

机器学习Machine Learning内容准备

[Scikit learn中文社区：](#)

[一文读懂机器学习分类算法：](#)

机器学习中问题分类：

回归还是分类 (regression or classification)

1. Regression: predict continuous values
 1. What is the value of house in California?
 2. What is the probability that a user will click on this ad?
2. Classification
 1. Is a given email spam or not?
 2. Is this an image of dog, cat, or hamster?

机器学习关键点

1. 数据 (data) 从过去的观察中收集数据 (训练集)
2. 模型：用于模拟数据中的规律，devised to capture the patterns in the data
3. 预测：把模型用于未来数据预测

机器学习模型主要分类

Supervised Learning (监督学习)

- 根据之前的数据来构造模型，对未来数据进行预测

Unsupervised Learning (无监督学习)

- 常用于发现数据中的模式

Reinforcement Learning (加强学习)

- 在不确定情况下 (有限知识下) 进行动态预测

机器学习主要模型

Nearest Neighbor Classifier (近邻算法)

训练集：

- N数据: $D^{TRAIN} = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$
- 每一个 $x_n \in \mathbb{R}^D$ 都是一个特征向量(feature vector)
- 每一个 $y_n \in [C] = \{1, 2, \dots, C\}$ 都称之为一个标签/类/种类 (label/class/category)
 - 当 $C = 2$ 时候我们称之为二分类 binary classification
 - 常用符号: $\{0, 1\}, \{-1, 1\}$
- 用于学习一个function $f: \mathbb{R}^D \rightarrow [C]$ 用于未来的预测

Nearest Neighbor

- 总结概括：针对一个新给的点，找到距离它最近的K个点，然后这K个点投票决定新的点的类别
- Hyperparameter: K (K nearest neighbor)，因为要投票，所以K一般选择奇数
- $x(1) = x_{nn(x)}$ where $nn(x) \in [N] = \{1, 2, \dots, N\}$, i.e. the index to one of the training instance

- $nn(x) = \operatorname{argmin}_{n \in [N]} \|x - x_n\|_2 = \operatorname{argmin}_{n \in [N]} \sqrt{\sum_{d=1}^D (x_d - x_{nd})^2}$, where we use L2/Euclidean distance for $\|\cdot\|_2$
- Classification Rule:
 - $y = f(x) = y_{nn(x)}$
- 所有的NNC算法都会给我们生成一个决策边界(decision boundary)
 - 随着K的增大, 边界越发平滑

测试集: 用于衡量我们的算法是否准确

- M samples $D^{TEST} = \{(x_1, y_1), (x_2, y_2), \dots, (x_M, y_M)\}$
- 全新的数据, 和训练集不应该有任何重叠 (no overlap with training set)
- 通过测试集的准确率 (accuracy) 来衡量我们的模型是否精准
- 测试集的结果 make sense 仅当训练数据和测试数据 correlated
 - 更标准的假设: 所有的点 (任意集合中) 都是 i.i.d
- 这样一来测试误差对于固定的分类器而言就是一个随机变量(RV)

不同的距离: 之前使用 Euclidean distance, 两点之间的直线距离

- Manhattan distance, 走格子的长度 $\|x - x_n\|_1 = \sum_{d=1}^D |x_d - x_{nd}|$
- Generally L_p distance: $\|x - x_n\|_p = (\sum_d |x_d - x_{nd}|^p)^{1/p}$

一些数据预处理

- Normalize data
 - $\bar{x}_d = \frac{1}{N} \sum_n x_{nd}$
 - $s_d^2 = \frac{1}{N-1} \sum_n (x_{nd} - \bar{x}_d)^2$
 - $x_{nd} \leftarrow \frac{x_{nd} - \bar{x}_d}{s_d}$

验证集 Development/Validation data

- L samples: $D^{DEV} = \{(x_1, y_1), (x_2, y_2), \dots, (x_L, y_L)\}$
- 用来调整超参数
- 和训练集合&测试集合都不应该有重叠

训练模式

- 训练集合用于训练, 验证集用于调整训练集的超参数, 最后用测试集来衡量模型好坏

S-fold cross validation 交叉验证

- 将数据分成 S 个集合
- 每一次用一个集合来做验证集, 剩余做测试集
- 使用平均表现最好的超参数
- Special case: $S = N$ 我们称之为 leave-one-out

NNC的优势与劣势

优势:

- 简单容易完成

劣势

- 计算花费时间, $O(ND)$ for each prediction naively
 - 因为我们需要计算每一个需要预测点和所有点之间的距离
 - Need to "carry" the training data around. 称之为 Nonparametric 方法
 - 样本本身不训练, 只是记忆住了训练集

NNC补充

- 所有特征应当被放在同一个量级
- 投票结果近的数据可以根据距离来归一化，这样距离越近的投票比重越大
- K取值
 - K小，低偏移，高方差
 - K大，高偏移，低方差

预期错误率 Expected Error

- 测试集error的expectation (因为他是随机变量，所以有期望值)
 - $\mathbb{E}[\epsilon^{TEST}] = \mathbb{E}_{(x,y) \sim P} \mathbb{I}[f(x) \neq y]$
 - The larger the test set, the better approximation
- Expected Error
 - Loss function, e.g.: $L(y', y) = \mathbb{I}[y' \neq y]$ called 0-1 loss
 - Define: the expected risk of f is:
 - $R(f) = \mathbb{E}_{(x,y) \sim P} L(f(x), y)$

线性回归Linear regression

e.g. 房屋价格表

- 给你房子的面积和销售价格，我们假设面积和销售价格成线性关系，尝试构造一条直线，根据未来房屋面积来预测销售价格
- How to measure error
 - absolute error: $|prediction - sale price|$
 - squared error: $(prediction - sale price)^2$ most common

Formal Setup for Linear regression

- Input: $x \in \mathbb{R}^D$ (features, covariates, context, predictions, etc)
- Output: $y \in \mathbb{R}$ (Response, targets, outcomes, etc)
- Training Data: $D = \{(x_n, y_n), n = 1, 2, 3, \dots, N\}$
- Linear Model: $f: \mathbb{R}^D \rightarrow \mathbb{R}$ with $f(x) = w_0 + \sum_{d=1}^D w_d x_d = w_0 + \mathbf{w}^T \mathbf{x}$
 - hyper-plane parametrized by \mathbf{w} and bias w_0
 - for notation convenience, we append 1 at the beginning, so we don't need to write bias outside
 - $\mathbf{w} = [w_0 \ w_1 \dots \ w_D]^T$ with $D + 1$ parameters, $\mathbf{x} = [1 \ x_1 \dots \ x_D]$
 - model becomes $f(x) = \mathbf{w}^T \mathbf{x}$

目标GOAL

- Minimize total squared error
 - Residual Sum of Squares (RSS) a function of \mathbf{w}
 - $RSS(\mathbf{w}) = \sum_n (f(\mathbf{x}_n) - y_n)^2 = \sum_n (\mathbf{x}_n^T \mathbf{w} - y_n)^2$
 - find $\mathbf{w} = argmin RSS(\mathbf{w})$
 - least mean squares solution, (empirical rist minimizer) 最小二乘法

how to reach the goal

- linear regression有直接的结果
- when D = 0 constant prediction, 无论x是什么，结果都是y
 - $RSS(w_0) = \sum_n (w_0 - y_n)^2 = Nw_0^2 - 2(\sum_n y_n)w_0 + cnt$
 - we have: $w_0^* = \frac{1}{N} \sum_n y_n$, i.e. the average
- when D = 1

