1. （必填）自己提出的问题的理解（罗列全部）：
2. 提出的问题1：P92页，为什么将权值向量和输入向量加以扩充，式6.3 6.4就变成了6.5 6.6的样子？

讨论后的理解：因为将权值向量和输入向量扩充之后，变成了和，用矩阵运算之后，可以得到常量b，所以式子就可以化简成6.5 6.6的样子；

1. 提出的问题2：改进的迭代算法IIS主要对哪些方面进行了改进提升了效率？

讨论后的理解：与通用迭代算法相比，当求到最大似然函数那一步时，模型不知道原始参数如何变化，改进的迭代尺度算法假设最大熵模型当前的参数向量是，对于 每个参数对应于一个变化量, 希望找到一个新的参数向量，使得当期模型的对数似然函数值增加。重复这一过程，直至找到对数似然函数的最大值。

1. （必填）别人提出的问题的理解（选择几个问题罗列，并给出理解）：
2. 问题3：:回归方法应用于分类有哪些难点，逻辑斯蒂回归模型是怎么解决的？

自己的理解：

1. 回归问题用于分类时，在分类模型中无法度量不同类别之间的差距；
2. 而且没有好的转换方法，可以将有三个及以上类的因变量转化为一个定量数据，用以进行线性回归；
3. 对于0/1二分类问题，线性回归得到的数值可以看作是属于这个类的概率。即如果令 , 那么可以用线性模型 进行回归，得到的P(X)就是Y属于类1的概率。但问题是线性回归可能会产生P(X)小于0或者大于1的数。
4. 逻辑斯蒂模型的出发点就是上面的第三个原因：对于0/1二分类问题， 利用逻辑斯蒂函数（logistic function）将P(X)的范围限[0,1]之间。
5. 问题4：逻辑斯蒂回归模型和最大熵模型有什么关系？

自己的理解：

1. 最大熵模型负责约束条件约束不到的范围，让概率分布尽可能均匀，这是对偶 问题的第一步，确定了一般形式，但没有确定具体参数 ；
2. 交叉熵负责能约束到的范围 ，即在已知样本数据的分布后，让模拟的分布和数据中的分布越相似越好，这是对偶问题的第二步，用最大似然确定最优参数；
3. 而在训练逻辑斯蒂回归模型的时候，是采用 了最大熵模型的一般形式 ，即对偶问题第一步的解，并且用梯度下降等优化方法确定了具体参数的最优解 ，即解决对偶问题的第二步。
4. 问题5：对数似然函数的极大似然估计法和梯度下降以及拟牛顿法各自的优缺点是什么自己的理解：

极大似然估计：

1. 比其他估计方法更加简单；
2. 收敛性：无偏或者渐近无偏，当样本数目增加时，收敛性质会更好；
3. 如果假设的类条件概率模型正确，则通常能获得较好的结果。但如果假设模型出现偏差，将导致非常差的估计结果

梯度下降法：

1. 优点：梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的；
2. 缺点：
   1. 靠近极小值时收敛速度减慢，求解需要很多次的迭代；
   2. 直线搜索时可能会产生一些问题；
   3. 可能会“之字形”地下降

