第 6 周

9、 应用机器学习的建议 Advice for Applying Machine Learning

9.1 决定下一步做什么 Deciding What to Try Next

我想做的是确保你在设计机器学习的系统时，你能够明白怎样选择一条最合适、最正确的道路。因此，在这节视频和之后的几段视频中，我将向你介绍一些实用的建议和指导，帮助你明白怎样进行选择。具体来讲，我将重点关注的问题是假如你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，你应如何决定接下来应该选择哪条道路？为了解释这一问题，我想仍然使用预测房价的学习例子，假如你已经完成了正则化线性回归，也就是最小化代价函数J的值，假如，在你得到你的学习参数以后，如果你要将你的假设函数放到一组新的房屋样本上进行测试，假如说你发现在预测房价时产生了巨大的误差，现在你的问题是要想改进这个算法，接下来应该怎么办？

实际上你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，也许你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。遗憾的是，我看到好多人花费了好多时间想收集更多的训练样本。他们总认为，要是我有两倍甚至十倍数量的训练数据，那就一定会解决问题的是吧？但有时候获得更多的训练数据实际上并没有作用。在接下来的几段视频中，我们将解释原因。

我们也将知道怎样避免把过多的时间浪费在收集更多的训练数据上，这实际上是于事无补的。另一个方法，你也许能想到的是尝试选用更少的特征集。因此如果你有一系列特征比如x_1, x_2, x_3等等。也许有很多特征，也许你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。或者也许你需要用更多的特征，也许目前的特征集，对你来讲并不是很有帮助。你希望从获取更多特征的角度来收集更多的数据，同样地，你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，我们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的平方，x_2的平方，x_1, x_2的乘积，我们可以花很多时间来考虑这一方法，我们也可以考虑其他方法减小或增大正则化参数λ的值。我们列出的这个单子，上面的很多方法都可以扩展开来扩展成一个六个月或更长时间的项目。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，比如他们会说“噢，我们来多找点数据吧”，然后花上六个月的时间收集了一大堆数据，然后也许另一个人说：“好吧，让我们来从这些房子的数据中多找点特征吧”。我很遗憾不止一次地看到

很多人花了至少六个月时间来完成他们随便选择的一种方法，而在六个月或者更长时间后，他们很遗憾地发现自己选择的是一条不归路。幸运的是，有一系列简单的方法能让你事半功倍，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，只要你使用，就能很轻松地排除掉很多选择，从而为你节省大量不必要花费的时间。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？

获得更多的训练实例——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。

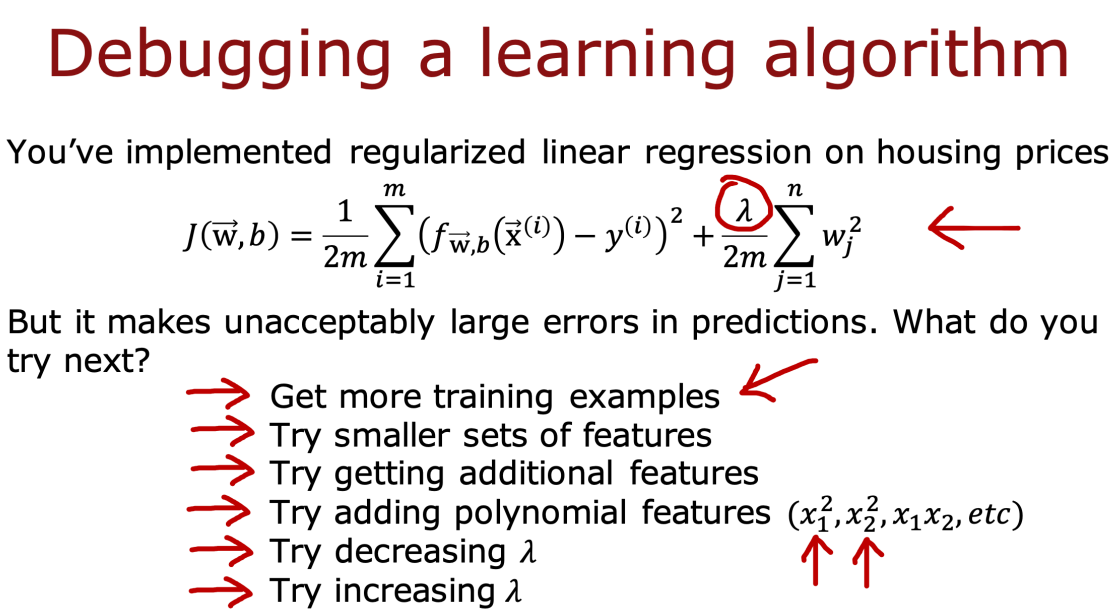
1.尝试减少特征的数量

2.尝试获得更多的特征

3.尝试增加多项式特征

4.尝试减少正则化程度λ

5.尝试增加正则化程度λ



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. Model Evaluation Challenge:
   - Difficult to assess complex models (e.g., with multiple features) visually.
   - Need systematic methods for evaluation, especially when it's hard to plot functions with multiple features.

2. Training and Test Sets:
    - Split dataset into training (70-80%) and test sets (20-30%).
    - Train model parameters on the training set.
    - Evaluate model performance on the test set.
3. Notation and Definitions:
    - m train: Number of training examples.
    - m test: Number of test examples.
    - Training examples: $(x1, y1), \ldots, \left(x^{\text{mtrain}}, y^{\text{mtrain}}\right)$.
    - Test examples: $(x^{1,\text{ test}}, y^{1,\text{ test}}), \ldots, (x^{m\text{ test},\text{ test}}, y^{\text{mtest},\text{ test}})$.
4. Evaluating Regression Models:
    - Use a cost function like squared error with regularization for training.
    - Compute test error $(J_{\text{test}}(w, b))$ as the average **squared** error on the test set (**without** regularization).
    - Compute training error $(J_{\text{train}}(w, b))$ as the average squared error on the training set (without regularization).
5. Evaluating Classification Models:
    - Minimize a cost function (e.g., logistic loss with regularization) for training.
    - For test and training errors, use either:
    - Logistic loss over the respective sets.
    - Fraction of misclassified examples in the test and training sets.
6. Interpreting Errors:
    - Low training error and high test error indicate overfitting (poor generalization).
    - Helps in understanding the model's ability to generalize to unseen data.
7. Decision Making:
    - The approach helps in choosing the right model complexity.
    - Further refinement needed for automated model selection.

9.2 评估一个假设 Evaluating a Model

Machine learning diagnostic

Diagnostic: A test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

Diagnostics can take time to implement but doing so can be a very good use of your time.

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。

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：

1.对于线性回归模型，我们利用测试集数据计算代价函数J

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：

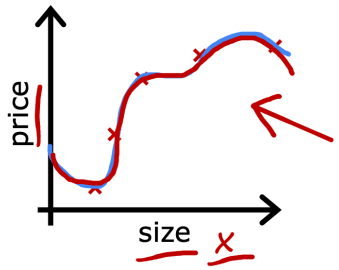
$$J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \log h_\theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log h_\theta(x_{test}^{(i)})$$

误分类的比率，对于每一个测试集实例，计算：

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

然后对计算结果求平均。

Evaluating your model



→ Model fits the training data well but will fail to generalize to new examples not in the training set.

$x = x_1$

$f_{\vec{w},b}(\vec{x}) = w_1 x + w_2 x^2 + \cdots + w_n x^n + b$

just x

$W_n x^4$

→ x_1 = size in feet²
→ x_2 = no. of bedrooms
→ x_3 = no. of floors
→ x_4 = age of home in years

$f(\vec{x})$

Dataset:

	size	price	
70%	2104	400	training set
	1600	330	
	2400	369	
	1416	232	
	3000	540	
	1985	300	
	1534	315	
30%	1427	199	test set
	1380	212	
	1494	243	

$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$\vdots$
$(x^{(m_{train})}, y^{m_{train}})$

m_{train} = no. training examples = 7

$(x_{test}^{(1)}, y_{test}^{(1)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. test examples = 3

Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function $J(\vec{w}, b)$

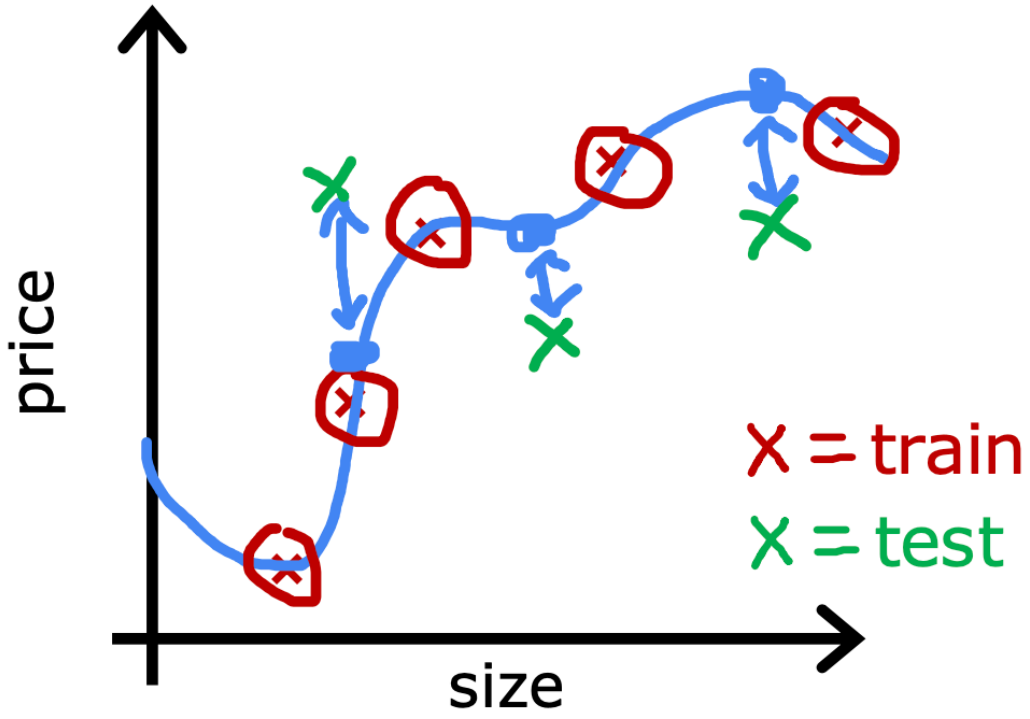
$$J(\vec{w}, b) = \min_{\vec{w}, b} \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)} \right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2 \right]$$

Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)} \right)^2 \right]$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}_{train}^{(i)}) - y_{train}^{(i)} \right)^2 \right]$$



$J_{train}(\vec{w}, b)$ will be low

$J_{test}(\vec{w}, b)$ will be high ←

Train/test procedure for classification problem

0/1

Fit parameters by minimizing $J(\vec{w}, b)$ to find $\vec{w}, b$

E.g.,

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

Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right) + \left(1 - y_{test}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right) \right]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y_{train}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right) + \left(1 - y_{train}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right) \right]$$

fraction of the test set and the fraction of the train set that the algorithm has misclassified.

$$\hat{y} = \begin{cases} 1 \text{ if } f_{\vec{w},b}\left(\vec{x}^{(i)}\right) \geq 0.5 \\ 0 \text{ if } f_{\vec{w},b}\left(\vec{x}^{(i)}\right) < 0.5 \end{cases}$$

count $\hat{y} \neq y$

$J_{test}(\vec{w}, b)$ is the fraction of the test set that has been misclassified.

$J_{train}(\vec{w}, b)$ is the fraction of the train set that has been misclassified.