- $RSS(w_0) = \sum_n (w_0 + w_1 x_n - y_n)^2$
- Find stationary points, points with zero gradient
 - $\frac{\partial RSS(\mathbf{w})}{\partial w_0} = 0 \rightarrow \sum_n (w_0 + w_1 x_n - y_n) = 0$
 - $\frac{\partial RSS(\mathbf{w})}{\partial w_1} = 0 \rightarrow \sum_n (w_0 + w_1 x_n - y_n) x_n = 0$
- stationary points就是最小minimizer吗?
 - 不一定, 对于convex的是的, RSS是convex的
- General Solution
 - $X^T X = \sum_n x_n x_n^T$
 - $\nabla RSS(w) = (X^T X)w - X^T y = 0$
 - $w^* = (X^T X)^{-1} X^T y$
 - Assuming $X^T X$ the covariance matrix is invertible
- What is the covariance matrix is not invertible
 - 为什么会这样:
 - 当N < D + 1, 数据的数量小于参数的数量, 比如我们现在有1个点, 但是我们同时有1个特征, 那么在平面内, 一个点不能确定唯一的直线
 - 如何解决:
 - not invertible:
 - the eigendecomposition $X^T X = U^T EU$
 - 其中E的对角线都是X的eigenvalue, 从左上角到右下角 $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_{D+1} \geq 0$
 - Inverse需要把E都变成1/E
 - non invertible意味着有eigenvalue是0
 - 所以我们就加上一些noise $\lambda > 0$, Hyperparameter, can be tuned by cross-validation
 - Solution becomes: $w^* = (X^T X + \lambda I)^{-1} X^T y$
 - 这时候它不再是minimizer对于原来的RSS
 - trade off:
 - easy to solve but not precise
 - not easy to solve but precise

和NNC做对比

parametric versus non-parametric

- parametric: 随着数据量的增加, 模型的大小不会增加
 - 线性回归, D+1个parameters, 和数据量的大小N无关
- Non parametric: 随着数据量的增加, 模型大小也会改变
 - NNC, 必须保证测试集不变之后才能进行预测, 模型的大小就是测试集数据量的大小

非线性数据中的线性回归 Linear regression with nonlinear basis

使用非线性映射 non linear mapping

- $\phi(x) : x \in \mathbb{R}^D \rightarrow z \in \mathbb{R}^M$
- 然后我们再使用线性回归, 希望在新的特征空间中线性回归能很好映射
- 模型 $f(x) = w^T \phi(x)$ where $w \in \mathbb{R}^M$
- 目标 $RSS(w) = \sum_n (w^T \phi(x_n) - y_n)^2$
- 同样的解:
 - $w^* = (\Phi^T \Phi)^{-1} \Phi^T y$ where $\Phi = [\phi(x_1)^T \ \phi(x_2)^T \ \dots \ \phi(x_N)^T]^T \in \mathbb{R}^{N \times M}$
 - E.g. polynomial basis function for D = 1

$$\Phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^M w_m x^m$$

- 新的空间中学习一个线性模型等价于在原来模型中学习一个M-degree的多项式模型

- 注意, 使用线性映射, fancy linear map没有任何帮助, 相当于什么都没做

- 过拟合欠拟合, 还是用我们上面提到的M作为类比

- 当M很小的时候, 欠拟合

- 训练误差和测试误差都很大

- 当M很大的时候, 过拟合

- 训练误差很小, 但是测试误差很大

- 越复杂的模型 \Rightarrow 训练和测试之间的误差会越来越大

- 解决方法

- 更多数据, Golden Rule

- 通过交叉验证法来确定degree M

- 正则化, regularization

- 新目标: $\varepsilon(w) = RSS(w) + \lambda R(w)$, find $w^* = argmin_w \varepsilon(w)$

- $R : \mathbb{R}^D \rightarrow \mathbb{R}^+$ is the regularizer

- 用于测量模型的复杂程度

- 常选择: $\|w\|_2^2, \|w\|_1$ etc

- $\lambda > 0$ is the regularization coefficient

- $\lambda = 0$ no regularization

- $\lambda \rightarrow +\infty, w \rightarrow argmin_w R(w)$

- It's like the TRADE_OFF between training error and complexity

- Solve the formula: $w^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y$

线性分类

二分类

- 类别数量 $C = 2$, labels: $\{-1, +1\}$ (cat or dog, fraud or not, up or down)

- 第一步, 选择模型

- 线性模型, 如何使用线性 $w^T x$ 来预测label? Sign of $w^T x$

$$sign(w^T x) = \begin{cases} +1 & \text{if } w^T x > 0 \\ -1 & \text{if } w^T x \leq 0 \end{cases}$$

- 第二步, 选择损失函数 loss function $L(y', y)$

- 0-1 loss: $L(y', y) = \mathbb{I}[y' \neq y]$

- 对于分类任务, 简便起见, 我们常常把loss看作是 $y w^T x = z$ 的一个函数

$$l_{0-1}(z) = \mathbb{I}[z \leq 0] = \begin{cases} +1 & \text{if } z \leq 0 \\ -1 & \text{if } z > 0 \end{cases}$$

- 问题: 0-1 loss不是convex的, minimizing 01 loss is NP-hard, we use Surrogate loss

- Perceptron loss: $l_{perceptron}(z) = max\{0, -z\}$ used in perceptron

- 0: predicting correctly

- $-z$: predict wrong, 这个更清楚的告诉你“错了多少”

- hinge loss: $l_{hinge}(z) = max\{0, 1 - z\}$ used in SVM and many others

- 对比perceptron loss, 向右平移一个单位

- logistic loss: $l_{logistic}(z) = log(1 + exp(-z))$ used in logistic regression, base of log doesn't matter

- 第三步, 找minimizer

$$w^* = argmin_{w \in \mathbb{R}^D} \sum_{n=1}^N l(y_n w^T x_n)$$

- where $l(\cdot)$ can be perceptron/hinge/logistic loss
- no closed form in general
- 可以使用general convex optimization methods
- 在perceptron loss上minimize not make sense, 之要 $w = 0$ 就已经是最小了

perceptron

- find the minimiser of: $F(w) = \sum_{n=1}^N l_{perceptron}(y_n w^T x_n) = \sum_{n=1}^N \max\{0, -y_n w^T x_n\}$
- Gradient Descent: move a bit in the negative gradient direction
 - $w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla F(w^{(t)})$
 - where $\nabla > 0$ is called step size or learning rate
- GD on perceptron loss
 - $\nabla F(w) = \sum_{n=1}^N \max\{0, -y_n w^T x_n\} = \sum_{n=1}^N -\mathbb{I}[y_n w^T x_n \leq 0] y_n x_n$
 - GD update $w \leftarrow w + \eta \mathbb{I}[y_n w^T x_n \leq 0] y_n x_n$
 - 注意 η 前面应该是负号, 但是被导数中的负号给抵消了
 - 很慢! 每次都需要过完所有的训练数据才能进行一次update
- SGD on perceptron loss
 - pick one example $n \in [N]$ uniformly at random
 - let $\tilde{\nabla} F(w) = -N \mathbb{I}[y_n w^T x_n \leq 0] y_n x_n$
 - $w \leftarrow w + \eta N \mathbb{I}[y_n w^T x_n \leq 0] y_n x_n = w \leftarrow w + \tilde{\eta} \mathbb{I}[y_n w^T x_n \leq 0] y_n x_n$
 - 每接触一个data我们就更新一次
- batch stochastic gradient descent
 - GD和SGD中间, 选择一个batch size数量的数据, 计算然后更新

