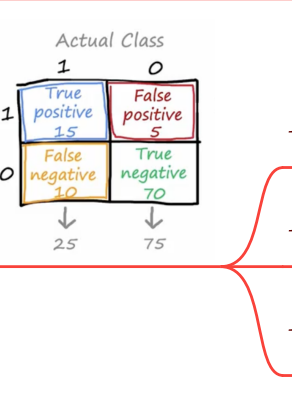
Machine Learning

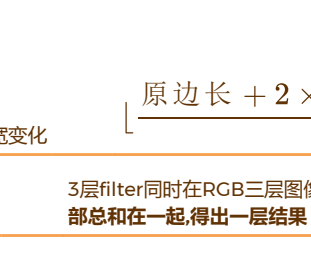
Techinques

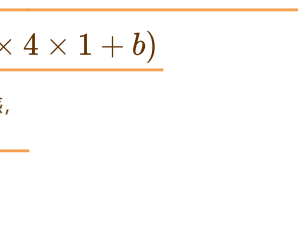
- 什么是Decision Boundary — 样本空间中，区分不同预测结果的分界线
- Cost Function
  - Mean Square Error
  - Cross Entropy
    - Binary Cross Entropy
    - Sparse Categorical Cross Entropy
    - 为什么logistic regression要用 Binary Cross Entropy作为损失函数 — 因为如果用MSE，损失函数图像将会有很多local minimum （non-convex），梯度下降会困在一个local minimum 里面， 而cross entropy的图像比较平滑，且只有一个global minimum（convex)
- Gradient Descent
  - 同时更新所有参数 $\vec{w}_i, b_i$
    - $w_i^{(j)} = w_i^{(j)} - \alpha\frac{\delta J(\vec{w},b)}{\delta w_i^{(j)}}$
    - $b^{(j)} = b^{(j)} - \alpha\frac{\delta J(\vec{w},b)}{\delta b}$
  - Gradient Descent with Momentum
    - $w_i^{(j)} = w_i^{(j)} - v_t$ — $v_t = \rho v_{t-1} + \alpha\frac{\delta J(\vec{w},b)}{\delta w_i^{(j)}}$ — ρ 是momentum的超参数，一般设置为0.9
    - $b^{(j)} = b^{(j)} - v_t$ — $v_t = \rho v_{t-1} + \alpha\frac{\delta J(\vec{w},b)}{\delta b}$
  - 如何判断梯度下降已经收敛?
    - 画出J(w,b) - #iteration图像
    - 如果两次iteration之间， $\Delta J(\vec{w},b) < \epsilon$ — ϵ 是人为设置的一盒很小的值， 例如0.001
  - 学习率 α
    - 设置太大 — 梯度下降不能收敛，在最小值左右来回横跳
    - 设置太小 — 梯度下降收敛太慢
    - 尝试 0.001 , 0.003 , 0.01 , 0.03 , 0.1 , 0.3 , 1 , 3 ......
  - Overfitting
    - Underfitting: High bias vs. Overfitting: High variance
    - addressing overfitting options:
      - Collect more data
      - Select features (to include or exclude)
      - Regularization (reduce the size of params) terms $\frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2$
- feature engineering
  - feature scaling
    - 将不同尺度的特征缩放到相似的尺度，例如正态化所有数据
    - 为什么我们需要特征缩放 — 如果没有特征缩放，不同权重可能相差几个数量级，损失函数的图像可能会非常"扁平"，梯度下降时损失函数会来回横跳
      - 加快梯度下降速度
      - 统一权重，增加模型准确性
    - mean normalization — 将样本平均的散布在坐标轴零点的周围，除以极差 — $\frac{x-\mu}{max-min}$
    - z-score normalization — 同样也是散布在零点周围，但是除以正态分布的方差 — $\frac{x-\mu}{\sigma}$
  - feature encoding — 将离散的特征编码到数字，例如红色，绿色，蓝色编码到0，1，2
  - One-Hot Encoding — 用0和1 encode features — 例如 — 耳朵的形状: {圆形，尖型，椭圆} 改为 耳朵是圆形吗 :{0,1}，耳朵是尖型吗：{0,1}, 耳朵是椭圆吗:{0，1}
  - 画出 correlation 热力图，去除feature
    - 去除与要预测的结果不相关的feature
    - 两个高度相关的feature，去除一个
    - 寻找可以合并的feature，例如房子的宽和高，合并为面积
- Model Selection
  - Cross Validation
    - Validation
      - 将数据集划分为training set (60%) , validation set (20%) , test set (20%)
      - 使用模型在training set上训练（包含正则项），并计算在validation set上的损失（不包含正则项） $J_{cv}(w^n, b^n)$
      - 选择在validation set上损失最小的模型，并计算模型在test set上的损失（不包含正则项），将其作为模型泛化能力的评估
    - K-Fold Cross Validation
      - 将数据随机分为平均的K份subset，每次训练用K-1个subset训练，留出一个subset作validation
      - 如果K就等于数据集的数量（相当于每次只留出一条数据作为validation set），那么这个叫做Leave-One-Out-Cross-Validation
    - Cross-Validation 如何解决Overfitting的问题?
      - 每次留出一部分数据作为验证集，重复多次，可以保证模型不仅仅在记忆数据
      - 每次validation都可以给我们一个相对客观的，对模型泛化能力的评价
  - High Variance vs. High Bias
    - 如果 loss on training set 非常大，我们说这个模型High Bias （underfitting)
    - 如果 loss on training set 很小，但是 loss on validation set 很大，，并且远大于training set loss那么我们说这个模型High Variance （overfitting)
    - 模型可以同时High Bias和High Variance，那说明模型某一部分underfitting，另一部分overfitting
    - 如何判断underfitting还是overfitting ?
      - 确定baseline performance
        - human level performance
        - peers' performance
        - 靠遗忘经验判断
      - 比较baseline和 training，training和validation 的差距
      - 画出学习曲线，画出validation error和training error随着数据集size的增长而增长
      - 神经网络 — 如果神经网络的尺寸够大，就不会有underfit的问题，合适的添加regularisation terms，也能有效抑制overfit problem
  - 为什么不直接使用test set选择模型? — 直接使用test set得来的损失J_test会比实际小（optimistic estimation），因为选择模型时，我们选择在测试数据上表现更好的模型，因此模型可能带有对测试数据的bias
- Error Analysis — 确定模型，输入输出 -----> 训练模型 -----> 寻找问题（bias， variance）
- Precision, Recall, and F1 Score
  - 如果数据是skewed classes 或者 rare class，使用accuracy并不能准确的反映模型的精度, — 例如模型识别的精度是99%，但是人群中只有0.5%的人患有这种疾病
  - 

