Tmsvm

Text Mining System Based on SVM

版本	1.1.0
作者	张知临
联系方式	zhzhl202@163.com
最后更新	2012/03/08
系统主页	http://code.google.com/p/tmsvm/

目录

第 1 章 简介：.....4
1.1 主要特征.....4
1.2 利用此系统可以做什么.....4
1.3 本系统欲解决的问题：.....5
1.4 程序文件说明.....5
1.5 调用方法：.....6
1.6 模型文件说明.....6
第 2 章 程序调用接口.....8
2.1 使用前必看.....8
2.2 输入格式及程序输出：.....8
2.3 在程序中直接使用.....8
2.3.1 训练 SVM 模型.....8
2.3.2 模型预测.....9
2.3.3 多模型预测.....10
2.3.4 结果分析.....11
2.3.5 分词.....12
2.3.6 特征选择.....12
2.3.7 构造 libsvm 与 liblinear 的输入格式.....13
2.3.8 SVM 参数搜索.....13
2.3.9 SVM 模型训练.....14
2.4 在命令行中直接使用.....14
2.4.1 自动文本 SVM 分类模型训练 auto_train.py.....14
2.4.2 文本 SVM 分类模型训练 train.py.....17
2.4.3 模型预测程序.....19
2.4.4 LSA 模型训练程序.....20
2.4.5 LSA 模型预测程序.....21
2.5 一些有用的工具.....22
2.5.1 结果分析程序 result_anlaysis.py.....22
2.5.2 选择子集 subset.py(libsvm).....24
2.5.3 SVM 参数选择 grid.py (libsvm).....25
2.6 系统使用示例.....26
2.6.1 一个完整的示例.....26
2.6.2 更多的案例.....27
第 3 章 技术细节.....28

3.1 Latent Semantic Analysis 28
3.2 Feature Selection.....28
3.3 SVM 参数选择.....29
3.4 Libsvm 与 liblinear 的多分类策略.....29
3.5 特征权重.....29
3.6 模型返回结果.....30
3.7 模型训练和测试流程.....30
3.8 重复样本对 SVM 模型的影响.....31
将分类应用与信息过滤.....33
第 4 章 源码剖析.....35
4.1 将单个文本构造成 SVM 的输入格式.....35
4.2 将所有文本构造成 SVM 的输入格式.....36
4.3 Libsvm 与 Liblinear 差异.....37
4.4 Result_analysis 37
4.4.1 递归保存结果 dict.....37
4.4.2 对特定类别计算指定阈值下的各种指标.....38
第 5 章 疑问点.....40
第 6 章 FAQ.....41
关于我.....42
Thanks.....43

1 简介：

文本挖掘无论在学术界还是在工业界都有很广泛的应用场景。而文本分类是文本挖掘中一个非常重要的手段与技术。现有的分类技术都已经非常成熟，SVM、KNN、Decision Tree、AN、NB 在不同的应用中都展示出较好的效果，前人也在将这些分类算法应用于文本分类中做出许多出色的工作。但在实际的商业应用中，仍然有很多问题没有很好的解决，比如文本分类中的高维性和稀疏性、类别的不平衡、小样本的训练、Unlabeled 样本的有效利用、如何选择最佳的训练样本等。这些问题都将导致 curve of dimension、过拟合等问题。

这个开源系统的目的是集众人智慧，将文本挖掘、文本分类前沿领域效果非常好的算法实现并有效组织，形成一条完整系统将文本挖掘尤其是文本分类的过程自动化。该系统提供了 Python 和 Java 两种版本。

1.1 主要特征

该系统在封装 libsvm、liblinear 的基础上，增加了特征选择、LSA 特征抽取、SVM 模型参数选择、libsvm 格式转化模块以及一些实用的工具。其主要特征如下：

1) 封装并完全兼容 libsvm、liblinear。
2) 基于 Chi 的 feature selection
3) 基于 Latent Semantic Analysis 的 feature extraction
4) 支持 Binary,Tf,log(tf),Tf*Idf,tf*rf,tf*chi 等多种特征权重
5) 文本特征向量的归一化
6) 利用交叉验证对 SVM 模型参数自动选择。
7) 支持 macro-average、micro-average、F-measure、Recall、Precision、Accuracy 等多种评价指标
8) 支持多个 SVM 模型同时进行模型预测
9) 采用 python 的 csc_matrix 支持存储大稀疏矩阵。
10) 引入第三方分词工具自动进行分词
11) 将文本直接转化为 libsvm、liblinear 所支持的格式。

1.2 利用此系统可以做什么

1、 对文本自动做 SVM 模型的训练。包括 Libsvm、Liblinear 包的选择，分词，词典生成，特征选择，SVM 参数的选优，SVM 模型的训练等都可以一步完成。

2、 利用生成的模型对未知文本做预测。并返回预测的标签以及该类的隶属度分数。可自动识别 libsvm 和 liblinear 的模型。

3、 自动分析预测结果，评判模型效果。计算预测结果的 F 值、召回率、准确率、Macro,Micro 等指标，并会计算特定阈值、以及指定区间所有阈值下的相应指标。

4、 分词。对文本利用 mmseg 算法对文本进行分词。

5、 特征选择。对文本进行特征选择，选择最具代表性的词。

6、 SVM 参数的选择。利用交叉验证方法对 SVM 模型的参数进行识别，可以指定搜索范围，大于大数据，会自动选择子集做粗粒度的搜索，然后再用全量数据做细粒度的搜索，直到找到最优的参数。对 libsvm 会选择 c,g(gamma)，对与 liblinear 会选择 c。

7、 对文本直接生成 libsvm、liblinear 的输入格式。libsvm、liblinear 以及其他诸如 weka 等数据挖掘软件都要求数据是具有向量格式，使用该系统可以生成这种格式：label index:value

8、 SVM 模型训练。利用 libsvm、liblinear 对模型进行训练。

9、 利用 LSA 对进行 Feature Extraction，从而提高分类效果。

1.3 本系统欲解决的问题：

1、文本分类的高维性和稀疏性

2、Unbalance 样本的问题。实际应用中，各个类别的数据集大小往往是不平衡的，尤其在 information filtering 领域。感兴趣的数据集相对于不感兴趣的数据集是非常小

3、Unlabeled 样本的利用问题。

4、样本的 Randomly sampling 问题。

5、……

1.4 程序文件说明

src:即该系统的源代码，提供了 5 个可以在 Linux 下可以直接调用的程序:

aauto_train.py、train.py、predict.py 为在 Linux 下通过命令行调用的接口。

tms.py 为在程序中调用的主文件，直接通过 import tms 即可调用系统的所有函数。

其他文件为程序中实现各个功能的文件。

lsa_src：LSA 模型的源程序。

dependence:系统所依赖的一些包。包括 libsvm、liblinear、Pymmseg 在 Linux32 位和 64 位以及 windows 下的支持包(dll,so 文件)。

tools:提供的一些有用的工具，包括 result_analysis.py 等。

java：java 版本的模型预测程序，

1.5 调用方法：

该系统可以在命令行（Linux 或 cmd 中）中直接使用，也可以在程序通过直接调用源程序使用。

在程序中使用。

```
import tms
#对 data 文件夹下的 binary_seged.train 文件进行训练。
tms.tms_train("../data/binary_seged.train")
#利用已经训练好的模型，对对 data 文件夹下的 binary_seged.test 文件预测
tms.tms_predict("../data/binary_seged.test","../model/tms.config")
#对预测的结果进行分析，评判模型的效果
tms. tms_analysis("../tms.result")
```

在命令行中调用

```
#对 data 文件夹下的 binary_seged.train 文件进行训练。
$python auto_train.py [options]    ../data/binary_seged.train
#利用已经训练好的模型，对对 data 文件夹下的 binary_seged.test 文件预测
$python predict.py ../data/binary_seged.train ../model/tms.config
#对预测的结果进行分析，评判模型的效果
$python result_anlaysis.py ../tms.result
```

上面的调用形式都是使用系统中默认的参数，更具体、灵活的参数见程序调用接口

1.6 模型文件说明

模型文件总共有 3 个，默认参数下为 tms.config、tms.model、dic.key。

其中 tms.config 中存储了模型训练的参数。

```
SvmType:liblinear
SvmParam:-c 0.176776695297
DicName:title.key
ModelName:title.model
LocalFun:tf
GlobalFun:one
WordSeg:0
Date:2011-11-22-01-25-10
Labels:
{
1.0,违规类
-1.0,正常类
}
```

SvmType:指代选择的是 libsvm 还是 liblinear

SvmParam:指代训练模型的参数。

DicName:词典存储的名称

ModelName:模型存储的名称.

