Contents

contents

1 1. error analysis

1.1 1.1. carrying out error analysis

例如，一个识别猫的项目，发现有一些是狗的图片被分成了猫，是不是应该用一些方法（例如，加入一些可以区分猫狗的特征，或者加入更多狗的图片去训练）让分类器对狗的识别表现得更好?

分析方法:

- 抽取大约 100 张 dev set 中标错的图片
- 看看多少张是狗

如果比例很低，那提高准确率的上限（ceiling）其实也很小，所以没必要。如果比例高，那就值得尝试。

还可以并行地评估不同的想法:

- 修复把狗识别成猫的图片
- 修复把大型猫科动物（如狮子、豹子等）识别成猫的图片
- 改进在模糊图片的识别准确率

可以画个表格，看每一种 case 的比例，并给每个 case 标上注释。而在看 case 的过程中，可能会发现新的问题（例如，加了 xxx 滤镜），那么，就加新的一列呗。。

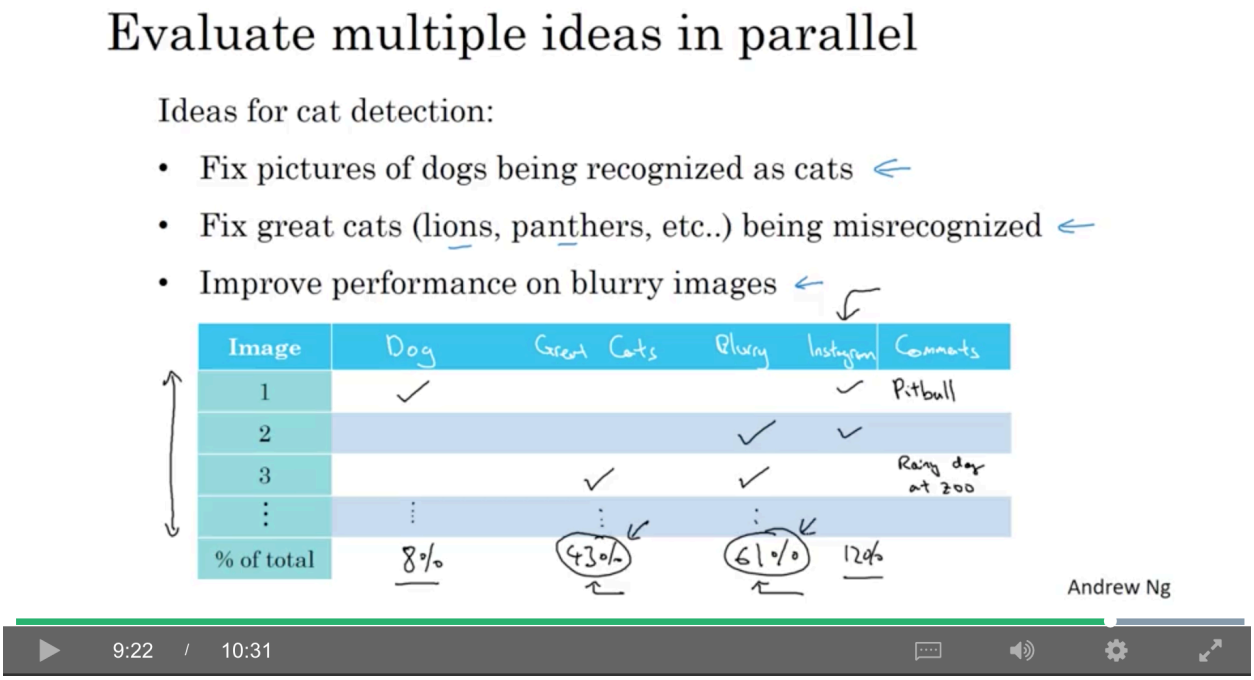

Figure 1: multiple-case-error-analysis.png

1.2 1.2. cleaning up incorrectly labeled data

可能有些数据的 label 标错了，分以下几种情况来看：

- 如果是训练集的数据：如果这种标错是**随机**的，因为 dl 对 random errors 是很 robust 的，所以只要数据集够大，且标错的数据很少，就可以不理。但 dl 对系统误差（systemetic errors）并不 robust，例如，如果标记人员把白色的狗都标成了猫，那 dl 也会倾向于把白色的狗识别成猫
- 如果是 dev/test set 中的数据，同样地画个表格，然后加一列，表示是否标注错误，然后关注以下三个数字：
  - overall dev set error: 例如是 10%
  - 因为标错数据导致的 error：例如在 100 个抽样的 case 里，有 6 个，那就是 6%*10%=0.6%
  - 因为其他原因导致的 error：10%-0.6%=9.4%

需要保证 dev 和 test set 是同一分布的，因为会影响到泛化；但训练集和 dev/test set 的分布有些小区别是可以接受的。

1.3 1.3. build your first system quickly, then iterate

举个语音识别的例子，可能有以下几个方向可以尝试来改进一个现有的系统：

- 有噪音的背景
  - 在咖啡厅的噪音
  - 汽车的噪音
- 方言
- 距离麦克风太远（远场语音识别）
- 孩子的声音（发音、常用词库的不同）
- 结巴（或者一些停顿的语气词）

如果想要开始一个新项目，建议：

- 设置 dev/test set 以及 metric
- 快速搭建一个初始的系统（不要搞得太复杂，quick and dirty 就行啦）
- 使用 bias/variance 分析，以及 error 分析来决定接下来 todo 的优先级

以上建议不适用于：

- 你已经有丰富经验的领域
- 学术界已经有很多很成熟的研究的领域

