- 按照`实验二要求.pdf`修改。
- 基本内容已经给出了，这个链接的前半部分是我自己的Part1
- 把环境修改成自己的环境
- 可以把KNN/Logistic Regression的伪代码部分写详细一点
- 如果要跑代码，使用jupyter notebook跑`src.ipynb`
- 删掉以上内容

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
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**Electrical Grid Stability Simulated Data Data Set**

这个数据集是探究在4个节点的星状供电网络的稳定性问题。

总共的实例大小为10000，属性为11个，然后一个label表明是稳定还是不稳定。

本实验的目的是预测在给定属性的条件下，系统是否稳定。

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`

```
data=np.loadtxt("./data/PartI/Data_for_UCI_named.csv",delimiter=",",skiprows=1,dtyp
e=str)
#"tau1","tau2","tau3","tau4","p1","p2","p3","p4","g1","g2","g3","g4","stab","stabf"
#0       1      2       3     4    5    6    7    8     9   10   11   12     13

data=data[:,[0,1,2,3,5,6,7,8,9,10,11,12]].copy()
#"tau1","tau2","tau3","tau4","p2","p3","p4","g1","g2","g3","g4","stab",
#   0     1     2      3      4    5    6    7    8    9    10   11
```

2. 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)
```

3. 归一化与划分数据集

   由于不同的属性值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。

   划分为X/y(abel)

```
X=data[:,:11].copy().astype("float")
y=data[:,11].astype("float")>0
X=(X-X.mean(axis=0))/X.std(axis=0)

np.save("./data/PartI/X.npy",X)
np.save("./data/PartI/y.npy",y)
```

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

   首先随机定一个k=3得到的结果如下。

   预测的代码为

   ```
   model=KNN(3)
   y_pred=model.predict(X_test[:],X_train[:],y_train)
   result=Counter(np.equal(y_pred[:],y_test[:])).most_common()
   print(result)
   ```

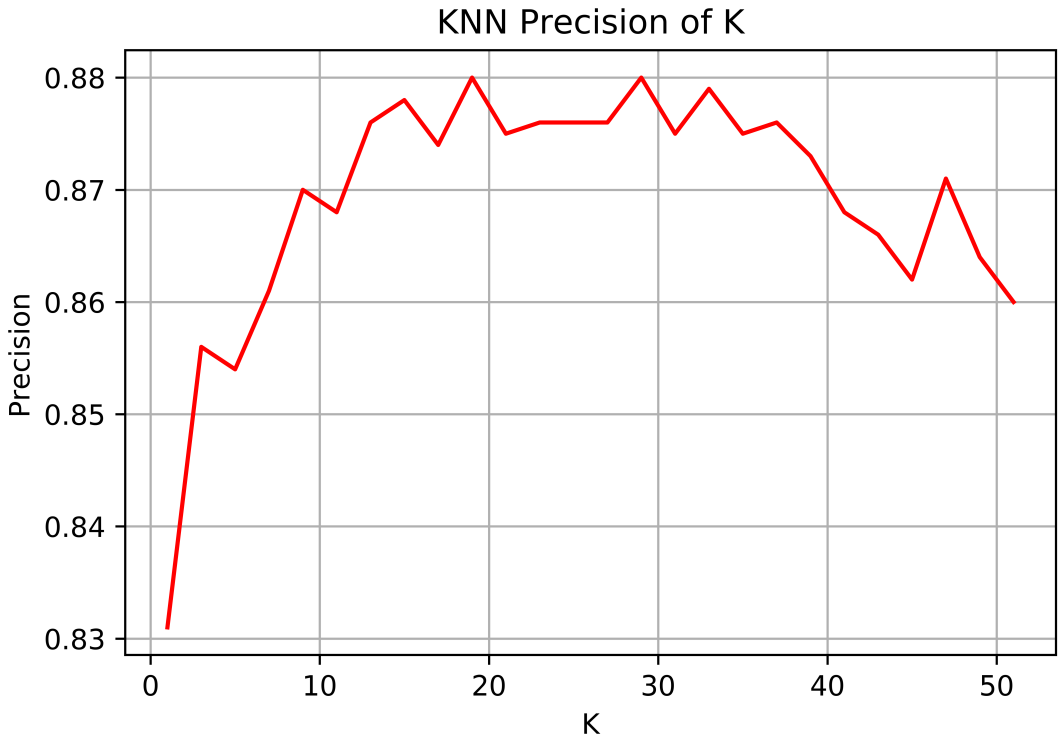
   结果为

   ```
   [(True, 802), (False, 198)]
   ```

   为了更加明确得到k的值对分类的影响，对不同的K做预测代码如下：

   ```
   plt.figure(1)
   plt.plot(range(1,52,2),[i[0][1]/1000 for i in result],'r')
   #plt.plot(range(1,52,2),[i[1][1]for i in result],'g')
   plt.title("KNN Precision of K")
   plt.ylabel("Precision")
   plt.xlabel("K")
   plt.grid(True)
   plt.savefig("./graph/PartI/KNN_Precision.png",dpi=1000)
   plt.show()
   ```

   得到的结果如下



所以大概取k=19能得到比较好的效果。

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('bool')
```

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, 762), (False, 238)]
```

   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在实例比较多的情况下，效果会比较好，但是预测的速度很慢。Logistic Regression的预测速度很快，但是它的参数$\mathbf{w}$的大小会成为它的限制。
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Logistic Regression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Logistic Regression在本实验中仅仅只有是几个参数，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。