

# 第六周学习笔记

2022-05-19

# 第一章 应用机器学习的建议

## 1.1 决定下一步做什么

在懂机器学习的人当中依然存在着很大的差距，一部分人确实掌握了怎样高效有力地运用这些学习算法。而另一些人他们可能对系统的学习不是那么熟悉了，他们可能没有完全理解怎样运用这些算法。因此总是把时间浪费在毫无意义的尝试学习上。

希望通过后续的学习能够明白怎样选择一条最合适、最正确的道路。

具体来讲，我们将重点关注的问题是假如我们在开发一个机器学习系统，或者想试着改进一个机器学习系统的性能，应该如何决定接下来应该选择哪条道路？

为了解释这一问题，我们仍然使用之前预测房价的学习例子。

Suppose you have implemented regularized linear regression to predict housing prices.

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

假如我们已经完成了正则化线性回归，也就是最小化代价函数  $J(\theta)$  的值。

在你得到你的学习参数以后，如果我们想要将自己得到的假设函数放到一组新的房屋样本上进行测试，如果发现在预测房价时产生了巨大的误差，现在的问题是要想改进这个算法，接下来应该怎么办？

实际上我们或许可以想出很多种方法来改进这个算法的性能。

其中一种办法是使用更多的训练样本。具体来讲，也许我们能想到通过电话调查或上门调查来获取更多的不同的房屋出售数据。

但有时候获得更多的训练数据实际上并没有作用。

另一个方法，我们也许能想到的是尝试选用更少的特征集。

如果我们有一系列特征比如  $x_1, x_2, x_3$  等等。也许有很多特征，也许可以花一点时间从这些特征中仔细挑选一小部分来防止过拟合，或者需要用更多的特征，也许目前的特征集对我们来讲并不是很有帮助。我们希望从获取更多特征的角度来收集更多的数据，同样地，我们也可以把这个问题扩展为一个很大的项目，比如使用电话调查来得到更多的房屋案例，或者再进行土地测量来获得更多有关，这块土地的信息等等，因此这是一个复杂的问题。

同样的道理，我们非常希望在花费大量时间完成这些工作之前，就能知道其效果如何。

我们也可以尝试增加多项式特征的方法，比如  $x_1^2, x_2^2, x_1x_2$ ，接下来可以花很多时间来考虑这些多项式特征的组合情况。

我们也可以考虑其他方法，比如减小或增大正则化参数  $\lambda$  的值。

遗憾的是，大多数人用来选择这些方法的标准是凭感觉的，也就是说，大多数人的选择方法是随便从这些方法中选择一种。

幸运的是，有一系列简单的方法能让我们的工作事半功倍，排除掉单子上的至少一半的方法，留下那些确实有前途的方法，同时也有一种很简单的方法，只要我们使用，就能很轻松地排除掉很多选择，从而为学习工作节省大量不必要花费的时间。

为了最终达到改进机器学习系统性能的目的，假设我们需要用一个线性回归模型来预测房价，当我们运用训练好了的模型来预测未知数据的时候发现有较大的误差，我们下一步可以做什么？

1. 获得更多的训练样本

上面的方法通常是有效的，但代价较大，下面的方法也可能有效，可考虑先采用下面的几种方法。

2. 尝试减少特征的数量

3. 尝试获得更多的特征

4. 尝试增加多项式特征

5. 尝试减少正则化程度  $\lambda$

6. 尝试增加正则化程度  $\lambda$

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

## 1.2 评估假设

介绍一下怎样用之前学过的算法来评估假设函数。

当我们确定学习算法的参数的时候，我们考虑的是选择合适的参数值来使训练误差最小化。

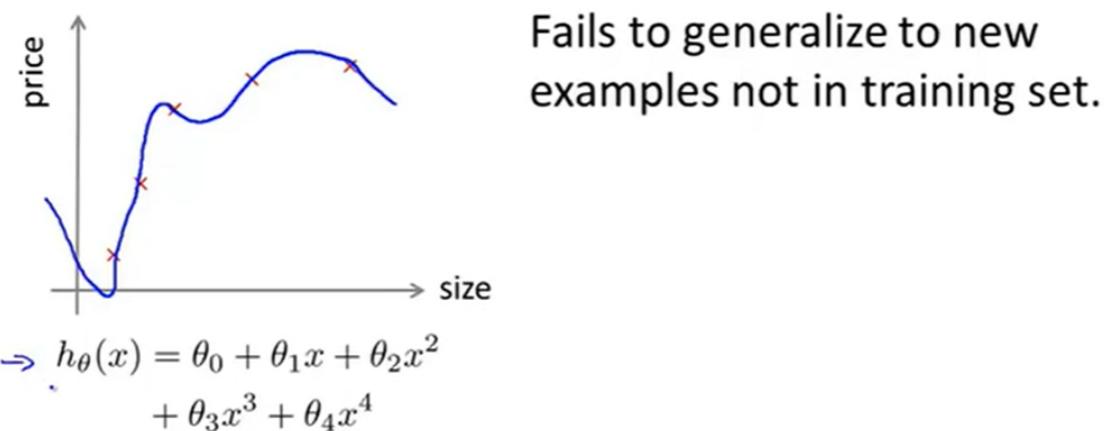
有人认为得到一个非常小的训练误差一定是一件好事，但我们已经知道，仅仅是因为这个假设具有很小的训练误差，并不能说明它就一定是一个好的假设函数。

而且我们也学习了过拟合假设函数的例子，所以这推广到新的训练集上是不适用的。

该如何判断一个假设函数是过拟合的呢？

对于这个简单的例子，我们可以对假设函数  $h(x)$  进行画图，然后观察图形趋势。

### Evaluating your hypothesis



但对于特征变量不止一个的这种一般情况，还有像有很多特征变量的问题，想要通过画出假设函数来进行观察，就会变得很难甚至是不可能实现。

因此，我们需要另一种方法来评估我们的假设函数过拟合检验。

为了检验算法是否过拟合，我们将数据分成训练集和测试集，通常用 70% 的数据作为训练集，用剩下 30% 的数据作为测试集。

很重要的一点是训练集和测试集均要含有各种类型的数据，通常我们要对数据进行“洗牌”，然后再分成训练集和测试集。

具体的实现方法如下所示：

## Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	
<hr/>		
1427	199	
1380	212	
1494	243	
<hr/>		