9.3 模型选择和交叉验证集 Model Selection and Training/Cross Validation/Test Sets

假设我们要在 10 个不同次数的二项式模型之间进行选择：

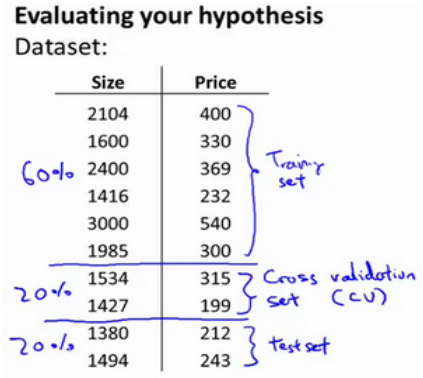
1. $h_\theta(x) = \theta_0 + \theta_1 x$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$

$\vdots$

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$

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。

即：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



模型选择的方法为：

1. 使用训练集训练出 10 个模型
2. 用 10 个模型分别对 cross validationset 计算得出 cross validation error（代价函数的值）
3. 选取代价函数值最小的模型 based on validation error
4. 用步骤 3 中选出的模型对 test set 计算得出 generalization error

Train/validation/test error

Training/cross validation/test set

Training error: $J_{train}(\vec{w}, b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2\right]$

Cross validation error: $J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}\left(\vec{x}_{cv}^{(i)}\right) - y_{cv}^{(i)}\right)^2\right]$ (validation error, dev error)

Test error: $J_{test}(\vec{w}, b) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) - y_{test}^{(i)}\right)^2\right]$

Model selection

d=1 1. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$ → $w^{<1>}, b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$

d=2 2. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + b$ → $w^{<2>}, b^{<2>}$ → $J_{cv}(w^{<2>}, b^{<2>})$

d=3 3. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + w_3x^3 + b$

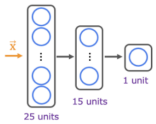
⋮

d=10 10. $f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x^2 + \cdots + w_{10}x^{10} + b$ → $w^{<10>}, b^{<10>}$ → $J_{cv}(w^{<10>}, b^{<10>})$

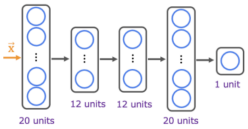
→ Pick $w_1x_1 + \cdots + w_4x^4 + b$ ($J_{cv}(w^{<4>}, b^{<4>})$)

Estimate generalization error using test the set: $J_{test}(w^{<4>}, b^{<4>})$

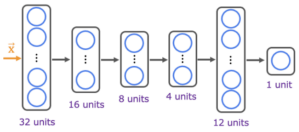
Model selection – choosing a neural network architecture

1. 

$w^{(1)}, b^{(1)}$ $J_{cv}(\mathbf{W}^{(1)}, \mathbf{B}^{(1)})$

→ 2. 

$w^{(2)}, b^{(2)}$ $J_{cv}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})$

3. 

$w^{(3)}, b^{(3)}$ $J_{cv}(\mathbf{W}^{(3)}, \mathbf{B}^{(3)})$

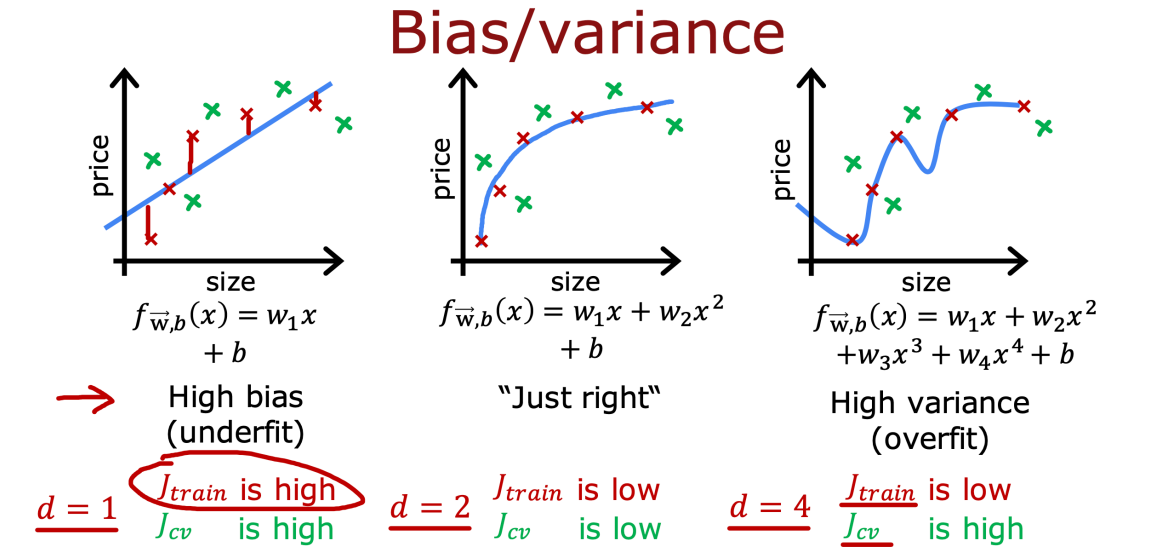
Train, CV

Pick $\mathbf{W}^{(2)}, \mathbf{B}^{(2)}$

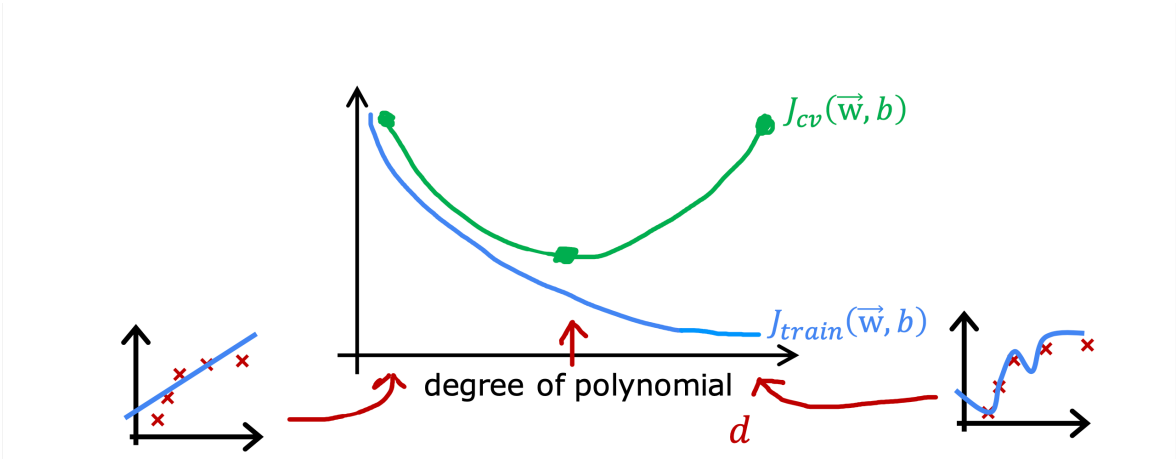
Estimate generalization error using the test set: $J_{test}(\mathbf{W}^{(2)}, \mathbf{B}^{(2)})$

9.4 诊断偏差和方差 Diagnosing Bias vs. Variance

当你运行一个学习算法时，如果这个算法的表现不理想，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。其实是一个很有效的指示器，指引着可以改进算法的最有效的方法和途径。在这段视频中，我想更深入地探讨一下有关偏差和方差的问题，希望你能对它们有一个更深入的理解，并且也能弄清楚怎样评价一个学习算法，能够判断一个算法是偏差还是方差有问题，因为这个问题对于弄清如何改进学习算法的效果非常重要，高偏差和高方差的问题基本上来说是欠拟合和过拟合的问题。

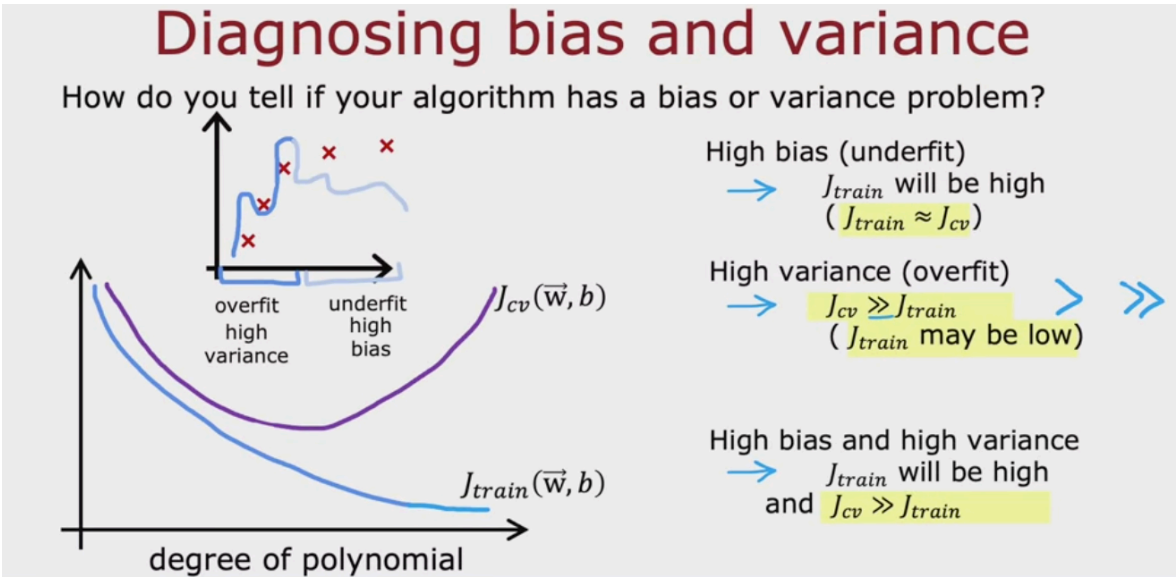


我们通常会通过将训练集和交叉验证集的代价函数误差与多项式的次数绘制在同一张图表上来帮助分析：



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

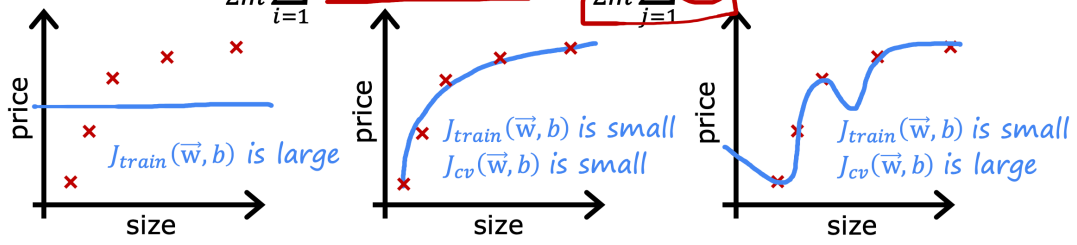


9.5 正则化和偏差/方差 Regularization and Bias_Variance

Linear regression with regularization

Model: $f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

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



Large λ
High bias (underfit)
$\lambda = 10{,}000$ $w_1 \approx 0, w_2 \approx 0$
$f_{\vec{w},b}(\vec{x}) \approx b$

λ

Intermediate λ

Small λ
High variance (overfit)
$\lambda = 0$

Choosing the regularization parameter λ

Model: $f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

1. Try $\lambda = 0$ → $\min_{\vec{w},b} J(\vec{w},b)$ → $w^{<1>}, b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$
2. Try $\lambda = 0.01$ → $w^{<2>}, b^{<2>}$ → $J_{cv}(w^{<2>}, b^{<2>})$
3. Try $\lambda = 0.02$ → $J_{cv}(w^{<3>}, b^{<3>})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $J_{cv}(w^{<5>}, b^{<5>})$

⋮

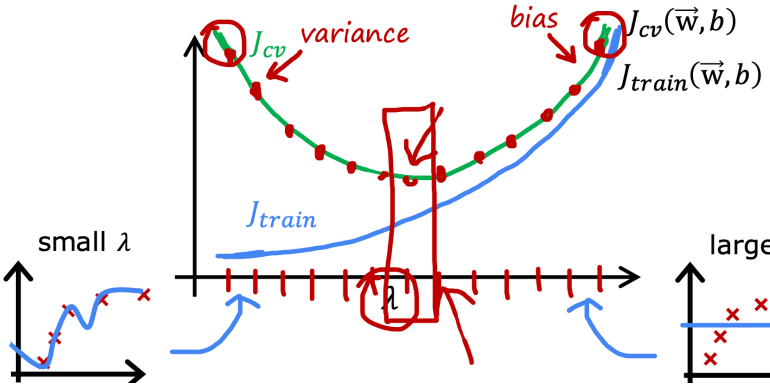
12. Try $\lambda \approx 10$ → $w^{<12>}, b^{<12>}$ → $J_{cv}(w^{<12>}, b^{<12>})$

