第一周机器学习

> 本文公式显示需要使用Mathjax，然后令人悲伤的是github不支持Mathjax 您可以将这篇md文件pull下来，使用您本地的markdown解析器解析 没有必要在公示显示上浪费时间，您也可以下载我本地生成的html用浏览器打开即可 或者您也可以下载我上传到github上的pdf *Mathjax开源项目地址*

绪论

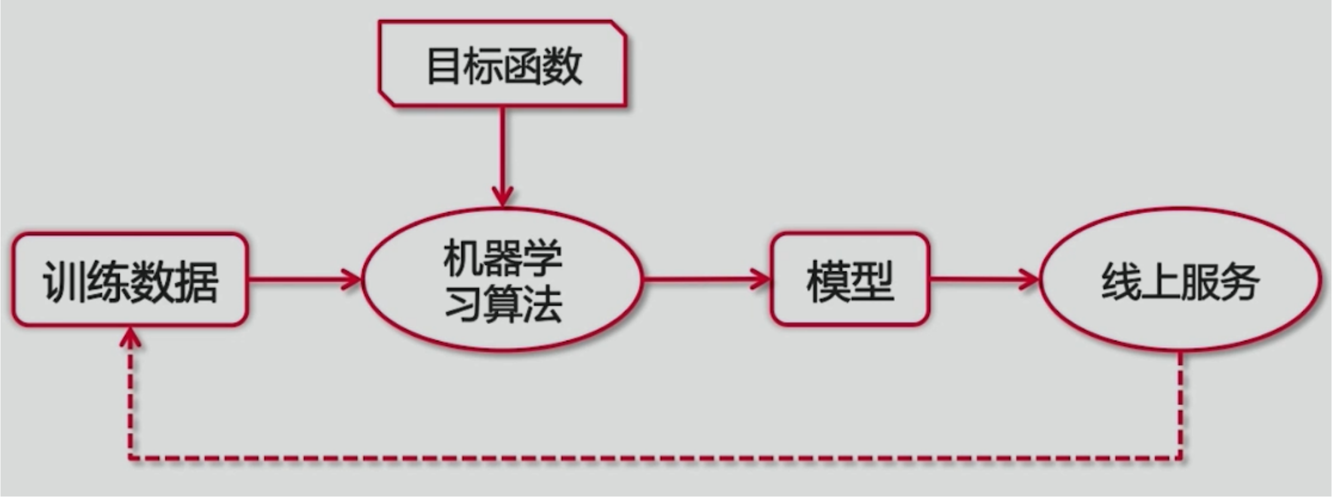
机器学习简介

机器学习是一种将无序数据转换为价值的方法。
机器学习的价值-从数据中抽取规律，并用来预测未来

机器学习应用举例

- 分类问题-图像识别、垃圾邮件识别
- 回归问题-股价预测、房价预测
- 排序问题-点击率预估、推荐
- 生成问题-图像生成、图像风格转换、图像文字描述生成

机器学习的应用流程



机器学习岗位职责

- 数据处理(采集+去噪)
- 模型训练(特征+模型)
- 模型评估与优化(MSE、F1-score、AUC+调参)
- 模型应用(A/B测试)

线性回归(Linear regression)

cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(y^{(i)} - \theta^T x^{(i)})^2$$

来源于假设误差 ε_i 服从正态分布，然后对参数θ进行极大似然估计，经过运算后得出$J(\theta)$取最小时，似然函数最大，从而推出这个式子。
此外，对这个$J(\theta)$求偏导，令其偏导数为0（这里涉及到矩阵偏导数计算），即可得到正规方程(normal equation)。

梯度下降法

分类

- mini-batch
- batch
- random
- SGD(动量梯度下降，有助于解决局部最值和鞍点问题)

Code

参考代码，自己就一些细节进行优化
(https://www.cnblogs.com/focusonepoint/p/6394339.html)

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np
from numpy import linalg



def gradientDescent(x, y, theta, m, alpha, maxIteration):
    '''
    使用批处理梯度下降算法计算theta
    '''
    # 得到x的转置
    # 即 x的第一行为x1 第二行为x2 第三行全部初始化为1
    xTrains = x.transpose()
    # theta 是一个列向量
    for i in xrange(0,maxIteration):
        # x矩阵(10*3)与theta(3*1)矩阵相乘
        # hypothesis(i) = x1(i)*theta1(i) + x2(i)*theta2(2) + 1*theta0(i)
        hypothesis = np.dot(x, theta)
        # 作差
        loss = hypothesis - y
        # 当loss的范数在我们的误差允许范围内 就停止循环
        if (linalg.norm(loss) < 1e-5):
            break
        # xTrains (3*10) * loss(10*1) = gradient(3*1)
        # 计算代价函数
        gradient = (1.0/m) * np.dot(xTrains, loss)
        theta = theta - alpha * gradient
    print('the number of iteration is %d' % i);
```

```
    return theta


# define the prepared 训练集
# the meaning of column : x1,x2,y
dataSet = np.array([
    [1.1,1.5,2.5],
    [1.3,1.9,3.2],
    [1.5,2.3,3.9],
    [1.7,2.7,4.6],
    [1.9,3.1,5.3],
    [2.1,3.5,6.0],
    [2.3,3.9,6.7],
    [2.5,4.3,7.4],
    [2.7,4.7,8.1],
    [2.9,5.1,8.8],
])


# print(dataSet)
m,n = np.shape(dataSet)
# print(m,n)
trainData = np.ones((m,n))
# 截取dataSet的前N-1列
trainData[:,:-1] = dataSet[:,:-1]
# 获取dataSet的最后一列
trainLabel = dataSet[:,-1]

# print(m,n)
theta = np.ones(n)
# print(theta)
alpha = 0.001

# the max time of iteration 这个值定义的尽量大(考虑计算机的性能)
maxIteration = 10000000
theta = gradientDescent(trainData, trainLabel, theta, m, alpha, maxIteration)
print('thec value of theta is:')
print(np.round(theta,2))


# a test for the algorithm
x = np.array([
    [3.1, 5.5],
    [3.3, 5.9],
    [3.5, 6.3],
    [3.7, 6.7],
    [3.9, 7.1]
])


# define a predict function used to test
def predict(x,theta):
    m, n = np.shape(x)
```

```
    xTest = np.ones((m, n+1))
    xTest[:, :-1] = x
    yPre = np.dot(xTest,theta)
    return yPre

print('the predicted value is')
yP = predict(x, theta)
print(np.round(yP,2))


```

运行结果

