夜与周公

文本挖掘之特征选择(python 实现)

2013-08-15 10:32 夜与周公 阅读(29398) 评论(17) 编辑 收藏

机器学习算法的空间、时间复杂度依赖于输入数据的规模，维度规约(Dimensionality reduction)则是一种被用于降低输入数据维数的方法。维度规约可以分为两类：

- 特征选择(feature selection)，从原始的d维空间中，选择为我们提供信息最多的k个维(这k个维属于原始空间的子集)
- 特征提取(feature extraction)，将原始的d维空间映射到k维空间中(新的k维空间不输入原始空间的子集)

在文本挖掘与文本分类的有关问题中，常采用特征选择方法。原因是文本的特征一般都是单词(term)，具有语义信息，使用特征选择找出的k维子集，仍然是单词作为特征，保留了语义信息，而特征提取则找k维新空间，将会丧失了语义信息。

对于一个语料而言，我们可以统计的信息包括文档频率和文档类比例，所有的特征选择方法均依赖于这两个统计量，目前，文本的特征选择方法主要有：DF, MI, IG, CHI，WLLR,WFO六种。

为了方便描述，我们首先一些概率上的定义：

p(t):一篇文档x包含特征词t的概率。

$p(\overline{C_i})$:文档x不属于C_i的概率。

$p(C_i|t)$:已知文档x的包括某个特征词t条件下，该文档属于C_i的概率

$p(\bar{t}|C_i)$: 已知文档属于C_i 条件下，该文档不包括特征词t的概率

类似的其他的一些概率如$p(C_i)$， $p(\bar{t})$， $p(\overline{C_i}|t)$等，有着类似的定义。

为了估计这些概率，我们需要通过统计训练样本的相关频率信息，如下表：

特征 \ 类别	C_j	$\overline{C_j}$	总数
t_i	A_{ij}	B_{ij}	$A_{ij}+B_{ij}$
$\bar{t}_i$	C_{ij}	D_{ij}	$C_{ij}+D_{ij}$
总数	$A_{ij}+C_{ij}$	$B_{ij}+D_{ij}$	N

其中：

A_{ij}: 包含特征词t_i，并且类别属于C_j的文档数量

B_{ij}: 包含特征词t_i，并且类别属于不C_j的文档数量

C_{ij}：不包含特征词t_i，并且类别属于C_j的文档数量 D_{ij}：不包含特征词t_i，并且类别属于不C_j的文档数量

$A_{ij}+B_{ij}$: 包含特征词t_i的文档数量 $C_{ij}+D_{ij}$：不包含特征词t_i的文档数量

A_{ij} + C_{ij}：C_j类的文档数量数据 B_{ij} + D_{ij}：非C_j类的文档数量数据

A_{ij} + B_{ij} + C_{ij} + D_{ij} = N :语料中所有文档数量。

有了这些统计量，有关概率的估算就变得容易，如：

$p(t_i)$ = $(A_{ij} + B_{ij}) / N$; $p(C_j)$ = $(A_{ij} + C_{ij}) / N$;

$p(C_j|t_j)$ = $A_{ij} / (A_{ij} + B_{ij})$

.....类似的一些概率计算可以依照上表计算。

介绍了事情发展的前因，现在进入正题：常见的四种特征选择方法如何计算。

1）DF(Document Frequency)

DF:统计特征词出现的文档数量，用来衡量某个特征词的重要性，DF的定义如下：

$$DF = \sum_{i=1}^{m} A_i$$

DF的动机是，如果某些特征词在文档中经常出现，那么这个词就可能很重要。而对于在文档中出现很少(如仅在语料中出现1次)特征词，携带了很少的信息量，甚至是"噪声"，这些特征词，对分类器学习影响也是很小。

DF特征选择方法属于无监督的学习算法(也有将其改成有监督的算法，但是大部分情况都作为无监督算法使用)，仅考虑了频率因素而没有考虑类别因素，因此，DF算法的将会引入一些没有意义的词。如中文的"的"、"是"， "个"等，常常具有很高的DF得分，但是，对分类并没有多大的意义。

2）MI(Mutual Information)

互信息法用于衡量特征词与文档类别直接的信息量，互信息法的定义如下：

$$I(t_i, C_j) = \log \frac{p(t_i|C_j)}{p(t_i)} \approx \frac{A_{ij} N}{(A_{ij} + C_{ij})(A_{ij} + B_{ij})}$$

继续推导MI的定义公式：

$$I(t_i, C_j) = \log \frac{p(t_i|C_j)}{p(t_i)} = \log p(t_i|C_j) - \log p(t_i)$$

从上面的公式上看出：如果某个特征词的频率很低，那么互信息得分就会很大，因此互信息法倾向"低频"的特征词。相对的词频很高的词，得分就会变低，如果这词携带了很高的信息量，互信息法就会变得低效。

