中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	**计科 2 班**	专业（方向）	**计算机科学与技术**
学号	**20337263**	姓名	**俞泽斌**

一、 实验题目

在给定文本数据集完成文本情感分类训练，在测试集完成测试， 计算准确率。

要求

- 文本的特征可以使用 TF 或 TF-IDF （可以使用 sklearn 库提取特征）
- 设计合适的网络结构，选择合适的损失函数利用训练集完成网络训练，

并在测试集上计算准确率

- 需提交实验报告+代码
- 实验报告应包含损失的可视化展示，以及学习率对准确率影响的可视化展示

二、 实验内容

1. 算法原理

首先是读入数据的部分，这里主要调用的是 re 的库，主要操作就是逐行读入数据，然后对于每行数据按照相同的格式进行分类，加入到两个列表中，分别是整体语句以及情感程度。

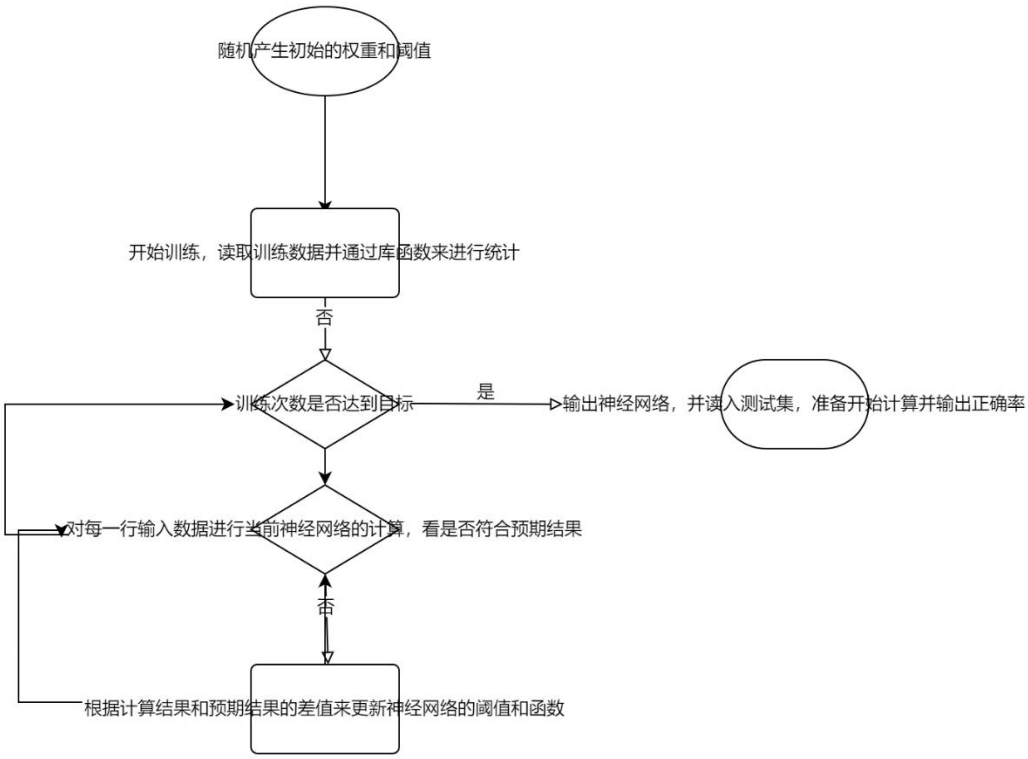
然后到了通过向量的方式来展示语句的方面上来了，具体的也是调用了 sklearn 里的 TfidfVectorizer（）函数中的 fit 和 transform 函数，可以通过这两个函数将语句统计并标准化。

至于神经网络，该算法主要的原理就是感知器的理解上了， BNN 神经网络，是由很多个单个神经元所组成，每一个神经元都是通过输入，然后进行一些操作，与阈值相比较，然后得出输出，这次的实验中我还设计了一个隐层，增加了神经网络的复杂性，希望能得到更好的结果。

这里对于这个神经网络，我们一开始的各项的权重包括阈值之类的都是通过随机数来生成的，然后通过 train 函数不断读取训练集中的数据，不断

地修改相关的权重和阈值，最终得到我们新的神经网络，然后读入 test 集合，对于每句话进行神经网络的判断，与一开始的第一项的情绪数字进行比较，从而得出是否相同，然后记录正确数，计算正确率。