LocalFun:特征权重中局部函数

GlobalFun:特征权重中全局因子

WordSeg:训练样本使用的分词工具

Labels:训练样本的各个类标签以及解释

dic.key 是模型的词典。

```
收藏  1 1
店铺  2 1
盖楼  3 1
活动  4 1
期  5 1
报名  6 1
人气  7 1
单品  8 1
相互  9 1
旺  10  1
链接  11  1
请  12  1
元  13  1
增加  14  1
```

词典包括 3 列：第一列为 term，第二列为 id，第三列位 global weight，因为在计算特征向量每个 term 的权重时，全局因子部分至于词有关，所以可以直接根据训练样本计算出来放置在词典中，方便使用。

tms.model 就是最终的 SVM 模型，或者是 libsvm 的模型或者是 liblinear 的模型。

2 程序调用接口

2.1 使用前必看

1. 将源代码下载下来就可以直接使用几乎所有的操作。
2. 如果想进一步使用 LSA 模型，需要安装 scipy 与 numpy。

2.2 输入格式及程序输出：

一般的输入格式为：

label value1 [value2]

其中 label 是定义的类标签，如果是 binary classification，建议 positive 样本为 1，negative 样本为-1。如果为 multi-classification。label 可以是任意的整数。

其中 value 为文本内容，可以分好词也可以没有分词。

label 和 value 以及 value1 和 value2 之间需要用特殊字符进行分割，如”\t”。

程序输出：

模型结果会放在指定保存路径下的“model”文件夹中，里面有 3 个文件，默认情况下为 dic.key 、 tms.model 和 tms.config 。其中 dic.key 为特征选择后的词典；tms.model 为训练好的 SVM 分类模型, tms.config 为模型的配置文件，里面记录了模型训练时使用的参数。临时文件会放在“temp”文件夹中。里面有两个文件：tms.param 和 tms.train。其中 tms.param 为 SVM 模型参数选择时所实验的参数。tms.train 是供 libsvm 和 liblinear 训练器所使用的输入格式。

2.3 在程序中直接使用

tms.py 是模型的主程序，通过调用 tms 可以进行模型训练、模型预测、结果分析。还可以进行分词、特征选择、SVM 参数选择、SVM 模型训练等等。

注意如果要再 IDLE 中直接使用，需要把该系统路径放入到 Python 搜索路径中去。

2.3.1 训练SVM模型

1. 程序介绍

训练的自动化程序，分词，先进行特征选择，重新定义词典，根据新的词典，自动选择SVM最优的参数。 然后使用最优的参数进行SVM分类，最后生成训练后的模型。

2. 调用方法

```
import  tms
```

tms.tms_train(filename,options)

3. 结果文件

模型文件（词典.key+模型.model+模型配置.config）和 临时文件（svm 分类数据文件.train 和参数选择文件.param）

4. 参数选择

必须参数：

1) *filename 训练文本所在的文件名*

可选参数：

1) *indexes需要训练的指标项 ， 默认为[1]*
2) *main_save_path 模型保存的路径.默认为"../"*
3) *stopword_filename 停用词的名称以及路径 ;默认不适用停用词*
4) *svm_type :svm类型：libsvm 或liblinear 。默认为"libsvm"*
5) *svm_param 用户自己设定的svm的参数,这个要区分libsvm与liblinear参数的限制。默认" "*
6) *config_name:模型配置文件的名称，默认为"tms.config"*
7) *dic_name 用户自定义词典名称;默认“dic.key”*
8) *model_name用户自定义模型名称 ;默认"svm.model"*
9) *train_name用户自定义训练样本名称 ； 默认“svm.train”*
10) *param_name用户自定义参数文件名称 ； 默认"svm.param"*
11) *ratio 特征选择保留词的比例 ； 默认 0.4*
12) *delete对于所有特征值为0的样本是否删除,True or False，默认：True*
13) *str_splitTag 分词所用的分割符号 ， 默认"^"*
14) *tc_splitTag训练样本中各个字段分割所用的符号 ， 默认"\t"*
15) *seg 分词的选择：0为不进行分词；1为使用mmseg分词；2为使用aliws分词，默认为0*
16) *param_select ;是否进行SVM模型参数的搜索。True即为使用SVM模型grid.搜索，False即为不使用参数搜索。默认为True*
17) *local_fun：即对特征向量计算特征权重时需要设定的计算方式:x(i,j) = local(i,j)*global(i).可选的有tf,binary,logtf。默认为"tf"*
18) *global_fun :全局权重的计算方式：有"one","idf","rf","chi" ,默认为"one"*

2.3.2 模型预测

1. 程序介绍

模型预测程序.输入需要预测的文件，以及模型的配置文件，既可利用已经训练好的模型对文件进行预测。

2. 调用方法

import tms

tms. tms_predict(filename,config_file,options)

3. 结果文件

模型预测的结果有两个。Label 和隶属度分数

4. 参数选择

必须参数：

filename：带预测文件的路径以及名称

config_file:已经训练好的模型的配置文件

可选参数：

result_save_path：预测结果保存路径及名称。默认为"../tms.result"

indexes:预测文件中需要预测的字段。默认为[1]

result_indexes:需要和预测结果一起输出的源文件中的字段。默认为[0]

str_splitTag 分词所用的分割符号 ，默认"^"

seg:是否进行分词。seg=0表示不对源文件进行分词，seg=1代表使用pymmseg进行分词。seg=2代表使用aliws进行分词

delete：代表是否要把所有特征都为0样本删除。默认在预测时候不删除。

change_decode：是否要进行编码转换，预测的样本要和训练的样本编码保持一致。默认不转换

in_decode：如果要进行编码转换，原先的编码符号。默认gbk.

out_encode:需要转换的编码，默认为utf-8

2.3.3 多模型预测

1. 程序介绍

多模型预测程序.输入需要预测的文件，以及多模型的配置文件和多模型需要预测文本的字段，既可利用已经训练好的模型对文件进行预测。

2. 调用方法

import tms

tms. tms_predict_multi(filename,config_file, indexes_lists,options)

3. 结果文件

模型预测的结果为所有模型的 Label 和隶属度分数

4. 参数选择

必须参数：

filename：带预测文件的路径以及名称

config_files:已经训练好的模型的配置文件

indexes_lists：每个模型需要预测的字段。

可选参数：

result_save_path：预测结果保存路径及名称。默认为"../tms.result"

result_indexes:需要和预测结果一起输出的源文件中的字段。默认为[0]

str_splitTag 分词所用的分割符号 ，默认"^"

seg:是否进行分词。seg=0表示不对源文件进行分词，seg=1代表使用pymmseg进行分词。seg=2代表使用aliws进行分词

delete：代表是否要把所有特征都为0样本删除。默认在预测时候不删除。

change_decode：是否要进行编码转换，预测的样本要和训练的样本编码保持一致。默认不转换

in_decode：如果要进行编码转换，原先的编码符号。默认gbk.

out_encode:需要转换的编码，默认为utf-8

2.3.4 结果分析

1. 程序介绍

模型结果分析程序。输入已经预测好的文件对分类准确率、Macro、Micro、F 值、Recall、Precision 进行计算，并会计算特定阈值下的各指标。 以及计算阈值区间内所有阈值的各指标，可以进行合适阈值的选择。

2. 调用方法

import tms

tms. tms_analysis(filename,options)

其中 filename 要包括 3 个字段：预测的 label、预测的分数、实际的 label

3. 结果文件

结果为计算出来的各种指标(分类准确率、Macro、Micro、F 值、Recall、Precision 等)

4. 参数选择

必要参数：

filename：输入的文件。通常文件包括3个字段：预测label,预测分数,真实label

可选参数：

output_file ：分析结果输出文件。默认为""，即输出在屏幕上。

indexes：即从文件中读入的数据，考虑到最终预测结果可能包含多个字段，所以可以指定具体需要读入的字段。默认为前3列。

step：选择进行分析的步骤：

1为多分类以及二分类的微观分类准确率，宏观分类准确率，所有类的分类准确率。

2为多分类以及二分类中各个类别的F值、召回率、准确率。

3为计算多分类以及二分类中对指定的类别，对特定阈值下的F值、召回率、准确率。

4为多分类以及二分类中计算所有类别的在阈值区间中的每个阈值每个类别的F值、召回率、准确率，旨在为用户分析出每个类别最好的阈值。

predicted_label_index：文件中预测标签的字段号，默认为0.

predicted_value_index：文件中预测分数的字段号，默认为1.

true_label_index:文件中真实标签的字段号，默认为2.

threshold=0.0：如果step=3,可以设定特定的阈值。默认为0.0

label=1:如果step=3:可以设定对特定类别特定阈值进行计算F值、Recall、Precision。

默认为类标签为1

min：如果step=4.设定阈值区间的最小值。默认情况下搜索区间为[0,0.1,...0.9]，所以该值默认为0

max:如果step=4.设定阈值区间的最大值，所以该值默认为1.