逻辑回归Logistic Regression

- find the minimizer of $F(w) = \sum_{n=1}^N l_{logistic}(y_n w^T x_n) = \sum_{n=1}^N \ln\{1 + e^{-y_n w^T x_n}\}$
- 我们不再是预测label, 而是预测每个label出现的概率regress the probability
- sigmoid function + linear model
 - sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$
 - $\sigma(w^T x) \geq 0.5 \Leftrightarrow w^T x \geq 0$ 和预测label所用的 $sign(w^T x)$ 一致
 - $w^T x$ 越大 $\Rightarrow \sigma(w^T x)$ 越大 \Rightarrow 我们对于预测label是1的自信心越大 (概率越大)
 - $\sigma(z) + \sigma(-z) = 1$ for all z
 - 所以一个label是1的概率+label是-1的概率=1 (符合我们对于概率的定义)
- 问题: 我们观察到的数据结果都是label = -1,+1, 而不是概率, 如何regress
 - Maximum likelihood estimation
 - $P(w) = \prod_{n=1}^N \mathbb{P}(y_n | x_n; w)$
 - $w^* = argmax_w P(w) = argmax_w \prod_{n=1}^N \mathbb{P}(y_n | x_n; w)$
 - $= argmax_w \sum_{n=1}^N ln \mathbb{P}(y_n | x_n; w) = argmin_w \sum_{n=1}^N -ln \mathbb{P}(y_n | x_n; w)$
 - $= argmin_w \sum_{n=1}^N -ln(1 + e^{-y_n w^T x_n}) = argmin_w F(w)$
 - So, 最小化logistic loss正是在sigmoid函数上做MLE
 - SGD

$$\begin{aligned}
w &\leftarrow w - \eta \tilde{\nabla}_w l_{logistic}(y_n w^T x_n) \\
&= w - \eta \left(\frac{-e^{-z}}{1 + e^{-z}} \Big|_{z=y_n w^T x_n} \right) y_n x_n \\
&= w + \eta \sigma(-y_n w^T x_n) y_n x_n \\
&= w + \eta \mathbb{P}(-y_n | x_n; w) y_n x_n
\end{aligned}$$

Newton method

- SDE使用的是1阶导数, newton用二阶导数
- $H_t = \nabla^2 F(w^{(t)}) \in \mathbb{R}^{D \times D}$ is the Hessian of F at $w(t)$ where
 - $H_{t,ij} = \frac{\partial^2 F(w)}{\partial w_i \partial w_j} \Big|_{w=w^{(t)}}$
 - double derivative according to different pairs of weights
 - for convex F, (so H_t is positive semidefinite) https://en.wikipedia.org/wiki/Definite_matrix
 - $w^{(t+1)} \leftarrow w^{(t)} - H_t^{-1} \nabla F(w^{(t)})$
 - no learning rate!
 - converge super fast in terms of #iteration needed

多种类分类模型

多种类逻辑回归

- 现在种类数量上升, 不再是binary classification
- 从二分法总结:
 - $f(x) = \begin{cases} 1 & \text{if } w^T x > 0 \\ 2 & \text{if } w^T x \leq 0 \end{cases} \rightarrow \begin{cases} 1 & \text{if } w_1^T x > w_2^T x \\ 2 & \text{if } w_1^T x \leq w_2^T x \end{cases} = \operatorname{argmax}_{k \in \{1,2\}} w_k^T x \text{ for any } w_1, w_2 \text{ s.t. } w = w_1 - w_2$
 - 把 $w_k^T x$ 想像成对于 class k 的分数
 - $\mathbb{P}(y=1|x; w) = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}} = \frac{1}{1+e^{-(w_1-w_2)^T x}} = \frac{1}{1+e^{-w_1^T x+w_2^T x}} = \frac{e^{w_1^T x}}{e^{w_1^T x}+e^{w_2^T x}} \propto e^{w_1^T x}$
- 多种类逻辑回归
 - Naturally, for multi class: $\mathbb{P}(y=k|x; w) = \frac{e^{w_k^T x}}{\sum_{k' \in [C]} e^{w_{k'}^T x}} \propto e^{w_k^T x}$ We called this Softmax Function
 - We can apply MLE on this $\mathbb{P}(y=k|x; w)$, $P(w) = \prod_{n=1}^N \mathbb{P}(y=n|x_n; w)$, want to maximize this
 - By taking negative log, equivalent to minimizing:
 - $F(W) = \sum_{n=1}^N \ln \left(\frac{\sum_{k' \in [C]} e^{w_{k'}^T x_n}}{e^{w_{y_n}^T x_n}} \right) = \sum_{n=1}^N \ln \left(1 + \sum_{k \neq y_n} e^{(w_k - w_{y_n})^T x_n} \right)$
 - This is called the multiclass logistic loss, a.k.a cross-entropy loss
 - Apply SGD, 假设 $g(W) = \ln \left(1 + \sum_{k \neq y_n} e^{(w_k - w_{y_n})^T x_n} \right)$, 它的导数是?
 - Focus on the k'th row
 - if $k \neq y_n$: $\nabla_{w_k} g(W) = \frac{e^{(w_k - w_{y_n})^T x_n}}{1 + \sum_{k' \neq y_n} e^{(w_{k'} - w_{y_n})^T x_n}} x_n^T = \mathbb{P}(y=k|x; W) x_n^T$
 - else $\nabla_{w_k} g(W) = \frac{-\sum_{k' \neq y_n} e^{(w_{k'} - w_{y_n})^T x_n}}{1 + \sum_{k' \neq y_n} e^{(w_{k'} - w_{y_n})^T x_n}} x_n^T = (\mathbb{P}(y_n|x_n; W) - 1) x_n^T$
 - 每一个 w_{y_n} 前面都有个符号, 所以分子是负的
- 在我们通过SGD学习过W之后我们可以
 - make a deterministic prediction $\operatorname{argmax}_{k \in [C]} w_k^T x$
 - make a randomized prediction according to $\mathbb{P}(k|x; W) \propto e^{w_k^T x}$

从多种类降低到二分

- 一对多分类
 - 假设我们有C个种类, 我们建造C个二分类器, 每个学习“这个东西是不是属于第k个类? ”
 - 对于每个example我们有k个label, 每个label对应它是否是类k
 - 然后针对每个example具体分类时候, 我们在它属于的类中随机挑选一个座位真正label进行输出
- 一对一分类