拟牛顿法：

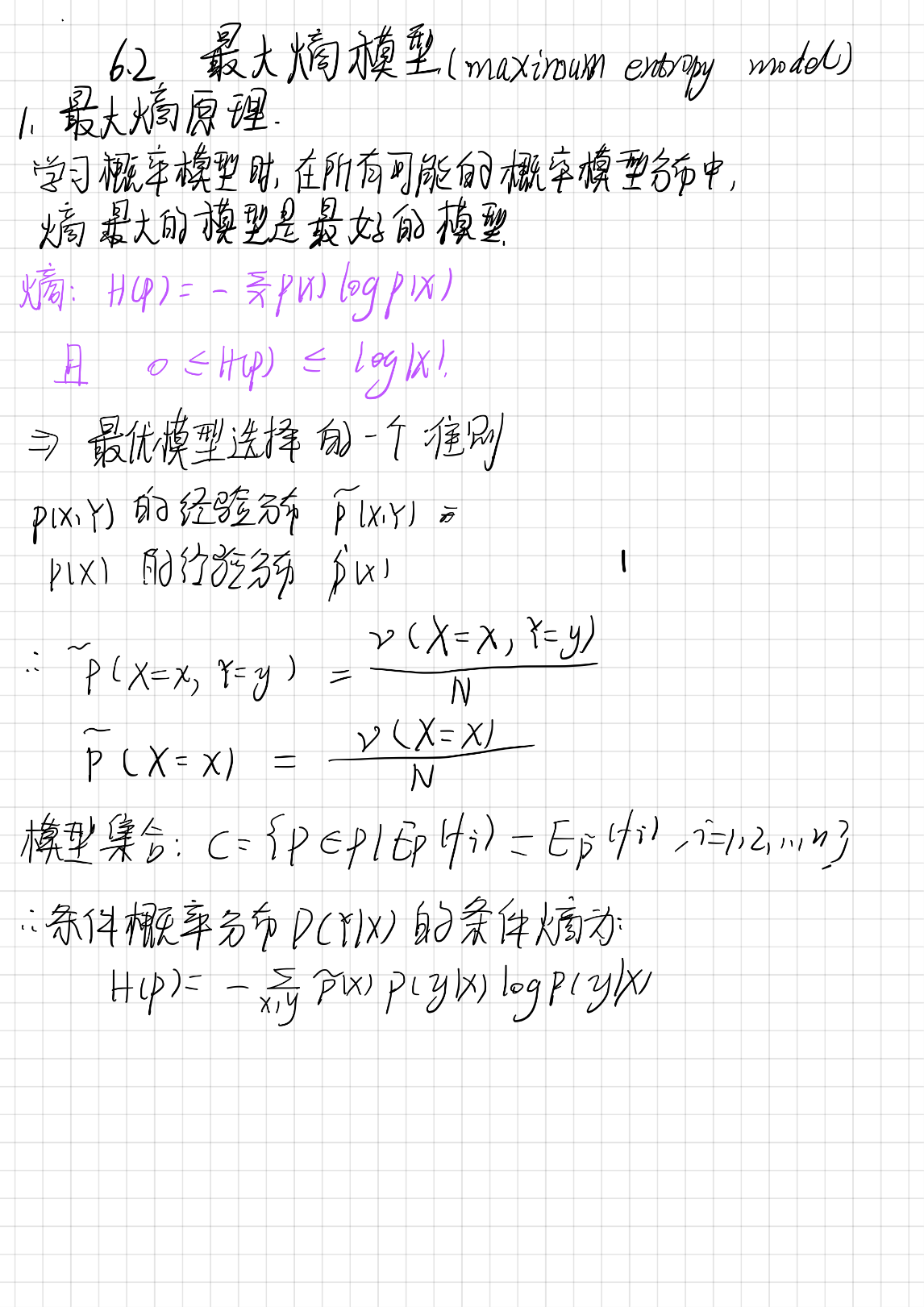
1. 优点：
   1. 拟牛顿法和最速下降法一样只要求每一步迭代时知道目标函数的梯度。通过测量梯度的变化，构造一个目标函数的模型使之足以产生超线性收敛性。这类方法大大优于最速下降法，尤其对于困难的问题。
   2. 另外，因为拟牛顿法不需要二阶导数的信息，所以有时比牛顿法更为有效；
2. （必填）读书计划