2.3.5 分词

1. 程序介绍

分词的主程序

2. 调用方法

```
import    tms
tms. tms_segment(filename,options)
```

3. 结果文件

结果为已经分好词的文件。

4. 参数选择

必须参数:

filename 训练文本所在的文件名，默认情况下，已经分好词。

可选参数:

indexs需要训练的指标项 ，默认为[1]

out_filename:分词后的结果保存的文件,如果为""，则保存在和输入文件同目录下的"segmented".默认情况下""

str_splitTag 分词所用的分割符号 ，默认"^"

tc_splitTag训练样本中各个字段分割所用的符号 ，默认"\t"

seg 分词的选择：0为不进行分词；1为使用mmseg分词；2为使用aliws分词，默认为0

2.3.6 特征选择

1. 程序介绍

特征选择的主程序，输入指定的文件，会自动生成词典，并根据卡方公式进行特征选择。

2. 调用方法

```
import    tms
tms. tms_feature_select(filename,options)
```

3. 结果文件

选择最具代表性的词典，默认为 dic.key。词典有 3 列：term、id、global_weight

4. 参数选择

必须参数:

filename 训练文本所在的文件名，默认情况下，已经分好词。

可选参数：

indexs需要训练的指标项 ， 默认为[1]

main_save_path 模型保存的路径.默认为"../"

stopword_filename 停用词的名称以及路径 ;默认不适用停用词

dic_name 用户自定义词典名称;默认“dic.key”

ratio 特征选择保留词的比例 ； 默认 0.4

str_splitTag 分词所用的分割符号 ， 默认"^"

tc_splitTag训练样本中各个字段分割所用的符号 ， 默认"\t"

global_fun ：全局权重的计算方式：有"one","idf","rf" ,默认为"one"

2.3.7 构造libsvm与liblinear的输入格式

1. 程序介绍

将已经分好词的文件转换为libsvm和liblinear的输入格式。

2. 调用方法

import tms

tms. cons_train_sample_for_svm(filename,dic_path,options)

3. 结果文件

适合 libsvm 与 liblinear 的格式输入文件 tms.train

4. 参数选择

必须参数：

filename 训练文本所在的文件名，默认情况下，已经分好词。

dic_path 词典所在的目录，将文本转换为向量。

可选参数：

sample_save_path 。转换好的文件保存的位置。默认情况下为"../svm.train"

indexs需要训练的指标项 ， 默认为[1]

*local_fun：即对特征向量计算特征权重时需要设定的计算方式:x(i,j) = local(i,j)*global(i).可选的有tf。默认为"tf"*

delete对于所有特征值为0的样本是否删除,True or False，默认：True

str_splitTag 分词所用的分割符号 ， 默认"^"

tc_splitTag训练样本中各个字段分割所用的符号 ， 默认"\t"

2.3.8 SVM参数搜索

1. 程序介绍

对SVM的参数进行搜索,如果是libsvm则搜索(c,gamma)，如果是liblinear则搜索(c)。当训练样本的容量大于3000时就会在粗粒度搜索时使用子集，子集的大小为[3000,5000]范围内。

2. 调用方法

import tms

tms. **tms_grid_pa**ram (problem_path, options)

3. 结果文件

返回最优的 c,g。对给定范围内的每对 c,g 做 5-folds 的交叉验证，选择效果最好的 c,g。对与 libsvm,选择最佳的 c,g。对与 Liblinear,返回最优的 c。

4. 参数选择

必须参数:

problem_path：SVM输入格式文件的路径即名称。

可选参数:

result_save_path:结果文件的保存路径:默认为"../svm.param"

svm_type :选择的SVM的类型，默认为libsvm

coarse_c_range ：粗粒度搜索时c搜索的范围，默认情况下为[-5,7],步长为2

coarse_g_range：粗粒度搜索时g搜索的范围，默认情况下为[3,-10]步长为-2

fine_c_step ：细粒度搜索时c的步长，默认情况下为0.5

fine_g_step ：细粒度搜索时c 的步长，默认情况下为0.5

2.3.9 SVM模型训练

1. 程序介绍

训练模型程序。输入参数，可以训练libsvm与liblinear的模型。

2. 调用方法

import tms

tms. tms_train_model (problem_path, options)

3. 结果文件

返回已经训练好的 SVM 模型。

4. 参数选择

必须参数:

problem_path :输入问题的路径即名称:

可选参数 :

svm_type :svm类型: libsvm 或liblinear 。默认为"libsvm"

param 用户自己设定的svm的参数,这个要区分libsvm与liblinear参数的限制。默认" " ，即软件中自带的参数。

model_save_path :模型保存的路径,默认情况下路径为"../svm.model"

2.4 在命令行中直接使用

2.4.1 自动文本SVM分类模型训练auto_train.py

1. 说明

此函数为文本 SVM 分类模型**自动**训练程序，给定训练文本及设置相应参数，即可得到训练好的模型。

2. **调用示例：**

usage:%prog [options] filename

$python auto_train.py [options] filename

3. **输入格式**

label value1 [value2]

其中 label 是定义的类标签，如果是 binary classification，建议 positive 样本为 1，negative 样本为-1。如果为 multi-classification。label 可以是任意的整数。

其中 value 为文本内容，可以有多个字段，如标题、内容。

label value 之间需要用特殊字符进行分割，如”\t”。

4. **结果**

模型结果会放在指定保存路径下的“model”文件夹中，里面有 3 个文件，默认情况下为 dic.key 、 tms.model 和 tms.config 。其中 dic.key 为特征选择后的词典；tms.model 为训练好的 SVM 分类模型, tms.config 为模型的配置文件，里面记录了模型训练时使用的参数。临时文件会放在“temp”文件夹中。里面有两个文件：tms.param 和 tms.train。其中 tms.param 为 SVM 模型参数选择时所实验的参数。tms.train 是供 libsvm 和 liblinear 训练器所使用的输入格式。

5. **参数说明：**

1) -p，--path，模型保存的路径。默认为 ”../”

2) -i，--indexes,输入文本中训练的模型的部分（从 0 开始编号），默认为[1]，输入时可用 –i 1,2,3 表示使用第 1,2,3 作为训练的内容

3) -w，(布尔型)如果使用此参数代表词典中不去除停用词。如果使用，必须将停用词文件以 stopwords.txt 命名，和训练文本放在同一路径下。默认情况下不使用此参数，即需将停用词文件 stopwords.txt 放在训练文本同一路径下

4) -A ,--tms_param。即 SVM 训练的参数，完全兼容 libtms，输入的格式为 –A “-s 0 –c 1.0 –g 0.25” 。为了避免和现在的参数相混淆，所以要加上双引号。

5) -n,--config_name. 指定模型配置文件的名称，默认为”tms.config”

6) -d，--dic_name。指定特征选择后词典的名称,默认为dic.key

7) -m, --model_name,指定生成的分类模型的名称，默认为tms.model

8) -t, --train_name,指定生成的分类模型的名称，默认为tms.train

9) -a, --param_name,指定生成的分类模型的名称，默认为tms.param

10) -r，--ratio。指定特征选择保留词的比例。默认为0.4

11) -T,--tc_splitTag。训练文本中各部分分割的符号,默认为”\t”

12) -S,--str_splitTag.训练文本中分词的分割词，默认为”^”

13) -v ，--svm_type。SVM 模型的类型，两个选项：”libsvm”和”liblinear”，默认情况下为libsvm.

14) -e --segment((布尔型))是否对文本进行分词，默认情况下不分词，如果输入-e，则表明对其进行分词。

15) -c --param_select。(布尔型)是否进行参数选择,默认为选择，如果输入-c ，则表明不需要进行选择参数。

16) -g --global_fun。特征权重中全局因子的计算方式，三个选择“idf”、”rf”、”one”。one 是指对所有的 term 全局因子都为 1.默认为”one”

17) -l --local_fun。特征权重中全局因子的计算方式，1 个选择“tf”。.默认为”tf”

18) -b ,--label_file。对与每个类别标签的说明文件，文件要以“label,descr”。默认情况下位””

2.4.2 文本SVM分类模型训练 train.py

1. 说明

   此函数为文本 SVM 分类模型训练程序，与 auto_train.py 不同的是，该函数可以使模型训练分步进行。

2. 调用示例：

   usage:%prog [options] filename

   filename 在1),2),3)中代表输入训练文本，在4),5)中代表SVM 的输入格式。

   1) 自动进行模型训练

   $python auto_train.py [options] -s 1 filename

   2) 特征选择

   $python auto_train.py [options] -s 2 filename

   3) 生成 SVM 模型的输入格式

   $python auto_train.py [options] -s 3 filename

   4) SVM 模型的参数选择

   $python auto_train.py [options] -s 4 filename

   5) 模型训练

   $python auto_train.py [options] -s 5 filename

3. 输入格式

   在 **1),2),3)**步骤：

   label value1 [value2]

   其中 label 是定义的类标签，如果是 binary classification，建议 positive 样本为 1，negative 样本为-1。如果为 multi-classification。label 可以是任意的整数。

   其中 value 为文本内容，可以有多个字段，如标题、内容。

   label value 之间需要用特殊字符进行分割，如”\t”。

   在 4),5)步骤：

   输入格式为 libsvm、liblinear 特有的格式。

   <label> <index1>:<value1> <index2>:<value2>

4. 结果

模型结果会放在指定保存路径下的“model”文件夹中，里面有 3 个文件，默认情况下为 dic.key 、 tms.model 和 tms.config 。其中 dic.key 为特征选择后的词典；tms.model 为训练好的 SVM 分类模型, tms.config 为模型的配置文件，里面记录了模型训练时使用的参数。临时文件会放在“temp”文件夹中。里面有两个文件：tms.param 和 tms.train。其中 tms.param 为 SVM 模型参数选择时所实验的参数。tms.train 是供 libsvm 和 liblinear 训练器所使用的输入格式。

5. 参数说明：

1) -s ,--step,即选择要进行的操作。1 为自动训练模型，即 auto_train.py 的功能。2 为特征选择。3 为根据训练样本生成 SVM 的输入格式。4 为 SVM 模型参数选择；5 为 SVM 训练

2) -p，--path，模型保存的路径，默认情况下为”../”

3) -i，--indexes,输入文本中训练的模型的部分（从 0 开始编号），默认为[1]，输入时可用 –i 1,2,3 表示使用第 1,2,3 作为训练的内容

4) -w，(布尔型)如果使用此参数代表词典中不去除停用词。如果使用，必须将停用词文件以 stopwords.txt 命名，和训练文本放在同一路径下。默认情况下不使用此参数，即需将停用词文件 stopwords.txt 放在训练文本同一路径下

5) -A ,--tms_param。即 SVM 训练的参数，完全兼容 libtms，输入的格式为 –A “-s 0 –c 1.0 –g 0.25” 。为了避免和现在的参数相混淆，所以要加上双引号。默认为“-s 0 –c 1.0 –g 0.25”.