2. 伪代码



3. 关键代码展示（带注释）

```
def loadData(filepath):
    data = []  # 除了开头的两个数字以外的字符
    data_emotion = []  # 情感程度
    pattern = r'(\d+) (\d) ([a-zA-Z]+) (.+)'
    with open(filepath) as fp:
        fp.readline()  # 读取第一行
        for line in fp:
            match_res = re.match(pattern, line)
            data_emotion.append(match_res.group(2))
            data.append(match_res.group(4))
        # print(data)
        # print(data_emotion)
        return data, data_emotion
```

一开始是一个关于读入数据的方面，这里调用的是 re 这个库里的 match 函数，因为一开始的数据都是一串字符，需要先进行统计之类的操作，然后才能交给 sk-learn 里面的库进行操作，得到需要的向量之类的数据。

```python
class Perceptron:  # data为输入的标准化过后的向量，target为标准
    def __init__(self, data, target):
        self.datanum = data.shape[0]  # 数据数目
        self.inputnum = data.shape[1]  # 输入层神经元个数
        self.hidnum = int((self.inputnum + 6) ** 0.5) + 1  # 隐藏层神经元个数
        self.outnum = 6  # 输出层神经元个数
        self.data = data  # 输入矩阵
        self.target = target  # 目标
        self.Target_vec = np.eye(6)  # one-shot形式的向量
        self.w1 = 2 * np.random.random((self.hidnum, self.inputnum)) - 1  # 输入层与隐藏层的权值
        self.w2 = 2 * np.random.random((self.outnum, self.hidnum)) - 1  # 隐藏层与输出层的权值
        self.b1 = 2 * np.random.random((self.hidnum, 1)) - 1  # 隐藏层的阈值
        self.b2 = 2 * np.random.random((self.outnum, 1)) - 1  # 输出层的阈值
        self.loss = 0
        self.allloss=[]
```

首先定义了一个 Perceptron 的类，来保存我的模型，模型用的是两层的神经元，然后保存了一下两层神经元的分别的权重和阈值。

```python
    def train(self, learn_rate):
        tdata = self.data
        tdata = tdata.toarray()
        # print(tdata)
        trying_times = 0
        while trying_times < train_times:
            trying_times += 1
            dif=0
            for i in range(self.datanum):
                x = tdata[i].reshape(-1, 1)
                tv = self.Target_vec[int(self.target[i]) - 1].reshape(-1, 1)  # 目标输出向量
                th = sgn(np.dot(self.w1, x) - self.b1)
                ty = sgn(np.dot(self.w2, th) - self.b2)
                t2 = (ty - tv) * (1 - ty) * ty
                t1 = np.dot(self.w2.T, t2) * (1 - th) * th
                self.w1 -= learn_rate * np.dot(t1, x.T)
                self.w2 -= learn_rate * np.dot(t2, th.T)
                self.b1 -= -learn_rate * t1
                self.b2 -= learn_rate * t2
                dif += np.square(ty-tv)
            self.loss=np.sum(dif)/(2*self.datanum)
            print(self.loss)
            self.allloss.append(self.loss)
```

接着是具体的有关训练时的操作，其实主要参考的还是 ppt 上的训练操作，先把得到的数据全都放到-1 到 1 的范围下，然后通过向量的操作来得到对应的解，也就是当前神经网络的输出，与目标的向量来进行比较，并且通过学习率等操作来修改更新我们两个权值和阈值。至于损失函数，我在这里用的是 mse，也就是均方误差的方法，将当前神经网络的输出，与目标的向量的平方差值通过 numpy 库内的 sum 函数来得到并且/2n 得到相对的数值，方便之后输出

```python
def test(self, data, target):
    rightnum = 0  # 正确数
    data_array = data.toarray()
    datanum = data.shape[0]
    for i in range(datanum):
        x = data_array[i].reshape(-1, 1)
        th = sgn(np.dot(self.w1, x) - self.b1)
        ty = sgn(np.dot(self.w2, th) - self.b2)
        emotion = np.argmax(ty)  # y中最大值的索引，代表当时的情感强度
        tt = int(target[i]) - 1
        if emotion == tt:
            rightnum += 1
            # print(out)
            # print(data[k])
        # rightrate=rightnum/data.shape[0]
    return rightnum / data.shape[0], rightnum
```