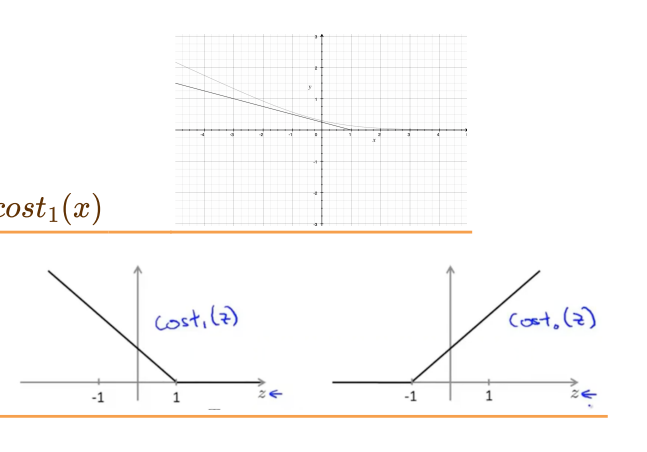
  - $Precision = \frac{True\ Positives}{\#Predicted\ Positives} = \frac{True\ Positives}{True\ Positives + False\ Positives}$ — 高精度意味着，如果模型告诉我这个病人有病，那么大概率这个病人真的有病
  - $Recall = \frac{True\ Positives}{\#Actual\ Positives} = \frac{True\ Positives}{True\ Positives + False\ Negtives}$ — 高Recall意味着，如果这个病人真的有病，那么大概率会被模型识别出来
  - $F1\ Score = \frac{1}{\frac{1}{2}(\frac{1}{P}+\frac{1}{R})} = \frac{2\times P\times R}{P+R}$ — Precision和Recall的harmonic mean，会偏向更小的一边
  - Trade off
- batch processing
  - 传统的Full Batch Learning — 在所有数据上做一遍foreward propagation，计算所有损失平均值，再做backward propagation，更新参数
  - mini-Batch Learning
    - 将所有数据划分成小的批次/子集，每个批次/子集做一遍backward propagation，更新梯度
    - 优缺点：梯地下降更慢，效率更低，但是对内存的要求降低
  - epoch — 每次用全部数据训练一遍后，称为一个epoch

