# Boosting 算法-ensemble 决策树

The idea of boosting is to train weak learners(models) sequentially, each trying to correct its predecessor.

Boosting 是一种ensemble的手段，是一族boosting算法的统称

Bagging 与 Boosting 都是ensemble 的方法，但是bagging是结合独立的分类器进行，而boosting是结合

注意在boosting 当中，下一个弱分类器是与之前所有分类器的分类结果互补，而不是仅仅与上一个分类器的结果互补。

1.Adaboost：

对于第一个分类器产生的结果，将错误划分的样本权重加大(注意再分配weight的时候需要normalization)，然后将加权的训练数据feed到下一个分类器中，直到most accurate predictor is built.

细节：

当分类正确，数据权重除以d, 否则数据的权重乘上d，那么这个d实际上也是会变化的。

之后我们不断的训练，得到了 f1,f2,f3…等等一系列的方法

之后aggregate的方法： 每一个classifier 都按照他们的准确度来加权

2.Gradient boosting

Gradient boosting tries to fit the new predictor to the residual error made by the previous predictor.

其算法目的在于fit到上一个model的残差：

F1(x) = y: F1根据(x,y) fit

h1(x) = y – F1(x): 计算残差

F2(x) = F1(x) + h1(x)

3.XGBoost(extreme gradient boosting)

由于Gradient Boosting 需要sequential 的训练tree,因此过程十分缓慢，XGBoosting是一种能够充分利用资源进行Gradient boosting的算法。

Ensemble: Bagging 与 Boosting 使用的场合并不一样

1.Bagging: overfitting 的时候才考虑使用 (e.g 决策树)

Random forrest – decision tree 的 bagging 形式

Boosting:

Boosting的用法在于，model不能够很好的fit到我们的data。

（Bagging）的用法在：model容易对data产生overfit的情况

Boosting：

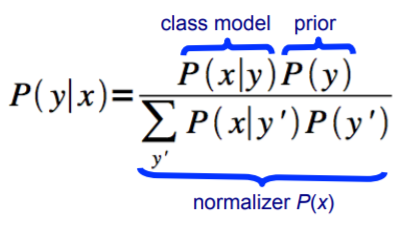
1. 找到classifier f1, 找到另一个classifier f2 用来与f1 互补.，然后再找f3 与 f2互补…

注意找的过程是顺序的，先f1,再f2, 不同于bagging是没有顺序的。

AdaBoost: 见上一页

# IAML 知识点复习

## 1. Naïve Bayes



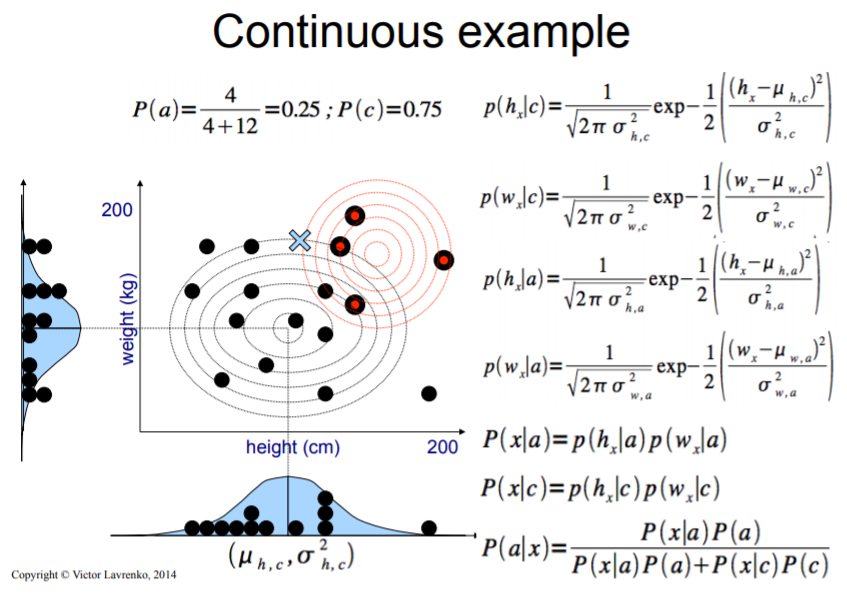
Naïve bayes 是一种生成模型(gegerative model), 是一种概率分类器(probabilistic classifier)

