概率有多大？所以这种情况下，组合这两种评估指标的合理方式可能是最大

化精确度。所以当某人说出唤醒词时，你的设备被唤醒的概率最大化，然后必须满足 24 小时内最多只能有 1 次假阳性，对吧？所以你的设备平均每天只会没有人真的在说话时随机唤醒一次。所以在这种情况下，准确度是优化指标，然后每 24 小时发生一次假阳性是满足指标，你只要每 24 小时最多有一次假阳性就满足了。



总结一下，如果你需要顾及多个指标，比如说，有一个优化指标，你想尽可能优化的，然后还有一个或多个满足指标，需要满足的，需要达到一定的门槛。现在你就有一个全自动的方法，在观察多个成本大小时，选出"最好的"那个。现在这些评估指标必须是在训练集或开发集或测试集上计算或求出来的。所以你还需要做一件事，就是设立训练集、开发集，还有测试集。在下一个视频里，我想和大家分享一些如何设置训练、开发和测试集的指导方针，我们下一个视频继续。

1.5 训练/开发/测试集划分（Train/dev/test distributions）

设立训练集，开发集和测试集的方式大大影响了你或者你的团队在建立机器学习应用方面取得进展的速度。同样的团队，即使是大公司里的团队，在设立这些数据集的方式，真的会让团队的进展变慢而不是加快，我们看看应该如何设立这些数据集，让你的团队效率最大化。

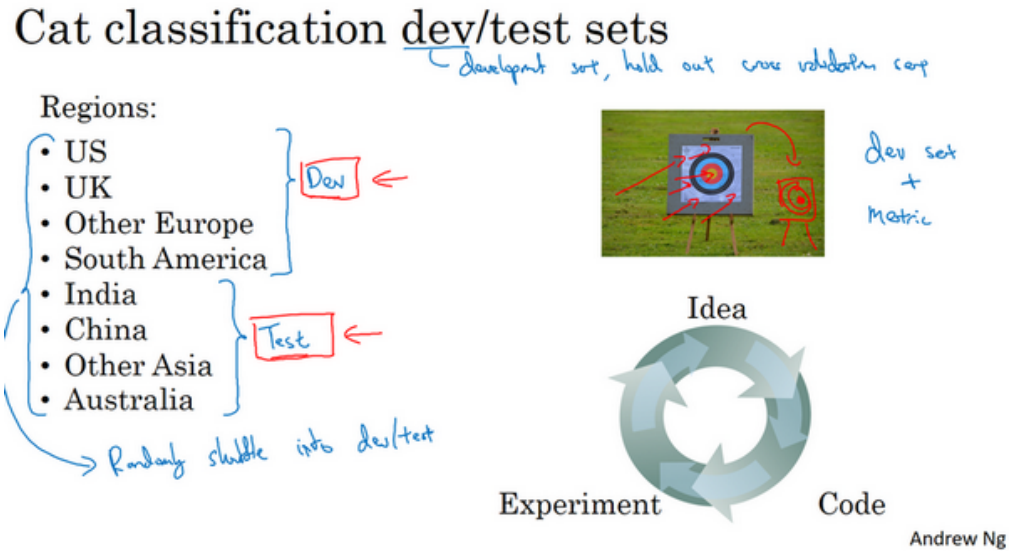
在这个视频中，我想集中讨论如何设立开发集和测试集，开发（dev）集也叫做（development set），有时称为保留交叉验证集（hold out cross validation set）。然后，机器学习中的工作流程是，你尝试很多思路，用训练集训练不同的模型，然后使用开发集来评估不同的思路，然后选择一个，然后不断迭代去改善开发集的性能，直到最后你可以得到一个令你满意的成本，然后你再用测试集去评估。

现在，举个例子，你要开发一个猫分类器，然后你在这些区域里运营，美国、英国、其他欧洲国家，南美洲、印度、中国，其他亚洲国家和澳大利亚，那么你应该如何设立开发集和测试集呢？