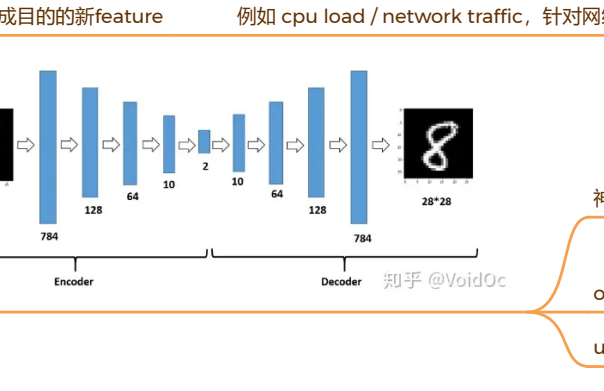
Model

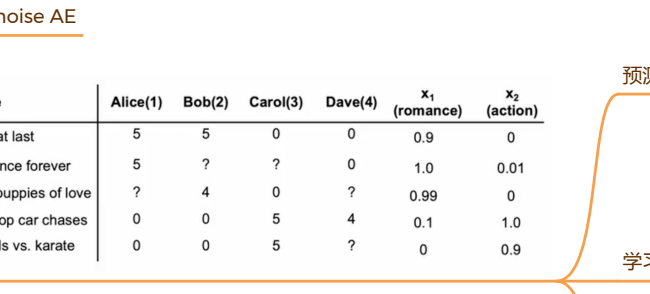
- 有监督学习
  - Linear Regression
    - Polynomial Regression — 如果同时使用x , x^2 , x^3作为input，记得rescaling x^2 x^3
    - Loss Function — Square Error $(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2$
    - Cost Function — Mean Square Error $J(\vec{w},b) = \frac{1}{2m}\sum_{i=1}^{m}(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2$ — 为什么多个1/2? — 为了后面平方求导方便
  - Logistic Regression
    - Sigmoid Function $f_{\vec{w},b}(x) = \frac{1}{1+e^{-\vec{w}x+b}}$
    - Loss function $L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$
      - Binary Cross Entropy
        - 当 y = 1 的时候 $-log(f_{\vec{w},b}(\vec{x}^{(i)}))$
        - 当 y = 0 的时候 $-log(1 - f_{\vec{w},b}(\vec{x}^{(i)}))$
        - $-y\times log(f_{\vec{w},b}(\vec{x}^{(i)})) - (1-y)\times log(1-f_{\vec{w},b}(\vec{x}^{(i)}))$
      - 为什么logistic regression要用 Binary Cross Entropy作为损失函数 — 因为如果用MSE，损失函数图像将会有很多local minimum （non-convex），梯度下降会困在一个local minimum 里面， 而cross entropy的图像比较平滑，且只有一个global minimum（convex)
    - Cost function $J(\vec{w},b) = \frac{1}{m}\sum_{i=1}^{m}L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n}w_j^2$
  - Multiclass classification
    - Activision Function of the output layer — Softmax — $a_n = \frac{e^{z_n}}{\sum_{n=1}^{N}e^{z_n}}$ — $z_n = \vec{w}_n\times a_{n-1} + b_n$
    - Loss Function — Sparse Categorical Cross Entropy — $loss(a_1,\dots,a_n,y) = \begin{cases}-log\ a_1 & \text{if } x=1,\\ \dots & \\ -log\ a_N & \text{if } x=N.\end{cases}$
    - 或者神经网络直接输出几个可能种类的one-hot encoding — Loss Function — $J(\vec{w},b) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^{(i)}log(f_{\vec{w},b}(\vec{x}^{(i)})) + (1-y_k^{(i)})log(1-f_{\vec{w},b}(\vec{x}))] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{S_l}\sum_{i=1}^{S_{l+1}}(w_{ij}^{(l)})^2$
  - Neural Network
    - forward propagation — 矩阵计算
    - backward propagation
      - gradient decscent — 偏导求导
      - computation graph
    - 梯度消失/爆炸 exploding / vanishing gradient
      - 在多层神经网络中，权重变化可能会呈现指数级变化，当层数太多，可能会导致梯度过大/过小
      - 一定程度上的 解决办法：初始化权重时，将参数初始化为 $Var(w_i) = \frac{2}{n}$
    - Activision fuction
      - output layer
        - sigmoid
          - binary classification