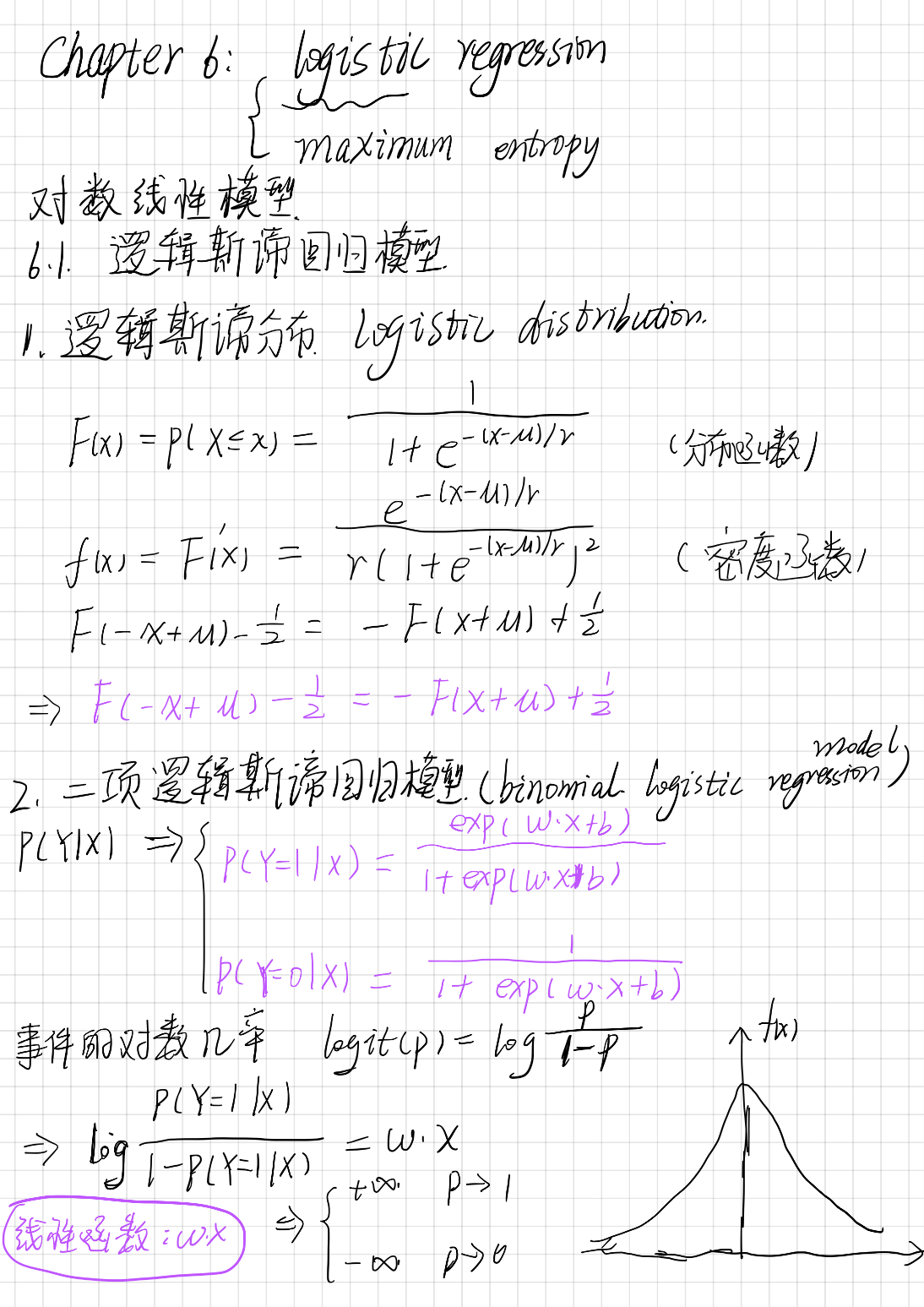
1、本周完成的内容章节：《机器学习方法》第六章

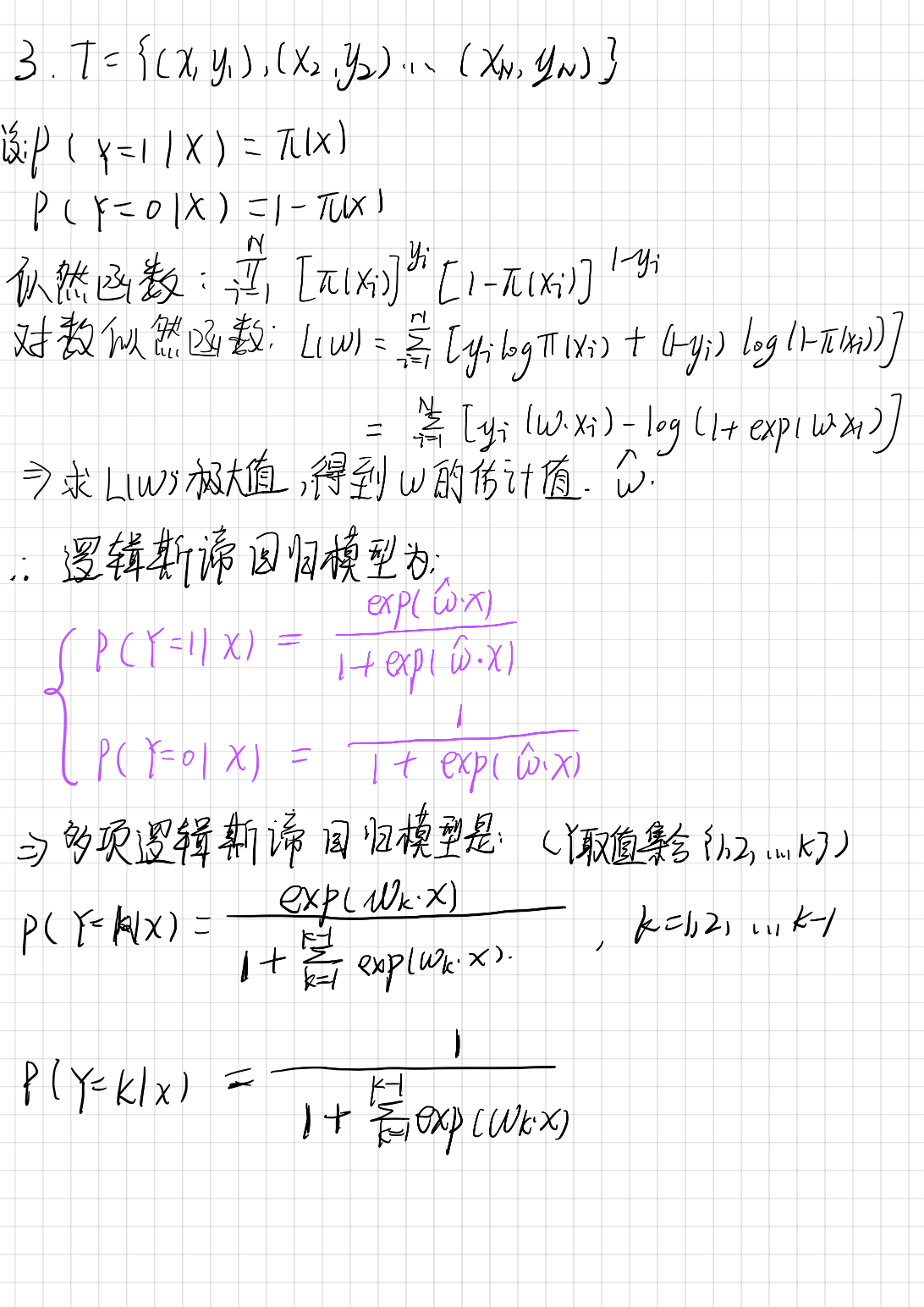
2、下周计划：《机器学习方法》第七章

四、（选做）读书摘要及理解或伪代码的具体实现（读书摘要、伪代码的具体实现代码等可以写到这个部分）

1、读书摘要及理解（选做）







2、最大熵模型实现：

l

def preprocess\_data():

# 对数据进行预处理

# 将数据的每一维的value变为index\_value的字符串，这样才能处理不同维度有相同取值的情况

# 例如 xi = [0, 0, 123, 42]

# 变为 xi = ["0\_0", "1\_0", "2\_123", "3\_42"]

images, labels = load\_data(size=20000)

X = []

for image in images:

temp = []

for j, value in enumerate(image):

temp.append(str(j) + "\_" + str(int(value)))

X.append(temp)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, labels)

return X\_train, X\_test, y\_train, y\_test

class MaximumEntropy:

# 最大熵模型，参数更新使用GIS

def \_\_init\_\_(self):

self.feature2index = None # 特征对应的下标，用于得到对应的权重值

self.feature\_count = defaultdict(int) # 特征在训练集中出现的次数

self.num\_of\_feature = None # 特征的数量

