中山大学数据科学与计算机学院

移动信息工程专业-人工智能

本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	**周五 5-6 节**	专业（方向）	**软件工程（移动信息工程）M2**
学号	**15352204**	姓名	**林丹**

一、 实验题目

实验二：K 近邻与朴素贝叶斯——分类和回归

二、 实验内容

1. 算法原理

输入：样本的特征向量（OneHot 矩阵/TF 矩阵/TF-IDF 矩阵）

分类问题：输出是测试样本的标签。

回归问题：输出的是测试样本的属性值。

- KNN (K-Nearest-Neighbor)

  样本间距离的计算：

  特征向量分别为 $X=(x_1,x_2,\dots x_n), Y=(y_1,y_2,\dots,y_n)$ 的两个样本距离

$$\left(\sum_{i=1}^{n}|x_i-y_i|^p\right)^{1/p}$$

  p=1 时，曼哈顿距离；

  p=2 时，欧氏距离；

  余弦距离：

$$\cos\left(\vec{A},\vec{B}\right)=\frac{\vec{A}\cdot\vec{B}}{|\vec{A}||\vec{B}|}$$

❖ KNN 分类：

  测试样本标签 = k 个距离最小的训练样本中标签的众数。

❖ KNN 回归：

$$测试样本属性值=\sum_i 权重_i \times k个距离最小的训练样本_i的属性值$$

- NB (Naïve Bayes)

  贝叶斯定理：

  X 是样本特征向量，C_i是样本标签，

$$P(C_i \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid C_i)P(C_i)}{P(\mathbf{X})}$$

  因为 P(X)是一个常数，所以有

$$P(C_i \mid \mathbf{X}) \propto P(\mathbf{X} \mid C_i)P(C_i)$$

  特征条件独立假设：

$$P(\mathbf{X} \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

  设一个样本的特征向量为：

$$\boldsymbol{X} = (x_1, x_2, \dots, x_n)$$

根据以上规则求出测试样本特征向量是属于各个标签概率 $P(C_i|X)$

❖ NB 分类：

  测试样本标签 C

$$C = arg \max_C P(C_i|\boldsymbol{X}) = arg \max_C P(\boldsymbol{X}|C_i)P(C_i) = arg \max_C \prod_{k=1}^{n} P(x_k|C_i)P(C_i)$$

$$P(x_k|C_i) = \frac{标签为C_i中出现单词x_k数目}{标签为C_i中所有单词总数} = \frac{nw_{e_i}(x_k)}{nw_{e_i}}$$

$$P(C_i) = \frac{标签为C_i的文章数目}{文章总数} = \frac{nd_{c_i}}{nd}$$

❖ NB 回归：

$P(C_i|\boldsymbol{X}) \propto P'(C_i|X)$

$$P'(C_i|\boldsymbol{X}) = \sum_{j=1}^{M} P(\boldsymbol{X}|d_j)P(C_i) = \sum_{j=1}^{M} \prod_{k=1}^{n} P(x_k|d_i)P(C_i)$$

$$P(x_k|d_i) = TF(词频) = \frac{文章d_i中出现单词x_k数目}{文章d_i中所有单词总数} = \frac{nd_{e_i}(x_k)}{nd}$$

$$P(C_i) = \frac{标签为C_i的文章数目}{文章总数} = \frac{nd_{c_i}}{nd}$$

  为了保证 $\sum_{i=1}^{6} P(C_i|\boldsymbol{X}) = 1$，概率归一化：

$$P(C_i|X) = \frac{P'(C_i|\boldsymbol{X})}{\sum_i P'(C_i|\boldsymbol{X})}$$

2. 伪代码

定义：

类 Passage（分类用）		
成员名	**成员类型**	**意义**
label	string	每一篇文章的 label
voc	vector<string>	每一篇文章文本（单词）

类 Passage（回归用）		
成员名	**成员类型**	**意义**
pro[6]	string	每一篇文章 6 种情感概率
voc	vector<string>	每一篇文章文本（单词）

变量名	变量类型	变量含义
PASSAGE_NUM	int	训练集文章上限
WORD_NUM	int	单词库中单词个数上限
k	int	KNN 算法中 K 值
knn	multimap<double,string>	KNN 算法（距离，标签）map
word	vector<string>	单词库
training	vector<Passage>	训练集的所有文章
newdata	vector<Passage>	测试集的所有文章
TF0, TF1	map<Pos, double>	辅助计算 TF 矩阵的 map
OneHot_training	int[][]	训练集 OneHot 矩阵
OneHot_new	int[][]	测试集 OneHot 矩阵
TF_training	double[][]	训练集 TF 矩阵
TF_new	double[][]	测试集 TF 矩阵
num_training	int	训练集文章总数
num_new	int	测试集文章总数
dis	double	两个特征向量距离
label	string[6]	6 种情感
predict	string	预测情感标签
alpha	double	拉普拉斯平滑系数α
pro_tmp	double[][]	$\prod_{k=1}^{n} P(x_k\|d_i)P(C_i)$
pro_sum	double	$\sum_{i=1}^{6} P(C_i)$（为概率归一化）
pro_emo	double[6]	KNN 回归各个情感概率
emotion	double[6]	标签为 i 的文章数量/文章总数
Times_training	int[][]	训练集文章中单词出现次数表
all_sum	int[6]	每一种情感包含的单词总数