          - multi-label classification — 一个神经网络输出多个问题答案，例如，图片里有车吗，有人吗，有房子吗，答案应该为一个向量例如 [0,0,1]，神经网络的输出层是一个激活函数为 sigmoid 的，有三个neuron的输出层，这与multiclass classification 很像，但请注意不要混淆了
        - softmax — multiclass classification
        - ReLU — non-negative value — 比如房价
        - Linear Activision Func — 线性问题 — 例如股票预测
      - hidden layer
        - most frequently used ReLU
        - 为什么不使用linear function — 因为linear function of linear function == linear function, 这跟直接使用线性回归没有区别
    - regularisation for neural network
      - random dropout
        - 每次训练（过一遍一整个训练集），每个neuron都有一定概率消失，在剩下的neuron组成的网络内训练，然后下次训练，放回原先的neuron，再按一定概率随机去除一些neuron — inverted dropout — 在去除一定比例（例如0.2）的neuron后，把这层剩余的神经元的输出 除以0.8，这样，输出的期望就不会变
        - 为什么dropout可以减少过拟合 — 下一层neuron不会单一依赖来自上层某个neuron的输出，而是倾向于依赖来自上层多个输出，这样自己的每个权重就会均匀分布，不会太大
        - 缺点 — Error和#iteration之间的关系多了随机性，不好从Error - #iteration图观察出过拟合 — 建议不要再测试神经网络的时候使用
      - Data Augmentation — 例如将图像数据拉伸，旋转，缩放，镜像
      - Early Stopping — 画出Error - #Iteration图，找到分岔点，让训练在分岔点停下
      - L2 Regularisation — 添加正则化项
    - Convolutional Neural Network
      - Convolution
        - 使用filter处理原图像，以达到检测edge等目的
          - 卷积后长宽变化 $\lfloor\frac{原边长 + 2\times padding - filter边长}{stride} + 1\rfloor$ — stride 步长
          - 3D filters — 3层filter同时在RGB三层图像上做卷积，最后**全部总和在一起,得出一层结果** — 例如 6x6x3 * 3x3x3 = 4x4x1
          - multi-filters — 多个filter，每个filter在同一个图像上做卷积，每个filter对应一个卷积结果
        - 反向传播的时候，神经网络自动更新卷积层参数
      - Padding — 在原图像周围添加一圈以达到convolution后图像尺寸不变，同时图像最外层的像素值不会被省略
        - Valid — 没有padding
        - Same — 添加到convolution前后图像大小不变
      - Bias — 在卷积后，在所有结果后加个b，然后使用ReLU激活函数 — $ReLU(4\times4\times1+b)$
      - Pooling
        - 窗口滑动计算 — 卷基层对像素位移很敏感，而池化层帮助消除对位移的敏感，同时还能减少后续神经网络的计算
        - Mean pooling — 计算窗口内平均值作为输出 — 保留更多的背景信息
        - Max pooling — 找到窗口内最大值进行输出 — 保留最突出信息，更突出边缘和纹理
    - Recurrent Neural Network
      - 

      - forward propagation
        - $a^{<t>} = g(w_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) = g(w_a[a^{<t-1>}, x^{<t>}] + b_a)$ — 激活函数一般是tanh
        - $y^{<t>} = g(W_{ya}a^{<t>} + b_y)$ — 激活函数可以是sigmoid（如果二分类）， ReLU
      - backword propagation — 逆着时间传播
      - architecture types
        - many-to-many
          - 

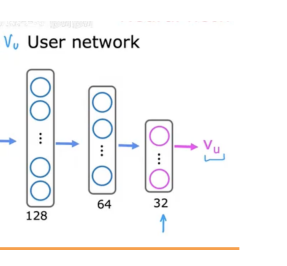
            一个输入对应一个输出
          - 
            如果输入输出长度不一样 — 开始只输入，不输出，后面只输出，不输入 — 前半叫encoder，后半叫decoder
        - 