- 训练C choose 2个训练器去学习“这个东西是k类还是k'类
- 对于每个新的example，我们对每个训练器问：这个东西是k还是k'，然后投票出结果
- 对比一对多，更robust，但是更慢
- Error-Correcting Output Codes (ECOC)
 - somehow将class编译成binary code的形式，来进行预测输出
 - 例如：对于digit分类，第一个bit可以作为奇数偶数的分类，奇数为1，偶数为0
 - 多个分类器ensemble最后出来的结果
 - code越不一致越好
- 树降低tree-based reduction
 - 训练C个二分类器，学习当前example属于哪个半边

Reduction	#Training points	Test time	remark
OvA	CN	C	not robust
OvO	C^2N	C^2	可以达到很小的训练误差
ECOC	LN	L	在设计code的时候需要多样性
Tree	(log_2C)N	log_2C	对于“极端分类”很好用

神经网络

Universal Approximation theorem (Cybenko, 89; Hornik 91)

- 一个向前传播的神经网络和一个隐藏层，可以近似所有的连续函数
 - 当然它可能需要很多神经单位，而且深度肯定也有帮助

核方法Kernel Method

- 之前问题：我们要如何选择一个非线性函数 $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$? 使用核方法
- 例子：正则化线性回归 $F(\mathbf{w}) = \|\Phi\mathbf{w} - \mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_2^2$
 - $w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^T\mathbf{y}$ 最优解

$$\Phi = \begin{bmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$
 - 如果我们令 $F(w)$ 的gradient 是 0
 - $\Phi^T(\Phi w^* - \mathbf{y}) + \lambda w^* = 0 \Rightarrow w^* = \frac{1}{\lambda} \Phi^T(\mathbf{y} - \Phi w^*) = \Phi^T\alpha = \sum_{n=1}^N \alpha_n \phi(x_n)$
 - 最小二乘法的解，其实是所有特征的线性结合
 - 假设我们知道 α 是什么，对于一个新的例子 x 在 w^* 上的预测变成
 - $w^{*T}\phi(x) = \sum_{n=1}^N \alpha_n \phi(x_n)^T \phi(x)$
 - 这样一来我们不需要知道 w^* 是什么，只需要新的特征空间的内积即可
 - 核方法就是在不知道 ϕ 是什么的前提下直接计算内积
 - 但什么是 α
 - 将 $w = \Phi^T\alpha$ 代入到 $F(w)$

$$G(\alpha) = F(\Phi^T\alpha) = \|\Phi\Phi^T\alpha - \mathbf{y}\|_2^2 + \lambda\|\Phi^T\alpha\|_2^2$$

$$= \|K\alpha - \mathbf{y}\|_2^2 + \lambda\alpha^T K \alpha \quad (K = \Phi\Phi^T)$$

$$= \alpha^T(K^2 + \lambda K)\alpha - 2\mathbf{y}^T K \alpha + \text{cnt.}$$
 - We call this Dual Formulation of linear regression
 - K 我们称之为 Gram Matrix 或者 Kernel Matrix，其中 (i, j) entry 是 $\phi(x_i)^T \phi(x_j)$
 - Gram Matrix versus Covariance Matrix

	dimension	entry(i, j)	property
$\Phi\Phi^T$	$N * N$	$\phi(x_i)^T \phi(x_j)$ 把向量直接相乘	两者都是对称并且 positive semidefinite
$\Phi^T\Phi$	$M * M$	$\sum_{n=1}^N \phi(x_n)_i \phi(x_n)_j$ 把每个向量的i和j位置拿出来相乘，最后在相加	

- 继续寻找 α

- 最小化 dual formulation, 将它关于 α 的导数设置为 0
- $0 = (K^2 + \lambda K)\alpha - Ky = K((K + \lambda I)\alpha - y) \Rightarrow \alpha = (K + \lambda I)^{-1}y$ is the minimizer
- 将 α 代入 w^* 我们有 $w^* = \Phi^T\alpha = \Phi^T(\Phi\Phi^T + \lambda I)^{-1}y$
- 之前结论: $w^* = (\Phi^T\Phi + \lambda I)^{-1}\Phi^Ty$
- 它们两个是一致的, 但是当 $N <= M$ 的时候, 计算 $(\Phi\Phi^T + \lambda I)^{-1}$ 要更快, 并且在新的特征空间里, 计算这个只需要内积
- 所以 $\phi(\cdot)$ 真正是什么不重要, 重要的是计算内积 $\phi(x)^T \phi(x')$ This is called the Kernel Trick

- 核函数 Kernel functions

- 定义: a function $k : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ is called a (positive definite) kernel function if there exists a function $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ so that for any $x, x' \in \mathbb{R}^D$ we have $k(x, x') = \phi(x)^T \phi(x')$
- 现在选择 nonlinear basis 变成选择核函数
- 核矩阵的(i,j) entry 变成 $k(x_i, x_j)$
- Mercer theory: k is kernel $\Leftrightarrow K$ is positive semidefinite for any N and any x_1, x_2, \dots, x_N
- 在新的 example x 上的预测结果是
 - $w^{*T} \phi(x) = \sum_{n=1}^N \alpha_n \phi(x_n)^T \phi(x) = \sum_{n=1}^N \alpha_n k(x_n, x)$
- Property: 如果 k_1 和 k_2 都是核函数, 则以下都是核函数
 - $\alpha k_1 + \beta k_2$ if $\alpha, \beta \geq 0$
 - $k_1 k_2$
 - e^k

支持向量机 Support Vector Machine

二分支持向量机 Primal Formulation

- 一句话总结: 使用 hinge loss, L2 正则化的线性模型
 - hinge loss: $l_{hinge}(z) = \max\{0, 1 - z\}$
- 对于线性模型 (w, b) 这意味着:
 - $\min_{w,b} \sum_n \max\{0, 1 - y_n(w^T \phi(x_n) + b)\} + \frac{\lambda}{2} \|w\|_2^2$
 - $y_n \in \{-1, +1\}$
 - non linear mapping ϕ
- 几何解释:
 - 当我们的数据线性可分的时候, 有无数个超平面可以做到没有训练误差, 那我们需要哪一个超平面
 - 距离 data 最远的那个超平面
- 点 x 到超平面 $\{x : w^T x + b = 0\}$ 的距离
 - 假设我们点到平面的映射是 $x - l \frac{w}{\|w\|_2^2}$
 - 把这个映射的点代入超平面的方程: $w^T(x - l \frac{w}{\|w\|_2^2}) + b = w^T x - l \|w\| + b \Rightarrow l = \frac{w^T x + b}{\|w\|_2}$
 - 所以距离 l 是 $\frac{|w^T x + b|}{\|w\|_2}$
 - 对于一个将 (x, y) 正确分类的超平面, 距离为 $\frac{y(w^T x + b)}{\|w\|_2}$
- 我们的目标, maximizing the margin
 - margin 的定义: 训练集中所有点到超平面距离最小的那个距离
 - $\text{MARGIN OF } (w, b) = \min_n \frac{y_n(w^T \phi(x_n) + b)}{\|w\|_2}$
 - Goal: want to solve: $\max_{w,b} \min_n \frac{y_n(w^T \phi(x_n) + b)}{\|w\|_2} = \max_{w,b} \frac{1}{\|w\|_2} \min_n y_n(w^T \phi(x_n) + b)$
 - Note: rescaling (w, b) 不会改变我们的 hyperplane, 所以我们可以 rescale 来保持 $\min_n y_n(w^T \phi(x_n) + b) = 1$
 - 举例, 假设我们在二维坐标系中, 我们的超平面是一条直线, 45 度, $x = y$ 转换成 $x_1 - x_2 = 0$, 这个时候 $2x_1 - 2x_2 = 0$ 但是这并不

