早上老大问我有没有做过关乎NLP的词频统计，当时没听太懂，就说没有做过。可是回过头来想想，记得研究生的时候还上过text mining这节课，当时还介绍过情感分析的东西。本想登入账号进去看看课件，但是突然想起来我们的账号已经被注销了，资料没了，一时是淡淡的忧伤。然后努力地回忆，想从残存的记忆中提取一些有用的东西，加上google了下，发现应用一下还是挺容易的，于是乎。。。

                                                                                                                                                                     ----此为前言

**Unicode**

本来对于python也不熟练，编码的问题也了解不多，踩了很多坑。侥幸成功后，本着求甚解的态度，深入了解一下unicode编码的规则。

首先大家最熟悉的肯定是ascii编码（BTW，读作“as-key”），8位，有256种可能，但是只用到了前127位，首位是0，编码了常用的标点符号，以及26个英文字母的大小写。一个英文字母，对应于一个唯一的8位二进制数，对于英语来说，这种实现很完美，但是对于汉字，就不适用了。

汉字的组成方式和英语不同，一个汉字就得对应于一个编码，然而汉字的数目很庞大，8位的二进制数显然不够用，于是乎就想到了用两个字节，16位来唯一编码一个汉字。但是呢，如果不同的国家对于自家的文字都这么干就会造成混乱，不统一，计算机就不能正确识别其他国家的文字，因为编码不同。这时候一个叫ISO的组织跳了出来，准备指定一个统一的标准，这个标准就是unicode。unicode就用16位，两个字节的标准，为世界上所有字符编码，2^16=65536，好像足够应付当下所有语言的字符。为了兼容以前的ascii编码，unicode保持它们原有的编码不变，只是将其长度由8位扩展为16位，并且高8位全部为0。

unicode的缺点也显而易见，对于英语来说，它比原来浪费了近1倍的存储空间。因为ascii 编码对于每个字母只需要1个字节，而unicode需要两个字节。如果只是存在硬盘上，这个缺点也没什么，因为硬盘便宜嘛。当互联网出现，这种浪费给传输带了的压力就不容忽视。于是乎就出现了面向传输的标准UTF。UTF-8就是一次传输8位，一个字节，而UTF-16就是一次两个字节。后者我没有仔细了解，但是大概原理和前者差不多吧，这里我只简单介绍下UTF-8的编码方式。

**UTF-8**

UTF-8 就是一套以8位为一个编码单位的可变长编码，可！变！长！（让我想到了哈夫曼，虽然原理不同，但是所实现的功能是一样的，可变长能节省空间）。它将一个字符编码为1到4个字节，如下所示：

U+  0000 ~ U+ 007F: 0XXXXXXX

U+  0080 ~ U+ 07FF: 110XXXXX 10XXXXXX

U+  0800 ~ U+ FFFF: 1110XXXX 10XXXXXX 10XXXXXX

U+10000 ~ U+1FFFF: 11110XXX 10XXXXXX 10XXXXXX

我们先从一个例子入手，【我爱你】的unicode 编码 为 "\u6211\u7231\u4f60"，其中\u是转移符，表示这个unicode编码。每个汉字由4个16进制的数表示，一共是16位。根据上面的编码规则，【我爱你】属于第三行的那一类，那么他们的UTF-8编码就是三个字节。那么UTF-8和unicode之间具体如何转换呢，UTF-8是这样做的：

1. 单字节的字符，字节的第一位设为0，因此对于英文文本，UTF-8码只占用一个字节，和ASCII码完全相同（这就解决了unicode的弊端）

2. n个字节的字符(n > 1)，第一个字节前n位设为1， 第n+1位设为0，后面字节的前两位都设为10，这n个字节的其余空位填充该字符的unicode码。

以上就是转换规则。

下面我们把【爱】从unicode转换为UTF-8编码，

             7        2        3        1

       0111  0010  0011  0001                         二进制的 7231

         0111       00 1000      11  0001             二进制的 7231

1110 XXXX  10XX XXXX 10XX XXXX             模板

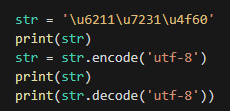
1110 0111   1000 1000  1011 0001              代入模板

      E       7       8       8        B       1                UTF-8码

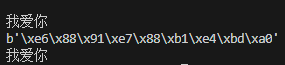
这就是将unicode编码的7231按照UTF-8标准编码为字节序列E788B1。

至于改为unicode 能节省多少空间，我没有统计过，当然也并不知道，但是想来有这样的设计，必然优于unicode编码吧（我就是这么迷信权威）。

为了更好得理解，我么再来看一个python3的例子，如下图



其中，str是以unicode编码的，表示【我爱你】。然后我们可以通过encode这个函数，将str重新用utf-8来编码。经过utf-8编码后，这个str也就有了decode方法，可以重新解码为unicode。我们来看一下结果的三个输出，如下图：



可以看到，第一个输出就是unicode编码所代表的字符。第二个输出是经过了utf-8编码，是以二进制表示的字节，每个字占用三个字节，其中【爱】所表示的和我们先前算出来的一致，为【e788b1】，其中\x是转义符，表示16进制。