一个重要假设： 属性之间是independence 的

Gaussian naïve Bayes: 当属性是连续的时候, 比如有

P(y) 的计算: 依然按照training set种不同的y 的比例进行计算。

P(x | y)的计算: 对于每一类的 y, 对其每一个feature，计算对应的mean 和 std： 每一个feature都可以用gaussian model 进行拟合，然后将对应feature 依次带入各个gaussian model，将结果相乘就为p(x | y)。 如下图



Naïve bayes的问题：

1. 对于0频次项，需要进行laplace平滑处理： 对于所有的count都加上一个非常小的正数。

2. 是基于假设： word independence 进行的，如果feature之间不是独立的，那么对效果有一定影响

Naïve Bayes处理missing data的方式： 直接跳过，不计数。

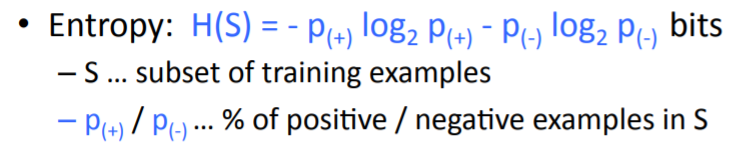
## 2.决策树

唯一可以解释的机器学习算法

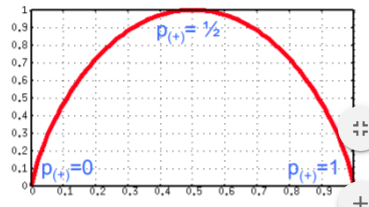
ID3算法： 每一次都选择最好的属性来进行分割，在后序产生的其他结点种选择出其他的属性继续分割，直到node是pure的。

关键：如何选择属性？- 按照属性分割的话则是越纯净pure越好。

熵(entropy):



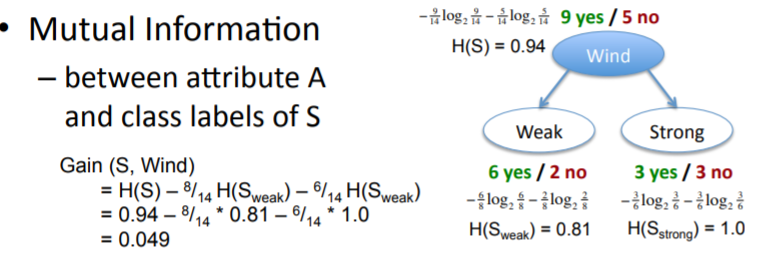
Entropy的范围是[0,1]



交叉熵(cross-entropy): -y1logy2^ - y2logy2^ -````: 范围是0-+∞

选择属性的标准：Information Gain(信息增益)

Gain = 节点的熵[按某属性] – 数量权1\*子节点1的熵[属性] – 数量权2\*子节点2的熵[属性]-….



决策树的一大问题： overfitting，它总可以在training data 上达到100%的准确度。

解决overfitting： 1.根据给定的条件(如深度限制/最大叶子个数)提前停止

2.剪枝： 是一种贪心算法：

（1）首先尝试依次remove每一个叶子节点

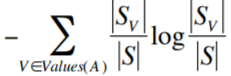
（2）然后看remove 掉哪一个节点对validation set的提升最大，就真实的remove掉这个节点

（3）重复此过程直到在validation set上不再有提升

决策树是一种discriminative 的方法。

使用Information grain 也存在问题：gain 趋向于使得节点越纯净越好，因此会首先考虑一些诸如日期/编号等等属性作为分割，这样子节点是perfect pure的。这样的做法一来overfitting，二来对新的没见过的数据无法处理。

因此使用GainRatio = Gain/SplitEntropy

其中Gain为information gain， splitEntropy = 

对于连续数值的决策树：

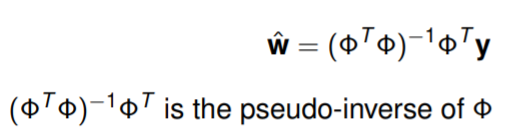
决策树的优点：1. 可解释 2.test 非常快 3.可以处理missing data