会改变我们这条超平面直线的坐标位置，它一直是45度角从左下到右上

- Margin becomes: $\frac{1}{\|w\|_2}$
- Goal:
 - $\max_{w,b} \frac{1}{\|w\|_2}$ such that $\min_n y_n(w^T \phi(x_n) + b) = 1$
 - 这等价于 $\min_{w,b} \frac{1}{2} \|w\|_2^2$ such that $y_n(w^T \phi(x_n) + b) \geq 1 \quad \forall n$
 - 1/2 is for easier derivative
 - SVM也被称为max-margin分类器，以上的限制条件叫做hard margin条件
- 如果数据不是线性可分，那么 $y_n(w^T \phi(x_n) + b) \geq 1 \quad \forall n$ 明显是不能达到的
 - 我们引入一个soft margin限制条件
 - $y_n(w^T \phi(x_n) + b) \geq 1 - \xi_n \quad \forall n$ where ξ is called the slack variables $\xi \geq 0$
 - 我们希望 ξ 越小越好，所以我们的目标变成
 - $\min_{w,b,\{\xi_n\}} \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$ such that $y_n(w^T \phi(x_n) + b) \geq 1 - \xi_n \quad \forall n, \xi_n \geq 0, \forall n$
 - where C is the hyperparameter to balance the two goals
 - 这个目标等价于
 - $\min_{w,b} \frac{1}{2} \|w\|_2^2 + C \sum_n \max\{0, 1 - y_n(w^T \phi(x_n) + b)\} = \xi_n, \forall n$
 - 等价于
 - $\min_{w,b} C \sum_n \max\{0, 1 - y_n(w^T \phi(x_n) + b)\} + \frac{1}{2} \|w\|_2^2$
 - 或者说：
 - $\min_{w,b} \sum_n \max\{0, 1 - y_n(w^T \phi(x_n) + b)\} + \frac{\lambda}{2} \|w\|_2^2$ with $\lambda = 1/C$
 - 这和最小化L2正则的hinge loss一模一样

Lagrangian Duality 拉格朗日对偶性

- 基础问题：我们想解决：
 - $\min_w F(w)$ such that $h_j(w) \leq 0, \forall j \in [J]$, where functions h_1, \dots, h_J 叫做constraints
 - 很明显SVM是当 $J = 2N$ 时候的一种情况
 - $F(w, b, \{\xi_n\}) = C \sum_n \xi_n + \frac{1}{2} \|w\|_2^2$
 - $h_n(w, b, \{\xi_n\}) = 1 - y_n(w^T \phi(x_n) + b) - \xi_n, \forall n \in [N]$
 - $h_{N+n}(w, b, \{\xi_n\}) = -\xi_n, \forall n \in [N]$
- Lagrangian:
 - The lagrangian of previous problem is:
 - $L(w, \{\lambda_j\}) = F(w) + \sum_{j=1}^J \lambda_j h_j(w)$, where $\lambda_i \geq 0$ are called Lagrangian multipliers
 - Note that:
 - $\max_{\{\lambda_j \geq 0\}} L(w, \{\lambda_j\}) = \begin{cases} F(w) & \text{if } h_j(w) \leq 0, \forall j \in [J] \\ +\infty & \text{else} \end{cases}$
 - Thus:
 - $\min_w \max_{\{\lambda_j \geq 0\}} L(w, \{\lambda_j\}) \Leftrightarrow \min_w F(w) \text{ s.t. } h_j(w) \leq 0, \forall j \in [J]$
- Duality对偶性
 - 我们通过交换 $\min \max$ 来定义他的对偶问题dual problem
 - $\max_{\{\lambda_j \geq 0\}} \min_w L(w, \{\lambda_j\})$
 - 我们假设 w^* 和 $\{\lambda_j^*\}$ 分别是 primal 和 dual 问题的 solution, 我们有:
 - $\max_{\{\lambda_j \geq 0\}} \min_w L(w, \{\lambda_j\}) = \min_w L(w, \{\lambda_j^*\}) \leq L(w^*, \{\lambda_j^*\}) \leq \max_{\{\lambda_j \geq 0\}} L(w^*, \{\lambda_j\}) = \min_w \max_{\{\lambda_j \geq 0\}} L(w, \{\lambda_j\})$
 - This is called Weak Duality
 - Strong duality
 - 当 F, h_1, \dots, h_m 是convex, 在某些条件下
 - $\max_{\{\lambda_j \geq 0\}} \min_w L(w, \{\lambda_j\}) = \min_w \max_{\{\lambda_j \geq 0\}} L(w, \{\lambda_j\})$
- Karush-Kuhn_tucker (KKT) condition
 - 如果 strong duality 成立, 那么我们有:

- $$F(w^*) = \min_w \max_{\{\lambda_j \geq 0\}} L(w, \{\lambda_j\}) = \max_{\{\lambda_j \geq 0\}} \min_w L(w, \{\lambda_j\})$$
- $= \min_w L(w, \{\lambda_j^*\}) \leq L(w^*, \{\lambda_j^*\}) = F(w^*) + \sum_{j=1}^J \lambda_j^* h_j(w^*) \leq F(w^*)$
 - 以上所有不等式都是等式
 - 最后一部分意味着 $\sum_{j=1}^J \lambda_j^* h_j(w^*) = 0, \forall j \in [J]$
 - $\min_w L(w, \{\lambda_j^*\}) = L(w^*, \{\lambda_j^*\})$ 意味着 w^* 是 $L(w, \{\lambda_j^*\})$ 的 minimizer, 那么它的导数就是0, 我们有一些必要条件
 - Stationarity: $\nabla_w L(w^*, \{\lambda_j^*\}) = \nabla F(w^*) + \sum_{j=1}^J \lambda_j^* \nabla h_j(w^*) = 0$
 - Complementary slackness: $\lambda_j^* h_j(w^*) = 0, \forall j \in [J]$
 - Feasibility: $h_j(w^*) \leq 0$ and $\lambda_j^* \geq 0, \forall j \in [J]$
 - 这些必要条件同时会充分当 F 是 convex 并且 h_1, \dots, h_J 是连续可导的 convex 函数

