1.L1 、L2损失函数

L2损失：平方损失函数（回归损失函数）： $L(Y,f(X))=(Y-f(X))^2$， 计算预测值与真实值之间距离的平方和, 如果误差＞1，误差就会被放大很多

L1损失：绝对损失函数： $L(Y,f(X))=|Y-f(X)|$ 鲁棒性更强

常用损失函数

1.平方损失函数

2.绝对值损失函数

3.0-1损失函数

4.对数损失函数（逻辑回归）

$L(y,p(y|x))=-\log p(y|x)$

在当前模型的基础上，对于样本X，其预测值为Y，也就是预测正确的概率。由于概率之间的同时满足需要使用乘法，为了将其转化为加法，我们将其取对数。最后由于是损失函数，所以预测正确的概率越高，其损失值应该是越小，因此再加个负号取个反。

2.L1、L2正则

（机器学习中，如果参数过多，模型过于复杂，容易造成过拟合（overfit）。(即模型在训练样本数据上表现的很好，但在实际测试样本上表现的较差，不具备良好的泛化能力))

概括：

L1和L2是正则化项，又叫做惩罚项，是为了限制模型的参数，防止模型过拟合而加在损失函数后面的一项。

区别：

L1是模型各个参数的绝对值之和。

L2是模型各个参数的平方和的开方值。

L1会趋向于产生少量的特征，而其他的特征都是0. 因为最优的参数值很大概率出现在坐标轴上，这样就会导致某一维的权重为0 ，产生稀疏权重矩阵

L2会选择更多的特征，这些特征都会接近于0。 最优的参数值很小概率出现在坐标轴上，因此每一维的参数都不会是0。当最小化||w||时，就会使每一项趋近于0

L1范式和L2范式为什么可以防止过拟合：

L1范数符合拉普拉斯分布，是不完全可微的。表现在图像上会有很多角出现。这些角和目标函数的接触机会远大于其他部分。就会造成最优值出现在坐标轴上，因此就会导致某一维的权重为0 ，产生稀疏权重矩阵，进而防止过拟合。

L2范数符合高斯分布，是完全可微的。和L1相比，图像上的棱角被圆滑了很多。一般最优值不会在坐标轴上出现。L2正则化使值最小时对应的参数变小。

L1、L2具体过程：

首先针对L1范数$\phi(w) = |w|$，当采用梯度下降方式来优化目标函数时，对目标函数进行求导，正则化项导致的梯度变化当$w_j > 0$时取1，当$w_j < 0$时取-1.

从而导致的参数w_j减去了学习率与(13)式的乘积，因此当w_j大于0的时候，w_j会减去一个正数，导致w_j减小，而当w_j小于0的时候，w_j会减去一个负数，导致w_j又变大，因此这个正则项会导致参数w_j取值趋近于0，也就是为什么L1正则能够使权重稀疏，这样参数值就受到控制会趋近于0。L1正则还被称为 Lasso regularization。

然后针对L2范数$\phi(w) = \sum_{j=1}^{n} w_j^2$，同样对它求导，得到梯度变化为$\frac{\partial \Phi(w)}{\partial w_j} = 2w_j$(一般会用$\frac{\lambda}{2}$来把这个系数2给消掉)。同样的更新之后使得$w_j$的值不会变得特别大。在机器学习中也将L2正则称为weight decay，在回归问题中，关于L2正则的回归还被称为Ridge Regression岭回归。weight decay还有一个好处，它使得目标函数变为凸函数，梯度下降法和L-BFGS都能收敛到全局最优解。

需要注意的是，L1正则化会导致参数值变为0，但是L2却只会使得参数值减小，这是因为L1的导数是固定的，参数值每次的改变量是固定的，而L2会由于自己变小改变量也变小。而(12)式中的

λ

也有着很重要的作用，它在权衡拟合能力和泛化能力对整个模型的影响，越大，对参数值惩罚越大，泛化能力越好。

3.逻辑回归

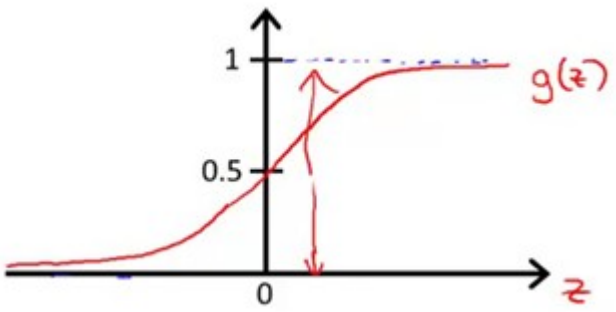
逻辑回归原理

1.找一个合适的预测函数（Andrew Ng的公开课中称为hypothesis），一般表示为**h**函数，该函数就是我们需要找的分类函数，它用来预测输入数据的判断结果。这个过程时非常关键的，需要对数据有一定的了解或分析，知道或者猜测预测函数的“大概”形式，比如是线性函数还是非线性函数。

所以利用了Logistic函数（或称为Sigmoid函数），函数形式为：

$$g(z) = \frac{1}{1+e^{-z}} \qquad (1)$$

对应的函数图像是一个取值在0和1之间的S型曲线（图1）。



2.构造一个Cost函数（损失函数），该函数表示预测的输出（**h**）与训练数据类别（**y**）之间的偏差，可以是二者之间的差（**h-y**）或者是其他的形式。综合考虑所有训练数据的“损失”，将Cost求和或者求平均，记为**J(θ)**函数，表示所有训练数据预测值与实际类别的偏差。逻辑回归的损失函数是 对数损失函数

函数来衡量**h**函数预测的好坏是合理的。

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases} \quad (5)$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$
$$= -\frac{1}{m} [\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))] \quad (6)$$

3.显然，**J(θ)**函数的值越小表示预测函数越准确（即**h**函数越准确），所以这一步需要做的是找到J(θ)函数的最小值。找函数的最小值有不同的方法，Logistic Regression实现时有的是梯度下降法（Gradient Descent）。

逻辑回归和SVM区别

相同点:

1. 都是分类算法
2. 都是监督学习算法
3. 都是判别模型
4. 都能通过核函数方法针对非线性情况分类
5. 目标都是找一个分类超平面
6. 都能减少离群点的影响

不同点:

1. 损失函数不同，逻辑回归是对数损失函数，svm是hinge loss，自带L2正则项
2. 逻辑回归在优化参数时所有样本点都参与了贡献，svm则只取离分离超平面最近的支持向量样本。这也是为什么逻辑回归不用核函数，它需要计算的样本太多。并且由于逻辑回归受所有样本的影响，当样本不均衡时需要平衡一下每一类的样本个数。
3. 逻辑回归对概率建模，svm对分类超平面建模
4. 逻辑回归是处理经验风险最小化，svm是结构风险最小化。这点体现在svm自带L2正则化项，逻辑回归并没有
5. 逻辑回归通过非线性变换减弱分离平面较远的点的影响，svm则只取支持向量从而消去较远点的影响
6. 逻辑回归是统计方法，svm是几何方法

逻辑回归与最大熵

最大熵在解决二分类问题时就是逻辑回归，在解决多分类问题时就是多项逻辑回归。此外，最大熵与逻辑回归都称为对数线性模型(log linear model)。

逻辑回归与线性回归

sigmoid在逻辑回归中起到了两个作用，一是将线性函数的结果映射到了(0,1)，一是减少了离群点的影响。

它们要解决的问题不一样，前者解决的是regression问题，后者解决的是classification问题，前者的输出是连续值，后者的输出是离散值，而且前者的损失函数是输出y的正态分布，后者损失函数是输出的伯努利分布。

4.GDBT和RF区别 GDBT和Xgboost区别

bagging----RF 旨在降低方差

stacking

boosting---GDBT Adaboost xgboost 旨在降低偏差

随机森林RF和GBDT的区别

1）随机森林采用的bagging思想，而GBDT采用的boostingboosting思想。 2）组成随机森林的树可以是分类树，也可以是回归树；而GBDT只能由回归树组成。 3）组成随机森林的树可以并行生成；而GBDT只能是串行生成。 4）对于最终的输出结果而言，随机森林采用多数投票等；而GBDT则是将所有结果累加起来，或者加权累加起来。 5）随机森林对异常值不敏感；GBDT对异常值非常敏感。 6）随机森林对训练集一视同仁；GBDT是基于权值的弱分类器的集成。

7）随机森林是通过减少模型方差提高性能；GBDTGBDT是通过减少模型偏差提高性能。

Xgboost和GBDT的区别

Xgboost优点(与GBDT对比)

1.xgboost对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。

3.xgboost在代价函数里增加了正则项，用于控制模型复杂度。正则项里包含了叶子的结点个数，每个叶子结点输出的score的L2的模的平方和。正则项降低了模型的方差，使模型更简单，防止过拟合

4.**支持并行化**，**直接的效果是训练速度快**，boosting技术中下一棵树依赖上述树的训练和预测，所以树与树之间应该是只能串行！但是，同层级节点可并行。具体地，对于某个节点，节点内选择最佳分类点，进行枚举的时候可以并行

hard voting classifier

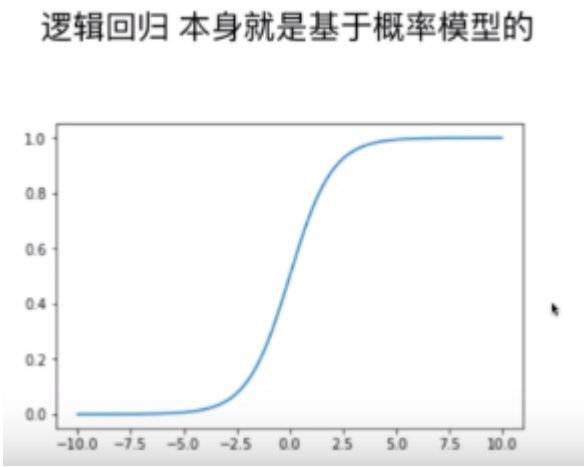
```
from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(estimators=[
    ('log_clf',LogisticRegression()),
    ('svm_clf',SVC()),
    ('dt_clf',DecisionTreeClassifier())
],voting='hard')

voting_clf.fit(x_train,y_train)
voting_clf.score(x_test,y_test)
```

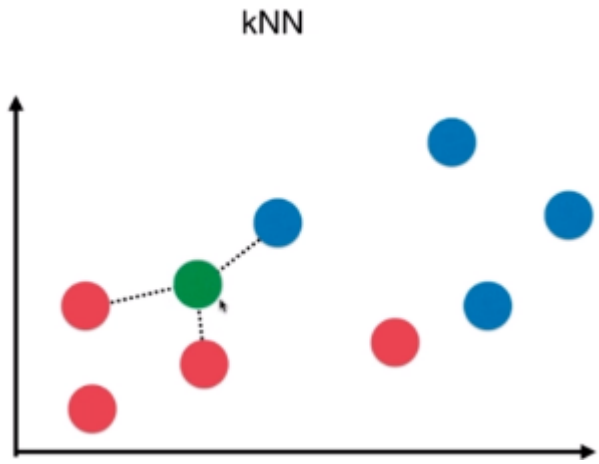
soft voting classifer

不仅要看投票数，还要看权值（需要每一个模型都能估计概率 predict_proba）

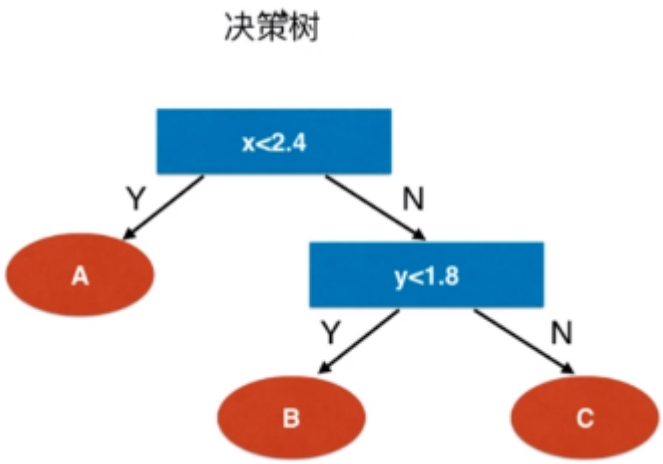
逻辑回归本身就是基于概率模型的----可以预测概率



KNN---(例如本题：分给红色的点 2/3)---可以预测概率



决策树---可以预测概率（计算概率同knn）



svc---（把probability:boolean,optional(default=false)）,默认是false,把false改成true就可以计算每一个数据样本分给某个类相应的概率

```
#soft voting classifer
from sklearn.ensemble import VotingClassifier
voting_clf = VotingClassifier(estimators=[
    ('log_clf',LogisticRegression()),
    ('svm_clf',SVC(probability=True)),
    ('dt_clf',DecisionTreeClassifier())
],voting='soft')

voting_clf2.fit(x_train,y_train)
voting_clf2.score(x_test,y_test)
```

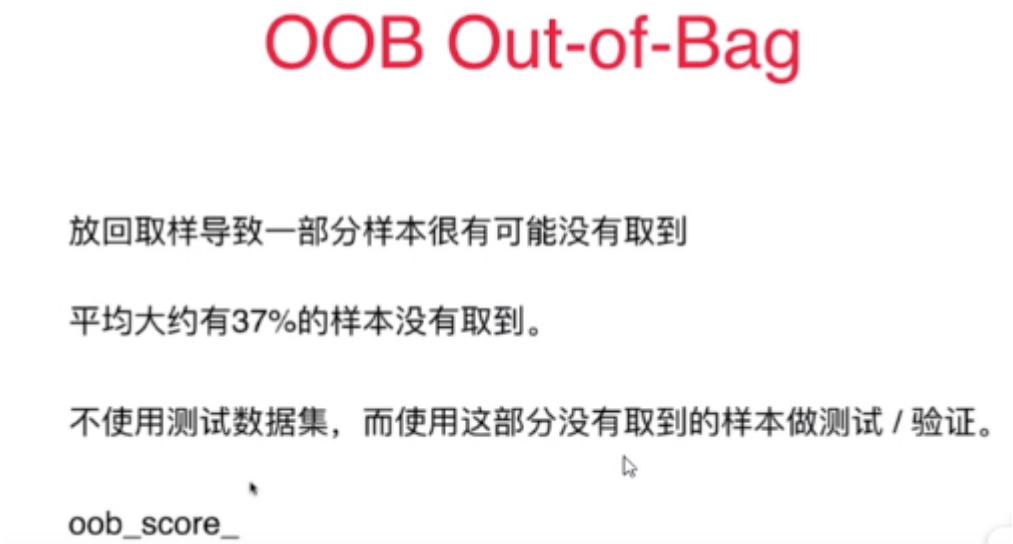
如何创建差异性----每个子模型只看样本数据的一部分

取样-放回取样、不放回取样

放回取样-bagging--更常用（又名bootstrap）

```
#使用bagging
from sklearn.tree import DecisionTreeClassifier()
from sklearn.ensemble import BaggingClassifier()
/*
**DecisionTreeClassifier()决策树模型
**n_estimators 集成500个这样的模型
**max_samples每一个子模型相应的看几个样本数据
**bootstrap=True 放回取样
*/
bagging_clf =
BaggingClassifier(DecisionTreeClassifier(),n_estimators=500,max_samples=100,bootstrap=True)
bagging_clf.fit(x_train,y_train)
bagging_clf.score(x_test,y_test)
```