Probability	double[6][]	6 种情感中单词出现的总次数，或者 P(Xj|Ei)

1） KNN 分类：

```
label[6] ← { "anger","disgust","fear","joy","sad","surprise" }
for r←0 to <num_new by r++
    for c←0 to num_training by c++
        for j←0 to word.size() by j++
            dis ← dis + (TF_training[c][j] - TF_new[r][j] )^2
            knn.insert((dis,label))
        end
    end
    count[6] ← 0
    for i←0 to k by i++
        if map[i].label="anger"          then count[0]++
        else if map[i].label="disgust"   then count[1]++
        else if map[i].label="fear"      then count[2]++
        else if map[i].label="joy"       then count[3]++
        else if map[i].label="sad"       then count[4]++
        else if map[i].label="surprise"  then count[5]++
    end
    predict ← label[x] which x makes max( count[6] )
end
```

2）KNN 回归：

```
for r←0 to <num_new by r++
    for i←0 to 6 by i++
        pro_emo[i] ← 0
    end
    for( int c=0; c<num_training; c++)
        dis ← 0
        double len1 ← 0, len2 ← 0, dot_product ← 0
        for j←0 to word.size() by j++
            dis += ( TF_training[c][j] - TF_new[r][j] )^2
            len1 += TF_training[c][j]
            len2 += TF_new[r][j]
            dot_product += TF_training[c][j] * TF_new[r][j]
        end
        dis= dot_product / sqrt( len1 * len2 )
        knn.insert((dis,c))
    end
```

```
    label[6] ← { "anger", "disgust", "fear", "joy", "sad", "surprise" } ;
    knnit ← knn.begin();
    knnend ← knn.end();
    double pro_sum = 0;
```

```
    for i←0 to 6 by i++
        knnit = knn.begin();
        for x←0 to k && knnit!=knnend by x++, knnit++
            pro_emo[i] += training[ knnit->second ].pro[i] / knnit->first  ;
        end
    end
    for i←0 to 6 by i++
        pro_sum += pro_emo[i]
    end
    for i←0 to 6 by i++
        pro_emo[i] /= pro_sum
    end
end
```

3）NB 分类：

```
string label[6]={ "anger", "disgust", "fear", "joy", "sad", "surprise" }
for i←0 to 6 by i++
    for t←0 to num_training by t++
        if training[t].label=label[i] then
            emotion[i] ++ ;
            for j←0 to word.size() by j++
                Probability[i][j] += Times_training[t][j]
            end
        endif
    end
end
for i←0 to 6 by i++
    for j←0 to word.size() by j++
        all_sum[i] += Probability[i][j]
    end
end
for i←0 to 6 by i++
    for j←0 to word.size() by j++
        Probability[i][j] ← (Probability[i][j]+alpha)/(all_sum[i]+alpha*word.size())
    end
end
```

```
for x←0 to num_new by x++
    for i←0 to 6 by i++
        pro[i] ← emotion[i] / num_training
    end
    for j←0 to voc.size() by j++
        if voc[j] exist in word [loc], then
            for i←0 to 6 by i++
                pro[i] *= Probability[i][loc] ;
            end
        else
            for i←0 to 6 by i++
                pro[i] = pro[i] * alpha / (all_sum[i]+alpha*word.size())
            end
        endif
    end
    predict ← label[x], which makes pro[x]=max( pro[6] )
end
```

4） NB 回归：

```
for x←0 to num_new by x++
    for t←0 to num_training by t++
        for i←0 to 6 by i++
            pro_tmp[t][i] ← training[t].pro[i] ;
        end
    end
    pro_sum ← 0
    for j←0 to voc.size() by j++
        for t←0 to num_training by t++
            for i←0 to 6 by i++
                if voc[j] exist in word [loc], then
                    pro_tmp[t][i] *= TF_training[t][loc] ;
                else
                    pro_tmp[t][i] = pro_tmp[t][i] * alpha /
                        ( training[t].voc.size() + alpha*word.size())
                endif
            end
        end
    end
    for t←0 to num_training by t++
        for i←0 to 6 by i++
            p.pro[i] += pro_tmp[t][i]
        end
    end
    for i←0 to 6 by i++
        pro_sum += p.pro[i]
    end
    for i←0 to 6 by i++
        p.pro[i] /= pro_sum ;
    end
end
```