支持向量机 dual formulation

- Recall primal formulation:
 - $\min_{w,b,\{\xi_n\}} \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n$ such that $y_n(w^T \phi(x_n) + b) \geq 1 - \xi_n \quad \forall n, \xi_n \geq 0, \forall n$
 - $-\xi_n \leq 0$
 - Lagrangian is:
 - $L(w, b, \{\xi_n\}, \{\alpha\}, \{\lambda_n\}) = \frac{1}{2} \|w\|_2^2 + C \sum_n \xi_n - \sum_n \lambda_n \xi_n + \sum_n \alpha_n (1 - y_n(w^T \phi(x_n) + b) - \xi_n)$
 - where $\alpha_1, \dots, \alpha_N \geq 0$ and $\lambda_1, \dots, \lambda_N \geq 0$ are lagrangian multipliers
 - Let L equal to this, by stationary condition:
 - There exists primal and dual variables $w, b, \{\xi_n\}, \{\alpha\}, \{\lambda_n\}$ such that $\nabla_{w,b,\{\xi_n\}} L = 0$
 - So we have:
 - $\frac{\partial L}{\partial w} = w - \sum_n y_n \alpha_n \phi(x_n) \Rightarrow w = \sum_n y_n \alpha_n \phi(x_n)$
 - $\frac{\partial L}{\partial b} = -\sum_n \alpha_n y_n = 0$
 - $\frac{\partial L}{\partial \xi_n} = C - \lambda_N - \alpha_n = 0, \forall n$
 - 现在我们用第一个把 L 中的 w 替换掉
 - $L = \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} \alpha_m \alpha_n y_m y_n \phi(x_m)^T \phi(x_n)$
 - 为了找到 dual formulation 的解, 我们去解:
 - $\max_{\{\alpha_n\}, \{\lambda_n\}} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} \alpha_m \alpha_n y_m y_n \phi(x_m)^T \phi(x_n)$ s.t. $\sum_n \alpha_n y_n = 0, C - \lambda_n - \alpha_n = 0, \alpha_n \geq 0, \lambda_n \geq 0, \forall n$
 - 最后三个可以写成 $0 \leq \alpha_n \leq C, \forall n$
 - 这个就是 dual formulation of SVM
- 核化SVM
 - $\max_{\{\alpha_n\}, \{\lambda_n\}} \sum_n \alpha_n - \frac{1}{2} \sum_{m,n} \alpha_m \alpha_n y_m y_n k(x_m, x_n)$ s.t. $\sum_n \alpha_n y_n = 0, 0 \leq \alpha_n \leq C, \forall n$
 - 注意我们这里不用知道具体的 $\phi(x)$ 是什么
- 根据 dual formulation 的解 $\{\alpha_n^*\}$ 重新计算 primal solution w^*, b^*
 - Recall: $w^* = \sum_n \alpha_n^* y_n \phi(x_n) = \sum_{n: \alpha_n > 0} \alpha_n^* y_n \phi(x_n)$
 - a point with $\alpha_n^* > 0$ is called a "support vector", the name SVM
 - 为了计算 b , 我们需要使用 complementary slackness
 - $\lambda_n^* \xi_n^* = 0$ & $\alpha_n^*(1 - y_n(w^{*T} \phi(x_n) + b^*) - \xi_n^*) = 0$
 - 对于某些支持向量 $\phi(x_n)$ 如果我们有 $0 < \alpha_n^* < C$ 那么我们有 $\lambda_n^* = C - \alpha_n^* > 0$
 - 根据第一个条件, 我们知道 $\xi_n^* = 0$ (这些例子可以被正确的分类, 所以不需要 soft margin)
 - 再根据第二个条件, 我们知道 $1 = y_n(w^{*T} \phi(x_n) + b^*)$
 - 所以我们有: $b^* = y_n - w^{*T} \phi(x_n) = y_n - \sum_m y_m \alpha_m^* k(x_m, x_n)$
 - 现在给你一个新的例子 x , 它预测的结果为:
 - $SGN(w^{*T} \phi(x_n) + b^*) = SGN(\sum_m y_m \alpha_m^* k(x_m, x_n) + b^*)$
- 支持向量的几何解释:
 - 一个支持向量满足 $\alpha_n^* \neq 0$ and $1 - y_n(w^{*T} \phi(x_n) + b^*) - \xi_n^* = 0$
 - 当 $\xi_n^* = 0, y_n(w^{*T} \phi(x_n) + b^*) = 1$, 那么这个点距离超平面的距离是 $\frac{1}{\|w^*\|_2}$

- 当 $\xi_n^* < 1$ 这个点被正确的分类，但是不满足 large margin constraint
- 当 $\xi_n^* > 1$ 这个点被错误分类
- 其他被正确分类但不是支持向量的点， $\alpha_n^* = 0$
- 核方法的一个劣势：non-parametric，需要保证所有的训练集合不动
 - for SVM, #support vectors << N, 只有很小很小的一些数据起到分类做支持向量的作用，其他的点在训练集中都没什么用

决策树decision tree

决策树，因为要画树的模型图，手写笔记

- 非线性
- 同时可以分类或者回归，主要看分类
- 可解释性比较强
- 集成学习很好用

Clustering聚类分析

- 非监督学习聚类分析中数据没有label, 所以没有ground-truth value去衡量你的标准
- 输入：data points $x_1, \dots, x_N \in \mathbb{R}^D$ 并且我们想要K个聚类
- 输出：把数据聚类,
 - 找到 assignment $\gamma_{nk} \in \{0, 1\}$ 对于每个点 $n \in [N]$ 和 $k \in [K]$ 保证 $\sum_{k \in [K]} \gamma_{nk} = 1$ for any fixed n
 - 说人话：每个点有且只有分去一个聚类
 - 找到所有聚类的中心 $\mu_1, \dots, \mu_K \in \mathbb{R}^D$
- 我们把它做成一个优化问题 K-means objective: find γ_{nk} and μ_k to minimize
 - $F(\{\gamma_{nk}\}, \{\mu_k\}) = \sum_{n=1}^N \sum_{k=1}^K \gamma_{nk} \|x_n - \mu_k\|_2^2$
 - 每个点到自己属于聚类中心的距离和
 - 但是找到具体的 minimizer 是一个 NP-hard 的问题
- K-means algorithm
 - 初始化
 - 分散找K个中心 μ_1, \dots, μ_K , 每一个点都分去最近的一个中心
 - $\gamma_{nk} = \mathbb{I}[k == argmin_c \|x_n - \mu_c\|_2^2]$
 - 更新center:
 - $\mu_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$
 - 如果没有 converge，则给每个点重新分配中心，重复计算
- 收敛
 - 会收敛到 local minimum, 为什么
 - objective每一次都在降低
 - objective被0 bound住
 - #可能的分配结果是有限的 (K^N)
 - 但是：
 - 可能会有 exponential 的循环来手链
 - 有可能不会收敛到全局最小

高斯混合模型Gaussian mixture models

- GMM 是一种概率模型的聚类分析
 - 比 K-means 函数可解释性更强
 - 可以被看作一种 soft version of K-means
- 和 sigmoid 函数进行比较
 - 我们使用 sigmoid 来“解释”label 的概率，同样的我们用“概率”来解释每个点属于哪一个 cluster，或者哪一个概率分布
 - What probabilistic model generates data like this

