中山大學 SUN YAT-SEN UNIVERSITY 实验报告

学院：数据科学与计算机学院　　专业：计算机科学与技术　　科目：人工智能
学号：17341213　　姓名：郑康泽

一. 算法原理

1. 感知机学习算法（PLA）

1）算法介绍：

PLA 是针对于二分类{+1, -1}问题的一个方法，通过划分一个超平面来区分数据，并且根据超平面的线性性质可以得出，PLA 不适用于划分非线性可分的数据集

2）算法数学描述：

通过一个共享的权重向量 $\boldsymbol{w} = (w_1, w_2, \ldots, w_d)$和某个样例的特征向量 $\boldsymbol{x} = (x_1, x_2, \ldots, x_d)$来计算该样例的分数，通过与某个阈值 θ 比较大小，来判断样例的类别：

$$\text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \theta\right)$$

将阈值 θ 转化成模型待学习的参数：

$$\text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \theta\right)$$

$$= sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) + (-\theta) \times (+1)\right)$$

$$= sign\left(\sum_{i=0}^{d} w_i d_i\right)$$

$$= sign(\widetilde{\boldsymbol{W}}^T \widetilde{\boldsymbol{X}})$$

其中：

$$\widetilde{W} = (w_0, w_1, w_2, \ldots, w_d)$$
$$\tilde{X} = (+1, x_1, x_2, \ldots, x_d)$$

3）算法步骤：

a）给每个样本的特征向量前加一维常数项 1

b）随机初始化$(\mathrm{d}+1)$维的权重向量$\widetilde{\boldsymbol{W}}_0$

c）遍历训练样本，每当找到一个预测错误的样本 $\boldsymbol{x}_i$，则更新权重向量，直到所有的训练样本都预测正确

$$\widetilde{\boldsymbol{W}}_{t+1} \leftarrow \widetilde{\boldsymbol{W}}_t + y_i \widetilde{\boldsymbol{x}}_i$$

d）通过$\text{sign}(\widetilde{\boldsymbol{W}}_{final} \widetilde{\boldsymbol{x}})$来预测一个样本 $\mathbf{x}$ 的标签

4）算法改进：

由于 PLA 是不适用于非线性可分的数据集的，而现实中许多数据集也不是线性可分的，所以需要其他方法来解决这个问题：

a) 设置迭代次数，到一定程度就返回此时的$\widetilde{W}$，不管它到底满不满足所有训练集；

b) 找一个$\widetilde{W}$，使得在训练集里以此$\widetilde{W}$来划分后，分类错误的样本最少，即相当于有一个口袋有一个$\widetilde{W}$，把算到的$\widetilde{W}$和口袋里的$\widetilde{W}$比对，放入比较好的一个$\widetilde{W}$，这种算法又被称为口袋算法。

2. 逻辑回归算法（LR）

1) 算法介绍：

LR 是一种软分类模型，通过计算每个分类的概率，然后比较概率大小来判断样例的类别

2) 算法数学描述：

a) 与 PLA 类似，通过一系列权重来计算某个样例的分数：$s = \sum_{i=0}^{d} w_i x_i = \widetilde{W}^T \tilde{x}$

b) 计算出来的分数 s 的范围是$(-\infty, +\infty)$，我们利用 *logistics function* 来将 s 映射到$[0, 1]$，公式如下：

$$\theta(s) = \frac{e^s}{1+e^s} = \frac{1}{1+e^{-s}}$$

当 s 无穷小时，该数据属于正类别的概率为 0；当 s=0 时，该数据属于任一类别的概率相同；当 s 无穷大时，该数据属于正类别的概率为 1

c) 利用逻辑回归函数构建预测函数 $h(\tilde{x})$，$h(\tilde{x})$相当于样本属于正类的概率，属于负类的概率为 $1- h(\tilde{x}) = h(-\tilde{x})$

$$h(\tilde{x}) = \theta(s) = \frac{1}{1+e^{-\widetilde{W}^T \tilde{x}}}$$

d) 利用最大似然估计取负对数，作为目标函数$L(\widetilde{W})$

$$L(\widetilde{W}) = -\sum_{n=1}^{N} (y_i \log(h(\tilde{x}_i)) + (1-y_i)\log(1-h(\tilde{x}_i)))$$

e) 使用梯度下降来优化问题，通过对目标函数来获得梯度函数：

$$\nabla L(\widetilde{W}) = -\sum_{n=1}^{N} (y_n - h(\tilde{x}_n))(\tilde{x}_n)$$