2 2. mismatched training and dev/test set

2.1 2.1. training and testing on different distributions

例如，开发一个应用来识别猫，有两种来源，一种是用户上传的清晰度比较差的图（最终目标，可能只有 1w），还有一种是从网上爬的高清图（可能有 20w）。有以下两种方法：

- 方案 1（不采纳）：全部 21w 数据混在一起 shuffle，然后 20500 做 train，2500 做 dev，2500 做 test。
  - 优点：train/dev/test 都来自同一分布，易于管理
  - 缺点：dev 有一大部分是来自网络的，而非目标分布的。但**设置 dev 的目的就是找到团队的目标**
- 方案 2：将所有 20w 的网络图像放入 train，dev/test 全部都来自用户上传 (2500+2500)，剩下的 5000 个用户上传的可以也放进 train 中
  - 缺点：train 和 dev/test 的分布不一致，接下来会讲如何优化

2.2 2.2. bias and variance with mismatched data distributions

例如，bayes error 接近 0%，training-error 是 1%，而 dev-error 是 10%，这个从 train 到 dev 的 gap，难以一下子判断出原因，因为有以下两个因素：

- 因为算法只看见了 train 的数据，而没有见过 dev
- dev 的数据分布与 train 不同

因此，定义一个新的数据集：training-dev set，**与 train 同分布，但训练时并未使用**。

- train-error 是 1%，training-dev error 是 9%，dev-error 是 10%，那么就是一个 variance 问题，因为这说明在同样分布的数据集上，泛化能力不行。
- train-error 是 1%，training-dev error 是 1.5%，dev-error 是 10%，那这是一个 data-mismatch 问题，因为两者分布不同导致了这个 gap
- train-error 是 10%，training-dev error 是 11%，dev-error 是 12%，但因为 bayes error 是 0%，所以，是一个 avoidable bias 问题
- train-error 是 10%，training-dev error 是 11%，dev-error 是 20%，那这就是一个 avoidable bias 加上 data mismatch 的问题

总结如下：

图中的 dev 和 test 的 gap 表示对 dev 的过拟合程度，如果过拟合得比较严重，那么就需要增大 dev 的 size。

如果出现图中右边的情况，也就是在 dev/test 上的表现比 train/training-dev 要好，那有可能是，训练集比 dev/test 更难训练。

一个更通用的总结：

图中，第一列是通用的数据集，第二列是特定问题的数据集，第一行是人类的表现，第二行是在参与了训练的数据上的 error，第三行是在没有参与训练的数据上的 error。