```
the number of iteration is 114575
thec value of theta is:
[ 0.71  1.39 -0.38]
the predicted value is
[ 9.5 10.2 10.9 11.6 12.3]
[Finished in 2.2s]
```

优化技巧

- Feature Scaling（特征缩放）
    - 归一化
        1. 线性归一化
            - $ {x}' = \frac{x - \min(x)}{\max(x) - \min(x)} $
        2. 标准差归一化
            - $ x^* = \frac{x- \overline x}{s} $
        3. 非线性归一化
- 多项式回归
    - $ h_\theta(x) = \theta_0 + \theta_1x + \theta_2x^2 $
    - $ h_\theta(x) = \theta_0 + \theta_1x + \theta_2x^2 + \theta_3x^3 $
    - $ h_\theta(x) = \theta_0 + \theta_1x + \theta_2\sqrt{x} $
    - 上面的举例只是为了说明，x_i的取值可以不是x的一次多项式，但是这里要注意的是特征缩放在这里显得尤为重要
- α选取技巧
    - 如果J（θ)的值随着θ的取值单调递增或者出现震荡，那么α应该选的小一点

Normal Equation（正规方程法）

思想

$$
\begin{aligned}
J_\theta(x) &= \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x) - y)^2 \\
&= a\theta^2 + b\theta + c
\end{aligned}
$$

微积分思想：求导后令导数为零解方程可以求出极值点θ
对于θ是一个n维向量的情况，可以利用多元函数取极值的必要条件，即偏导数为0

结论

$$\theta = (X^T X)^{-1} X^T y$$

Note

1. No need to do feature scaling
2. 只适用于线性模型，不适合逻辑回归模型等其他模型
3. the pseudo inverse of matrix
    - redundant features (x中存在线性相关的量)
    - too many features (eg. m <= n 数据个数小于特征参数)

Code

这里我使用上一个梯度下降法的例子作为对比,采用相同的数据对比运行结果

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

import numpy as np

def normalEqation(x, y):
    '''
    使用正规方程法算法计算theta
    '''
    # 得到x的转置
    xTrains = x.transpose()
    m,n = np.shape(x)
    # theta 为 n维列向量
    theta = np.linalg.pinv(np.dot(xTrains,x))
    theta = np.dot(theta,xTrains)
    theta = np.dot(theta,y)
    return theta


# define the prepared 训练集
# the meaning of column : x1,x2,y
dataSet = np.array([
    [1.1,1.5,2.5],
    [1.3,1.9,3.2],
    [1.5,2.3,3.9],
    [1.7,2.7,4.6],
    [1.9,3.1,5.3],
    [2.1,3.5,6.0],
    [2.3,3.9,6.7],
    [2.5,4.3,7.4],
    [2.7,4.7,8.1],
    [2.9,5.1,8.8],
```

```
])


# print(dataSet)
m,n = np.shape(dataSet)
# print(m,n)
trainData = np.ones((m,n))
# 截取dataSet的前N-1列
trainData[:,:-1] = dataSet[:,:-1]
# 获取dataSet的最后一列
trainLabel = dataSet[:,-1]



theta = normalEqation(trainData, trainLabel)
print('thec value of theta is:')
print(np.round(theta,2))


# a test for the algorithm
x = np.array([
    [3.1, 5.5],
    [3.3, 5.9],
    [3.5, 6.3],
    [3.7, 6.7],
    [3.9, 7.1]
])


# define a predict function used to test
def predict(x,theta):
    m, n = np.shape(x)
    xTest = np.ones((m, n+1))
    xTest[:, :-1] = x
    yPre = np.dot(xTest,theta)
    return yPre

print('the predicted value is')
yP = predict(x, theta)
print(np.round(yP,2))
```

运行结果:

```
thec value of theta is:
[ 0.61  1.45 -0.34]
the predicted value is
[ 9.5 10.2 10.9 11.6 12.3]
[Finished in 0.2s]
```

由此可以知道，在特征矩阵维度不是太大情况下，对于线性回归模型，normal equation 是一个优先选用的方法。

Logistic Regression

logistic function

由于线性回归的假设函数不再适用于分类问题，因此我们需要一个函数来应用于分类问题的拟合。 一般来说，回归不用在分类问题上，因为回归是连续型模型，而且受噪声影响比较大。如果非要应用进入，可以使用logistic回归。

我们可以使用logistic regression解决分类问题，Logistic回归是二分类任务的首选方法，下面讨论二分类的问题。

$$h_\theta(x) = g(\theta^T x)$$

logistic function(sigmoid function):

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

这里

$$h_\theta(x) = P(y = 1|x; \theta)$$

含义是在x已知条件下，给定参数θ，事件*y=1*发生的概率

logistic回归本质上是线性回归，只是在特征到结果的映射中加入了一层函数映射，即先把特征线性求和，然后使用函数g(z)将最为假设函数来预测。g(z)可以将连续值映射到0和1上。

对g(z)的解释：将任意的输入映射到[0,1]区间上，我们在线性回归中可以得到一个预测值，再将该值映射到Sigmoid函数，这样我们就实现了由值到概率的转换，也就是分类任务。

Note：

> 当y等于1时，假设函数计算出的概率应该大于0.5，即θ的转置乘以x需要大于等于0 当y等于0时，假设函数计算出的概率应该小于0.5，即θ的转置乘以x需要小于0 另外需要注意的是阈值0.5在一些情况下是可以改变的，从而获得我们所希望的特征

cost function

$$J_\theta(x) = \frac{1}{m} \sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$$

这里我们将 cost function 定义为

$$cost(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & y = 1 \\ -\log(1 - h_\theta(x)) & y = 0 \end{cases}$$

例如，y = 1时 h_\theta(x) \rightarrow 1 ,cost = 0 表示误差很小。 此时，若 h_\theta(x) \rightarrow 0 , cost \rightarrow \infty 表示误差很大

Simple Classification(简单分类算法)

> Note：y=0 or 1

这里对cost function进行优化，表示为：

$$cost(h_\theta(x), y) = -y\log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x))$$

> 这里的cost function 实际上也是由对θ的极大似然估计推导出来的。

by the way,remind:

$$J_\theta(x) = \frac{1}{m}\sum_{i=1}^{m} cost(h_\theta(x^{(i)}), y^{(i)})$$

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

ok,接着我们对 J_\theta(x) 计算偏微分

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial\theta_j}cost(h_\theta(x^{(i)}), y^{(i)})$$

$$= \frac{1}{m}\sum_{i=1}^{m}(-y\frac{1}{h_\theta(x)}\frac{\partial}{\partial\theta_j}h_\theta(x) - (1-y)\frac{1}{1-h_\theta(x)}(-1)\frac{\partial}{\partial\theta_j}h_\theta(x))$$