则权重的更新公式为：

$$\widetilde{W}_{t+1} \leftarrow \widetilde{W}_t - \eta \nabla L(\widetilde{W}_t)$$

3) 算法步骤：

a) 给每一个样本的特征向量加一维常数项 1

b) 随机初始化$(d+1)$维的权重向量$\widetilde{W}_0$

c) 计算当前梯度$\nabla L(\widetilde{W}_t) = -\sum_{n=1}^{N} (y_n - h(\tilde{x}_n))(\tilde{x}_n)$

d) 根据梯度更新权重$\widetilde{W}_{t+1} \leftarrow \widetilde{W}_t - \eta \nabla L(\widetilde{W}_t)$

e) 重复步骤 c)和 d)直到满足一定的收敛条件

f) 根据模型最后的权重，通过h($\tilde{x}$)的概率取值来预测某个样例 **x**

二. 伪代码

1. 感知机学习算法(PLA):

```
Procedure PLA(data, label, train_times)
input: data[i]represents x̃, and label[i] representss corresponding class(0 or 1), and
train_times represents how many times you want to train
output: W̃

n ← the number of columns of data
m ← the number of rows of data
w ← initialize to a vector of n 0s
for i ←1 to train_times
    for j ← 1 to m
        if sign(w * data[j]) != label[j]          // '*' is vector product
            then w += data[j] * label[j]
                break
return w
```

2. 逻辑回归算法(LR):

```
Procedure LR(data, label, train_times)
input: data[i]represents x̃, and label[i] representss corresponding class(0 or 1), and
train_times represents how many times you want to train
output: W̃

n ← the number of columns of data
m ← the number of rows of data
w ← initialize to a vector of n 0s
for i ← 1 to train_times
    gradient ← 0
    for j ← 1 to m
        gradient += (1/(1+e^(-data[j]·w)) − label[j]) * data[j]
    w -= gradient

return w
```

三. 代码解释

1. 感知机学习算法(PLA)

1) 划分数据集为训练集和验证集，经过测试，1/4 为验证验证机集，3/4 为

训练集的分法，可以使得训练出来的$\widetilde{w}$在验证集上获得最高的准确率

```
def division(read_path, train_path, test_path, k):
    '''
    :param read_path: 数据集路径
    :param train_path: 训练集路径
    :param test_path: 验证集路径
    :param k: 划分数据为k份
    :return: 空
    '''
    train_data = []
    test_data = []
    # 读取数据集
    total_data = pd.read_csv(read_path, header=None)
    total_data = np.array(total_data)
    # np.random.shuffle(total_data)
    num = len(total_data)
```

```
    # 划分数据集
    for i in range(int(num/k)):
        test_data.append(total_data[i])
    for i in range(int(num/k), num):
        train_data.append(total_data[i])
    # 写入
    train_data = pd.DataFrame(train_data)
    test_data = pd.DataFrame(test_data)
    train_data.to_csv(train_path, header=False, index=False, line_terminator='\n')
    test_data.to_csv(test_path, header=False, index=False, line_terminator='\n')
```

2）读取训练集和验证集，并把每个特征向量加工为增广特征向量以及将标签为 0 转为标签为-1

```
def get_data(path):
    '''
    :param path: 文件路径
    :return: 加工后的增广特征向量和修改后的标签
    '''
    # 读取文件
    data = pd.read_csv(path, header=None)
    data = np.array(data)
    # 第一列插入一
    data = np.insert(data, obj=0, values=1, axis=1)
    # 将标签0改为-1
    label = data[:, NUM_OF_FEATURE+1]
    for i in range(len(label)):
        if label[i] == 0:
            label[i] = -1
    return data[:, :NUM_OF_FEATURE+1], label
```