- (1,1) 是 human-error
- (2,1) 是 train-error
- (3,1) 是 training-dev error

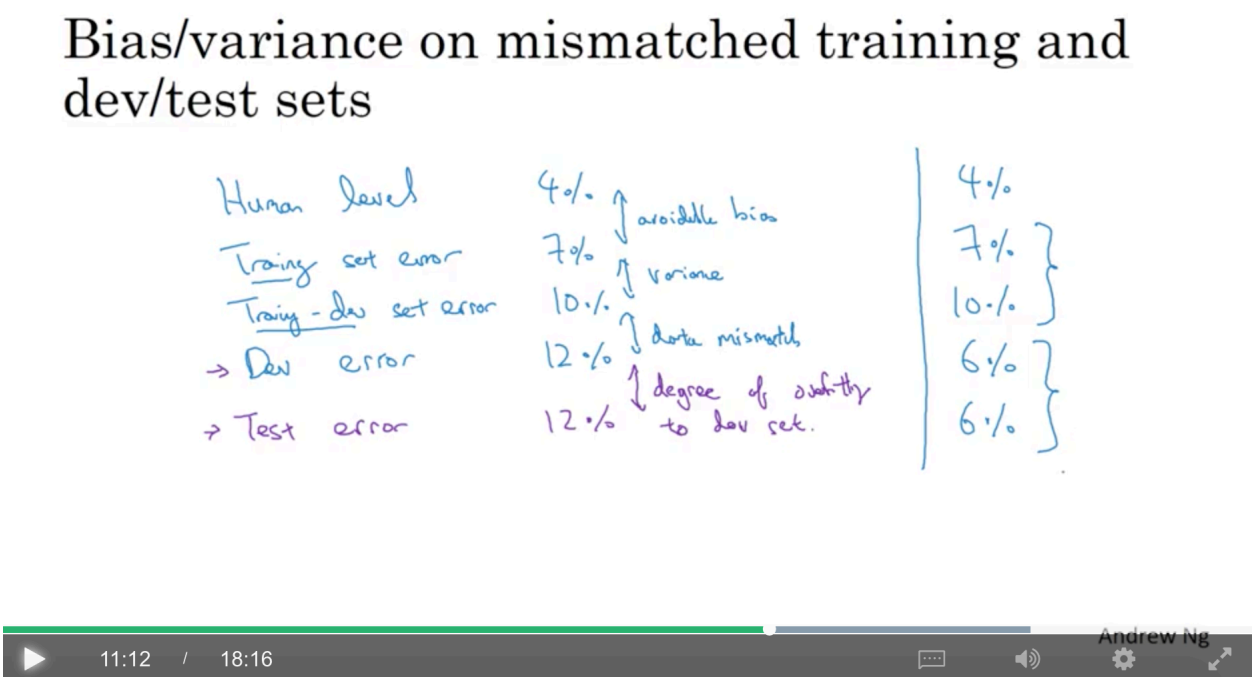

Figure 2: bias-variance-analysis-on-mismatched-training-and-dev-test-sets.png

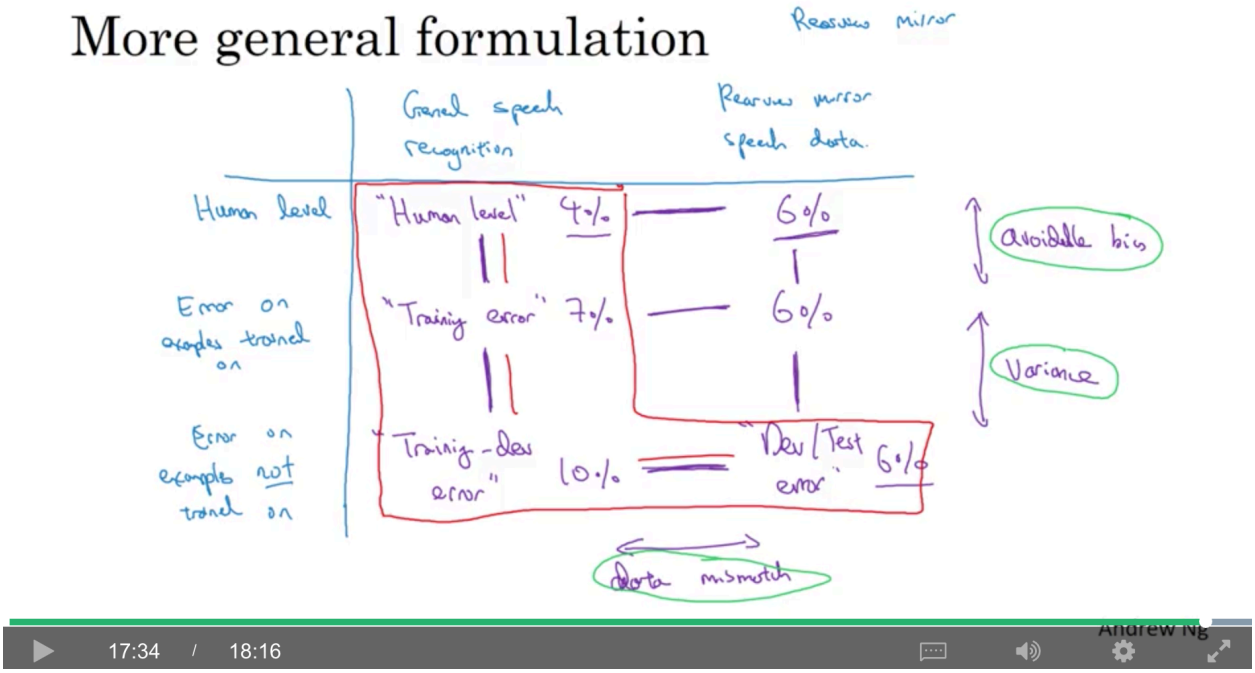

Figure 3: more-general-formula.png

- (3,2) 是 dev/test error
- (2,1)-(1,1)，即 train-error 与 human-error 之差，是 avoidable bias
- (3,1)-(2,1)，即 training-dev error 与 train-error 之差，是 variance
- (3,2)-(3,1)，即 dev-error 与 training-dev error 之差，是 data mismatch
- 而 (1,2) 是人工对目标数据集的标注 error
- (2,2) 是有了一些人工标注的目标数据集后，把它们拿来训练，这部分的 error
- 而看 (1,2)-(1,1) 或者 (2,2)-(2,1)，能够看出目标问题的难易程度

2.3 2.3. addressing data mismatch

- 进行人工的 error analysis，以理解 train 和 dev/test 的 difference
- 收集更像 dev/test 的训练数据，或者构造与 dev/test 更像的数据

人工数据合成：可以将安静环境中的语音和汽车环境的噪音进行合成。

注意：例如有 1w 小时的安静环境中的语音和 1 小时的汽车环境的噪音，可以把这 1 小时复制 1w 遍，和安静环境的语音进行合成。虽然在人听起来，这合成的都不一样，但这样可能会让模型对这 1 小时的噪音 (因为汽车噪音非常多，这 1 小时只是其中很小的一个子集)overfitting。而事实上，获取 1w 小时的不同噪音其实成本并不大，所以建议去搞 1w 小时的不同噪音。

3 3. learning from multiple tasks

3.1 3.1. transfer learning

例如，已经训练好了一个图像分类的模型，想做放射诊断，可以保持前面的网络结构和权重不变 (pre-train)，只把最后一层的权重随机初始化，然后 fine-tune：

