感知机与逻辑回归

18308133 刘显彬

1.实验原理

1.1 感知机

1.1.1 感知机本质上是一种线性模型，该模型认为一个数据的label值与该数据的特征有着某种线性关系，其主要思想是：特征的不同维度的权重对结果的贡献是不一样的，感知机算法的目的就是找到这样的一组权重值：能使得通过这一组权重与特征的线性变换之后产生的**预测结果与实际结果值最接近**。设权重向量为W，输入单个数据的特征矩阵为x,特征维度为N，它的实际标签为$label$：则线性映射的结果为：

$$y = \sum_{i=1}^{N} W_i * x_i \tag{1}$$

但是线性变换的结果值域为$\mathbb{R}$，并不能直接得到一个$label \in \{-1, 1\}$,因此我们还需将其通过一个符号函数$sign(x)$($x > 0$时结果为1，其余情况为-1)转换值域，还可以设定一个额外的阈值$threshold$，表示一个常数项的偏置，记最终的输出为h：

$$\begin{aligned} h(x) &= sign(y - threshold) && (2)\\ &= sign(\sum_{i=1}^{N} W_i * x_i - threshold * (1)) && (3)\\ &= sign(\sum_{i=0}^{N} W_i * x_i) \quad let\ W_0 \leftarrow threshold, x_0 \leftarrow 1 && (4)\\ &= sign(\widetilde{W}^T \widetilde{x}) && (5) \end{aligned}$$

1.1.2 现在感知机算法需要找出参数权重：W的最佳值，使所有预测值与实际值最接近。当$y(\widetilde{W}^T * \widetilde{x}) < 0 (*)$时，预测是错误的，因为预测值的label与实际label是相反的，事实上，我们关心是否能正确分类，因此，使误分类尽可能少是我们的最终目的。于是，我们可以用：误分类的点与正确分类之间的距离来表征我们PLA模型的表现，也就是误分类点距离划分平面$(\widetilde{W}^T * \widetilde{x}) = 0$的距离$d(x)$:

$$d(x) = \frac{\left|\widetilde{W}^T \widetilde{x}\right|}{\|W\|} = -y\frac{\widetilde{W}^T \widetilde{x}}{\|W\|} \tag{6}$$

最后的等号是显然的，因$-y(\widetilde{W}^T * \widetilde{x}) > 0 (*)$，且y的模为1，因此可以去掉分子的绝对值符号。

因此我们可以用所有误分类点与划分平面的距离和作为我们优化的目标：

$$\begin{aligned} L(\widetilde{W}) &= -\sum_{x \in M} y\frac{\widetilde{W}^T \widetilde{x}}{\|W\|}, \quad M: misclassification && (7)\\ &= -\frac{1}{\|W\|}\sum y * (\widetilde{W}^T \widetilde{x}) && (8) \end{aligned}$$

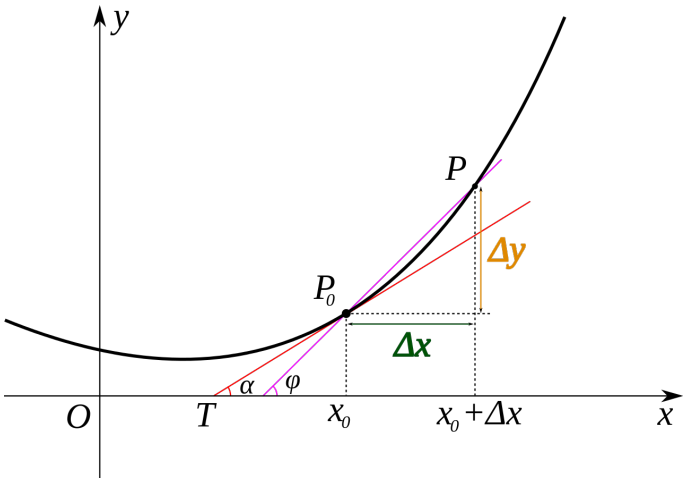
也就是原来我们想要的结果：找到一组W使误分类结果尽可能接近正确结果，可以建模成:

$$\widetilde{W} = \arg\min_{\widetilde{W}} L(\widetilde{W}) \tag{9}$$

$$= \arg\min_{\widetilde{W}} \sum_{x \in M} -y * (\widetilde{W}^T \widetilde{x}) \tag{10}$$