其中一种做法是，你可以选择其中 4 个区域，我打算使用这四个（前四个），但也可以是随机选的区域，然后说，来自这四个区域的数据构成开发集。然后其他四个区域，我打算用这四个（后四个），也可以随机选择 4 个，这些数据构成测试集。

事实证明，这个想法非常糟糕，因为这个例子中，你的开发集和测试集来自不同的分布。我建议你们不要这样，而是让你的开发集和测试集来自同一分布。我的意思是这样，你们要

记住，我想就是设立你的开发集加上一个单实数评估指标，这就是像是定下目标，然后告诉你的团队，那就是你要瞄准的靶心，因为你一旦建立了这样的开发集和指标，团队就可以快速迭代，尝试不同的想法，跑实验，可以很快地使用开发集和指标去评估不同分类器，然后尝试选出最好的那个。所以，机器学习团队一般都很擅长使用不同方法去逼近目标，然后不断迭代，不断逼近靶心。所以，针对开发集上的指标优化。

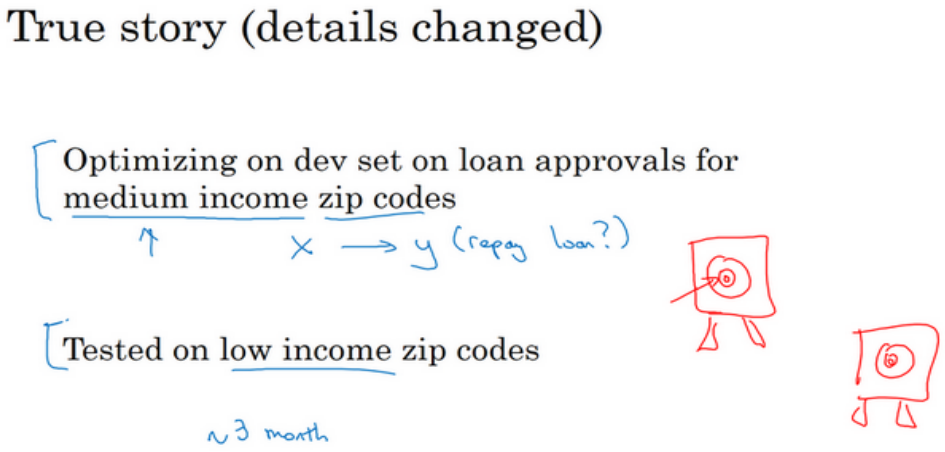


然后在左边的例子中，设立开发集和测试集时存在一个问题，你的团队可能会花上几个月时间在开发集上迭代优化，结果发现，当你们最终在测试集上测试系统时，来自这四个国家或者说下面这四个地区的数据（即测试集数据）和开发集里的数据可能差异很大，所以你可能会收获"意外惊喜"，并发现，花了那么多个月的时间去针对开发集优化，在测试集上的表现却不佳。所以，如果你的开发集和测试集来自不同的分布，就像你设了一个目标，让你的团队花几个月尝试逼近靶心，结果在几个月工作之后发现，你说“等等”，测试的时候，"我要把目标移到这里"，然后团队可能会说"好吧，为什么你让我们花那么多个月的时间去逼近那个靶心，然后突然间你可以把靶心移到不同的位置？"。

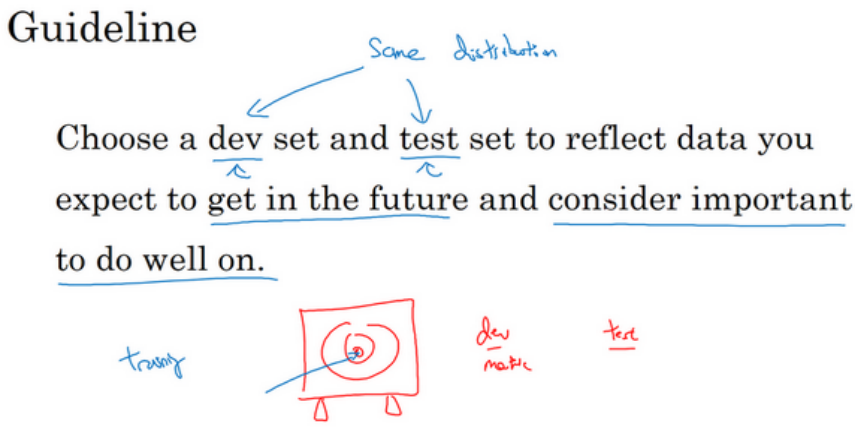
所以，为了避免这种情况，我建议的是你**将所有数据随机洗牌，放入开发集和测试集，所以开发集和测试集都有来自八个地区的数据，并且开发集和测试集都来自同一分布，**这分布就是你的所有数据混在一起。