这是测试函数，也就是对测试集来工作的函数，主要的思想其实类似于训练集了，就是通过一些操作来将输入的数据经过神经网络得到输出，然后与本来的数据内进行比对，如果比对成功，就说明该神经网络在当前的输入为正确的，正确数目加一，同时最后得出正确率来输出。

```
def main():
    #for i in range(20):
    learn_rate = 0.002
    train, train_target = loadData('./Classification/train.txt')  # 读取训练集
    test, test_target = loadData('./Classification/test.txt')  # 读取测试集
    vectorizer.fit(train)
    standard_train = vectorizer.transform(train)  # 标准化
    # vectorizer.fit(test)
    standard_test = vectorizer.transform(test)
    model = Perceptron(standard_train, train_target)
    start_time = time.process_time()
    model.train(learn_rate)
    truerate, truenum = model.test(standard_test, test_target)
    end_time = time.process_time()
    print("正确个数为", truenum, "正确率为", truerate * 100, "%")
    print(f'耗时:{end_time - start_time}s')
    printfig(model.allloss)
    #print(model.allloss)
    #printfig(all_truerate, all_learnrate)
def printfig(allloss):
    plt.plot(allloss)
    plt.legend()
    #plt.grid(True, linestyle='--', alpha=0.5)
    plt.show()
```

这里就是主题的 main 函数了，其实有了上面的基础，main 函数中最大的功能可能就是 sk-learn 这个库里的函数调用了，毕竟通过输入得到的都是字符，需要 sk-learn 库中的函数进行由字符到向量的转化。然后通过产生一个 model 对象来先通过训练集进行训练模型，完善了里面的权值和阈值，然后用测试集来测试，并且统计里面的正确数和正确率。

4. 创新点&优化（如果有）

```
    printfig(model.allloss)
    #print(model.allloss)
    #printfig(all_truerate, all_learnrate)
def printfig(allloss):
    plt.plot(allloss)
    plt.legend()
    #plt.grid(True, linestyle='--', alpha=0.5)
    plt.show()
```