1.1.3 梯度下降

那么什么情况才是$L(\widetilde{W})$最小的时候？我们知道，对一个函数$f(x)$来说，其极大\极小值所在的位置，导数值是0，也就是$\frac{\partial f}{\partial x} = 0$，那么在极大极小值附近的位置导数是什么情况呢？参考下图，在P_0点，我们可以看到，$\frac{\partial f}{\partial x} > 0$，也就是指向其增长的方向：这也正是导数符号的意义，其大小表示了这一点的倾斜程度，那么直观上我们反过来想，与导数符号相反的方向，不就是让函数值下降的方向了吗。



我们可以把这一直观的结论延伸到多元函数$f(X)$中[1]，并用多元函数梯度的概念代替一元函数导数的概念，即对于任意一点$X_0(x_1, x_2, \dots)$，在这点上的梯度为$\nabla f(X_0)$表示了该点最陡峭的方向，以及陡峭的程度，那么，我们沿着这个方向的反向走，也就可以找到一个极小值点（即谷底），梯度的大小（陡峭的程度）稍后就会用到。

现在我们就可以利用梯度下降这个策略来走到一个极小值点：假设目前的权重向量是W_i，我们先计算$\nabla L(W_i)$，然后我们往这个方向的反向走，即：$\widetilde{W}_{i+1} = \widetilde{W}_i - \frac{\nabla L(\widetilde{W}_i)}{\left\|\nabla L(\widetilde{W}_i)\right\|}$，但是这里只是利用了方向信息，我们这一步要更新多少怎么来决定呢？上面一段提到，梯度的大小反应了陡峭的程度：或许越陡峭说明越鼓励往这里多走，因此我们可以简单地设置每一步的步长与$\|\nabla h(W_i)\|$为正比，所以我们的自变量（参数）更新可以表示为：

$$\widetilde{W}_{i+1} = \widetilde{W}_i - \eta * \nabla h(\widetilde{W}_i), \text{其中}\eta\text{是一个超参数，需要人为设定，称为学习率} \tag{11}$$

批梯度下降：当计算损失函数$L(W)$时，我们利用全部样本点的时候，也即损失函数一次计算了$|M|$个点，这时候的梯度下降称为批梯度下降，这时候损失函数往往要平均化处理，即除以$|M|$，否则一次梯度更新的∇L就会一次加上了所有样本处的梯度值，取平均之后，说明这一次的梯度更新考虑了所有的样本点，同时也不会造成梯度更新跨度过大。

随机梯度下降：当我们的损失函数每次计算只涉及到一个样本点的时候，此时损失函数退化为公式（6），这时候的梯度下降称为随机梯度下降，因为每次更新都只和一个点有关。

mini-batch梯度下降：是上面二者的折中方案，即一次取部分样本计算损失函数以及计算平均梯度。

1.1.4 PLA的梯度推导：

这只是个线性模型，可以很简单由向量求导就给出PLA的梯度：

$$\nabla L(\widetilde{W}) = (\frac{\partial L(W)}{\partial W_0}, \frac{\partial L(W)}{\partial W_1}, \ldots, \frac{\partial L(W)}{\partial W_n}) \tag{12}$$