3）IG(Information Gain)

信息增益法，通过某个特征词的缺失与存在的两种情况下，语料中前后信息的增加，衡量某个特征词的重要性。

信息增益的定义如下：

$$G(t_i) = \{-\sum_{j=1}^{m} p(C_j) \log p(C_j)\} + \{p(t_i)[\sum_{j=1}^{m} p(C_j|t_i) \log p(C_j|t_i)]\} + p(\bar{t}_i)[\sum_{j=1}^{m} p(C_j|\bar{t}_i) \log p(C_j|\bar{t}_i)]$$

依据IG的定义，每个特征词t_i的IG得分前面一部分：$\{-\sum_{j=1}^{m} p(C_j)\log p(C_j)\}$计算值是一样，可以省略。因此，IG的计算公式如下：

$$G(t_i)\approx\{\frac{A_{ij}+B_{ij}}{N}[\sum_{j=1}^{m}\frac{A_{ij}}{A_{ij}+B_{ij}}\log\frac{A_{ij}}{A_{ij}+B_{ij}}]\}+\{\frac{C_{ij}+D_{ij}}{N}[\sum_{j=1}^{m}\frac{C_{ij}}{C_{ij}+D_{ij}}\log\frac{C_{ij}}{C_{ij}+D_{ij}}]\}$$

IG与MI存在关系：

$$G(t_i)=\sum_{j=1}^{m}p(t_i,C_j)I(t_i,C_j)+\sum_{j=1}^{m}p(\overline{t_i},C_j)I(\overline{t_i},C_j)$$

因此，IG方式实际上就是互信息$I(t_i,C_j)$与互信息$I(\overline{t_i},C_j)$加权。

4）CHI(Chi-square)

CHI特征选择算法利用了统计学中的"假设检验"的基本思想：首先假设特征词与类别直接是不相关的，如果利用CHI分布计算出的检验值偏离阈值越大，那么更有信心否定原假设，接受原假设的备则假设：特征词与类别有着很高的关联度。CHI的定义如下：

$$\chi(t_i,C_j)=\frac{N(A_{ij}D_{ij}-C_{ij}B_{ij})^2}{(A_{ij}+C_{ij})(B_{ij}+D_{ij})(A_{ij}+B_{ij})(C_{ij}+D_{ij})}$$

对于一个给定的语料而言，文档的总数N以及C_j类文档的数量，非C_j类文档的数量，他们都是一个定值，因此CHI的计算公式可以简化为：

$$\chi(t_i,C_j)=\frac{(A_{ij}D_{ij}-C_{ij}B_{ij})^2}{(A_{ij}+B_{ij})(C_{ij}+D_{ij})}$$

CHI特征选择方法，综合考虑文档频率与类别比例两个因素

5）WLLR(Weighted Log Likelihood Ration)

WLLR特征选择方法的定义如下：

$$WLLR(t_i,C_j)=p(t_i|C_j)\log\frac{p(t_i|C_j)}{p(t|\overline{C_j})}$$

计算公式如下：

$$WLLR(t_i,C_j)=\frac{A_{ij}}{A_{ij}+C_{ij}}\log\frac{A_{ij}(B_{ij}+D_{ij})}{B_{ij}(A_{ij}+C_{ij})}$$

6)WFO（Weighted Frequency and Odds）

最后一个介绍的算法，是由苏大李寿山老师提出的算法。通过以上的五种算法的分析，李寿山老师认为，"好"的特征应该有以下特点：

- 好的特征应该有较高的文档频率
- 好的特征应该有较高的文档类别比例

WFO的算法定义如下：

如果$p(t_i|C_j)/p(t_i|\overline{C_j})>1$：

$$WFO(t_i,C_j)=p(t_i|C_j)^{\lambda}\log\frac{p(t_i|C_j)^{1-\lambda}}{p(t|\overline{C_j})}$$

否则：

$$WFO(t_i,C_j)=0$$

不同的语料，一般来说文档词频与文档的类别比例起的作用应该是不一样的，WFO方法可以通过调整参数λ，找出一个较好的特征选择依据。

----------分割线----------

介绍完理论部分，就要给出代码了（只给出公式，不给出代码的都是调戏良家的行为~）。文本挖掘之文本表示一文，利用了sklearn开源工具，自然先首先sklearn工具，可惜的是sklearn文本的特征选择方法仅提供了CHI一种。为此在sklearn框架下，尝试自己编写这些特征选择方法的代码，自己动手，丰衣足食。

