****

**计算机学院**

**数据仓库与数据挖掘课程项目报告**

**设计题目**： 基于支持向量机的文本分类

**学生姓名**： 蒋敏超，应宇杰，林鸿浩

**学 号**：19071216，19151633，19070119

**目 录**

1.项目背景 3

2.项目分工 4

3.实验环境 4

4.主要设计思想 4

4.1 实验工具介绍 4

4.2 实验数据 5

4.3数据预处理方法设计 5

4.4数据挖掘算法的选择与设计 6

4.5性能评估方法 7

5.1 已实现的功能点 9

5.2创新点 10

5.3项目实现的主要步骤 10

5.4建模与测试 11

5.5使用说明 16

6.结果分析 17

7. 收获和建议 19

8.项目源代码（每个部分写明是哪位同学完成） 20

# 1.项目背景

随着经济全球化的不断扩大，各国各地区的文化交流与商业贸易越加频繁，熟练掌握一门甚至多门外语也越来越有必要。

当你上搜索引擎试图查找专业资料，往往会发现外文文献更多。此时，如果你对外语比较熟练，便能更快更准更有效地获取想要的资料。

掌握外语，有利于扩大交友圈，结识到更优秀更有趣的外国友人，了解到国内外文化的差异，同时对自己的性格和习惯也有很大帮助。

当你出国旅行，如果能自由地与当地人交流，跟蹩脚地比画着、看着所谓的旅行攻略相比，游玩的感受会完全不同。不少行业和职位都要求应聘者至少得懂几门外语，以便业务更好地开展。从职业发展的角度来看，外语也是一项备受关注的专业技能，能够提升个人的求职竞争力。

而学好一门外语，大量的原文阅读必不可少。它有助于培养语感，丰富词汇积累，了解所学语言国家的文化，或者更直接的，有效提高你的外语成绩。

但是外文如此之多，如何根据自己现有的水准去选择适合自己阅读的文章呢？而这个适合，又希望能选择能较为顺畅地阅读的，或挑战性地阅读的文章。

当今，Microsoft Word 已应用了显示有关文件阅读水平的信息的功能，其使用的主要是Flesch阅读简易测验。但是这个模型仅仅考察了平均句子长度和每个单词的平均音节数，无法达到准确区分不同难度的英语文章。

所以我们就想到使用NLP提取特征，使用SVM作为分类器，来制作基于支持向量机的文本分类。

# 2.项目分工

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 学号 | 姓名 | 任务 | 完成工作量 | 确认签名 |
| 19151633 | 应宇杰 | 完成课程项目报告和数据清洗操作 | 30% |  |
| 19071216 | 蒋敏超 | 完成项目思路和代码实现 | 40% |  |
| 19070119 | 林鸿浩 | 完成PPT | 30% |  |

# 3.实验环境

Windows10操作系统，使用Jupter Notebook作为集成开发环境

Python3.7.11

Sklearn1.0.1

Jupyterlab3.2.1

numpy1.19.5

pandas1.3.4

testmetric

lightgbm

# 4.主要设计思想

## 4.1 实验工具介绍

**Jupter Notebook:**

Jupyter Notebook是一个开源的Web应用程序，允许用户创建和共享包含代码、方程式、可视化和文本的文档。它的用途包括：数据清理和转换、数值模拟、统计建模、数据可视化、机器学习等等。

**SNE**：

一种数据可视化的工具，能够将高维数据降到2-3维（降维），画成可视图，以便于观察特征分布情况。

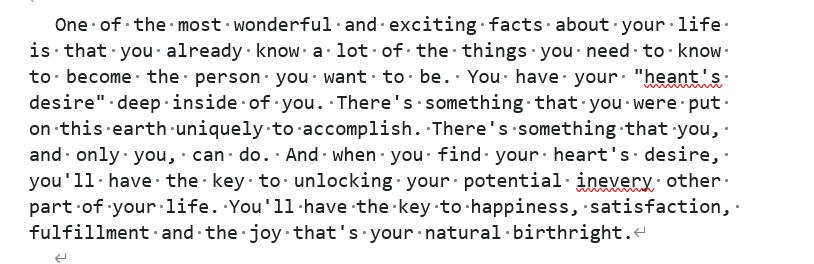
## 4.2 实验数据

我们将所有英文文本分为小学英语、初中英语、高中英语、四级英语、六级英语五大类，每类各收集了样本60篇，其中，样本的20%作为样本集进行训练测试，剩余80%用于训练。样本来源于英语教学课本文章和课外阅读文章。