OOB



```
#使用OOb
from sklearn.tree import DecisionTreeClassifier()
from sklearn.ensemble import BaggingClassifier()
/*
**DecisionTreeClassifier()决策树模型
**n_estimators 集成500个这样的模型
**max_samples每一个子模型相应的看几个样本数据
**bootstrap=True 放回取样
**oob_score=True 不使用测试数据集，而使用这部分没有取到的样本做测试/验证
*/
bagging_clf =
BaggingClassifier(DecisionTreeClassifier(),n_estimators=500,max_samples=100,bootstrap=True,oo
b_score=True)
bagging_clf.fit(x,y)//所有数据传入进去
bagging_clf.oob_score_
```

Bagging的更多探讨

Bagging的思路极易并行化处理

n_jobs

```
#使用n_jobs
from sklearn.tree import DecisionTreeClassifier()
from sklearn.ensemble import BaggingClassifier()
/*
**DecisionTreeClassifier()决策树模型
**n_estimators 集成500个这样的模型
**max_samples每一个子模型相应的看几个样本数据
**bootstrap=True 放回取样
**oob_score=True 不使用测试数据集，而使用这部分没有取到的样本做测试/验证
**n_jobs=-1 并行处理
*/
bagging_clf =
BaggingClassifier(DecisionTreeClassifier(),n_estimators=500,max_samples=100,bootstrap=True,oo
b_score=True,n_jobs=-1)
bagging_clf.fit(x,y)//所有数据传入进去
bagging_clf.oob_score_
```

Bagging的更多探讨

针对特征进行随机采样

Random Subspaces

既针对样本，又针对特征进行随机采样

Random Patches

```
#bootstrap_featrues
from sklearn.tree import DecisionTreeClassifier()
from sklearn.ensemble import BaggingClassifier()
/*
**DecisionTreeClassifier()决策树模型
**n_estimators 集成500个这样的模型
**max_samples每一个子模型相应的看几个样本数据
**bootstrap=True 放回取样
**oob_score=True 不使用测试数据集，而使用这部分没有取到的样本做测试/验证
**n_jobs=-1 并行处理
**max_features==5每一个子模型相应地取几个特征
**bootstrap_features==True 有放回地取特征
*/
Random Patches =
BaggingClassifier(DecisionTreeClassifier(),n_estimators=500,max_samples=100,bootstrap=True,oo
b_score=True,n_jobs=-1,max_features==5,bootstrap_features=True)
Random Patches.fit(x,y)//所有数据传入进去
Random Patches.oob_score_
```

随机森林RF

随机森林

Bagging

Base Estimator: Decision Tree

决策树在节点划分上，在随机的特征子集上寻找最优划分特征

```
from sklearn.ensemble import RandomForestClassifier
/*
**random_state随机的种子
**max_leaf_nodes通过限制最大叶子节点数,可以防止过拟合,默认是"None"
*/
rf_clf =
RandomForestClassifier(n_estimators=500,max_leaf_nodes=16,random_state=666,oob_score=True,n_j
obs=-1)
rf_clf.fit(x,y)
rf_clf.oob_score_
```

基本思想就是构造多棵相互独立的CART决策树，形成一个森林，利用这些决策树共同决策输出类别。随机森林算法秉承了bagging方法的思想，以构建单一决策树为基础，同时也是单一决策树算法的延伸和改进。

在整个随机森林算法的过程中，有**两个随机过程**：

1.输入数据是随机的：从全体训练数据中选取一部分来构建一棵决策树，并且是有放回的选取 2.每棵决策树的构建所需的特征是从全体特征中随机选取的

随机森林算法具体流程： 1.从原始训练数据中随机选取n个数据作为训练数据输入（通常n远小于全体训练数据N，这样就存在一部分“袋外数据”始终不被取到，它们可以直接用于测试误差（而无需单独的测试集或验证集）。 2.选取了要输入的训练数据后，开始构建决策树，具体方法是每一个结点从全体特征集M中选取m个特征进行构建（通常m远小于M）。 3.在构造每棵决策树时，选取基尼指数最小的特征来分裂节点构建决策树。决策树的其他结点都采取相同的分裂规则进行构建，直到该节点的所有训练样例都属于同一类或者达到树的最大深度。（该步骤与构建单一决策树相同） 4.重复第2步和第3步多次，每一次输入数据对应一颗决策树，这样就得到了一个随机森林，用于对预测数据进行决策。

随机森林算法的注意点： 1.在构建决策树的过程中不需要剪枝 2.整个森林中树的数量和每棵树的特征需要人为设定 3.**构建决策树的时候分裂节点的选择依据最小基尼系数** （基尼系数表示被分错的概率） 4.在训练时，对于预选变量个数和随机森林中树的个数这两个参数的调优很重要。

随机森林的优点： 1.两个随机性的引入，使得随机森林抗噪声能力强，**方差小泛化能力强，不易陷入过拟合**（因此不需要剪枝） 2.易于高度**并行训练**，且训练速度快，适合处理大数据
3.由于随机森林对误差率是无偏估计，因此在算法中**不需要再进行交叉验证或者设置单独的测试集来获取测试集上误差的无偏估计**

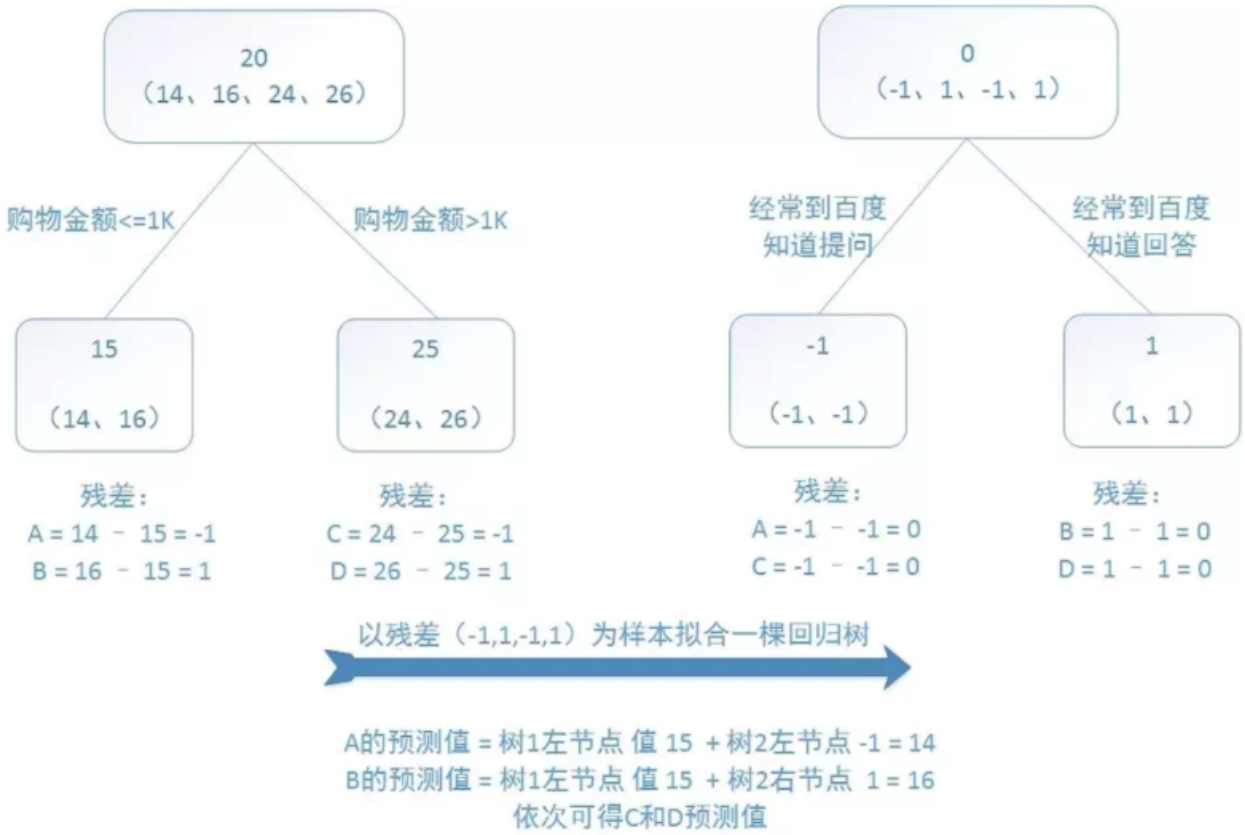
```
from sklearn.ensemble import AdaboostClassifier
from sklearn.tree import DecisionTreeClassifier
//max_depth限制最大深度，防止过拟合。最大深度的增加虽然可以增加对训练集拟合能力的增强,但这也就可能意味着其泛化能力的下降
ada_clf = AdaboostClassifier(DecisionTreeClassifier(max_depth=2),n_estimators=500)
ada_clf.fit(x_train,y_train)
ada_clf.score(x_test,y_test)
```

GBDT

GBDT是第一棵树分类后的残差，作为第二棵树的输入;依次类推直到残差为0或者到达树深

以决策树（CART）为基学习器的GB算法

训练集是4个人，A，B，C，D年龄分别是14，16，24，26。样本中有购物金额、上网时长、经常到百度知道提问等特征。提升树的过程如下：



Gradient Boosting

训练一个模型m1，产生错误e1

针对e1训练第二个模型m2，产生错误e2

针对e2训练第三个模型m3，产生错误e3...

最终预测结果是：m1 + m2 + m3 + ...

```
#Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
gb_clf = GradientBoostingClassifier(max_depth=2,n_estimators=30)
gb_clf.fit(x_train,y_train)
gb_clf.score(x_test,y_test)
```

Boosting解决回归问题

```
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
```

GGBDT是GB和DT的结合。要注意的是这里的决策树是回归树，GBDT中的决策树是个弱模型，深度较小一般不会超过5，叶子节点的数量也不会超过10，对于生成的每棵决策树乘上比较小的缩减系数（学习率<0.1），有些GBDT的实现加入了随机抽样（subsample 0.5<=f <=0.8）提高模型的泛化能力。通过交叉验证的方法选择最优的参数。因此GBDT实际的核心问题变成怎么基于$\{(x_i, r_{im})\}_{i=1}^{n}$使用CART回归树生成$h_m(x)$

Adaboost

每一个训练样本都被赋予一个权重，表明它被某个分类器选入训练集的概率。如果某个样本点已经被准确地分类，那么在构造下一个训练集中，它被选中的概率就被降低；相反，如果某个样本点没有被准确地分类，那么它的权重就得到提高。通过这样的方式，AdaBoost方法能“聚焦于”那些较难分（更富信息）的样本上。

Xgboost

Xgboost优点(与GBDT对比)

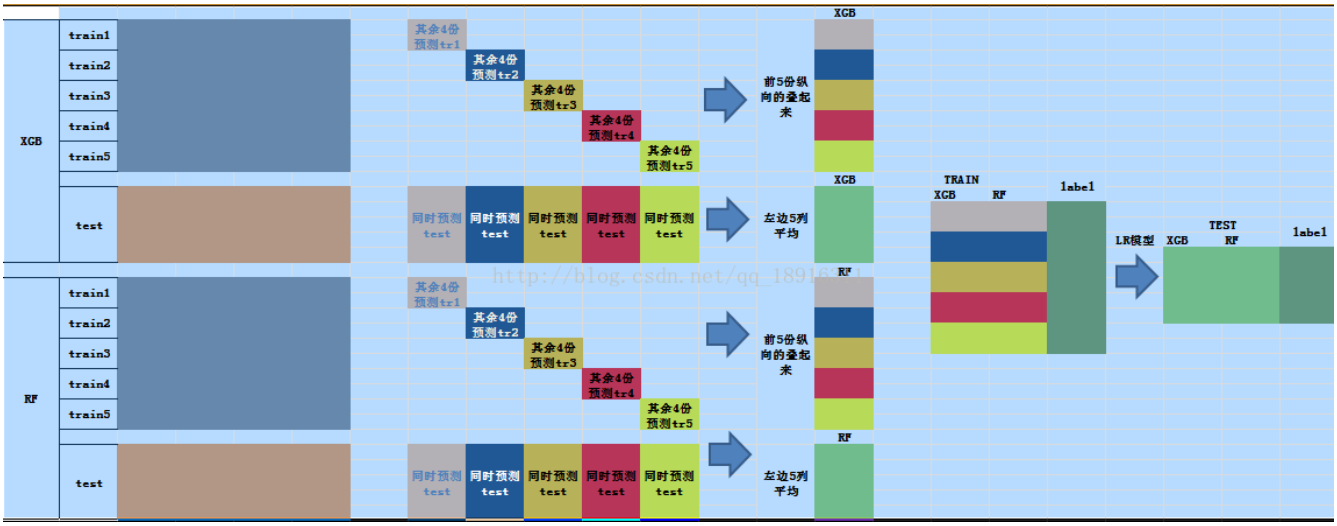
1.xgboost对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数。

3.xgboost在代价函数里增加了正则项，用于控制模型复杂度。正则项里包含了叶子的结点个数，每个叶子结点输出的score的L2的模的平方和。正则项降低了模型的方差，使模型更简单，防止过拟合

4.**支持并行化，直接的效果是训练速度快**，boosting技术中下一棵树依赖上述树的训练和预测，所以树与树之间应该是只能串行！但是，**同层级节点可并行。具体地，对于某个节点，节点内选择最佳分类点，进行枚举的时候可以并行**