3. 关键代码截图（带注释）

1） KNN 分类：

```
double max_rate = 0;     // 对不同的K,计算准确率最大值
int max_rate_k = 0;      // 准确率最大值对应的K
for( int k=3; k<=40; k++ )    //KNN算法不同K值
{
    int num_correct = 0;         //测试结果准确的个数
    for( int r=0; r<num_new; r++)  //新的数据集每一篇文章，第r篇文章
    {
        string answer = SplitVec1[2*r+1] ;  //新集第r篇文章的label ，正确答案
        multimap<double,string,less<double>> knn ;
        //对测试集每一篇文章有一个knn的排序，前k个距离最近的文章（距离，标签）对

        for( int c=0; c<num_training; c++) //对训练集的遍历
        {
            double dis=0 ;           // 新集第r篇与训练集第c篇
            for( int j=0; j<word.size(); j++)
            {   // 欧式距离
                dis += ( TF_training[c][j] - TF_new[r][j] )*( TF_training[c][j] - TF_new[r][j] ) ;
            }
            dis = sqrt( dis ) ;  // 只是对距离排序，开根号可以省略
            knn.insert( multimap<double,string>::value_type(dis,SplitVec0[2*c+1])) ;
            // 插入测试集第r篇与训练集第c篇 （距离，标签），自动按照距离排序
        }
```

```
        int count[6] ;
        for(int i=0; i<6; i++)
            count[i] = 0;
        // 对距离最近的K个训练集文章进行“多数投票”
        string label[6]={ "anger", "disgust", "fear", "joy", "sad", "surprise" } ;
        multimap<double,string>::iterator knnit = knn.begin();
        for( int i=0; i<k; i++, knnit++)
        {
            string tmp = knnit->second ;
            // 对前K个训练集文章的标签计数
            if( tmp=="anger")           count[0]++ ;
            else if( tmp=="disgust")    count[1]++ ;
            else if( tmp=="fear")       count[2]++ ;
            else if( tmp=="joy")        count[3]++ ;
            else if( tmp=="sad")        count[4]++ ;
            else if( tmp=="surprise")   count[5]++ ;
            else cout<<endl<<"error:" <<knnit->second <<endl ;
        }
```

```
        int maxNum = 0;      //前k名出现的标签次数的最大值
        int max_i = 0;       //最大值对应的标签下标
        //找出前k名出现的标签次数的最大值
        for( int i=0; i<6; i++ )
        {
            if ( count[i]>maxNum )
            {
                maxNum = count[i] ;
                max_i = i;
            }
        }
        string predict = label[max_i] ;
        //根据最大值对应的标签下标，找到预测的标签结果
        ofile_new <<r+1 <<"," <<predict <<endl ;
        if( predict == answer ) // 验证
            num_correct ++ ;    // 计算准确的个数
    }
    double rate = num_correct*100.0 / num_new ;  // 计算准确率
    printf("%d num_correct=%d  rate= %f\n", k,num_correct, rate );
    // 计算不同K值中，使得准确率最大的K值
    if( rate > max_rate )
    {
        max_rate = rate ;   // 最大准确率
        max_rate_k = k ;    // 准确率最大的K值
    }
}
```

2） KNN 回归

```
double pro_emo[6] ; // 每篇文章6种情感概率
for( int k=15; k==15; k++ )     //KNN算法不同K值
{
    for( int r=0; r<num_new; r++)  //新的数据集每一篇文章，第r篇文章
    {
        for( int i=0; i<6; i++)
            pro_emo[i] = 0 ;   // 每篇文章6种情感概率初始化为1

        multimap<double,int,greater<double>> knn ; //余弦距离
        //multimap<double,int,less<double>> knn ;     //欧式距离
        //对测试集每一篇文章有一个knn的排序，前k个距离最近的文章（距离，标签）
```

```
        multimap<double,int,greater<double>> knn ; //余弦距离
        //对测试集每一篇文章有一个knn的排序，前k个距离最近的文章（距离，标签）
        for( int c=0; c<num_training; c++) //对训练集的遍历
        {
            double dis=0 ;          // 新集第r篇与训练集第c篇
            double len1 = 0, len2 = 0, dot_product = 0 ; // 辅助计算余弦距离
            for( int j=0; j<word.size(); j++)
            {   // 余弦距离
                dis += (TF_training[c][j]-TF_new[r][j])*(TF_training[c][j]-TF_new[r][j]) ;
                len1 += TF_training[c][j]*TF_training[c][j] ;
                len2 += TF_new[r][j]*TF_new[r][j] ;
                dot_product += TF_training[c][j] * TF_new[r][j] ;
            }
            dis= dot_product / sqrt( len1 * len2 ) ;
            knn.insert( multimap<double,int>::value_type(dis,c)) ;
            // 插入测试集第r篇与训练集第c篇 （距离，标签），自动按照距离排序
        }
```

```
         string label[6]={ "anger", "disgust", "fear", "joy", "sad", "surprise" } ;
         multimap<double,int>::iterator knnit = knn.begin();
         multimap<double,int>::iterator knnend = knn.end();
         double pro_sum = 0;
         for( int i=0; i<6; i++)
         {
             knnit = knn.begin();
             for( int x=0; x<k,knnit!=knnend; x++, knnit++)
             {
                 pro_emo[i] += training[ knnit->second ].pro[i] / knnit->first  ;
             }
         }
         //概率归一化
         for( int i=0; i<6; i++)
         {
             pro_sum += pro_emo[i];
         }
         for( int i=0; i<6; i++)
         {
             pro_emo[i] /= pro_sum ;
         }
         ofile_new <<r+1 <<"," <<pro_emo[0] <<"," <<pro_emo[1] <<"," <<pro_emo[2]
                   <<"," <<pro_emo[3] <<"," <<pro_emo[4] <<"," <<pro_emo[5]<<endl ;
    }
}
```