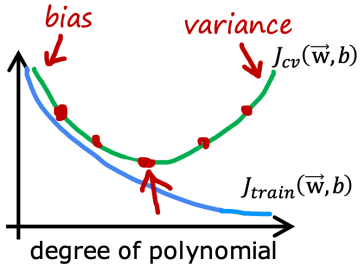
Pick $w^{<5>}, b^{<5>}$

Report test error: $J_{test}(w^{<5>}, b^{<5>})$

Bias and variance as a function of regularization parameter λ

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

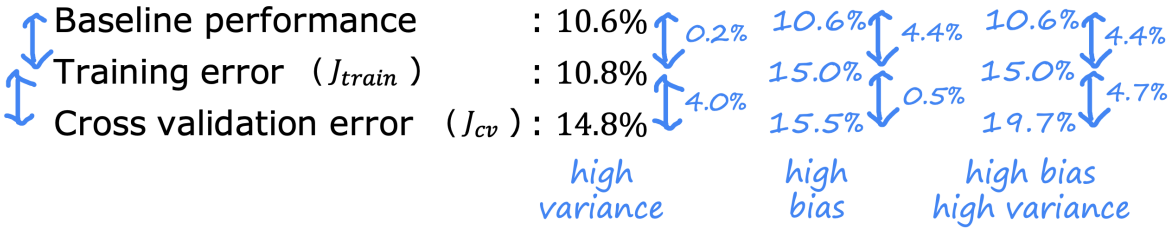




- 当 λ 较小时，训练集误差较小（过拟合）而交叉验证集误差较大
- 随着 λ 的增加，训练集误差不断增加（欠拟合），而交叉验证集误差则是先减小后增加

Establishing a baseline level of performance

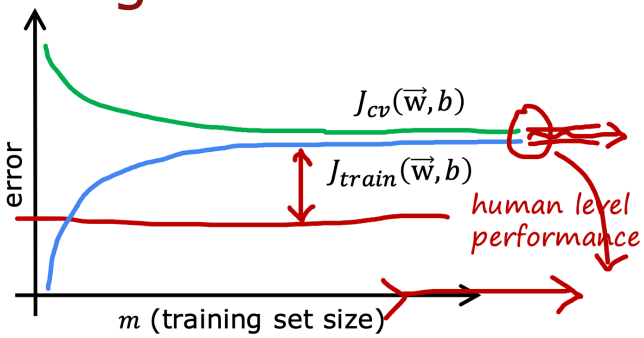
Bias/variance examples



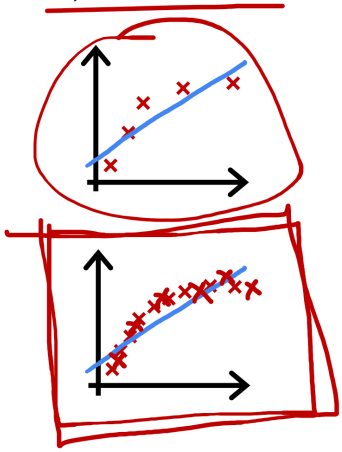
9.6 学习曲线 Learning Curves

1. Learning Curves Overview:
   - Learning curves depict how a model's performance evolves as the number of training examples increases.
   - They plot training and cross-validation errors against the number of training examples (m train).
2. Plotting Learning Curves:
   - The x-axis represents m train, the training set size.
   - The y-axis represents the error (either J_{cv} or J_{train}).
3. Learning Curve for a Quadratic Model:
   - As m train increases, J_{cv} typically decreases, indicating better generalization.
   - J train tends to increase with m train as it becomes harder to fit all training examples perfectly.
4. **High Bias (Underfitting) Scenario:**
   - With a simple model (e.g., linear fit for complex data), both J_{train} and J_{cv} are high.
   - Both errors plateau as m train increases, indicating limited learning from additional data.
   - **More training data won't significantly improve performance.**
5. **High Variance (Overfitting) Scenario:**
   - With a complex model (e.g., high-order polynomial), there is a large gap between J_{train} and $J_{\text{cv.}}$.
   - J_{train} is low, but J_{cr} is much higher and decreases slowly as m train increases.
   - **More training data can help reduce overfitting and improve performance.**
6. Practical Application of Learning Curves:
   - Visualizing learning curves helps diagnose bias and variance issues.
   - They guide decisions on whether adding more data is likely to improve model performance.
7. Computational Expense:
   -While informative, plotting learning curves can be computationally expensive due to training multiple models on varying sizes of training data.

High bias

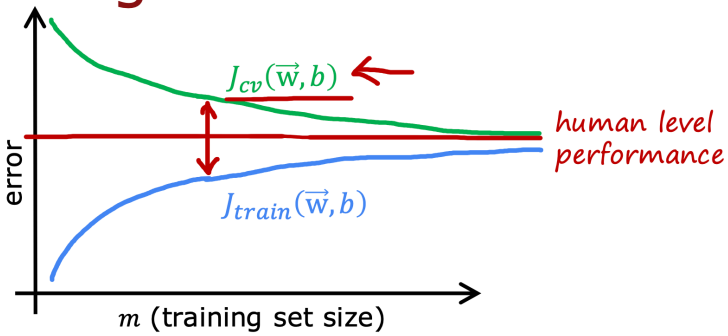


$f_{\vec{w},b}(x) = w_1 x + b$

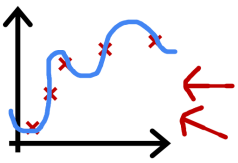


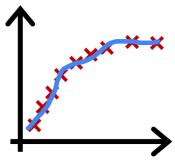
if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

High variance



$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$
(with small λ)

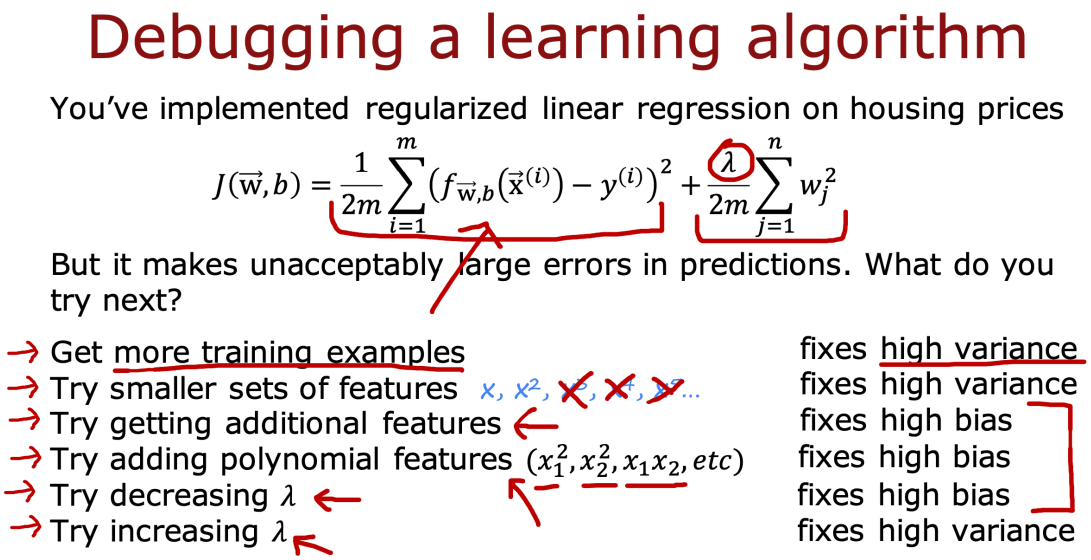




if a learning algorithm suffers from high variance, getting more training data is likely to help.

9.7 决定下一步做什么 Deciding What to Do Next Revisited

Note that decreasing λ means paying less attention to the regularization term but more on training set error



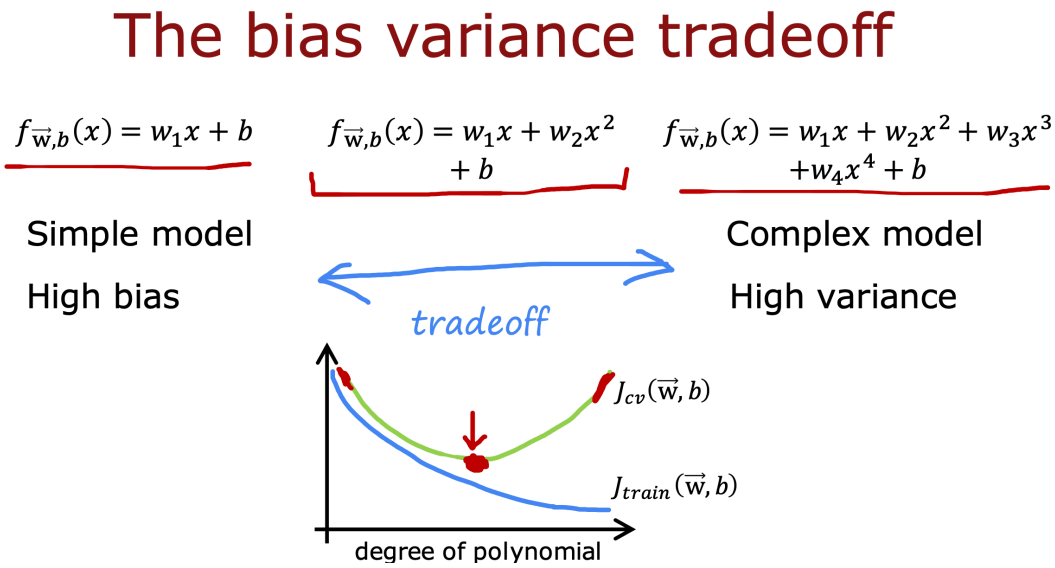
Bias/variance and neural networks

Summary of How Neural Networks Address High Bias and High Variance:

1. Bias-Variance Tradeoff in Traditional Machine Learning:
   - Traditional methods often involve a tradeoff between bias (underfitting) and variance (overfitting).
   - Example: Choosing the degree of a polynomial or the regularization parameter (λ).
2. Neural Networks as Low-Bias Machines:
   - Large neural networks, when trained on moderate-sized datasets, typically exhibit low bias.
   - This means they can fit the training set well, assuming the dataset isn't extremely large.
3. Recipe for Reducing Bias and Variance:
   - Step 1: Train the algorithm and check its performance on the training set (evaluate J_{train}).
   - If high bias is present (high J_{train}), use a bigger network.
   - Step 2: After reducing bias, check performance on the cross-validation set (evaluate J_{cv}).
   - If high variance is present (high J_{cv}), try getting more data.
   - Iterate through these steps until acceptable performance is achieved on both training and cross-validation sets.
4. Large Networks and Regularization:
   - Large neural networks are prone to overfitting, but this can be mitigated with appropriate regularization.

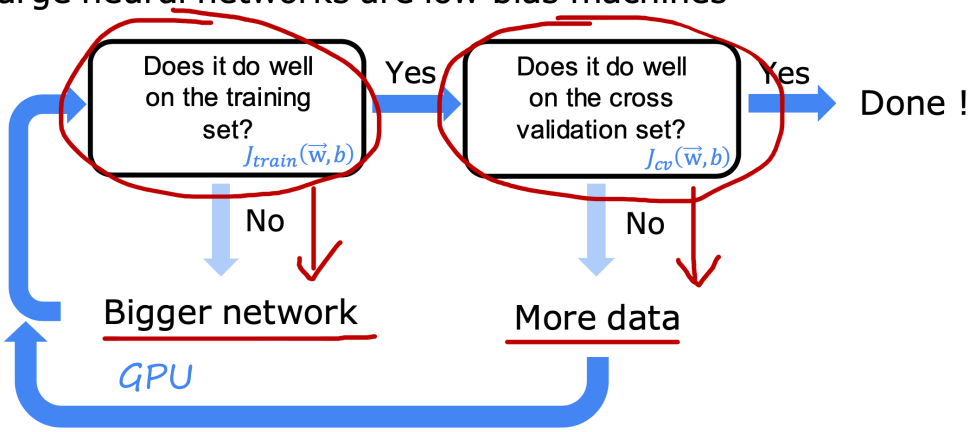
- Regularization in neural networks usually involves adding a term to the cost function that penalizes large weights.

5. Implementing Regularization in TensorFlow:
   - In TensorFlow, regularization can be added by specifying the "kernel_regularizer' parameter in the layers of the network.
   - Example: 'keras.layers.Dense(units, activation, kerne1_regularizer=keras.regularizers.12(0.01))"
6. Advantages of Neural Networks:
   - Neural networks offer a solution to the bias-variance dilemma by allowing for large, complex models that can still generalize well with proper regularization.
   - The main limitation is computational expense, as larger networks require more resources to train.
7. Guiding Neural Network Development:
   - Measuring bias and variance remains a crucial part of developing and tuning neural network models.
   - This approach is a powerful tool for quic y improving machine learning systems, particularly in the realm of deep learning.