Stacking

- Stacking简单理解就是讲几个简单的模型，一般采用将它们进行K折交叉验证输出预测结果，然后将每个模型输出的预测结果合并为新的特征，并使用新的模型加以训练。



XGB模型，把train分train1~train5,共5份，用其中4份预测剩下的那份,同时预测test数据，这样的过程做5次,生成5份train（原train样本数/5）数据和5份test数据。然后把5份预测的train数据纵向叠起来，把test预测的结果做平均。

RF模型和XGB模型一样，再来一次。这样就生成了2份train数据和2份test数据（XGB重新表达的数据和RF重新表达的数据）

然后用LR模型，进一步做融合，得到最终的预测结果。

5.CNN、DNN

卷积运算

对应相乘再相加

softmax函数

softmax用于多分类过程中，它将多个神经元的输出，映射到（0,1）区间内，可以看成概率来理解，从而来进行多分类

常用的激活函数及其作用：

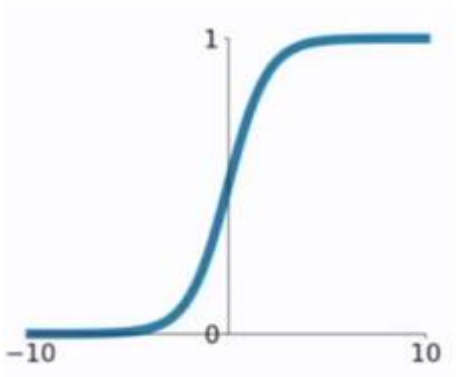
作用：通过加入非线性因素，提高模型对数据的拟合能力，以此解决线性不可分的问题

在深度学习中，常用的激活函数主要有：sigmod函数，tanh函数，ReLu函数

sigmod函数：

sigmod函数将取值为 (−∞,+∞)(−∞,+∞) 的数映射到 (0,1)(0,1) 之间。sigmoid函数的公式以及图形如下：

$$g(z) = \frac{1}{1 + e^{-1}}$$



sigmod函数缺点：

1.当z值非常大或者非常小时，sigmod导数趋于0，这会导致权重w的梯度将接近于0，即出现梯度消失现象

2.函数的输出不是以0为为均值。那么对于一个多层的sigmod神经网络来说，如果你输入的都是正数，在反向传播中w的梯度传播到网络的某一处时，权值的变化是要么全正要么全负，模型拟合的过程就会

非常慢

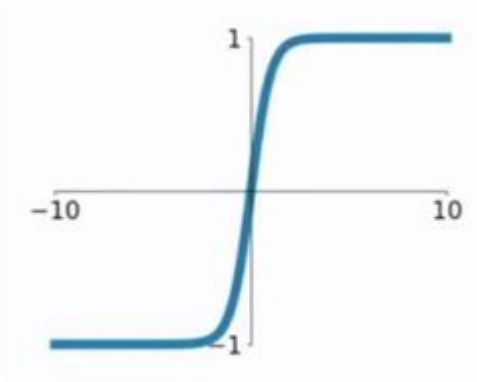
所以，

sigmoid函数可用在网络最后一层，作为输出层进行二分类，尽量不要使用在隐藏层。

tanh函数：

tanh函数相较于sigmoid函数要常见一些，该函数是将取值为 (−∞,+∞)(−∞,+∞) 的数映射到 (−1,1)(−1,1) 之间，其公式与图形为：

$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$



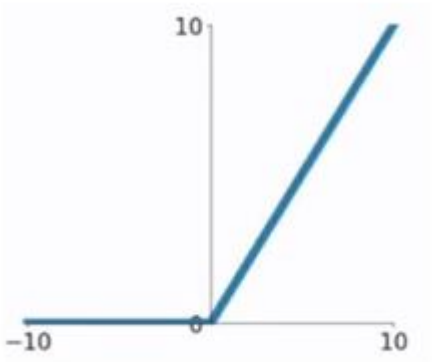
tanh函数在 00 附近很短一段区域内可看做线性的。由于tanh函数**均值**为 0 ，因此弥补了sigmoid函数均值为 0.5的缺点。

tanh函数的**缺点**同sigmoid函数的第一个缺点一样，当 z **很大或很小**时，g′(z)接近于 0，会导致梯度很小，权重更新非常缓慢，即**梯度消失问题**。

ReLU函数

ReLU函数又称为**修正线性单元（Rectified Linear Unit）**，是一种分段线性函数，其弥补了sigmoid函数以及tanh函数的**梯度消失问题**。ReLU函数的公式以及图形如下：

$$g(z) = \begin{cases} 0, & z < 0 \\ z, & z > 0 \end{cases}$$



ReLU函数的**优点**： 在输入为正数的时候,不存在梯度消失问题。 计算速度要快很多。ReLU函数只有线性关系，不管是前向传播还是反向传播，都比sigmod和tanh要快很多。（sigmod和tanh要计算指数，计算速度会比较慢） ReLU函数的**缺点**： 当输入为负时，梯度为0，会产生梯度消失问题。

神经网络：

神经网络的学习就是学习如何利用矩阵的线性变换加激活函数的非线性变换，将原始输入空间投向线性可分/稀疏的空间去分类/回归。**增加节点数：增加维度，即增加线性转换能力。增加层数：增加激活函数的次数，即增加非线性转换次数。**

对卡在局部极小值的处理方法：

1.调节步伐：调节学习速率，使每一次的更新“步伐”不同。常用方法：随机梯度下降、小批量梯度下降、增加动量

BGD(Batch gradient descent)批量梯度下降法：**每次迭代使用所有的样本**

每次迭代都需要把所有样本都送入，每训练一组样本就把梯度更新一次。。

SGD（Stochastic gradientdescent）随机梯度下降法：**每次迭代使用一组样本**

针对BGD算法训练速度过慢的缺点，提出了SGD算法，SGD算法是从样本中随机抽出一组，训练后按梯度更新一次，然后再抽取一组，再更新一次，在样本量及其大的情况下，可能不用训练完所有的样本就可以获得一个损失值在可接受范围之内的模型了

MBGD（Mini-batch gradient descent）小批量梯度下降：**每次迭代使用b组样本**

SGD相对来说要快很多，但是也有存在问题，由于单个样本的训练可能会带来很多噪声，使得SGD并不是每次迭代都向着整体最优化方向，因此在刚开始训练时可能收敛得很快，但是训练一段时间后就会变得很慢。**在此基础上又提出了小批量梯度下降法，它是每次从样本中随机抽取一小批进行训练，而不是一组。**

2.优化起点：合理初始化权重（weights initialization）、预训练网络（pre-train），使网络获得一个较好的“起始点”，如最右侧的起始点就比最左侧的起始点要好。

常用方法有：高斯（正态）分布初始权重（Gaussian distribution）、均匀分布初始权重（Uniform distribution）

初始化权重的意义：决定了loss在loss function中从哪个点开始作为起点训练网络。常用方法介绍：

均匀分布初始化权重：**[0,1]范围的均匀分布** 可以使用TensorFlow提供的tf.random_uniform()函数。该函数默认在[0，1]范围内取值。

正态分布初始化权重：上面尝试的权重初始化方法都是在权重的取值要靠近0而不能太小的方向上进行着。正态分布正好符合这个方向，其大部分取值靠近0。

浅层VS深层：

相比浅层神经网络，深层神经网络可以用**更少的数据量**来学到更好的拟合。深层的前提是：空间中的元素可以由卷积迭代而来的。

防止过拟合：

过拟合具体表现在：模型在训练数据上损失函数较小，预测准确率较高；但是在测试数据上损失函数比较大，预测准确率较低。

防止过拟合的方法：

1. L 2正则化
2. Dropout
3. 每个epoch之后shuffle训练数据，设置early-stopping
4. 加Batch Normalization（对隐藏层的神经元在激活值获得之后，非线性变换之前进行强制归一化）

名词解释：

epoch：当一个完整的数据集通过了神经网络一次并且返回了一次，这个过程称为一个 epoch。随着 epoch 数量增加，神经网络中的权重的更新次数也增加，曲线从欠拟合变得过拟合。

batch size：一个 batch 中的样本总数。记住：batch size 和 number of batches 是不同的

batch 是什么？---------在不能将数据一次性通过神经网络的时候，就需要将数据集分成几个 batch。

iteration: 迭代数是 batch 需要完成一个 epoch 的次数。记住：在一个 epoch 中，batch 数和迭代数是相等的。

具体解释：

1. L 2正则化：L 2正则表达式是 所有参数平方和的开方值。符合高斯分布，是完全可微的。L 2正则化使值最小时对应的参数变小。
2. Dropout：在每个训练批次中，让一半的隐层节点值为0。也就是在向前传播时，让某个神经元的激活值以一定的概率p停止工作，这样可以使模型泛化性更强。

3. 将shuffle参数设置为True, 则训练数据将在每个epoch混洗；**设置early-stopping**（在训练过程中，记录到目前为止最好的验证集精度，当连续10次Epoch（或者更多次）没达到最佳精度时，则可以认为精度不再提高了，则停止训练，将停止之后的权重作为网络的参数），所以early stopping要做的就是在中间点停止迭代过程。我们将会得到一个中等大小的w参数，会得到与L2正则化相似的结果，选择了w参数较小的神经网络。
4. 加Batch Normalization，对隐藏层每个神经元做归一化。即可以想象成在每个隐层上又加了一层BN操作层，它位于x=wu+b激活值获得之后，非线性函数变换之前。**对于每个隐层神经元，把输入分布强制拉回到均值为0方差为1的比较标准的正态分布，使得非线性变换函数的输入值落入对输入比较敏感的区域，以此避免梯度消失问题。**因为梯度一直都能保持比较大的状态，所以收敛地快，能提升效果。

CNN卷积神经网络的层级结构

CNN的三个基本层：

（1）卷积：对图像元素的矩阵变换，是提取图像特征的方法，多种卷积核可以提取多种特征。一个卷积核覆盖的范围体现**权值共享**。一次卷积运算提取的特征往往是局部的，难以提取出比较全局的特征，因此需要在一层卷积基础上继续做卷积计算，这也就是多层卷积。

（2）池化：池化层夹在连续的卷积层中间， 用于压缩数据和参数的量，减小过拟合。池化层用的方法：Max pooling, average pooling。池化层的具体作用：特征不变性，特征降维，一定程度防止过拟合。 （3）全连接：softmax分类

CNN所有层级结构：

1. 数据输入层 （预处理）：

   [1去均值：把输入数据各个维度都中心化为0，其目的就是把样本的中心拉回到坐标系原点上。

   [2归一化：幅度归一化到同样的范围，即减少各维度数据取值范围的差异而带来的干扰，比如，我们有两个维度的特征A和B，A范围是0到10，而B范围是0到10000，如果直接使用这两个特征是有问题的，好的做法就是归一化，即A和B的数据都变为0到1的范围。

   [3PCA：用PCA降维

2. 卷积计算层

   在这个卷积层，有两个关键操作： • 局部关联。每个神经元看做一个滤波器(filter) • 窗口(receptive field)滑动， filter对局部数据计算

   卷积层遇到的几个名词： • 深度/depth（上一层卷积核的个数（滤波器的个数）就是下一层卷积层的深度） • 步长/stride （窗口一次滑动的长度） • 填充值/zero-padding

3. ReLU激励层

   把卷积层输出结果做非线性映射。CNN采用的激励函数一般为ReLU(The Rectified Linear Unit/修正线性单元)，它的特点是收敛快，求梯度简单，但较脆弱。

   激励层的实践经验： ①不要用sigmoid！不要用sigmoid！不要用sigmoid！ ② 首先试RELU，因为快，但要小心点 ③ 如果2失效，请用Leaky ReLU或者Maxout ④ 某些情况下tanh倒是有不错的结果，但是很少

4. 池化层：max pooling

5. 全连接层

   通常在CNN的尾部进行重新拟合，减少特征信息的损失。做完Max Pooling后，我们就会把这些数据“拍平”，丢到Flatten层，然后把Flatten层的output放到full connected Layer里，采用softmax对其进行分类。

CNN卷积神经网络的优点

局部连接、权值共享

CNN与DNN的区别

DNN的输入是向量形式，并未考虑到平面的结构信息，在图像和NLP领域这一结构信息尤为重要，例如识别图像中的数字，同一数字与所在位置无关（换句话说任一位置的权重都应相同），CNN的输入可以是tensor，例如二维矩阵，通过filter获得局部特征，较好的保留了平面结构信息。

6.LDA主题模型

LDA主题模型，它能够**将文档集中每篇文档的主题以概率分布的形式给出。**从而通过分析一些文档抽取出它们的主题分布出来后，便能够依据主题（分布）进行主题聚类或文本分类。同一时候，它是一种典型的词袋模型，即一篇文档是由一组词构成，词与词之间没有先后顺序的关系。此外，一篇文档能够包括多个主题，文档中每一个词都由当中的一个主题生成。

人类是怎么生成文档的呢？LDA的这三位作者在原始论文中给了一个简单的样例。比方假设事先给定了这几个主题：Arts、Budgets、Children、Education，然后通过学习训练。获取每一个主题Topic相应的词语。例如以下图所看到的：

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

然后以一定的概率选取上述某个主题，再以一定的概率选取那个主题下的某个单词，不断的反复这两步，终于生成例如以下图所看到的的一篇文章（当中不同颜色的词语分别相应上图中不同主题下的词）：

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

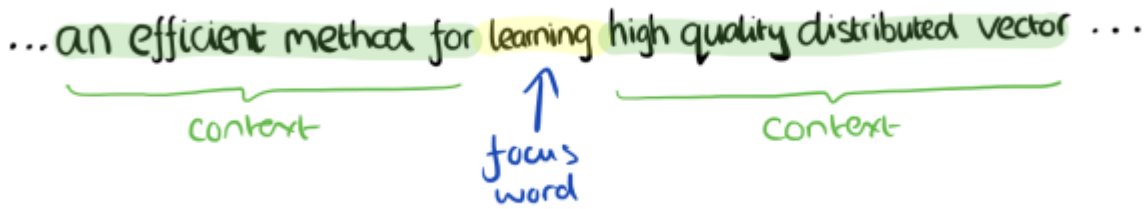
而当我们看到一篇文章后。往往喜欢推测这篇文章是怎样生成的，**我们可能会觉得作者先确定这篇文章的几个主题，然后环绕这几个主题遣词造句，表达成文。**