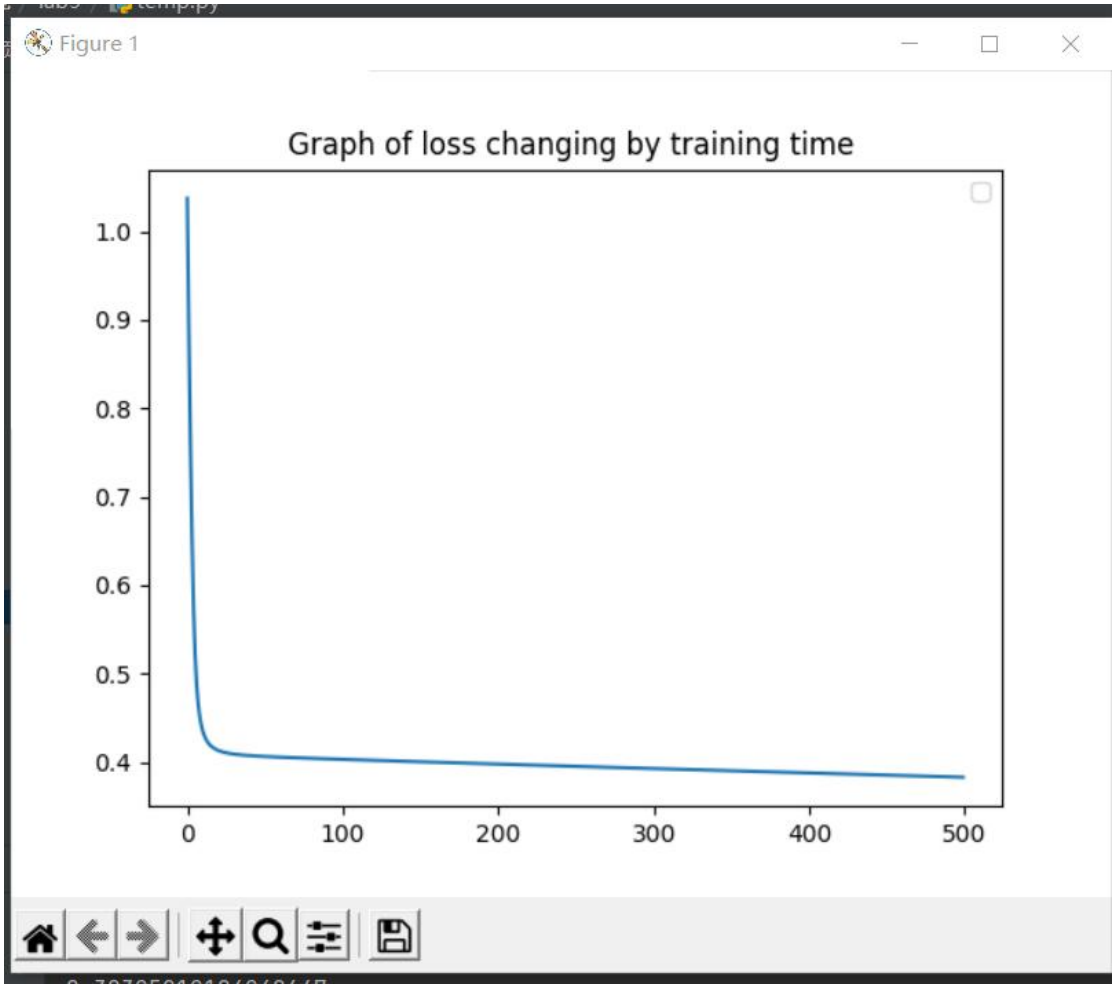
调用了 plt 库来实现具体的关于损失的图像实现，能够更直白地看出随着训练次数的增加，损失的变化。

```
pattern = r'(\d+) (\d) ([a-zA-Z]+) (.+)'
with open(filepath) as fp:
    fp.readline()  # 读取第一行
    for line in fp:
        match_res = re.match(pattern, line)
        data_emotion.append(match_res.group(2))
        data.append(match_res.group(4))
    # print(data)
    # print(data_emotion)
    return data, data_emotion
```

采用 re 库来对数据进行预处理，使得数据可以先行统计里面的参数和大小，方便之后调用 sk-learn 库以及后续操作

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）



首先是损失关于训练次数的增长而减小的图，可以看到，在较短的训练时间内，

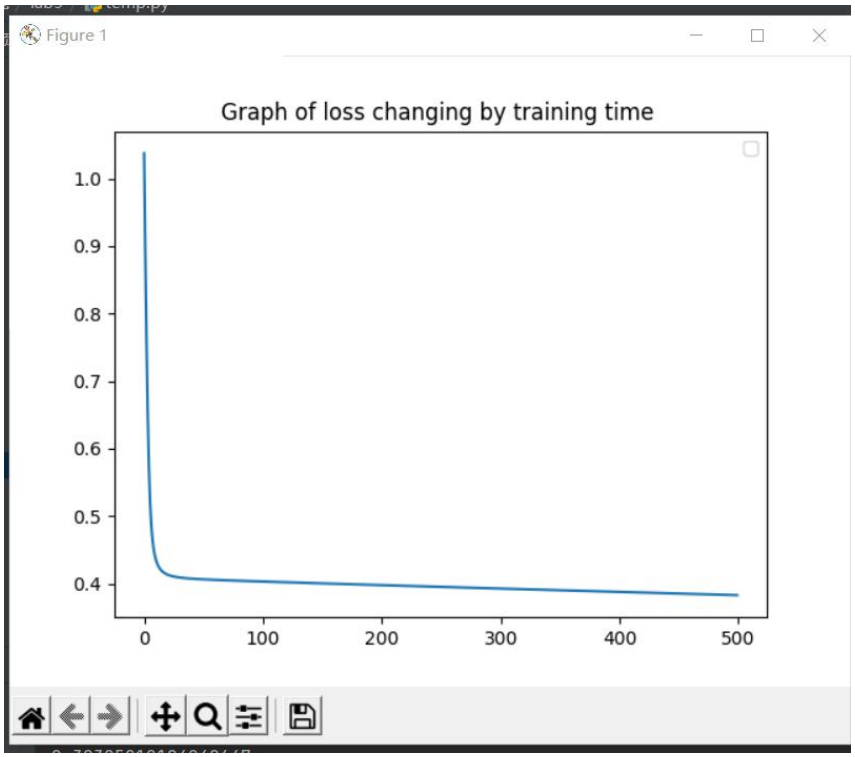
损失已经开始慢慢收敛，收敛到基本在 0.4 左右。

```
0.383103021499570 76
正确个数为 344 正确率为 34.4 %
耗时:24.71875s
```