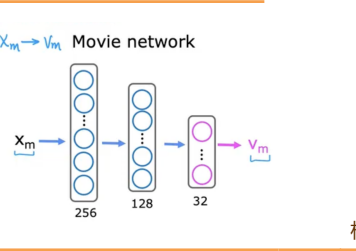
          many-to-one — 输入整个句子后，最后输出一次
        - 
          one-to-many — 最开始一个输入，后面每次输入为上一次输出
      - gradient explode or vanish — 当句子很长的时候，并且最后几个单词对最开始的单词有依赖性，那么就会出现这个问题，梯度太大会这太小
      - Gated Recurrent Unit GRU
        - $\tilde{c}^{<t>} = tanh(w_c[\Gamma_r\times c^{<t-1>}, x^{<t>}] + b_c)$ — candidate memory
        - $\Gamma_r = \sigma(w_r[c^{<t-1>}, x^{<t>}] + b_r)$ — calculating relevance
        - $\Gamma_u = \sigma(w_u[c^{<t-1>}, x^{<t>}] + b_u)$ — gate of memory 是个连续的值,0-1，但是十分接近0和1
        - $c^{<t>} = \Gamma_u\times\tilde{c}^{<t>} + (1-\Gamma_u)\times c^{<t-1>}$ — 门开就记录记忆，门关就不记录
        - $a^{<t>} = c^{<t>}$
      - Long Short Term Memory LSTM
        - $\tilde{c}^{<t>} = tanh(w_c[a^{<t-1>}, x^{<t>}] + b_c)$ — candidate memory
        - $\Gamma_u = \sigma(w_u[a^{<t-1>}, x^{<t>}] + b_u)$ — gate of memory 是个连续的值, 0-1，但是十分接近0和1
        - $\Gamma_f = \sigma(w_f[a^{<t-1>}, x^{<t>}] + b_f)$ — forget gate
        - $c^{<t>} = \Gamma_u\times\tilde{c}^{<t>} + \Gamma_f\times c^{<t-1>}$
        - $\Gamma_o = \sigma(w_o[a^{<t-1>}, x^{<t>}] + b_o)$ — output gate
        - $a^{<t>} = \Gamma_o\times tanh(c^{<t>})$
    - Natural Language Processing NLP
      - word representation
        - one-hot encoding (BoW, Bag of Words)
          - 先建立一个字典，假设有10000个词汇，每个词汇对应一个向量 — 例如 — $man = [0,0,\dots,0,1,0,\dots,0]$ — 这个向量的长度为10000
          - 相似的词汇之间没有逻辑关系，并不能很好的表示词汇之间的关系，例如man，woman，模型不能很好的泛化
        - word embedding
          - 用词汇的特征向量表示词汇，例如，[性别，水果，年龄，....]
          - 相似的词汇之间有数学关系，例如 $v_{man} - v_{women} \approx v_{king} - v_{queen}$
      - 训练embedding matrix E， word2vec
        - 随机初始化E权重，矩阵E的shape为 字典内有多少个词汇 * 每个词汇有多少个特征 — 以及每个词汇的one-hot encoding O_c
          - 例如，假设man在E里属于第6275个单词 — $man = [0_1, 0_2, \dots, 0_{6274}, 1_{6275}, 0_{6276}, \dots, 0_{10000}]$ — 这个向量的长度为10000
          - 那么得到man单词的词汇向量只需计算 $O_{man}\times E$ — 其实就是找到E的第6275列
        - 给句子挖空，拆解成context/target pairs，训练神经网络更新E内的权重
          - 神经网络: $O_c \to E \to Softmax \to \hat{y}$ — Softmax直接分类10000个单词效率低 — hierarchical softmax，类似于决策树
          - 可能的context / target pairs
            - target前几个单词
            - target前后几个单词
            - 随机选择一定窗口内（例如10个单词内）的两个单词，一个作为context，一个作为target — 有更小的可能选择到the，a，of这些常见的单词
        - skip-gram
          - 神经网络: $O_c \to E \to e_c = E\cdot O_c \to Softmax \to \hat{y}$
            - $p(t|c) = \frac{e^{w_t^Te_c}}{\sum_{j=1}^{10000}e^{w_j^Te_c}}$ — Softmax直接分类10000个单词效率低 — hierarchical softmax，类似于决策树
            - $Loss = \sum_{i=1}^{10000} -y_i log\hat{y}_i$
          - 随机选择一定窗口内（例如10个单词内）的两个单词，一个作为context，一个作为target — 有更小的可能选择到the，a，of这些常见的单词
  - Support Vector Machine
    - prediction
      - $\vec{w}\times\vec{x} + b \ge 1$ — 预测 y = 1
      - $\vec{w}\times\vec{x} + b \le -1$ — 预测 y = 0
    - Cost Function
      - Recitifed Binary Cross Entropy — $min_\theta C\sum_{i=1}^{m}[y\times cost_1(\vec{w}\times x+b) + (1-y)\times cost_0(\vec{w}\times x+b)] + \frac{1}{2}\sum_{j=1}^{n}w_j^2$ — $cosy_1(x), cost_0(x)$ == rectified -log sigmoid(x) — $cost_1(x)$
        