- GMM一般有density function: $p(x) = \sum_{k=1}^K \omega_k N(x|\mu_k, \Sigma_k)$
 - K : Gaussian components的数量, (和我们想要多少个聚类 K 一致)
 - $\omega_1, \dots, \omega_K$ mixture weights, 每个分布 K 都有一个自己的weight
 - μ_k and Σ_k 第 k 个高斯分布的mean和covariance matrix
 - N : The density function for a Gaussian
- 潜变量latent variable: $z \in [K]$
 - $p(x) = \sum_{k=1}^K p(x, z=k) = \sum_{k=1}^K p(z=k)p(x|z=k) = \sum_{k=1}^K \omega_k N(x|\mu_k, \Sigma_k)$
 - z here is discrete, if z is continuous, use integral
 - x 和 z 都是随机变量
 - x 是观察到的
 - z 不是观察到的, 是潜在的
- Learning GMM
 - 目标: find $\theta = \{\omega_k, \mu_k, \Sigma_k\}_{k=1}^K$
 - 我们同时还会学习潜变量 z_n $p(z_n = k|x_n) \triangleq \gamma_{nk} \in [0, 1]$
 - i.e.给每个点一个soft assignment到每个cluster, 和K-means中的hard assignment完全相反
 - 这个学习的过程我们使用Expectation-Maximization (EM)算法

EM algorithm

- preview of EM for learning GMMs
 - Step 0: Initialize $\omega_k, \mu_k, \Sigma_k$ for each $k \in [K]$
 - Step 1: (E-step) update the "soft assignment", fixing parameters
 - $\gamma_{nk} = p(z_n = k|x_n) \propto \omega_k N(x_n|\mu_k, \Sigma_k)$
 - Step 2: (M-step) update the model parameter, fixing assignments
 - $\omega_k = \frac{\sum_n \gamma_{nk}}{N}$, 把每个点属于当前gaussian聚类的“概率”加起来再除以点的数量
 - $\mu_k = \frac{\sum_n \gamma_{nk} x_n}{\sum_n \gamma_{nk}}$ 每个点的坐标, 根据它对于类 k 的“概率”来一定程度上的修改中心
 - $\Sigma_k = \frac{1}{\sum_n \gamma_{nk}} \sum_n \gamma_{nk} (x_n - \mu_k)(x_n - \mu_k)^T$
 - 回到Step 1如果不收敛
- 一般的EM算法
 - In general EM is a heuristic to solve MLE with latent variable, i.e. find the maximize of:
 - $\ln p(\theta) = P(\theta) = \sum_{n=1}^N \ln p(x_n; \theta) = \sum_{n=1}^N \ln \int_{z_n} p(x_n, z_n; \theta) dz_n$
 - θ 是概率模型的参数, 如高斯模型: $\theta = \{\omega_k, \mu_k, \Sigma_k\}$
 - x_n 是观察到的随机变量
 - z_n 是潜变量latent variable
- EM for GMM和K-mean的关联
 - K-means事实上是一种特殊形式的EM
 - 假设 $\Sigma_k = \sigma^2 I$ 对于某些 σ , 只有 ω_k 和 μ_k 是参数
 - 当 $\sigma \rightarrow 0$ EM就是K-means

判别模型和生成模型discriminative model & generative model

- 知乎网页解释: <https://www.zhihu.com/search?type=content&q=%E5%88%A4%E5%88%AB%E6%A8%A1%E5%9E%8B%E4%B8%8E%E7%94%9F%E6%88%90%E6%A8%A1%E5%9E%8B>
- 基本上来说, 给定一个训练数据 (X, Y) , 现在新来一个样本 x , 我们想要预测它的类别 y
- 我们最终的目的是求得最大的条件概率 $P(y|x)$ 作为新样本的分类。
- 判别模型
 - 根据训练数据得到分类函数和分界面, 比如说根据SVM模型得到一个分界面, 然后直接计算条件概率 $P(y|x)$, 我们将最大的 $P(y|x)$ 作为新样本的分类。判别式模型是对条件概率建模, 学习不同类别之间的最优边界, 无法反映训练数据本身的特性, 能力有限, 其只能告诉我们分类的类别。
 - 常用算法

- 线性回归, 逻辑回归, 线性分类, 支持向量机, CART (classification and regression tree), 神经网络, 高斯过程, 条件随机场(CRF)
 - 会因为数据的不平衡而导致性能下降
- 生成模型
 - 一般会对每一个类建立一个模型, 有多少个类别, 就建立多少个模型。比如说类别标签有 {猫, 狗, 猪}, 那首先根据猫的特征学习出一个猫的模型, 再根据狗的特征学习出狗的模型, 之后分别计算新样本 xxx 跟三个类别的联合概率 $P(x, y)$, 然后根据贝叶斯公式:

$$P(y|x) = \frac{P(x,y)}{P(x|y)}$$
 分别计算 $P(y|x)$, 选择三类中最大的 $P(y|x)$ 作为样本的分类
 - 常用算法:
 - naive bayes, KNN, GMM, HMM, 贝叶斯网络, Sigmoid belief network, 马尔可夫随机场, LDA文档生成模型

机器学习中关于准确率的测量:

- TP: 实际为positive, 测试结果为positive
- FP: 实际为negative, 测试结果为positive
- TN: 实际为negative, 测试结果为negative
- FN: 实际为positive, 测试结果为negative

记忆方法: 所有True的东西测试结果和实际结果的都相同, 所有False的东西, 后面的positive/negative说明测试的结果是positive/negative, 然后false表示这个测试结果是错误的

1. Accuracy

$$\frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

1. 最常见的方法, 但是在数据不平衡情况下不是很好用 (比如针对病例的测试, 患病率很低)
2. 例子: 针对100个肿瘤进行分类, 恶性 (positive) 良性 (negative)
 - 1个TP, 1个FP, 8个FN, 90个TN
 - $Accuracy = 0.91$
 - 看上去我们做的还不错, 其实针对91个良性肿瘤, 我们的对90个测试结果都是正确的, 但是针对9个恶性肿瘤, 其实我们只发现了其中1个是恶性, 其他的8个我们都没有测试出来
2. Precision: what proportion of positive identifications was actually correct (所有测试结果显示为positive的例子中, 我们有多少做出了正确的判断)

$$\frac{TP}{TP + FP} \quad (2)$$

1. 上面的例子中: $precision = 0.5$
3. Recall: What proportion of actual positives was identified correctly (针对所有实际情况为positive的例子中, 我们有多少做出了正确的判断)

$$\frac{TP}{TP + FN} \quad (3)$$

1. 上面例子中 $recall = \frac{1}{1+8} = 0.11$
2. 只针对11%的恶性肿瘤 (positive) 做出了正确的判断! 这不行

我们需要注意, 随着precision上升, recall会下降 (如果只通过提高或者降低门槛的bar) 这个时候我们需要其他的方法

4. F-measure

$$F = 2 \times \frac{precision * recall}{precision + recall} = \frac{2}{recall^{-1} + precision^{-1}} \quad (4)$$