这里有另一个例子，这是个真实的故事，但有一些细节变了。所以我知道有一个机器学习团队，花了好几个月在开发集上优化，开发集里面有中等收入邮政编码的贷款审批数据。那么具体的机器学习问题是，输入x为贷款申请，你是否可以预测输出y，y是他们有没有还贷能力？所以这系统能帮助银行判断是否批准贷款。所以开发集来自贷款申请，这些贷款申

请来自中等收入邮政编码，**zip code** 就是美国的邮政编码。但是在这上面训练了几个月之后，团队突然决定要在，低收入邮政编码数据上测试一下。当然了，这个分布数据里面中等收入和低收入邮政编码数据是很不一样的，而且他们花了大量时间针对前面那组数据优化分类器，导致系统在后面那组数据中效果很差。所以这个特定团队实际上浪费了 3 个月的时间，不得不退回去重新做很多工作。



这里实际发生的事情是，这个团队花了三个月瞄准一个目标，三个月之后经理突然问"你们试试瞄准那个目标如何？"，这新目标位置完全不同，所以这件事对于这个团队来说非常崩溃。



所以我建议你们在设立开发集和测试集时，要选择这样的开发集和测试集，能够反映你未来会得到的数据，认为很重要的数据，必须得到好结果的数据，特别是，这里的开发集和测试集可能来自同一个分布。所以不管你未来会得到什么样的数据，一旦你的算法效果不错，要尝试收集类似的数据，而且，**不管那些数据是什么，都要随机分配到开发集和测试集上。**因为这样，你才能将瞄准想要的目标，让你的团队高效迭代来逼近同一个目标，希望最好是同一个目标。

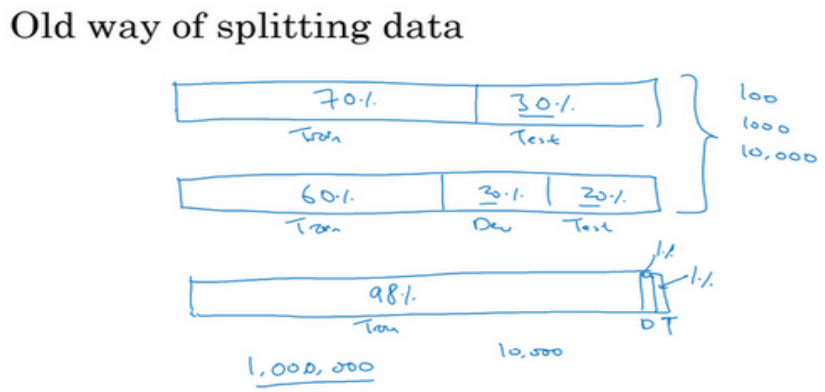
我们还没提到如何设立训练集，我们会在之后的视频里谈谈如何设立训练集，但这个视

频的重点在于，设立开发集以及评估指标，真的就定义了你要瞄准的目标。我们希望通过在同一分布中设立开发集和测试集，你就可以瞄准你所希望的机器学习团队瞄准的目标。而设立训练集的方式则会影响你逼近那个目标有多快，但我们可以在另一个讲座里提到。我知道有一些机器学习团队，他们如果能遵循这个方针，就可以省下几个月的工作，所以我希望这些方针也能帮到你们。