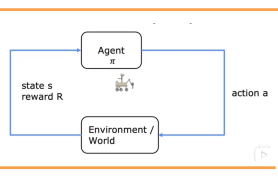
      - 当 C 很小的时候（正则化力度很大），SVM会选取margin最大的decision boundary
    - Kernel
      - 把数据集中每一个点作为一个feature，将要预测的点 x，计算 $l_n = sim(x, f_n)$
        - 用 $\vec{x} = [l_0(=1), l_1, \dots, l_n]$表示$x$ — prediction
          - $\vec{w}\times\vec{x} + b \ge 1$ — 预测 y = 1
          - $\vec{w}\times\vec{x} + b \le -1$ — 预测 y = 0
        - sim函数可以是不同的kernel function
      - Gaussian Similarity — $exp(-\frac{||x-l_n||^2}{2\sigma^2})$
    - multi-classification — 训练多个SVM模型，每个SVM负责将一种类别和其他类别分开
  - Decision Tree
    - 怎么决定每个节点用什么feature分 — Maximize purity
      - 离散的值
        - maximize the purity意味着减少impurity，我们用信息熵来度量impurity，其公式为: — $H(p_1) = -p_1\times log_2(p_1) - (1-p_1)\times log_2(1-p_1)$ — p_1 是 选项一的个数除以总数
        - 计算Information gain来决定使用那个feature，使用information gain最大的feature — $IG = H(p_1^{root}) - (w^{left}\times H(p_1^{left}) + w^{right}\times H(p_1^{right}))$
      - 连续的值 — 尝试不同的分割点，计算Information Gain，使用IG最大的分割点
    - 什么时候**停止**继续往下分
      - 当纯度达到100%
      - 当纯度提升不再明显
      - 当节点深度超过最大深度 — 设置最大深度可以防止过拟合
      - 当落入这个节点的example小于threshold — 防止过拟合
    - One-Hot Encoding — 用0和1 encode features — 例如 — 耳朵的形状: {圆形，尖型，椭圆} 改为 耳朵是圆形吗 :{0,1}，耳朵是尖型吗：{0,1}, 耳朵是椭圆吗:{0，1}
    - Regression Tree
      - 用Decision Tree predicting continuous variable
      - 用数据的方差代替熵
        - 计算加权方差 — $w_{left}\times Var(left) + w_{right}\times Var(right)$
        - 计算减少的方差，使用减少方差最多的feature — $Var(Orginal\ Set) - Weighted\ Variance$
    - Tree ensemble — 构建多个Decision Tree，并在预测时遵循最多的投票结果
    - Random Forest
      - sampling with replacement 放回取样 — 例如，四个颜色的牌 红 黄 绿 蓝，每次取样后放回，连续取样四次，得到 红 绿 蓝 蓝。重复n次，得到n个子数据集
      - 每个子数据集构建一个Decision Tree，在做预测的时候，共同投票决定，投票最多的结果为最终结果
      - 这样得出的Tree大多都很相似，如何避免这个问题? — 每个树在构建的时候都只随机选择一部分feature，作为构建的节点 — 例如原先有100个feature，那么每个树在构建的时候都只随机取10个feature作为分类的节点
    - XGBoost
      - 在resampling 的时候，有更高的几率采样到之前已经构建好的Tree表现不好的数据上
      - 自带regularization，防止过拟合
      - 在处理结构化数据时推荐
- 无监督学习
  - K-means
    - 随机选择K个点作为centroid
    - repeat
      - assign point：把每个点分给最近的centroid，分给同一个centroid的点叫做一个cluster
      - update centroid：计算每个cluster的中心点（mean point），作为新的centroid
    - 一直到每个cluster内的点不在发生变化
    - cost function — distortion algo — $J(c^{(1)},..,c^{(m)},\mu_1,..,\mu_K) = \frac{1}{m}\sum_{i=1}^{m}||x^{(i)} - \mu_{c^{(i)}}||^2$
    - K-means 有可能会卡在local minimum — 解决办法：多跑几次，每次随机初始centroid，选择损失最小的
    - 如何确定K的值
      - 看我们要什么样的结果
      - 或者画出J损失随K的变化图，选择最明显的拐点
  - Anomaly Detection
    - $p_x = \prod_{j=1}^{n}p(x_j;\mu_j,\sigma_j^2)$
      - anomaly if $p_x < \epsilon$
      - 有n个正态分布(mean μ, standard deviation σ)的feature x
    - 无监督学习 特征选择更重要，因为没有一个label来给你确定哪些feature有关，哪些feature无关
    - 如果有的feature不是normal distribution，我们可以对x做一些转换，让其近似于一个normal distribution — 对数据集做转换后，记得对预测值也做相应转换
    - 创造达成目的的新feature — 例如 cpu load / network traffic，针对网络中心里的计算机
  - Autoencoder AE
    - 