并且当时的正确率基本在 34 左右，测试的为测试集里的数据。此时学习率为 0.002

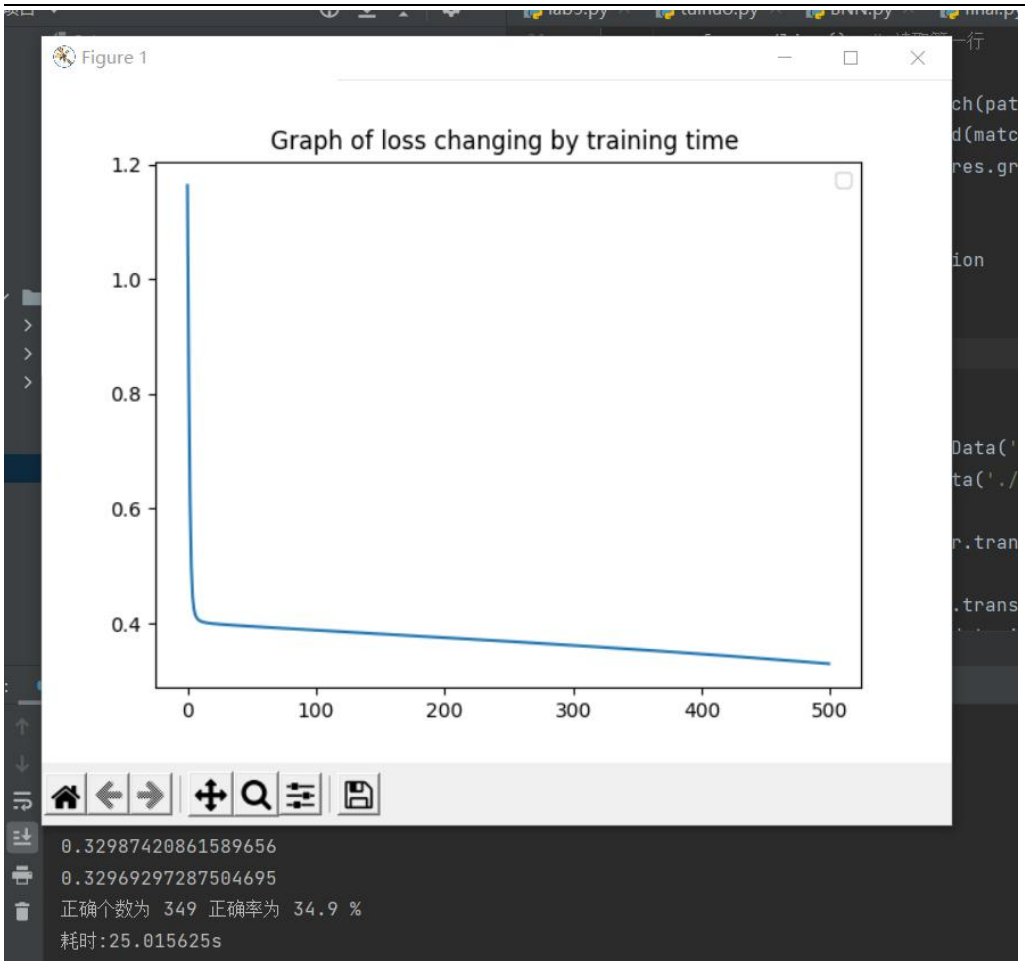
2. 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