接下来，实际上你的开发集和测试集的规模，如何选择它们的大小，在深度学习时代也在变化，我们会在下一个视频里提到这些内容。

1.6 开发集和测试集的大小（Size of dev and test sets）

在上一个视频中你们知道了你的开发集和测试集为什么必须来自同一分布，但它们规模应该多大？在深度学习时代，设立开发集和测试集的方针也在变化，我们来看看一些最佳做法。



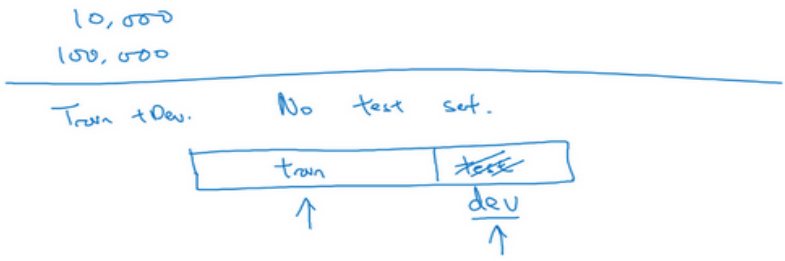
你可能听说过一条经验法则，在机器学习中，**把你取得的全部数据用 70/30 比例分成训练集和测试集。**或者如果你必须设立训练集、开发集和测试集，你会这么分 **60%训练集，20%开发集，20%测试集。**在机器学习的早期，这样分是相当合理的，特别是以前的数据集大小要小得多。所以如果你总共有 100 个样本，这样 70/30 或者 60/20/20 分的经验法则是相当合理的。如果你有几千个样本或者有一万个样本，这些做法也还是合理的。

但在现代机器学习中，我们更习惯操作规模大得多的数据集，比**如说你有 1 百万个训练样本，这样分可能更合理，98%作为训练集，1%开发集，1%测试集，我们用*D*和*T*缩写来表示开发集和测试集。**因为如果你有 1 百万个样本，那么 1%就是 10,000 个样本，这对于开发集和测试集来说可能已经够了。所以在现代深度学习时代，有时我们拥有大得多的数据集，所以使用小于 20%的比例或者小于 30%比例的数据作为开发集和测试集也是合理的。而且因为深度学习算法对数据的胃口很大，我们可以看到那些有海量数据集的问题，有更高比例的数据划分到训练集里，那么测试集呢？

要记住，**测试集的目的是完成系统开发之后，测试集可以帮你评估投产系统的性能。方针就是，令你的测试集足够大，能够以高置信度评估系统整体性能。**所以除非你需要对最终投产系统有一个很精确的指标，一般来说测试集不需要上百万个例子。对于你的应用程序，也许你想，有 10,000 个例子就能给你足够的置信度来给出性能指标了，也许 100,000 个之类的可能就够了，这数目可能远远小于比如说整体数据集的 30%，取决于你有多少数据。

Size of test set

→ Set your test set to be big enough to give high confidence in the overall performance of your system.



对于某些应用，**你也许不需要对系统性能有置信度很高的评估，也许你只需要训练集和开发集。我认为，不单独分出一个测试集也是可以的。**事实上，有时在实践中有些人会只分成训练集和测试集，他们实际上在测试集上迭代，所以这里没有测试集，他们有的是训练集和开发集，但没有测试集。如果你真的在调试这个集，这个开发集或这个测试集，这最好称为开发集。

不过在机器学习的历史里，不是每个人都把术语定义分得很清的，有时人们说的开发集，其实应该看作测试集。但如果你只要有数据去训练，有数据去调试就够了。你打算不管测试集，直接部署最终系统，所以不用太担心它的实际表现，我觉得这也是很好的，就将它们称为训练集、开发集就好。然后说清楚你没有测试集，这是不是有点不正常？我绝对不建议在搭建系统时省略测试集，因为有个单独的测试集比较令我安心。因为你可以使用这组不带偏差的数据来测量系统的性能。但如果你的开发集非常大，这样你就不会对开发集过拟合得太厉害，这种情况，只有训练集和测试集也不是完全不合理的。不过我一般不建议这么做。