其中

$$\frac{1}{h_\theta(x)} = 1 + e^{-\theta^T x}$$

$$\frac{1}{1-h_\theta(x)} = \frac{1}{1-\frac{1}{1+e^{-\theta^T x}}}$$

$$= \frac{1+e^{-\theta^T x}}{e^{-\theta^T x}}$$

$$= 1 + e^{\theta^T x}$$

$$\frac{\partial}{\partial\theta_j}h_\theta(x) = -(1+e^{-\theta^T x})^{-2}(e^{-\theta^T x})(-x_j)$$

$$= (h_\theta(x))^2 x_j e^{-\theta^T x}$$

将上面式子代入 \frac{\partial}{\partial\theta_j}J(\theta) 得

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(-y(1+e^{-\theta^T x})(h_\theta(x))^2 x_j e^{-\theta^T x} +$$

$$(1-y)(1+e^{\theta^T x})(h_\theta(x))^2 x_j e^{-\theta^T x})$$

$$= \frac{1}{m}\sum_{i=1}^{m} -y h_\theta(x) x_j e^{-\theta^T x} + (1-y) h_\theta(x) x_j$$

$$= \frac{1}{m}\sum_{i=1}^{m} -y(1-h_\theta(x))x_j + h_\theta(x)x_j - y h_\theta(x) x_j$$

$$= \frac{1}{m}\sum_{i=1}^{m} -y x_j + h_\theta(x) x_j$$

$$= \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x) - y)x_j$$

这里我们推出一个重要的结论

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j$$

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta) = \theta_j - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j$$

> **Note:** 这里的 h_\theta(x^{(i)}) 与 线性回归模型中的 h_\theta(x^{(i)}) 定义不一样，尽管计算出来的 \frac{\partial}{\partial\theta_j}J(\theta) 形式相同

Code

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-

'''
本例程是根据学生两门课的成绩判断是否录取
'''


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

dataStr = '''
34.62365962451697,78.0246928153624,0
30.28671076822607,43.89499752400101,0
```

```
35.84740876993872,72.90219802708364,0
60.18259938620976,86.30855209546826,1
79.0327360507101,75.3443764369103,1
45.08327747668339,56.3163717815305,0
61.10666453684766,96.51142588489624,1
75.02474556738889,46.55401354116538,1
76.0987867022625,87.42056971926803,1
84.43281996120035,43.53339331072109,1
95.86155507093572,38.22527805795094,0
75.01365838958247,30.60326323428011,0
82.30705337399482,76.48196330235604,1
69.36458875970939,97.71869196188608,1
39.53833914367223,76.03681085115882,0
53.9710521485623,89.20735013750205,1
69.07014406283025,52.74046973016765,1
67.94685547711617,46.67857410673128,0
70.66150955499435,92.92713789364831,1
76.97878372747498,47.57596364975532,1
67.37202754570876,42.83843832029179,0
89.67677575072079,65.79936592745237,1
50.534788289883,48.85581152764205,0
34.21206097786789,44.20952859866288,0
77.9240914545704,68.9723599933059,1
62.27101367004632,69.95445795447587,1
80.1901807509566,44.82162893218353,1
93.114388797442,38.80067033713209,0
61.83020602312595,50.25610789244621,0
38.78580379679423,64.99568095539578,0
61.379289447425,72.80788731317097,1
85.40451939411645,57.05198397627122,1
52.10797973193984,63.12762376881715,0
52.04540476831827,69.43286012045222,1
40.23689373545111,71.16774802184875,0
54.63510555424817,52.21388588061123,0
33.91550010906887,98.86943574220611,0
64.17698887494485,80.90806058670817,1
74.78925295941542,41.57341522824434,0
34.1836400264419,75.2377203360134,0
83.90239366249155,56.30804621605327,1
51.54772026906181,46.85629026349976,0
94.44336776917852,65.56892160559052,1
82.36875375713919,40.61825515970618,0
51.04775177128865,45.82270145776001,0
62.22267576120188,52.06099194836679,0
77.19303492601364,70.45820000180959,1
97.77159928000232,86.7278223300282,1
62.07306379667647,96.76882412413983,1
91.56497449807442,88.69629254546599,1
79.94481794066932,74.16311935043758,1
99.2725269292572,60.99903099844988,1
90.54671411399852,43.39060180650027,1
34.52451385320009,60.39634245837173,0
50.2864961189907,49.80453881323059,0
```

```
49.58667721632031,59.80895099453265,0
97.64563396007767,68.86157272420604,1
32.57720016809309,95.59854761387875,0
74.24869136721598,69.82457122657193,1
71.79646205863379,78.45356224515052,1
75.3956114656803,85.75993667331619,1
35.28611281526193,47.02051394723416,0
56.25381749711624,39.26147251058019,0
30.05882244669796,49.59297386723685,0
44.66826172480893,66.45008614558913,0
66.56089447242954,41.09209807936973,0
40.45755098375164,97.53518548909936,1
49.07256321908844,51.88321182073966,0
80.27957401466998,92.11606081344084,1
66.74671856944039,60.99139402740988,1
32.72283304060323,43.30717306430063,0
64.0393204150601,78.03168802018232,1
72.34649422579923,96.22759296761404,1
60.45788573918959,73.09499809758037,1
58.84095621726802,75.85844831279042,1
99.82785779692128,72.36925193383885,1
47.26426910848174,88.47586499559782,1
50.45815980285988,75.80985952982456,1
60.45555629271532,42.50840943572217,0
82.22666157785568,42.71987853716458,0
88.9138964166533,69.80378889835472,1
94.83450672430196,45.69430680250754,1
67.31925746917527,66.58935317747915,1
57.23870631569862,59.51428198012956,1
80.36675600171273,90.96014789746954,1
68.46852178591112,85.59430710452014,1
42.0754545384731,78.84478600148043,0
75.47770200533905,90.42453899753964,1
78.63542434898018,96.64742716885644,1
52.34800398794107,60.76950525602592,0
94.09433112516793,77.15910509073893,1
90.44855097096364,87.50879176484702,1
55.48216114069585,35.57070347228866,0
74.49269241843041,84.84513684930135,1
89.84580670720979,45.35828361091658,1
83.48916274498238,48.38028579728175,1
42.2617008099817,87.10385094025457,1
99.31500880510394,68.77540947206617,1
55.34001756003703,64.9319380069486,1
74.77589300092767,89.52981289513276,1
'''

tmpdataList = dataStr.split()
dataList = []
for data in tmpdataList:
    data = data.split(',')
    dataList.append(data)
del tmpdataList
```

```python

# define the prepared 训练集
# the meaning of column : x1,x2,y
dataSet = np.array(dataList)
dataSet = dataSet.astype(np.float64)

def shuffleData(dataSet):
    # 打乱数据
    np.random.shuffle(dataSet)
    m,n = np.shape(dataSet)

    trainData = np.ones((m,n))
    trainData[:,:-1] = dataSet[:,:-1]
    # 获取dataSet的最后一列 并 强制类型转换
    trainLabel = dataSet[:,-1]
    return trainData,trainLabel


# 这里我们使用matplot先看一下数据
negativeData = dataSet[dataSet[:,-1] == 0.0]
positiveData = dataSet[dataSet[:,-1] == 1.0]
trainLabel = dataSet[:,-1].astype(np.float64)


fig,ax = plt.subplots(figsize=(10,5))
ax.scatter(positiveData[:,0],positiveData[:,1],s = 30,c = 'b',marker = 'o',label = 'Admited')
ax.scatter(negativeData[:,0],negativeData[:,1],s = 30,c = 'r',marker = 'x',label = 'Not Admited')
ax.legend()
ax.set_xlabel('Exam 1 Score')
ax.set_ylabel('Exam 2 Score')
plt.show()

