**1. 人工智能的技术学习**

**1.1. 基于工程数据集的划分**

1. 机器视觉
2. 自然语言
3. 商业数据分析

**1.2. 数据集，特征抽取，分类**

1. 模式识别
2. 机器学习

**1.3. 技术讲解的结构**

1. 分类
2. 特征学习（深度神经网络 / 卷积神经网络 / 循环神经网络(lstm)）
3. 数据集

**1.4. 技术框架选择**

1. Maxnet
2. Tensorflow
3. **Pytorch**

* UI：Qt
  + |- OpenCV
  + |- Numpy
  + |- Matplotlib

**2. 分类**

**2.1. 逻辑回归（分类模型）的数学模型**

* 目标：
  1. 分类的原理
  2. 分类结果的概率（Sigmoid函数 + Softmax函数）

1. 回顾线性回归模型

* y=xW+by=xW+b
* 实际的模型
  + y′=xW+b+ϵy′=xW+b+ϵ
  + y−y′=ϵy−y′=ϵ
  + ϵϵ服从Gauss分布
* x1,x2,…,xn→y1,y2,…,ynx1,x2,…,xn→y1,y2,…,yn

1. 逻辑分类模型

* yA=xWA+bAyA=xWA+bA
* yB=xWB+bByB=xWB+bB
* y={1,yA−yB>00,其他y={1,yA−yB>00,其他
* y={1,x(WA−WB)>00,其他y={1,x(WA−WB)>00,其他
* xW′+b′=yxW′+b′=y
* 条件
  + y值必须是离散的
* 误差模型
  + y=Wx+b+ϵy=Wx+b+ϵ
* 对一个样本xx
  + p(y=1)=p(xW+b+ϵ>0)=p(ϵ>−xW−b)p(y=1)=p(xW+b+ϵ>0)=p(ϵ>−xW−b)
* 如果假设p服从正态分布
  + p(y=1)=1−Fϵ(−xW)p(y=1)=1−Fϵ(−xW)
* probit模型
* 要命的问题：
  + 高斯分布没有累积分布函数；
  + 近似高斯累积分布函数的函数：Lapalce分布函数
* 逻辑分布函数：
  + p(x)=e−x(1+e−x)2p(x)=e−x(1+e−x)2
* Sigmoid(x)=11+e−xSigmoid(x)=11+e−x
  + 逻辑分布中的x替换为-1.702x ，则曲线与高斯累积分布函数的曲线，基本上完全重合。
* 多个样本误差概率
  + p(Y)=∏yi∈Yp(yi)p(Y)=∏yi∈Yp(yi)
* ∏h(Xi)yi(1−h(Xi))1−yi∏h(Xi)yi(1−h(Xi))1−yi
* 概率最大，找到W b是的概率最大
* 最小值模型
  + 求自然对数，取负数。
  + L=−∑iyiln(h(Xi))−(1−yi)(1−h(Xi))L=−∑iyiln(h(Xi))−(1−yi)(1−h(Xi))
* 这个函数的最小值，无法使用最小二乘法求解：
  + 梯度下降法
* 这个最小值的求解函数：
  + 俗称损失函数
* 总结：
  + 逻辑回归模型：
    - y=Sigmoid(xW+b)∈[0,1)y=Sigmoid(xW+b)∈[0,1)
  + 损失模型：
    - 交叉熵函数
  + 求解模型：
    - 梯度下降法
* y=[10]y=[10]
* y=[01]y=[01]
* y=[x1x2]y=[x1x2]
* x1>x2x1>x2属于第一类，且概率是x1x1

**2.2. 梯度下降法**

**2.2.1. 数据怎么表示Tensor**

1. 数据表示方式
   1. Python的数据表达式/数据变量
      * int，float
      * list，tuple
   2. numpy
      * 向量 + 向量运算
   3. Tensor
      * 向量 + 向量运算 + 求导
2. PyTorch的张量的表示

In [ ]:

**import** **torch**

*# help(torch)*

*# help(torch.tensor)*

*# help(torch.storage)*

* 张量的构造：
  + **BoolTensor**
  + ByteTensor
  + CharTensor
  + **DoubleTensor**
  + **FloatTensor**
  + **IntTensor**
  + **LongTensor**
  + ShortTensor
* 张量：
  + 构造器
  + 运算符
    - 数学运算
    - 数据结构运算
      * 下标运算
  + 数据
  + 函数
    - 基本操作（数据结构）
    - 数学运算（四则+向量+矩阵运算+矩阵分析）
    - 自动求导

In [ ]:

**import** **torch**

help(torch.FloatTensor.\_\_init\_\_)

* 编程语言：
  + Cython

In [ ]:

**import** **torch**

t = torch.FloatTensor((1, 1, 2))

print(t.shape)

In [ ]:

t2 = torch.Tensor([1,2,3,4]) *# FloatTensor /DoubleTensor*

print(t2)

help(torch.Tensor.\_\_init\_\_)

* 提供工厂模式
  + 函数构造对象

In [ ]:

**import** **torch**

*# help(torch.tensor)*

*# help(torch.zeros)*

*# 特殊张量的工厂函数*

* 推荐方式

In [ ]:

**import** **torch**

tr = torch.tensor([1,2,3], dtype=torch.float64)

print(tr)

1. Tensor的下标操作运算

In [ ]:

**import** **torch**

t = torch.tensor(

[

[1, 2, 3, 4],

[4, 5, 6, 7],

[6, 7, 8, 9]

]

)

t

In [ ]:

t[1]

In [ ]:

t[0:2:1]

In [ ]:

t[slice(0,2,1)]

In [ ]:

t[...]

* 多个参数

In [ ]:

t[0,1]

In [ ]:

t[0:2:, 1]

In [ ]:

t[...,1]

* 支持数组

In [ ]:

**import** **torch**

**import** **numpy** **as** **np**

t = torch.tensor(

[

[1, 2, 3],

[4, 5, 6],

[6, 7, 8]

]

)

idx = np.array(

[

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

)

print(idx[ [0,1] ])

print(t[ [1, 2] ])

* 支持逻辑数组

In [ ]:

**import** **torch**

**import** **numpy** **as** **np**

t = torch.tensor(

[

[1, 2, 3],

[4, 5, 6],

[6, 7, 8]

]

)

idx = np.array(

[

[1, 2, 3],

[4, 5, 6],

[7, 8, 9]

]

)

*# print(idx > 3)*

*# print(t > 4 )*

*# t [ t > 3] = 88*

*# print(t)*

print(t[t>3])

In [ ]:

t == t

t.T

**2.2.2. 求导实现（自动求导）**

* 自定求导的规则
  + 对值求导
  + 上下变化：函数：在函数的环境中
    - requires\_grad = False/True （未来是的时候，会被自定跟踪）
    - grad\_fn
    - grad返回导数（某个值点）
    - dtype：float求导
* f′(x)=limϵ→0f(x+ϵ)−f(x−ϵ)2ϵf′(x)=limϵ→0f(x+ϵ)−f(x−ϵ)2ϵ

In [ ]:

**import** **torch**

*# 1. 定义求导的值*

x = torch.Tensor([5]) *# , dtype=torch.float*

x.requires\_grad=**True**

*# 2. 运算的函数与求导的值有关*

y = x \*\* 2

*# 3. 求导*

y.backward()

*# 4. 输出导数*

print(x.grad)

In [ ]:

y\_ = 2 \* x

print(y\_)

**2.2.3. 梯度下降的实现**

* 梯度下降算法模型：
  + x−=grad∗ηx−=grad∗η
    - grad决定方向正确
    - ηη决定速度

1. numpy
   * y=x2−2x+1=(x−1)2y=x2−2x+1=(x−1)2

In [ ]:

**import** **numpy** **as** **np**

x = 0

grad\_fn = **lambda** x: 2 \*x - 2

learning\_rate = 0.01

epoch = 1000

x\_list = []

*# 迭代梯度下降*

**for** e **in** range(epoch):

*# 1. 求导*

x\_grad = grad\_fn(x)

*# 2. 迭代更新x*

x -= learning\_rate \* x\_grad

*# 3. 记录x*

x\_list.append(x)

print(x)

%**matplotlib** inline

**import** **matplotlib.pyplot** **as** **plt**

plt.plot(range(epoch), x\_list)

plt.show()

1. tensor实现

In [ ]:

**import** **torch**

x = torch.Tensor([0.0])

x.requires\_grad = **True**

learning\_rate = 0.01

epoch = 1000

x\_list = []

**for** e **in** range(epoch):

*# 1. 函数（损失函数）*

y = x \*\* 2 - 2 \* x + 1

*# 2. 求导*

y.backward(retain\_graph=**True**)

*# 费求导跟踪环境*

**with** torch.autograd.no\_grad():

*# 3. 更新*

x -= learning\_rate \* x.grad *# 这个操作被自定求导进行了跟踪*

*# 4. 记录x*

x\_list.append(x.detach().clone().numpy())

x.grad.zero\_()

print(x.detach().clone().numpy())

print(x)

%**matplotlib** inline

**import** **matplotlib.pyplot** **as** **plt**

plt.plot(range(epoch), x\_list)

plt.show()

**2.3. 分类器（逻辑回归）的学习实现**

* 使用鸢尾花数据
* 逻辑回归模型：
  1. 输出模型
     + y=S(xW)y=S(xW)
  2. 误差模型
     + L=crossentropyloss(x)L=crossentropyloss(x)
       - 隐含的W（Wb）x(x,1)W（Wb）x(x,1)
  3. 求最小值（梯度下降法）
  4. 最总通过w得到一个最小w是的损失最下
     + 起到分类的作用

1. 输出模型的实现
   * 鸢尾花数据集

In [ ]:

**import** **sklearn**

**import** **sklearn.datasets**

**import** **torch**

data, target = sklearn.datasets.load\_iris(return\_X\_y=**True**)

data, target

In [ ]:

x = torch.Tensor(data[0:100])

y = torch.Tensor(target[0:100])

x, y

In [ ]:

*# 逻辑回归模型*

w = torch.randn(1, 4) *# 迭代更新*

b = torch.randn(1)

y\_ = torch.nn.functional.linear(input=x, weight=w, bias=b)

print(y\_.shape)

sy\_ = torch.sigmoid(y\_)

print(sy\_.shape)

1. 损失模型

In [ ]:

**import** **sklearn**

**import** **sklearn.datasets**

**import** **torch**

data, target = sklearn.datasets.load\_iris(return\_X\_y=**True**)

x = torch.Tensor(data[0:100])

y = torch.Tensor(target[0:100]).view(100, 1)

*# 逻辑回归模型*

w = torch.randn(1, 4) *# 迭代更新*

b = torch.randn(1)

y\_ = torch.nn.functional.linear(input=x, weight=w, bias=b)

print(y\_.shape)

sy\_ = torch.sigmoid(y\_)

print(sy\_.shape)

loss\_logit = torch.nn.functional.binary\_cross\_entropy\_with\_logits(y\_, y, reduction="sum")

loss = torch.nn.functional.binary\_cross\_entropy(sy\_, y, reduction="sum")

print(loss, loss\_logit)

1. 分类器实现

In [ ]:

**import** **sklearn**

**import** **sklearn.datasets**

**import** **torch**

data, target = sklearn.datasets.load\_iris(return\_X\_y=**True**)

x = torch.Tensor(data[0:100])

y = torch.Tensor(target[0:100]).view(100, 1) *# 数据格式*

*# 逻辑回归模型*

w = torch.randn(1, 4) *# 迭代更新*

b = torch.randn(1)