总结一下，在大数据时代旧的经验规则，这个 70/30 不再适用了。**现在流行的是把大量数据分到训练集，然后少量数据分到开发集和测试集，特别是当你有一个非常大的数据集时。**以前的经验法则其实是为了确保开发集足够大，能够达到它的目的，就是帮你评估不同的想法，然后选出A还是B更好。测试集的目的是评估你最终的成本偏差，你只需要设立足够大的测试集，可以用来这么评估就行了，可能只需要远远小于总体数据量的 30%。

所以我希望本视频能给你们一点指导和建议，让你们知道如何在深度学习时代设立开发和测试集。接下来，有时候在研究机器学习的问题途中，你可能需要更改评估指标，或者改动你的开发集和测试集，我们会讲什么时候需要这样做。

1.7 什么时候该改变开发/测试集和指标？（When to change dev/test sets and metrics）

你已经学过如何设置开发集和评估指标，就像是把目标定在某个位置，让你的团队瞄准。但有时候在项目进行途中，你可能意识到，目标的位置放错了。这种情况下，你应该移动你的目标。

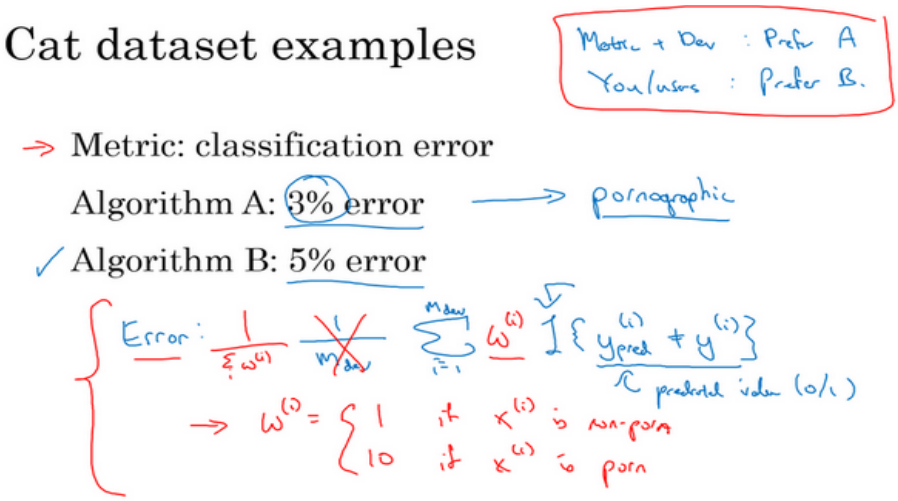
Metric: classification error

Algorithm A: 3% error

Algorithm B: 5% error

我们来看一个例子，假设你在构建一个猫分类器，试图找到很多猫的照片，向你的爱猫人士用户展示，你决定使用的指标是分类错误率。所以算法A和B分别有 3%错误率和 5%错误率，所以算法A似乎做得更好。

但我们实际试一下这些算法，你观察一下这些算法，算法A由于某些原因，把很多色情图像分类成猫了。如果你部署算法A，那么用户就会看到更多猫图，因为它识别猫的错误率只有 3%，但它同时也会给用户推送一些色情图像，这是你的公司完全不能接受的，你的用户也完全不能接受。相比之下，算法B有 5%的错误率，这样分类器就得到较少的图像，但它不会推送色情图像。所以从你们公司的角度来看，以及从用户接受的角度来看，算法B实际上是一个更好的算法，因为它不让任何色情图像通过。