# 下面是逻辑回归算法
def sigmoid(z):
    return (1.0 / (1.0 + np.exp(-z)))


def model(X,theta):
    return sigmoid(np.dot(X,theta))

# x2 x1 x0
# res = model(trainData,theta)
def cost_function(X,y,theta):
    h_x = model(X,theta)
    left = -y*np.log(h_x)
    right = (1-y)*np.log(1-h_x)
    return np.sum(left - right) / (len(X))

# x = cost_function(trainData,trainLabel,theta)
def gradient(X,y,theta):
    grad = np.zeros(theta.shape)
```

```
    error = (model(X,theta) - y).ravel()
    for j in xrange(len(theta.ravel())):
        term = np.multiply(error, X[:,j])
        grad[j] = np.sum(term) / len(X)
    return grad
# 3种梯度下降方法  1.批处理 2.小批处理 3.随机处理
# 数据量较小，直接批处理即可
def batchGradientDescent(dataSet, alpha, maxIteration, thresh):
    X,y = shuffleData(dataSet)
    m,n = np.shape(X)
    k = 1.0 / m
    theta = np.zeros((n,))

    trainX = X.transpose()
    for i in xrange(0,maxIteration):
        error = model(X, theta) - y
        _gradient = k * np.dot(trainX, error)
        if (np.linalg.norm(_gradient) < thresh[0]):
            print('hit thresh1')
            break
        # print(gradient(X,y,theta))
        # print(_gradient)
        cost1 = cost_function(X,y,theta)
        theta = theta - alpha * _gradient
        cost2 = cost_function(X,y,theta)
        if  abs(cost2 - cost1) < thresh[1]:
            print('hit thresh2')
            break
        # print(theta)
    print('the number of iteration is %d' % (i+1))
    # print(error)
    return theta

# theta = batchGradientDescent(dataSet,alpha =0.001,maxIteration = 1000000,thresh = (1e-6,1e-6))
# print(theta)

'''
hit thresh2
the number of iteration is 109902
[ 0.04771429  0.04072397 -5.13364014]

这个数据说明当迭代次数为110000次时，cost function下降就跟缓慢了
'''

theta = batchGradientDescent(dataSet,alpha =0.001,maxIteration = 1000000,thresh = (0.05,1e-6))
print(theta)
# theta = batchGradientDescent(dataSet,alpha =0.001,maxIteration = 1000000,thresh = (1e-6,1e-6))
# print(theta)