当学习率为 0.002 的时候



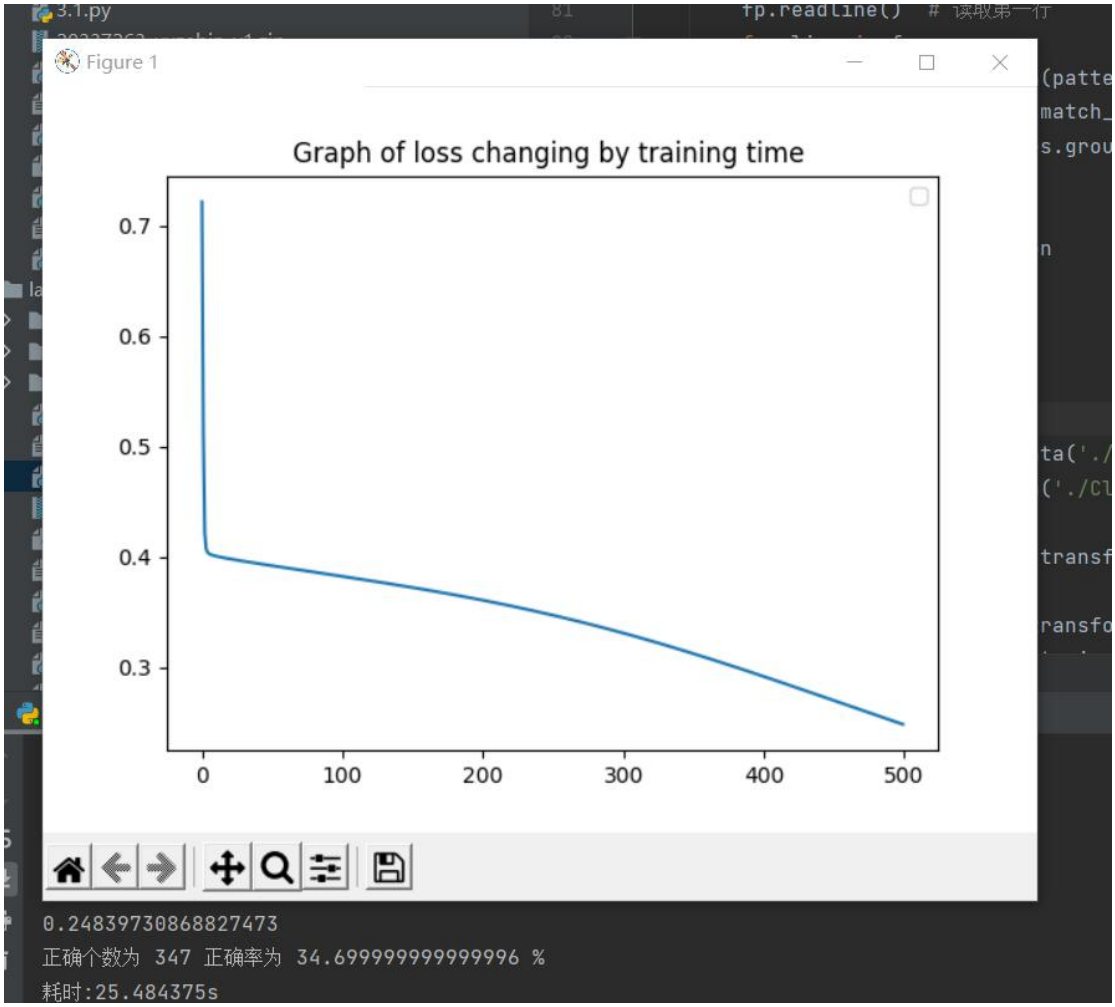

```
0.383103021499570 76
正确个数为 344 正确率为 34.4 %
耗时:24.71875s
```

当学习率为 0.005 的时候



可以看到，收敛的更快，而且正确率有一点点的增长。

当学习率为 0.01 的时候



正确率又有一点点增长，且收敛变慢，可能需要进一步的训练。

所以综上，在学习率较低的情况下，增长一点学习率可能会提高一点点正确率

|----------如有优化，请重复 1，2，分析优化后的算法结果----------|

四、 参考资料

https://blog.csdn.net/qq_37373203/article/details/82791200?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216541562841678143299665 4%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=165415628416781432996654&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-82791200-null-null.142^v11^control,157^v13^control&utm_term=python%E5%8D%95%E5%B1%82%E6%84%9F%E7%9F

%A5%E5%99%A8%E5%AD%A6%E4%B9%A0%E7%8E%87&spm=1018.2226.3001.4187