self.N = None # 训练样本的数量

self.y\_values = None # y可能的取值，用于计算P(y|X)

self.num\_of\_y = None # y取值的种类的个数

self.w = None # 特征函数对应的权重，每一个特征对应一个特征函数

self.least\_update = 0.01 # 最少的更新量，如果某次迭代w中每个参数更新量都少于它则终止

self.X = None

self.y = None

def init\_parameter(self, X, y):

self.X, self.y = X, y

self.feature2index, self.feature\_count = self.count\_feature()

self.num\_of\_feature = len(self.feature2index)

self.N = len(X)

self.y\_values = np.unique(y)

self.w = [0] \* self.num\_of\_feature # 各个特征函数对应的权重，也就是特征对应的权重

def count\_feature(self):

# 统计X，y中的feature

# 具体来说就是对于每一个数据条目 Xi, yi

# 对于Xi中的每一个属性Xij, (Xij, yi)为一个feature

# 这个feature对应的特征函数可理解为如果样本的第j维数据为Xij且label为yi则取值为1，否则为0

feature\_count = defaultdict(int) # feature计数

feature2index = {} # 将feature对应到index的字典，用于根据feature得到对应的权重值

for i in range(len(self.X)):

Xi = self.X[i]

yi = self.y[i]

for j in range(len(Xi)):