3）统计训练出来的$\widetilde{w}$在验证集上正确率

```python
def test(test_path, w):
    '''
    :param test_path: 验证集的路径
    :param w: 训练得到的权重w
    :return: 在验证集上的准确率
    '''
    # 获取数据
    data, label = get_data(test_path)
    # 统计预测正确的个数
    num = len(data)
    correct = 0
    for i in range(num):
        predict = np.sign(np.dot(w, data[i]))
        if predict == label[i]:
            correct += 1
    return float(correct) / float(num)
```

4）简单版 PLA

```python
def train(train_path, test_path, train_time):
    '''
    :param train_path: 训练集的路径
    :param test_path: 验证集的路径
    :param train_time: 训练次数
    :return: 无
    '''
    # 初始化权重向量
    w = np.zeros(NUM_OF_FEATURE + 1)
    # 获取数据
    data, label = get_data(train_path)
    # 训练
    num = len(data)
    # index = [i for i in range(num)]
    for i in range(train_time):
        # random_index = random.sample(index, 500)
        # for j in random_index:
        #     if np.sign(np.dot(w, data[j])) != label[j]:
        #         w += 10 * label[j] * data[j]
        for j in range(num):
            # 发现分类错误的数据
            if np.sign(np.dot(w, data[j])) != label[j]:
                w += label[j] * data[j]
                break
        y.append(test(test_path, w))
        # 每10次输出准确率
        if i > 0 and i % 10 == 0:
            print('train step:', i)
            print('accuracy:', '{:.2%}'.format(test(test_path, w)))
```

5）改进版 PLA：每次找到错误分类的数据，不断尝试修改$\widetilde{\boldsymbol{w}}$的权重直至在验证集上的准确率有所提升

```
def train(train_path, test_path, train_time):
    '''
    :param train_path: 训练集的路径
    :param test_path: 验证集的路径
    :param train_time: 训练次数
    :return: 无
    '''
    # 初始化权重w
    old_w = np.zeros(NUM_OF_FEATURE + 1)
    # 获得数据
    data, label = get_data(train_path)
    # 训练
    num = len(data)
    # index = [i for i in range(num)]
    old_accuracy = 0
```

```
    for i in range(train_time):
        # random_index = random.sample(index, 500)
        # for j in random_index:
        learning_rate = 1
        for j in range(num):
            # 找到分类错误的数据
            if np.sign(np.dot(old_w, data[j])) != label[j]:
                # 找到适合修改权重向量w的权重learning_state直到在验证集上准确率有所提高
                new_w = old_w + learning_rate * label[j] * data[j]
                # new_w = w + learning_rate * label[j] * data[j]
                new_accuracy = test(test_path, new_w)
                iterative_time = 0
                while iterative_time <= 10 and new_accuracy < old_accuracy:
                    learning_rate *= 0.1
                    new_w = old_w + learning_rate * label[j] * data[j]
                    new_accuracy = test(test_path, new_w)
                old_accuracy = new_accuracy
                old_w = new_w
                break
```

```
        # 每100次输出准确率
        if i % 100 == 0:
            print('train step:', i)
            print('accuracy: {:.2%}'.format(old_accuracy))
            print()
```

2. 逻辑回归算法(LR)

1) 读取训练集和验证集的数据，并将特征向量加工为增广特征向量

```
def get_data(path):
    '''
    :param path: 文件路径
    :return: 加工后的增广特征向量和修改后的标签
    '''
    # 读取文件
    data = pd.read_csv(path, header=None)
    data = np.array(data)
    # 第一列插入1
    data = np.insert(data, obj=0, values=1, axis=1)
    label = data[:, NUM_OF_FEATURE+1]
    return data[:, :NUM_OF_FEATURE+1], label
```