3）NB 分类

```
/* 分割单词时，NB分类计算概率 */
double pro[6] ;  // 每种情感的概率
for (int x = 0; x<num_new; x++) //测试集的第x篇文章
{
    Passage p;                          //构造第x篇文章
    p.label = SplitVec1[2*x+1] ;        //每一篇文章的label
    string tmp = SplitVec1[2*x] ;       //第x篇文章没有截断前
    split( p.voc, tmp , is_any_of(" "), token_compress_on ); //构造第x篇文章的单词
```

```
    // 每篇文章的情感概率初始化 P(Ei)
    for( int i=0; i<6; i++)
        pro[i] = 1.0 * emotion[i] / num_training;

    double max_pro = 0 ;    // 情感概率最大值
    int max_i = 0;          // 情感概率最大值对应情感下标
```

```
    double max_pro = 0 ;    // 情感概率最大值
    int max_i = 0;          // 情感概率最大值对应情感下标
    for( int j=0; j<p.voc.size(); j++ ) // 查看文章中每个单词
    {
        string subtmp = p.voc[j] ;  // 第x篇文章的第j个单词
        //单词是否已经存在单词库
        vector<string>::iterator iterloc = find( word.begin(), word.end() , subtmp ) ;
        if( iterloc!= word.end() )  //单词表已经存在这个单词
        {
            int loc = iterloc-word.begin() ;    //第j个单词所在单词库位置
            for( int i=0; i<6; i++)
                pro[i] *= Probability[i][loc] ;
            // 根据NB分类公式，累积 P(Xj|Ei)P(Ei)
        }
        else //新单词,原本概率为0，进行拉普拉斯平滑
        {
            for( int i=0; i<6; i++)
            {   //拉普拉斯平滑
                pro[i] = pro[i] * alpha / ( all_sum[i] + alpha*word.size()) ;
            }
        }
    }
```

```
    // 计算测试集第x篇文章，哪一种情感概率高
    for( int i=0; i<6; i++)
    {
        if( pro[i]>max_pro && pro[i]<=1 )
        {
            max_pro = pro[i] ;
            max_i = i ;
        }
    }
    ofile_new << x+1 <<"," <<label[max_i] <<endl ;
    if( label[max_i] == p.label )
    {   // 预测结果==正确答案?
        num_correct ++ ; //计算准确率
    }
}
/* FINISHED */
```

4）NB 回归

```
/* 稀疏矩阵三元组转成二维数组  */
map<Pos, double>::iterator it = TF_map.begin();
map<Pos, double>::iterator end = TF_map.end();
for( ; it!=end; it++ )
    TF_training[it->first.row][it->first.col] = it->second ;
/* FINISHED 稀疏矩阵三元组转成二维数组  */
```

```
/* 单词在训练集的次数矩阵->TF词频矩阵 laplace平滑 */
for( int i=0; i<num_training; i++ )
{   // 每篇文章单词共同的分母
    double denominator = training[it->first.row].voc.size() + alpha*word.size() ;
    for( int j=0; j<word.size(); j++ )
        TF_training[i][j] = ( TF_training[i][j] + alpha )/denominator;
}
/* FINISHED  单词在训练集的次数矩阵->TF词频矩阵  */
```

```
for (int x = 0; x<num_new; x++) //测试集的第x篇文章
{
    Passage p;      //构造第x篇文章
    string tmp = SplitVec1[7*x] ;           //第x篇文章没有截断前
    split( p.voc, tmp , is_any_of(" "), token_compress_on ); //构造第x篇文章的单词

    /* 构造pro_tmp[t][i] 第t篇文章第i种情感的累加分量  ∏[P(Xj|dt )P(Ei)]*/
    for( int t=0; t<num_training; t++)
    {
        for( int i=0; i<6; i++ )
        {   // 初始化为P(Ei)
            pro_tmp[t][i] = training[t].pro[i] ;
        }
    }
    double pro_sum = 0 ;  // 每篇文章6种情感概率求和,初始化为0 (为归一化)
```

```
    for( int j=0; j<p.voc.size(); j++ ) //文章句子里的第j个单词
    {   // 遍历训练集，找出标签为Ei的文章
        string subtmp = p.voc[j] ;     // 第x篇文章的第j个单词
        //单词是否已经存在单词库
        vector<string>::iterator iterloc = find(word.begin(),word.end(),subtmp);
        vector<string>::iterator wordend = word.end() ;
        for( int t=0; t<num_training; t++)  //训练集中的第t篇文章
        {
            for( int i=0; i<6; i++)     //第x篇文章的第i种情感标签
            {
                if( iterloc!= wordend )  //单词表已经存在这个单词
                {
                    int loc = iterloc-word.begin(); //第j个单词所在单词库位置
                    pro_tmp[t][i] *= TF_training[t][loc] ;
                }
                else //新单词
                {
                    pro_tmp[t][i] = pro_tmp[t][i] * alpha /
                        ( training[t].voc.size() + alpha*word.size()) ;
                    // 拉普拉斯平滑
                }
            }
        }
    } /* FINISHED 构造pro_tmp[t][i] */
```

```
    /* 情感为Ei的概率=对所有文章的累积分量求和 */
    for( int t=0; t<num_training; t++)
    {
        for( int i=0; i<6; i++)
        {   //得到该文章为情感为Ei的概率
            p.pro[i] += pro_tmp[t][i] ;
        }
    }
    /* FINISHED 该文章情感为Ei的概率 */
    /*  6种情感概率归一化 */
    for( int i=0; i<6; i++)
    {
        pro_sum += p.pro[i] ;
    }
    for( int i=0; i<6; i++ )
    {
        p.pro[i] /= pro_sum ;
    }
    /* FINISHED 6种情感概率归一化 */
    ofile_new <<x+1 <<"," <<p.pro[0] <<"," <<p.pro[1] <<"," <<p.pro[2]
                    <<"," <<p.pro[3] <<"," <<p.pro[4] <<"," <<p.pro[5] <<endl ;
} // end for(int x = 0; x==0; x++)  //新集的第x篇文章
```