'''
```

```
hit thresh1
the number of iteration is 40046
[ 0.02721656  0.01899417 -2.37028409]
[Finished in 8.2s]
按照梯度下降停止大概需要40000次迭代
'''
```

> 这里实际上，如果数据经过预处理以及miniBatch后获得的数据精度比较高

Advanced optimization

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS 后面三种算法不需要给出学习率 \alpha ，且运算速度较快，但是算法较为复杂，选修。

多类别处理

遇到y的取值不仅仅是0,1情况时，可以将一类与其余类化为两种模型，然后用划分两类的分类算法计算出h(x)，最后每一类都对应一个h(x)，训练出模型后，判断 \max h_\theta(x) 对应的类即为最后输出。

关于机器学习的一些概念补充

下采样与上采样

下采样，对于一个不均衡的数据，让目标值(如0和1分类)中的样本数据量相同，且以数据量少的一方的样本数量为准。
上采样就是以数据量多的一方的样本数量为标准，把样本数量较少的类的样本数量生成和样本数量多的一方相同，称为上采样。

交叉验证

交叉验证的基本思想是把在某种意义下将原始数据(dataset)进行分组,一部分做为训练集(train set),另一部分做为验证集(validation set or test set),首先用训练集对分类器进行训练,再利用验证集来测试训练得到的模型(model),以此来做为评价分类器的性能指标。

二分类模型评估方法

以正例（恐怖分子）的识别为例子

真正例（True Positive，TP）：预测值和真实值都为1 假正例（False Positive，FP）：预测值为1，真实值为0(去真)
真负例（True Negative，TN）:预测值与真实值都为0 假负例（False Negative，FN）：预测值为0，真实值为1(存伪)

召回率（也叫查全率）

$$召回率 = \frac{真正例}{真正例 + 假负例}$$

> 正确判为恐怖分子占实际所有恐怖分子的比例。 在某些情况中，我们也许需要以牺牲另一个指标为代价来最大化精度或者召回率。 比如检测癌症

精确度(precision,也叫查准率)

$$精确度() = \frac{真正例}{真正例 + 假正例}$$

> 在所有判为恐怖分子中，真正的恐怖分子的比例。

准确率（accuracy）

$$accuracy = \frac{TP+TN}{P+N} = \frac{TP+TN}{TP+TN+FP+FN}$$

正则化(Regularization)

欠拟合(underfitting)和过拟合(overfitting)

How to addressing overfitting

1. Reduce number of featrues
2. Regularization
   - keep all the feature,but reduce magnitude/values of feature.
     it works well when we have a lot of features,each of which contributs a bit to predicting y.
3. Regularization used in linear Regression
   - J(\theta) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})^2+\lambda\sum_{j=1}^{n}\theta_j^2]

> λ 称为 regularization parameter Note:加上 \theta^2 是一种形式，有时也可以选择加上 |\theta|

3.1 Gradient descent

$$
\begin{aligned}
&Repeat\\
&\quad\{\\
&\quad\quad \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_0^{(i)}\\
&\quad\quad \theta_j := \theta_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j]\\
&\quad\quad (j = 1, 2, , \dots, n)\\
&\quad\}
\end{aligned}
$$

- 其中

$$\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

3.2 Normal Equation

$$\theta = (X^TX + \lambda\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}_{n\times n})^{-1}X^Ty$$

4. Regularization used in logistic Regression

Neural networks(神经网络)

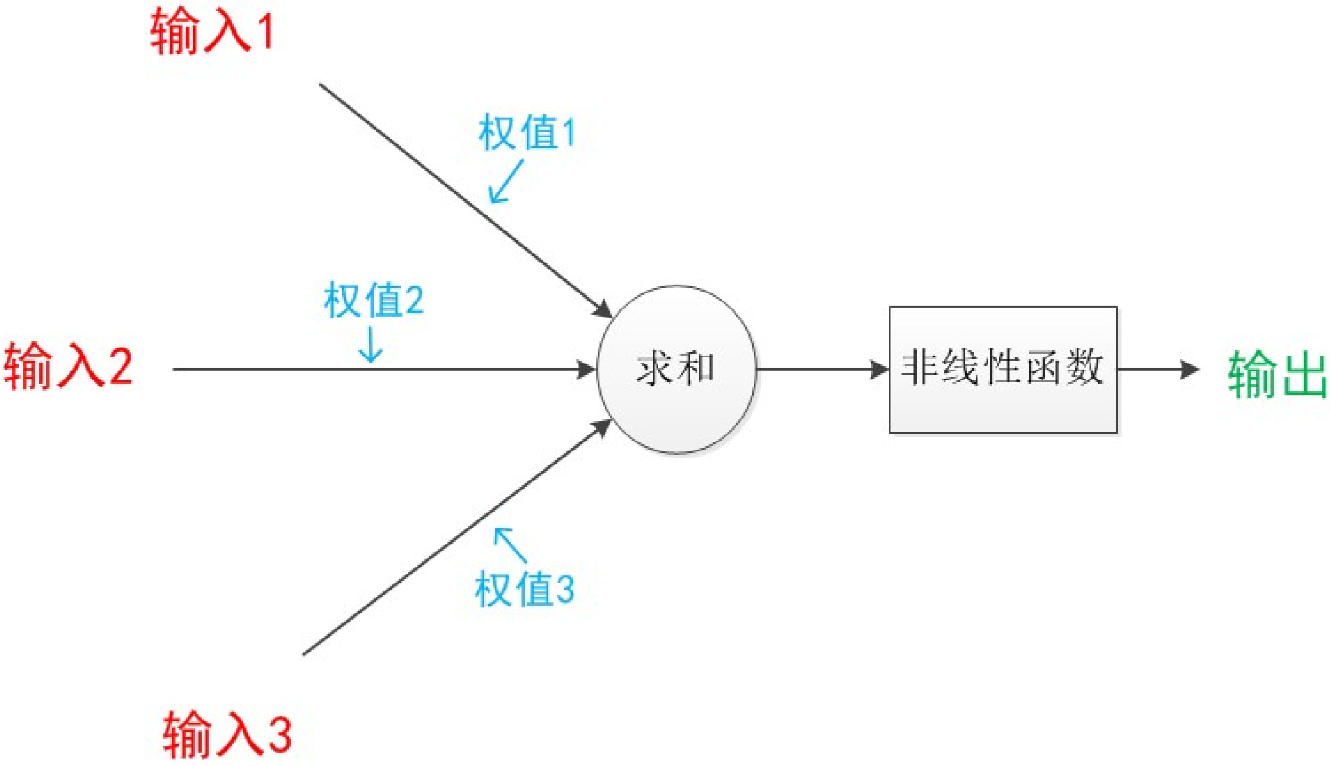
Typeical Application（应用领域）

Example	Principle
Ad.userinfo	Online Advertising(Stardard NN)
Image	Phototapping(CNN convoliutional nerual network)
Audio	Speech recognation(RNN recurrent nerual network)
Machine translation	RNN
Autonomous driving	hybrid neural network + custum nerual network

Concepts

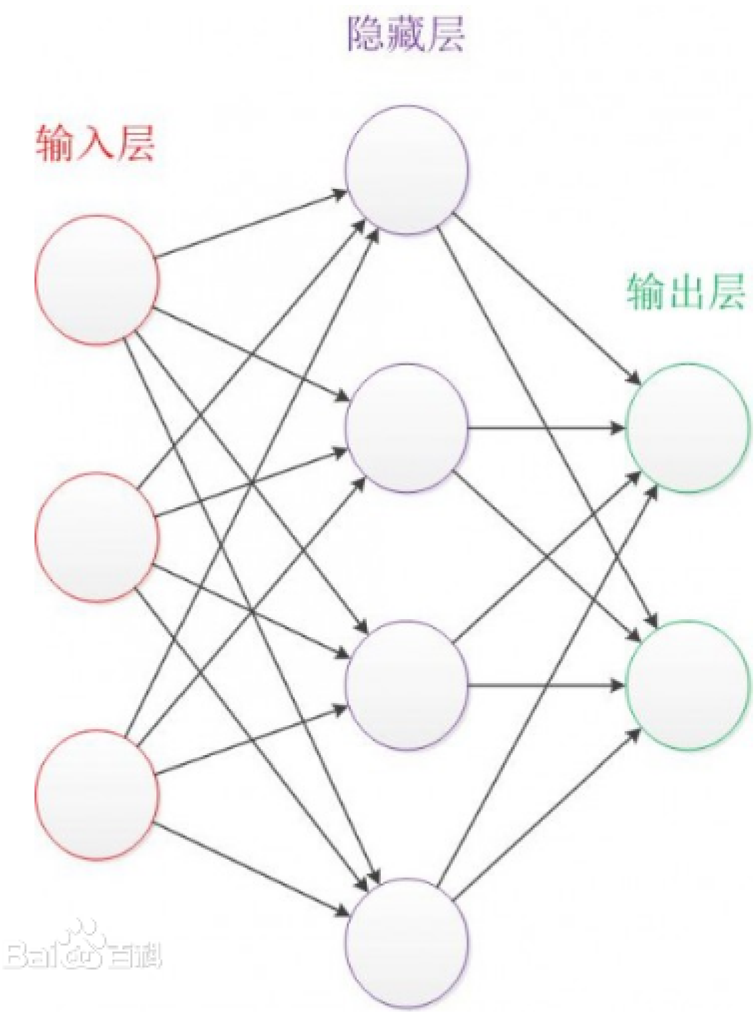
- Structured Data
  - Data in the database(have rows and cols)
  - 一般是离散的、有组织结构的
- Unstructured Data
  - Audio、Image、Text
  - 一般是连续的、无组织结构的

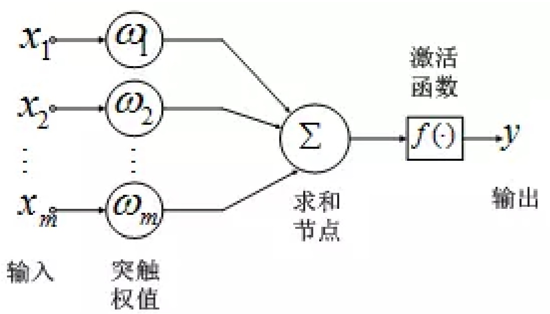
神经元



Layer

神经网络是分层的
一般来说
Layer 1: Input Layer
Layer 2~N-1: Hidden Layer **Layer N**: Output Layer





Note:

> 如果没有隐藏层，只有输入层和输出层，那么我们将这种神经网络称为“感知器”（Perceptron） 在“感知器”中，有两个层次。分别是输入层和输出层。输入层里的“输入单元”只负责传输数据，不做计算。输出层里的“输出单元”则需要对前面一层的输入进行计算。 感知器只能做简单的线性分类任务,对XOR（异或）这样的简单分类任务无法解决。

Definations:

1. a_i^{(j)} :"activation" of unit i in layer j
2. \theta^{(j)} :matrix of weights controlling function mapping from layer_j to layer_{j+1}

Examples:

1. 输出仅有一个的神经网络

$$
\begin{aligned}
a_1^{(2)} &= g(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3) \\
a_2^{(2)} &= g(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3) \\
a_3^{(2)} &= g(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3) \\
h_\theta(x) = a_1^{(3)} &= g(\theta_{10}^{(2)}a_0 + \theta_{11}^{(2)}a_1 + \theta_{12}^{(2)}a_2 + \theta_{13}^{(2)}a_3)
\end{aligned}
$$

Forward propagation

令

$$
\begin{aligned}
z_1^{(2)} &= \theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3 \\
z_2^{(2)} &= \theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3 \\
z_3^{(2)} &= \theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3
\end{aligned}
$$

则有

$$
\begin{aligned}
a_0^{(2)} &= 1 \\
a_1^{(2)} &= g(z_1^{(2)}) \\
a_2^{(2)} &= g(z_2^{(2)}) \\
a_3^{(2)} &= g(z_3^{(2)}) \\
z^{(3)} &= \theta^{(2)}a^{(2)} \\
h_\theta(x) = a^{(3)} &= g(z^{(3)})
\end{aligned}
$$

以上过程称为 **Forward propagation**
if network has s_j uints in layer j, s_{j+1} uints in layer j+1,then \theta^{(j)} will be of dimension s_{j+1}\times(s_j+1)

Multi-class classification

若是表示多个输出，那么 h_\theta(x) 维度将大于1，变成一个向量矩阵，这个时候输出也就变成了多为

cost function

对于

$$
(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(21)}), \ldots, (x^{(m)}, y^{(m)})
$$

这m个样本数据训练出来的神经网络来说，我们定义:
L = total number of layers in network
s_l = no. of units(not counting bias unit) in layer l