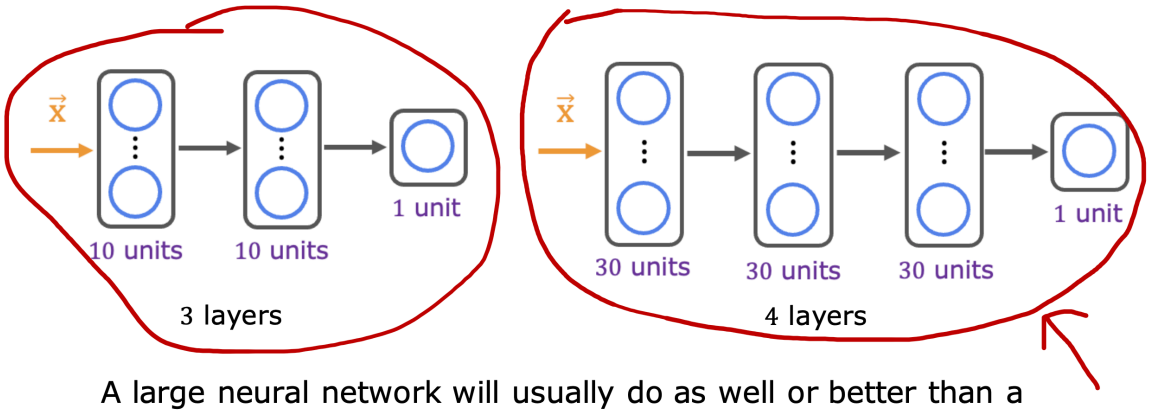


Neural networks and bias variance

Large neural networks are low bias machines



Neural networks and regularization



A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

Neural network regularization

$$J(\mathbf{W},\mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L\left(f\left(\vec{\mathbf{x}}^{(i)}\right), y^{(i)}\right) + \frac{\lambda}{2m}\sum_{all\ weights\ \mathbf{W}}\left(w^2\right)$$

Unregularized MNIST model

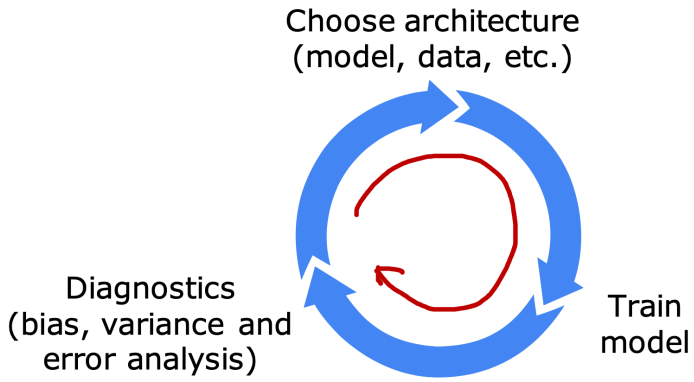
```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

Regularized MNIST model

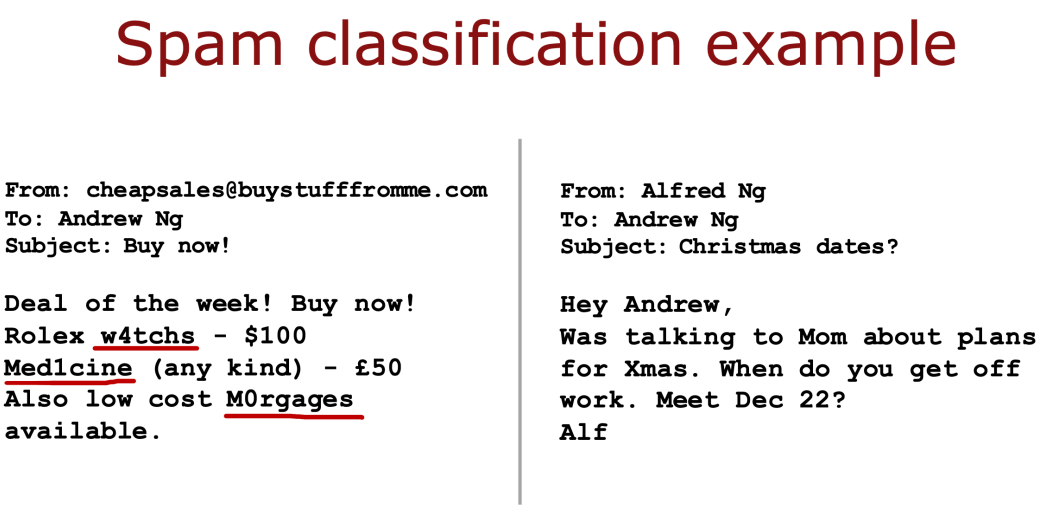
```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

10、 机器学习系统的设计 Machine Learning Development Process

10.1 Developing a Machine Learning System and the Iterative Loop



本周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 100×1。

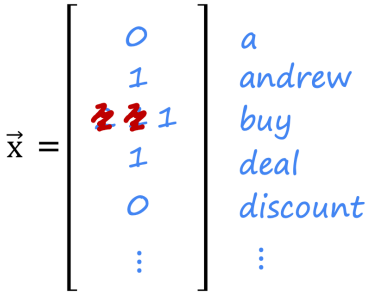
为了构建这个分类器算法，我们可以做很多事，例如：

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. 基于邮件的路由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（把 **watch** 写成 **w4tch**）开发复杂的算法

Building a spam classifier

Supervised learning: $\vec{x}$ = features of email
y = spam (1) or not spam (0)

Features: list the top 10,000 words to compute $x_1, x_2, \cdots, x_{10,000}$




```
From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.
```

Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., "Honeypot" project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body. E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings. E.g., w4tches, med1cine, m0rtgage.

我们将在随后的课程中讲误差分析，我会告诉你怎样用一个更加系统性的方法，从一堆不同的方法中，选取合适的那一个。

10.2 误差分析 Error Analysis

Summary of Error Analysis in Machine Learning Development:

1. Error Analysis Overview:
   - Second most important diagnostic tool after bias and variance analysis.
   - Involves **manually examining misclassified examples** to **understand where the algorithm is failing.**
2. Process of Error Analysis:
   - Examine a subset of examples **that the algorithm misclassified.**

- **Categorize errors** into common themes or traits.
- **Count** the number of examples **in each category** to identify prevalent error types.

3. Example: Email Spam Classifier:
   - Analyze a set of 100 misclassified emails from a cross-validation set.
   - Categorize errors: e.g., pharmaceutical spam, deliberate misspellings, unusual email routing, phishing emails.
   - Use the analysis to prioritize the most common error types for improvement.
4. Decision Making Based on Error Analysis:
   - Helps in deciding where to focus efforts: more data, new features, algorithm modifications.
   - Example: If pharmaceutical spam is a common error, **focus on collecting more data or developing specific features** for it.
5. Sampling for Error Analysis:
   - If the misclassified set is large, randomly sample a manageable subset (e.g., around 100 examples).
   - This provides sufficient statistics for identifying common error types.
6. Applications and Limitations:
   - Particularly **useful for problems where humans perform well,** as human judgement can identify errors.
   - **More challenging for tasks where human judgement is not effective.**
7. Impact of Error Analysis:
   - Provides clarity on which aspects of the model or data need improvement.
   - Helps avoid spending time on less impactful modifications.
8. Next Steps:
   - Exploring efficient methods **for adding data to improve model performance**.
   - Focus on how to effectively gather more data in response to high variance issues identified through error analysis.

10.3 Adding data

Adding Data Strategically:

- Focus on adding data types where the algorithm is underperforming, as identified by error analysis.
- Example: If pharmaceutical spam is a common error, concentrate on adding more examples of such emails.

Data Augmentation:

- Modify existing training examples to create new ones.

- Common in image and audio data applications.
- Techniques include rotating images, adding noise to audio, changing contrast, etc.
- Ensures that modifications are representative of variations in the test set.

Advanced Data Augmentation:

- More complex transformations like warping images or combining audio with different background noises.
- Helps the algorithm learn to recognize patterns under various conditions.

Data Synthesis:

- Creating new training examples from scratch, not by modifying existing ones.
- Useful in computer vision tasks, like generating text images in different fonts and colors for OCR.
- Can be labor-intensive but effective in generating large amounts of realistic data.

Model-Centric vs. Data-Centric Approach:

- Traditional focus has been on improving the model or algorithm.
- A data-centric approach involves engineering the data to enhance performance.
- This can include collecting specific types of data, data augmentation, or synthesis.

Efficiency in Data Engineering:

- Engineering data can often be a more efficient way to improve performance compared to algorithmic tweaks.
- Tailoring data to the specific needs and weaknesses of the algorithm.

Upcoming Topic: Transfer Learning:

- Utilizing data from different but related tasks to improve performance in a target task.
- Particularly useful when data is scarce in the target domain.

Summary:

- Efficient and effective data addition involves a strategic focus on where the algorithm needs improvement.
- Both data augmentation and synthesis are powerful tools, especially in applications like image and audio processing.
- A shift towards a data-centric approach can sometimes yield better results than focusing solely on the model.

Adding data

→ Add more data of everything. E.g., "Honeypot" project.

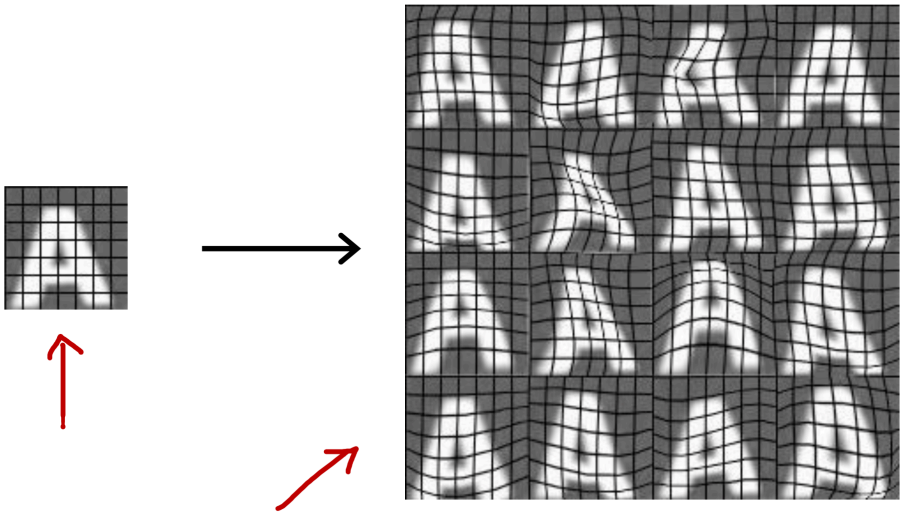
→ Add more data of the types where error analysis has indicated it might help.

Pharma spam

E.g., Go to unlabeled data and find more examples of Pharma related spam.

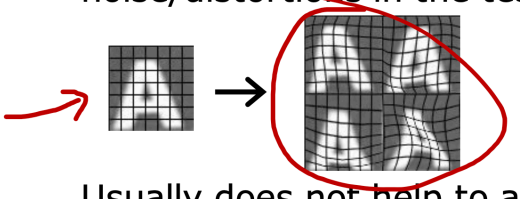
(X,Y)

Data augmentation by introducing distortions



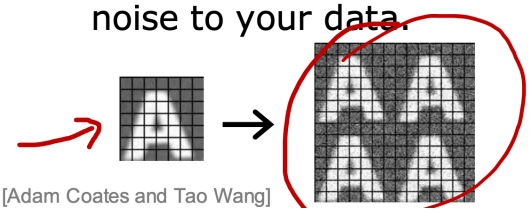
Data augmentation by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



Audio:
Background noise,
bad cellphone connection

Usually does not help to add purely random/meaningless noise to your data.

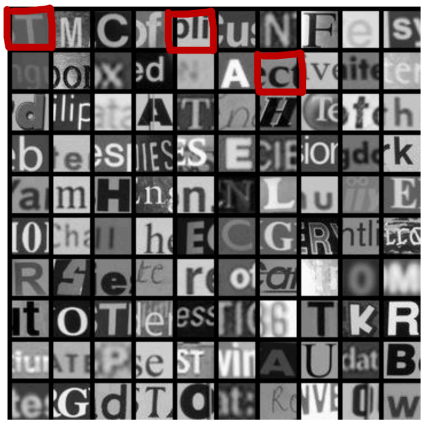


[Adam Coates and Tao Wang]

x_i=intensity (brightness) of pixel i
$x_i \leftarrow x_i$ + random noise

Artificial data synthesis for photo OCR

Artificial data synthesis for photo OCR



Real data

Abcdefg

Abcdefg

Abcdefg

Abcdefg

Abcdefg

Artificial data synthesis for photo OCR

Real data

Synthetic data

10.4 Transfer learning

using data from a different task.

Concept of Transfer Learning:

- Transfer learning is a technique where you use data and learned features from one task to improve performance on a different but related task.
- Especially useful when you have limited data for your target task.

Process of Transfer Learning:

- Start by training a large neural network on a big dataset (e.g., 1 million images of various classes).
- Transfer the learned features (weights and biases) of the initial layers to a new neural network, but replace the output layer to suit your specific task (e.g., recognizing digits 0-9).

Two options for further training:

- Train only the new output layer's parameters (useful for very small datasets).
- Train all parameters in the network, using the transferred weights as initial values (better for slightly larger datasets).

Benefits of Transfer Learning:

- Allows for training on smaller datasets by utilizing pre-learned features.
- Effective in leveraging large pre-trained models available online, saving time and computational resources.
- Particularly beneficial when pre-training and fine-tuning tasks have similar types of input data (like images or audio).

Intuition Behind Effectiveness:

- Early layers of neural networks often learn to detect general features (like edges, corners, or basic shapes in images) that are useful across a wide range of tasks.
- These generalized features can be effectively transferred to a new task.