6) –n,--config_name. 指定模型配置文件的名称，默认为”tms.config”

7) -d，--dic_name。指定特征选择后词典的名称,默认为 dic.key

8) -D,--dic_path。词典所在的路径及名称

9) -m, --model_name,指定生成的分类模型的名称，默认为tms.model

10) -t, --train_name,指定生成的分类模型的名称，默认为tms.train

11) -a, --param_name,指定生成的分类模型的名称，默认为tms.param

12) -r，--ratio。指定特征选择保留词的比例。默认为0.4

13) -T，--tc_splitTag。训练文本中各部分分割的符号，默认为”\t”

14) -S,--str_splitTag.训练文本中分词的分割词，默认为”^”

15) -v ，--svm_type。SVM 模型的类型，两个选项：”libsvm”和”liblinear”，默认情况下为libsvm.

16) -e --segment((布尔型))是否对文本进行分词，默认情况下不分词，如果输入-e，则表明对其进行分词。

17) -c --param_select。(布尔型)是否进行参数选择，默认为选择，如果输入-c ，则表明不需要进行选择参数。

18) -g --global_fun。特征权重中全局因子的计算方式，三个选择“idf”、”rf”、”one”。one 是指对所有的 term 全局因子都为1.默认为”one”

19) -l --local_fun。特征权重中全局因子的计算方式，1 个选择“tf”。.默认为”tf”

2.4.3 模型预测程序

1. 说明

此函数为文本 SVM 分类模型**预测**程序，给定测试文本及设置相应参数，即可为样本进行预测。

2. **调用示例：**

usage:%prog [options] filename config_file

3. **输入格式**

没有特定的输入格式。具体需要预测的内容可以通过 –i 指定。

4. **结果**

预测的结果会写入到用户指定的文件中。其中第一列为预测的标签，第二列为预测的分数（属于该类的隶属度），其余列为指定的需要同结果一同输出的内容。

5. **参数说明：**

-i，--indexes,输入文本中训练的模型的部分（从 0 开始编号），默认为1，输入时可用 –i 1,2,3 表示使用第1,2,3 作为训练的内容

-r，----result_indexes。指定与预测分数一块输出的文本的指标项，其中预测分数放在第一列，其余的依次排列。默认为1，调用方式为 –r 1,2,3

-e --segment((布尔型))是否对文本进行分词，默认情况下不分词，如果输入-e，则表明对其进行分词。

-R,--result_save 。结果保存的路径及文件名称。

-T，--tc_splitTag。训练文本中各部分分割的符号，默认为"\t"

-S,--str_splitTag.训练文本中分词的分割词，默认为"^"

2.4.4 LSA模型训练程序

1. **说明**

此函数为 LSA 模型**训练**程序，给定测试文本及训练好的 SVM 模型和词典的长度，即可为 LSA 模型进行训练。

LSA 模型虽然在实际实验中没有得到预想的效果，但是在某些场景下应该会有用，所以 LSA 模型训练与预测的程序仍然保留。

2. **调用示例：**

usage:%prog [options] filename svm_model M

完整形式：

python lsa_train.py -f ../ sample.test -R ../ result/score.result -i 1,2

-D ../model/dic.key -M ../model/tms.model -r 0,1,2

其含义为对 sample.test 中的第 1,2 列(列从 0 开始，1,2 要融合在一起)进行预测，结果放在 score.result 文件中，其中第一列为分数，其余列为指定的第 0，1,2 列。指定词典以及训练好的模型。

3. 输入格式

4. 结果

参数说明：

-p，--path，模型保存的路径，默认情况下为"../"

-e, --threshold 。LSA 模型选取 top n 阈值。默认情况下为 1.0

-K,--K 。选取的前 k 个特征根。

-f,--for_lsa_train 。SVM 模型预测训练文本，并构造适合 LSA 模型的训练文本。默认为"for_lsa.train"

-t,--train_name；LSA 模型做出的 SVM 训练文本格式,默认为"lsa.train"

-m,--model_name；LSA 模型的名称.默认为"lsa.model"

-A,--tms_param；即 SVM 训练的参数，完全兼容 libtms，输入的格式为 –A "-s 0 –c 1.0 –g 0.25" 。为了避免和现在的参数相混淆，所以要加上双引号。默认为"-s 0 –c 1.0 –g 0.25"

-a, --param_name,指定生成的分类模型的名称，默认为 tms.param

2.4.5 LSA模型预测程序

1. 说明

此函数为文本 LSA 模型**预测**程序，给定测试文本及设置相应参数，即可为样本进行预测。

2. 调用示例：

usage:%prog [options] filename dic_path model_path sa_path lsa_model_path

完整形式：

Python lsa_predict_py -i 6 -r 0 -R im_lsa_20.result lsa.test im.key im.model lsa lsa.model

其含义为对 **lsa.test** 中的第 6 列(列从 0 开始)进行预测，结果放在 **im_lsa_20.result** 文件中，并将原文件的第 0 列和结果一起输出。其中词典为 im.key，SVM 模型为 im.model，LSA 矩阵前缀为 lsa，LSA 模型为 lsa.model。

3. **输入格式**

没有特定的输入格式。具体需要预测的内容可以通过 –i 指定。

4. **结果**

预测的结果会写入到指定的文件中。其中第 0 列为 SVM 模型预测的分数，第 1 列为 LSA 模型预测分数。其余列为指定的需要同结果一同输出的内容。

5. **参数说明：**

-i，--indexes,输入文本中训练的模型的部分（从 0 开始编号），默认为1，输入时可用 –i 1,2,3 表示使用第1,2,3 作为训练的内容

-r，----result_indexes。指定与预测分数一块输出的文本的指标项，其中预测分数放在第一列，其余的依次排列。默认为1，调用方式为 –r 1,2,3

-R,--result_save 。结果保存的路径及文件名称。

2.5 一些有用的工具

2.5.1 结果分析程序 result_anlaysis.py

如果需要对模型的参数进行调整，则需要对模型预测的指标进行计算。该模块主要读出程序的结果，然后计算相应的 F 值、召回率、正确率，以及做一些统计分析，并能根据相应的指标选择最优的阈值。

1. **说明**

可以对分类的结果进行统计分析。包括分类准确率、F 值、召回率、准确率、宏观分类准确率、微观分类准确率、设定阈值的 F 值、召回率、准确率。

2. **调用示例：**

usage:%prog [options] filename

完整形式：

3. **输入格式**

   通常情况下，结果文件中需要有预测标签、预测分值、真实标签这 3 个字段。

4. **结果**

   返回计算的指标。根据选择的 step 最后返回的结果会有所不同。

5. **参数说明：**

默认情况下将会对分类准确率、F 值、召回率、

-s ，--step 。选择的步骤：

1 为多分类以及二分类的微观分类准确率，宏观分类准确率，所有类的分类准确率。

2 为多分类以及二分类中各个类别的 F 值、召回率、准确率。

3 为计算多分类以及二分类中对指定的类别，对特定阈值下的 F 值、召回率、准确率。

4 为多分类以及二分类中计算所有类别的在阈值区间中的每个阈值每个类别的 F 值、召回率、准确率，旨在为用户分析出每个类别最好的阈值。

-i，--indexes 。输入的数据文件字段。默认情况下位[0,1,2]，即第 0 列为类标签，第 1 列为预测分数，第 2 列为实际的类标签。

-p,-- predicted_label_index 。指定预测类标签字段的位置，默认为第 0 列

-v,-- predicted_value_index 。指定预测分数字段的位置，默认为第 1 列

-t","--true_label_index。指定实际类标签字段的位置，默认为第 2 列

-e","--threshold"。在-s 3 时设定的阈值.设定阈值，计算在该阈值下的指定类别的 F 值、召回率、准确率。默认情况下为 0

-l","--label"。在-s 3 时设定的类别.设定需要的计算的类别，计算类别下在设定阈值下的F 值、召回率、准确率。默认情况下为1

-o","—output，指定分析结果输出位置，默认为标准屏幕输出

-m","—min，设定阈值的最小范围，在-s 4 时，指定搜索阈值的最小范围

-M","—max，设定阈值的最大范围，在-s 4 时，指定搜索阈值的最小范围

通常搜索会在[min,max）下取步长0.1 进行搜索

2.5.2 选择子集 subset.py(libsvm)