$\frac{70\%}{30\%}$

Training set →  $(x^{(1)}, y^{(1)})$   
 $(x^{(2)}, y^{(2)})$   
 $\vdots$   
 $(x^{(m)}, y^{(m)})$

Test set →  $(x_{test}^{(1)}, y_{test}^{(1)})$   
 $(x_{test}^{(2)}, y_{test}^{(2)})$   
 $\vdots$   
 $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

$m_{test} = \text{no. of test example}$

Andrew Ng

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

- 对于线性回归模型，我们利用测试集数据计算代价函数  $J(\theta)$

## Training/testing procedure for linear regression

→ - Learn parameter  $\theta$  from training data (minimizing training error  $J(\theta)$ )  $\frac{70\%}{30\%}$

- Compute test set error:

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

- 对于逻辑回归模型，我们除了可以利用测试数据集来计算代价函数外：

## Training/testing procedure for logistic regression

→ - Learn parameter  $\theta$  from training data  $m_{test}$

- Compute test set error:

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

还有另一种形式的测试度量，可能更易于理解，叫做错误分类，也被称为 0-1 分类错误。

对于每一个测试样本，我们需要计算：

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

然后对计算结果求平均。

- Misclassification error (0/1 misclassification error):

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

error

$$\text{Test error} = \frac{1}{m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} \text{err}(h_\theta(x_{\text{test}}^{(i)}), y_{\text{test}}^{(i)}).$$

### 1.3 模型选择和训练、验证、训练集

假设我们要在 10 个不同次数的二项式模型之间进行选择一个合适的假设函数，类型如下所示：

#### Model selection

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

显然越高次数的多项式模型越能够适应我们的训练数据集，但是适应训练数据集并不代表着能推广至一般情况，我们应该选择一个更能适应一般情况的模型。

我们需要使用交叉验证集来帮助选择模型。

交叉验证集的实现方法为：使用 60% 的数据作为训练集，使用 20% 的数据作为交叉验证集，使用 20% 的数据作为测试集，如下图所示：

## Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	
1600	330	
60% 2400	369	Training set
1416	232	
3000	540	
1985	300	
20% 1534	315	Cross validation set (CV)
1427	199	
20% 1380	212	Test set
1494	243	

$m_{cv} = \text{no. of cv samples}$ 
 $(x_{cv}^{(i)}, y_{cv}^{(i)})$ 
 $m_{test}$ 
 $(x_{test}^{(i)}, y_{test}^{(i)})$

模型的方法为：

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

现在我们已经定义了训练集，交叉验证集和测试集，同样我们也可以定义训练误差，交叉验证误差和测试误差。如下图所示：

### Train/validation/test error

Training error:

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

Cross Validation error:

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

Test error:

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

对于上面的十个模型，接下来我们的做法，不是像原来一样用测试集来测试这些假设，而是用交叉验证集来测试，然后计算出  $J_{cv}(\theta)$  的值来观察这些假设模型在交叉验证集上的效果如何。

