环境

```
OS: Manjaro 18.0.2 Illyria
Kernel: x86_64 Linux 4.19.13-1-MANJARO
python 3.7
numpy 1.15
matplotlib 3.0.2
networkx 2.2
jupyter 4.4.0
```

注：以下的程序划分仅仅是为了便于检查，实际的代码运行使用jupyter notebook. 程序为`Lab2.ipynb`打开后使用`Kernel-->Restart & Run All`

Part I

数据描述

选用的数据为Gas sensors for home activity monitoring Data Set

这个数据集是在不同的环境(background/banana/wine)中，计量8个感受器的测量值。以及另外附加的温度和湿度感受器。

总共的实例大小为919438，属性选择8个sensor的测量数据。

本实验的目的是由几个感受器测量的数据判断是属于哪个环境，为了简便起见，仅仅选用banana/wine的二分类任务。

预处理

1. 读取数据。这一点由于数据集本身的格式比较简单，所以可以直接使用`np.loadtxt`

   ```
   metadata=np.loadtxt("./data/HT_Sensor_UCIsubmission/HT_Sensor_metadata.dat",skiprow
   s=1,dtype=str)
   dataset = np.loadtxt('./data/HT_Sensor_UCIsubmission/HT_Sensor_dataset.dat',
   skiprows=1)
   ```

2. 将文字表述的类别改成数字。

3. 选定特定时间区间的数据。

   这是由于本身给定的数据是先在空环境中测量，然后放入特定的物品，然后过一段时间撤掉物品。所以有效的时间区间就是那段放入特定物品的区间。预处理可以是使用`matadata`给出的时间描述来抽取。

4. 随机化。

   由于测量的数据，是按照某种时间顺序来记录的，所以需要将数据随机化，避免过度的关联。

   ```
   np.random.shuffle(data)
   ```

5. 归一化。

   由于不同的感受器测量得到的数据并不是统一在某个区间，而是相对于自身的数据有不同的偏重。所以归一化能避免不同的度量大小对数据的影响。

```
data=(data-data.mean(axis=0))/data.std(axis=0)
```

6. 划分训练集和测试集；这一点我放在了KNN/Logistic Regression的部分

```
X_test=data[:test_size,2:10]
y_test=data[:test_size,0].astype('int')

X_train=data[test_size:,2:10]
y_train=data[test_size:,0].astype('int')

print(X_test.shape,y_test.shape,X_train.shape,y_train.shape,data.shape)
```

KNN

1. 算法

   划分的训练集已知其label,然后对测试集中的每个点，都计算它与训练集中点的距离，然后找到最近的k个，来判断属于哪一个。

```
for x in test_X:
    NN={distance(x,t) for t in train_x}
    kNN=NN.min(k)
    x.label=kNN.most_common_label
```

```
  class KNN():
      def __init__(self, k=5):
          self.k = k

      def predict(self, X_test, X_train, y_train):
          y_pred=np.empty(X_test.shape[0],dtype=int)
          for i,X in enumerate(X_test):
              if i%100 ==0:
                  print(i)
              y_pred[i]=np.bincount(y_train[np.argsort(np.linalg.norm(X-
X_train,axis=1))[:self.k]]).argmax()
          return y_pred
```

使用`numpy`还是比较简单的矩阵计算。

2. 结果

   由于数据集过于庞大，KNN测试非常非常慢，而且得到的结果非常准确。在使用k=3，仅仅测试1000的点，准确率达到100%. 于是决定减小训练集为整个数据集的1/2, 然后将k=1, 相当于最临近邻居。这样测试集数量减小了，而且训练速度更快了，但还是整整跑了一个中午。最后得到的结果如下。

预测的代码为

```
model=KNN(1)
y_pred=model.predict(X_test[:],X_train[:],y_train)
result=Counter(np.equal(y_pred[:],y_test[:])).most_common()
print(result)
```

结果为

```
[(True, 41638), (False, 7)]
```

可以说准确率非常高，又鉴于测试速度过于缓慢，就没有必要继续往下测试。

Logistic Regression

1. 算法

具体算法也就是对$\frac{1}{1+e^{-w^\top x}}$求一个loss函数，然后使用梯度下降，最终得到一个更新w的式子为

$$\mathbf{w} = \mathbf{w} + \alpha \sum_{i=1}^{N} \left[\left(y_i - \sigma(\mathbf{w}^\top \mathbf{x}_i) \right) \mathbf{x}_i \right]$$