2) 简单版 LR

```
def train(train_path, test_path, train_time):
    '''
    :param train_path: 训练集的路径
    :param test_path: 验证集的路径
    :param train_time: 训练次数
    :return: 无
    '''
    # 初始化权重向量w
    w = np.zeros(NUM_OF_FEATURE + 1)
    # 获取数据
    data, label = get_data(train_path)
    # 训练
    num = len(data)
    learning_rate = 0.00001
```

```
    for i in range(train_time):
        # 下降梯度
        gradient = 0.0
        for j in range(num):
            res = np.dot(w, data[j])
            if res >= 0:
                res = (1.0 / (1.0 + np.exp(-res)) - label[j]) * data[j]
            else:
                res = (np.exp(res) / (1.0 + np.exp(res)) - label[j]) * data[j]
            gradient += res
        # 更新
        w -= learning_rate * gradient
        accuracy = test(test_path, w)
        # 每100次输出准确率
        if i % 100 == 0:
            print('train step:', i)
            print('learning_rate:', learning_rate)
            print('accuracy:', accuracy)
            print()
```

3）改进版 LR：学习率过大就会震荡，特别是快到目标函数最低点的时候，所以在训练过程中，如果遇到正确率下降的时候，减小学习率，使得学习率不会震荡。

```
def train(train_path, test_path, train_time):
    '''
    :param train_path: 训练集的路径
    :param test_path: 验证集的路径
    :param train_time: 训练次数
    :return: 无
    '''
    # 初始化权重向量w
    w_old = np.zeros(NUM_OF_FEATURE + 1)
    # 获取数据
    data, label = get_data(train_path)
    # 训练
    num = len(data)
    learning_rate = 1.0
    old_accuracy = 0.0
```

```python
for i in range(train_time):
    gradient = 0.0
    for j in range(num):
        res = np.dot(w_old, data[j])
        if res >= 0:
            res = (1.0 / (1.0 + np.exp(-res)) - label[j]) * data[j]
        else:
            res = (np.exp(res) / (1.0 + np.exp(res)) - label[j]) * data[j]
        gradient += res
    # 找到能使正确率上升的学习率
    w_new = w_old - learning_rate * gradien
    new_accuracy = test(test_path, w_new)
    while new_accuracy < old_accuracy:
        learning_rate *= 0.1
        w_new = w_old - learning_rate * gradient
        new_accuracy = test(test_path, w_new)
    w_old = w_new
    old_accuracy = new_accuracy
    # 输出正确率
    print(learning_rate)
    print('train step:', i,  '  accuracy:', '{:.2%}'.format(new_accuracy))
    print()
```

四. 实验结果以及分析

1. 感知机学习算法(PLA)模型分析：

1）简单版 PLA 的类别分布：

```
train step: 25
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 741
the number of predict 1 1259
accuracy: 57.30%

train step: 50
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1452
the number of predict 1 548
accuracy: 60.05%

train step: 75
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 661
the number of predict 1 1339
accuracy: 65.60%

train step: 100
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1022
the number of predict 1 978
accuracy: 67.05%
```

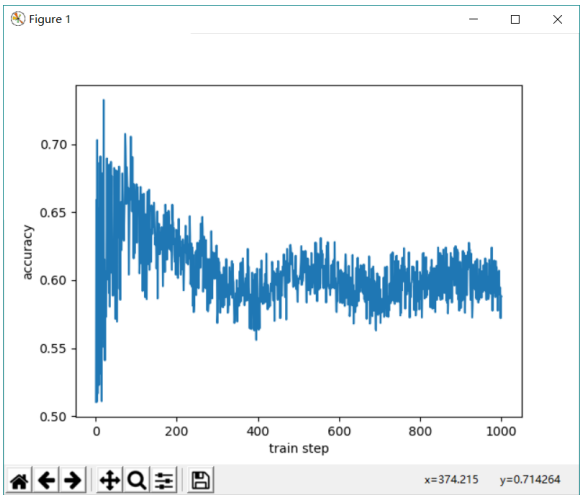
因为训练次数太少，看不出有明显的震荡现象，但是从类别的分布可以看出，1 标签和-1 标签的个数一直在交替，这可能是震荡的原因。至于为什么会震荡，是因为目标函数计算的不是整体的损失，而是单个的损失，所以根据此目标函数调整后，不一定会减少整体的损失。除非数据集线性可分，否则 PLA 是不会收敛的。