缺点：1. Greedy， 所以可能不能够找到全局最优的树 2. Axis-aligned split

随机森林： bagging 的决策树： 随机选取属性于instance，然后不剪枝的长出k棵树，然后投票。

## 3.Linear Regression

在高维空间拟合的是超平面



其中W是design matrix [1,x]

缺点： 对outlier非常敏感。

使用rbf-based: 对原有的属性rbf转换，然后做linear regression。注意每一个feature都对应一种转换。



其他的转换： Gaussian/polynomial/sigmoid

为什么使用这些函数进行转换？原数据点是线性不可分的情况下，对input space的转换可以使其成为线性可分的。

## 4.Logistic Regression

Linear + sigmoid

Discriminative

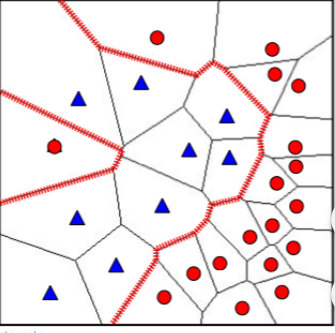
## 5.SVM

梯度下降方法见吴恩达机器学习SVM，其最主要的优势在于不怕高维数据。

## 6.KNN

KNN是discriminative的，并且其决策边界是non-linear的。

K越小则越容易过拟合, 容易受到outlier 的影响。



# Feature Engineering

数据和特征决定了机器学习的上限，而模型和算法只是逼近这个上限而已。

（The success of machine learning depends on how you present the data）

特征工程：目的是最大限度的从原始数据中提取特征以供算法和模型使用

## 特征选择(Feature selection)

如何的检测一个feature 的重要性？

（1）filter 方法

1. 余弦相似度：给定X与Y， 那么 cos<a,b> = a\*b / |a||b|, 其范围是【-1，1】，-1 代表向量方向相反，1代表向量方向相同，0代表正交。

2.pearson 相关系数： 检测y与feature 之间的线性关系。取值【-1，1】，-1代表绝对负相关，1代表绝对正相关。0代表没有任何关系。

Pearson系数实际上是中心化之后的余弦相似度，是为了填补在维度上的空缺。

（2） wrapper 方法：

1.前向搜索： 使用模型svm，knn，Random forrest等，依次添加特征集合进行行训练，看看哪一种效果最好。

2.反向搜索： 先使用全部的feature，然后依次拿掉使用的feature，查看error

（3）Embedded 特征选择

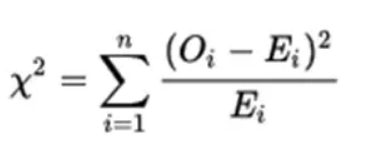
1.基于L1惩罚项的选择。加入L1惩罚项，因为L1具有sparse的特点，其会筛选掉不重要的特征（或已经保留了的相关的重复特征）。

2. 基于模型的选择：每次拿出一个feature， 训练一个模型（如决策树/随机森林），仅仅依据这一个的效果判断。

P-value 与 卡方检验(chi-square)

有假设H0

Chi-square: 用来检验期望值与实际测试值之间的差异，其公式为



p-value 的得到: 通过计算chi-square ，然后查表得出。 我们的假设是根据H0来得出的，chi-square也是根据H0 来得出的，而当p-value<0.05的时候，通常意味着假设H0是错误的。也意味着这个独立变量(feature)比较重要,与输出(y)有关系。

## 特征工程

特征工程包括：1.数据处理 2.特征选择（见上） 3.维度压缩

特征工程方法：

## Categorical features

categorical features: 通常都需要一些处理, 庞大的基数将导致稀疏，并且很难去impute missing data。

那么针对categorical features， 通常采用的是one-hot encoding 的方式， 在使用one-hot 的时候，我们可以少用一列来减少“共线性”。例如gender 特征的取值为male/female,那么我们可以仅对male 进行coding，因为male与female encode了很多相同的信息；如果有k个取值，那么可以仅仅使用k-1进行encoding。

hash encoding: 压缩到比one hot encoding 更小的维度中，(通过使用哈希函数)。 但是这个过程中可能会产生不同的feature 映射到相同的hash表示，称作collision

label encoding: 给每一个能取到的值一个数字ID，这样做不会增加任何维度。对于tree-based algorithm非常有用，但是由于附加了顺序因素，通常重复几次随机映射-训练循环来解决这个问题：