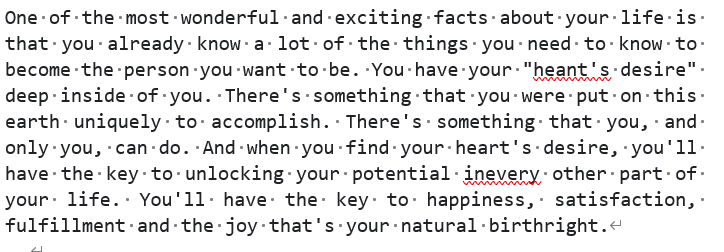
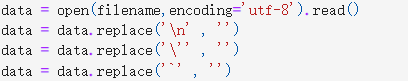
## 4.3数据预处理方法设计

总共清洗了两次，第一次清洗时我们去除了无用符号，比如换行符，第二次清洗我们将文本内的所有英文字母均转换成小写字母，删除了所有的标点和字符，删除文本停用词，将文本词干化，以方便进行下一步的特征提取。

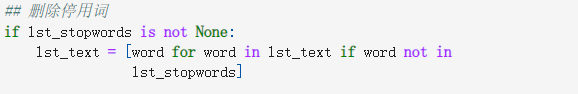
原文本：

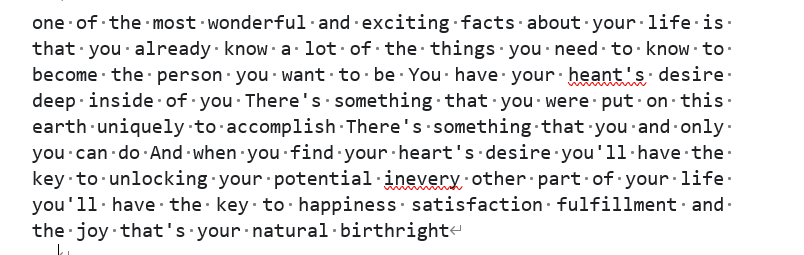


第一次清洗无用符号：



第二次清洗标点符号和把字母转化成小写





## 4.4数据挖掘算法的选择与设计

本次模型的分类器我们小组选取有SVM, RandomForest, Bayes, LogisticRegression, Adaboost, Knn, Decision\_Tree 六大分类器模型。

我们以精确性、召回率、特异性、曲线下面积、F1分数和准确度六项为评估模型优劣的指标，在独立数据集上比较测试集上的其他六个分类器，以得到最适合本次问题的模型。

**Table1.** 在独立数据集上比较测试集上的其他六个分类器

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Svm | RF | Bayes | LG | Ada | knn | DT |
| ACC | **0.87** | 0.7 | 0.73 | 0.73 | 0.3 | 0.73 | 0.6 |
| AUC | **0.97** | 0.96 | 0.95 | 0.93 | 0.83 | 0.87 | 0.73 |
| Pre | **0.86** | 0.73 | 0.63 | 0.77 | 0.12 | 0.72 | 0.6 |
| Recall | **0.84** | 0.67 | 0.68 | 0.75 | 0.29 | 0.68 | 0.56 |
| SP | **0.97** | 0.93 | 0.93 | 0.94 | 0.82 | 0.93 | 0.90 |
| F1-score | **0.84** | 0.67 | 0.64 | 0.73 | 0.17 | 0.68 | 0.55 |

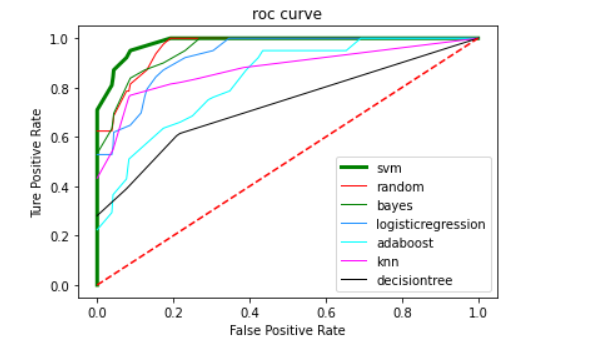
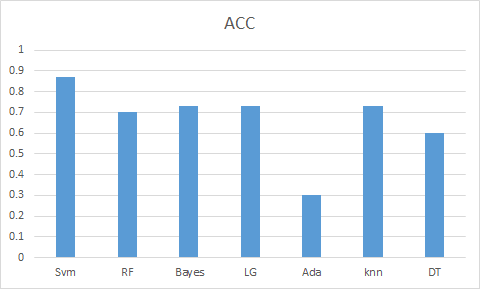


