“智慧政务”中的文本挖掘应用

摘要

本文在利用自然语言处理和文本挖掘的方法前提下，对群众留言划分和热点整理问题进行大数据分析，综合运用 Python、Excel 等多种工具，求得了原题方案中的查准率、查全率等相关信息，绘制系统的 PR 曲线来判断系统的优劣，并建立模型进行评价。

问题一的关键在于对留言问题分类的方法。首先，我们整理出出各个区的留言数量和分类留言的情感倾向，做出相应的图表并作出分析；然后进行留言主题分析，对于语料进行 LDA 建模，使用最大似然估计进行最优化主题个数的选取。选定主题个数并将高频词取出做出图表分析，计算出各个类别的查准率、查全率，绘制系统的 PR 曲线来判断系统的优劣并使用 F-Score 对分类方法进行评价。我们在解决该问题中使用的是朴素贝叶斯算法建立模型，我们首先将一些无关字符与词语剔除，留下一些关键性词语，利用该词语在哪类中的概率最高从而进行分类。以下是部分程序编码用来分类。（网上查找）

```
#encoding:UTF-8
'''
Author: marco lin
Date: 2015-08-28
'''


from numpy import *
import pickle
import jieba
import time


stop_word = []
'''
    停用词集, 包含“啊，吗，嗯”一类的无实意词汇以及标点符号
'''
def loadStopword():
    fr = open('stopword.txt', 'r')
    lines = fr.readlines()
    for line in lines:
        stop_word.append(line.strip().decode('utf-8'))
    fr.close()

'''
    创建词集
    params:
        documentSet 为训练文档集
    return:词集, 作为词袋空间
'''
def createVocabList(documentSet):
```

```
        vocabSet = set([])
        for document in documentSet:
            vocabSet = vocabSet | set(document) #union of the two sets
        return list(vocabSet)

    '''
        载入数据
    '''
    def loadData():
        return None

    '''
       文本处理，如果是未处理文本，则先分词（jieba 分词）,再去除停用词
    '''
    def textParse(bigString, load_from_file=True):    #input is big string,
#output is word list
        if load_from_file:
            listOfWord = bigString.split('/ ')
            listOfWord = [x for x in listOfWord if x != ' ']
            return listOfWord
        else:
            cutted = jieba.cut(bigString, cut_all=False)
            listOfWord  = []
            for word in cutted:
                if word not in stop_word:
                    listOfWord.append(word)
            return [word.encode('utf-8') for word in listOfWord]

    '''
        交叉训练
    '''
    CLASS_AD        = 1
    CLASS_NOT_AD    = 0

    def testClassify():
        listADDoc = []
        listNotADDoc = []
        listAllDoc = []
        listClasses = []

        print "----loading document list----"

        #两千个标注为广告的文档
        for i in range(1, 1001):
```

```
            wordList = textParse(open('subject/subject_ad/%d.txt' % i).read())
            listAllDoc.append(wordList)
            listClasses.append(CLASS_AD)
        #两千个标注为非广告的文档
        for i in range(1, 1001):
            wordList = textParse(open('subject/subject_notad/%d.txt' % i).read())
            listAllDoc.append(wordList)
            listClasses.append(CLASS_NOT_AD)

        print "----creating vocab list----"
        #构建词袋模型
        listVocab = createVocabList(listAllDoc)

        docNum = len(listAllDoc)
        testSetNum  = int(docNum * 0.1);

        trainingIndexSet = range(docNum)    # 建立与所有文档等长的空数据集（索引）
        testSet = []                         # 空测试集

        # 随机索引，用作测试集，同时将随机的索引从训练集中剔除
        for i in range(testSetNum):
            randIndex = int(random.uniform(0, len(trainingIndexSet)))
            testSet.append(trainingIndexSet[randIndex])
            del(trainingIndexSet[randIndex])

        trainMatrix = []
        trainClasses = []

        for docIndex in trainingIndexSet:
            trainMatrix.append(bagOfWords2VecMN(listVocab,
listAllDoc[docIndex]))
            trainClasses.append(listClasses[docIndex])

        print "----traning begin----"
        pADV, pNotADV, pClassAD = trainNaiveBayes(array(trainMatrix),
array(trainClasses))

        print "----traning complete----"
        print "pADV:", pADV
        print "pNotADV:", pNotADV