1.randomly cat\_var->num\_var;

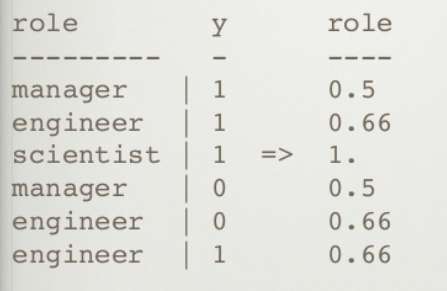
2. retrain

3. average 训练结果, 来缓解误差

Count encoding: 对于每一个variable，使用他们在training data中的count作为表示。对于没见过的variable 可以用1表示。但是也可能造成collision因为有相同的count。此外，对outliers 也非常敏感【???】。

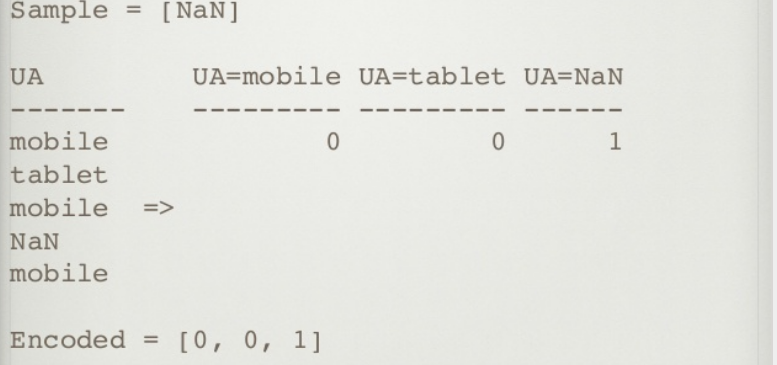
LabelCount encoding：结合label encoding 和 count encoding， 先计算各个var的count，然后按照count的值排label。

Target encoding：结合用例的ground truth来进行encoding，用于二分类问题或者回归问题。Encoding 方式如下：

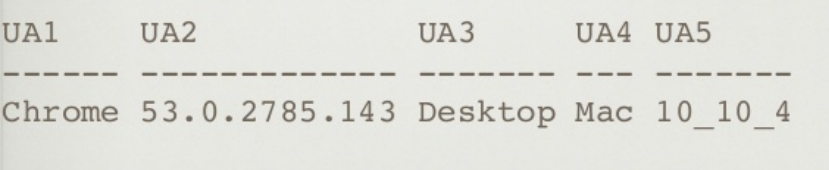


Category embedding： 通常比one hot有更好的效果，使用NN来创建dense的表示方式。

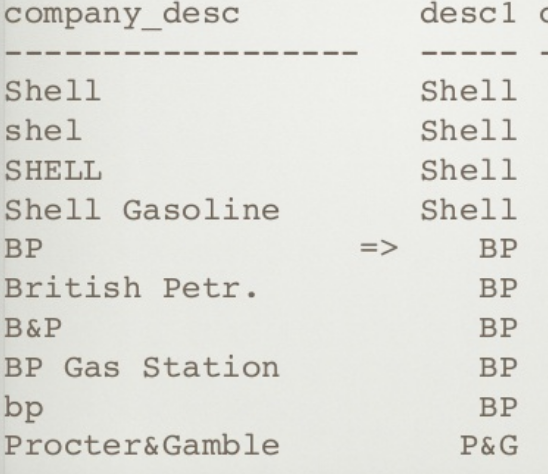
NaN encoding：由于NaN可能也会包含信息，那么可以把NaN也进行encoding