笔者实现了三种特征选择方法：IG,MI和WLLR，看官如果对其他特征选择方法感兴趣，可以尝试实现一下~ 好了，啥也不说了，上代码，特征选择模块代码：

```
#!/usr/bin/env python
# coding=gbk

import os
import sys

import numpy as np

def get_term_dict(doc_terms_list):
    term_set_dict = {}
    for doc_terms in doc_terms_list:
        for term in doc_terms:
            term_set_dict[term] = 1
    term_set_list = sorted(term_set_dict.keys())       #term set
    term_set_dict = dict(zip(term_set_list, range(len(term_set_li
    return term_set_dict

def get_class_dict(doc_class_list):
    class_set = sorted(list(set(doc_class_list)))
    class_dict = dict(zip(class_set, range(len(class_set))))
    return  class_dict

def stats_term_df(doc_terms_list, term_dict):
    term_df_dict = {}.fromkeys(term_dict.keys(), 0)
    for term in term_set:
        for doc_terms in doc_terms_list:
            if term in doc_terms_list:
                term_df_dict[term] +=1
    return term_df_dict

def stats_class_df(doc_class_list, class_dict):
    class_df_list = [0] * len(class_dict)
    for doc_class in doc_class_list:
        class_df_list[class_dict[doc_class]] += 1
    return class_df_list

def stats_term_class_df(doc_terms_list, doc_class_list, term_dict
    term_class_df_mat = np.zeros((len(term_dict), len(class_dict)
    for k in range(len(doc_class_list)):
        class_index = class_dict[doc_class_list[k]]
        doc_terms = doc_terms_list[k]
        for term in set(doc_terms):
            term_index = term_dict[term]
            term_class_df_mat[term_index][class_index] +=1
    return  term_class_df_mat

def feature_selection_mi(class_df_list, term_set, term_class_df_m
    A = term_class_df_mat
    B = np.array([(sum(x) - x).tolist() for x in A])
    C = np.tile(class_df_list, (A.shape[0], 1)) - A
```

```
    N = sum(class_df_list)
    class_set_size = len(class_df_list)

    term_score_mat = np.log(((A+1.0)*N) / ((A+C) * (A+B+class_set
    term_score_max_list = [max(x) for x in term_score_mat]
    term_score_array = np.array(term_score_max_list)
    sorted_term_score_index = term_score_array.argsort()[: : -1]
    term_set_fs = [term_set[index] for index in sorted_term_score

    return term_set_fs

def feature_selection_ig(class_df_list, term_set, term_class_df_m
    A = term_class_df_mat
    B = np.array([(sum(x) - x).tolist() for x in A])
    C = np.tile(class_df_list, (A.shape[0], 1)) - A
    N = sum(class_df_list)
    D = N - A - B - C
    term_df_array = np.sum(A, axis = 1)
    class_set_size = len(class_df_list)

    p_t = term_df_array / N
    p_not_t = 1 - p_t
    p_c_t_mat =  (A + 1) / (A + B + class_set_size)
    p_c_not_t_mat = (C+1) / (C + D + class_set_size)
    p_c_t = np.sum(p_c_t_mat  *  np.log(p_c_t_mat), axis =1)
    p_c_not_t = np.sum(p_c_not_t_mat *  np.log(p_c_not_t_mat), ax

    term_score_array = p_t * p_c_t + p_not_t * p_c_not_t
    sorted_term_score_index = term_score_array.argsort()[: : -1]
    term_set_fs = [term_set[index] for index in sorted_term_score

    return term_set_fs

def feature_selection_wllr(class_df_list, term_set, term_class_df
    A = term_class_df_mat
    B = np.array([(sum(x) - x).tolist() for x in A])
    C_Total = np.tile(class_df_list, (A.shape[0], 1))
    N = sum(class_df_list)
    C_Total_Not = N - C_Total
    term_set_size = len(term_set)

    p_t_c = (A + 1E-6) / (C_Total + 1E-6 * term_set_size)
    p_t_not_c = (B +  1E-6) / (C_Total_Not + 1E-6 * term_set_size
    term_score_mat = p_t_c  * np.log(p_t_c / p_t_not_c)

    term_score_max_list = [max(x) for x in term_score_mat]
    term_score_array = np.array(term_score_max_list)
    sorted_term_score_index = term_score_array.argsort()[: : -1]
    term_set_fs = [term_set[index] for index in sorted_term_score

    print term_set_fs[:10]
    return term_set_fs

def feature_selection(doc_terms_list, doc_class_list, fs_method):
    class_dict = get_class_dict(doc_class_list)
    term_dict = get_term_dict(doc_terms_list)
    class_df_list = stats_class_df(doc_class_list, class_dict)
    term_class_df_mat = stats_term_class_df(doc_terms_list, doc_c
    term_set = [term[0] for term in sorted(term_dict.items(), key
    term_set_fs = []

    if fs_method == 'MI':
        term_set_fs = feature_selection_mi(class_df_list, term_se
    elif fs_method == 'IG':
        term_set_fs = feature_selection_ig(class_df_list, term_se
    elif fs_method == 'WLLR':
        term_set_fs = feature_selection_wllr(class_df_list, term_
```