$$= -\sum_{X_i \in M} Y_i * (X_{i1}, X_{i2}, \ldots, X_{i3}) \tag{13}$$

$$= -\sum_{x \in M} Y_i * X_i \tag{14}$$

当用随机梯度下降算法时，流程为：

- 选出一个误分类点(x_i, y_i)
- 迭代更新

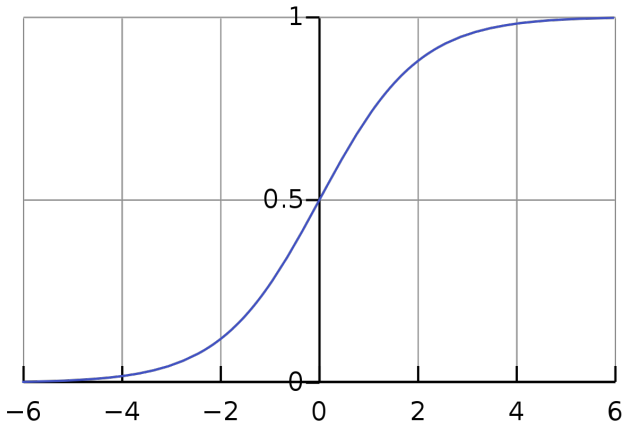
$$\widetilde{W} = \widetilde{W} - (-y_i * x_i) = \widetilde{W} + y_i * x_i \tag{15}$$

1.2 逻辑回归

逻辑回归（LR）同样是与线性变换有关，与PLA不同的是，在输入X经过第一层的线性变换之后产生的输出$O1$，LR将$O1$经过的是一个叫$logistic$的函数：

$$F(x) = \frac{1}{1 + e^{-x}} \tag{16}$$

该函数的样子为：



该函数将值域$\mathbb{R}$映射为$(0, 1)$，因此，通过logistic函数的输出，可以被视作为一种预测概率：

最终的输出为

$$\pi(\widetilde{x}) = \frac{1}{1 + e^{-\widetilde{W}\widetilde{x}}} \tag{17}$$

我们同样通过梯度下降这一手段来对参数$\widetilde{W}$进行计算，因此我们需要定义一个损失函数，用其来评估我们的参数的表现，我们发现，某个样本(x, y)预测正确的概率可以表示为：

$$P(y|x) = \pi(\widetilde{x})^y + (1-\pi(\widetilde{x}))^{(1-y)},\ where\ y \in \{0,1\} \tag{18}$$

因此当所有样本的预测正确概率越高时，这组参数就越好，也就是说，可以将以上概念建模成：

$$\widetilde{W}^* = \arg\max_{\widetilde{W}} \prod_{X_i \in X} P(Y_i|X_i) \tag{19}$$

$$\Leftrightarrow \arg\max_{\widetilde{W}}\ log \prod_{X_i \in X} P(Y_i|X_i) \tag{20}$$

$$= \arg\max_{\widetilde{W}}\ \sum_{X_i \in X} (log(\pi(\widetilde{X_i})) * Y_i + log((1-\pi(\widetilde{X_i}))) * (1-Y_i)) \tag{21}$$

$$= \arg\max_{\widetilde{W}}\ \sum_{X_i \in X} (Y_i * (\widetilde{W}\widetilde{X_i}) - log(1+e^{\widetilde{W}\widetilde{X_i}})) \tag{22}$$

其中，式（20）的等号成立来自于log函数的单调性，并不是函数意义上的等价；当我们定义损失函数为：

$$L(\widetilde{W}) = -log \prod_{X_i \in X} P(Y_i|X_i) \tag{23}$$

转化为求令该损失函数最小化，由此可以运用PLA一节分析的梯度下降算法，来获取权重更新。

$$\nabla L(\widetilde{W}) = \nabla(-\sum_{X_i \in X} (Y_i * (\widetilde{W}\widetilde{X_i}) - log(1+e^{\widetilde{W}\widetilde{X_i}}))) \tag{24}$$

$$= -\sum_{X_i \in X} (Yi * \widetilde{X_i}) + \sum_{X_i \in X} (\frac{\widetilde{X_i} * e^{\widetilde{W}\widetilde{X_i}}}{1+e^{\widetilde{W}\widetilde{X_i}}}) \tag{25}$$

$$= \sum_{X_i \in X} (-Yi * \widetilde{X_i} + \widetilde{X_i} * \pi(\widetilde{X_i})) \tag{26}$$

$$= \sum_{X_i \in X} \widetilde{X_i}(-Yi + \pi(\widetilde{X_i})) \tag{27}$$

因此梯度更新为：

$$\widetilde{W} = \widetilde{W} - \eta \nabla L(\widetilde{W}) \tag{28}$$

$$= \widetilde{W} - \eta \sum_{X_i \in X} \widetilde{X_i}(-Yi + \pi(\widetilde{X_i})) \tag{29}$$

$$= \widetilde{W} + \eta \sum_{X_i \in X} \widetilde{X_i}(Yi - \pi(\widetilde{X_i})) \tag{30}$$