Expansion encoding：由一列属性创建出更多相关的属性



Consolidation encoding: 将不同的取值映射到相同的值，通常用来纠正spelling errors， 轻微的描述不一致等等。

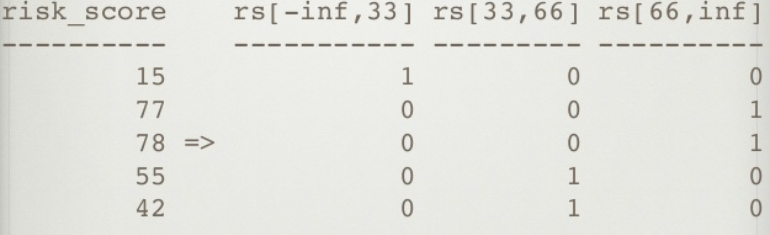


## Numeric features

相对categorical的feature，数值的特征更容易fed到model中，并且更容易处理missing data

Rounding: 降低浮点数的精度

Binning：将连续的rea value 放入不同的bin中，进行离散化处理。此方法对超出的变量也有很高的鲁棒性。



Scaling: 消除不同的feature 之间的range 不同所造成的影响。例如计算欧式距离的时候会造成影响(譬如kmeans算法以及knn)，另一方面使用scaling可以让梯度下降更快速。

Scaling 的方式：

1. Standard Scaling (Z-scaling): x-mean/std [标准化，standardization]

2.MinMax scaling: x-min/x-max [归一化，min-max normalization]，在nomalize 之前一定要进行outlier 的去除，否则仅仅是缩放。

3.Root scaling / log scaling

处理missing data：

1. 填充：Mean, Median(比Mean来讲，消除outlier影响的能力更强)，使用model填充

2.忽略

处理时间数据： projecting to a circle，首尾相连： 最大值与最小值之间的distance是min 与 min 之间的distance +1

降维:

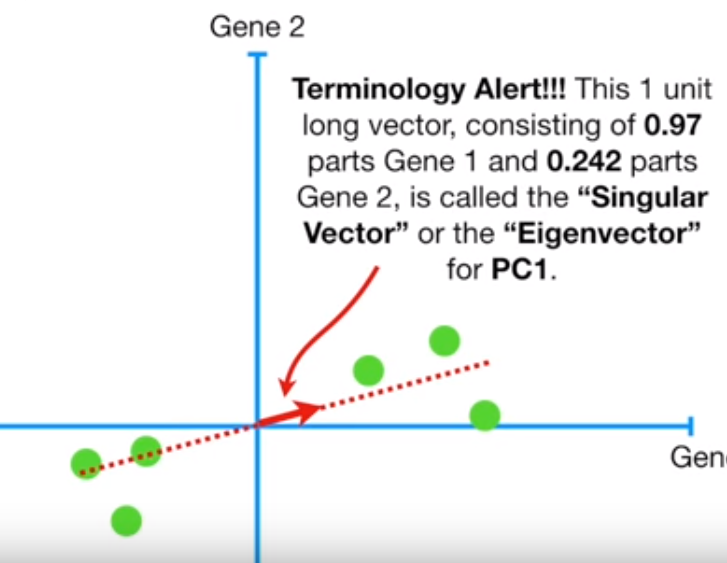
PCA / SVD / LDA / LSA

PCA 与 SVD

PCA:

1. 0中心化，所有的feature 都减去其均值

2. 寻找主成分，PC1是数据在上分布的最离散的方向，而PC1实际上是各个feature的线性组合



那么我们在主成分的方向取长度为1的向量（称为Eigenvector），而这个方向是由各个feature 的线性组合决定的（根据loading score加权【决定斜率】），特征值（eigenvalue）：就是各个在PC1方向上的投影到原点的距离的平方和。 Eigenvalue开根号即是singular value。

计算variation： Eigenvalue【pc1】/（N-1） = variation for PC1

Eigenvalue【pc2】/（N-1） = variation for PC2

# -------------------吴恩达机器学习------------------

# Machine Learning

Supervised Learning： teach – 给出label

(SVM: 可以解决infinite features 的输入)

Supervised learning 又可以分为分类与回归两大类

-Classification: 分类问题

-Regression: 回归，预测real value实数

Unsupervised learning： self-learn

给出data， 非监督学习目的在于找出data的结构(cluster) [举例:cocktail party]

机器学习的本质:

通过训练集，学习到一个函数(hypothesis) h.