图 1分类器ACC值评估柱状图

图 2分类器评估的Roc曲线

经过我们的调参后发现，SVM的各项指标与其他6种分类器相比，都排在前列，尤其是ACC项，比其余分类器高14%以上，而AUC项更是达到了97%。因此综合考虑，SVM是本次建模算法分类器的不二之选。

## 4.5性能评估方法

**4.5.1 前提条件**

要确定这些，首先必须了解文档的分类是否为真阳性（TP）、假阳性（FP）、真阴性（TN）或假阴性（FN）。

以上几项分类在二分类中混淆矩阵如下：

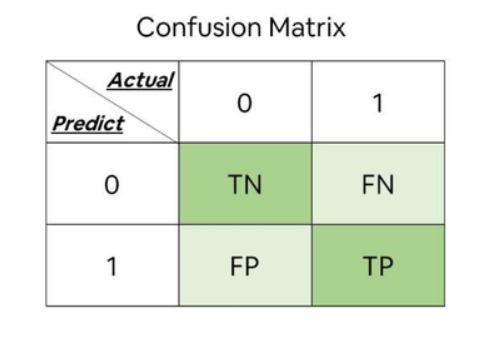


图 3 文档分类混淆矩阵图

其中：

* TP被确定为与某一类别相关的正确分类的文件。
* FP被错误地确定为与类别相关的文档。
* FN确定为未标记为与某一类别相关，但应标记的文件。
* TN文件不应标记为属于特定类别，也没有标记的。

由于我们是多分类，我们选择先分别计算每一类别的指标，再求出它们的平均值，以得到最终模型的指标。

此外，我们还需了解ROC曲线。我们在0-1中选取阈值Thresholds，根据不同的阈值划分分类的结果与正确值比较，确定预测分类的正确性，并以Fpr（False positive rate）为横坐标，Tpr（True positive rate）为纵坐标，由此绘制ROC曲线。

**4.5.2评估指标**

**准确度**通常被用作分类技术的衡量标准。然而，与精确性和召回率相比，准确度值更不愿意改变正确决策的数量：

**特异性**描述识别出的负例(TN)占所有负例(TN+FP)的比例

**召回率**指的是所有真实为正类（TP+FN）中，被判定为正类（TP）占的比例。

根据公式得

**精确率**指的是所有被判定为正类（TP+FP）中，真实的正类（TP）占的比例。

根据公式得：

**F1分数（F1-score）**是分类问题的一个衡量指标,它是精确率和召回率的调和平均数

**AUC（Area Under Curve）**被定义为ROC曲线下的面积。在许多方面例，ROC曲线不能清楚解释哪种分类器效果更好，AUC作为一个值，AUC越大的分类器效果越好。因此，我们经常使用AUC值作为模型的评价标准。

**5.项目实验过程**

## 5.1 已实现的功能点

1. **辅助备考习题挑选**。此模型能帮助你找到更加适合你现有水平的阅读理解、选词填空等习题，从而更好地备考。
2. **日常英文阅读分类**。日常阅读对于提升英语水平十分有必要。有了本模型，你便能快速分辨文章难度，自主选择能较为顺畅地阅读的，或挑战性地阅读的文章，以满足不同需求。
3. **日常英语写作评估**。作文的批改一直是费时费力的一大难题。本模型能帮助你评估写作的水平，甚至能在此基础上提出改进意见，提高写作水准。
4. **测试自身英语水平**。通过给你测试不同等级的英语文本，或将你写的英语作文进行评估，能更独立自主地帮你迅速明确自身英语水平，以更好的规划之后的学习安排。