我们将选择交叉验证误差最小的那个假设作为我们的模型。

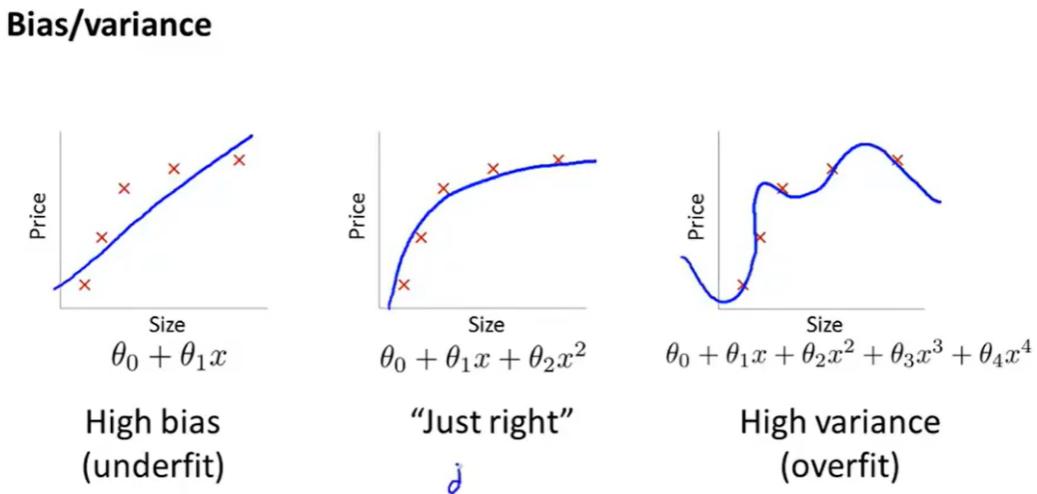
## 1.4 诊断偏差和方差

当我们运行一个学习算法时，如果这个算法的最终表现不理想，那么多半是出现两种情况：要么是偏差比较大，要么是方差比较大。

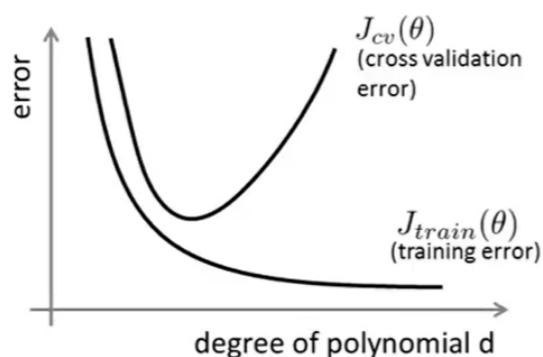
换句话说，出现的情况要么是欠拟合，要么是过拟合问题。

那么这两种情况，哪个和偏差有关，哪个和方差有关，或者是不是和两个都有关？搞清楚这一点非常重要，因为能判断出现的情况是这两种情况中的哪一种。

关于欠拟合和过拟合我们之前已经学习过，接下来不再赘述，用下图来解释：



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



其中  $d$  表示多项式的最高项数的值

训练误差计算步骤为： $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$ 。

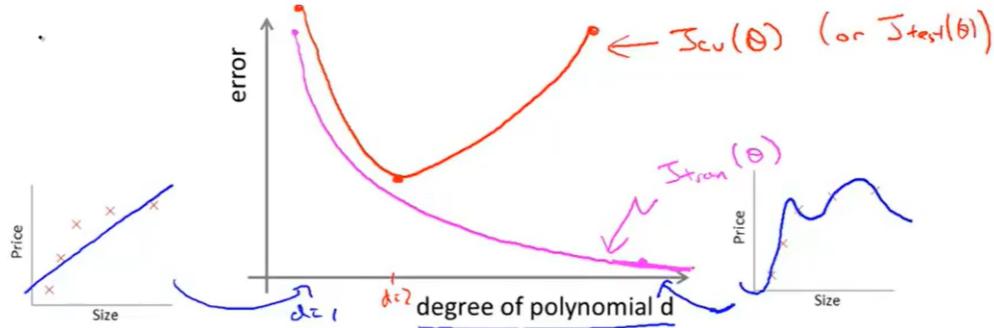
交叉验证误差的计算步骤为： $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^m (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ 。

我们的算法检验步骤可以如下图所示：

### Bias/variance

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

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



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

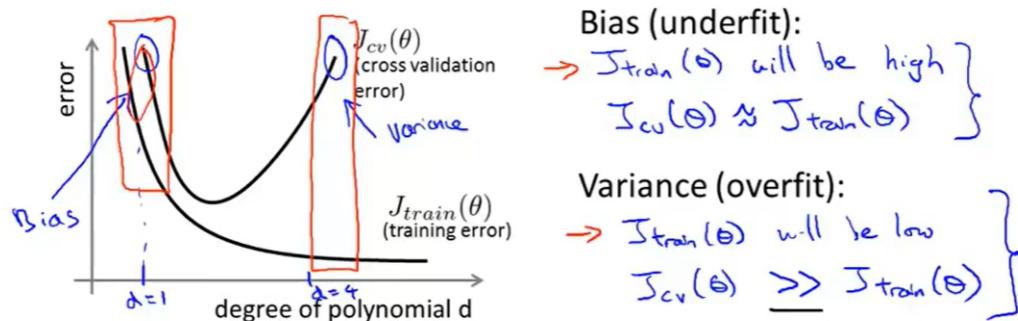
对于交叉验证集，当  $d$  较小时，模型拟合程度低，误差较大；但是随着  $d$  的增长，误差呈现先减小后增大的趋势，转折点是我们的模型开始过拟合训练数据集的时候。

如果我们的交叉验证集误差较大，我们如何判断是方差还是偏差呢？

根据上面的图表，我们知道：

### Diagnosing bias vs. variance

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



综上所述，我们可以这样总结：

训练集误差和交叉验证集误差近似时：偏差/欠拟合；

交叉验证集误差远大于训练集误差时：方差/过拟合。

## 1.5 正则化和偏差/方差

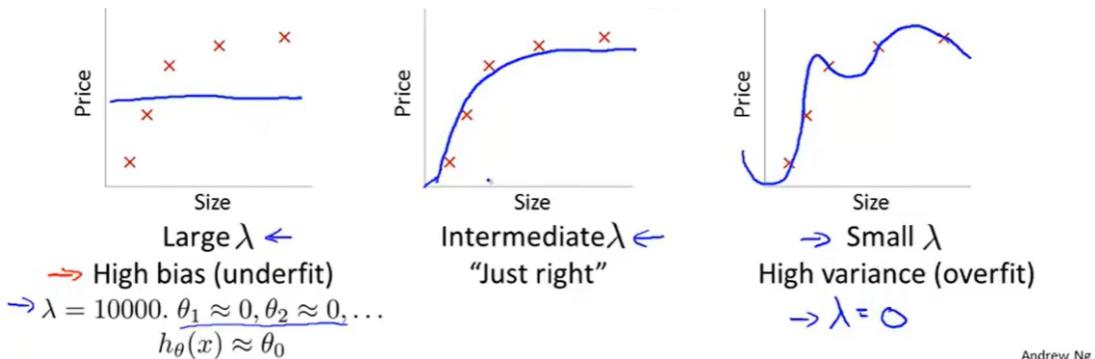
在我们在训练模型的过程中，一般会使用一些正则化方法来防止过拟合。

但是我们可能会正则化的程度太高或太小了，即我们在选择  $\lambda$  的值时也需要思考与刚才选择多项式模型次数类似的问题。

假设我们要对这样一个高阶的多项式进行拟合，多项式为  $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$ ，为了防止过拟合，我们要使用正则化方法，此时的代价函数如下：

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

然后我们分析以下三种情况：



如上图所示， $\lambda$  过大时，拟合的函数会是一条直线，当  $\lambda$  过小时，则会出现过拟合的情况。

我们选择一系列的想要测试的  $\lambda$  值，通常是 0—10 之间的呈现 2 倍关系的值，如 0.01, 0.02, 0.04, 0.08, 0.15, 0.32, 0.64, 1.28, 2.56, 5.12, 10。