LDA就是要干这事：**依据给定的一篇文档，推测其主题分布（**你计算机给我推测分析网络上各篇文章分别都写了些啥主题，且各篇文章中各个主题出现的概率大小（主题分布）是啥。）

7.Word2Vec <-> 17.语言模型 || glove词向量

1.原始CBOW

Cbow模型的训练输入是某一个特征词的上下文相关的词对应的词向量，而输出就是这特定的一个词的词向量。比如下面这段话，我们的上下文大小取值为4，特定的这个词是"Learning"，也就是我们需要的输出词向量,上下文对应的词有8个，前后各4个，这8个词是我们模型的输入。由于CBOW使用的是词袋模型，因此这8个词都是平等的，也就是不考虑他们和我们关注的词之间的距离大小，只要在我们上下文之内即可。



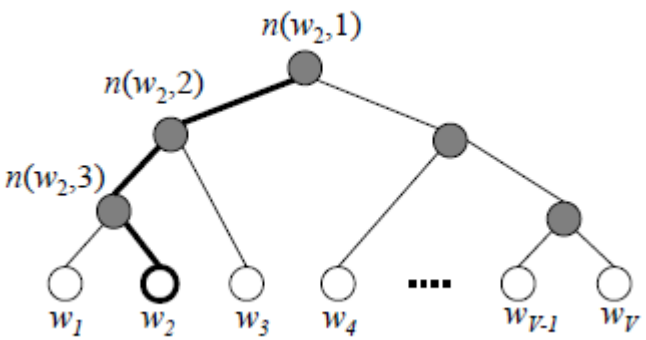
这样我们这个CBOW的例子里，我们的输入是8个词向量，输出是所有词的softmax概率（训练的目标是期望训练样本特定词对应的softmax概率最大），对应的CBOW神经网络模型输入层有8个神经元，输出层有词汇表大小个神经元。隐藏层的神经元个数我们可以自己指定。通过DNN的反向传播算法，我们可以求出DNN模型的参数，同时得到所有的词对应的词向量。这样当我们有新的需求，要求出某8个词对应的最可能的输出中心词时，我们可以通过一次DNN前向传播算法并通过softmax激活函数找到概率最大的词对应的神经元即可。

2.改进CBOW

word2vec也使用了CBOW与Skip-Gram来训练模型与得到词向量，但是并没有使用传统的DNN模型。**最先优化使用的数据结构是用霍夫曼树来代替隐藏层和输出层的神经元，霍夫曼树的叶子节点起到输出层神经元的作用，叶子节点的个数即为词汇表的小大。 而内部节点则起到隐藏层神经元的作用。**

word2vec对这个模型做了改进， **（改进1）首先，对于从输入层到隐藏层的映射**，没有采取神经网络的线性变换加激活函数的方法，**而是采用简单的对所有输入词向量求和并取平均的方法。**比如输入的是三个4维词向量：(1,2,3,4),(9,6,11,8),(5,10,7,12)(1,2,3,4),(9,6,11,8),(5,10,7,12),那么我们word2vec映射后的词向量就是(5,6,7,8)(5,6,7,8)。**由于这里是从多个词向量变成了一个词向量。**

第二个改进就是**(改进2)从隐藏层到输出的softmax层这里的计算量个改进**。为了避免要计算所有词的softmax概率，**word2vec采样了霍夫曼树来代替从隐藏层到输出softmax层的映射。用霍夫曼树来代替隐藏层和输出层的神经元，霍夫曼树的叶子节点起到输出层神经元的作用，叶子节点的个数即为词汇表的大小。 而内部节点则起到隐藏层神经元的作用。**由于我们把之前所有都要计算的从输出softmax层的概率计算变成了一棵二叉霍夫曼树，那么我们的softmax概率计算只需要沿着树形结构进行就可以了。如下图所示，我们可以沿着霍夫曼树从根节点一直走到我们的叶子节点的词w2。



和之前的神经网络语言模型相比，**我们的霍夫曼树的所有内部节点就类似之前神经网络隐藏层的神经元,其中，根节点的词向量对应我们的投影后的词向量，而所有叶子节点就类似于之前神经网络softmax输出层的神经元，叶子节点的个数就是词汇表的大小。**在霍夫曼树中，隐藏层到输出层的softmax映射不是一下子完成的，而是沿着霍夫曼树一步步完成的，因此这种softmax取名为"Hierarchical Softmax"。

如何“沿着霍夫曼树一步步完成”呢？**在word2vec中，我们采用了二元逻辑回归的方法，即规定沿着左子树走，那么就是负类(霍夫曼树编码1)，沿着右子树走，那么就是正类(霍夫曼树编码0)。判别正类和负类的方法是使用sigmoid函数，**

这里总结下基于Hierarchical Softmax的CBOW模型算法流程，**梯度迭代使用了随机梯度上升法：**

输入：基于CBOW的语料训练样本，词向量的维度大小M，CBOW的上下文大小2c,步长η

输出：霍夫曼树的内部节点模型参数θ，所有的词向量w

1. 基于语料训练样本建立霍夫曼树。
2. 随机初始化所有的模型参数θ，所有的词向量w
3. 进行梯度上升迭代过程

3.改进Skip-Gram

现在我们先看看基于Skip-Gram模型时，Hierarchical Softmax如何使用。此时输入的只有一个词w,输出的为2c个词向量context(w)。

我们对于训练样本中的每一个词，该词本身作为样本的输入， 其前面的c个词和后面的c个词作为了Skip-Gram模型的输出,，期望这些词的softmax概率比其他的词大。

1.我们需要先将词汇表建立成一颗霍夫曼树。

2.对于从输入层到隐藏层（投影层），这一步比CBOW简单，由于只有一个词，所以，即Xw就是词w对应的词向量。

3.通过梯度上升法来更新我们的θ和Xw，注意这里的Xw周围有2c个词向量，**Skip-Gram模型并没有和CBOW模型一样对输入进行迭代更新，而是对2c个输出进行迭代更新。**

4.glove模型

我的理解是skip-gram、CBOW每次都是用一个窗口中的信息更新出词向量，但是Glove则是用了全局的信息（共现矩阵），也就是多个窗口进行更新.

注意，glove模型的代价函数如下，建议记住。具体由来下面会介绍。N是词汇表的大小，Vi,Vj是单词i和j的词向量，bi,bj是两个标量（作者定义的偏差项，）N是词汇表的大小（共现矩阵维度为N*N）

$$J=\sum_{i,j}^{N} f(X_{i,j})(v_i^T v_j+b_i+b_j-log(X_{i,j}))^2$$

具体权重函数f(x)应该是怎么样的呢？ 首先应该是非减的，其次当词频过高时，权重不应过分增大，作者通过实验确定权重函数f(x)为：

$$f(x)=\begin{cases}(\frac{x}{x_{max}})^{0.75}, & x<x_{max}\\ 1, & x>=x_{max}\end{cases}$$

具体方法概述：

首先基于语料库构建词的共现矩阵，然后基于共现矩阵和GloVe模型学习词向量。** 开始 -> 统计共现矩阵 -> 训练词向量 -> 结束**

1.统计共现矩阵

设共现矩阵为X，其元素为Xij . Xij的意义为：在整个语料库中，单词i和单词j共同出现在一个窗口中的次数。举个例子，设有语料库，i love you but you love him i am sad

这个小小的语料库只有1个句子，涉及到7个单词：i、love、you、but、him、am、sad。 如果我们采用一个窗口宽度为5（左右长度都为2）的统计窗口，那么就有以下窗口内容：

窗口标号	中心词	窗口内容
0	i	i love you
1	love	i love you but
2	you	i love you but you
3	but	love you but you love
4	you	you but you love him
5	love	but you love him i
6	him	you love him i am
7	i	love him i am sad
8	am	him i am sad
9	sad	i am sad

窗口0、1长度小于5是因为中心词左侧内容少于2个，同理窗口8、9长度也小于5。 以窗口5为例说明如何构造共现矩阵：中心词为love，语境词为but、you、him、i；则执行：

$$X_{love,but} += 1$$

$$X_{love,I} += 1$$

使用窗口将整个语料库遍历一遍，即可得到共现矩阵X。

2.使用glove模型训练词向量:

那么作者为什么这么构造模型呢？首先定义几个符号：

$$X_i = \sum_{j=1}^{N} X_{i,j}$$

其实就是矩阵单词i那一行的和；

$$P_{i,k} = \frac{X_{i,k}}{X_i}$$

条件概率，表示单词k出现在单词i语境中的概率；

$$ratio_{i,j,k} = \frac{P_{i,k}}{p_{j,k}}$$

两个条件概率的比率。作者的灵感是这样的： 作者发现，ratio{i,j,k}这个指标是有规律的，规律统计在下表：

***ratio**i,j,k*的值**	**单词j,k相关**	**单词j,k不相关**
单词i,k相关	趋近1	很大
单词i,k不相关	很小	趋近1

思想：假设我们已经得到了词向量，如果我们用词向量Vi,Vj,Vk通过某种函数计算ratioi,jk 能够同样得到这样的规律的话，就意味着我们词向量与共现矩阵具有很好的一致性，也就说明我们的词向量中蕴含了共现矩阵中所蕴含的信息。设用词向量Vi,Vj,Vk计算ratioi,j,k的函数为g(vi,vj,vk)（我们先不去管具体的函数形式），那么应该有：

$$\frac{P_{i,k}}{p_{j,k}} = ratio_{i,j,k} = g(v_i, v_j, v_k)$$

即：

$$\frac{P_{i,k}}{p_{j,k}} = g(v_i, v_j, v_k)$$

即二者应该尽可能地接近；很容易想到用二者的差方来作为代价函数：

$$J = \sum\nolimits_{i,j,k}^{N} (\frac{P_{i,k}}{p_{j,k}} - g(v_i, v_j, v_k))^2$$

但是仔细一看，模型中包含3个单词，这就意味着要在NNN的复杂度上进行计算，太复杂了，最好能再简单点。

作者的脑洞是这样的：

1. 要考虑单词i和单词j之间的关系，那g(vi,vj,vk)中大概要有这么一项吧：Vi-Vj；嗯，合理，在线性空间中考察两个向量的相似性，不失线性地考察，那么vi−vj大概是个合理的选择；
2. ratioi,j,k是个标量，那么g(vi,vj,vk)最后应该是个标量啊，虽然其输入都是向量，那内积应该是合理的选择，于是应该有这么一项吧：(Vi-Vj)^T(Vk) (行向量*列向量)
3. 然后作者又往(vi−vj)^T * vk的外面套了一层指数运算exp(). 套上之后，我们的目标是让以下公式尽可能地成立：

$$\frac{P_{i,k}}{p_{j,k}} = g(v_i, v_j, v_k)$$

即：

$$\frac{P_{i,k}}{p_{j,k}} = exp((V_i - Vj)^T * V_k)$$

即：

$$\frac{P_{i,k}}{p_{j,k}} = exp((V_i^T * V_k - V_j^T * V_k)$$

即：

$$\frac{P_{i,k}}{p_{j,k}} = \frac{exp(V_i^T * V_k)}{exp(V_j^T * V_k)}$$

然后就发现找到简化方法了：只需要让上式分子对应相等，分母对应相等，即：

$$P_{i,j} = exp(V_i^T * V_k)$$

两边取个对数：

$$log(P_{i,j}) = V_i^T * V_k$$

那么代价函数就可以化简为：

$$J = \sum_{i,j}^{N} log(P_{i,j} - V_i^T V_j)^2$$

现在只需要在N*N的复杂度上进行计算，而不是*NN*N，现在关于为什么第3步中，外面套一层exp()就清楚了，正是因为套了一层exp()，才使得差形式变成商形式，进而等式两边分子分母对应相等，进而简化模型。 现将代价函数中的条件概率展开：

$$log(P_{i,j}) = V_i^T V_j$$

即为：

$$log(X_{i,j}) - log(X_i) = V_i^T V_j$$

将其变为：

$$log(X_{i,j}) = V_i^T V_j + b_i + b_j$$

于是代价函数就变成了：

$$J = \sum_{i,j}^{N} (v_i^T v_j + b_i + b_j - log(X_{i,j}))^2$$

然后基于出现频率越高的词对儿权重应该越大的原则，在代价函数中添加权重项，于是代价函数进一步完善：

$$J = \sum_{i,j}^{N} f(X_{i,j})(v_i^T v_j + b_i + b_j - log(X_{i,j}))^2$$

8.WordEmbedding

如果将**word**看作文本的最小单元，可以将**Word Embedding**理解为一种**映射**，其过程是：**将文本空间中的某个word，通过一定的方法，映射或者说嵌入（embedding）到另一个数值向量空间**（之所以称之为**embedding**，是因为这种表示方法往往伴随着一种**降维**的意思。

WordEmbedding的输入：

Word Embedding的输入是原始文本中的一组不重叠的词汇，假设有句子：**apple on a apple tree**。那么为了便于处理，我们可以将这些词汇放置到一个**dictionary**里，例如：**["apple", "on", "a", "tree"]**，这个**dictionary**就可以看作是**Word Embedding**的一个输入。

WordEmbedding的输出：

Word Embedding的输出就是每个**word**的向量表示。

WordEmbedding的类型：

- 基于频率的**Word Embedding**（Frequency based embedding）
- 基于预测的**Word Embedding**（Prediction based embedding）

基于频率的Word Embedding

8.1TF-IDF

词频（TF） = 某个词在文章中的出现次数

逆文档频率（IDF） = log（语料库的文档总数/包含该词的文档总数+1）

8.2共现矩阵 <> 7.4 glove模型

glove模型，定义一个比如大小为5的共现窗口

基于预测的Word Embedding

词的表示中如果蕴含了上下文信息，那么将会更加接近自然语言的本质；并且，由于相似的词有相似的表示方法，甚至可以进行一些运算，例如：**人类-男人=女人。**但是，上述讨论中，有一个很大的缺陷，那就是词的向量表示维度过大，一个词要用大量其余的词来表示，为后续运算带来了很大的麻烦。因此，我们需要找到一种更好的表示方法，这种方法需要满足如下两点要求：

1.携带上下文信息

2.词的表示是稠密的

方法：CBOW、Skip-Gram

9.tf-idf

tf:

词频（TF） = 某个词在文章中的出现次数

文章有长短之分，为了便于不同文章的比较,做"词频"标准化。

词频（TF） = 某个词在文章中的出现次数 / 文章总词数

或者 词频（TF） = 某个词在文章中的出现次数 / 拥有最高词频的词的次数

Idf:

逆文档频率（IDF） = log（语料库的文档总数/包含该词的文档总数+1）

TF-IDF = 词频（TF) * 逆文档频率（IDF）

优点：

TF-IDF算法的优点是简单快速，结果比较符合实际情况。

缺点：

单纯以"词频"衡量一个词的重要性，不够全面，有时重要的词可能出现次数并不多。而且，这种算法无法体现词的位置信息，出现位置靠前的词与出现位置靠后的词，都被视为重要性相同，这是不正确的。

（一种解决方法是，对全文的第一段和每一段的第一句话，给予较大的权重）

10.SVM

SVM原理

基于训练样本D 在二维空间中找到一个符合这样条件的超平面来分开二类样本。这个超平面离两类样本都足够远，也就是使得“间隔”最大。即最终确定的参数w和b,使得r最大。即要：

min (1/2)||w||^2

实际中，对某个实际问题函数来寻找一个合适的空间进行映射是非常困难的，幸运的是，在计算中发现，我们需要的只是两个向量在新的映射空间中的内积结果，而映射函数到底是怎么样的其实并不需要知道。这就需要引入了核函数的概念。**核函数是这样的一种函数：**仍然以二维空间为例，假设对于变量x和y，将其映射到新空间的映射函数为φ，则在新空间中，二者分别对应φ(x)和φ(y)，他们的内积则为<φ(x),φ(y)>。我们令函数Kernel(x,y)=<φ(x),φ(y)>=k(x,y)，**可以看出，函数Kernel(x,y)是一个关于x和y的函数！而与φ无关！**这是一个多么好的性质！我们再也不用管φ具体是什么映射关系了，只需要最后计算Kernel(x,y)就可以得到他们在高维空间中的内积，这样就可以直接带入之前的支持向量机中计算！

RBF是最常用的核函数，RBF核函数可以将维度扩展到无穷维的空间，RBF对应的是泰勒级数展开，在泰勒级数中，一个函数可以分解为无穷多个项的加和，其中，每一个项可以看做是对应的一个维度，这样，原函数就可以看做是映射到了无穷维的空间中. 通过计算间隔和松弛变量等的最大化，可以对问题进行求解。

逻辑回归和SVM区别

相同点:

1. 都是分类算法
2. 都是监督学习算法
3. 都是判别模型
4. 都能通过核函数方法针对非线性情况分类
5. 目标都是找一个分类超平面
6. 都能减少离群点的影响

不同点:

1. **损失函数不同，逻辑回归是对数损失函数，svm是hinge loss，自带L2正则项**
2. **逻辑回归在优化参数时所有样本点都参与了贡献，svm则只取离分离超平面最近的支持向量样本。这也是为什么逻辑回归不用核函数，它需要计算的样本太多。并且由于逻辑回归受所有样本的影响，当样本不均衡时需要平衡一下每一类的样本个数。**
3. 逻辑回归对概率建模，svm对分类超平面建模
4. 逻辑回归是处理经验风险最小化，svm是结构风险最小化。这点体现在svm自带L2正则化项，逻辑回归并没有
5. 逻辑回归通过非线性变换减弱分离平面较远的点的影响，svm则只取支持向量从而消去较远点的影响
6. 逻辑回归是统计方法，svm是几何方法

SVM的参数

SVM模型有两个非常重要的参数C与gamma。

1.C是惩罚系数，即对误差的宽容度。c越高，说明越不能容忍出现误差,容易过拟合。C越小，容易欠拟合。C过大或过小，泛化能力变差

2.gamma是选择RBF函数作为kernel后，该函数自带的一个参数。隐含地决定了数据映射到新的特征空间后的分布，gamma越大，支持向量越少，σ越小（σ^2是正太分布的方差，描述数据的离散程度），数据分布越集中，曲线越瘦高，**那么会造成只作用于支持向量样本附近，对于未知样本分类效果很差**，就会存在训练准确率很高而测试准确率不高的可能，也就是过训练。 而若gamma太小，会造成平滑效应太大，无法在训练集上得到比较好的结果，更不用说测试集准确率。

11.LexRank 、TextRank

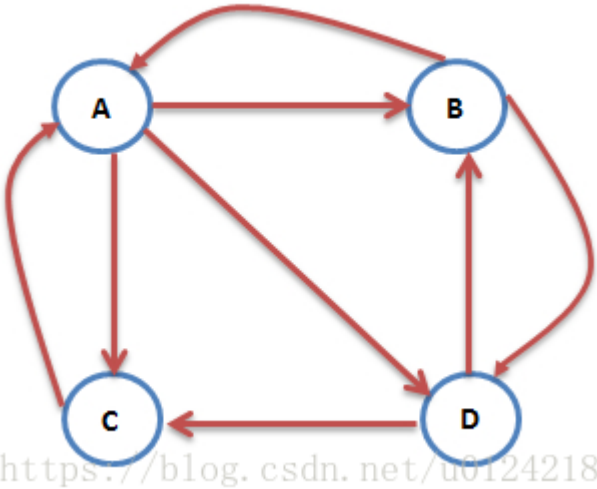
PageRank:

对于某个互联网网页A来说，该网页PageRank的计算基于下面两个基本如果： □ 数量如果：在Web图模型中，**如果一个页面节点接收到的其它网页指向的入链数量越多**，那么这个页面越重要。 □ 质量如果：指向页面A的入链质量不同，**质量高的页面会通过链接向其它页面传递很多其它的权重**。所以越是质量高的页面指向页面A，则页面A越重要。

PageRank的计算充分利用了两个如果：数量如果和质量如果。过程例如以下： **1）在初始阶段：**网页通过链接关系构建起Web图，每一个页面设置同样的PageRank值，通过若干轮的计算，会得到每一个页面所获得的终于PageRank值。随着每一轮的计算进行，网页当前的PageRank值会不断得到更新。

2）在一轮中更新页面PageRank得分的计算方法：在一轮更新页面PageRank得分的计算中，**每一个页面将其当前的PageRank值平均分配到本页面包括的出链上**，这样每一个链接即获得了对应的权值。**而每一个页面将全部指向本页面的入链所传入的权值求和，就可以得到新的PageRank得分。**当每一个页面都获得了更新后的PageRank值，就完毕了一轮PageRank计算。

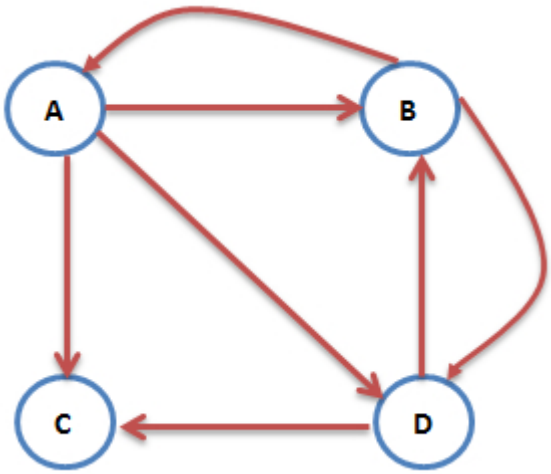
Case1.网页都有出入链



此种情况下的网页A的PR值计算公式为：

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{1}$$

Case2存在没有出链的网页:

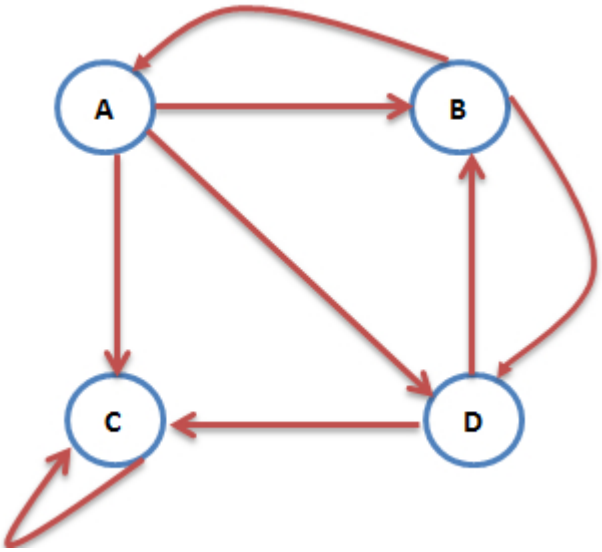


网页C是没有出链。因为C没有出链，所以对A,B,D网页没有PR值的贡献。PageRank算法的策略：从数学上考虑，为了满足Markov链，设定C对A,B,C,D都有出链（也对他自己也出链~）。你也可以理解为：没有出链的网页，我们强制让他对所有的网页都有出链，即让他对所有网页都有PR值贡献。

此种情况PR(A)的计算公式：

$$PR(A) = \frac{PR(B)}{2} + \frac{PR(C)}{4}$$

Case3存在只对自己有出链的网页：



C是只对自己出链的网页。

此时访问C时，不会傻乎乎的停留在C页面，一直点击C-Link循环进入C，即C网页只对自己的网页PR值有贡献。正常的做法是，进入C后，存在这种情况：在地址输入栏输入A/B/C/D的URL地址，然后跳转到A/B/C/D进行浏览，这就是PageRank算法解决这种情况的策略：设定存在一定概率为α，用户在地址栏输入A/B/C/D地址，然后从C跳转到A/B/C/D进行浏览。

此时PR(A)的计算公式为：

$$PR(A)=\alpha(\frac{PR(B)}{2})+\frac{(1-\alpha)}{4}$$ ，一般取值α=0.85

一般情况下，一个网页的PR值计算公式为：

$$PR(p_i)=\alpha\sum_{p_j\in M_{p_i}}\frac{PR(p_j)}{L(p_j)}+\frac{(1-\alpha)}{N}$$

注：Mpi是有出链到pi的所有网页集合，L(pj)是有网页pj的出链总数，N是网页总数，α一般取值为0.85（阻尼系数）

所有网页PR值同时计算需要迭代计算：一直迭代计算，停止直到下面2情况之一发生：**每个网页的PR值前后误差dleta_pr小于自定义误差阈值，或者迭代次数超过了自定义的迭代次数阈值**

TextRank:

基于TextRank的自动文摘属于自动摘录，通过选取文本中重要度较高的句子形成文摘，其主要步骤如下：

（1）预处理：将输入的文本或文本集的内容分割成句子得 $T=[S_1,S_2,\cdots,S_m]$，构建图G = （V,E），其中V为句子集，对句子进行分词、去除停止词，得 $S_i=[t_{i,1},t_{i,2},\cdots,t_{i,n}]$，其中 $t_{i,j}\in S_j$ 是保留后的候选关键词。

（2）句子相似度计算：构建图G中的边集E，基于句子间的内容覆盖率，给定两个句子 S_i ， S_j ，采用如下公式进行计算：

$$Similarity(S_i,S_j)=\frac{\left|\{t_k\vee t_k\in S_i\wedge t_k\in S_j\}\right|}{\log\left||S_i|\right|+\log\left||S_j|\right|}$$

若两个句子之间的相似度大于给定的阈值，就认为这两个句子语义相关并将它们连接起来，即边的权值； $w_{ji}=Similarity(S_i,S_j)$

（3）句子权重计算：根据公式，迭代传播权重计算各句子的得分；

$$WS(V_i) = (1-d) + d * \sum_{V_j \in In(V_i)} \frac{w_{ji}}{\sum_{V_k \in Out(V_j)} w_{jk}} WS(V_j)$$

和pageRank的公式相比，基本上就是把原来对应边的部分添加了权重，边的数量和改成了权重和，很好理解。

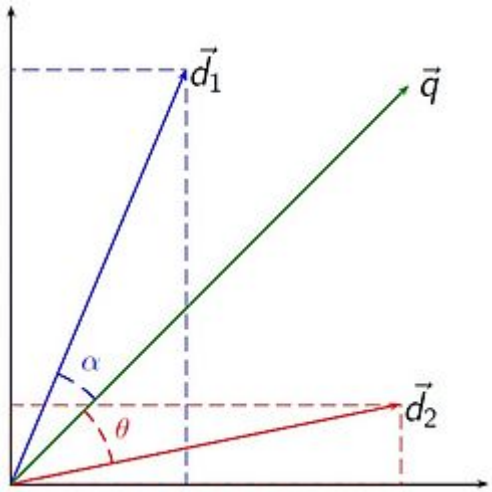
（4）抽取文摘句：将（3）得到的句子得分进行倒序排序，抽取重要度最高的T个句子作为候选文摘句。

（5）形成文摘：根据字数或句子数要求，从候选文摘句中抽取句子组成文摘。

使用 TextRank 算法计算图中各节点的得分时，同样需要给图中的节点指定任意的初值，通常都设为1。然后递归计算直到收敛，即图中任意一点的误差率小于给定的极限值时就可以达到收敛，一般该极限值取 0.0001。

其它计算相似度的方法有：基于编辑距离，基于语义词典，余弦相似度等。

12.VSM向量空间模型



我们把文档看作一系列词(Term)，每一个词(Term)都有一个权重(Term weight)，不同的词(Term)根据自己在文档中的权重来影响文档相关性的打分计算。 于是我们把所有此文档中词(term)的权重(term weight) 看作一个向量。 Document = {term1, term2, …… ,term N} Document Vector = {weight1, weight2, …… ,weight N} 同样我们把查询语句看作一个简单的文档，也用向量来表示。 Query = {term1, term 2, …… , term N} Query Vector = {weight1, weight2, …… , weight N}

我们把所有搜索出的文档向量及查询向量放到一个N维空间中，每个词(term)是一维。

我们认为两个向量之间的夹角越小，相关性越大。 所以我们计算夹角的余弦值作为相关性的打分，夹角越小，余弦值越大，打分越高，相关性越大。

我们只要比较下图中的α,θ的余弦值的大小，余弦值越大，相似度越高。公式如下：

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{||A|| ||B||} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}.$$

这里的 A_i, B_i 分别代表向量A和B的各分量。

14.关联规则

如何来度量一个规则是否够好？有两个量，**置信度(Confidence)和支持度(Support)**。假设有如下表的购买记录。

置信度表示了这条规则有多大程度上值得可信。设条件的项的集合为A,结果的集合为B。**置信度计算在A中，同时也含有B的概率。即Confidence(A==>B)=P(B|A)。**例 如计算"如果Orange则Coke"的置信度。由于在含有Orange的4条交易中，仅有2条交易含有Coke.其置信度为0.5。