## 5.2创新点

1、用了两种特征提取方式，一个是基于文本，一个是基于词频（TF），既考虑了词元个体在文章中的重要性，也考虑了文本的整体性特征

2、运用SVD（奇异值分解）算法进行降维，适用于任意的矩阵的分解，将原始的数据集矩阵分解，即将TF生成的6648维的稀疏矩阵降维至50维，这样做实际上是去除了噪声和冗余信息，减小大量批处理计算，简化运算，降低建模实现的时间复杂度，以此达到了优化数据、提高结果的目的

## 5.3项目实现的主要步骤

本主题要求我们构建模型来评估我们在常见情况下可能看到的英语文本的阅读难度，以便将文本作为阅读材料适当难度的语言测试。此外，还需详细说明模型的建立过程，并且将建立的模型进行改进优化，推广用于其他语言的文本阅读难度分析。基于此，我们的工作主要是包括以下内容：

1. 原始数据样本首先进行初步数据清洗，删除无用的字符
2. 将原始数据划分训练集和测试集，80%训练集，20%测试集
3. 提取NLP/文本feature，最终得到12维特征
4. 将文本数据再一次进行清洗，通过TF-IDF抽取特征
5. 奇异值分解降维后，将上两步得到的维度特征整合
6. 将训练集的维度特征导入到SVM训练，并在独立测试集上进行测试
7. 对测试集分类的结果进行评估

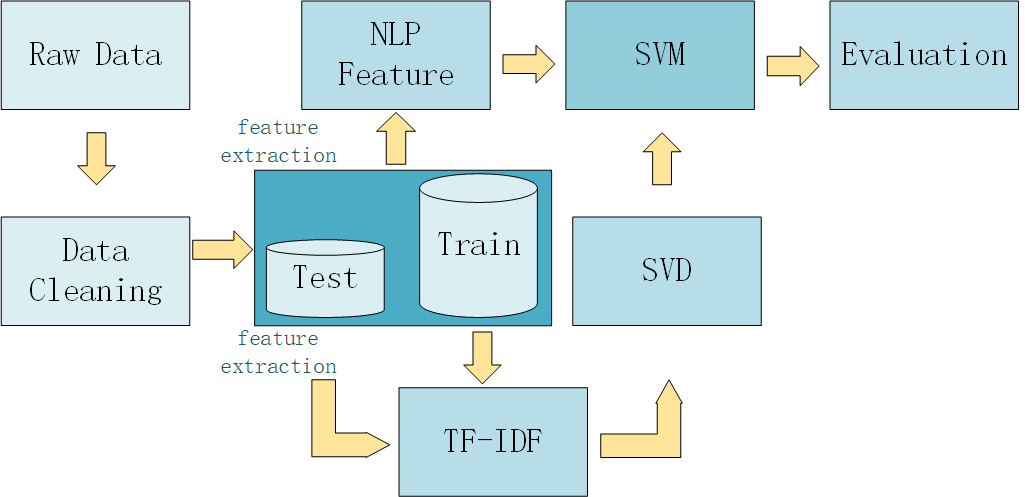


图 4模型建立流程图

## 5.4建模与测试

**5.4.1 提取NLP特征**

提取NLP特征首先，我们提取每个英文文本的以下特征：文档的字符数：文档中的字符总数；文档字数：文档中的总字数；文档的平均单词密度：文件中使用的单词的平均长度；一篇完整文章中标点符号出现的次数：文档中的非定时符号总数；词性标注的频率分布：包括名词的数量，动词、形容词、副词、代词等；停止字的数量；句子数量；平均句子长度；结果结构如图3所示。

