程序设计大作业：单词查询

禹顺尧 2019202290

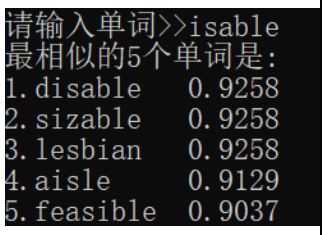
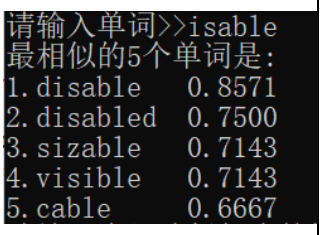
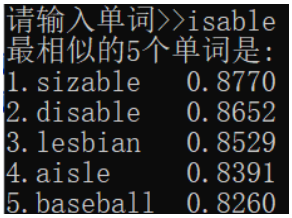
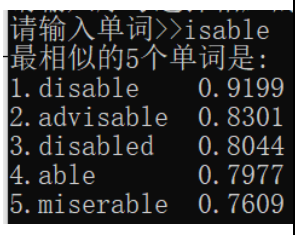
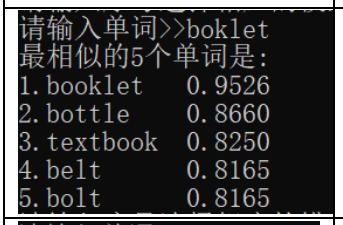
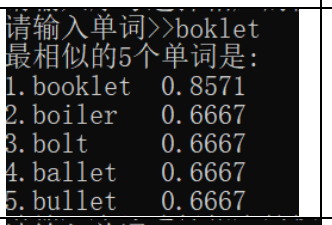
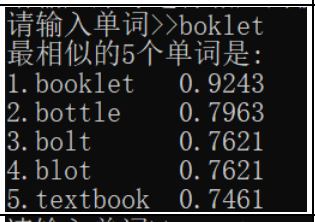
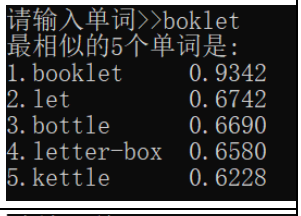
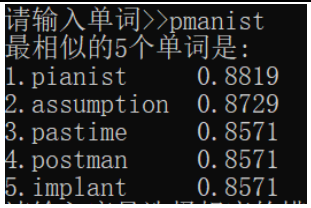
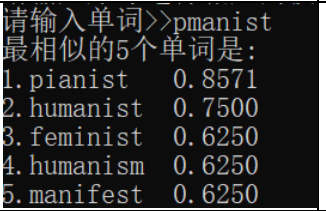
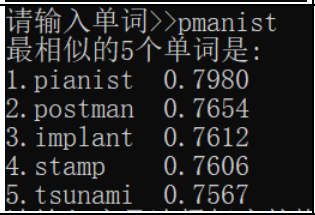
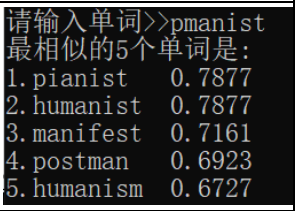
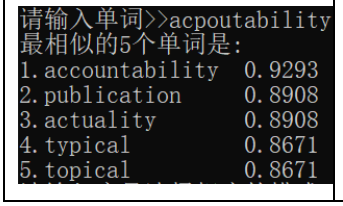
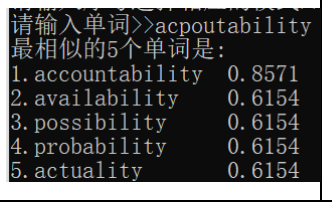
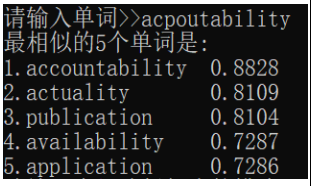
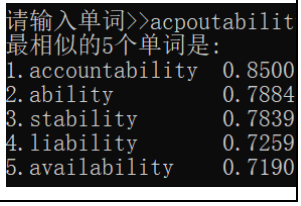
一．程序说明

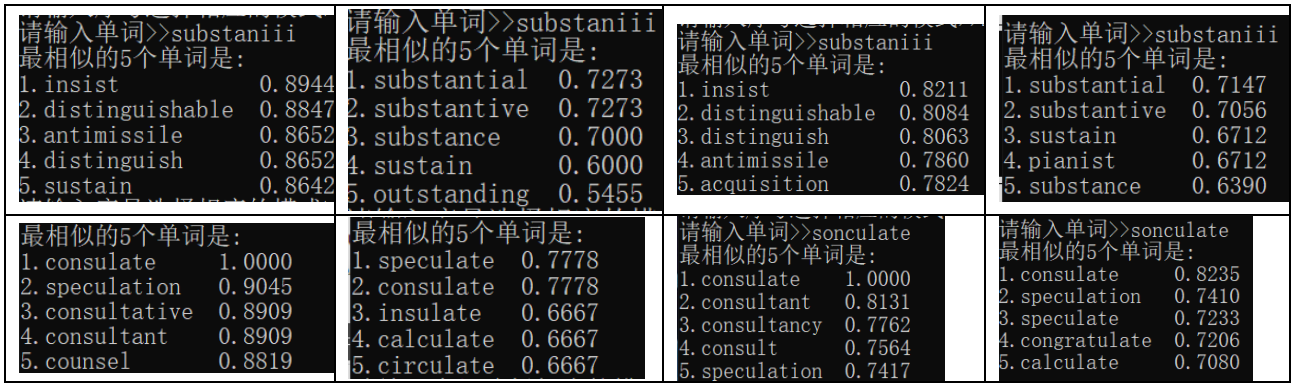
本节简要说明应用程序的框架。源程序除主程序文件 main.cpp 外由 5 个主要部分构成，分别编码在不同的头文件和源代码文件中（IDE：CodeBlocks）。