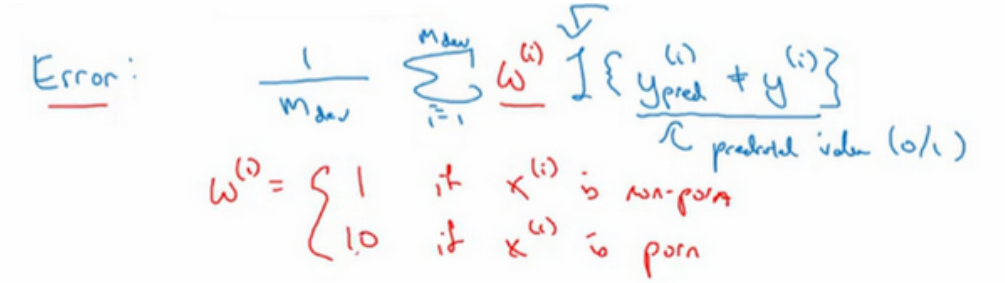


那么在这个例子中，发生的事情就是，算法 A 在评估指标上做得更好，它的错误率达到 3%，但实际上是个更糟糕的算法。在这种情况下，评估指标加上开发集它们都倾向于选择算

法A，因为它们会说，看算法 A 的错误率较低，这是你们自己定下来的指标评估出来的。但你和你的用户更倾向于使用算法B，因为它不会将色情图像分类为猫。所以当这种情况发生时，当你的评估指标无法正确衡量算法之间的优劣排序时，在这种情况下，原来的指标错误地预测算法 A 是更好的算法这就发出了信号，你应该改变评估指标了，或者要改变开发集或测试集。在这种情况下，你用的分类错误率指标可以写成这样：

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

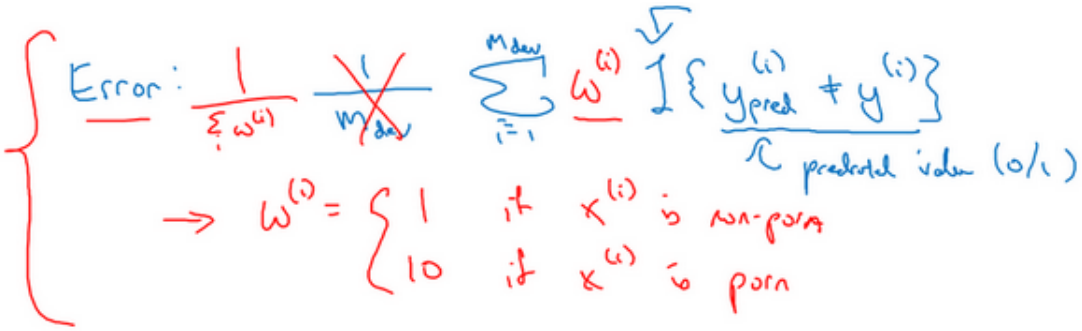
m_{dev}是你的开发集例子数，用$y_{pred}^{(i)}$表示预测值，其值为 0 或 1，I这符号表示一个函数，统计出里面这个表达式为真的样本数，所以这个公式就统计了分类错误的样本。这个评估指标的问题在于，它对色情图片和非色情图片一视同仁，但你其实真的希望你的分类器不会错误标记色情图像。比如说把一张色情图片分类为猫，然后推送给不知情的用户，他们看到色情图片会非常不满。



其中一个修改评估指标的方法是，这里（$\frac{1}{m_{dev}}$与$\sum_{i=1}^{m_{dev}} I\{y_{pred}^{(i)} \neq y^{(i)}\}$之间）加个权重项，即：

$$Error = \frac{1}{m_{dev}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

我们将这个称为$w^{(i)}$，其中如果图片$x^{(i)}$不是色情图片，则$w^{(i)} = 1$。如果$x^{(i)}$是色情图片，$w^{(i)}$可能就是 10 甚至 100，这样你赋予了色情图片更大的权重，让算法将色情图分类为猫图时，错误率这个项快速变大。**这个例子里，你把色情图片分类成猫这一错误的惩罚权重加大 10 倍。**