- 如果数据集比较小，可以只 re-train 最后一层的权重
- 如果数据集够大，可以重训所有权重

when transfer learning makes sense

- A 和 B 两个任务有相同类型的 input x
- A 的数据比 B 要多很多
- A 的低阶特征或许对 B 任务是有用的

3.2 3.2. multi-task learning

例如无人驾驶，需要同时识别行人/车/路标/红绿灯，这样，目标 $y^{(i)}$ 就是一个 4x1 的向量，所以矩阵 Y 的 shape 为 (4,m)。所以，

$$Loss = \frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{4} L(\hat{y}_j^{(i)}, y_j^{(i)})$$

$$L(\hat{y}_j^{(i)}, y_j^{(i)}) = -y_j^{(i)} log\hat{y}_j^{(i)} - (1-y_j^{(i)}) log(1-\hat{y}_j^{(i)})$$

而这个和 softmax regression 不同，因为 softmax regression 要求 $y^{(i)}$ 只能是 4 类之一（即这个 4x1 的向量是 one-hot），而这里一个 $y^{(i)}$ 可以同时是多类（即这个 4x1 的可以同时有多位是 1）

如果对于一个样本有一些 label 没有被标注，例如，某张图，标注人员看不出里面有没有行人，可能行人这一维标了一个『?』。对于这种情况，上式的 $\sum_{j=1}^{4}$ 修改为只对有 label 不为『?』的 $y_j^{(i)}$ 和 $\hat{y}_j^{(i)}$ 的 loss 求和。

when multi-task learning makes sense

- 一系列的 task，如果共享一些低层次的特征，会训练得更好
- （不一定必要，但经常是对的）每个 task 的数据量差别不大，因为其他 task 的数据量加起来，会对单一 task 的数据有补充

- 能够训练一个足够大的网络，来使每个 task 都有很好的表现。因为，如果一个网络不够大，可能与各个 task 分开训练相比，反而准确率会更低

实际应用中，transfer learning 比 multi-task 要多一些。

4 4. end-to-end deep learning

4.1 4.1. what is end-to-end deep learning?

例如，语音识别任务，传统方法可能先通过 MFCC 之类的方法，人工提取特征，然后基于这些特征使用 ml 算法提取 phonemes(音素，是声音的基本单位)，然后得到对应的 word，然后把 words 串起来，得到 transcript。而 end-to-end，就是直接输入语音，输出 transcript。

- 当数据量比较小的时候，例如只有 3k 小时的数据，可能传统方法比 end-to-end 效果还要好
- 而当数据量比较大时，例如 1w，甚至 10w 的时候，可能 end-to-end 会突然变好
- 当数据量中等时，可以分段，例如，从语音到 phonemes 可以 end-to-end，然后后面的步骤按传统方法来搞

例如，人脸闸机的任务，业界最有效的方法并不是直接输入一张图像，识别出这是哪个人，因为人出现的位置可能不同，由于距离的远近，人脸的大小也会有很大的差别，比较有效的方法是分阶段的：

- 首先，用一个探测器识别出人脸的位置
- 然后放大人脸，使人脸位于图像的中心，然后拿这幅图像去识别这个人的身份（其实就是把两张图作为输入，然后判断是不是同一个人，所以如果一个公司有 1w 人，就是快速判断这张图和这 1w 个人哪个最像）

如果需要通过一张手掌的 x 光片，判断孩子的年龄，可以先用 nn 提取出骨骼，然后通过骨骼的长度，查表去判断孩子的年龄。而如果想直接从图片到年龄，实际上并没有足够多的数据，所以可行性相对差一些。

4.2 4.2. whether to use end-to-end deep learning

优点：

- let data speak：更容易学到数据内在的统计学特征，而非被迫去反映人的先见
- 所需的人工设计的特征更少了，可以简化整个工作流程

缺点：

- 可能需要非常大量的数据
- 排除了一些具有潜在用途的手工设计组件。当数据量不够多时，一个精心手工设计的系统实际上中以将更多的人对于这个问题的知识注入算法之中

所以核心问题是：是否有足够的数据去学习出具有能够映射 x 到 y 所需复杂度的函数。

- 可以用 dl 来学习一些独立的组成部分
- 根据你想学习的任务获取数据的难易程度，认真选择 x->y 的映射类型