    - 神经网络得输入和输出完全相同
      - 前半部分encoder将原始数据压缩成一个向量
      - 后半部分叫做decoder，将向量还原成数据
      - 误差函数可以是还原前后得MSE
    - over complete — 中间的隐藏单元比输入输出多
    - under complete — 中间的隐藏单元比输入少，可以用于压缩
    - 作用
      - 压缩数据，减少数据维度
      - 异常检测，用decoder还原数据，如果还原误差较大,可以认为该数据是异常数据
      - 图片去燥 Denoise AE
- recommender system
  - Collaborative Filtering
    - 

Movie	Alice(1)	Bob(2)	Carol(3)	Dave(4)	x1 (romance)	x2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

    - 预测Alice对Movie 1st的评分 — $w^{Alice}\times x^{(i)} + b^{Alice}$ — $x^{(i)} = [x_1\ x_2]^T$
    - 学习，预测 用户评分 w_A, b_A, w_B, b_B — cost function
      - 对于单个用户j — $J(w^j,b^j) = \frac{1}{2}\sum_{i:r(i,j)}(w^{(j)}\times x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(w_k^{(j)})^2$
      - 对于所有用户来说 — $J\begin{pmatrix}w^{(1)} & \dots & w^{(n_u)}\\ b^{(1)} & \dots & b^{(n_u)}\end{pmatrix} = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)}(w^{(j)}\times x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(w_k^{(j)})^2$
    - 学习电影情感分数 x_1, x_2 — cost function — $J(x_1,\dots x_{n_m}) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r(i,j)}(w^{(j)}\times x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$
    - 以上公式综合起来的
      - cost function — $\frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)}(w^{(j)}\times x^{(i)} + b^{(j)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(w_k^{(j)})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^{(i)})^2$
      - Gradient Descent
        - $w_i^{(j)} = w_i^{(j)} - \alpha\frac{\delta J(\vec{w},b,\vec{x})}{\delta w_i^{(j)}}$
        - $b^{(j)} = b^{(j)} - \alpha\frac{\delta J(\vec{w},b,\vec{x})}{\delta w^{(j)}}$
        - $x_i^{(j)} = x_i^{(j)} - \alpha\frac{\delta J(\vec{w},b,\vec{x})}{\delta x_i^{(j)}}$
    - for binary labels — Loss function — $L(f_{w,b,x}(x), y_{(i,j)}) = -y^{(i,j)}\log(f_{w,b,x}(x)) - (1-y^{(i,j)})\log(1-f_{w,b,x}(x))$ — $f_{w,b,x}(x) = Sigmoid(w^{(j)}\times x^{(i)} + b^{(j)})$
    - mean normalization — 如果一个用户还没有给任何一个电影打分，那么用现有所有该电影的评分的平均值代表其评价 — 为什么我们需要 mean normalisation
      - 更符合直觉，更合理
      - 让协同过滤运行的更快
    - 缺点
      - cold start problem — 新电影预测不准
      - 缺少side information — 并不能知道电影或者是用户的 side information，例如用户多大，喜欢什么电影
  - Content-based filtering
    - 收集用户和电影的不同特征，例如
      - $x_u =$ [用户的性别，年龄，......，国籍，看过的电影，对各个类别电影的平均评分]
      - $x_m =$ [电影的类别，出版日期，......，平均评分]
    - 使用两个神经网络分别将两组特征转换为**相同维度的**特征向量，例如
      - 