Restrictions in Application:

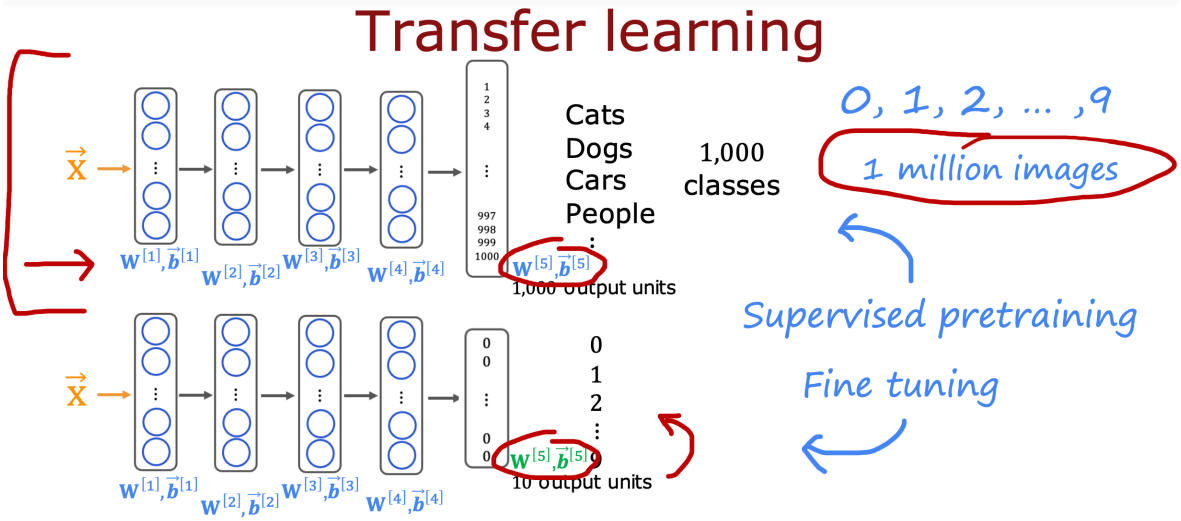
- The pre-training task should have the same type of input data as the fine-tuning task. For instance, a network pre-trained on images wouldn't be effective for audio data.

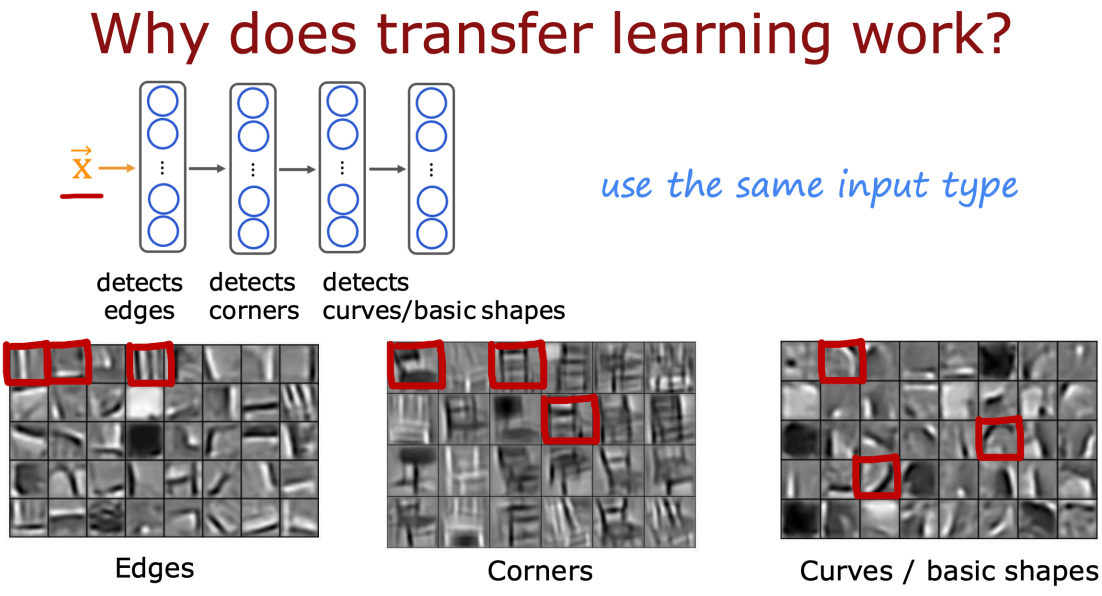
Examples of Pre-trained Models:

- GPT-3, BERT, and models trained on ImageNet are examples of pre-trained models that can be fine-tuned for specific tasks.
- The machine learning community often shares pre-trained models, enabling others to build upon this foundational work.

Summary:

- Transfer learning is a powerful technique for improving performance on a target task by using knowledge gained from a different task.
- It is particularly useful when dealing with limited data in the target task and can significantly speed up the development process.



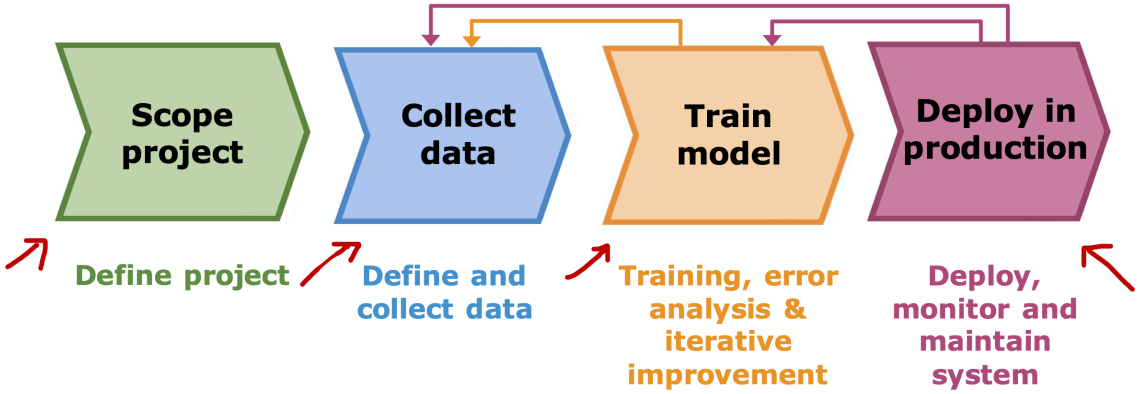


10.5 Full cycle of a machine learning project

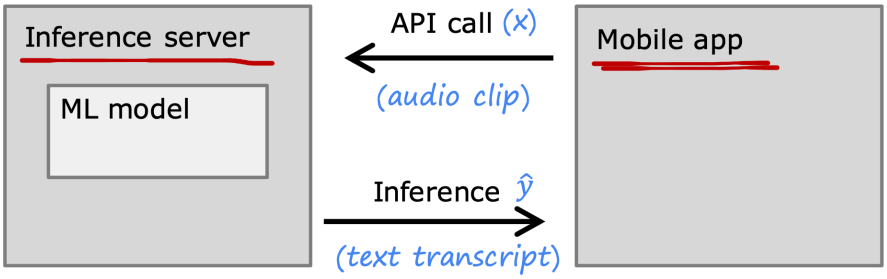
Project Scoping:
Define the project scope and objectives. For instance, building a voice search application for speech recognition.
Data Collection:
Gather necessary data for training the machine learning model. This includes identifying and obtaining relevant datasets.
Model Training:
Train your machine learning model using the collected data. This involves choosing an appropriate algorithm, training it, and performing error analysis for iterative improvements.
Collect More Data (if needed):
Based on insights from error analysis and bias-variance considerations, collect additional data to improve the model. This may involve targeting specific types of data that the model is underperforming on.
Deploying in Production:
Implement the model in a production environment where it can be used by end-users. This step involves setting up an inference server to make predictions based on user inputs.
Monitor and Maintain:
Continuously monitor the model's performance in the production environment and maintain it. This includes logging data, detecting performance degradation, and updating the model as needed.
Software Engineering Requirements:
Depending on the scale of the application, significant software engineering may be required for efficient and reliable predictions, scaling to a large number of users, and handling data privacy and consent.
MLOps (Machine Learning Operations):
This involves practices for systematically building, deploying, and maintaining machine learning systems. It covers aspects like model optimization, scaling, monitoring, and updates.

In summary, the full cycle of a machine learning project encompasses defining the project scope, data collection, model training, possible additional data collection based on error analysis, deployment, continuous monitoring and maintenance, and addressing software engineering and ethical considerations.

Full cycle of a machine learning project



Deployment



→ Software engineering may be needed for:
- Ensure reliable and efficient predictions
- Scaling
- Logging
- System monitoring
- Model updates

MLOps
machine learning operations

10.6 类偏斜的误差度量 Error Metrics for Skewed Classes

在前面的课程中，我提到了误差分析，以及设定误差度量值的重要性。那就是，设定某个实数来评估你的学习算法，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，这有时会对于你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实例，只有很少或没有其他类的实例。

When working with machine learning models, especially in cases where there is a **significant imbalance between positive and negative examples,** traditional accuracy metrics may not be sufficient or informative. A classic scenario where this occurs is in disease detection, where the disease might be rare in the population.

Imbalanced Classes:

- In cases like rare disease detection, the number of positive examples (people with the disease) might be much smaller compared to negative examples (people without the disease).

Problem with Accuracy Metric:

- If a disease is present in only a small fraction of the population, a model that predicts 'no disease' for all patients might still achieve high accuracy. For example, if 0.5% have the disease, predicting 'no disease' every time results in 99.5% accuracy.

10.7 Precision and Recall Metrics

- These metrics are more informative in imbalanced scenarios:
- **Precision:**
    - Of all the cases we predicted as positive, how many were actually positive?
    - It's calculated as **True Positives / (True Positives + False Positives).**
- **Recall:**
    - Of all the actual positive cases, how many did we correctly identify?
    - It's calculated **as True Positives / (True Positives + False Negatives).**

Example Calculation:

- Suppose a model's predictions on a test set result in:
    - 15 True Positives (disease correctly identified),

  - 5 False Positives (no disease, but identified as disease),
  - 10 False Negatives (disease missed),
  - 70 True Negatives (correctly identified as no disease).
- The precision would be 15 / (15 + 5) = 0.75 or 75%.
- The recall would be 15 / (15 + 10) = 0.6 or 60%.

Interpreting Precision and Recall:

- **High Precision:** Indicates a high percentage of positive predictions were correct.
- **High Recall:** Indicates the model is good at capturing a high percentage of actual positives.

Importance in Disease Detection:

- In medical diagnostics, especially for rare diseases, precision and recall are crucial. High precision ensures that most of the positive predictions are accurate, and high recall ensures that the model doesn't miss too many actual cases of the disease.

Trade-offs:

- There is often a **trade-off between precision and recall**. Improving one can sometimes reduce the other. The challenge is to find a balance that suits the specific requirements of the application.

In summary, precision and recall are essential metrics for evaluating models in situations with imbalanced classes, such as rare disease detection, where traditional accuracy might not be a reliable indicator of model performance. They help in understanding the model's ability to correctly predict positives and its effectiveness in not missing actual positive cases.

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。这时，误差的大小是不能视为评判算法效果的依据的。

查准率（**Precision**）和**查全率**（**Recall**） 我们将算法预测的结果分成四种情况：

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：查准率=**TP/(TP+FP)**。例，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)**。例，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。

这样，对于我们刚才那个总是预测病人肿瘤为良性的算法，其查全率是 0, which is a useless model.

Rare disease classification example

Train classifier $f_{\vec{w},b}(\vec{x})$ ($y = 1$ if disease present, $y = 0$ otherwise)

Find that you've got 1% error on test set
(99% correct diagnoses)

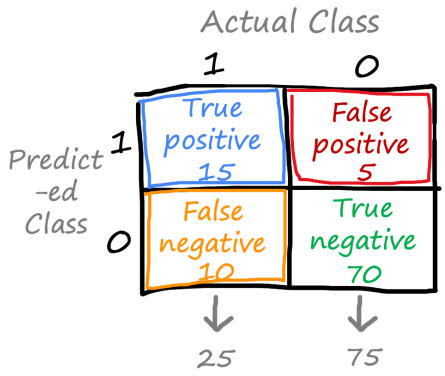
Only 0.5% of patients have the disease

```
print("y=0")
```

99.5% accuracy, 0.5% error ←
1%
1.2%

Precision/recall

$y = 1$ in presence of rare class we want to detect.