2.伪代码

Algorithm 1 PLApredict(X,W,b)

Input: X :输入数据, W : 权重, b :常数偏置
Output: Y:预测值
reurn Sign(W*X+b)

Algorithm 2 PLAtrain(X, l, W, b, iters, η)

Input: X: 输入 ;l: 标签; $iters$:迭代次数; η:学习率
Output: W, b : 更新后的W,b
$N \leftarrow len(X)$
$iter, i \leftarrow 0$
while $iter \leq iters$ **do**
$y \leftarrow label[i]$
$x \leftarrow X[i]$
//找到误分类点
if predict(x,W,b) != y **then**
$iter \leftarrow iter + 1$
//对W进行梯度下降更新
$dW \leftarrow -y * x$
$W \leftarrow W - \eta * dW$
end if
$i \leftarrow (i + 1)\%N$
end while
return W,b

Algorithm 3 LRpredict(X, W, b)

Input: X :输入数据, W : 权重, b :常数偏置
Output: Y:预测值
$Y1 \leftarrow W * X + b$
$Y \leftarrow \frac{1}{1+e^{-Y1}}$
return Y

Algorithm 4 LRtrain(X, l, W, b, iters, η)

Input: X: 输入 ;l: 标签; $iters$:迭代次数; η:学习率
Output: W, b : 更新后的W,b
$N \leftarrow len(X)$
$iter, i \leftarrow 0$
for $iter \leftarrow 0$ to iters **do**
$p \leftarrow$ predict(X,W,b)
//对W进行梯度下降更新
$dW \leftarrow \sum_{i=0}^{N} X_i * (-l_i + p_i)$
$dW \leftarrow \frac{dW}{N}$
$W \leftarrow W - \eta * dW$
end for
return W,b

3.关键代码分析

3.1 PLA模型

3.1.1 预测模块

预测包括了一次线性变换$Y = W * X + b$以及根据Y的正负性进行分类$\{-1, 1\}$，其中$y = 2x - 1$实现的是$\{0, 1\}$->${-1,1}的转换。

```
def signV(X:Union[np.ndarray, pd.DataFrame]):
    return 2*(X>0).astype('int')-1

  def predict(self, datas):
    return signV(datas[:,:self.dims].dot(self.W[:-1])+self.W[-1])
```

3.1.2 训练模块($code/pla.py(54-69)$)

开始遍历所有训练数据，逐个进行预测，$iter$记录迭代次数（这里理解为修正的误分类点的次数），$error_num$是记录对所有数据的一次完整遍历中的错误次数，当一次完整遍历完成时计算错误率，当小于阈值时即终止迭代（16行）；根据上述分析可知，更新的原理为：$dW = -y_i * x_i$，$W = W - \eta * dW = W + y_i * x_i$（14行），其中$x_i, y_i$为错误分类点，上述公式中，$y_i$的值需为$\{-1, 1\}$，因此在判断预测的时候，需要将给定的label值$\{0, 1\}$转换为$\{-1, 1\}$，因此这里用了线性变换$y = 2x - 1$完成这一映射（14行）。

```
iter = 0
while (iter < iters):
  error_num = 0       # record the misclassification times over each look of the whole dataset
  # training on all dataset
  for epoch in range(len(trainset)):
    data, label = trainset[epoch], labels[epoch] # get one single data
    if iter >= iters:
      break
      # if misclassification -> update iteration times, error times and W
      if self.predSingle(data) != 2*label-1:  # convert{0,1}->{-1,1} using map: y=2x-1
        iter += 1
        error_num += 1
        # dW = - y_i*x_i
        self.W += (lrate*(2*label-1)*data[:self.dims]).reshape(-1,1)
        # while error_rate in trainset smaller than given threshold, BREAK
        if error_num/len(trainset) <= error_rate : break
```