支持度计算在所有的交易集中，既有A又有B的概率。例如在5条记录中，既有Orange又有Coke的记录有2条。则此条规则的支持度为2/5=0.4。现在这条规则可表述为，如果一个顾客购买了Orange,则有50%的可能购买Coke。而这样的情况（即买了Orange会再买Coke）会有40%的可能发生。

算法原理和过程

1.找出所有的频繁项目集，并计算其支持度。这些项集出现的频繁性至少和预定义的最小支持度一 样。

2.然后由频集产生强关联规则，这些规则必须满足最小支持度和最小可信度。

3.然后使用第1步找到的频繁项目集产生期望的规则，产生只包含集合的项的所有规则，其中 每一条规则的右部只有一项。

15.决策树

首先，在了解树模型之前，自然想到树模型和线性模型有什么区别呢？其中最重要的是，树形模型是一个一个特征进行处理，之前线性模型是所有特征给予权重相加得到一个新的值。决策树与逻辑回归的分类区别也在于此，**逻辑回归是将所有特征变换为概率后，通过大于某一概率阈值的划分为一类，小于某一概率阈值的为另一类；而决策树是对每一个特征做一个划分。**另外逻辑回归只能找到线性分割（输入特征x与logit之间是线性的，除非对x进行多维映射），而决策树可以找到非线性分割。

而树形模型更加接近人的思维方式，可以产生可视化的分类规则，产生的模型具有可解释性（可以抽取规则）。树模型拟合出来的函数其实是分区间的阶梯函数。

决策树学习：采用自顶向下的递归的方法，基本思想是以信息熵为度量构造一棵熵值下降最快的树，到叶子节点处熵值为0（叶节点中的实例都属于一类）。

决策树的生成：

决策树思想，实际上就是寻找最纯净的划分方法，**这个最纯净在数学上叫纯度，纯度通俗点理解就是目标变量要分得足够开（y=1的和y=0的混到一起就会不纯）。另一种理解是分类误差率的一种衡量。实际决策树算法往往用到的是，纯度的另一面也即不纯度，下面是不纯度的公式。不纯度的选取有多种方法，每种方法也就形成了不同的决策树方法，比如**ID3算法**使用信息增益作为不纯度；**C4.5算法**使用信息增益率作为不纯度；**CART算法使用**基尼系数**作为不纯度。

决策树要达到寻找最纯净划分的目标要干两件事，建树和剪枝

建树：

（1）如何按次序选择属性（最小熵，最大增益，最大增益率，最小基尼系数）

也就是首先树根上以及树节点是哪个变量呢？这些变量是从最重要到次重要依次排序的，那怎么衡量这些变量的重要性呢？ **ID3算法**用的是信息增益，**C4.5****算法**用信息增益率；**CART算法****使用基尼系数。决策树方法是会把每个特征都试一遍，然后选取那个，能够使分类分的最好的特征，也就是说将A属性作为父节点，产生的纯度增益（GainA）要大于B属性作为父节点，则A作为优先选取的属性。

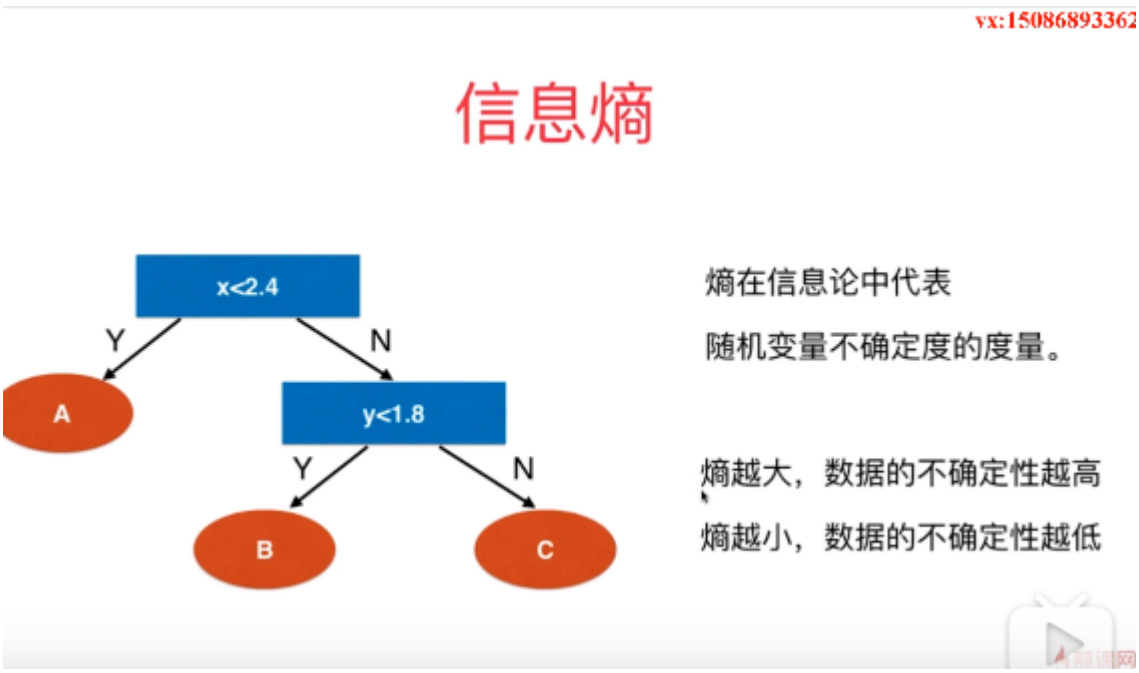
2） 如何分裂训练数据（对每个属性选择最优的分割点）

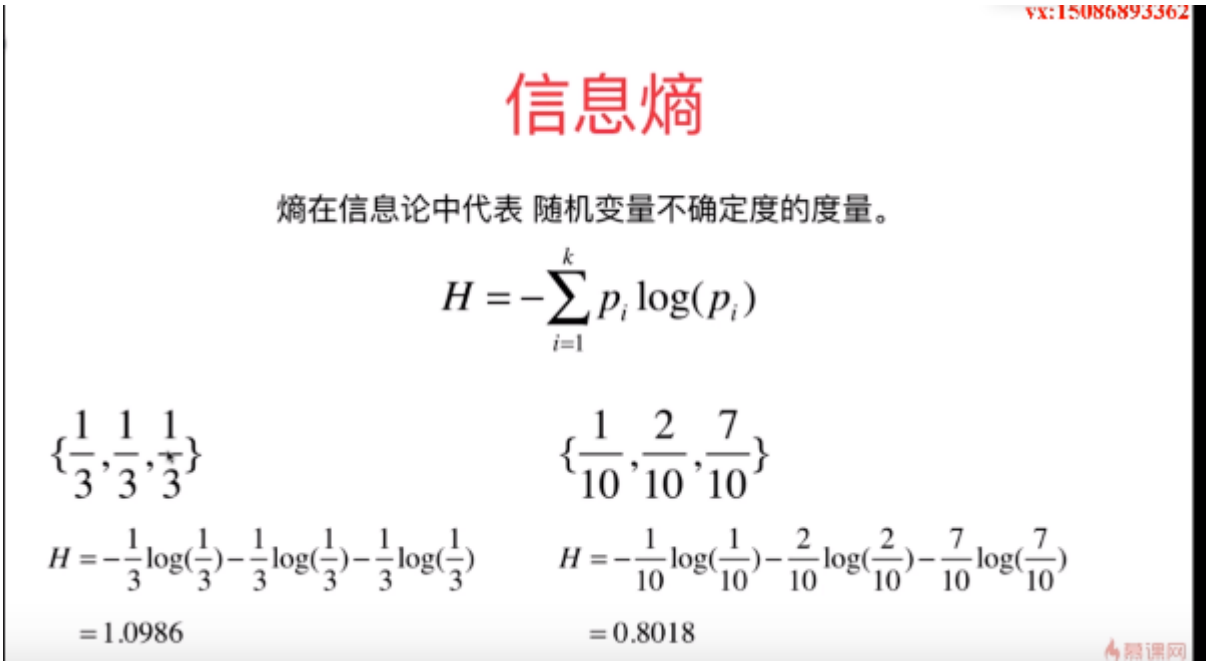
如何分裂数据也即分裂准则是什么？依然是通过不纯度来分裂数据的，通过比较划分前后的不纯度值，来确定如何分裂。

-

-

-





信息熵表示不确定性

信息熵越小，表示不确定性越低，即稳定性越高，因为右侧有一类占了70%，这个类别影响力更大

CART分类树在很多书籍和资料中介绍比较多，但是再次强调GDBT中使用的是回归树。作为对比，先说分类树，我们知道CART是二叉树，CART分类树在每次分枝时，是穷举每一个feature的每一个阈值，根据GINI系数找到使不纯性降低最大的的feature以及其阀值，然后按照feature<=阈值，和feature>阈值分成的两个分枝，每个分支包含符合分支条件的样本。用同样方法继续分枝直到该分支下的所有样本都属于统一类别，或达到预设的终止条件，若最终叶子节点中的类别不唯一，则以多数人的类别作为该叶子节点的性别。回归树总体流程也是类似，不过在每个节点（不一定是叶子节点）都会得一个预测值，以年龄为例，该预测值等于属于这个节点的所有人年龄的平均值。分枝时穷举每一个feature的每个阈值找最好的分割点，但**衡量最好的标准不再是GINI系数，而是最小化均方差--即（每个人的年**

龄-预测年龄）^2 的总和 / N，或者说是每个人的预测误差平方和 除以 N。这很好理解，被预测出错的人数越多，错的越离谱，均方差就越大，通过最小化均方差能够找到最靠谱的分枝依据。分枝直到每个叶子节点上人的年龄都唯一（这太难了）或者达到预设的终止条件（如叶子个数上限），若最终叶子节点上人的年龄不唯一，则以该节点上所有人的平均年龄做为该叶子节点的预测年龄。

16.K-Means

核心思想：由用户指定k个初始质心（initial centroids），以作为聚类的类别（cluster），重复迭代直至算法收敛。

算法流程：

选取k个初始质心（作为初始cluster）；repeat：对每个样本点，计算得到距其最近的质心，将其类别标为该质心所对应的cluster；重新计算k个cluser对应的质心；until 质心不再发生变化

缺点：

k-Means是局部最优的，容易受到初始质心的影响。

17.语言模型 <> 7.Word2Vec(CBOW) <> N-gram <> 朴素贝叶斯

简单的说，**语言模型**（Language Model）是用来计算一个句子出现概率的模型，假设句子 $W = (w_1, w_2, ..., w_n)$,其中 w_i 代表句子中的第 i 个词语，则语句 W 以该顺序出现的概率可以表示为：p(S)=p(w1,w2,w3,w4,w5,...,wn) = p(w1)p(w2|w1)p(w3|w1,w2)...p(wn|w1,w2,...,wn-1)//链规则。那么，如何计算p(wi|w1,w2,...,wi-1)呢？最简单、直接的方法是直接计数做除法，如下：

p(wi|w1,w2,...,wi-1) = p(w1,w2,...,wi-1,wi) / p(w1,w2,...,wi-1) 但是，这里面临两个重要的问题：数据稀疏严重；参数空间过大，无法实用。

1.朴素贝叶斯

贝叶斯公式：

$$p(c|x) = \frac{p(x|c)p(c)}{p(x)}$$

判断一条微信朋友圈是不是广告。前置条件是：我们已经拥有了一个平日广大用户的朋友圈内容库，这些朋友圈当中，如果真的是在做广告的，会被“热心网友”打上“广告”的标签，我们要做的是把所有内容分成一个一个词，每个词对应一个维度，构建一个高维度空间 (别担心，这里未出现向量计算)。当出现一条新的朋友圈new_post，我们也将其分词，然后投放到朋友圈词库空间里。

这里的X表示多个特征（词）x1,x2,x3...组成的特征向量。

P(ad|x)表示：已知朋友圈内容而这条朋友圈是广告的概率。

利用贝叶斯公式，进行转换：

P(ad|X) = p(X|ad) p(ad) / p(X)

P(not-ad | X) = p(X|not-ad)p(not-ad) / p(X)

比较上面两个概率的大小，如果p(ad|X) > p(not-ad|X),则这条朋友圈被划分为广告，反之则不是广告。

朴素贝叶斯假设：**如果认为每个词都是独立的特征，那么朋友圈内容向量可以展开为分词(x1,x2,x3...xn)**，因此有了下面的公式推导：

P(ad|X)*p(x) = p(X|ad)p(ad) = p(x1, x2, x3, x4...xn | ad) p(ad), p(x)对于所有类别为常数

假设所有词相互条件独立，则进一步拆分：

P(ad|X)*p(x) = p(x1|ad)p(x2|ad)p(x3|ad)...p(xn|ad) p(ad)

至此，P(xi|ad)很容易求解，P(ad)为词库中广告朋友圈占所有朋友圈（训练集）的概率。虽然现实中，一条朋友圈内容中，相互之间的词不会是相对独立的，因为我们的自然语言是讲究上下文的,所以由朴素贝叶斯引出n-garm

2.n-gram

N-gram模型是一种语言模型（Language Model，LM），语言模型是一个基于概率的判别模型，它的输入是一句话（单词的顺序序列），输出是这句话的概率，即这些单词的联合概率（joint probability）。

N-gram本身也指一个由N个单词组成的集合，各单词具有先后顺序，且不要求单词之间互不相同。常用的有 Bi-gram (N=2) 和 Tri-gram (N=3)，一般已经够用了。例如在上面这句话里，我可以分解的 Bi-gram 和 Tri-gram：

Bi-gram : {I, love}, {love, deep}, {deep, learning}

Tri-gram : {I, love, deep}, {love, deep, learning}

N-gram中概率计算:

假设我们有一个由n个词组成的句子S=(w1,w2,⋯,wn) 如何衡量它的概率呢？让我们假设，每一个单词wi都要依赖于从第一个单词w1到它之前一个单词wi−1的影响：

p(S)=p(w1w2⋯wn)=p(w1)p(w2|w1)⋯p(wn|wn−1⋯w2w1)不过这个衡量方法有两个缺陷：

- **参数空间过大**，概率 p(wn|wn−1⋯w2w1)的参数有O(n) 个。
- **数据稀疏严重**，词同时出现的情况可能没有，组合阶数高时尤其明显。

为了解决第一个问题(参数空间过大)，我们引入**马尔科夫假设（Markov Assumption）：一个词的出现仅与它之前的若干个词有关。**

如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为 Bi-gram： p(S)=p(w1w2⋯wn)=p(w1)p(w2|w1)⋯p(wn|wn−1) 如果一个词的出现仅依赖于它前面出现的两个词，那么我们就称之为 Tri-gram： p(S)=p(w1w2⋯wn)=p(w1)p(w2|w1)⋯p(wn|wn−1wn−2)