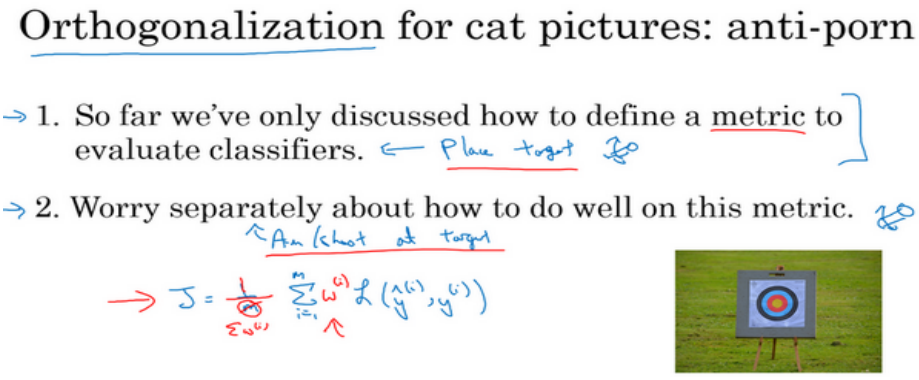
如果你希望得到归一化常数，在技术上，就是$w^{(i)}$对所有i求和，这样错误率仍然在 0 和 1 之间，即：

$$Error = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m_{dev}} w^{(i)} I\{y_{pred}^{(i)} \neq y^{(i)}\}$$

加权的细节并不重要，实际上要使用这种加权，你必须自己过一遍开发集和测试集，在开发集和测试集里，自己把色情图片标记出来，这样你才能使用这个加权函数。

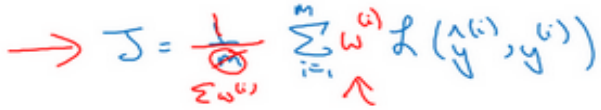
但粗略的结论是，**如果你的评估指标无法正确评估好算法的排名，那么就需要花时间定义一个新的评估指标。**这是定义评估指标的其中一种可能方式（上述加权法）。评估指标的意义在于，准确告诉你已知两个分类器，哪一个更适合你的应用。就这个视频的内容而言，我们不需要太注重新错误率指标是怎么定义的，关键在于，**如果你对旧的错误率指标不满意，那就不要一直沿用你不满意的错误率指标，而应该尝试定义一个新的指标，能够更加符合你的偏好，定义出实际更适合的算法。**

你可能注意到了，到目前为止我们只讨论了如何定义一个指标去评估分类器，也就是说，我们定义了一个评估指标帮助我们更好的把分类器排序，能够区分出它们在识别色情图片的不同水平，这实际上是一个正交化的例子。



我想你处理机器学习问题时，应该把它切分成独立的步骤。一步是弄清楚如何定义一个指标来衡量你想做的事情的表现，然后我们可以分开考虑如何改善系统在这个指标上的表

现。你们要把机器学习任务看成两个独立的步骤，用目标这个比喻，第一步就是设定目标。所以要定义你要瞄准的目标，这是完全独立的一步，这是你可以调节的一个旋钮。如何设立目标是一个完全独立的问题，把它看成是一个单独的旋钮，可以调试算法表现的旋钮，如何精确瞄准，如何命中目标，定义指标是第一步。