3.2 LR模型

3.2.1 预测模块($code/lr.py(63-78)$)

预测包括了两部分：线性变换$Y=W*X+b$、logistic输出$prob=1/(1+e^{-Y})$，predict预测函数将根据当前LR模型是否处于训练状态决定输出的是概率$prob$还是先转化为标签值$\{0,1\}$再输出，阈值为0.5

```
1  def Fullnet(self, X):
2    return X[:,:self.dims-1].dot(self.W[:-1]) + self.W[-1]
3
4  def logistic(self,X):
5    return 1/(1+np.exp(-X))
6
7  def predict(self, datas:Iterable, label:Union[np.ndarray, None]=None):
8    prob = self.logistic(self.Fullnet(datas))
9    if self.trainMode:
10     return prob
11   else:
12     predLable = (prob > 0.5).astype('int')
13     if label is not None:
14       label = label.reshape((-1,1))
15       return predLable, (predLable == label).mean()
16     else: return predLable
```

3.2.2 训练模块($code/lr.py(51-60)$)

LR模块用的是批梯度下降进行参数更新，这里的批大小为$batchsize$，训练流程为：获取每一批次的数据$miniX$，然后预测他们的输出概率Y，计算Loss作为损失记录（非必需），然后根据上述分析的$dW=-X*(label-Y)$，$W=W-\eta*dW$（7行）进行梯度更新。

```
1  for iter in range(iters):
2    for miniX, minilabel in trainset.Batch(batchsize=batchsize):   # get minibatch
   data
3      # Y = np.maximum(np.minimum(self.predict(miniX), 0.95), 0.05)
4      Y = self.predict(miniX)          # get forward pass output (probability of
   intput)
5      Loss.append(self.Loss(Y, minilabel))
6      ac = ((Y > 0.5).astype('int') == minilabel).mean()
7      dW = miniX.T.dot(Y-minilabel)    # dW = - X*(label - Y)
8      dW /= len(miniX)
9      self.W -= lrate*dW               # W -= lrate*dW
```

3.3 数据划分

这次仍然是用K折交叉验证的方法来评定模型，因此数据集将被随机打乱，然后分成K份，取出1份作为验证集，其余作为训练集。

为了方便起见，将K折数据划分以及LR模型中采用的miniBatch将数据分批的方法封装在类DataLoader中，其原理非常简单，代码如下（code/util.py）：

```
class DataLoader:
    def __init__(self, datas:Iterable) -> None:
        self.datas = datas

    def KfolderData(self, K:int, shuffle=False):
        # data should include labels
        if shuffle : np.random.shuffle(self.datas)
        numOfPart = int(len(self.datas)/K)
        for i in range(K):
            trainset = np.append(self.datas[numOfPart*(i+1):, :],
self.datas[:numOfPart*i,:], axis=0)
            valset = self.datas[numOfPart*i:numOfPart*(i+1),:]
            yield trainset[:,:-1], trainset[:,-1], valset[:,:-1], valset[:,-1]

    def Batch(self, batchsize=100, shuffle=True, throw=True):
        # with label
        if shuffle: np.random.shuffle(self.datas)
        n = len(self.datas)/batchsize
        n = int(n) if throw else np.ceil(n) # whether throw away the last set

        for i in range(n):
            yield self.datas[batchsize*i:batchsize*(i+1),:-1],
self.datas[batchsize*i:batchsize*(i+1),-1].reshape(-1,1)
```

4.实验结果与分析

4.1 PLA

设置K折交叉验证的K为7，迭代次数=100，学习率设置为[0.01, 0.1, 1]:

```
 56044 -- /Users/liuxb/2021_au/al_rao/lab/lab3/code/pla.py
the average ac with lrate 0.001000 is 0.696022
the average ac with lrate 0.010000 is 0.651989
the average ac with lrate 0.100000 is 0.629722
the average ac with lrate 1.000000 is 0.682762
```