4. 创新点&优化（如果有）

平滑的必要性：

无论是分类中求$P(x_k|C_i)$ 还是回归求$P(x_k|d_i)$，都会因为x_k不存在使得 P 为 0，到时后面连乘会导致计算的概率为 0.

朴素贝叶斯的拉普拉斯平滑

α :平滑参数，一般$\alpha \in [0,1]$；

W：训练集单词库大小（不重复单词数）

- **NB 分类：**

原始公式：

$$P(x_k|C_i) = \frac{标签为C_i中出现单词x_k数目}{标签为C_i中所有单词总数} = \frac{nw_{e_i}(x_k)}{nw_{e_i}}$$

平滑公式：

$$P(x_k|C_i) = \frac{标签为C_i中出现单词x_k数目 + \alpha}{标签为C_i中所有单词总数 + \alpha \times W} = \frac{nw_{e_i}(x_k) + \alpha}{nw_{e_i} + \alpha \times W}$$

平滑公式依据：

$\sum_k^W P(x_k|C_i) = 1$

- **NB 回归：**

原始公式：

$$P(x_k|d_i) = TF(词频) = \frac{文章d_i中出现单词x_k数目}{文章d_i中所有单词总数} = \frac{nd_{e_i}(x_k)}{nd}$$

平滑公式：

$$P(x_k|d_i) = \frac{文章d_i中出现单词x_k数目 + \alpha}{文章d_i中所有单词总数 + \alpha \times W} = \frac{nd_{e_i}(x_k) + \alpha}{nd + \alpha \times W}$$

平滑公式依据：

$\sum_k^W P(x_k|d_i) = 1$

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）【小数据集】

1）KNN 分类，训练集：

Words (split by space)	label
hello introduce these are TAs	joy
hello this is a test	joy
some TAs have no girlfriend	sad
some TAs have girlfriend	joy
TAs live happy	joy
hello you all have no girfriend	sad

测试集：

Words (split by space)	label
some of you have no girlfirend	?

结果展示：（包括 TF 矩阵和 knn 前 K 个距离和标签，最终预测结果）

```
单词数量17
I can not find test hello this is my girlfriend She luhan's am his girfriend stand grilfriend
** TF_training **
0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0.25 0 0.25 0.25 0.25 0 0 0 0 0
0.2 0 0.2 0 0 0 0 0 0 0 0 0 0.2 0.2 0.2 0 0
0.166667 0.166667 0.166667 0 0 0 0 0 0 0.166667 0 0 0 0.166667 0 0.166667 0

** TF_new **
0.222222 0.111111 0.222222 0 0 0 0 0 0 0 0 0.111111 0.111111 0 0 0.111111 0.111111
**********

Onew passage:
  K =3
  0.323942,anger
  0.371849,sad
  0.371849,sad

Predict: No.1,  label:sad
```

分析：输出结果和手算的 TF 矩阵和 KNN 算法一样。正确。

2）KNN 回归

训练集：

Words (split by space)	anger	disgust	fear	joy	sad	surprise
I buy an apple phone	0	0	0	0.8	0	0.2
I eat the big apple	0	0.1	0	0.6	0	0.3
the apple products are too expensive	0.3	0.1	0.1	0.1	0.3	0.1

测试集：

Words (split by space)	anger	disgust	fear	joy	sad	surprise
my friend has an apple	?	?	?	?	?	?