        print "pClassAD:", pClassAD
        print "ad: %d, not ad:%d" % (CLASS_AD, CLASS_NOT_AD)

        args = dict()
```

```
        args['pADV'] = pADV
        args['pNotADV'] = pNotADV
        args['pClassAD'] = pClassAD

        fw = open("args.pkl", "wb")
        pickle.dump(args, fw, 2)
        fw.close()

        fw = open("vocab.pkl", "wb")
        pickle.dump(listVocab, fw, 2)
        fw.close()

        errorCount = 0
        for docIndex in testSet:
            vecWord = bagOfWords2VecMN(listVocab, listAllDoc[docIndex])
            if classifyNaiveBayes(array(vecWord), pADV, pNotADV, pClassAD) !=
listClasses[docIndex]:
                errorCount += 1
                doc = ' '.join(listAllDoc[docIndex])
                print "classfication error", doc.decode('utf-8',
"ignore").encode('gbk')
        print 'the error rate is: ', float(errorCount) / len(testSet)

    # 分类方法(这边只做二类处理)
    def classifyNaiveBayes(vec2Classify, pADVec, pNotADVec, pClass1):
        pIsAD = sum(vec2Classify * pADVec) + log(pClass1)    #element-wise mult
        pIsNotAD = sum(vec2Classify * pNotADVec) + log(1.0 - pClass1)

        if pIsAD > pIsNotAD:
            return CLASS_AD
        else:
            return CLASS_NOT_AD

    '''
        训练
        params:
            tranMatrix 由测试文档转化成的词空间向量 所组成的 测试矩阵
            tranClasses 上述测试文档对应的分类标签
    '''
    def trainNaiveBayes(trainMatrix, trainClasses):
        numTrainDocs = len(trainMatrix)
        numWords = len(trainMatrix[0]) #计算矩阵列数, 等于每个向量的维数
        numIsAD  = len(filter(lambda x: x == CLASS_AD, trainClasses))
        pClassAD = numIsAD / float(numTrainDocs)
```

```
    pADNum = ones(numWords); pNotADNum = ones(numWords)
    pADDenom = 2.0; pNotADDenom = 2.0

    for i in range(numTrainDocs):
        if trainClasses[i] == CLASS_AD:
            pADNum += trainMatrix[i]
            pADDenom += sum(trainMatrix[i])
        else:
            pNotADNum += trainMatrix[i]
            pNotADDenom += sum(trainMatrix[i])

    pADVect = log(pADNum / pADDenom)
    pNotADVect = log(pNotADNum / pNotADDenom)

    return pADVect, pNotADVect, pClassAD

'''
    将输入转化为向量，其所在空间维度为 len(listVocab)
    params:
        listVocab-词集
        inputSet-分词后的文本，存储于 set
'''
def bagOfWords2VecMN(listVocab, inputSet):
    returnVec = [0]*len(listVocab)
    for word in inputSet:
        if word in listVocab:
            returnVec[listVocab.index(word)] += 1
    return returnVec

'''
    读取保存的模型，做分类操作
'''
def adClassify(text):
    fr = open("args.pkl", "rb")
    args = pickle.load(fr)
    pADV        = args['pADV']
    pNotADV     = args['pNotADV']
    pClassAD    = args['pClassAD']
    fr.close()

    fr = open("vocab.pkl", "rb")
    listVocab = pickle.load(fr)
    fr.close()
```

```
        if len(listVocab) == 0:
            print "got no args"
            return

        text = textParse(text, False)
        vecWord = bagOfWords2VecMN(listVocab, text)
        class_type = classifyNaiveBayes(array(vecWord), pADV, pNotADV, pClassAD)

        print "classfication type:%d" % class_type


if __name__ == "__main__":
    loadStopword()
    while True:
        opcode = raw_input("input 1 for training, 2 for ad classify: ")
        if opcode.strip() == "1":
            begtime = time.time()
            testClassify()
            print "cost time total:", time.time() - begtime
        else:
            text = raw_input("input the text:")
            adClassify(text)
```