*# 学习目标：w，b*

w.requires\_grad= **True**

b.requires\_grad= **True**

epoch = 10000

learning\_rate = 0.0001

**for** e **in** range(epoch):

*# 1. 计算输出*

y\_ = torch.nn.functional.linear(input=x, weight=w, bias=b)

sy\_ = torch.sigmoid(y\_)

*# 2. 计算误差*

loss = torch.nn.functional.binary\_cross\_entropy(sy\_, y, reduction="mean")

*# 3. 求导*

loss.backward()

*# 4. 建立无导跟踪环境*

**if** e % 500 == 0:

**with** torch.autograd.no\_grad():

*# 5. 更新w,b*

w -= learning\_rate \* w.grad

b -= learning\_rate \* b.grad

*# 6. 清零*

w.grad.zero\_()

b.grad.zero\_()

*# 预测，观察损失*

sy\_[sy\_ > 0.5] = 1

sy\_[sy\_ <=0.5] = 0

correct\_rate = (sy\_ == y).float().mean()

print(F"轮数：**{**e**:**05d**}**,损失：**{**loss**:**10.6f**}**, 测试准确率：**{**correct\_rate \* 100.0**:**8.2f**}**%")

* 作业:
  1. 独立完成鸢尾花分类；
  2. 整理笔记
     + 每一个代码都跑一遍
     + 补充你自己的理解
     + 补充某些函数帮助，并加上自己的理解；
  3. 四个特征取两个特征来训练，并且可视化分类的效果；（可选）
     + matplotlib可视化；
  4. 提交到git服务器，更新：README.md

**1. 特征学习的总结**

1. 定义与初始化需要训练的权重系统【重复】
   * 一个张量是否可训练（requires\_grad=True）
2. 根据神经网络的模型，做前向计算 【核心】
   * 预测值-> softmax/sigmoid转换为概率 -> 使用最大分量下标作为分类的结果，值作为分类的概率
     + 做概率阈值过滤
3. 根据预测值与真实值做损失计算【核心】
   * 保证损失最小。
   * 梯度下降
4. 对损失值求导 【关键】
   * 利用导数与学习率更新我你们的可训练权重系数 【重复】

* 最终得到一组训练好的权重：
  + 对分类是最有利

**2. PyTorch神经网络结构**

* Layer(运算与权重系数 = Layer【固化了可训练的参数】) | - Conv2D | - MaxPool2D | - Relu | - Linear | - .....

1. conv2d Layer
   * 运算固化
     + 通过重载实现重定义forward(x)
       - 提供可调用对象 conv = Conv2d(....) y =conv.forward(x) === y = conv(x)
   * 运算的权重系数自动的管理
     + 在构造器实现自动跟踪：
       - self.任何便layer中的参数被自动跟踪
   * 基本上所有的函数都封装成Layer
   * 系数访问：parameters()返回所有权重矩阵张量
   * 注意：
     + 使用PyTorch的实现基本上满足95%的需求

In [ ]:

**from** **torch.nn** **import** \*

*# help(Conv2d)*

1. Module封装
   * 自动构建权重
   * 自动跟踪可训练参数（self.layer）
   * forward用户的运算实现接口（layer必须在构造器调用，必须是成员，这样定义的layer的参数才会被跟踪）
     + 可调用对象
   * 参数的返回访问
2. 编程模式：
   * layer实现具体的运算
     + 继承Layer实现自己的运算（层）
     + 直接使用Pytorch提供的Layer
   * Module来把各种Layer组装为网络结构
     + 继承Module使用Layer拼装各种网络结构
   * 损失函数
     + 调用损失Layer
   * 权重更新
     + Optimizier（Adam, SGD, .....）:对学习率的处理（学习率是固定）

In [ ]:

**from** **torch.nn** **import** MSELoss

*# help(MSELoss)*

**3. 图像数据集**

**4. 经典的卷积神经网络实现图像分类**