最终得到的算法为

```
initialize(W)
while not converge:
    W=updata_according_to_the_formula_above(W)
```

```
LOSS=[]
class LogisticRegression():
    def __init__(self,lr=0.1):
        self.lr=lr

    def sigmoid(self,Z):
        return 1/(1+np.exp(-Z))

    def loss(self,y,y_hat):
        return -np.mean(y * np.log(y_hat)+(1-y)*np.log(1-y_hat))

    def fit(self,X_train,y_train,epochs=5000):
        limit=1/math.sqrt(X_train.shape[1])
        self.W=np.random.uniform(-limit,limit,(X_train.shape[1],))

        for i in range(epochs):
            y_hat=self.sigmoid(X_train @ self.W)
            self.W -= self.lr * (X_train.T @ (y_hat - y_train) / y_train.shape[0])
            temp_loss=self.loss(y_train,y_hat)
            LOSS.append((i,temp_loss))
            if i %100 ==0:
```

```
                print(i,temp_loss)

    def predict(self,X_test):
        y_pred=self.sigmoid(X_test @ self.W)>0.5
        return y_pred.astype('int')
```

2. 结果

   Logistic Regression得到的结果不理想。

   迭代5000次，得到的结果为

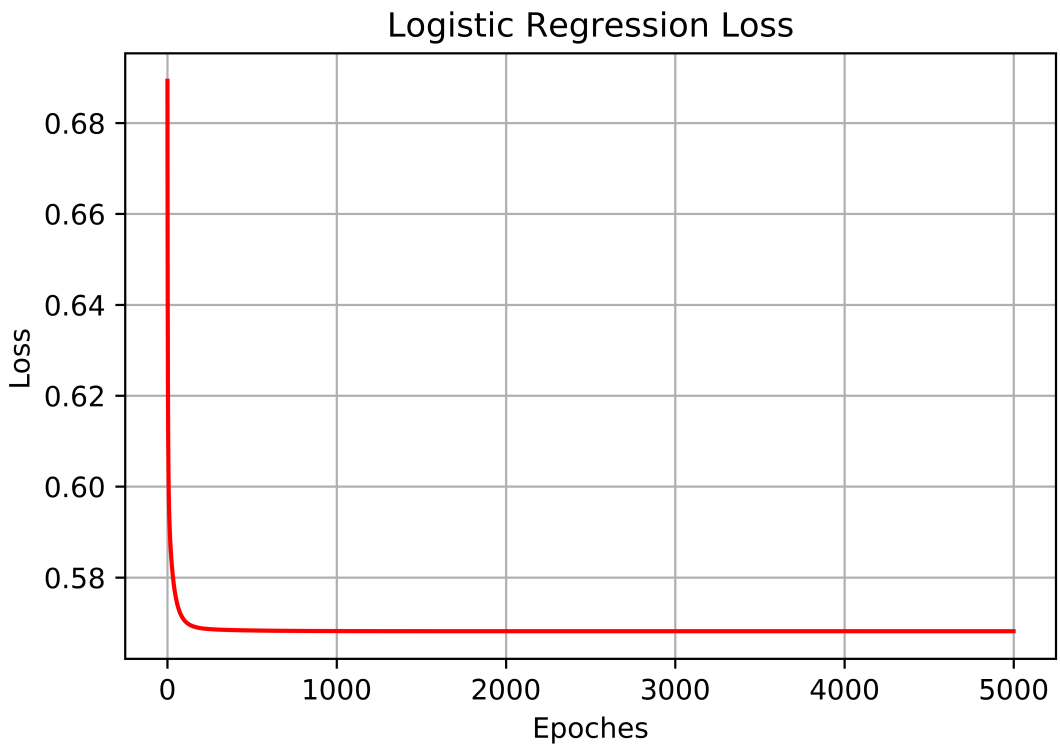
```
model=LogisticRegression(0.25)
model.fit(X_train,y_train,5000)
y_pred=model.predict(X_test)

Counter(y_pred == y_test).most_common()
```

   最终的结果为

```
[(True, 28829), (False, 12816)]
```

   准确率为69.23%

   另外得到的损失函数的大小为



   可以看到基本上已经没有下降的空间了，证明训练已经完全。

结果讨论

1. KNN的准确度明显比Logistic Regression(LR)高很多很多

2. KNN的测试速度比LR慢很多。这是由于KNN要与训练集中的每个点来计算距离，然后得到结果，而LR仅仅计算一个函数，参数都训练出来的。
3. 可以猜想KNN的准确率很高的原因是与每个点比较，已经完全获知了分类的信息。而LR在本实验中仅仅只有8个参数，完全不足以划分数据集，或者说，数据集本身并不能用一个超平面划分出来。这一点联想到深度学习，也许可以用多层的w参数来拟合分类。