Precision:
(of all patients where we predicted $y = 1$, what fraction actually have the rare disease?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15+5} = 0.75$$

Recall: ←
(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

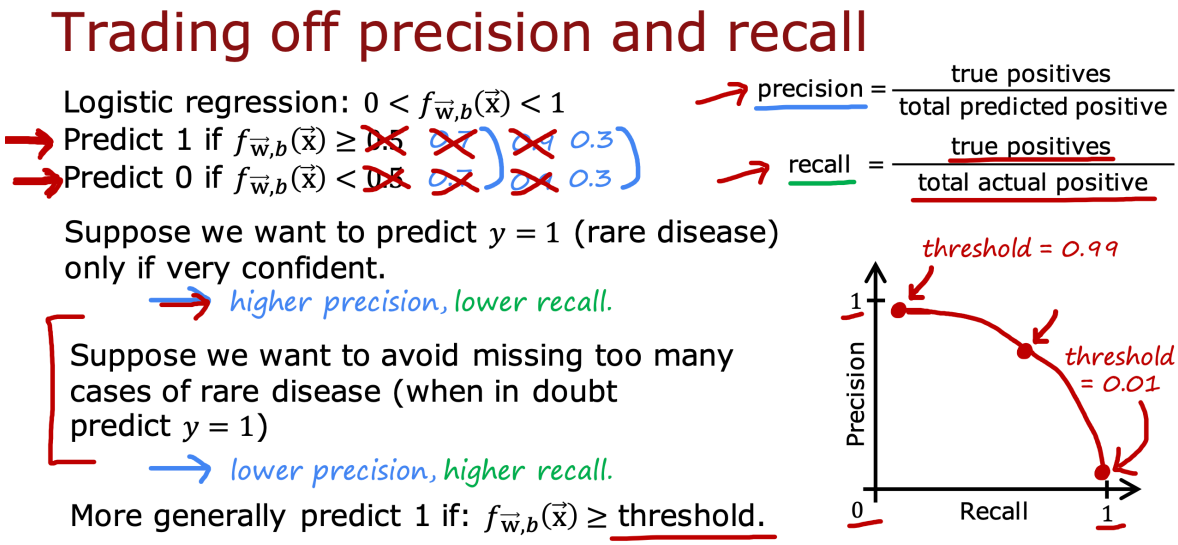
$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$

```
print("y=0")
```

10.8 查准率和查全率之间的权衡 Trade-Offs between Precision and Recall

The concept of trade-offs between precision and recall is fundamental in machine learning, particularly in scenarios involving imbalanced datasets, such as disease detection. This trade-off is crucial for optimizing the performance of classification models. Let's delve into the details:

假使，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。



查准率**(Precision)=TP/(TP+FP)** 例，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，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：

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：

$$F_1Score: 2\frac{PR}{P+R}$$

我们选择使得 **F1** 值最高的阀值。

F1 score

How to compare precision/recall numbers?

	Precision (P)	Recall (R)	Average	F_1 score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.501	0.0392

```
print("y=1")
```

Average = $\frac{P+R}{2}$

F1 score = $\frac{1}{\frac{1}{2}\left(\frac{1}{P}+\frac{1}{R}\right)} = 2\frac{PR}{P+R}$

- **Higher Thresholds:** Increasing the threshold for predicting a positive class (e.g., setting it to 0.7 or 0.9 instead of 0.5 in logistic regression) will result in higher precision but lower recall. This means the model is more conservative about predicting positives, so when it does, it's more likely to be correct. However, it might miss more actual positive cases.
- **Lower Thresholds:** Conversely, lowering the threshold (e.g., setting it to 0.3) results in higher recall but lower precision. The model predicts positives more liberally, catching more actual positive cases but increasing the risk of false positives.

Balancing Precision and Recall:

- The challenge is to find the right balance between precision and recall that fits the specific needs of your application. For example, in medical diagnostics for a deadly disease, you might prioritize high recall to ensure fewer missed cases of the disease, even if it means tolerating more false positives.

F1 Score - A Combined Metric:

- To balance precision and recall, the F1 Score is often used. It is the harmonic mean of precision and recall, calculated as 2 * (Precision * Recall) / (Precision + Recall). The F1 Score gives more weight to lower values, ensuring that both precision and recall are reasonably high.
- Example: If a model has a precision of 90% and a recall of 30%, the F1 Score would be much lower than the average of precision and recall, reflecting the imbalance.

Practical Implications:

- Choosing a Threshold: The choice of the threshold depends on the specific requirements of the application. For instance, in fraud detection, you might prefer a lower threshold to flag potential fraud cases for further investigation.
- Automating Threshold Selection: The F1 Score can be used to automate the selection of the threshold that best balances precision and recall.

Harmonic Mean vs. Arithmetic Mean:

- The arithmetic mean of precision and recall would not accurately reflect the effectiveness of a model, especially when one of them is very low. The F1 Score, being a harmonic mean, addresses this by emphasizing the lower of the two values.

In summary, the precision-recall trade-off is a crucial aspect of optimizing classification models, especially in cases with imbalanced classes. The F1 Score provides a useful metric for balancing these two measures, ensuring that a model is neither too conservative nor too liberal in predicting the positive class. This balance is particularly important in critical applications like medical diagnostics, fraud detection, and others where the consequences of false negatives or false positives are significant.

11、 Decision tree model

Decision trees are a powerful and widely-used machine learning algorithm, particularly effective for classification problems. They are intuitive and easy to implement, making them a popular choice in various applications, including competitions. Let's explore the fundamentals of decision trees using your cat classification example:

11.1 Basic Concept of Decision Trees

- A decision tree is a flowchart-like structure where each internal node represents a "test" on an attribute (e.g., ear shape), each branch represents the outcome of the test, and each leaf node represents a class label (decision taken after computing all attributes).
- The paths from root to leaf represent classification rules.

Working of a Decision Tree:

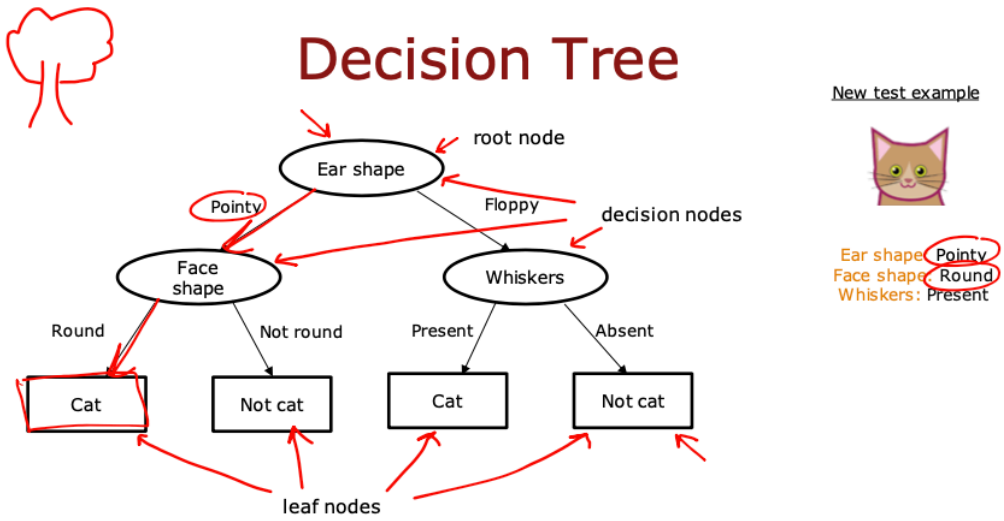
- In your cat classification example, the decision tree takes an animal's characteristics (ear shape, face shape, whiskers) and classifies the animal as a cat or not.
- The process **starts at the root node** (top of the tree), checks the **attribute**, and follows the branch corresponding to **the attribute's value** to a **decision node (feature binary classification),** repeating this process until it reaches a **leaf node**, which gives the **prediction/classification.**

Decision Tree discrete Features

- Discrete
- Categorical
- Binary classifications for each feature
  There are also continuous features.

Terminology:

- Root Node: The topmost node that represents the entire population or sample.
- Decision Nodes: Nodes that split for the value of a certain attribute.
- Leaf Nodes: Terminal nodes that predict the outcome (label).



Construction of a Decision Tree:

- The **decision of how to split at each node** is made based on a **metric** like **Gini impurity**, **information gain**, or **reduction in variance.**
- The tree is constructed in a top-down manner, choosing the splits that best separate the data for the categories at each step.

11.2 Learning Process

Decision 1: How to choose what feature to split on at each node?

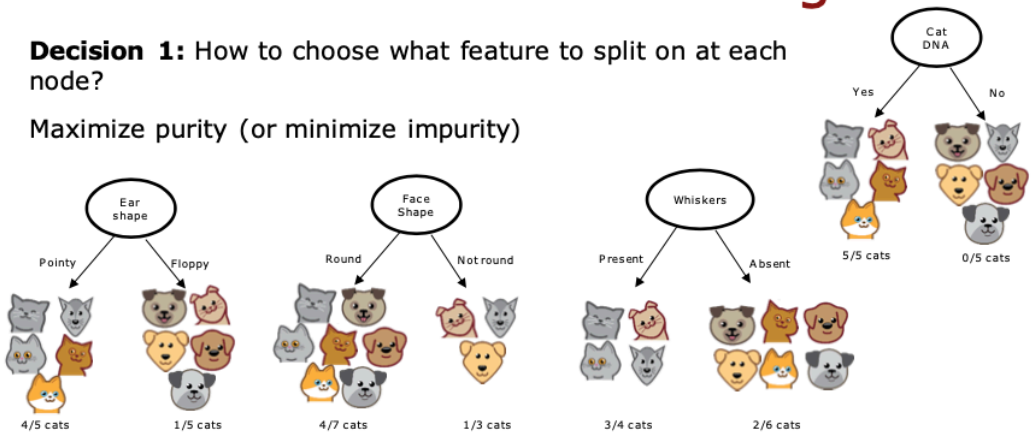
Choosing the Feature to Split On:

- At each node, you need to decide which feature to use for splitting the data.
- The goal is to **maximize** the "**purity**" of the nodes after the split. Purity refers to **how mixed the classes are in each node.** Higher purity means the node predominantly contains examples from a single class.

Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?

Maximize purity (or minimize impurity)



-

Splitting the Dataset:

- Based on the chosen feature, the dataset is split into subsets. Each subset corresponds to one of the possible values of the feature.
- For instance, if the chosen feature is 'ear shape' with values 'pointy' and 'floppy', the dataset will be divided into two groups accordingly.

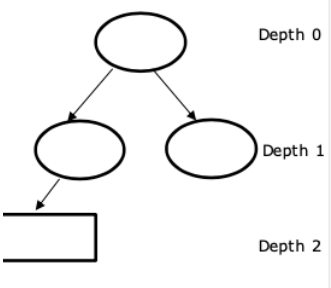
Decision 2: When do you stop splitting?

Deciding When to Stop Splitting:

- You need criteria to decide when to **stop** further splitting and **create a leaf node.**

Common criteria include:

- **When a node is 100% one class**
- **Maximum Tree Depth (self-defined):** Limiting the depth of the tree to prevent it from growing too large. By keeping the depth **small**, we **avoid overfitting**.

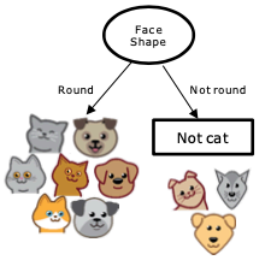


- **Minimum Improvement:** Ceasing splits if the **increase in purity** falls below a certain **threshold**.
- **Minimum Number of Samples:** Stopping if the number of samples at a node falls **below** a specified count (no longer worth splitting since the majority in a splitting result is already pure enough).
- These criteria help to **prevent overfitting** and maintain a manageable tree size.

Decision Tree Learning

Decision 2: When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



Creating Leaf Nodes:

- Once a decision is made to stop splitting, a leaf node is created.
- The leaf node represents a class label. It is determined based on the majority class in that node or through other statistical measures.

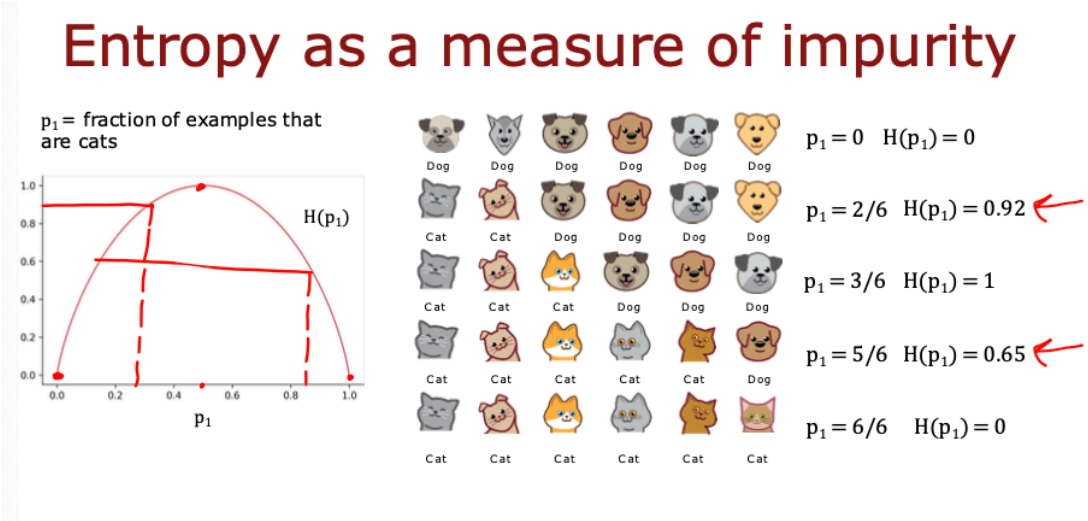
Iterating the Process:

- This process is iterated for each branch of the tree until all branches end in leaf nodes.