1. 说明

   此函数为从大数据集总选择较小的子集。此工具由 Chih-Jen Lin 提供

2. 调用示例：

Usage: %s [options] dataset number [output1] [output2]

python subset.py –s 0 data.txt 3000 output1.txt output2.txt

即从 data.txt 中选择 3000 个样本，并将选择的子集输出到 output1.txt，其余的部分输出到 output2.txt 中。其中参数-s 如果为 0，则子集选择为分层抽样，即子集仍然会保持原数据集中各个类的比例。如果-s 为 1，则从源数据集中随机选择，不会考虑各个类的比例。

3. 输入格式

   1) 如果设置-s 0 则，需要以下输入格式：

      第一列为类别，如果为 **binary** 分类。最好 Positive 为 1，Negative 为 -1。如果为**多**分类，正常样本为-1，其余的类可以选择 1,2,3,4……

      其余列为内容，可以有多列内容。

   2) 如果设置-s 1,则输入格式没有限制

4. 结果

   选择的子集输出到 output1.txt，其余的部分输出到 output2.txt 中

5. 参数说明：

   -s 子集选择的方法，默认为 0

      0 – 分层抽样

      1 – 随机选择

   output1： 子集的输出(optional)

   output2： 剩余部分的输出 (optional)

2.5.3 SVM参数选择 grid.py (libsvm)

1. 说明

   此函数为从为 SVM 模型搜索最优的参数(c,g)。此工具由 Chih-Jen Lin 提供。

2. 调用示例：

Usage: grid.py [-log2c begin,end,step] [-log2g begin,end,step] [-v fold]
[-tmstrain pathname] [-gnuplot pathname] [-out pathname] [-png pathname]
[additional parameters for tms-train] dataset

示例：python grid.py –log2c -5,15,2 –log2g -3,13,2 –v 5 tms.train

即对 tms.train 进行参数搜索，c 从-5 到 15，步长为 2，g 从-3 到 13，步长为 2。其中 tms.train 是 libtms 特定的输入格式：

<label> <index1>:<value1> <index2>:<value2> ...

<label>为类标签。<index1>整数，必须要按升序排列。可以通过调用 OLE_LINK19 来生成相应的输入格式。

3. 输入格式

   <label> <index1>:<value1> <index2>:<value2> ...

   .

   .

   <label>为一个整数值，代表着类标签。<index>:<value>指定对应特征上的值。<index>是从 1 开始的整数值，必须保持升序。如果第 i 个特征上的值为 0，则可以省略不写。

4. 结果

   该程序产生的结果文件较多，多为一些中间结果，最优的 c 与 g 可以在控制台查看。

5. 参数说明：

   -log2c begin,end,step 设置参数 c 搜索的范围及步长

   -log2g begin,end,step 设置参数 g 搜索的范围及步长

   -v fold 设置交叉验证的 folds，默认情况下为 5

   -tmstrain pathname 设置 tms-train 程序的路径,默认为当前路径

   -gnuplot pathname 设置 gnuplot 程序的路径，默认为/usr/bin/gnuplot

6. Note

   对于大数据集(样本>5000)，一般要进行两步：先选取子集选择子集 subset.py 然后进行粗粒度搜索，再对全数据集进行细粒度搜索。

粗粒度搜索是指在一个较大的范围内加大步长。细粒度搜索是指在进行粗粒度搜索后，得到最优的 c,g，然后在这个值周围选取局部区域，调小步长，再进行搜索。即可得到最优的参数

2.6 系统使用示例

具体的示例代码见 src/example.py

2.6.1 一个完整的示例

```
'''假设data文件夹下有一个post.train和post.test的训练样本和测试样本，每一行有3个字段：label title content。样本都没有分词
该例子需要完成：
1、对title进行分词、训练，模型保存在../data/post/ 下，所有的文件都有title命名，SVM模型选择使用libsvm，核函数使用rbf，选择选择保留top 40%的词，特征权重使用tf*idf
2、对title和content一起进行分词、训练，模型保存在../data/post/ 下，所有的文件都有title_content命名，SVM模型选择使用liblinear，选择选择保留top 20%的词，特征权重使用tf
3、先对post.test进行分词，然后使用已经训练好的模型对post.test进行预测。结果以post.result命名，将原label与结果一同输出。
4、计算模型的预测F值、Recall、Precision，并将结果输出在屏幕上。
5、计算从[0,1]区间内各个阈值下对应的F值、Recall、Precision，将结果保存在post.analysis
'''
```

- tms.tms_train("../data/post.train",indexes=[1],main_save_path="../data/",stopword_filename="../data/stopwords.txt",svm_type="libsvm",
  svm_param="-t 2",config_name="title.config",dic_name="title.key",
  model_name="title.model",train_name="title.train",
  param_name="title.param",ratio=0.4,seg=1,local_fun="tf",
  global_fun="idf")
- tms.tms_train("../data/post.train",indexes=[1,2],main_save_path="../data/",stopword_filename="../data/stopwords.txt",svm_type="liblinear",config_name="title_content.config",dic_name="title_content.key",model_name="title_content.model",train_name="title_content.train",param_name="title_content.param",ratio=0.2,seg=1,local_fun="tf",global_fun="one")
- tms.tms_predict_multi("../data/post.test",config_files=["../data/model/title.config","../data/model/title_content.config"],indexes_lists=[[1],[1,2]],result_save_path="../data/post.result",result_indexes=[0],seg=1)
- tms.tms_analysis("../data/post.result",step=2,output_file="",indexes=[0,1,2],predicted_label_index=0,predicted_value_index=1,true_label_index=2)

- `tms.tms_analysis("../data/post.result",step=4,output_file="../data/post.analysis",min=0,max=1,indexes=[0,1,2],predicted_label_index=0,predicted_value_index=1,true_label_index=2)`

2.6.2 更多的案例

```
import tms
'''采用程序默认参数对模型训练、预测、结果分析'''
#模型训练，输入的文件为binary_seged.train,需要训练的为文件中的第1个字段(第0个字段为lablel),保存在data文件夹中。特征选择保留top %10的词，使用liblinear
tms.tms_train("../data/binary_seged.train")

#模型预测，
tms.tms_predict("../data/binary_seged.test","../data/model/tms.config",result_save_path="../data/binary_seged.result")

#对结果进行分析
tms.tms_analysis("../data/binary_seged.result")

'''配置多个模型进行预测'''
tms.tms_predict_multi("../data/binary_seged.test", ["../data/libsvm_model/tms.config","../data/liblinear_model/tms.config"],indexes_lists=[[1],[1]],result_save_path="../data/binary_seged.result")

#对预测结果进行分析
tms.tms_analysis("../data/binary_seged.result",indexes=[0,1,2,3,4],true_label_index=4)

'''对文件进行分词'''
tms.tms_segment("../data/binary.train", indexes=[1])

'''特征选择'''
tms.tms_feature_select("../data/binary_seged.train", indexes=[1], global_fun="idf", dic_name="test.key", ratio=0.05, stopword_filename="")

'''将输入文件构造为libsvm和liblinear的输入格式'''
tms.cons_train_sample_for_svm("../data/binary_seged.train", "../data/model/dic.key", "../data/tms.train", [1])
```

3 技术细节

3.1 Latent Semantic Analysis

假设原来的词-文档矩阵为 $X_{m,n}$，即有 m 个 term，n 篇文档。$d_j = X_{.j}$ 表示第 j 篇文档的向量。$X_{m,n} = U_{m,n} \times S_{n,n} \times (V_{n,n})^T$，经过 SVD 分解后，选择前 k 个特征值 $X'_{m,n} = U'_{m,k} \times S'_{k,k} \times (V'_{n,k})^T$ 后。再去重组文档的特征向量，$d_j = (S^i V^i)_{.j}$，这样新的文档特征向量就由原来的 m 维降至 k 维。而一个新的文档即可通过 $d = d^T U'$，映射到 U 空间上。其实还有另外一种方法，就是 $d = d^T U' S'$，但是在实验中发现，前一种映射效果会更好一点。另外 wikipedia 上对 LSA 也有很详细的阐述

本系统将 LSA 用来 Classification 上的方法是一种叫做 local relevancy weighted LSI 的方法。其主要步骤为

* 模型训练
  - ① 训练初始分类器 C_0
  - ② 对训练样本预测，生成初始分值 $f(rs_i) = \frac{1}{1+e^{-a(rs_i+b)}}$
  - ③ 文档特征向量变换 $\vec{d}_i' = \vec{d}_i * f(rs_i)$
  - ④ 设定阈值，选择 top n 文档作为局部 LSA 区域
  - ⑤ 对局部词/文档 矩阵做 SVD 分解。得到 U、S、V 矩阵
  - ⑥ 将其他的训练样本映射到 U 空间中
  - ⑦ 对所有经过变换后的训练样本进行训练，得到 LSA 分类器
* 模型预测
  - ① 利用 C_0 预测得到其初始分值
  - ② 文档特征向量变换
  - ③ 映射到 U 空间
  - ④ 利用 LSA 模型进行预测得分