- datadef.h/cpp　　负责定义全局变量
- WordVectors.h/cpp　　定义 Word 类,声明并实现成员变量与成员函数
- mode.h/cpp　　(1)构建词典　(2)两种单词查询的模式
- sml_algorithm.h/cpp　　负责实现若干相似度的算法
- Word_Methods.h/cpp　　实现程序中所需的函数

二．算法效果说明

本节通过截图展示各相似度算法的效果对比。单词由随机算法通过增加，删除，替换，交换字母生成。

余弦相似度	编辑距离相似度	调整余弦相似度	扩展向量后的余弦相似度
			
			
			
			



通过对比，发现算法效果：编辑距离相似度>调整余弦相似度>余弦相似度。余弦相似度只关注字母的组合，而编辑距离相似度不仅考虑了字母的组合，而且考虑了字母的排列，因此编辑距离相似度的算法效果更好。

程序中还定义了欧几里得距离相似度，算法效果与余弦相似度类似。

另外，在测试过程中发现，如果输入的单词拼写有误，且单词长度较短时，算法结果可能与预期相悖。推测原因在于拼写有误的单词可能与除原单词以外的某些单词更加相似。

三．程序亮点

1.为体现相同字母组合单词的差别，将向量扩充到 702 维，其中 1-26 维表示 a-z 26 个字母，27-702 维表示任取两个字母的排列(aa,ab,ac,……) (详见 sml_algrithm.cpp 中的函数 extended_vec_sml())。函数返回向量的余弦相似度。下面以截图体现其算法效果：

由以上四例可看出，扩充向量后算法能够对相同字母的不同单词进行区分，而普通的余弦相似度算法不能做到这一点。

2.参考网上的文献，对余弦相似度算法进行调整。(均值由另外的程序从词典中统计得来）由上一节的结果对比看出算法效果略有优化。

正因为余弦相似度在数值上的不敏感，会导致这样一种情况存在：

用户对内容评分，按5分制，X和Y两个用户对两个内容的评分分别为（1,2）和（4,5），使用余弦相似度得到的结果是0.98，两者极为相似。但从评分上看X似乎不喜欢2这个 内容，而Y则比较喜欢，余弦相似度对数值的不敏感导致了结果的误差，需要修正这种不合理性就出现了调整余弦相似度，即所有维度上的数值都减去一个均值，比如X和Y的评分均值都是3，那么调整后为（-2，-1）和（1,2），再用余弦相似度计算，得到-0.8，相似度为负值并且差异不小，但显然更加符合现实。

那么是否可以在（用户-商品-行为数值）矩阵的基础上使用调整余弦相似度计算呢？从算法原理分析，复杂度虽然增加了，但是应该比普通余弦夹角算法要强。

```
double adj_cos_sml(Word a,Word b)
{
    int t=0;
    double al=0,bl=0;
    for (int i=0;i<=25;i++)
    {
        t+=sq(a.vec[i]-b.vec[i]);
        al=al+sq(a.vec[i]-tj[i]);
        bl=bl+sq(b.vec[i]-tj[i]);
    }
    return (al+bl-t)/(2*sqrt(al)*sqrt(bl));
}
```

3.编辑距离算法。由动态规划得出的答案表示修改的次数。通过 $1-\frac{f[l1][l2]}{\max(l1,l2)}$ 归一化，表示出相似度，方便比较大小。

```
double ed_dstc_sml(Word a,Word b)
{
    int l1=a.len;
    int l2=b.len;
    int f[51][51]={0};
    for (int i=0;i<=l2-1;i++) f[0][i+1]=i+1;
    for (int i=0;i<=l1-1;i++) f[i+1][0]=i+1;
    for (int i=1;i<=l1;i++)
        for (int j=1;j<=l2;j++)
        {
            int k;
            if (a.proto[i-1]==b.proto[j-1]) k=0;else k=1;
            f[i][j]=min(min(f[i-1][j]+1,f[i][j-1]+1),f[i-1][j-1]+k);
        }
    double p=double (f[l1][l2])/max(l1,l2);
    return 1-p;
}
```

4. 函数 get_a_word() ： 配合全局变量 n(代表模式的数字)，a_word(储存待查询的单词)，通过递归调用，实现了模式的即时切换，并保证了程序的鲁棒性。

```
void get_a_word(int k)
{
    char s[51];
    if (k==0) printf("请输入单词,或输入序号更改模式>>");
    fflush(stdin);
    gets(s);
    if (strcmp(s,"1")==0||strcmp(s,"2")==0)
    {
        if (strcmp(s,"1")==0) n=1;else n=2;
        get_a_word(0);
        return;
    }
    if (strcmp(s,"3")==0) exit(0);
    while (judge(s)==false)
        {
            printf("输入有误，请输入单词或序号>>");
            get_a_word(1);
            return;
        }
    a_word.set(trans(s));
}
```

其中函数 judge 简单判断输入的字符串是否为一个英文单词：

```
bool judge(char *s)//判段读入的字符串是不是一个英文单词
{
    int ll=strlen(s);bool flag=false;
    if (ll==0) return false;
    for (int i=0;i<=ll-1;i++)
        if (s[i]<'A'||s[i]>'z'||(s[i]>'Z'&&s[i]<'a'))
        {
           if (s[i]!='\''&&s[i]!='-') return false;
        }
        else flag=true;
    return flag;
}
```

参考： https://blog.csdn.net/weixin_39050022/article/details/80732249

https://www.jianshu.com/p/a617d20162cf