我们同样把数据分为训练集、交叉验证集和测试集。

### Choosing the regularization parameter $\lambda$

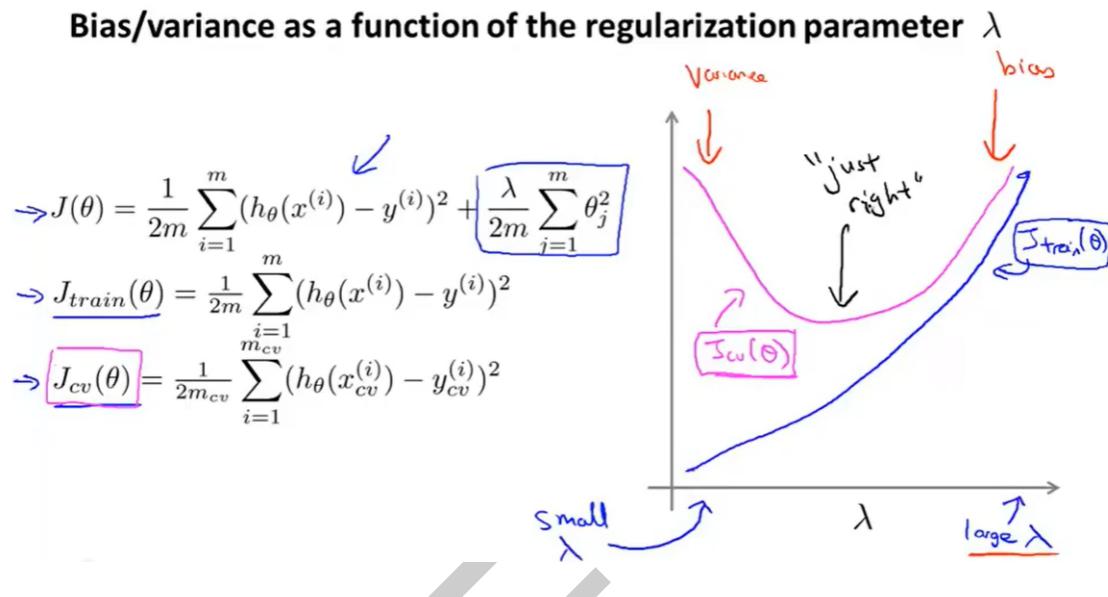
**Model:**  $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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

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

选择  $\lambda$  的方法为：

1. 使用训练集训练出 12 个不同程度正则化的模型
2. 用 12 个模型分别对交叉验证集计算出交叉验证误差的值
3. 选择交叉验证误差最小的模型作为我们最终使用的模型
4. 运用步骤 3 中选出模型对测试集计算得出推广误差，我们也可以同时将训练集和交叉验证集模型的代价函数误差与  $\lambda$  的值绘制在一张图表上：



当  $\lambda$  较小时，训练集误差较小（过拟合），而交叉验证集误差较大。

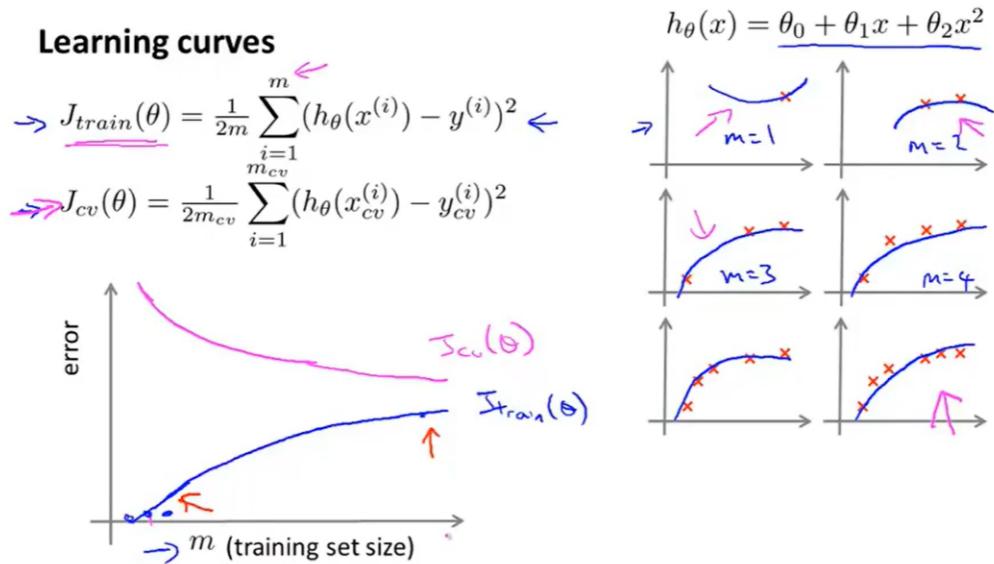
随着  $\lambda$  的增加，训练集误差不断增加（欠拟合），而交叉验证集误差则是先减小后增加。

## 1.6 学习曲线

学习曲线就是一种很好的工具，可以使用学习曲线来判断某一个学习算法是否处于偏差、方差问题，或者是两种问题皆有。

学习曲线是将训练集误差和交叉验证集误差作为训练集样本数量的函数绘制的图表。

思想是：当训练较少行数据的时候，训练的模型将能够非常完美地适应较少的训练数据，但是训练出来的模型却不能很好地适应交叉验证集数据或测试集数据。



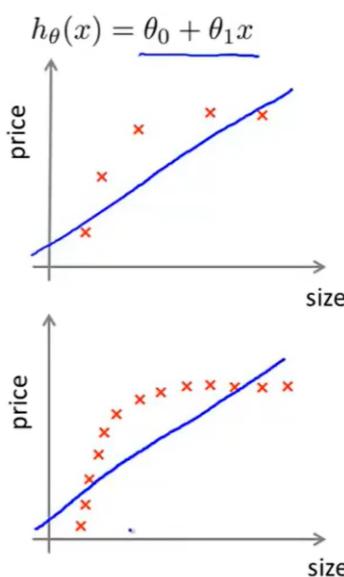
当  $m$  很小时，即当训练样本很少时，对每一个训练样本都能很容易的拟合到很好，所以训练误差将会很小。