3.2 Feature Selection

特征选择是就是依据某种权重计算公式从词典中选择一些有代表性的词。常用的特征选择的方法有很多种，Chi、Mutual Information、Information Gain。另外 TF、IDF 也可以作为特征选择的一种方法。在这个问题上很多人做了大量的实验，

Chi 方法是效果最好的一种，所以本系统中采用了这种方法。关于特征选择无论是 Wikipedia 还是 Paper 中都有很细致的讲解。

3.3 SVM参数选择

Libsvm 中最重要的两个参数为 C 和 gamma。C 是惩罚系数，即对误差的宽容度。c 越高，说明越不能容忍出现误差。C 过大或过小，泛化能力变差。gamma 是选择 RBF 函数作为 kernel 后，该函数自带的一个参数。隐含地决定了数据映射到新的特征空间后的分布，gamma 越大，支持向量越少，gamma 值越小，支持向量越多。支持向量的个数影响训练与预测的速度。这个问题 **Chih-Jen Lin** 在其主页上有详细的介绍。

而 Liblinear 的 C 参数也是非常重要的。

因此在系统中会通过 5-flods 交叉验证的方法对一定范围内的 C,gamma 进行 grid 搜索，关于 grid 搜索可以参考论文以及 libsvm 中 tool 文件夹中 grid.py 源文件。grid 搜索是可以得到全局最优的参数的。

为了加快 SVM 参数搜索的效率，采用两种粒度的搜索-**粗粒度**和**细粒度**，两种搜索方式的区别就是搜索步长不同。粗粒度是指搜索步长较大，为了能在较大的搜索范围内找到一个最优解所在的大体区域。细粒度搜索搜索步长较小，为了能在一个较小范围内找到一个精确参数。

而对与大样本的文件，使用上面的方法仍然会比较耗费时间。为了进一步提高效率，同时在保证得到全局最优的情况下，先对选择大样本的**子集**进行粗粒度的搜索，然后得到在得到的最优区间内对全量样本进行细粒度的搜索。

3.4 Libsvm 与liblinear的多分类策略

libsvm 的多分类策略为 one-againt-one。总共有 k*(k-1)/2 个 binary classifier，对这 k*(k-1)/2 个 binary classifier 的 value 进行遍历，如果第 i 个类和第 j 个类 binary 的 classifier 的 value 大于 0，则会给第 i 个类投 1 票，否则给第 j 个类投 1 票。选择最终获得投票数最多的类作为最终的类别。

而 liblinear 的策略为 one-against-rest。总共有 k 个 binary classifier。从所有 binary classifier 中选择值最大多对应的类别作为最终的预测类标签。

3.5 特征权重

文档特征向量的特征权重计算的一般公式为 $a_{ij} = Local(i,j) * global(i)$，即第 i 个 term 在第 j 篇文档向量中的权重。其中 Local(i,j)被称为局部因子，与 term 在文档中出现的次数有关。global(i)又称为 term 的全局因子，与在整个训练集中 term

出现有关。通常我们熟悉的公式都可以转化为这一个通用的表达式。如最常用的 tf 形式 $Tf : L(i, j) = a_{i,j}; G(i) = 1$，

tf*idf 形式 $Tf * Idf : L(i, j) = \log(a_{i,j} + 1); G(i) = \log(\frac{N}{n})$。因此我们就可以在构造词典的时候就计算 term 的全局因子，把这个值放在词典中，然后在计算特征权重的时候直接调用。

在 Classification 中哪种特征权重的计算方式最好？？tf*idf ？在文献中最常用的是 tf*idf，但是其效果并一定好。曾经有人也在这上面做了一些工作，比如新加坡国立大学的 Man Lan 曾在 ACM 和 AAAI 上发表过文章来阐述这个问题。Zhi-Hong Deng 也对各种 feature weight 的方法做了系统的比较，最终的结论是 tf*idf 并不是最佳的，而最简单的 tf 表现不错，一些具有区分性的方法比如 tf*chi 等效果差强人意。

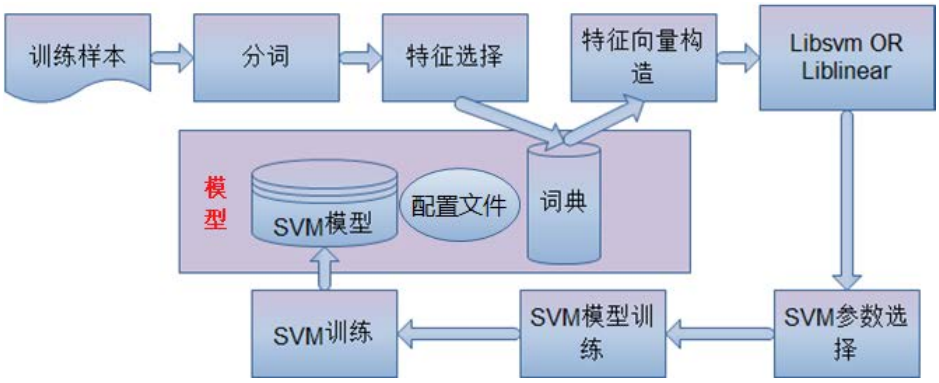
后来 Man Lan 在 09 年发表了一篇论文，对 term weighting 方法做了一个综合细致的阐述，并对其提出的 tf*rf 方法做了各方面的论证，

3.6 模型返回结果

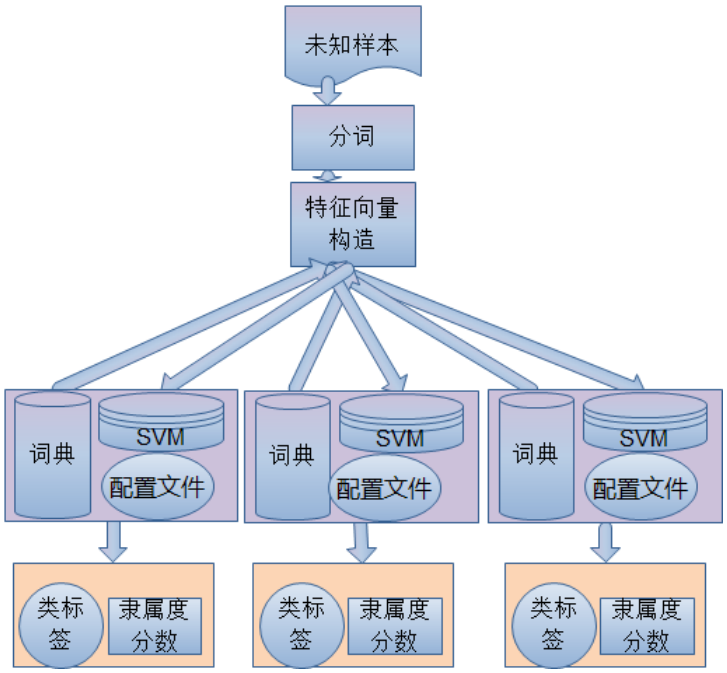
模型会返回两个结果：label 和 score，其中 label 即其预测的标签。而 score 是该样本属于该类的隶属度，分值越大，代表属于该类的置信度越大。具体的计算方式则是根据公式 $score = \frac{\sum s_i}{2*k} + \frac{k}{2*n}$，，其中 k 为所有支持判别类得个数，n 为所有类别个数，si 为所有支持判别类的分数。返回 score 的好处是对与 information filtering 问题，因为训练样本的 unbalance 和 randomly sampling 问题，依据判别的标签得到的结果准确率较低，因此需要通过阈值控制。

3.7 模型训练和测试流程

训练过程：对文本自动做 SVM 模型的训练。包括 Libsvm、Liblinear 包的选择，分词，词典生成，特征选择，SVM 参数的选优，SVM 模型的训练等都可以一步完成。示意图见下面（图有点丑，表达清意思就好^_^）