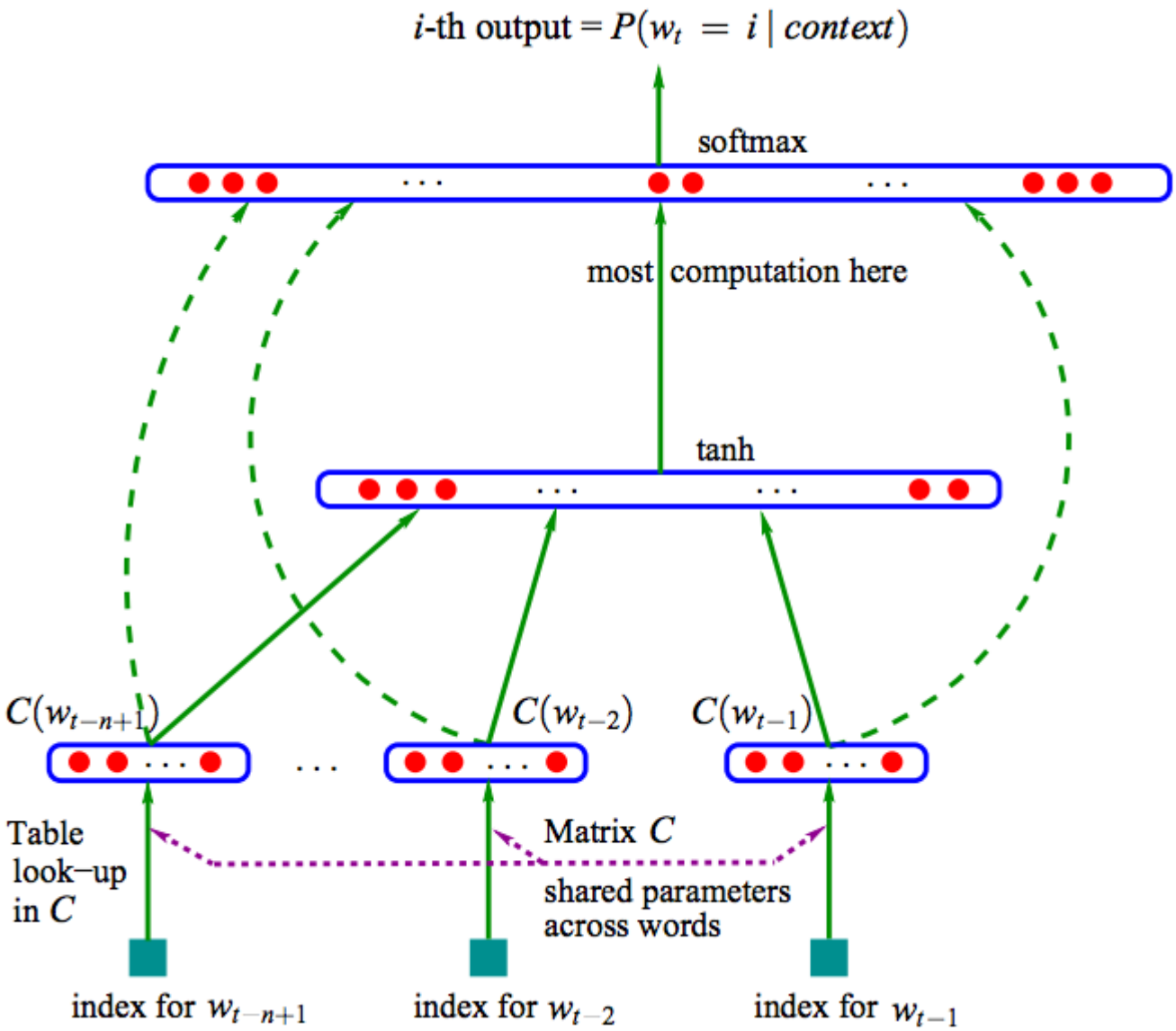
N-gram的 N可以取很高，然而现实中一般 bi-gram 和 tri-gram 就够用了。

那么，如何计算其中的每一项条件概率 p(wn|wn−1⋯w2w1)呢？答案是极大似然估计（Maximum Likelihood Estimation，MLE），说人话就是数频数： p(Wn|Wn−1)=C(Wn−1Wn) / C(Wn−1)

p(Wn|Wn−1Wn−2)=C(Wn−2Wn−1Wn)C(Wn−1Wn) p(Wn|Wn−1⋯W2W1)=C(W1W2⋯Wn)C(W1W2⋯Wn−1)

为了解决第二个问题（数据稀疏严重），因此，我们要进行**数据平滑（data Smoothing）**，数据平滑的目的有两个：一个是使所有的N-gram概率之和为1，使所有的n-gram概率都不为0。它的本质，是重新分配整个概率空间，使已经出现过的n-gram的概率降低，补充给未曾出现过的n-gram。方法有拉普拉斯平滑，即强制让所有的n-gram至少出现一次，只需要在分子和分母上分别做加法即可。

3.神经网络语言模型---CBOW是在此基础上进行改进的



来看 NNLM（神经网络语言模型） 构造 ,首先，引入了**词向量**的概念，即将词表中的词语 w 表示为一个固定长度为M向量的形式 $C(w)$ ，（m 为人工定义的词向量的长度），这样整个词表可以用 一个 $m \times |V|$ 的矩阵表示。现在找到词 wt的上下文 context(wt)，这里 Bengio 设定的**上下文 context(wt)是词wt 的前** $n-1$ **个词语。**

神经网络的输入层：把这 n-1个词语的词向量首尾相接的拼起来，形成维度为 (n−1)m**的向量来当做神经网络的输入，所以 NNLM 输入层的大小已经确定为 (n−1)m

隐层：隐层的规模就是人工指定了

神经网络的输出层：输出层的大小为 |V|(词典的大小)，因为共有 |V|个词语，所以输出层维度为 |V|， wt在词表 V中的下标对应的维度就是映射 wt的得分,而 softmax 正好可以把该得分归一化为概率。

4.CBOW (Word2Vec)

对比原始神经网络语言模型使用的数据结构是用霍夫曼树来代替隐藏层和输出层的神经元，霍夫曼树的叶子节点起到输出层神经元的作用，叶子节点的个数即为词汇表的小大。 而内部节点则起到隐藏层神经元的作用。输入是上下两个窗口。

18.马尔可夫链

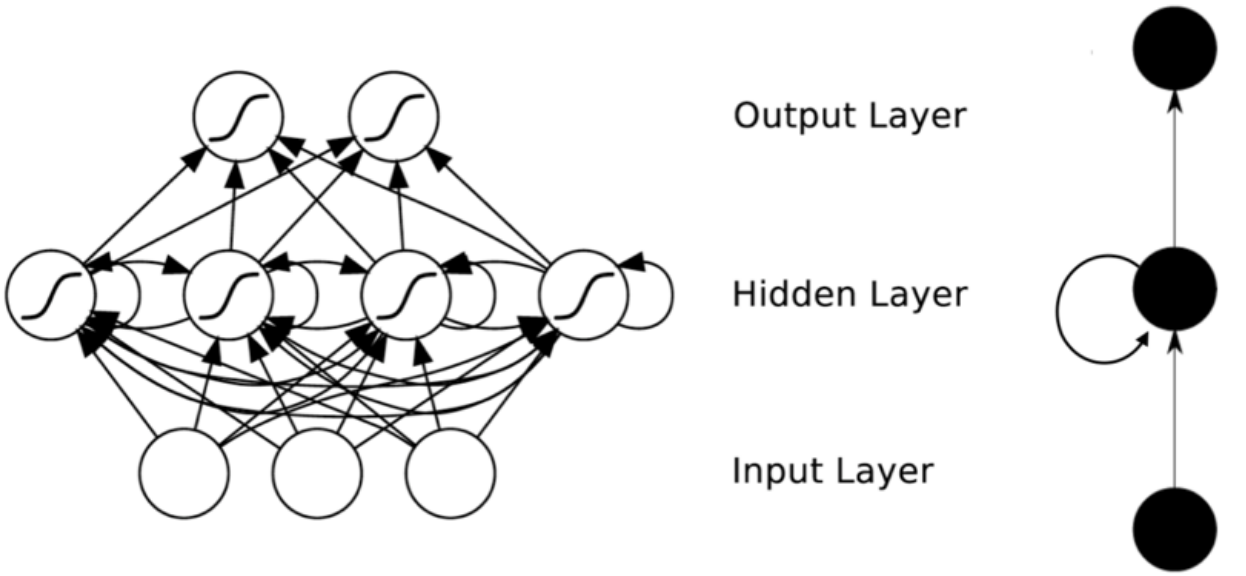
19.向量空间模型

20.RNN

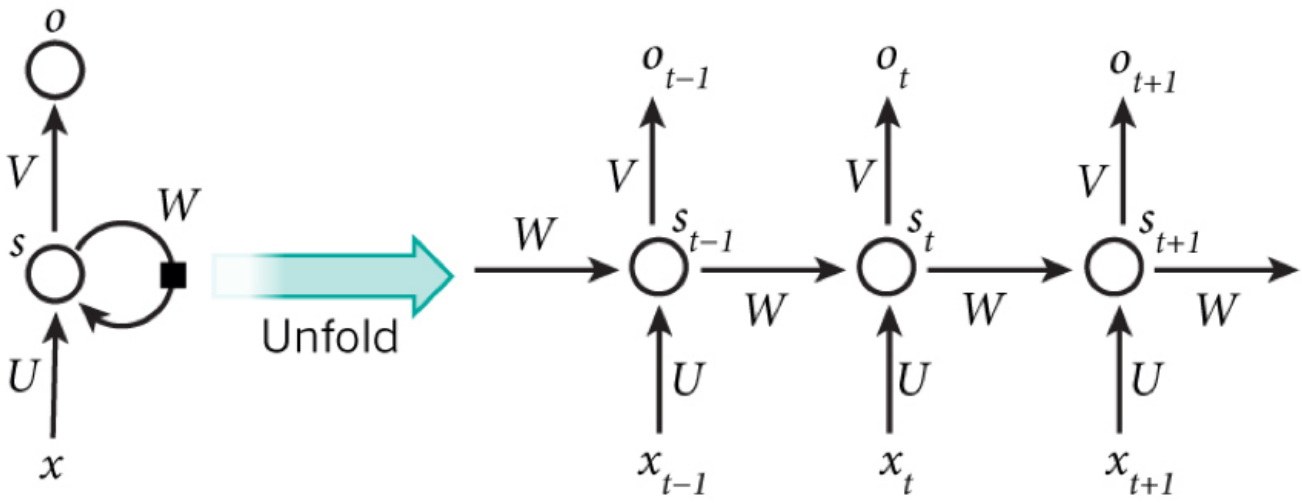
1.为什么具有记忆功能?

因为此神经元结点上一时刻隐层的状态参与到了这个时刻的计算过程中。

在RNN中，神经元的输出可以在下一个时间戳直接作用到自身，**即第i层神经元在m时刻的输入，除了（i-1）层神经元在该时刻的输出外，还包括其自身在（m-1）时刻的输出**！表示成图就是这样的：



我们可以看到在隐含层节点之间增加了互连。为了分析方便，我们常将RNN在时间上进行展开，得到如图6所示的结构



（引出LSTM的背景知识）

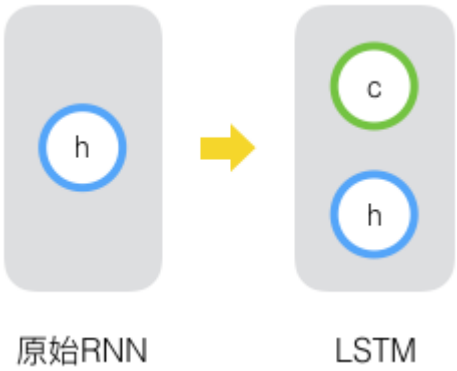
RNN可以看成一个在时间上传递的神经网络，它的深度是时间的长度！正如我们上面所说，“梯度消失”现象又要出现了，只不过这次发生在时间轴上。对于t时刻来说，它产生的梯度在时间轴上向历史传播几层之后就消失了，根本就无法影响太遥远的过去。因此，之前说“所有历史”共同作用只是理想的情况，在实际中，这种影响也就只能维持若干个时间戳。**为了解决时间上的梯度消失，机器学习领域发展出了长短时记忆单元。**LSTM

2.为什么LSTM记的时间长

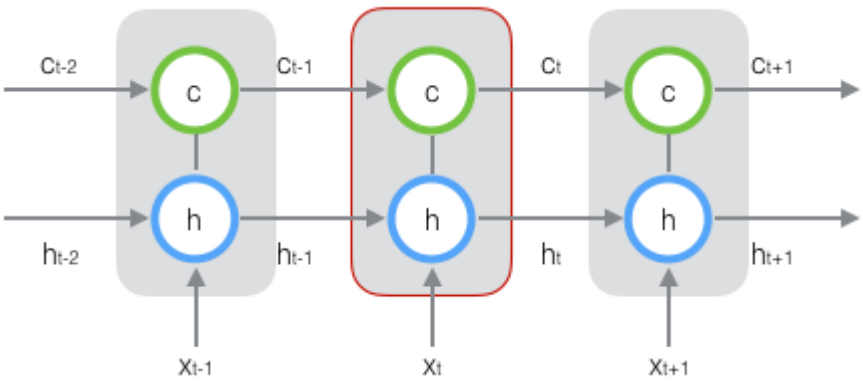
因为在rnn原始隐藏层的一个h状态上增加增加了一个状态c（单元状态），让它来保存长期的状态，并通过三个门控制长期c。这种结构使得误差向上一个状态传递时几乎没有衰减，所以权值调整的时候，对于很长时间之前的状态带来的影响和结尾状态带来的影响可以同时发挥作用，最后训练出来的模型就具有较长时间范围内的记忆功能。