改进版 PLA 的类别分布：

```
train step: 3
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 877
the number of predict 1 1123
accuracy: 74.60%

train step: 6
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1068
the number of predict 1 932
accuracy: 76.15%

train step: 9
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1066
the number of predict 1 934
accuracy: 76.25%

train step: 12
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1041
the number of predict 1 959
accuracy: 76.30%

train step: 15
the number of real -1: 1021
the number of real 1: 979
the number of predict -1: 1041
the number of predict 1 959
accuracy: 76.30%
```

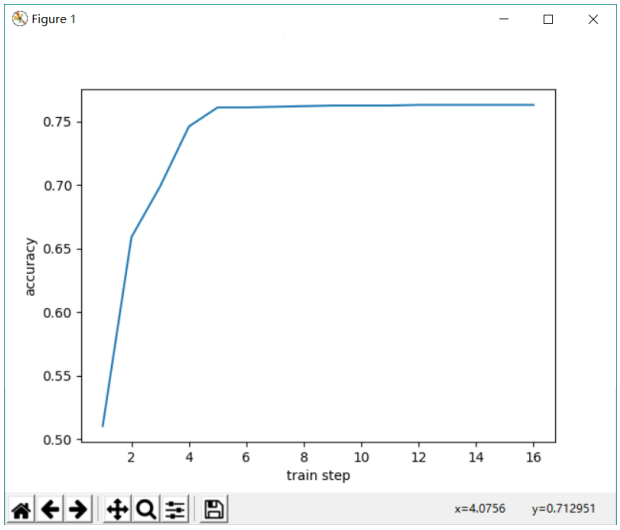
改进版的 PLA 大概在训练次数 12 次左右的时候，就找不到能使准确率网上升的修改$\widetilde{\boldsymbol{W}}$的权重了，所以准确率就停在了 76.30%了，但是明显比简单版的 PLA 的效果好。