模型测试过程：

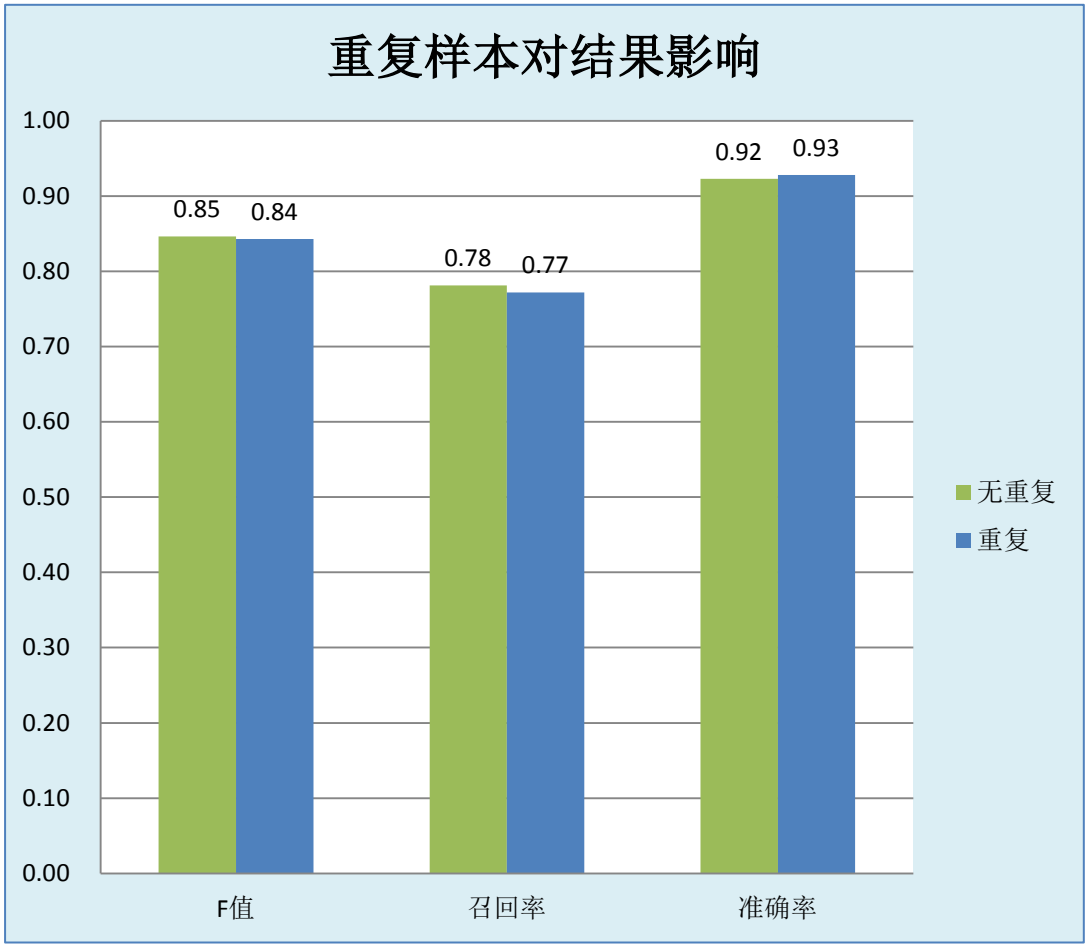


3.8 重复样本对SVM模型的影响

重复样本对于 SVM 模型有怎样的影响呢？

我自己做了个实验，用来看重复样本的影响。

原有一个训练样本共有 Positive 样本 1000，Negative 样本 2000，然后将 Positive 样本*2，构造了一个 Positive 样本 2000，Negative 样本 2000 的训练样本。然后测试一个包含 Positive 样本 4494 ，Negative 样本 24206 的样本。最终的结果如下：



从结果上来看：在 F 值上，无重复的样本会比重复样本稍高(图中保留了 2 位小数，其实差异不超过 0.5%)。而正确率上，重复样本会比无重复样本稍高。

然后我又把模型放入到一个包含 3 千万样本中去测试，具体的指标无法测算。但是感觉还是重复样本会好一点。

具体分析：

1、 一个样本被重复的多次，意义上相当于增加了该样本的权重。在 SVM 有一种 Weighted Instance。在正常样本难免会有些误判，如果同一条样本同时出现在 Positive 和 Negative 类中，包含重复样本的 Positive 类就会把 Negative 类误判的样本的影响抵消。而在 SVM 分类中对这些离群点会用惩罚函数进行控制。

2、 但是如果保留重复样本，会增加样本的量，对 libsvm 来说，分类的复杂度为 $O(N_{sv}^3)$，而且如果一个样本是支持向量，那么所有重复的样本也都会被加入到支持向量中去。而且如果要为 SVM 模型选择合适的参数的，如果在 SVM 选择的是 RBF 核函数，挑选合适的惩罚 cost 和 RBF 的参数 gramma，如果在都是在[1,5,0.5]进行挑选，则总共会有

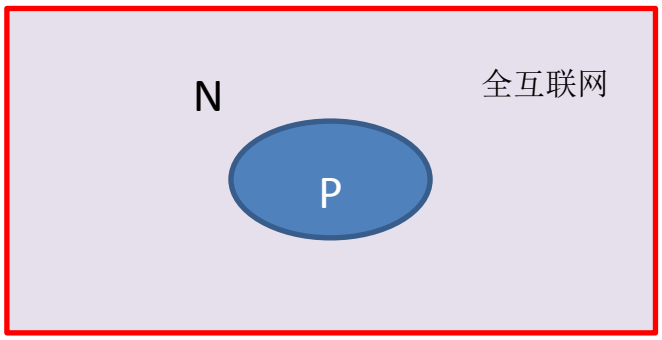
9*9=81 组参数需要挑选，在每组参数下如果要进行 5-flods 的交叉验证，则需要 81*5=405 次训练与测试的过程。如果每次训练与测试花费 2 分钟（在样本达到 10 万数量级的时候，libsvm 的训练时间差不多按分钟计算），则总共需要 405*2/60=12.3 小时，所以说训练一个好的 SVM 模型十分不容易。因此如果去掉重复样本对训练效率来说大有裨益。

3.9 将分类应用与信息过滤

分类应用与信息过滤，对最终效果影响最大的是什么？分类算法？词典大小？特征选择？模型参数？这些都会影响到最终的过滤效果，但是如果说对过滤效果影响最大的，还是训练样本的采样。

现在基于机器学习的分类算法一般都是基于一个假设：训练集和测试集的分布是一致的，这样在训练集上训练出来的分类器应用与测试集时其效果才会比较有效。

但是信息过滤面对的数据集一般是整个互联网，而互联网的数据集一般很难去随机采样。如下图所示：通常来说，信息过滤或其它面向全互联网的应用在分类，选择数据集时，需要包含 P（Positive，即用户感兴趣的样本），N(Negative，即用户不关心、不敢兴趣的样本)。最理想的情况是:P 选择是用户感兴趣的，而 N 是全网中除去 P，显而易见 N 是无限大的，而且很难估计其真正的分布，即无法对其随机取样。



同样面对整个互联网的应用时网页分类，网页分类应用一般会选择 Yahoo!或者是专门整理网页分类专门网站的网页作为初始训练样本。

信息过滤的样本一般来说，感兴趣的样本是很好随机采样的。但是与感兴趣相对于的是正常样本，这个很难去选择。而正常样本对全网测试效果是影响非常大的。我曾经做过一个实验：

首先，有一个包含 5 万条样本的数据集，有 2.5 万条 Positive 样本，2.5 万条 Negative 样本。这里的 Negative 样本是以前用关键字的方法找出的不正确的样本。用 4 万条样本做训练样本，用 1 万条样本做测试样本。训练出得模型交叉验证的

结果可以达到 97%以上。在测试样本中的测试效果，然后选定阈值为 0.9，这是的召回率可以达到 93%，正确率为 96%。

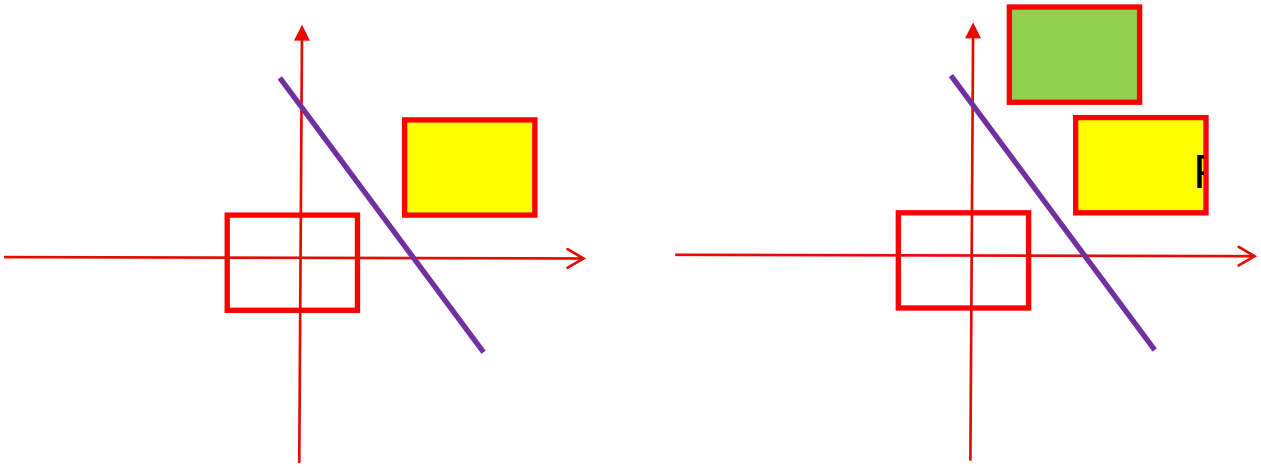
然后把这个模型放到一个包含 3 千万条中去测试，设置阈值为 0.9，共找出疑似违规样本 300 万条。对这个实验来说，召回的样本实在是太多了，其正确率是很低的。

然后，我又更换了一下正常样本。从这 3 千万样本中随机采样出 3 万条样本，然后经过校验，将其中 Positive 的样本剔除掉。剩下大约 2 万 7 千条样本放入到训练样本重新训练。

