# Linux相关

## **2019.10.26**

### **1.Ubuntu设置字体大小**

|  |
| --- |
| (1)安装gnome-tweaks桌面配置工具，sudo apt  install    gnome-tweaks  (2)运行gnome-tweaks  (3)在字体中设置合适放大（缩小）倍数 |

### **2.Ubuntu解决WPS光标不跟随问题**

|  |
| --- |
| (1)安装qt4-qtconfig,sudo apt-get install qt4-qtconfig  (2)安装ibus-qt,sudo apt-get install ibus-qt4  (3)运行qt4-qtconfig,在q t4-qtconfig “界面”选项卡默认输入法选项中看到ibus，并选定，然后设置XIM输入风格为光标跟随风格。 |

### Ubuntu双系统下挂载Windows分区

|  |
| --- |
| 1.查看磁盘信息  (1)设备文件名称 file name  (2)挂载目录 mount point ，创建挂载目录 sudo mkdir +路径  (3)文件类型 type  (4)挂载参数 options，默认defaults  (5)是否备份 dump 0或1  (6)开机自检 pass：一般设置为2。如果不想自检，那就设置为0，挂载点为根目录的设备，设置为1，其它需要自检的设备，设置为2。  2.修改fstab文件  打开/etc.fstab文件，将磁盘或分区的文件名称和挂载信息填写：  sudo gedit /etc/fstab  例如：/dev/sdb2 /media/csl/文件 ntfs defaults 0 0  重启即可挂载 |

### Ubuntu18.04开机出现检测到系统程序出现问题

|  |
| --- |
| 打开终端，输入命令：  sudo gedit /etc/default/apport  将文件中的enabled = 1 修改为0，然后保存 |

### notepadqq不小心将工具栏删除后，无法找回

|  |
| --- |
| 解决方案：到/home/.config/Notepadqq/文件夹下，删除所有文件，然后重启notepadqq，个人习惯设置需要重新设置。 |

## 2019.10.27

### 1.

|  |
| --- |
|  |

# Python相关

## 2019.10.16

### **1. array 与 matrix 的区别**

|  |
| --- |
| (1)np.array( )更具通用性，np.mat()只适用于二维矩阵，维数超过2以后，np.mat()就不适用了。  (2)mat()函数与array()函数生成矩阵所需的数据格式有区别：mat()函数中数据可以为字符串以分号(；)分割，或者为列表形式以逗号（，）分割。而array()函数中数据只能为逗号。  (3) mat()函数与array()函数生成的矩阵计算方式不同  (4)利用np.mat()将数组转换为矩阵 |

### **2.np.multiply()、np.dot()和星号（\*）三种乘法运算的区别**

|  |
| --- |
| (1)np.multiply()  函数作用：数组和矩阵对应位置相乘，输出与相乘数组/矩阵的大小一致。  (2)np.dot()  函数作用：对于秩为1的数组，执行对应位置相乘，然后再相加；对于秩不为1的二维数组，执行矩阵乘法运算；超过二维的可以参考numpy库介绍  (3)星号（\*）乘法运算  作用：对数组执行对应位置相乘；对矩阵执行矩阵乘法运算 |

## **2019.10.18**

### **1.求矩阵的逆**

|  |
| --- |
| 示例代码如下：  import numpy as np  a = np.array([[1,1,1],[1,0,-1],[1,-1,0]])  array([[ 1, 1, 1],  [ 1, 0, -1],  [ 1, -1, 0]])  print(np.linalg.inv(a))  [[ 0.33333333 0.33333333 0.33333333]  [ 0.33333333 0.33333333 -0.66666667]  [ 0.33333333 -0.66666667 0.33333333]]  A = mat(a)  A.I |

## 2019.10.18

# 吴恩达机器学习笔记

## 线性回归

1.1 代价函数

|  |
| --- |
| 线性回归的代价函数： |

### 1.2特征缩放

|  |
| --- |
| 特征缩放(feature scaling)大致的思路是：梯度下降算法中，在有多个特征的情况下，如果你能确保这些不同的特征都处在一个相近的范围，这样梯度下降法就能更快地收敛。  执行特征缩放时，我们通常的目的是将特征的取值约束到−1到+1的范围内。  最简单的方法是令 ， 是平均值， 是标准差。  注意：特征缩放其实并不需要太精确，其目的只是为了让梯度下降能够运行得更快一点，让梯度下降收敛所需的循环次数更少一些而已。 |