反过来，当  $m$  的值逐渐增大时，那么想对每一个样本拟合到很好就显得愈发的困难了，训练集误差就会越来越大。

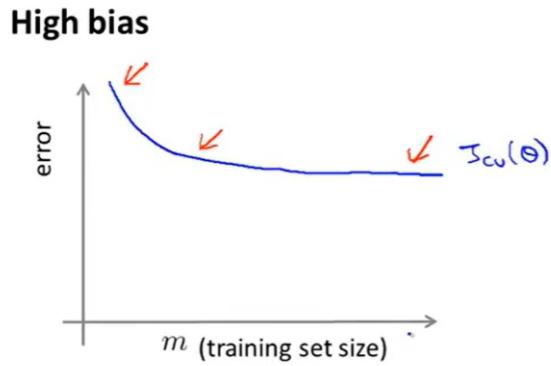
而交叉验证集的误差正好与训练集误差的情况相反。

### 1.6.1 如何利用学习曲线识别高偏差/欠拟合

作为例子，我们尝试用一条直线来适应下面的数据：



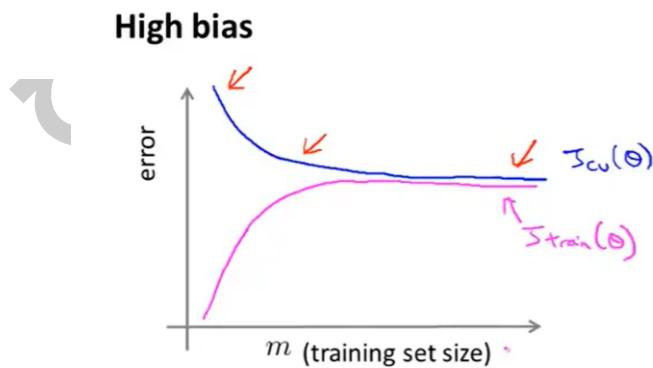
可以看出，无论训练集有多么大，这条直线都不会有太大改观。



接下来看一下交叉验证集误差：

当样本很小时，误差很大；当样本数量逐渐增大时，误差会逐渐减少。

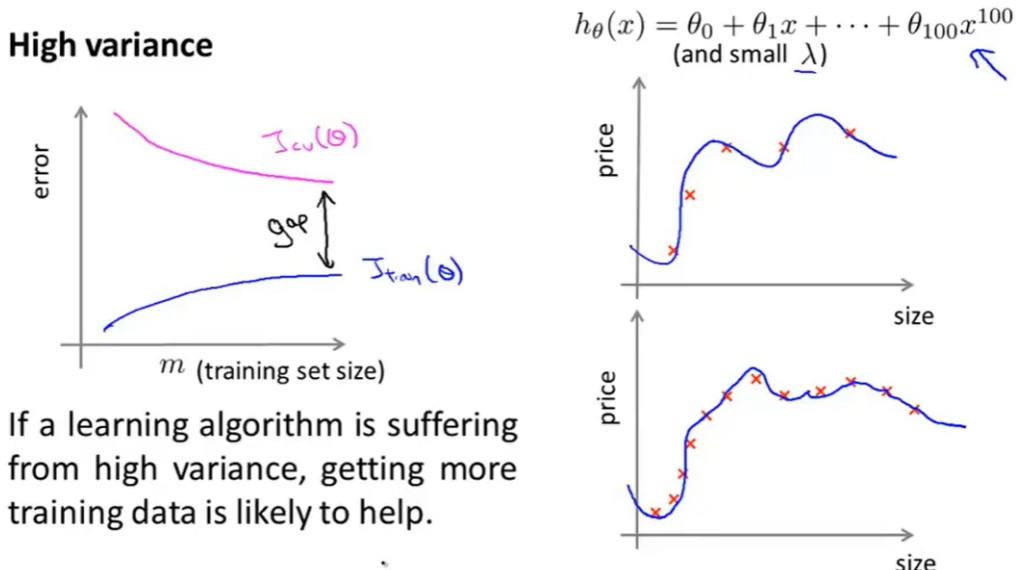
训练集误差则正好相反。



也就是说在高偏差/欠拟合的情况下，增加数据到训练集不一定能有帮助。

### 1.6.2 如何利用学习曲线识别高方差/过拟合

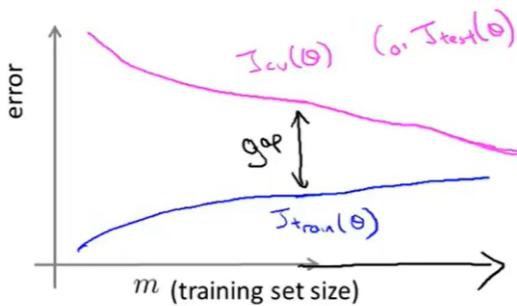
假设我们使用一个非常高次的多项式模型，并且正则化非常小，可以看出，当交叉验证集误差远大于训练集误差时，往训练集增加更多数据可以提高模型的效果。



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



### High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help. ↩

算法处于高方差情形时，最明显的一个特点是，在训练误差和交叉验证误差之间有一段很大的差距。但随着训练数据的增多，两条学习曲线会逐渐靠近。

## 1.7 决定下一步做什么

我们前面已经学习了如何去评价一个学习算法，并且讨论了模型选择问题，偏差和方差的问题。

这些诊断法则怎样帮助我们判断，哪些方法可能有助于改进学习算法的效果，而哪些可能是徒劳的呢？

回顾上面的回归模型预测房价，我们提出的六种方法。

1. 获得更多的训练样本——解决高方差
2. 尝试减少特征的数量——解决高方差
3. 尝试获得更多的特征——解决高偏差
4. 尝试增加多项式特征——解决高偏差
5. 尝试减少正则化程度  $\lambda$ ——解决高偏差
6. 尝试增加正则化程度  $\lambda$ ——解决高方差