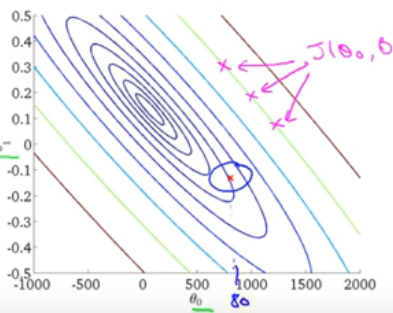
# Linear Regression:

Univariate linear regression(单一变量的线性回归): H(x) = θ0 + θ1\*x

Cost function:

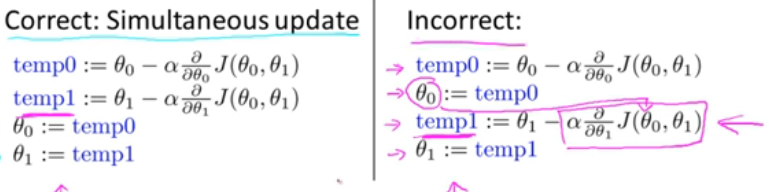
使用欧式距离(squared error cost function)J = 1/2m \* (h(x) - y), 对大多regression 都使用的多。

Console plot： 同一个椭圆上的cost J是相同，越靠近中心则J越小，横纵坐标为参数。每一个点都代表一种参数的组合，即一种hypothesis 函数 h。



优化linear regression 的一种方法：

Gradient descent： 注意梯度下降中，在任何一步当中，所有参数的更新都是同步的。



当learning rate 太小的时候： gradient descent 将会非常缓慢。

当learning rate太大的时候：无法收敛到最小值，将会在周围震荡。

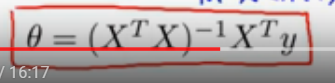
当参数使得J到达局部最小值的时候，将不变。

注意： 最基本的gradient descent算法在running 的过程中，每一步的步长都会自动的逐渐减少（因为随着slope的平滑，偏导项会逐渐减小）。

解Linear regression的另一种方式： 数值计算

首先构造矩阵 [1,x], 其中1 为列向量(size = m\*1)，x为特征矩阵(size=m\*n)[m 个instance, n个feature]， 设ground truth 是 size = m\*1 的列向量 y。

那么最优参数为：



此解法不需要feature scaling，但是如果使用梯度下降解法，那么feature scaling是十分重要的。

Gradient descent解法：需要选择learning rate，并且要许多迭代，可能会慢。但是如果由上百万的feature，使用梯度下降会更好。

Normal equation：不需要learning rate，非常快速。 当feature数量很多的时候，由于主要的计算开销在X矩阵的转置，大概为O(n^3),因此当feature 很多的时候考虑使用gradient descent方法。

对于公式中(x^Tx)不可逆的情况（发生的情况非常小）

造成不可逆的原因: 1. Redundant features： 即当矩阵中的两行有线性关系的时候。

2. 特征数量过多: m<<n

Linear regression 优缺点：

优点： 简单

缺点: 1.只能拟合线性关系

2. 对outlier 非常敏感

3. 假设数据的feature是独立的互不影响的，而实际上很多数据variable之间并不独立。

## Logistic Regression

(虽然叫做regression，但其实是分类算法)

Logistic regression = linear regression + sigmoid 1/（1+e^-z）

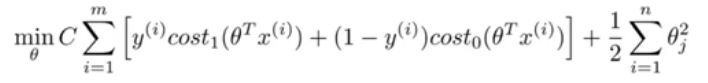
P(y=0|x;θ) + P(y=1|x;θ) = 1

## SVM

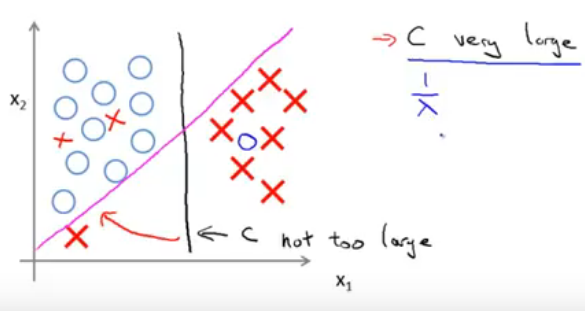
Logistic regression的损失函数:

Cost = min: 1/m \* [-(ylogy^ + (1-y)\*log(1-y^))] + lamda/2m \* |w|(norm2)

那么SVM实际上是对cost 函数修改: C\*loss + regularization



C 代表的是我们考虑loss 的程度，C越大，则越考虑越多的loss成分。C的另一个作用：当C非常大的时候，决策边界会考虑更多的loss，因此如果存在outlier，C也要求边界严格分类(容易产生overfitting)。当C不是那么大的时候，可能依然能够得到正确的结果。



该损失函数 在优化的时候，就是在解决max margin的问题

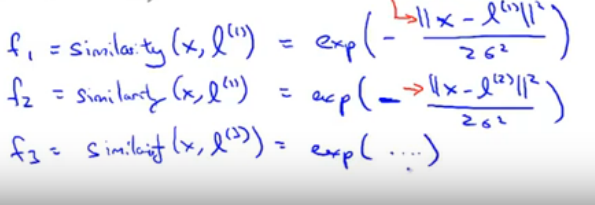
SVM的损失函数是hinge loss：

当y = 1, Wx > 1的时候，loss = 0； 当y=0, Wx<-1的时候， loss = 0.否则都有不同程度的loss。

Kernels：rbf做法：

1.在特征空间中设置中心点C（landmark）

2.定义C个rbf函数：



3.每一个函数f就是一个kernel k(x,c)，而实际上衡量的是x与c的距离。【x离c比较远，则根据公式接近0；如果x距c比较近，那么其接近1】，每一个kernel 就给出一个新的feature。

那么这些C如何选择？

对于每一个出现在training set中的样例，我们都用它作为一个中心点C，生成一个kernel 函数。 然后对于每一个training instance，用所有的kernel函数生成它自己新的feature。

Linear Kernel: 不使用kernel, 直接使用, 原始特征

Polynomial: 对原始特征进行多项式处理, 一般不用

Gaussian kernel / rbf

SVM的多分类问题：

K 个class， 则设定k个不同的SVM，判别每一类是/不是属于这个类。

Logistic regression vs SVM

当维度n 非常大的时候，通常使用linear kernel即可。

当维度n中等的时候，通常使用rbf kernel

当维度非常小，但数据很多的时候：create more features， 然后使用SVM without kernel。

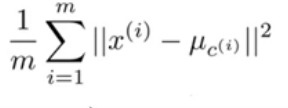
SVM 解决的优化问题： 永远convex

# Clustering Algorithm

## K-means

k-means for non-separated cluster: 当数据点的cluster 特征并不明显的时候，也可以直接进行kmeans分类

k-means 的优化目标： intra-class distance

 (Distortion)

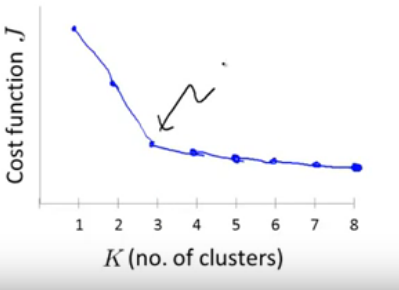
那么如何randomly initialize 中心点？

---Randomly pick K training examples

Kmeans 可以掉入local optima， 收到initialization 的影响 --- 解决: try 不同的初始化方法, 选择能够最优化intra-class distance 的一种方式

How to choose K?

1. Elbow method

 选择这个k, 因为cost 的下降从elbow处减缓了

但是如果没有elbow的情况：

另一种 选择k的方式：

调整k, 看看哪个k能够让目标的performance 更好。

# Dimensionality Reduction – 数据降维

原因: Data compression – save memory/ training faster/ less noises

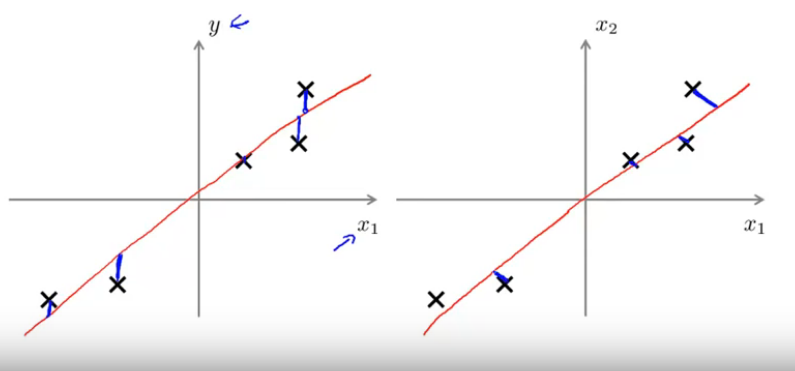
数据降维的另一个使用: 进行数据可视化(将原有的数据维度映射到2维空间z1,z2，并进行数据可视化)