输出结果：

```
单词数量15
I buy an apple phone eat the big products are too expensive my friend has
** TF_training **
0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0 0 0 0
0.2 0 0 0.2 0 0.2 0.2 0.2 0 0 0 0 0 0 0
0 0 0 0.166667 0 0 0.166667 0 0.166667 0.166667 0.166667 0.166667 0 0 0

** TF_new **
0 0 0.2 0.2 0 0 0 0 0 0 0 0.2 0.2 0.2
**********
```

id	anger	disgust	fear	joy	sad	surprise
1	0.12	0.08	0.04	0.44	0.12	0.2

分析：经验证，输出结果和手算结果一样。正确。

3）NB 分类

训练集：

Words (split by space)	label
hello introduce these are TAs	joy
hello this is a test	joy
some TAs have no girlfriend	sad
some TAs have girlfriend	joy
TAs live happy	joy
hello you all have no girfriend	sad

测试集：

Words (split by space)	label

some of you have no girlfirend	?

平滑取 α=0.6，输出结果：

```
anger, 0
disgust, 0
fear, 0
joy, 3.023e-010
sad, 1.09332e-008
surprise, 0
predict: sad
```

分析：经验证，输出结果和手算结果一样。正确。

4）NB 回归

训练集：

Words (split by space)	anger	disgust	fear	joy	sad	surprise
I buy an apple phone	0	0	0	0.8	0	0.2
I eat the big apple	0	0.1	0	0.6	0	0.3
the apple products are too expensive	0.3	0.1	0.1	0.1	0.3	0.1

测试集：

Words (split by space)	anger	disgust	fear	joy	sad	surprise
my friend has an apple	?	?	?	?	?	?

OneHot 矩阵

```
1 1 1 1 1 0 0 0 0 0 0 0
1 0 0 1 0 1 1 1 0 0 0 0
0 0 0 1 0 0 1 0 1 1 1 1
```

输出：

id	anger	disgust	fear	joy	sad	surprise
1	0.026694	0.021913	0.008898	0.711686	0.026694	0.204117

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

2.1. KNN 分类

- **TF 矩阵，欧式距离：**

K 值	预测正确个数	rate（%）
3	76	24.437299
4	84	27.009646
5	86	27.652733
6	97	31.189711
7	92	29.581994

8	90	28.938907
9	91	29.26045
10	92	29.581994
11	92	29.581994
12	100	32.154341
13	101	32.475884
14	104	33.440514
15	100	32.154341
16	94	30.22508
17	97	31.189711
18	102	32.797428
19	104	33.440514
20	98	31.511254
21	94	30.22508
22	103	33.118971
23	100	32.154341
24	98	31.511254
25	97	31.189711
26	99	31.832797
27	95	30.546624
28	90	28.938907
29	93	29.903537
30	95	30.546624

分析：

当 K=14，TF 矩阵，欧式距离，准确率有 33.440514%

- **OneHot 矩阵，欧式距离：**

K 值	预测正确个数	rate
3	108	34.7267
4	114	36.6559
5	108	34.7267
6	113	36.3344
7	107	34.4051
8	100	32.1543
9	114	36.6559
10	117	37.6206
11	124	39.8714

12	121	38.9068
13	123	39.5498
14	121	38.9068
15	122	39.2283
16	126	40.5145
17	124	39.8714
18	124	39.8714
19	116	37.299
20	116	37.299
21	120	38.5852
22	118	37.9421
23	119	38.2637
24	121	38.9068
25	119	38.2637
26	122	39.2283
27	120	38.5852
28	123	39.5498
29	120	38.5852
30	120	38.5852

分析：

当 K=16, OneHot 矩阵，欧式距离，有准确率 40.5145%

- **OneHot 矩阵，余弦距离：**

K 值	预测正确个数	rate(%)
3	121	38.9068
4	123	39.5498
5	132	42.4437
6	127	40.836
7	125	40.1929
8	127	40.836
9	133	42.7653
10	134	43.0868
11	135	43.4084
12	126	40.5145
13	126	40.5145
14	130	41.8006

15	125	40.1929
16	127	40.836
17	129	41.4791
18	126	40.5145
19	121	38.9068
20	123	39.5498
21	118	37.9421
22	120	38.5852
23	121	38.9068
24	121	38.9068
25	118	37.9421
26	120	38.5852
27	123	39.5498
28	116	37.299
29	118	37.9421
30	122	39.2283

分析：当取 OneHot 矩阵，余弦距离时：

K=5, rate=42.4437%

K=14, rate=41.8006%

- **TF 矩阵，余弦距离：**

K 值预测	正确个数	rate(%)
3	108	34.726688
4	107	34.405145
5	118	37.942122
6	122	39.228296
7	119	38.263666
8	122	39.228296
9	121	38.906752
10	122	39.228296
11	126	40.514469
12	115	36.977492
13	122	39.228296
14	127	40.836013
15	124	39.871383
16	123	39.549839
17	124	39.871383
18	119	38.263666
19	124	39.871383