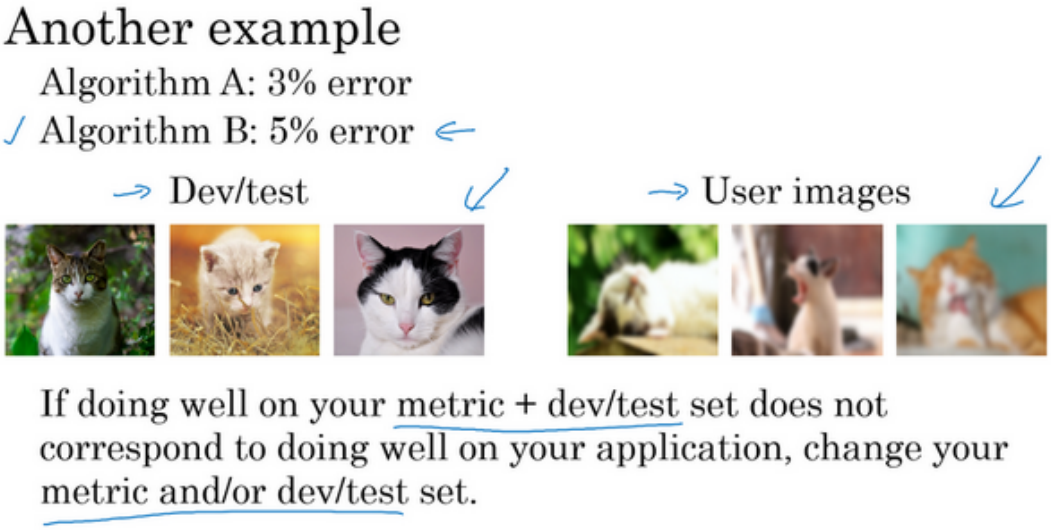


然后第二步要做别的事情，在逼近目标的时候，也许你的学习算法针对某个长这样的成本函数优化，$J = \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)})$，你要最小化训练集上的损失。你可以做的其中一件事是，修改这个，为了引入这些权重，也许最后需要修改这个归一化常数，即：

$$J = \frac{1}{\sum w^{(i)}} \sum_{i=1}^{m} w^{(i)} L(\hat{y}^{(i)}, y^{(i)})$$

再次，如何定义J并不重要，关键在于正交化的思路，把设立目标定为第一步，然后瞄准和射击目标是独立的第二步。换种说法，我鼓励你们将定义指标看成一步，然后在定义了指标之后，你才能想如何优化系统来提高这个指标评分。比如改变你神经网络要优化的成本函数J。

在继续之前，我们再讲一个例子。假设你的两个猫分类器A和B，分别有用开发集评估得到 3%的错误率和 5%的错误率。或者甚至用在网上下载的图片构成的测试集上，这些是高质量，取景框很专业的图像。但也许你在部署算法产品时，你发现算法B看起来表现更好，即使它在开发集上表现不错，你发现你一直在用从网上下载的高质量图片训练，但当你部署到手机应用时，算法作用到用户上传的图片时，那些图片取景不专业，没有把猫完整拍下来，或者猫的表情很古怪，也许图像很模糊，当你实际测试算法时，你发现算法B表现其实更好。



这是另一个指标和开发集测试集出问题的例子，问题在于，你做评估用的是很漂亮的高分辨率的开发集和测试集，图片取景很专业。但你的用户真正关心的是，他们上传的图片能不能被正确识别。那些图片可能是没那么专业的照片，有点模糊，取景很业余。

所以方针是，如果你在指标上表现很好，在当前开发集或者开发集和测试集分布中表现很好，但你的实际应用程序，你真正关注的地方表现不好，那么就需要修改指标或者你的开发测试集。换句话说，如果你发现你的开发测试集都是这些高质量图像，但在开发测试集上做的评估无法预测你的应用实际的表现。因为你的应用处理的是低质量图像，那么就应该改变你的开发测试集，让你的数据更能反映你实际需要处理好的数据。

但总体方针就是，如果你当前的指标和当前用来评估的数据和你真正关心必须做好的事情关系不大，那就应该更改你的指标或者你的开发测试集，让它们能更够好地反映你的算法需要处理好的数据。

有一个评估指标和开发集让你可以更快做出决策，判断算法A还是算法B更优，这真的可以加速你和你的团队迭代的速度。所以我的建议是，即使你无法定义出一个很完美的评估指标和开发集，你直接快速设立出来，然后使用它们来驱动你们团队的迭代速度。如果在这之后，你发现选的不好，你有更好的想法，那么完全可以马上改。对于大多数团队，我建议最好不要在没有评估指标和开发集时跑太久，因为那样可能会减慢你的团队迭代和改善算法的速度。本视频讲的是什么时候需要改变你的评估指标和开发测试集，我希望这些方针能让你的整个团队设立一个明确的目标，一个你们可以高效迭代，改善性能的目标。