把得到的新模型放到 3 千万样本中测试，同样设置阈值为 0.9，共找出疑似样本 15 万。正确率可以达到 70%左右。所以正常样本的随机选择对分类来说同样至关重要。

举一个小例子：

下图左面的图是用 P 和 N 训练出得模型。右面的图中有一个未知的类 C，根据已知的模型，他应该会被分入到 P 中，但是实际上他是不属于 P 的。一般情况下，这种情况可以用阈值来控制。



3.10 训练样本如何选择

4 源码剖析

请原谅作者在有些地方没有按照 OOP 思想设计程序，函数式的流程仅仅是为了能过更好的表达流程，并尽可能的做到函数的复用。

4.1 将单个文本构造成SVM的输入格式

```
def cons_pro_for_svm(label,text,dic,local_fun,global_weight):
    '''根据构造的输入的类标签和以及经过分词后的文本和词典，SVM分类所用的输入格式，会对特征向量进行归一化
        注意：这个实现已经去除了全局因子的影响，意味着特征权重直接使用词频。
    x begin from 1'''
    y=[float(label)]
    x={}
    real_x={} #因为x的keys可能是无序的，所以要先对x中的进行排序，然后
    if len(global_weight)<1:
        for i in range(len(dic)+1):
            global_weight[i]=1

    #构造特征向量
    for term in text:
        term  = term.strip()
        if dic.has_key(term) :
            index = int(dic.get(term))
            if x.has_key(index):
                x[index]+=1.0
            else:
                x[index]=1.0
    # 计算特征向量的特征权重
    for key in x.keys():
        x[key] = local_fun(x[key])*global_weight.get(key)

    #计算特征向量的模
    vec_sum = 0.0
    for key in x.keys():
        if x[key]!=0:
```

```
25              vec_sum+=x[key]**2.0
26          #对向量进行归一化处理。
27          vec_length=math.sqrt(vec_sum)
28          if vec_length!=0:
29              for key in x.keys():
30                  x[key]=float(x[key])/vec_length
31          return y,[x]
```

该段代码的目的是对每个样本根据词典和局部函数和全局因子计算该样本的特征向量，然后转换为 SVM 特定的输入格式。

SVM 的输入格式是类标签为 list,所有的属性值为 list,每个样本的属性为一个 dict()，key 为 index,value 为其属性值。因为 `global_weight` 即全局因子可能不填入，所以有个默认值 `dict()`，如果为默认值就会自动填入 `1`。第 `1-6` 做的就是这个事情。

第 8-16 行就是根据比对词典，找出每个词出现的次数。

第 17-19 行就是计算向量的特征权重，根据公式 $a_{ij}^{'} = local(i,j) * global(i)$ 来计算。

第 21-30 就是对特征向量归一化

最后返回类标签 y 和特征属性 x

4.2 将所有文本构造成SVM的输入格式

```
def cons_train_sample_for_cla(filename,indexs,local_fun,dic_path,sample_save_path,delete,str_splitTag,tc_splitTag):
    '''根据提供的词典，将指定文件中的指定位置上的内容构造成SVM所需的问题格式，并进行保存'''
1       dic_list,global_weight = fileutil.read_dic_ex(dic_path,dtype=str)
2       if type(local_fun)==types.StringType:
3           local_fun = measure.local_f(local_fun)
4
5       f= file(filename,'r')
6       fs = file(sample_save_path,'w')
7       for line in f.readlines():
8           text = line.strip().split(tc_splitTag)
```

```
9           text_temp=""
10          if len(text)<indexs[len(indexs)-1]+1:
11              continue
12          for i in indexs:
13              text_temp+=str_splitTag+text[i]
14          y,x =
    ctmutil.cons_pro_for_svm(text[0],text_temp.strip().split(str_spl
    itTag),dic_list,local_fun,global_weight)
15          if delete == True and len(x[0])==0:
16              continue
17          save_dic_train_sample(fs,y,x)
18      f.close()
19      fs.close()
```

该段程序的目的是从二进制文件中读入所有的样本，然后将其转换为 SVM 的格式并进行保存。

值得注意的是在 2-3 行用了 Python 的函数式编程，将一个函数直接传递给一个变量。

第 8-13 的目的是因为一个样本中可能有多个字段（如标题、内容），如果想把多个字段合成一个字段，则需要用原本分词的符号将他们链接起来。

而第 17 行是将得到的类标签和属性值以特定格式写入到文件中。

4.3 Libsvm与Liblinear差异

4.4 Result_analysis

4.4.1 递归保存结果dict

结果分析总共有 4 个不同的过程，这 4 个过程产生的结果统一用 dict()进行存储，这样就产生一个问题：当写入文件时，对不同层次的 dict 该怎么处理，比如{1:2,2:3} 和{1:{0.1:1},2:{2:{2:3}}}和{1:{0.1:[1,2,3]},2:{0.2:2}} 这种不同深度且存储类型不一应该怎么才能以规范的形式写入到文件中呢？

OK，我们考虑使用递归来实现。

递归将不同层次的词典以规范的形式写入到文件

```
1   def save_result(f,rate_dic,count=0):
```

```
2    if type(rate_dic)==types.DictType:
3        f.write("\t")
4        temp=count
5        for key in sorted(rate_dic.keys()):
6            f.write("\n")
7            f.write("\t"*temp+str(key))
8            count=temp
9            count+=1
10           save_result(f,rate_dic[key],count)
11   else:
12       if type(rate_dic) in(types.ListType,types.TupleType):
13           f.write("\t"*count)
14           for value in rate_dic:
15               f.write(str(value)+"\t")
16       else:
17           f.write("\t"*count)
18             f.write(str(rate_dic)+"\t")
```

让我们来分析一下这段程序：

如果现在的`rate_dic`变量类型为词典，就会对遍历所有的`key`，并将该`key`对应的`value`赋予`rate_dic`再次进行递归调用保存函数。直到`rate_dic`不是词典类型。第 12、16 行所示，对于不同的元素类型类型，以不同的方式写入。

如果仅仅是解析词典，将里面的元素写入文件很简单，但是如果要根据词典的层次输出缩进的个数就必须要由技巧了。

该函数还有另外一个参数 count，其作用就是记录当前的深度，以便输出相应的缩进。而为了保证同一层次上的元素缩进相同就要记录当前的深度，第 4 行非常关键。他可以保证所有的同一层次上的 key 都有相同的缩进。第 9 行就负责将深度加 1。而函数又使用了默认参数 count=0,就意味着调用者可以不用管初始的缩进。

4.4.2 对特定类别计算指定阈值下的各种指标

对特定类别计算指定阈值下的 F 值、Recall、Precision

```
def cal_f_by_threshold(true_lab,pre_lab,pre_value,label,threshold):
    true_sum , pre_sum , right_sum =0.0,0.0,0.0
    f_sc,recall,precision=0.0,0.0,0.0
    rate = dict()
    for j in range(len(true_lab)):
        if true_lab[j]==label:
            true_sum+=1.0
            if pre_lab[j]==label and pre_value[j]>=threshold:
                right_sum+=1.0
        if pre_lab[j]==label and pre_value[j]>=threshold:
            pre_sum+=1.0
    recall = right_sum/true_sum
    if pre_sum!=0:
        precision=right_sum/pre_sum
    if recall+precision!=0:
        f_sc = 2*recall*precision/(recall+precision)
    rate[label]=[f_sc,recall,precision]
     return rate
```

该程序是十分简单的，只要是知道各个指标的计算公式就好了。

	actual class (expectation)	
predicted class (observation)	**tp** (true positive) Correct result	**fp** (false positive) Unexpected result
	fn (false negative) Missing result	**tn** (true negative) Correct absence of result

$$F = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} \mathrm{Precision} = \frac{tp}{tp + fp} \mathrm{Recall} = \frac{tp}{tp + fn}$$

另一个值得注意的是，程序返回的结果仍然是dict形式，主要是为了结果的统一便于接收以及保存。

5 疑问点

在开发这个系统的过程中也有一些没有解决的问题，现在列在这里，希望有解决方法的同学联系我 ^_^，zhzhl202@163.com。

1、在实现有关 Feature Extraction 的算法中，参考了有关 LSA、PCA 用在 Classification 中的 Paper，也实现了其中一些比较好的算法，但是在测试的时候发现有些算法并不如说的那么有用，现在也没有找到一个非常有效的有关在 Text Classification 中做 Feature extraction 的方法。所以如果大家有好的方法，请告诉我^_^.

6 FAQ

关于我

浙江大学研究生，专注 Data Mining、Text Mining、Recommendation System、Information Filtering。熟悉 Python、Java，了解 C++,喜欢 Linux 的高效、也喜欢 windows 的花哨。关注开源，爱好篮球，足球。

Contact me:

张知临

Mail： zhzhl202@163.com

weibo:张知临 Zjuer

Thanks

本系统引用了 libsvm、liblinear 的包，非常感谢 Chih-Jen Lin 写出这么优秀的软件。本系统还引用了 Pymmseg，非常感谢 pluskid 能为 mmseg 写出 Python 下可以直接使用的程序。

感谢杨铮、江洋、敏知、施平等人的帮助，特别感谢丽红与家人的关心与支持。