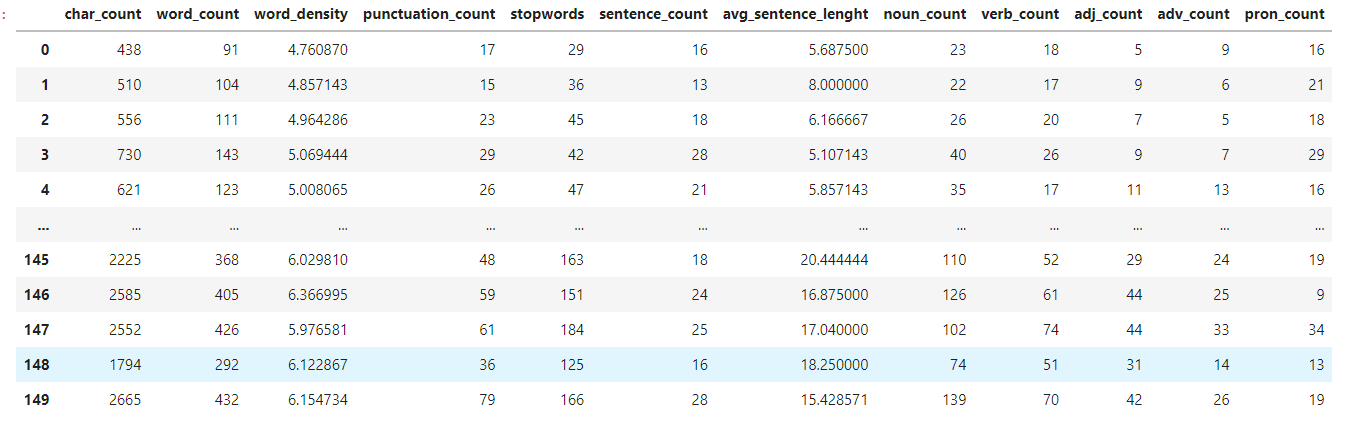
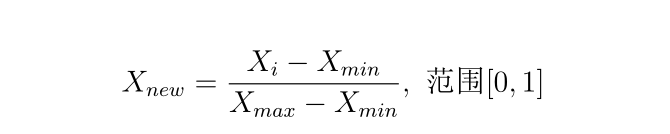


图 5 NLP特征提取

然后，我们根据训练集和测试集进行分类，并对它们进行规范化根据以下公式从所有文本中提取特征：



此时，我们已经获得了NLP特征的训练集和测试集

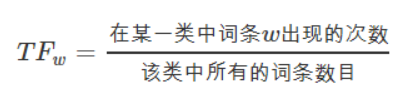
**5.4.2使用 TF-IDF 特征**

TF-IDF（Term Frequency-Inverse Document Frequency），是一种用于信息检索与数据挖掘的常用加权技术，可以评估一个词在一个文件集或者一个语料库中对某个文件的重要程度。一个词语在一篇文章中出现的次数越多

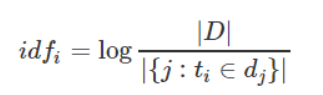
，同时在所有文章中出现的次数越少，越能够代表该文章的中心意思，字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

**TF（Term Frequency）**：表示词条（关键字）在文本中出现的频率，公式表示为

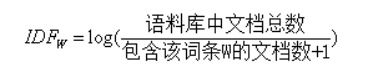
其中 ni,j 是该词在文件 dj 中出现的次数，分母则是文档 dj 中所有词汇出现的次数总和。又为防止它偏向长的文件，这个数字通常会被归一化(一般是词频除以文章总词数)，即



**IDF（Inverse Document Frequency）**：某一特定词语的IDF，可以由总文件数目除以包含该词语的文件的数目，再将得到的商取对数得到，主要思想是在语料库中包含某个词条的文档越少，该词条IDF越大，说明这个词就有很强的类别区分能力。其公式表示为



其中，|D|是语料库中文件总数， |{j:ti∈dj}| 表示包含词语 ti 的文件数目（即 ni,j≠0 的文件数目）。又由于若该词语不在语料库中，就会导致分母为零，因此一般情况下使用 1+|{j:ti∈dj}|表示分母，即



**TF-IDF**：由TF与IDF相乘可得，即



如此，某一个文件中高频出现的词条，以及该词条在整个语料库文件中低频出现的现象，可以产生出高权重的TF-IDF。也就是说，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。由此，我们便得到了TF-IDF特征。

**5.4.3特征降维融合**

此时得到的特征极有可能非常稀疏、噪声很大，并且在很多维度上非常冗余。因此，为了找出能够捕捉单词和文档关系的少数潜在主题，我们希望能降低矩阵 A 的维度。

这种降维可以使用截断 SVD 来执行。

**SVD**，即奇异值分解，是线性代数中的一种技术，其公式为

该技术将任意矩阵 M 分解为三个独立矩阵的乘积：M=U\*S\*V，其中 S 是矩阵 M 奇异值的对角矩阵。很大程度上，截断 SVD 的降维方式是：选择奇异值中最大的 t 个数，且只保留矩阵 U 和 V 的前 t 列。直观来说，可以看作经SVD处理后，只保留了我们变换空间中最重要的 t 维。