# 对于一个数据条目Xi，它的每一个维度的取值Xij和yi拼接成为一个feature

feature\_count[str(Xi[j]) + "\_" + str(yi)] += 1

for index, key in enumerate(list(feature\_count.keys())):

feature2index[key] = index

# 特征的计数验证

assert sum(feature\_count.values()) == len(self.X) \* len(self.X[0])

print("count feature finish")

return feature2index, feature\_count

def fit(self, X, y, max\_iter=500):

self.init\_parameter(X, y)

C = max([len(Xi) for Xi in X]) # 特征数量的最大值

for i in range(0, max\_iter):

new\_w = self.w[:]

model\_result = self.E\_model() # 模型期望

data\_result = self.E\_data() # 经验期望

# 更新每个特征的权值

for i, w in enumerate(new\_w):

new\_w[i] += 1.0 / C \* log(data\_result[i] / (model\_result[i] + 0.00001) )

# 检查是否收敛

update\_too\_little = True

for i in range(len(new\_w)):

if abs(new\_w[i] - self.w[i]) > self.least\_update:

update\_too\_little = False

break

if update\_too\_little:

# 结束迭代

break

# 更新权重

self.w = new\_w[:]

print("fit finish")

# for k, v in self.feature2index.items():

# print(k, self.w[v])

def E\_data(self):

# 计算特征函数(也就是特征)关于经验分布P\_(x,y)的期望值

# 即计算\sum\_(x,y) P\_(x,y)f(x,y) 这里P后面的下划线应该在P上面，表示P的经验分布

# 注意上面的求和算术只是一个f对应的期望值

result = [0] \* self.num\_of\_feature # 每一个特征都有一个期望值

for feature, count in self.feature\_count.items():

result[self.feature2index[feature]] += self.feature\_count[feature] / self.N

return result

def Pyx(self, Xi):

# p(y|x) = [\sum\_(x,y) exp(w\*f(x,y)) ] / Zw , Zw为归一化参数，可以先计算出分子然后再归一化

pyx = []

pyx\_sum = 0.0

for yi in self.y\_values:

temp\_sum = 0

# p(y|x)公式里面的求和是对所有可能的x,y的取值求和

for xi in Xi:

# 存在一个(xij, yi)即表明有一个特征函数取1，则应该加上对应的权重

feature = str(xi) + "\_" + str(yi)

if feature in self.feature2index:

temp\_sum += self.w[self.feature2index[feature]]

pyx.append([yi, exp(temp\_sum)])

pyx\_sum += exp(temp\_sum)

pyx = list(map(lambda x: [x[0], x[1] / pyx\_sum], pyx))

return pyx

def E\_model(self):

# 计算特征函数关于模型P(y|X)与经验分布P\_(x)的期望值

# \sum\_(x,y)p\_(x)p(y|x)f(x,y)

result = [0] \* self.num\_of\_feature # 每个feature有一个取值

# 上面的式子是对所有可能的x,y求和，先循环遍历所有X

for i in range(self.N):

Xi = self.X[i]

# 先计算P(y|x)

pyx = self.Pyx(Xi)

# 再计算 \sum\_(x,y)p\_(x)p(y|x)f(x,y)

for y, prob in pyx:

for xi in Xi:

feature = str(xi)+"\_"+str(y)

if feature in self.feature\_count.keys():

result[self.feature2index[feature]] += prob \* 1.0 / self.N

return result

def predict(self, X):

preds = []

for Xi in X:

pyx = self.Pyx(Xi)

max\_prob = 0

pred\_y = None

for y, prob in pyx:

if prob > max\_prob:

pred\_y = y

max\_prob = prob

preds.append(pred\_y)

return preds

if \_\_name\_\_ == "\_\_main\_\_":

debug = 0

if debug == 1:

X = []

y = []

with open("ch06-data.txt", "r") as f:

lines = f.readlines()

for line in lines:

splited = line.strip().split("\t")

temp = []

for i, feature in enumerate(splited[1:]):

temp.append(str(i) + "\_" + str(feature))

X.append(temp)

y.append(splited[0])

model = MaximumEntropy()

model.fit(X, y)

preds = model.predict(X)

print(preds)

print(y)

else:

X\_train, X\_test, y\_train, y\_test = preprocess\_data()

model = MaximumEntropy()

model.fit(X\_train, y\_train)

preds = model.predict(X\_test)

# print(np.array(preds))

# print(y\_test.flatten())

print("accurcy:", accuracy\_score(np.array(preds), y\_test.flatten()))