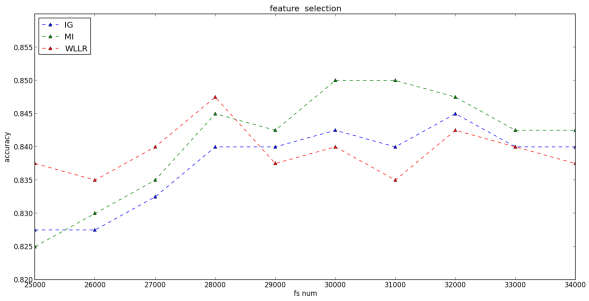
```
    return term_set_fs
```

在movie语料里面比较着三种特征选择方法，调用方法如下：

```
#!/usr/bin/env python
# coding=gbk

import os
import sys

import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_files
from sklearn.cross_validation import train_test_split
from sklearn.feature_extraction.text import  CountVectorizer
from sklearn.naive_bayes import MultinomialNB

import feature_selection

def text_classifly_twang(dataset_dir_name, fs_method, fs_num):
    print 'Loading dataset, 80% for training, 20% for testing...'
    movie_reviews = load_files(dataset_dir_name)
    doc_str_list_train, doc_str_list_test, doc_class_list_train,

    print 'Feature selection...'
    print 'fs method:' + fs_method, 'fs num:' + str(fs_num)
    vectorizer = CountVectorizer(binary = True)
    word_tokenizer = vectorizer.build_tokenizer()
    doc_terms_list_train = [word_tokenizer(doc_str) for doc_str i
    term_set_fs = feature_selection.feature_selection(doc_terms_l

    print 'Building VSM model...'
    term_dict = dict(zip(term_set_fs, range(len(term_set_fs))))
    vectorizer.fixed_vocabulary = True
    vectorizer.vocabulary_ = term_dict
    doc_train_vec = vectorizer.fit_transform(doc_str_list_train)
    doc_test_vec= vectorizer.transform(doc_str_list_test)

    clf = MultinomialNB().fit(doc_train_vec, doc_class_list_train
    doc_test_predicted = clf.predict(doc_test_vec)

    acc = np.mean(doc_test_predicted == doc_class_list_test)
    print 'Accuracy: ', acc

    return acc


if __name__ == '__main__':
    dataset_dir_name = sys.argv[1]
    fs_method_list = ['IG', 'MI', 'WLLR']
    fs_num_list = range(25000, 35000, 1000)
    acc_dict = {}

    for fs_method in fs_method_list:
        acc_list = []
        for fs_num in fs_num_list:
            acc = text_classifly_twang(dataset_dir_name, fs_metho
            acc_list.append(acc)
        acc_dict[fs_method] = acc_list
        print 'fs method:', acc_dict[fs_method]

    for fs_method in fs_method_list:
```

```
        plt.plot(fs_num_list, acc_dict[fs_method],  '--^',  label
        plt.title('feature  selection')
        plt.xlabel('fs num')
        plt.ylabel('accuracy')
        plt.ylim((0.82, 0.86))

    plt.legend( loc='upper left', numpoints = 1)
    plt.show()
```

输出的结果：



从上面的图看出：分类的性能随着特征选择的数量的增加，呈现“凸”形趋势：1）在特征数量较少的情况下，不断增加特征的数量，有利于提高分类器的性能，呈现“上升”趋势；2）随着特征数量的不断增加，将会引入一些不重要的特征，甚至是噪声，因此，分类器的性能将会呈现“下降”的趋势。这张“凸”形趋势体现出了特征选择的重要性：选择出重要的特征，并降低噪声，提高算法的泛化能力。

参数文献：

1.Y. Yang and J. Pedersen. 1997. A comparative study on feature selection in text categorization.

2.Shoushan Li, Rui Xia, Chengqing Zong and Chu-Ren Huang.2009.A Framework of Feature Selection Methods for Text Categorization

3.老板的课件

« 上一篇： C++模板专门化与重载
» 下一篇： 数组中子数组之和最大问题

分类 机器学习，文本挖掘与情感分析