我们类比**logistic regression**的 J((\theta)

$$
J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2
$$

Neural network:

$$h_\theta(x) \in \mathbf{R}^K$$

$$(h_\theta(x))_i = i^{th} output$$

那么在神经网络中，cost function定义为

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log((h_\theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)] +$$

$$\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{j=1}^{S_l}\sum_{j=1}^{S_{l+1}}(\theta_{ji}^{(l)})^2$$

> Note:

1. l-1表示去掉输出层
2. i= 1 \to s_l 表示去掉 \theta_{j0} 这一列
3. j= 1 \to s_{j+1} 表示全部行

Backpropagetion algorithm(反向传播算法)

如果看不懂可以借鉴 Blog链接

1. Forward propagation

$$a^{(1)} = x$$

$$z^{(2)} = \theta^{(1)}a^{(1)}$$

$$a^{(2)} = g(z^{(2)})(add\ a_0^{(2)})$$

$$z^{(3)} = \theta^{(2)}a^{(2)}$$

$$a^{(3)} = g(z^{(3)})(add\ a_0^{(3)})$$

$$z^{(4)} = \theta^{(3)}a^{(3)}$$

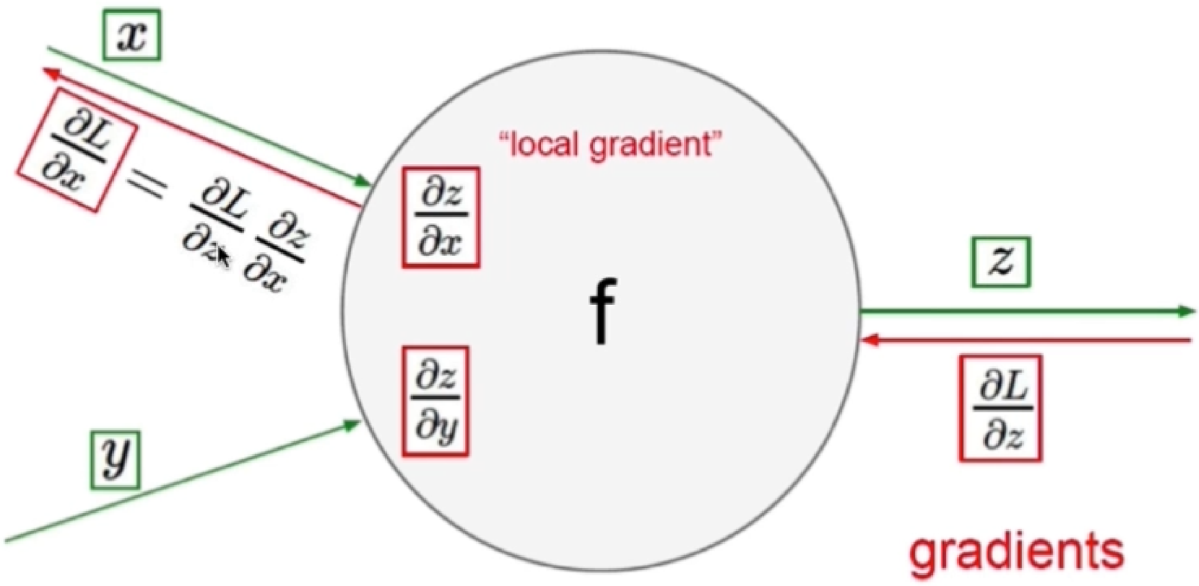
$$a^{(4)} = h_\theta(x) = g(z^{(4)})$$

2. 为了计算导数项，引入Back propagation algorithm
   Intuition: \delta^{(l)} = "error"\ of\ node\ j\ in\ layer\ l

$$\delta^{(l)} = \frac{\delta}{\delta z_j^{(l)}} cost(i)$$

$$eg.\ cost(i) = y^{(i)}log(h_\theta(z^{(i)})) + (1-y^{(i)})log(h_\theta(z^{(i)}))$$

直观理解



Example:
For each output unit(layer L = 4)

$$
\begin{aligned}
\delta_j^{(4)} &= a_j^{(4)} - y_j \ (a_j^{(4)} = h_\theta(x)_j) \\
\delta_j^{(3)} &= (\theta^{(3)})^T \delta^{(4)} \ .* \ g'(z^{(3)}) \\
\delta_j^{(2)} &= (\theta^{(2)})^T \delta^{(3)} \ .* \ g'(z^{(2)}) \\
g'(z^{(3)}) &= a^{(3)} \ .* \ (1 - a^{(3)})
\end{aligned}
$$

Step:
Training set {(x^{(1)},y^{(1)}),(x^{(2)},y^{(21)}),...,(x^{(m)},y^{(m)})}
Set \Delta_{ij}^{(l)} = 0 \ (for\ all\ l,i,j) (use to compute \frac{\delta}{\delta\theta_{ij}^{(l)}}J(\theta))

For i = 1 to m
set a^{(1)} = x^{(i)}
Perform forward propagation to compute a^{(i)} for l=2,3,...,L
Using y^{(i)} ,compute \delta^(l) = a^{(l)} - y^{(i)}
Compute \delta^{(L-1)},\delta^{(L-2)},...,\delta^{(2)}
\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)}\delta_j^{(l+1)}
\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T
D_{ij}^{(l)} := \frac{1}{m}\Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)} \ if j\ \ne 0 D_{ij}^{(l)} := \frac{1}
{m}\Delta_{ij}^{(l)} \ if \ j = 0
\frac{\delta}{\delta_{ij}^{(l)}}J(\theta) = D_{ij}^{(l)}

Gradient checking(梯度检测)

原理:

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}(\text{双 侧 差 分 })$$
$$eg.\epsilon = 10^{-4}$$

As for \vec\theta
\vec\theta \in \mathbf{R}^{n}
\vec\theta = [\theta_1,\theta_2,...,\theta_n]

$$\frac{d}{d\theta}J(\theta_1) \approx \frac{J(\theta_1+\epsilon,\theta_2,...,\theta_n)-J(\theta_1-\epsilon,\theta_2,...,\theta_n)}{2\epsilon}$$

$$\frac{d}{d\theta}J(\theta_n) \approx \frac{J(\theta_1,\theta_2+\epsilon,...,\theta_n)-J(\theta_1,\theta_2-\epsilon,...,\theta_n)}{2\epsilon}$$

$$\vdots$$

$$\frac{d}{d\theta}J(\theta_n) \approx \frac{J(\theta_1,\theta_2,...,\theta_n+\epsilon)-J(\theta_1,\theta_2,...,\theta_n-\epsilon)}{2\epsilon}$$

check that D_{vect} \approx gradApprox
gradApprox is calculated by

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$$

D_{vect} is calculated bt Backpropation

Note:

> 1. 使用反向传播计算 D_{vect}
> 2. 使用梯度检验计算 gradApprox
> 3. 确保 D_{vect} \approx gradApprox
> 4. 不再使用gradient checking,using backprop for learning

Important:

> Be sure to disable your gradient checking code before training your classifier.If yourun numerical gradient computation on evety iteration of gradient descent,your code will be bery slow

Random initialization(随机初始化)

关于 \vec\theta 的初始化一般具有两种方案

1. \vec\theta = \vec{0}
    - After each update,parameters corresponding to inputs going into each of two hidden units are identical
2. Initial each \theta_{ij}^{(l)} to a random value in [-\epsilon,\epsilon]

显然我们选用方案2作为我们在神经网络中的theta参数的初始化方案

Summary

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get h_\theta(x^{(i)}) for any x^{(i)}
3. Implement code to compute cost function J(\theta)
4. Implement backprop to compute partial derivatives \frac{\delta}{\delta\theta_{jk}} J(\theta) .for i = 1:m,Perform forward propagation and back propagation using example (x^{(i)},y^{(i)}) ,(Get activations a^{(l)} and delta terms \delta^{(l)} for l=2,3,...,L)
5. Use gradient to compare \frac{\delta}{\delta\theta_{jk}} J(\theta) .computed using backpropagation vs using numerical estimate of gradient of J(\theta)
6. Use gradient descent or advanced optimization method with back propogation to try to minimize J(\theta) as a function of parameters \theta

Note:

> J(\theta) is non-convex function in neural network,So,we can only get a local minimum.

Code

这里我们使用tensorflow来逐步构建一个简单的神经网络模型。

version 1

搜索 cifar-10 下载python格式的图片数据，一共有十类，这里我们使用二分类逻辑回归实现建模