问题二：对于热点问题的处理我们使用了数据挖掘算法之聚类规则挖掘。

我们经常接触到的聚类分析，一般都是数值聚类，一种常见的做法是同时提取 N 种特征，将它们放在一起组成一个 N 维向量，从而得到一个从原始数据集合到 N 维向量空间的映射——你总是需要显式地或者隐式地完成这样一个过程，然后基于某种规则进行分类，在该规则下，同组分类具有最大的相似性。

假设我们提取到原始数据的集合为(x_1, x_2, …, x_n)，并且每个 xi 为 d 维的向量，K-means 聚类的目的就是，在给定分类组数 k（$k \le n$）值的条件下，将原始数据分成 k 类

$S = \{S_1, S_2, \ldots, S_k\}$，在数值模型上，即对以下表达式求最小值：

$$\arg\min_{\mathbf{S}} \sum_{i=1}^{k} \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \boldsymbol{\mu}_i\|^2$$

这里μ_i 表示分类 S_i 的平均值。

那么在计算机编程中，其又是如何实现的呢？其算法步骤一般如下：

1、从 D 中随机取 k 个元素，作为 k 个簇的各自的中心。

2、分别计算剩下的元素到 k 个簇中心的相异度，将这些元素分别划归到相异度最低的簇。

3、根据聚类结果，重新计算 k 个簇各自的中心，计算方法是取簇中所有元素各自维度的算术平均数。

4、将 D 中全部元素按照新的中心重新聚类。

5、重复第 4 步，直到聚类结果不再变化。

6、将结果输出。

用数学表达式来说，

设我们一共有 N 个数据点需要分为 K 个 cluster ，k-means 要做的就是最小化

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|x_n - \mu_k\|^2$$

这个函数，其中 r_{nk} 在数据点 n 被归类到 cluster k 的时候为 1 ，否则为 0 。直接寻找 r_{nk} 和 μ_k 来最小化 J 并不容易，不过我们可以采取迭代的办法：先固定 μ_k ，选择最优的 r_{nk} ，很容易看出，只要将数据点归类到离他最近的那个中心就能保证 J 最小。下一步则固定 r_{nk}，再求最优的 μ_k。将 J 对 μ_k 求导并令导数等于零，很容易得到 J 最小的时候 μ_k 应该满足：

$$\mu_k = \frac{\sum_n r_{nk} x_n}{\sum_n r_{nk}}$$

亦即 μ_k 的值应当是所有 cluster k 中的数据点的平均值。由于每一次迭代都是取到 J 的最小值，因此 J 只会不断地减小（或者不变），而不会增加，这保证了 k-means 最终会到达一个极小值。虽然 k-means 并不能保证总是能得到全局最优解，但是对于这样的问题，像 k-means 这种复杂度的算法，这样的结果已经是很不错的了。

首先 3 个中心点被随机初始化，所有的数据点都还没有进行聚类，默认全部都标记为红色，然后进入第一次迭代：按照初始的中心点位置为每个数据点着上颜色，重新计算 3 个中心点，由于初始的中心点是随机选的，这样得出来的结果并不是很好，接下来是下一次迭代的结可以看到大致形状已经出来了。再经过两次迭代之后，基本上就收敛了。不过正如前面所说的那样 k-means 也并不是万能的，虽然许多时候都能收敛到一个比较好的结果，但是也有运气不好的时候会收敛到一个让人不满意的局部最优解。K-means 的源码实现（网上查找）

```
1  #!/usr/bin/python
2
3 from __future__ import with_statement
4 import cPickle as pickle
5 from matplotlib import pyplot
6 from numpy import zeros, array, tile
7 from scipy.linalg import norm
8 import numpy.matlib as ml
9 import random
10
11 def kmeans(X, k, observer=None, threshold=1e-15, maxiter=300):
12     N = len(X)
13     labels = zeros(N, dtype=int)
14     centers = array(random.sample(X, k))
```

```
15      iter = 0
16
17      def calc_J():
18          sum = 0
19          for i in xrange(N):
20              sum += norm(X[i]-centers[labels[i]])
21          return sum
22
23      def distmat(X, Y):
24          n = len(X)
25          m = len(Y)
26          xx = ml.sum(X*X, axis=1)
27          yy = ml.sum(Y*Y, axis=1)
28          xy = ml.dot(X, Y.T)
29
30          return tile(xx, (m, 1)).T+tile(yy, (n, 1)) - 2*xy
31
32      Jprev = calc_J()
33      while True:
34          # notify the observer
35          if observer is not None:
36              observer(iter, labels, centers)
37
38          # calculate distance from x to each center
39          # distance_matrix is only available in scipy newer than
0.7
```

```
40          # dist = distance_matrix(X, centers)
41          dist = distmat(X, centers)
42          # assign x to nearst center
43          labels = dist.argmin(axis=1)
44          # re-calculate each center
45          for j in range(k):
46              idx_j = (labels == j).nonzero()
47              centers[j] = X[idx_j].mean(axis=0)
48
49          J = calc_J()
50          iter += 1
51
52          if Jprev-J < threshold:
53              break
54          Jprev = J
55          if iter >= maxiter:
56              break
57
58      # final notification
59      if observer is not None:
60          observer(iter, labels, centers)
61
62  if __name__ == '__main__':
63      # load previously generated points
64      with open('cluster.pkl') as inf:
65          samples = pickle.load(inf)
```

```
66      N = 0
67      for smp in samples:
68          N += len(smp[0])
69      X = zeros((N, 2))
70      idxfrm = 0
71      for i in range(len(samples)):
72          idxto = idxfrm + len(samples[i][0])
73          X[idxfrm:idxto, 0] = samples[i][0]
74          X[idxfrm:idxto, 1] = samples[i][1]
75          idxfrm = idxto
76
77      def observer(iter, labels, centers):
78          print "iter %d." % iter
79          colors = array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
80          pyplot.plot(hold=False)  # clear previous plot
81          pyplot.hold(True)
82
83          # draw points
84          data_colors=[colors[lbl] for lbl in labels]
85              pyplot.scatter(X[:, 0], X[:, 1], c=data_colors,
alpha=0.5)
86          # draw centers
87            pyplot.scatter(centers[:, 0], centers[:, 1], s=200,
c=colors)
88
89                pyplot.savefig('kmeans/iter_%02d.png' % iter,
format='png')
```

90

91 kmeans(X, 3, observer=observer)

问题三：对于答复意见的评价我们使用的是数据挖掘算法之神经网络

参考文献：博客园网站数据挖掘相关材料。