20	127	40.836013
21	122	39.228296
22	122	39.228296
23	120	38.585209
24	123	39.549839
25	115	36.977492
26	119	38.263666
27	121	38.906752
28	116	37.299035
29	120	38.585209
30	122	39.228296

分析：当取 TF 矩阵，余弦距离时：

K=14, rate=40.836013%

总结：对比 4 种 KNN 方法，K=14~16 表现较好，(训练样本有 623 个)，故测试集采用 K=14

特征向量	距离计算方法	准确率
OneHot 矩阵	欧氏距离	33.440514%
TF 矩阵	欧氏距离	38.9068%
OneHot 矩阵	余弦距离	41.8006%
TF 矩阵	余弦距离	40.836013%

最终选择 TF 矩阵 余弦距离 K=14，计算 test_set。

2. KNN 回归

！验证集存在 8 个测试样本在训练集出现过（所以样本间距离 dis=0，做分母结果得到 nan，如下图所示），但是由于其答案和测试样本属性值不符合，认为是脏数据，忽略。

id	anger	disgust	fear	joy	sad	surprise
1	0.0849936	0.0520399	0.156251	0.285744	0.191957	0.229015
2	0.0851208	0.0515494	0.156331	0.286649	0.19117	0.22918
3	0.0846877	0.0516578	0.156177	0.286136	0.191951	0.22939
4	0.0846207	0.0515854	0.155777	0.286726	0.192289	0.229003
5	0.0851212	0.051534	0.155548	0.287574	0.190813	0.229409
6	0.0852406	0.0517704	0.154886	0.286753	0.192047	0.229303
7	0.0851008	0.051914	0.156782	0.284209	0.193863	0.228131
8	0.0844735	0.051568	0.1554	0.288241	0.191251	0.229066
9	0.084365	0.0516684	0.155979	0.286849	0.19199	0.229148
10	0.0842203	0.0516653	0.156056	0.285902	0.193215	0.228941
11	0.0848791	0.051844	0.156406	0.285606	0.192271	0.228995
12	nan	nan	nan	nan	nan	nan
13	0.0852078	0.0520252	0.155473	0.28534	0.192808	0.229146
14	0.0847847	0.0518766	0.156171	0.283472	0.194855	0.228841
15	0.0847605	0.051943	0.156238	0.28616	0.192121	0.228777
16	0.0845571	0.0514358	0.154705	0.288261	0.191229	0.229812

为此，更新了 KNN 回归代码：如果距离为 0，那么直接复制训练集的情感概率。

特征向量	距离计算方法	K 值	回归相关系数
OneHot 矩阵	欧氏距离	14	0.298814889
TF 矩阵	欧氏距离	14	0.279291779

3. NB 分类

alpha	预测正确个数	准确率
0.02	123	0.3955
0.04	124	0.39871
0.06	125	0.40193
0.08	127	0.40836
0.1	127	0.40836
0.2	132	0.42444
0.3	134	0.43087
0.4	135	0.43408
0.5	137	0.44051
0.6	141	0.45338
0.62	142	0.45659
0.64	142	0.45659
0.66	140	0.45016
0.68	140	0.45016
0.7	139	0.44695
0.8	139	0.44695
0.9	140	0.45016
0.92	140	0.45016
0.94	139	0.44695
0.96	137	0.44051
0.98	137	0.44051
1	137	0.44051

4. NB 回归

平滑系数 alpha	回归系数
0.02	0.35448
0.04	0.35661
0.06	0.35467
0.08	0.35072
0.1	0.34584
0.2	0.31889
0.4	0.27339
0.6	0.25205
0.8	0.25488

1	0.27161

四、 思考题

1.

步骤2：根据相似度加权

计算test1与每个train的距离，选取TopK个训练数据
把该距离的倒数作为权重，计算test1属于该标签的概率：

$$P(test1\ is\ happy) = \frac{train1\ probability}{d(train1, test1)} + \frac{train2\ probability}{d(train2, test1)} + \frac{train3\ probability}{d(train3, test1)}$$

思考：为什么是倒数呢？

思考：同一测试样本的各个情感概率总和应该为1 如何处理？

1) 为什么是倒数？

测试样本的情感概率=距离最近的 k 个训练样本情感概率*权重

权重=1/距离，因为我们认为距离越小的两个文本的相似度越大，权重应该越大，所以权重和距离是反相关关系。

拓展：采用标准化距离 (Standardized Distance) .

考虑这不是一个文本数据，而是一个预测房价的训练数据，那么每个样本的数据变量可能有不同的度量尺度，例如，如果一个变量以年收入为基础（大约是几千几万的数据），而另一个则是基于年龄（大约是几十的数据），那么收入将对计算出的距离有更大的影响。标准化权重，使得年收入和年龄的权重取值为[0,1]。标准化公式：

$$X_s = \frac{X - Min}{Max - Min}$$

2) 同一个测试样本的各个情感综合为 1，如何处理？

根据概率论，一个样本的不同取值的概率求和为 1.