具体解释：

其实，**长短时记忆网络**的思路比较简单。原始RNN的隐藏层只有一个状态，即h，它对于短期的输入非常敏感。那么，假如我们再增加一个状态，即c，让它来保存长期的状态，那么问题不就解决了么？如下图所示：

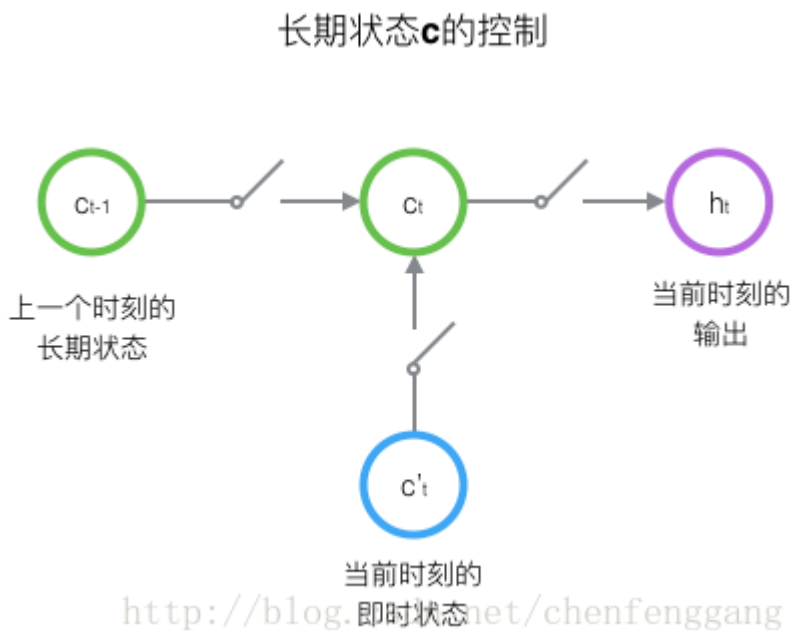


新增加的状态c，称为**单元状态(cell state)**。我们把上图按照时间维度展开：



上图仅仅是一个示意图，我们可以看出，在t时刻，LSTM的输入有三个：当前时刻网络的输入值$\mathbf{x}_t$、上一时刻LSTM的输出值$\mathbf{h}_{t-1}$、以及上一时刻的单元状态$\mathbf{c}_{t-1}$；LSTM的输出有两个：当前时刻LSTM输出值$\mathbf{h}_t$、和当前时刻的单元状态$\mathbf{c}_t$。注意$\mathbf{x}$、$\mathbf{h}$、$\mathbf{c}$都是**向量**。

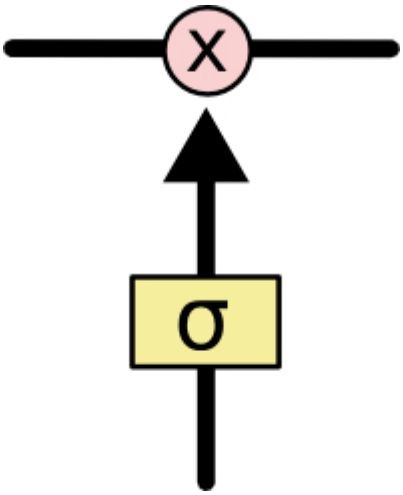
LSTM的关键，就是怎样控制长期状态c。在这里，LSTM的思路是使用三个控制开关。第一个开关，负责控制继续保存长期状态c；第二个开关，负责控制把即时状态输入到长期状态c；第三个开关，负责控制是否把长期状态c作为当前的LSTM的输出。三个开关的作用如下图所示：



接下来，我们要描述一下，输出h和单元状态c的具体计算方法。

前面描述的开关是怎样在算法中实现的呢？这就用到了门（gate）的概念。门实际上就是一层全连接层，它的输入是一个向量，输出是一个0到1之间的实数向量。假设W是门的权重向量，是偏置项，那么门可以表示为：

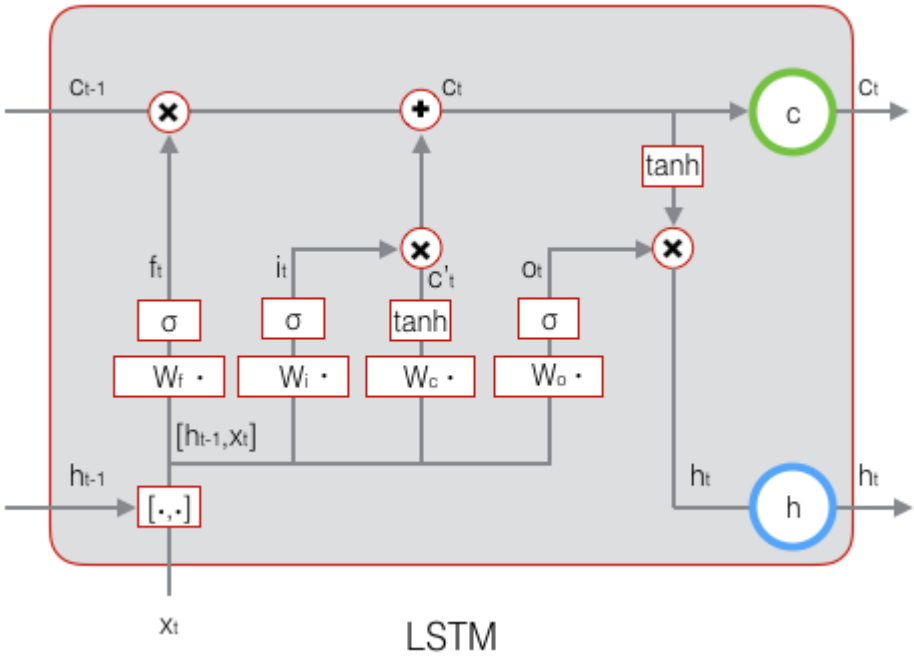
$$g(\mathbf{x}) = \sigma(W\mathbf{x} + \mathbf{b})$$



门 可以实现选择性地让信息通过，主要是通过一个 sigmoid 的神经层 和一个逐点相乘的操作来实现的。sigmoid 层输出（是一个向量）的每个元素都是一个在 0 和 1 之间的实数，表示让对应信息通过的权重（或者占比）。比如， 0 表示“不让任何信息通过”， 1 表示“让所有信息通过”。

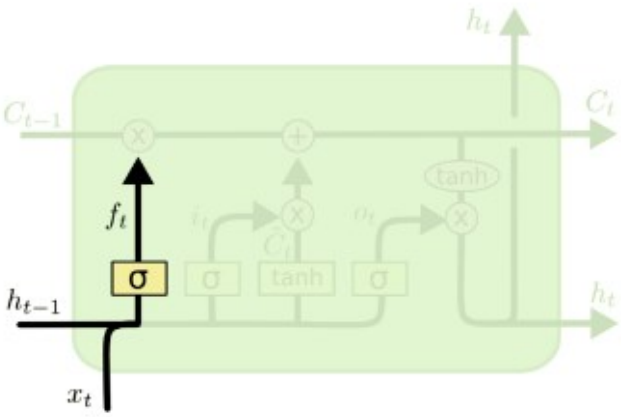
LSTM通过三个这样的本结构来实现信息的保护和控制。这三个门分别输入门、遗忘门和输出门。

LSTM用两个门来控制单元状态c的内容，一个是**遗忘门（forget gate）**，它决定了上一时刻的单元状态$\mathbf{c}_{t-1}$有多少保留到当前时刻$\mathbf{c}_t$；另一个是**输入门（input gate）**，它决定了当前时刻网络的输入$\mathbf{x}_t$有多少保存到单元状态$\mathbf{c}_t$。LSTM用**输出门（output gate）**来控制单元状态$\mathbf{c}_t$有多少输出到LSTM的当前输出值$\mathbf{h}_t$。



遗忘门：

最左边的遗忘门，遗忘们决定有多少重Ct-1 传播到Ct。



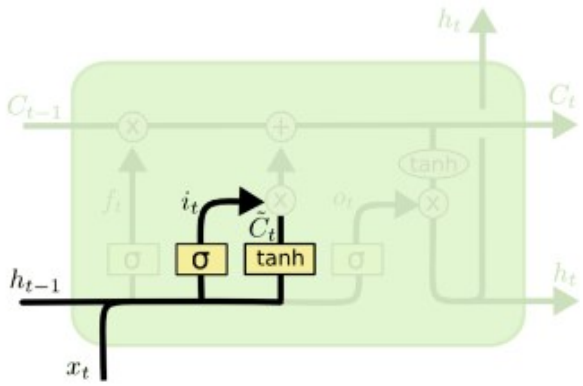
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

在我们 LSTM 中的第一步是决定我们会从细胞状态中丢弃什么信息。这个决定通过一个称为忘记门层完成。该门会读取ht−1 和xt ,输出一个在 0到 1之间的数值给每个在细胞状态ct-1中的数字。1 表示“完全保留”，0 表示“完全舍弃”。其中ht−1表示的是上一个cell的输出，xt 表示的是当前细胞的输入。σ 表示sigmod函数。

让我们回到语言模型的例子中来基于已经看到的预测下一个词。在这个问题中，细胞状态可能包含当前主语的性别，因此正确的代词可以被选择出来。当我们看到新的主语，我们希望忘记旧的主语

输入门：

输入门决定当前输入有多少保存到Ct



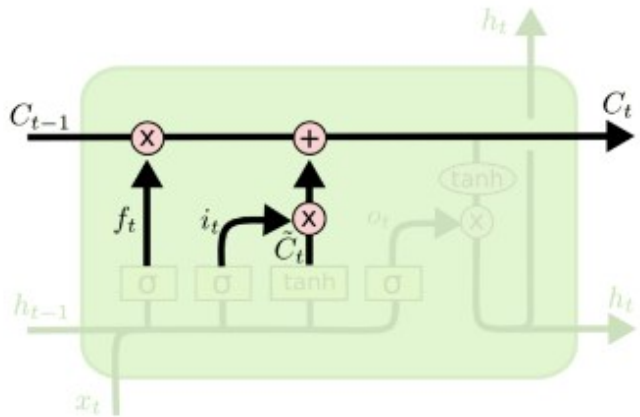
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

下一步是决定让多少新的信息加入到 cell 状态 中来。实现这个需要包括两个 步骤：首先，一个叫做“input gate layer ”的 sigmoid 层决定哪些信息需要更新；一个 tanh 层生成一个向量，也就是备选的用来更新的内容，即Ct(~)

在下一步，我们把这两部分联合起来，对 cell 的状态进行一个更新。

新的保存信息：

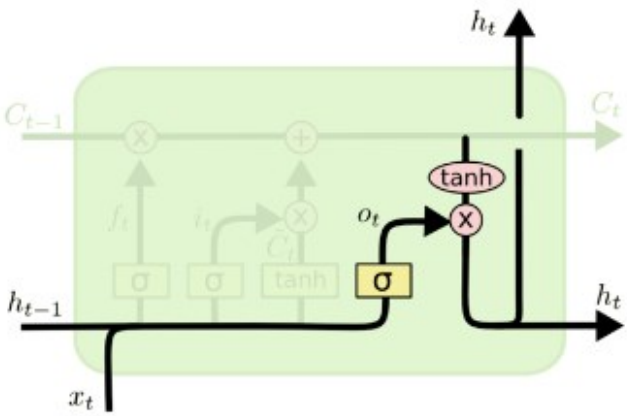
所以最终的保存信息为：符号*表示**按元素乘**



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

现在是更新旧细胞状态的时间了，Ct−1 更新为Ct 。前面的步骤已经决定了将会做什么，我们现在就是实际去完成。我们把旧状态与ft相乘，丢弃掉我们确定需要丢弃的信息。接着加上it∗C˜t 。这就是新的候选值，根据我们决定更新每个状态的程度进行变化。在语言模型的例子中，这就是我们实际根据前面确定的目标，丢弃旧代词的性别信息并添加新的信息的地方。

输出：



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

最终，我们需要确定输出什么值。这个输出将会基于我们的细胞状态，但是也是一个过滤后的版本。首先，我们运行一个 sigmoid 层来确定细胞状态的哪个部分将输出出去。接着，我们把细胞状态通过 tanh 进行处理（得到一个在 -1 到 1 之间的值）并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

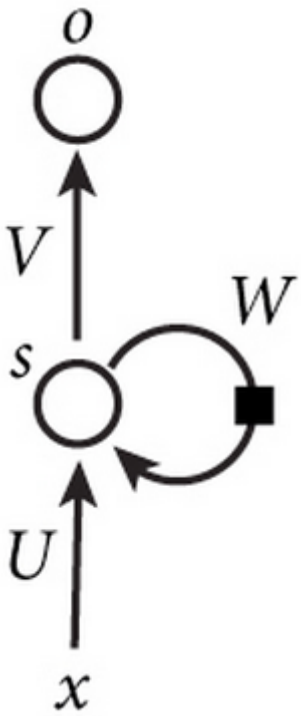
在语言模型的例子中，因为他就看到了一个 代词，可能需要输出与一个 动词 相关的信息。例如，可能输出是否代词是单数还是负数，这样如果是动词的话，我们也知道动词需要进行的词形变化。

3.RNN特点

RNN（Recurrent Neural Network）是一类**用于处理序列数据的神经网络**【1】。首先我们要明确什么是序列数据，摘取百度百科词条：时间序列数据是指在不同时间点上收集到的数据，这类数据反映了某一事物、现象等随时间的变化状态或程度。这是时间序列数据的定义，当然这里也可以不是时间，比如文字序列，**但总归序列数据有一个特点——后面的数据跟前面的数据有关系。**

我们从基础的神经网络中知道，神经网络包含输入层、隐层、输出层，通过激活函数控制输出，层与层之间通过权值连接。激活函数是事先确定好的，那么神经网络模型通过训练“学“到的东西就蕴含在“权值“中。 基础的神经网络只在层与层之间建立了权连接，**RNN最大的不同之处就是在层之间的神经元之间也建立的权连接。【2】**只要知道上一时刻的隐藏状态ht−1与当前时刻的输入xt，就可以计算当前时刻的隐藏状态ht。

权值共享【3】在RNN中U、V、W的参数都是共享的，也就是只需要关注每一步都在做相同的事情，只是输入不同，这样来降低参数个数和计算量。



LSTM在RNN基础上解决了其时间序列上梯度消失的问题【4】

21.HMM

22.CRF