1. $F = 1.0$: perfect precision and recall
2. $F = 0$: either precision or recall is 0
3. 在NLP中有广泛的应用比如named entity recognition and word segmentation
5. ROC curve (receiver operating characteristic curve)

$$TPR(\text{True positive rate}) = \frac{TP}{TP + FN} = recall \quad (5)$$

$$FPR(\text{false positive rate}) = \frac{FP}{FP + TN} \quad (6)$$

- ROC curve: TPR vs. FPR
 - 通过变化二分类问题中的门槛，比如降低门槛会导致更多的positive case，可以同时增加False positive和True positive
- AUROC (area under the ROC curve) AUC provide an aggregate measure of performance across all possible classification thresholds
 - Range from 0 to 1: 100% wrong -> AUC = 0; 100% correct -> AUC = 1
 - AUC is classification threshold invariant: measure the quality of the model irrespective of what threshold is chosen
 - In spam detection we won't use this: you likely want to prioritize minimizing false positives (results in a increase of false negatives)

机器学习中如何处理过拟合问题

Definition: training set have great performance but not test dataset

- Golden rule: 去找更多的训练数据 (More training data set)
- Data augmentation: 比如图片处理中通过改变图片位置，姿态，尺度照片亮度，平移，反转，切割
- 正则化 (regularization) , 是代数几何中的概念
 - 使用正则化可以防止过拟合的原理
 - [机器学习中使用正则化来防止过拟合是什么原理](#)
 - [为什么L1稀疏, L2平滑?](#)
 - Instead of aiming to minimize loss we now minimize loss+complexity
 - 当我们过拟合的时候，构造出来的函数大概率会有很多剧烈的曲折，局部切线斜率巨大对应函数在某些极小的区间中变化很大，而从变量的因素来看，导致这个结果只有系数过大，所以我们需要用正则项来约束系数的大小，不要让他们过大
 - L2 regularization: encourages weights to be small

$$\|w\|_2^2 = w_1^2 + w_2^2 + \dots + w_n^2 \quad (7)$$

5. L1 vs. L2
 - L2 penalizes $weight^2$
 - L1 penalizes $|weight|$
 - Different derivatives L1 : -1 or 1, L2 : w ;
 - 想象L2每次把weight中的某个常数值去掉，让它指向0，L1每次都是减去一个常数，有可能最后就停在0上了，这也是为什么L1正则化产生稀疏解，因为我们可以抛开那些变成0的weight，只用剩下的部分。而L2永远不会收缩到0，使用比较小的权值，产生的就不是稀疏效果而是平滑
4. Early stop
5. Batch Normalization (批标准化)
 - [什么是批标准化 \(Batch Normalization\)](#)
6. Dropout layer按照一定概率将某些神经元暂时从神经网络中扔掉
 - 削弱neuron之间的联合适应性
 - 对dropout之后的网络进行训练，相当于做了data augmentation，因为我们总可以找到一个样本在原有的网络上也能达到dropout之后的效果
7. 交叉验证 (cross validation)
8. 减少网络层数，神经元数量

深度学习：

神经网络激活函数：

- 使用激活函数的目的：让网络变得非线性，否则无论多深的网络，多层线形网络通过矩阵运算，效果等同于一层神经网络
- Sigmoid

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} = 1 - Sigmoid(x) \quad (8)$$

也被称为logistic function
导数可以使用自己表示

$$S'(x) = S(x)(1-S(x)) \quad (9)$$

2. ReLU (Rectified Linear Unit)

1. 小于0都是0，大于0都是自己
2. 基本线性，函数梯度（导数）不会根据x增大或者减小而消失（不像sigmoid）

3. tanh(x)

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (10)$$

神经网络分类函数Softmax

$$\text{Softmax} = \frac{e^{z_i}}{\sum e^{z_j}} \text{ for all } j \quad (11)$$

1. 把最后train出来的结果改成0到1之间的数字，并且保证所有的数据加起来等于1，所以最终构成了一个概率分布(probability distribution)

梯度消失与梯度爆炸：

1. 消失：Sigmoid函数当x比较大或者比较小的时候，非常接近0或者1，导数值非常小
 1. 如果我们使用标准方法初始化，N(0, 1)的方式来初始化权重
2. 爆炸：初始化成为一个非常大的值，比如用Relu function，当神经网络很深的时候，梯度呈指数型增长，会造成一个非常非常大的权重更新，尤其在RNN中
3. 解决方案：
 1. 好的参数初始化
 2. 非饱和激活函数：Relu
 3. 批规范化
 4. LSTM

深度学习 --- 优化入门二（SGD、动量(Momentum)、AdaGrad、RMSPROP、Adam详解）

动量Momentum

1. 问题开始：病态曲率，每次梯度下降的方向并非是最低值，比如在“山谷”中的结构来回反复跳跃，而不是直接穿过山沟
2. Newton method
 1. 问题，计算量过大
3. Momentum：累计过去我们学习过的步骤来确定，每一步走多远不光取决于当前的斜率，还取决于过去我们走的卒读，如果和过去走的方向一致，走得更快，如果和过去不一样，走的慢一点

其他优化方法：

1. AdaGrad (自适应梯度算法)

1. 就是将每一维各自的历史梯度的平方叠加起来，然后更新的时候除以该历史梯度值即可

$$\theta_i = \theta_i + \nabla(w_i)^2 \quad (12)$$

2. 更新 w_i 的时候：

$$w_i = w_i - \frac{\alpha}{\sqrt{\theta_i} + \delta} \nabla w_i \quad (13)$$

1. $\theta < 1$ 放大学习效率 $\theta > 1$ 减少学习效率
 1. 刚开始大步大步的走，后面小步小步的走
 2. 问题：很有可能过去累积的过多，导致我们后面走的过度小心，随着训练周期的生长，学习率降低的很快
2. RMSProp (均方根支柱)

1. 对过去和现在做一个平衡

$$\theta_i = \beta\theta_i + (1 + \beta)(\nabla w_i)^2 \quad (14)$$

2. 参数更新阶段和daGrad一样
3. Adam 自适应动量优化

1. 使用动量方法，计算包含过去
2. 使用RMSProp算法，计算衰减学习率
3. 然后更新参数深度学习优化：

过去的问题

1. 什么是resnet与扩充知识：
 1. [你必须要知道CNN模型: ResNet](#)
 2. [一文读懂VGG网络](#)
 3. [深度学习卷积神经网络-AlexNet](#)
2. 什么是BatchNorm层
 1. [什么是批标准化 \(Batch Normalization\)](#)
 2. 如果我们决定放入BN层，我们放在哪里，所有层都需要还是
 1. 答案：比如CNN中我们放在Convolution layer与activation function层中间所有层都需要

```
1 h = conv2d(x, h_out, 1, stride=stride, scope="conv_1")
2 h = batch_norm(h, is_training=is_training, scope="bn_1")
3 h = tf.nn.relu(h)
4 h = conv2d(h, h_out, 3, stride=1, scope="conv_2")
5 h = batch_norm(h, is_training=is_training, scope="bn_2")
6 h = tf.nn.relu(h)
7 h = conv2d(h, n_out, 1, stride=1, scope="conv_3")
8 h = batch_norm(h, is_training=is_training, scope="bn_3")
```