方法：归一化。已知样本可能取值有 6 种，

$$sum = \sum_{i=1}^{6} P(e_i)$$

$$P(e_i) = \frac{P(e_i)}{sum}$$

2.

不同距离度量方式

- 距离公式：

L_p距离(所有距离的总公式)：

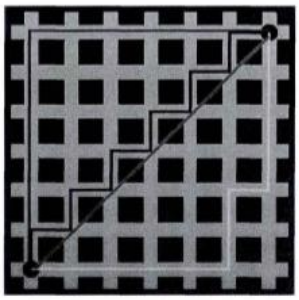
- $L_p(x_i,x_j)=\left\{\sum_{i=1}^{n}\left|x_i^{(l)}-x_j^{(l)}\right|^p\right\}^{\frac{i}{p}}$

- $p=1$：曼哈顿距离；
- $p=2$：欧式距离，最常见。

思考：在矩阵稀疏程度不同的时候，这两者表现有什么区别，为什么？

曼哈顿距离：横纵坐标绝对值之和的值代表。欧式距离：两点直接的直线距离。

下图中除了两点直线外，其他长度均为曼哈顿距离：



一般来说，两个文本的特征向量形如：

$$X=(x_1,x_2,\dots x_n)$$
$$Y=(y_1,y_2,\dots y_n)$$

为了简便，现在设两个向量 $X=(x_1,x_2)$ 和 $Y=(y_1,y_2)$

设

$$a=(x_1-y_1)^2$$
$$b=(x_2-y_2)^2$$

曼哈顿距离：$|x_1-y_1|+|x_2-y_2|=\sqrt{(x_1-y_1)^2}+\sqrt{(x_2-y_2)^2}=\sqrt{a}+\sqrt{b}$

欧氏距离：$\sqrt{(x_1-y_1)^2+(x_2-y_2)^2}=\sqrt{a+b}$

$$\sqrt{a}+\sqrt{b}>\sqrt{a+b}$$

曼哈顿距离>欧氏距离，且从上图也可以看出，两点之间直线最短，欧氏距离较小。

$$\left(\sqrt{a}+\sqrt{b}\right)^2-\left(\sqrt{a+b}\right)^2=2\sqrt{ab}$$

合理假设，稀疏矩阵的稀疏程度越大，$(x_i-y_i)^2$中出现x_i或者y_i为 0 的概率越大，$\sqrt{ab}$越大，所以曼哈顿距离 - 欧氏距离的差越大。

从运算上看，欧氏距离由于要平方在开方，运算量比曼哈顿距离要打；

从距离的意义看，欧氏距离的几何含义更强，因为他是“两点之间的直线距离”

3.

Example

ID	text	class label
1	good,thanks	joy
2	No impressive, thanks	sad
3	Impressive good	joy
4	No, thanks	?

ID	goods	thanks	no	impressive	class label
1	1	1	0	0	joy
2	0	1	1	1	sad
3	1	0	0	1	joy
4	0	1	1	0	?

Bernoulli Model（伯努利模型）：

$P_{(thanks|joy)} = 1/2$

Multinomial Model（多项式模型）：

$P_{(thanks|joy)} = 1/4$

思考题：这两个模型分别有什么优缺点

伯努利模型：

$$P(x_k|C_i) = \frac{标签为C_i中出现文章x_k数目}{标签为C_i中所有文章总数} = \frac{nd_{e_i}(x_k)}{nd_{e_i}}$$

伯努利模型的统计更多从文章的角度考虑，是一种“离散”的感觉。联想到 KNN 分类算法（考虑前 K 个最近样本中出现次数最多的标签）。因此，伯努利模型更适合于 NB 分类问题。

多项式模型：

$$P(x_k|C_i) = \frac{标签为C_i中出现单词x_k数目}{标签为C_i中所有单词总数} = \frac{nw_{e_i}(x_k)}{nw_{e_i}}$$

对比与伯努利模型，多项式模型更多的考虑整一个数据集而不是离散的文章上，更多“连续”的感觉，因此更适合于 NB 回归问题。

无论是伯努利模型还是多项式模型都没有考虑到高频词汇的影响。如“my”“a”，这两种模型中他们的概率都比较大，却并不能很好的说明了对情感的影响。高频词汇会使得一些有特征的单词比如“scandal”，“vote”的$P(x_k|C_i)$ 变小。

4. 如果测试集中出现了一个之前全词典中没有出现过的词该如何解决？

在 KNN 中，对于新的单词就插入到全词典中去。在计算样本距离的时候，原本的训练集对应新单词的 OneHot 矩阵，TF 矩阵补 0.

在 NB 中，对于新的单词，因为训练集中没有的单词这个单词，所以$P(x_k|C_i)$ 原本为 0.

可以利用拉普拉斯平滑，$P(x_k|C_i) = \frac{nd_{e_i}(x_k)+\alpha}{nd+\alpha\times W} = \frac{\alpha}{\alpha\times W}$

完。

谢谢评阅 :)

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想