        $f(x_u) = v_u$
      - 

        $f(x_m) = v_m$ — 相似的特征向量意味着相似的电影 — $||v_m^k - v_m^i||^2$ small
    - 计算两个特征的点积作为预测结果 — $v_u\cdot v_m$
    - cost function — $J = \sum_{(i,j):r(i,j)=1}(v_u^{(j)}\cdot v_m^{(i)} - y^{(i,j)}) + NN\ Regularization\ term$
    - 从大目录里推荐
      - retrieval step
        - 用常理选出一些名单，例如
          - 与用户最后一个看的电影最相似的10个电影
          - 现在最火的10部电影
          - 该国家最火的10部电影
        - 去除其中重合的部分
      - ranking step — 把选出来的电影和用户的特征向量做点积，按结果从高到低排列
- reinforcement learning
  - 概念
    - states — 所有可能的状态 — 例如 直升机的高度，朝向，前/后飞，左/右滚
    - actions — 所有可以采取的行动 — 例如 向左，向右
    - rewards R — 所有可能的回报 — 例如 100，0，40
    - discount factor γ — 回报随时间的折现率
    - return — 折现后的回报 — $return = R_1 + \gamma R_2 + \gamma^2R_3 + \dots$
    - policy π — 采取行动的规则 — 例如 去最近的奖励，去return最高的奖励
    - Markov Decision Process （MDP）
      - 未来的状态 s' 只取决于现在的状态 s 以及即将采取的行动 a
      - 

    - $Q(s,a)$ — 能获得的最好的return，如果你在state s，采取action a后
  - Bellman equation
    - $Q(s,a) = R(s) + \gamma\times max_{a'}Q(s',a')$
      - s': 在采取a行动后的状态
      - a': 在采取a行动后，下一步动作
    - 在不确定的（stochastic）环境中 — $Q(s,a) = R(s) + \gamma\times E[max_{a'}Q(s',a')]$
  - Deep Reinforcement Learning
    - 用到的神经网络
      - 输入，火箭的状态s = [高度，角度，...]
      - 中间：任意层隐藏神经元
      - 输出层：单个神经元，输出 预测Q（s，a)
        - a可以是 nothing，left，main，right
        - 改进：输出层四个神经元，分别对应Q(s,nothing); Q(s,left); Q(s,main); Q(s,right)
    - 算法
      - 随机初始化神经网络参数，得到 Q
      - 重复
        - 随机尝试不同的动作
        - 记录最近10000（replay buffer）条数据，其前后状态，动作，以及回报 $(s,a,R(s),s')$
        - 训练神经网络 Q_{new} — 用以上随机尝试得到的例子
          - $x = (s,a)$
          - $y = R(s) + \gamma\times max_{a'}Q(s',a')$
          - 训练新的神经网络使其 $Q_{new}(s,a) \approx y$
        - set $Q = Q_{new}$
  - epsilon-greedy policy
    - 有epsilon的概率随机做选择 — 有1-epsilon的概率做回报最大的选择
    - 为什么我们需要epsilon greedy policy — 为了避免初始化神经网络后，神经网络从来不做某些决策
    - 在训练的开始，建议把epsilon设置得很大，随着训练慢慢减小
  - mini-batch — 每次训练神经网络，把replay buffer分为几个子集，用子集挨个训练神经网络，更新权重
  - soft update
    - 在set Q = Q_new的时候
      - $w = 0.99\times w_{old} + 0.01\times w_{new}$
      - $b = 0.99\times w_{old} + 0.01\times w_{new}$
    - 防止神经网络训练结果反而更差