当我们在进行神经网络拟合时，我们可以选择一个相对简单的神经网络模型，隐藏单元比较少的，甚至只有一个隐藏单元。像这样一个简单的神经网络模型，参数不会很多，很容易出现欠拟合和高偏差。

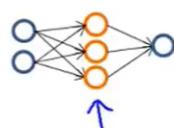
这种比较小型的神经网络的最大优势在于计算量较少。

我们也可以选择一个较大型的神经网络模型，比如每一层中的隐藏单元数很多，或者有很多个隐藏层。这种比较复杂的神经网络模型，参数一般较多，更容易出现过拟合和高方差的情况。

这种比较大型的神经网络的特点是计算量较大。但是可以通过正则化手段来调整而更加适应数据。

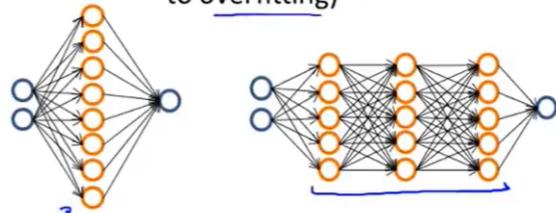
### Neural networks and overfitting

→ “Small” neural network  
(fewer parameters; more prone to underfitting)



Computationally cheaper

→ “Large” neural network  
(more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization ( $\lambda$ ) to address overfitting.

通常选择较大的神经网络并采用正则化处理会比采用较小的神经网络效果要好。

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

# 第二章 机器学习系统的设计

## 2.1 首先要做什么

这一节的学习以一个垃圾邮件分类器算法为例进行讨论。

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

为了构建这个分类器算法，我们可以做很多事，例如：

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

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

当研究者想着用不同方法来尝试去提高精度的时候，可能已经超越了很多人了。

## 2.2 误差分析

我们将会讲到误差分析（*Error Analysis*）的概念。这会帮助我们更系统地做出决定。

如果我们要准备研究机器学习的东西，或者构造机器学习应用程序，最好的实践方法不是建立一个非常复杂的系统，拥有多么复杂的变量；而是构建一个简单的算法，这样我们可以很快地实现它。

每当吴恩达老师研究机器学习的问题时，他最多只会花一天的时间，就是字面意义上的 24 小时，来试图很快的把结果搞出来，即便效果不好。坦白的说，就是根本没有用复杂的系统，但是只是很快的得到的结果。即便运行得不完美，但是也把它运行一遍，

最后通过交叉验证来检验数据。

一旦做完模型，我们就可以画出模型的学习曲线。通过画出学习曲线，以及检验误差，来找出自己的算法是否有高偏差和高方差的问题，或者别的问题。

在这样分析之后，再来决定用更多的数据训练，或者加入更多的特征变量是否有用。

这么做的原因是：这在我们刚接触机器学习问题时是一个很好的方法，我们刚开始并不能提前知道自己是否需要复杂的特征变量，或者是否需要更多的数据，还是别的什么。

提前知道自己应该做什么，是非常难的，因为缺少证据，缺少学习曲线。

因此，我们很难知道你应该把时间花在什么地方来提高算法的表现。

但是当我们实践一个非常简单即便不完美的方法时，也可以通过画出学习曲线来做出进一步的选择。我们可以用这种方式来避免一种电脑编程里的过早优化问题，这种理念是：我们必须用证据来领导我们的决策，怎样分配自己的时间来优化算法，而不是仅仅凭直觉，凭直觉得出的东西一般总是错误的。

除了画出学习曲线之外，一件非常有用的事是误差分析。

意思是说：当我们在构造垃圾邮件分类器时，我们可以去看一看我们的交叉验证数据集，然后亲自看一看哪些邮件被算法错误地分类。

因此，通过这些被算法错误分类的垃圾邮件与非垃圾邮件，我们可以发现某些系统性的规律：什么类型的邮件总是被错误分类。

经常地这样做之后，这个过程能启发我们去构造新的特征变量，或者告诉我们：现在这个系统的短处，然后启发如何去提高它。

构建一个学习算法的推荐方法为：

1. 从一个简单的能快速实现的算法开始，实现该算法并用交叉验证集数据测试这个算法

2. 绘制学习曲线，决定是增加更多数据，或者添加更多特征，还是其他选择

3. 进行误差分析：人工检查交叉验证集中我们算法中产生预测误差的样本，看看这些样本是否有某种系统化的趋势

以我们的垃圾邮件过滤器为例，误差分析要做的既是检验交叉验证集中我们的算法产生错误预测的所有邮件，看：是否能将这些邮件按照类分组。例如药品垃圾邮件，仿冒品垃圾邮件或者密码窃取邮件等。

然后看分类器对哪一组邮件的预测误差最大，并着手优化。思考怎样能改进分类器。例如，发现是否缺少某些特征，记下这些特征出现的次数。例如记录下错误拼写出现了多少次，异常的邮件路由情况出现了多少次等等，然后从出现次数最多的情况开始着手

优化。

事实上误差分析并不总能帮助我们判断应该采取怎样的行动。

有时我们需要尝试不同的模型，然后进行比较，在模型比较时，用数值来判断哪一个模型更好更有效，通常我们是看交叉验证集的误差。

在我们的垃圾邮件分类器例子中，对于“我们是否应该将 discount/discounts/discounted/directly 处理成同一个词？”如果这样做可以改善我们算法，我们会采用一些截词软件。

误差分析不能帮助我们做出这类判断，我们只能尝试采用和不采用截词软件这两种不同方案，然后根据数值检验的结果来判断哪一种更好。

因此，当我们在构造学习算法的时候，我们总是会去尝试很多新的想法，实现出很多版本的学习算法；如果每一次实践新想法的时候，我们都要手动地检测这些例子，去看看是表现差还是表现好，那么这很难让我们做出决定。比如说到底是否使用词干提取，是否区分大小写。