## PCA (Principal Component Analysis)

PCA 在做的事情: 找到一个超平面，使得所有数据点到该平面（k个vector决定的平面）的距离的平方之和（projection error）是最小的。

PCA 与 linear regression的关系？

Linear regression的cost 是residue 残差， 而PCA中造成cost 的是到平面的的距离(projection error). 如下图，左图代表linear regression, 右图为PCA. 此外Linear regression 是predict 一个 real value的，而PCA是用来进行数据降维的。



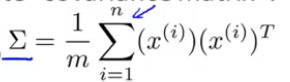
PCA 算法 Implementation:

1. 进行数据pre-processing: 进行中心化(选择性进行数据scaling)

计算每一个feature 的 mean， 然后x = x- μ

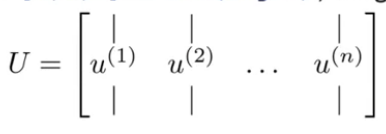
2. 原始数据有n dimension, 希望降维至k, 则首先计算covariance matrix

如何计算covariance matrix？ 协方差矩阵sigma的计算为： 其中sigma 矩阵为n\*n 的矩阵



然后通过奇异值分解计算 eigenvector of sigma:

[U,S,V] = svd(Sigma)

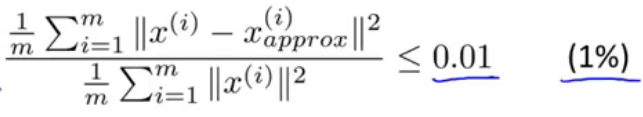


U: n\*n matrix, 而U matrix 的每一个column 即为一个Principal Component, 那么如果希望降维到k，则只需要选择U matrix的前K个column即可。

映射原始数据： 使用截断的U matrix (n\*k), 并将其转置为k\*n的matrix， 之后该矩阵与X乘即为降维结果。

Total variation: 原数据点到原点的距离

如何选择K？- number of principal component



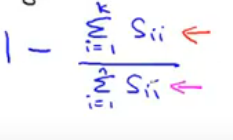
其中分子为projection error， 分母为total variation，<0.01 代表 99% 的variation 被保留。

在实践中， 先尝试k=1，然后持续增加到k保留到足够的variation为止。

而实际上，在调用[U,S,V] = svd(Sigma) 的时候，其中S矩阵是一个对角矩阵(除对角线全0)

S11 代表保留1 个Principal component的variation，S11+ S22代表保留前2个PC的variation，而S11+S22+S33+…+Snn代表了所有的Variation.

故上式子用S对角矩阵来表示为:



如何使用降维之后的数据还原原有的数据？

公式: X = U\_reduce \* Z , 其中U\_reduce 是(n\*k)的降维矩阵，Z是(k\*1)的特征向量

使用PCA的一些建议:

假设原数据有大量的feature，例如一副image 100x100的size， 那么算法将会极其缓慢，因为数据太多。那么需要使用PCA进行降维之后再进行训练。

注意： 我们根据training data的covariance matrix来进行svd解出U\_reduce, 然后使用相同的U\_reduce 在cross\_val 以及testing上，而不重新run PCA在test/cross\_val set上。

注意: PCA 进行降维，feature 数量减少，在一定程度上减轻了overfitting的问题。但是这是一种错误的使用方法，使用regularization来代替。原因：PCA 仅看training x，不看y，这可能会导致PCA 扔掉一些有用的信息，而regularization中看到了损失函数，考虑到了y。

在使用PCA之前，先在raw data上run一趟，如果效果不够好的话再考虑使用PCA. 不要盲目使用PCA.