2）简单版 PLA 的准确率随训练次数增加而变化的图：



可以看出，PLA 是不会收敛的，原因上面也讲过了。

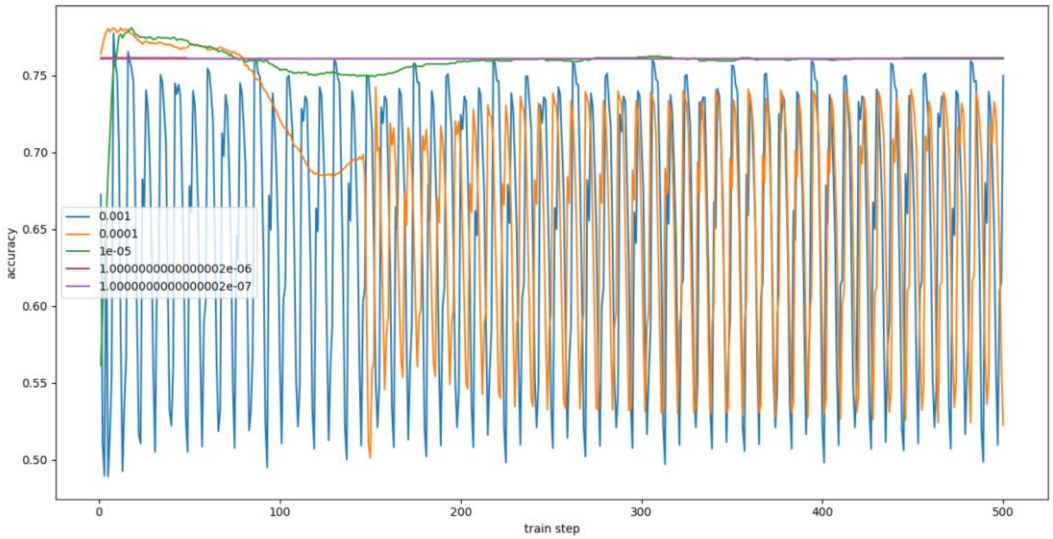
改进版 PLA 的准确率随训练次数增加而变化的图：



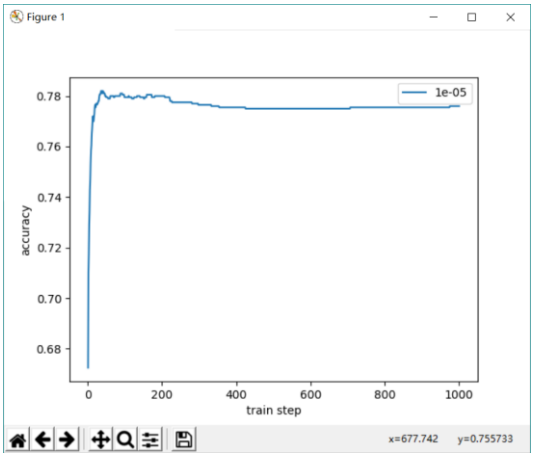
根据这张图，可以验证大概在训练 10 次后没法再提高准确率

2. 感知机学习算法(PLA)模型分析：

1）简单版 LR 的准确率随训练次数及学习率变化的图像：

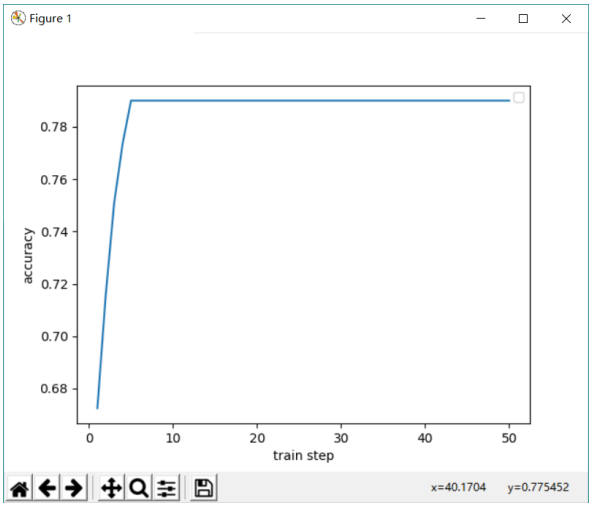


从图中可以看出，如果学习率为 0.001 的话，会产生震荡，无法收敛；学习率小点就可以收敛，但太小就会收敛慢，经过测试，学习率为 1e-5 最好，当学习率为 1e-5 的时候，图像如下：



准确率收敛在大概 78%这里

2) 改进版 LR 的准确率随训练次数变化的图像：



效果比简单版的好一点，准确率能达到 79%

五． 思考题

1. **有什么手段可以使 PLA 适用于非线性可分的数据集？**
   答：PLA 是一种简单的单层神经网络，可以在输入层与输出层中间加一层隐藏层，利用隐藏层在数据转换为线性可分的，然后再用 PLA 划分。
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。**
   答：η大的话，一开始收敛可能会比较快，但到最值点附近会因为η大的原因而不收敛，并出现震荡；η小的话，收敛速度会比较慢，但终归还是会收敛的，一般不震荡。
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？**
   答：不合适，梯度模长为 0 意味着目标函数达到最值点，但这一般没这么幸运，所以以梯度模长为 0 位终止条件可能会导致永远无法停下来。如果要判断模型收敛，可以判断前后几次的目标函数的值之差是否都在一定小的范围内，如果是，可以判断为已收敛。