```
#!/usr/bin/env python
# coding: utf-8


import tensorflow as tf
import os
import pickle
import numpy as np


CIFAT_DIR = '../cifar-10-batches-py'
print(os.listdir(CIFAT_DIR))


def load_data(filename):
    """read data from data file"""
    with open(os.path.join(filename), 'rb') as f:
        # data = pickle.load(f, encoding='bytes')

        # Python2.7代码
        data = pickle.load(f)
        return data['data'], data['labels']


class CifarData:
    def __init__(self, filenames, need_shuffle):
        all_data = []
        all_labels = []
```

```python
        # 关于zip函数 具体看
        # http://www.cnblogs.com/frydsh/archive/2012/07/10/2585370.html
        for filename in filenames:
            data, labels = load_data(filename)
            for item, label in zip(data, labels):
                # label一共有是个类别 每个类别各 5000各
                # 使用该判断获取类别
                if label in [0, 1]:
                    all_data.append(item)
                    all_labels.append(label)
        # 关于 vstack函数
        # https://www.cnblogs.com/nkh222/p/8932369.html
        self._data = np.vstack(all_data)
        # 归一化处理
        self._data = self._data / 127.5 - 1;
        self._labels = np.hstack(all_labels)
        print(self._data.shape)
        print(self._labels.shape)
        self._num_examples = self._data.shape[0]
        self._need_shuffle = need_shuffle
        self._indicator = 0
        if self._need_shuffle:
            self._shuffle_data()

    def _shuffle_data(self):
        # 【0,1,2,3,4】 => [2,1,3,4,0]
        p = np.random.permutation(self._num_examples)
        self._data = self._data[p]
        self._labels = self._labels[p]

    def next_batch(self, batch_size):
        """return batch_size examples as a batch """
        end_indicator = self._indicator + batch_size
        if end_indicator > self._num_examples:
            if self._need_shuffle:
                self._shuffle_data()
                self._indicator = 0
                end_indicator = batch_size
            else:
                raise Exception("have no more examples")
        if end_indicator > self._num_examples:
            raise Exception('batch size is larger than all examles')
        batch_data = self._data[self._indicator: end_indicator]
        batch_labels = self._labels[self._indicator: end_indicator]
        self._indicator = end_indicator
        return batch_data, batch_labels


train_filenames = [os.path.join(CIFAT_DIR, 'data_batch_%d' % i) for i in range(1, 6)]
test_filenames = [os.path.join(CIFAT_DIR, 'test_batch')]

train_data = CifarData(train_filenames, True)
test_data = CifarData(test_filenames, False)
```

```python
# batch_data, batch_labels = train_data.next_batch(10)
# print(batch_data,batch_labels)


# None 代表输入样本数是不确定的
x = tf.placeholder(tf.float32, [None, 3072])
# None
y = tf.placeholder(tf.int64, [None])
# 先构造一个 二分类器 因此输出为1
# (3072,1)
w = tf.get_variable('w', [x.get_shape()[-1], 1],
initializer=tf.random_normal_initializer(0, 1))
# (1, )
b = tf.get_variable('b', [1], initializer=tf.constant_initializer(0.0))
# [None,3072] *[3072,1] = [None,1]
y_ = tf.matmul(x, w) + b
# [None,1]
p_y_1 = tf.nn.sigmoid(y_)
# 这里-1参数表示缺省值 保证为1列即可
y_reshaped = tf.reshape(y, (-1, 1))
y_reshaped_float = tf.cast(y_reshaped, tf.float32)
# 计算loss
loss = tf.reduce_mean(tf.square(y_reshaped_float - p_y_1))
predict = p_y_1 > 0.5
correct_prediction = tf.equal(tf.cast(predict, tf.int64), y_reshaped)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float64))

with tf.name_scope('train_op'):
    # 这里1e-3是学习率 learning rate AdamOptimizer是梯度下降的一个变种
    train_op = tf.train.AdamOptimizer(1e-3).minimize(loss)

'''
到此为止我们的计算图搭建完成
'''

init = tf.global_variables_initializer()
batch_size = 20
train_steps = 100000
test_steps = 100

with tf.Session() as sess:
    sess.run(init)
    for i in range(train_steps):
        batch_data, batch_labels = train_data.next_batch(batch_size)
        loss_val, accu_val, _ = sess.run(
            [loss, accuracy, train_op],
            feed_dict={x: batch_data, y: batch_labels})
        if (i+1) % 500 == 0:
            print('[Train] Step: %d, loss: %4.5f,acc: %4.5f' % (i+1, loss_val,
accu_val))
        if(i+1) % 5000 == 0:
            test_data = CifarData(test_filenames, False)
            all_test_acc_val = []
```