但是通过一个量化的数值评估，我们可以通过查看这个数字，来做出误差是变大还是变小了的结论。我们可以通过它更快地实践自己的新想法，它基本上非常直观地告诉我们：自己的想法是提高了算法表现，还是让它变得更坏，这会大大提高实践算法时的速度。

所以吴恩达老师强烈推荐在交叉验证集上来实施误差分析，而不是在测试集上。

即使这从数学上讲是不合适的。但是吴恩达老师还是推荐我们在交叉验证向量上来做误差分析。

总结一下，当我们在研究一个新的机器学习问题时，老师们总是推荐自己实现一个较为简单快速、即便不是那么完美的算法。但几乎从未见过人们这样做。大家经常干的事情是：花费大量的时间在构造算法上，构造他们以为的简单的方法。因此，不要担心自己的算法太简单，或者太不完美，而是尽可能快地实现自己的算法。当自己有了初始的实现之后，它会变成一个非常有力的工具，来帮助我们决定下一步的做法。因为我们可以先看看算法造成的错误，通过误差分析，来看看他犯了什么错，然后来决定优化的方式。

另一件事是：假设我们有了一个快速而不完美的算法实现，又有一个数值的评估数据，这会帮助我们尝试新的想法，快速地发现我们尝试的这些想法是否能够提高算法的表现，从而我们会更快地做出决定，在算法中放弃什么，吸收什么误差分析可以帮助我们系统化地选择该做什么。

## 2.3 类偏斜的误差度量

本节将会考虑设定某个实数来评估我们的学习算法，并衡量它的表现，有了算法的评估和误差度量值。

有一件重要的事情要注意，就是使用一个合适的误差度量值，这有时会对于我们的学习算法造成非常微妙的影响，这件重要的事情就是偏斜类 (*skewed classes*) 的问题。

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

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

这种情况下，误差的大小是不能视为评判算法效果的依据的。我们引入查准率 (*Precision*) 和查全率 (*Recall*) 的概念，将算法预测的结果分成四种情况如下：

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

则有下面的式子：

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

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

下图是这四种情况的对照表。

		预测值	
		<b>Positive</b>	<b>Negative</b>
<b>实际值</b>	<b>Positive</b>	<b>TP</b>	<b>FN</b>
	<b>Negative</b>	<b>FP</b>	<b>TN</b>

## 2.4 查准率和查全率之间的权衡

在很多应用中，我们希望能够保证查准率和召回率的相对平衡。在这节中，将展示一些查准率和召回率作为算法评估度量值的更有效的方式。

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

### Trading off precision and recall

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

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

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

Suppose we want to predict  $y = 1$  (cancer)

only if very confident.

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

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

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

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

如果我们希望只在非常确信的情况下预测为真（肿瘤为恶性），即我们希望更高的查准率，我们可以使用比 0.5 更大的阀值，如 0.7, 0.9。

这样做我们会减少错误预测病人为恶性肿瘤的情况，同时却会增加未能成功预测肿瘤为恶性的情况。

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

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

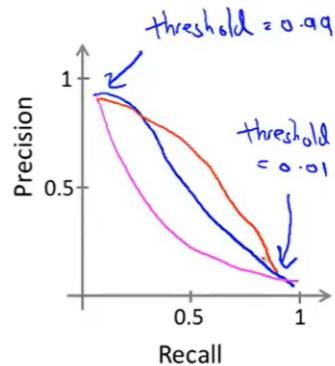
### Trading off precision and recall

- Logistic regression:  $0 \leq h_\theta(x) \leq 1$
- Predict 1 if  $h_\theta(x) \geq 0.5$  ~~0.7~~ ~~0.9~~ ~~0.3~~ ↗
- Predict 0 if  $h_\theta(x) < 0.5$  ~~0.7~~ ~~0.9~~ ~~0.3~~
- Suppose we want to predict  $y = 1$  (cancer) only if very confident.  
→ Higher precision, lower recall.
- Suppose we want to avoid missing too many cases of cancer (avoid false negatives).  
→ Higher recall, lower precision.

More generally: Predict 1 if  $h_\theta(x) \geq \text{threshold}$ .

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

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



我们希望有一个帮助我们选择这个阀值的方法。

一种方法是计算  $F1$  值 ( $F1\ Score$ ), 其计算公式为:

$$F1\ Score = 2 \frac{PR}{P + R}$$

我们选择使得  $F1$  值最高的阀值。

## 2.5 机器学习的数据

在之前的一些视频中, 吴恩达老师曾告诫大家不要盲目地开始, 而是花大量的时间来收集大量的数据, 因为数据有时是唯一能实际起到作用的。

得到大量的数据并在某种类型的学习算法中进行训练, 可以是一种有效的方法来获得一个具有良好性能的学习算法。而这种情况往往出现在这些条件对于我们的问题都成立。

并且我们能够得到大量数据的情况下, 这可以是一个很好的方式来获得非常高性能的学习算法。

多年前, 两位研究人员进行了一项有趣的研究, 他们尝试通过机器学习算法来区分常见的易混淆的单词, 他们尝试了许多种不同的算法, 并发现数据量非常大时, 这些不同类型的算法效果都很好。问题如下所示:

### Designing a high accuracy learning system

E.g. Classify between confusable words.

{to, two, too}, {then, than}

For breakfast I ate \_\_\_\_\_ eggs.