发现学习率在0.001以及1是较佳值，学习率的变化引起了准确率的波动，但是并没有明显的规律，这一现象可能与采取随机梯度下降的策略有关（因为随机梯度下降的更新不稳定，较大/较小的步长没有特别的规律）

4.2 LR

设置K折交叉验证的K为7，迭代次数=100，学习率设置考虑[0.01,0.1,1]，运行结果为：

```
average ac of lrate:0.010000 is 0.777333
average ac of lrate:0.100000 is 0.751689
/Users/liuxb/2021_au/al_rao/lab/lab3/code/lr.py:85: RuntimeWarning: divide by zero encountered in log
  return -(label.T.dot(np.log(Y))+(1-label).T.dot(np.log(1-Y))) / len(Y)
average ac of lrate:1.000000 is 0.673755
```

学习率在0.01的时候表现较佳，随着学习率上升，准确率有一个下降的趋势，而且在学习率较大时（=1）发现了一个除0的warning，这是因为logistic函数分母的指数项趋向0，即WX趋于无限大，这可能是因为学习率过大，导致梯度更新幅度太大，从而增长非常迅速，也可能在这个过程中远离了极值点。

5.思考题

5.1 随机梯度下降与批量梯度下降各自的优缺点?

随机梯度下降中，由于每次仅用单个数据进行梯度更新，因此可以预见的优势就是：处理速度快，梯度更新快；但同时，正如其名字所示，“随机”取样，意味着每次更新会用一个风险：取到了噪声样本，但仍然对待正常数据一样处理，因此梯度的更新可能是不稳定的。

批量梯度下降每次更新梯度因为考虑到了所有样本，所以缓解梯度更新不稳定这一现象，几乎总是朝着正确的方向去更新，但是也正如“批量”这个名字一样，完整处理一个大样本的时间消耗可能会很大，也就是说，梯度更新可能是缓慢但稳健的。

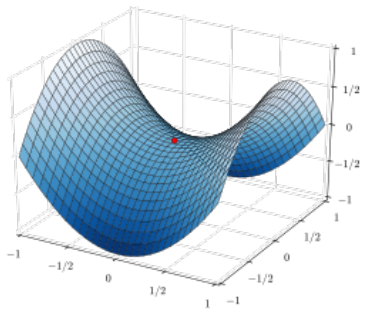
5.2 不同的学习率η对模型收敛有何影响？从收敛速度和是否收敛两方面来回答。

学习率大，意味着每次梯度更新的步长增加，收敛速度会加快，但是也正因为跨度大，所以很有可能在函数的“谷底”附近来回震荡，但就是不会落入谷底（跨越了极值点穿到了另一侧）；所以学习率高可能可以很快到达谷底附近，但此后陷入震荡，无法收敛。

反之，学习率低，步长小，收敛速度会变慢；但因为步长小，所以在极值附近但跨越了极值的可能性降低，也就是几乎总是能收敛的；但是同样有风险：更容易陷入不优的局部最小值（因为步长小，所以无法跨出）。

5.3 使用梯度的模长是否为零作为梯度下降的收敛终止条件是否合适，为什么？一般如何判断模型收敛?

不完全合适。梯度模长为0，意味着梯度向量平行于特征维度的超平面，但是此处不总是一个极值点：我们可以借助一维的$f(x) = x^3$的零点以及二维平面如下图所示的函数[2]来理解（saddle point）：



这些点的梯度/导数模长都为0，但是显然不是一个极值点。

可以通过判断Loss的值是否足够小来判断模型收敛，另外可以设置迭代的次数（也就是本文用的方法）来限制深度。

6.缺点

这次实验因为时间原因导致未能具体探讨模型的各种超参数对模型表现的影响，只稍微对学习率的变化造成的差异进行了一个主观的叙述；同时也没能用上更好的算法来优化模型。

7.Reference

1. https://zh.wikipedia.org/wiki/%E6%A2%AF%E5%BA%A6 ↩

2. https://en.wikipedia.org/wiki/Saddle_point ↩