```
            for j in xrange(test_steps):
                test_batch_data, test_batch_labels \
                 = test_data.next_batch(batch_size)
                test_acc_val = sess.run(
                    [accuracy],
                    feed_dict={
                        x: test_batch_data,
                        y: test_batch_labels
                    }
                )
                all_test_acc_val.append(test_acc_val)
            test_acc = np.mean(all_test_acc_val)
            print('[Test] Step: %d, acc: %4.5f ' % (i+1, test_acc))



```

运行结果:

```
[Train] Step: 98500, loss: 0.10032,acc: 0.90000
[Train] Step: 99000, loss: 0.10000,acc: 0.90000
[Train] Step: 99500, loss: 0.10080,acc: 0.90000
[Train] Step: 100000, loss: 0.05529,acc: 0.95000
(2000, 3072)
(2000,)
[Test] Step: 100000, acc: 0.81200

Process finished with exit code 0
```

version 2

这里我们继续使用该算法实现多分类器

```
#!/usr/bin/env python
# coding: utf-8


import tensorflow as tf
import os
import pickle
import numpy as np


CIFAT_DIR = '../cifar-10-batches-py'
print(os.listdir(CIFAT_DIR))


def load_data(filename):
    """read data from data file"""
    with open(os.path.join(filename), 'rb') as f:
        # data = pickle.load(f, encoding='bytes')

```

```python
        # Python2.7代码
        data = pickle.load(f)
        return data['data'], data['labels']


class CifarData:
    def __init__(self, filenames, need_shuffle):
        all_data = []
        all_labels = []
        # 关于zip函数 具体看
        # http://www.cnblogs.com/frydsh/archive/2012/07/10/2585370.html
        for filename in filenames:
            data, labels = load_data(filename)
            for item, label in zip(data, labels):
                all_data.append(item)
                all_labels.append(label)
        # 关于 vstack函数
        # https://www.cnblogs.com/nkh222/p/8932369.html
        self._data = np.vstack(all_data)
        # 归一化处理
        self._data = self._data / 127.5 - 1;
        self._labels = np.hstack(all_labels)
        print(self._data.shape)
        print(self._labels.shape)
        self._num_examples = self._data.shape[0]
        self._need_shuffle = need_shuffle
        self._indicator = 0
        if self._need_shuffle:
            self._shuffle_data()

    def _shuffle_data(self):
        # 【0,1,2,3,4】 => [2,1,3,4,0]
        p = np.random.permutation(self._num_examples)
        self._data = self._data[p]
        self._labels = self._labels[p]

    def next_batch(self, batch_size):
        """return batch_size examples as a batch """
        end_indicator = self._indicator + batch_size
        if end_indicator > self._num_examples:
            if self._need_shuffle:
                self._shuffle_data()
                self._indicator = 0
                end_indicator = batch_size
            else:
                raise Exception("have no more examples")
        if end_indicator > self._num_examples:
            raise Exception('batch size is larger than all examles')
        batch_data = self._data[self._indicator: end_indicator]
        batch_labels = self._labels[self._indicator: end_indicator]
        self._indicator = end_indicator
        return batch_data, batch_labels

```

```python

train_filenames = [os.path.join(CIFAT_DIR, 'data_batch_%d' % i) for i in range(1, 6)]
test_filenames = [os.path.join(CIFAT_DIR, 'test_batch')]

train_data = CifarData(train_filenames, True)
test_data = CifarData(test_filenames, False)
# batch_data, batch_labels = train_data.next_batch(10)
# print(batch_data,batch_labels)


# None 代表输入样本数是不确定的
x = tf.placeholder(tf.float32, [None, 3072])
# None
y = tf.placeholder(tf.int64, [None])
# 先构造一个 二分类器 因此输出为1
# (3072,10)
w = tf.get_variable('w', [x.get_shape()[-1], 10],
initializer=tf.random_normal_initializer(0, 1))
# (10, )
b = tf.get_variable('b', [10], initializer=tf.constant_initializer(0.0))
# [None,3072] *[3072,10] = [None,10]
y_ = tf.matmul(x, w) + b

# 关于softmax https://www.zhihu.com/question/23765351
# [[0,01,0.9,...,0.02],[]]
p_y = tf.nn.softmax(y_)
# 6 -->[0,0,0,0,0,1,0,0,0,0]
y_one_hot = tf.one_hot(y, 10, dtype=tf.float32)
loss = tf.reduce_mean(tf.square(y_one_hot - p_y))

'''
# [None,10]
p_y_1 = tf.nn.sigmoid(y_)
# 这里-1参数表示缺省值 保证为1列即可
y_reshaped = tf.reshape(y, (-1, 1))
y_reshaped_float = tf.cast(y_reshaped, tf.float32)
# 计算loss
loss = tf.reduce_mean(tf.square(y_reshaped_float - p_y_1))
'''

# indices
predict = tf.argmax(y_, 1)
correct_prediction = tf.equal(predict, y)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float64))

with tf.name_scope('train_op'):
    # 这里1e-3是学习率 learning rate AdamOptimizer是梯度下降的一个变种
    train_op = tf.train.AdamOptimizer(1e-3).minimize(loss)

'''
到此为止我们的计算图搭建完成
'''

```

```
init = tf.global_variables_initializer()
batch_size = 20
train_steps = 10000
test_steps = 100

with tf.Session() as sess:
    sess.run(init)
    for i in range(train_steps):
        batch_data, batch_labels = train_data.next_batch(batch_size)
        loss_val, accu_val, _ = sess.run(
            [loss, accuracy, train_op],
            feed_dict={x: batch_data, y: batch_labels})
        if (i+1) % 500 == 0:
            print('[Train] Step: %d, loss: %4.5f,acc: %4.5f' % (i+1, loss_val, accu_val))
        if(i+1) % 5000 == 0:
            test_data = CifarData(test_filenames, False)
            all_test_acc_val = []
            for j in xrange(test_steps):
                test_batch_data, test_batch_labels \
                 = test_data.next_batch(batch_size)
                test_acc_val = sess.run(
                    [accuracy],
                    feed_dict={
                        x: test_batch_data,
                        y: test_batch_labels
                    }
                )
                all_test_acc_val.append(test_acc_val)
            test_acc = np.mean(all_test_acc_val)
            print('[Test] Step: %d, acc: %4.5f ' % (i+1, test_acc))
```

Note

这两部分代码都没有用到hidden layer.
实际上，code 1 展示的是一个神经元，这里也可以认为是逻辑回归。也就是logistic regression 看做是仅仅含有一个神经元的单 层神经网络
code 2 实际上也就是多维的logistic regreesion,其实softmax regression可以看做是含有k个神经元的一层神经网络。