Algorithms

- Perceptron (Logistic regression)
- Winnow
- Memory-based
- Naïve Bayes

比如，在这样的句子中：早餐我吃了〇个鸡蛋（可选 *to, two, too*）。在这个例子中，“早餐我吃了 2 个鸡蛋”，这是一个易混淆的单词的例子。

于是那两位研究人员把诸如这样的机器学习问题，当做一类监督学习问题，并尝试将其分类，什么样的词，在一个英文句子特定的位置，才是合适的。

他们用了几种不同的学习算法，这些算法都是在他们 2001 年进行研究的时候，都已经被公认是比较领先的。

因此他们使用了一个方差，用于逻辑回归上的一个方差，被称作“感知器”(perceptron)。

他们也采取了一些过去常用，但是现在比较少用的算法，比如 Winnow 算法，很类似于回归问题，但在一些方面又有所不同，过去用得比较多，但现在用得不多。

还有一种基于内存的学习算法，现在也用得比较少了，而且他们用了一个朴素算法。

这些具体算法的细节不那么重要，我们下面希望探讨，什么时候我们会希望获得更多数据，而非修改算法。他们所做的就是改变了训练数据集的大小，并尝试将这些学习算法用于不同大小的训练数据集中，这就是他们得到的结果：

### Designing a high accuracy learning system

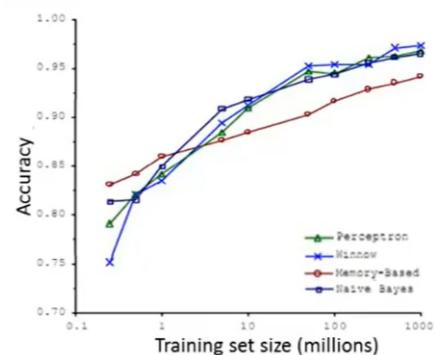
E.g. Classify between confusable words.

{to, two, too} {then, than}

→ For breakfast I ate two eggs.

Algorithms

- - Perceptron (Logistic regression)
- - Winnow
- - Memory-based
- - Naïve Bayes



这些趋势非常明显，首先大部分算法，都具有相似的性能，其次，随着训练数据集的增大，在横轴上代表以百万为单位的训练集大小，从 0.1 个百万到 1000 百万，也就是到了 10 亿规模的训练集的样本，这些算法的性能也都对应地增强了。

事实上，如果我们选择任意一个算法，可能是选择了一个“劣等的”算法，如果我们给这个劣等算法更多的数据，那么从这些例子中看起来的话，它看上去很有可能会其他算法更好，甚至会比“优等算法”更好。

由于这项原始的研究非常具有影响力，因此已经有一系列许多不同的研究显示了类似的结果。

这些结果表明，许多不同的学习算法有时倾向于表现出非常相似的表现，这还取决于一些细节，但是真正能提高性能的，是我们能够给一个算法大量的训练数据。

像这样的结果，引起了一种在机器学习中的普遍共识：“取得成功的人不是拥有最好算法的人，而是拥有最多数据的人”。

那么这种说法在什么时候是真，什么时候是假呢？

因为如果我们有一个学习算法，并且如果这种说法是真的，那么得到大量的数据通常是保证我们具有一个高性能算法的最佳方式，而不是去争辩应该用什么样的算法。

假如有这样一些假设，在这些假设下有大量我们认为有用的数据集。我们假设在我们的机器学习问题中，特征值  $x$  包含了足够的信息，这些信息可以帮助我们用来准确地预测  $y$ 。例如，如果我们采用了一些容易混淆的词，如：*two*、*to*、*too*，假如说它能够描述  $x$ ，捕捉到需要填写的空白处周围的词语，那么特征捕捉到之后，我们就希望有对于“早饭我吃了 () 个鸡蛋”，那么这就有大量的信息来告诉我中间我需要填的词是“两个”(*two*)，而不是单词 *to* 或 *too*，因此特征捕捉，哪怕是周围词语中的一个词，就能够给我足够的信息来确定出标签  $y$  是什么。

换句话说，从这三组易混淆的词中，我应该选什么词来填空。

那么让我们来看一看，大量的数据是有帮助的情况。

假设特征值有足够的信息来预测  $y$  值，假设我们使用一种需要大量参数的学习算法，比如有很多特征的逻辑回归或线性回归，或者用带有许多隐藏单元的神经网络，那又是另外一种带有很多参数的学习算法，这些都是非常强大的学习算法，它们有很多参数，这些参数可以拟合非常复杂的函数，因此我们要调用这些，我们将把这些算法想象成低偏差算法，因为我们能够拟合非常复杂的函数，而且因为我们有非常强大的学习算法，这些学习算法能够拟合非常复杂的函数。

很有可能，如果我们用这些数据运行这些算法，这种算法能很好地拟合训练集，因此，训练误差就会很低了。

现在假设我们使用了非常非常大的训练集，在这种情况下，尽管我们希望有很多参数，但是如果训练集比参数的数量还大，甚至是更多，那么这些算法就不太可能会过度拟合。也就是说训练误差有希望接近测试误差。

另一种考虑这个问题的角度是为了有一个高性能的学习算法，我们希望它不要有高的偏差和方差。因此偏差问题，我们将通过确保有一个具有很多参数的学习算法来解决，以便我们能够得到一个较低偏差的算法，并且通过用非常大的训练集来保证。

### Large data rationale

- Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→  $J_{train}(\theta)$  will be small.

- Use a very large training set (unlikely to overfit) low variance ←

→  $J_{train}(\theta) \approx J_{test}(\theta)$

→  $J_{test}(\theta)$  will be small

我们在此没有方差问题，我们的算法将没有方差，并且通过将这两个值放在一起，我们最终可以得到一个低误差和低方差的学习算法。这使得我们能够很好地测试测试数据集。

从根本上来说，这是一个关键的假设：特征值有足够的信息量，且我们有一类很好的函数，这是为什么能保证低误差的关键所在。

它有大量的训练数据集，这能保证得到更多的方差值，因此这给我们提出了一些可能的条件，如果你有大量的数据，而且你训练了一种带有很多参数的学习算法，那么这将会是一个很好的方式，来提供一个高性能的学习算法。

我觉得关键的测试：

首先，一个专家看到了特征值  $x$ ，能很有信心的预测出  $y$  值吗？

因为这可以证明  $y$  可以根据特征值  $x$  被准确地预测出来。

其次，我们实际上能得到一组庞大的训练集，并且在这个训练集中训练一个有很多参数的学习算法吗？如果你不能做到这两者，那么更多时候，你会得到一个性能很好的学习算法。