Measuring purity

11.3 Entropy as a Measure of Impurity

- p_{depth} = fraction of examples that **are target class**
- In the upcoming video, you'll learn about entropy, a concept used to measure the impurity of a node.
- **When the fraction of examples that are non-target class is 50%, this is a sign of highly impure ($H(p_i)$) based on the entroy (impurity) distribution.**

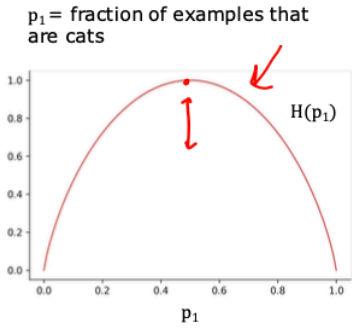


- Entropy helps in determining the effectiveness of a split. A good split is one that reduces entropy (or impurity) in the resulting nodes.

11.3.1 Entropy as Impurity Function $H(p_1)$

Note: Log base 2 just to make the peak of the curve equal to 1
If p1 or p0 is 0, we need to self-define 0 log(0) = 0 since in general log(0) is invalid.
Alternative: Gini impurity

Entropy as a measure of impurity



$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 \ log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

Note: "0 log(0)" = 0

Practical Implementation:

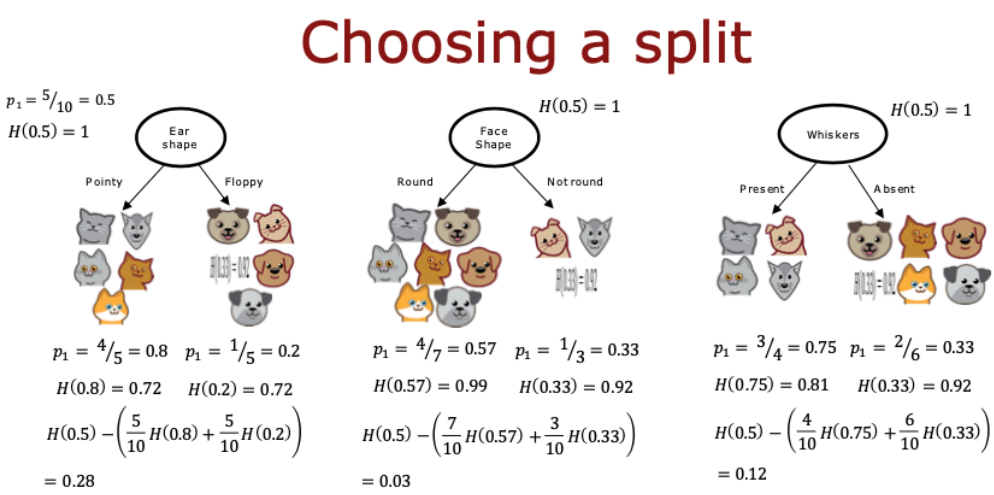
- While the concept might seem intricate, many machine learning libraries provide efficient implementations of decision trees, allowing you to leverage this powerful tool without having to handle the complexity of its underlying algorithm.

11.4 Choosing a split: information gain (reduction in Entropy)

The worst-case entropy - A **weighted average** of entropy to **account** for the **sample size fraction**.
Note:

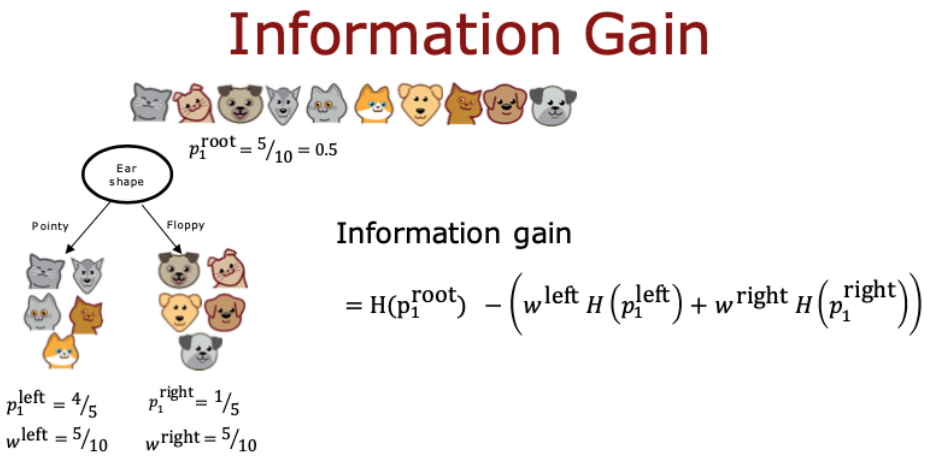
- The worst-case entropy is the entropy at the root node.
- Sample size fraction (w): samples in a feature value (w^{left} *or* w^{right}) branch/ totoal sample in that feature branch ($w^{left} + w^{right}$).

In the following example, splitting based on ear shape gives the highest information gain 0.28.



11.4.1 General formula of information gain

$$H\left(p_{depth}^{root}\right) - \left(w^{left}H\left(p_{depth}^{left}\right) + w^{right}H\left(p_{depth}^{right}\right)\right)$$



Decision Trees in Practice:

- Versatility: Can handle both **categorical** and numerical data.

Challenges:

- Overfitting: Decision trees can easily **overfit**, especially with a lot of features. Techniques like **pruning** (removal of sections of the tree that provide little power to classify instances) are used to avoid this.
- Complexity with Large Data: With a large number of features, the tree can become complex and less interpretable.

11.5 Decision Tree Process

- The training process involves **selecting the best attribute to split** the data at each step. This is done by **calculating the purity** of the node and **the information gain** achieved by each split.

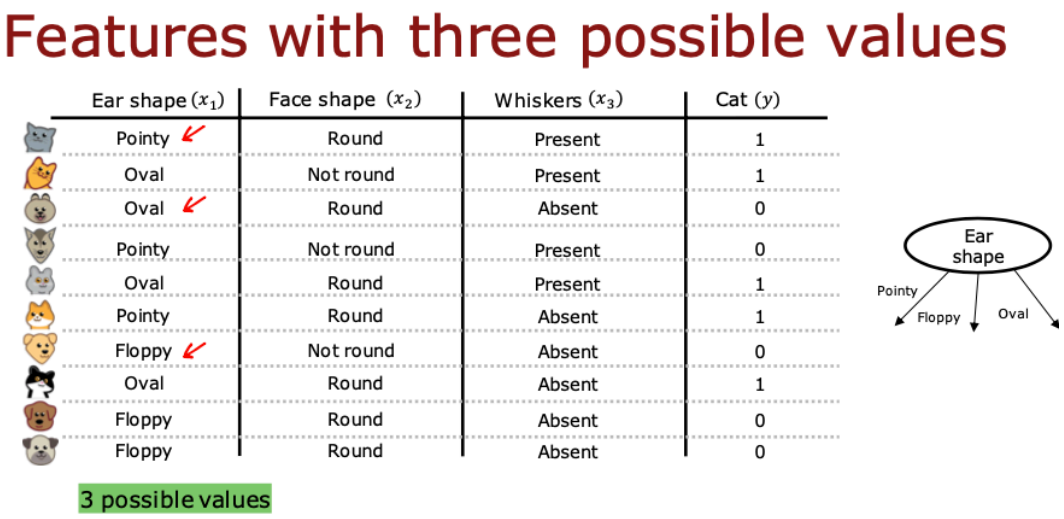
In the next step, you will learn how a decision tree is **trained using a dataset,** focusing on **selecting the splits based on certain criteria to create a model** that best fits the data. Decision trees are a **foundational** element for more complex models like **random forests** and **gradient boosting machines**, **which leverage multiple trees** for more robust predictions.

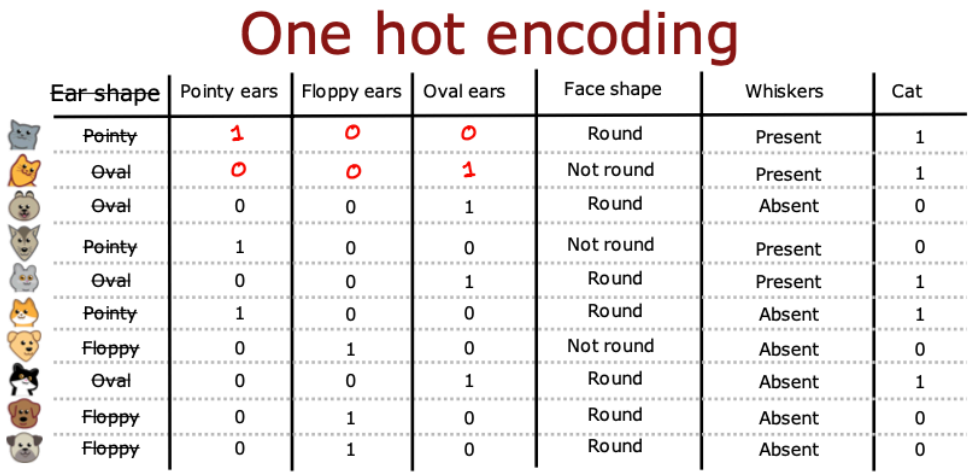
Recuesive Algorithm (function that clals itself)

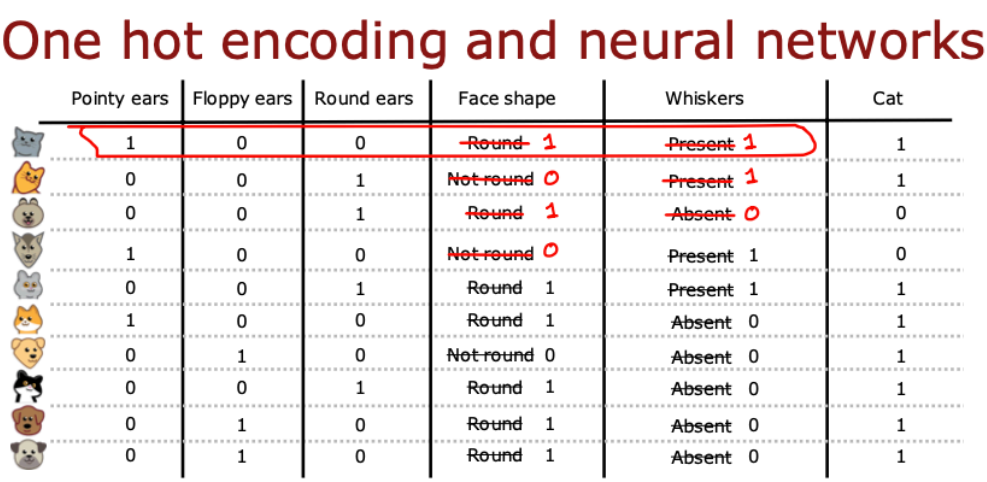
1. Start with all examples at the root node
2. Calculate information gain for all possible features, and pick the one with the highest information gain.
3. Split dataset according to selected feature, and create left and right branches of the tree
4. Keep repeating splitting process until stopping criteria is met:
    a. When a node is 100% one class
    b. When splitting a node will result in the tree exceeding a maximum
    c. depth
    d. Information gain from additional splits is less than threshold.
    e. When number of examples in a node is below a threshold

11.6 One-hot encoding for categorical features that are no longer binary classification.

If a categorical feature can take on k values, create k binary features (0 or 1 valued).