## 2. 逻辑回归（分类问题）

### **2.1 逻辑回归模型**

|  |
| --- |
| 逻辑回归模型：  其中：X为特征向量  g 代表逻辑函数是一个常用的逻辑常用函数为S形函数，公式为：<    的作用是，对于给定的输入变量，根据选择的参数计算输出变量输出等于1的可能性，即 |

### **2.2 逻辑回归代价函数**

|  |
| --- |
| 逻辑回归的代价函数为：    简化如下：    带入代价函数得到：    即：    将代入得 |

### **2.3 求代价最小参数**

|  |
| --- |
| 得到代价函数后，可以采用梯度下降算法来求得使代价函数最小的参数。  Repeat{    (simultaneously updata all  )  }  将 求偏导后得到：  Repeat{    (simultaneously updata all  )  } |

### **2.4 特征缩放**

|  |
| --- |
| 特征缩放可以提高梯度下降的收敛速度，这个特征缩放也适用于逻辑回归。如果特征范围差距很大的话，应用特征缩放的方法，可使梯度下降收敛更快。 |

## 3. 正则化

### **3.1 过拟合问题**

|  |
| --- |
| 欠拟合：不能很好地适应训练集；  过拟合：过于强调拟合原始数据，而丢失了算法的本质：预测新数据。  解决过拟合问题的两种办法：   1. 丢弃一些不能帮助我们正确预测的特征。 2. 正则化(Regularization)。保留所有的特征，但是减少参数的大小(magnitude)。 |

### **3.2 正则化代价函数**

|  |
| --- |
| (1)线性回归：    求最小化参数：  Repeat until convergence{      }  (2)正规方程    矩阵尺寸(n+1)\*(n+1)  (3)逻辑回归方程    求最小化参数：  Repeat until convergence{      } |

## 4. 神经网络

### **4.1 神经网络模型**

|  |
| --- |
|  |

### **4.2 代价函数**

|  |
| --- |
| m ：代表样本容量总数  k ：代表K分类  l ：代表神经网络的总层数  sl ：代表神经网络l层的激活单元总数 |

# Git相关

## 添加SSH密钥

|  |
| --- |
| (1)创建密钥  ssh-keygen -t rsa -C “邮箱”  (2)在GitHub上添加密钥  (3)测试是否成功  ssh -T git@github.com  (4)ubuntu查看id\_rsa.pub文件  cat .ssh/id\_rsa.pub |

## 设置自己的名字和邮箱

|  |
| --- |
| $ git config --global user.name "Your Name"  $ git config --global user.email "email@example.com" |

## 在本地创建版本库

|  |
| --- |
| 1. cd /e/LocalGithub/   (2)变为Git管理的仓库 git init |

## 添加远程库与本地库连接

|  |
| --- |
| git remote add origin 自己的版本库链接 |

## 传输文件

|  |
| --- |
| (1)添加到本地仓库  $ git add .  (2)添加提交描述  $ git commit -m '内容'  (3)把本地仓库代码提交  $ git push -u origin master |

## 问题汇总

|  |
| --- |
| 1.合并Github和本地代码仓库命令  $ git pull --rebase origin master |

# Matplotlib

## 基本语法

|  |
| --- |
| #基本用法  代码如下：  import matplotlib.pyplot as plt  import numpy as np  x = np.linspace(-1,1,50)  y = 2\*x+1  plt.plot(x,y)  plt.show() |

## Figure图像

|  |
| --- |
| #figuer 使用  代码如下：  x = np.linspace(-1,1,50)  y1 = 2\*x+1  y2 = x\*x+1  plt.figure(num=1)  plt.plot(x,y1)  plt.figure(num=2,figsize=(8,5))  plt.plot(x,y1,color = 'red',linewidth = 1.0,linestyle='--')  plt.plot(x,y2)  plt.show() |

## 设置坐标1

|  |
| --- |
| # 设置坐标轴1  代码如下：  x = np.linspace(-3,3,50)  y1 = 2\*x+1  y2 = x\*x+1  plt.figure(num=2,figsize=(8,5))  plt.plot(x,y1,color = 'red',linewidth = 1.0,linestyle='--')  plt.plot(x,y2)  plt.xlim((-1,2))  plt.ylim((-2,3))  plt.xlabel("X")  plt.ylabel("Y")  new\_ticks=np.linspace(-1,2,5)  print(new\_ticks)  plt.xticks(new\_ticks)  plt.yticks([-2,-1.8,-1,1.22,3],  [r'$really\ bad\ \alpha$',r'$bad$',r'$normal$',r'$good$',  r'$really\ good$'])  plt.show() |

## 设置坐标2

|  |
| --- |
| #设置坐标轴2  代码如下：  x = np.linspace(-3,3,50)  y1 = 2\*x  y2 = x\*x  plt.figure(num=2,figsize=(8,5))  plt.plot(x,y1,color = 'red',linewidth = 1.0,linestyle='--')  plt.plot(x,y2)  plt.xlim((-1,2))  plt.ylim((-2,3))  plt.xlabel("X")  plt.ylabel("Y")  new\_ticks=np.linspace(-1,2,5)  print(new\_ticks)  plt.xticks(new\_ticks)  plt.yticks([-2,-1.8,-1,1.22,3],  [r'$really\ bad\ \alpha$',r'$bad$',r'$normal$',r'$good$',  r'$really\ good$'])  #gca = 'get current axis'  ax = plt.gca()  ax.spines['right'].set\_color('none')  ax.spines['top'].set\_color('none')  ax.xaxis.set\_ticks\_position('bottom')  ax.yaxis.set\_ticks\_position('left')  ax.spines['bottom'].set\_position(('data',0))#outward,axes  ax.spines['left'].set\_position(('data',0))  plt.show() |

## Legend图例

|  |
| --- |
| #Legend 图例  代码如下：  x = np.linspace(-3,3,50)  y1 = 2\*x  y2 = x\*x  plt.figure(num=2,figsize=(8,5))  l1, = plt.plot(x,y1,color = 'red',linewidth = 1.0,linestyle='--',label='down')  l2, = plt.plot(x,y2,label='up')  plt.legend(handles=[l1,l2,],labels=['aaa','bbb'],loc='upper right')  plt.xlim((-1,2))  plt.ylim((-2,3))  plt.xlabel("X")  plt.ylabel("Y")  new\_ticks=np.linspace(-1,2,5)  print(new\_ticks)  plt.xticks(new\_ticks)  plt.yticks([-2,-1.8,-1,1.22,3],  [r'$really\ bad\ \alpha$',r'$bad$',r'$normal$',r'$good$',  r'$really\ good$'])  #gca = 'get current axis'  ax = plt.gca()  ax.spines['right'].set\_color('none')  ax.spines['top'].set\_color('none')  ax.xaxis.set\_ticks\_position('bottom')  ax.yaxis.set\_ticks\_position('left')  ax.spines['bottom'].set\_position(('data',0))#outward,axes  ax.spines['left'].set\_position(('data',0))  plt.show() |

## Annotation标注

|  |
| --- |
| #scastter画点图  代码如下：  #Annotation标注  x = np.linspace(-3,3,50)  y = 2\*x+1  plt.figure(num=2,figsize=(8,5))  plt.plot(x,y)  #gca = 'get current axis'  ax = plt.gca()  ax.spines['right'].set\_color('none')  ax.spines['top'].set\_color('none')  ax.xaxis.set\_ticks\_position('bottom')  ax.yaxis.set\_ticks\_position('left')  ax.spines['bottom'].set\_position(('data',0))#outward,axes  ax.spines['left'].set\_position(('data',0))  x0 = 1  y0 = 2\*x0+1  plt.scatter(x0,y0,s=50,color = 'b')  plt.plot([x0,x0],[y0,0],'k--',linewidth=2.5)  #method 1  ###################  plt.annotate(r'$2x+1=%s$'%y0,xy=(x0,y0),xycoords='data',xytext=(+30,-30),textcoords='offset points', fontsize=16,  arrowprops=dict(arrowstyle='->',connectionstyle='arc3,rad=.2'))  #method 2  ###################  plt.text(-3.7,3,r'$This\ is\ the\ some\ text.\sigma\_i\ \alpha\_t$',  fontdict={'size':16,'color':'red'})  plt.show() |

## tick 能见度

|  |
| --- |
| #代码如下  #Annotation标注  x = np.linspace(-3,3,50)  y = 0.1\*x  plt.figure(num=2,figsize=(8,5))  plt.plot(x,y,linewidth=30)  #gca = 'get current axis'  ax = plt.gca()  ax.spines['right'].set\_color('none')  ax.spines['top'].set\_color('none')  ax.xaxis.set\_ticks\_position('bottom')  ax.yaxis.set\_ticks\_position('left')  ax.spines['bottom'].set\_position(('data',0))#outward,axes  ax.spines['left'].set\_position(('data',0))  for label in ax.get\_xticklabels()+ax.get\_yticklabels():  label.set\_fontsize(15)  label.set\_bbox(dict(facecolor='white',edgecolor='red',alpha=0.9))  plt.show() |