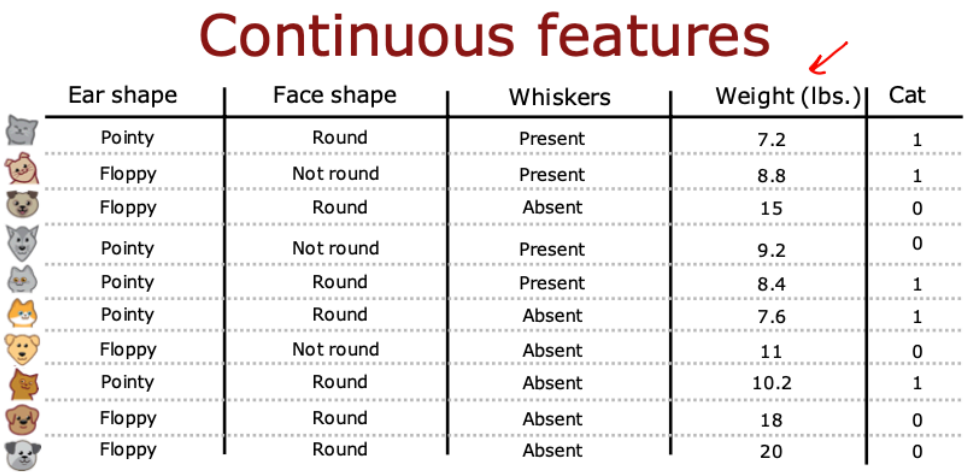


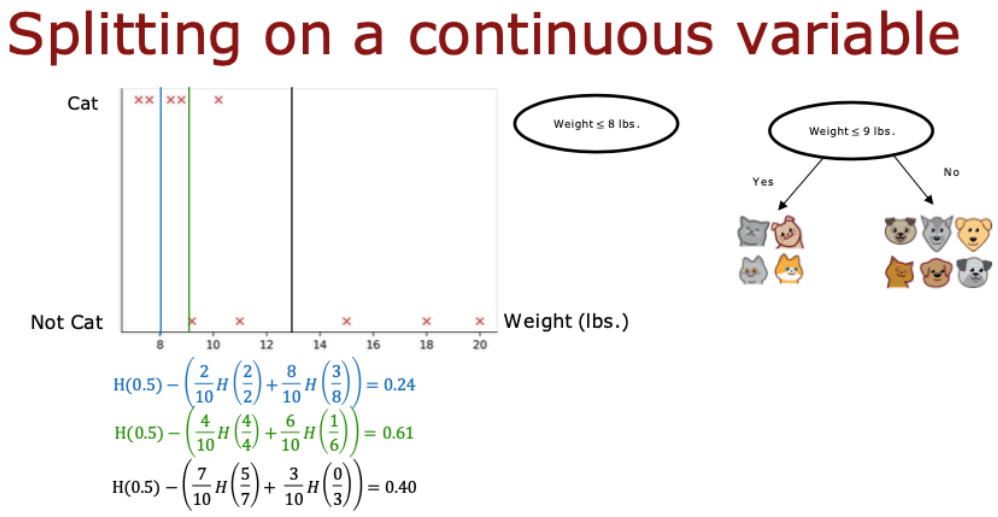




11.7 Continuous valued features

convention would be to **sort** all of the examples according to the weight or according to the value of this feature and take all the values that **are mid points** between the sorted list of training. Examples as the values for consideration for this threshold over here. This way, if you have 10 training examples, you will test nine different possible values for this threshold and then try to **pick the one that gives you the highest information gain.** And finally, if the information gained from splitting on a given value of this **threshold** is better than the information gain from splitting on any other feature, then you will decide to split that node at that feature.





11.8 Regression Trees

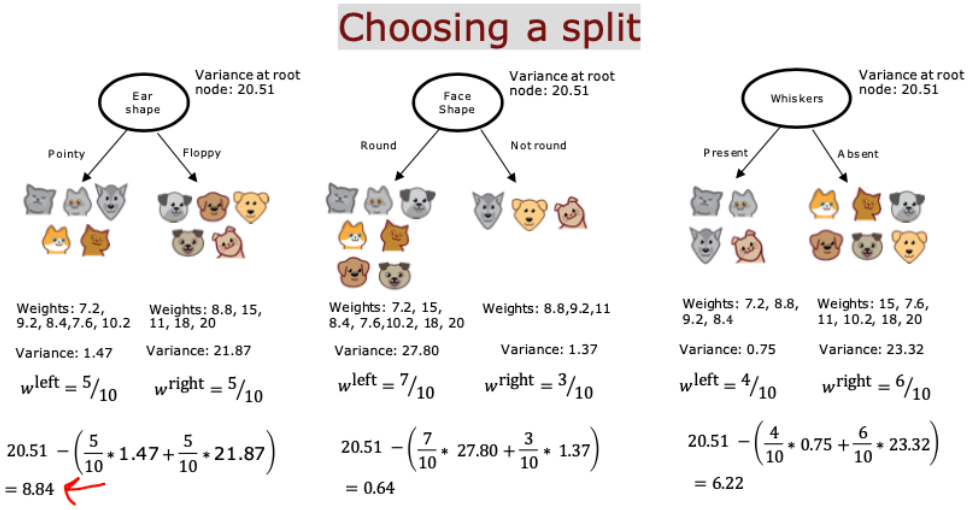
Regression with Decision Trees: Predicting a number.
The decision tree is going to make a prediction based on taking the **average** of the weights in the training examples.

Ear shape	Face shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not round	Present	8.8
Floppy	Round	Absent	15
Pointy	Not round	Present	9.2
Pointy	Round	Present	8.4
Pointy	Round	Absent	7.6
Floppy	Not round	Absent	11
Pointy	Round	Absent	10.2
Floppy	Round	Absent	18
Floppy	Round	Absent	20

X y

11.8.1 Choosing a split: reduction in Variance

Variane at node – weighted average variance (left + right)
→ **choose the maximum delta variance.**

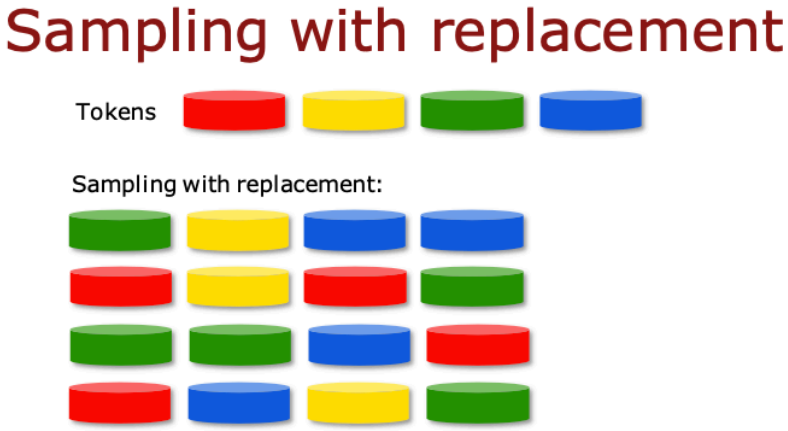


12、 Tree ensembles (multiple decision tree)

Using **multiple** decision trees since **single** decision tree model is **sensitive** to **small changes** in the data that the **single decision tree model sequence of feature eclassification alteres.**
Hence, **multiple** trees should be employed to dataset in **parallel** and **prioritize** (voting) to **predict** the **final** result.

12.1 Sampling with replacement: key technique in tree ensemble

We have N samples in a sample set. Randomly and blindly select from a sample set, put the selected sample back into the set, bindly select again for N times. This blindly drew samples, despite repeated samples, are the new set of samples for tree training.

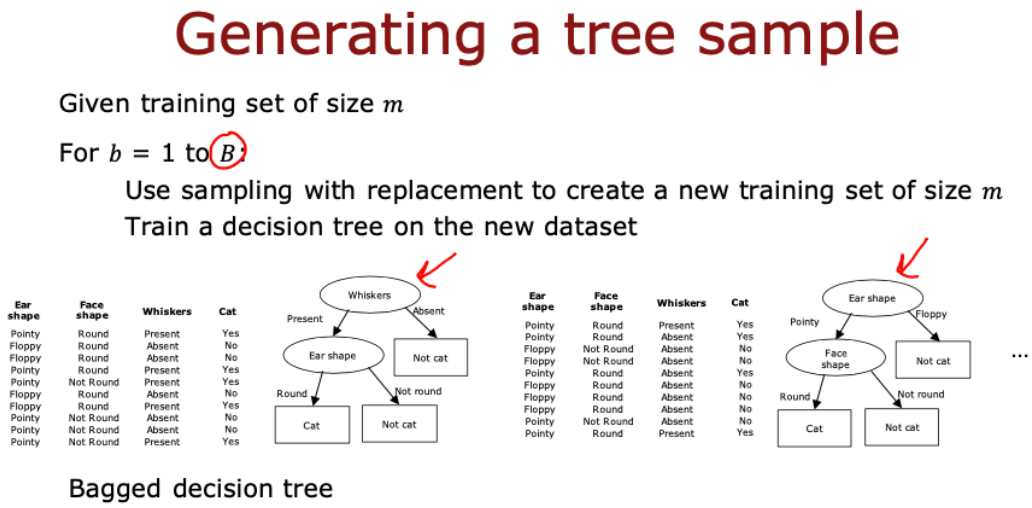


12.2 Random forest algorithm: sample with replacement implementation in tree ensemble

B is suggested 64 ≤ B ≤ 128. The larger the B, as long as within 128, the better the model perform.

12.2.1 Randomizing the feature choice when n ≥ 1000

At each node, when choosing a feature to use to split, if n **features are available, pick a random subset of**
$k < n$ **features** and allow the a**lgorithm** to only **choose** from that **subset of features with highest information gain.** In this case, $k = \sqrt{n}$



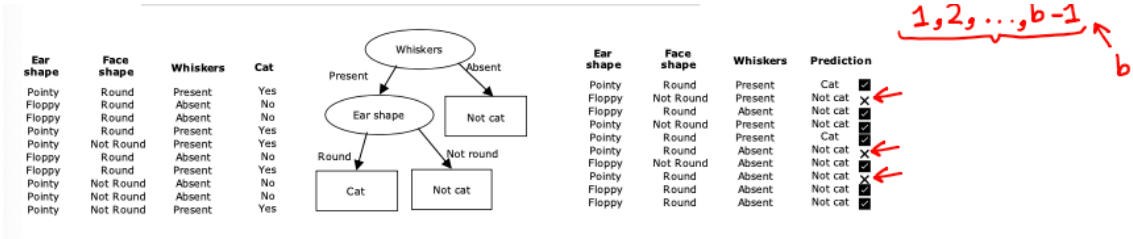
12.3 XGBoost (eXtreme Gradient Boosting): an improved version of random forest

12.3.1 Boosted trees intuition: correct trained tree faults

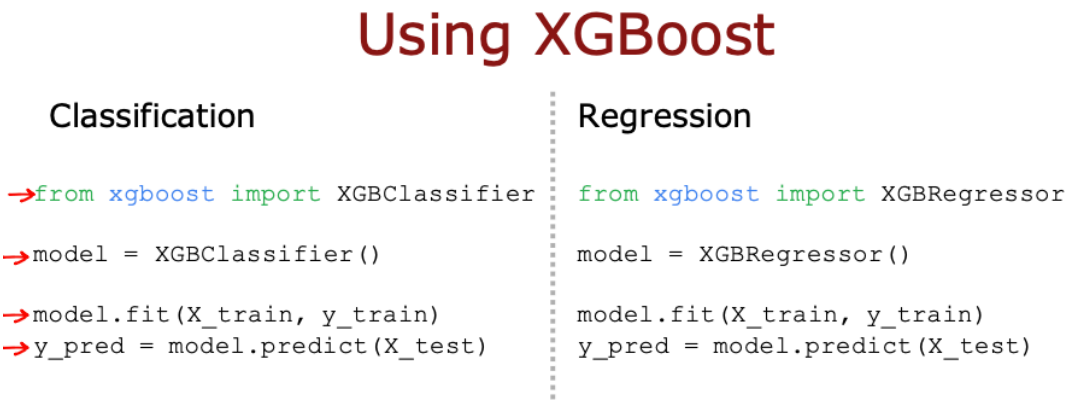
Given training set of size m

For b = 1 to B:
Use **sampling with replacement** to **create a new training set of size m.** But **instead** of picking from **all** examples with **equal (1/m) probability**, make it more likely to pick examples that the **previously** trained trees **misclassify**.
Train a decision tree on the new dataset

- Open source implementation of boosted trees
- Fast efficient implementation (**assign weights to samples as an improvement for sampling replacement**)
- Good choice of default splitting criteria and **criteria for when to stop splitting**
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)



12.3.2 Using XGBoost



12.4 When to use decision trees.

12.4.1 Decision Trees and Tree ensembles

• Works well **on tabular/spreadsheet** (**structured**) data

• **Not** recommended for unstructured data (images, audio, text)

• **Fast**

• Small decision trees may be human interpretable

12.4.2 Neural Networks

• Works well on **all** types of data, including tabular (structured) and unstructured data

• May be **slower** than a decision tree

• Works with **transfer learning**

• When building a system of **multiple models working together,** it might be easier to string together multiple neural networks