经过我们的反复试验得到，当t取值为50时，得到的评估指标最优。由此，我们便得到了50维TF-IDF特征。

将第一步得到的12维的特征与刚刚降维处理得到的50维进行拼接，得到62维特征组，为接下来的模型处理提供数据集。

**5.4.4模型SVM 参数选择选择**

支持向量机（SVM）分类器技术，由Khamar（2013）介绍分为两类困难，其基础是查看由数据类别（Sahami等人，1998年）。训练样本{（xi，ci）}Ni？1，输入主题在哪里对于第i个示例，is和is是匹配的期望响应（目标输出）。为了开始对于，它需要假设子集和主题所指示的主题（类别）子集表示为“线性可分”。为了将支持向量机分类器应用于用于等式9的超平面之间的主题分隔。这意味着支持向量机分类器即使在巨大的关键字数据集中，也可以执行此过程，因为其目的是扩展分割的边界数据，它利用而不是适合特征。支持向量机是利用预定义的分类文本。

其中x是向量机的输入，w是可调权重向量机，b是a，因此，我们可以将等式5改写为：

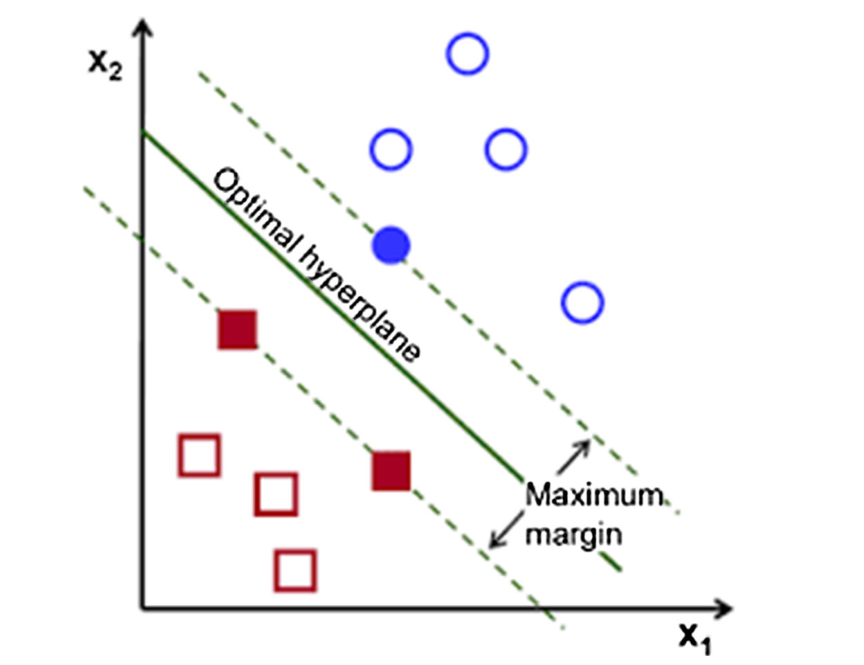


图 6二维输入空间最优超平面的几何构造

选取SVM的超参数c,gamma是十分重要的。我们使用的是网格遍历SVM的c,gamma超参数，以ACC作为评价指标。

首先选取

c = [ 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 1000 ]

gamma = [ 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]

将以上81种组合依次遍历，得到的评估结果如下图所示

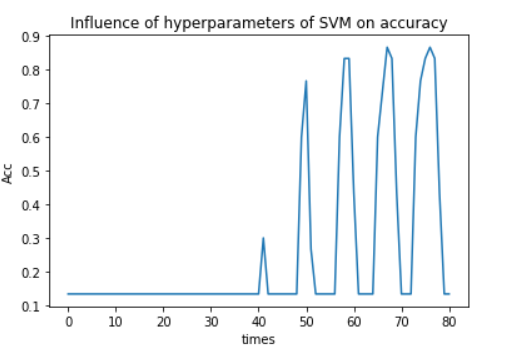


图 7 SVM第一次调参评估

根据本次调参输出的结果，找到最优的一组参数为 c=100 gamma = 0.1。随后，我们在参数附近取值，进行第二次遍历，本次取值如下

c = [ 75, 80, 85, 90, 95, 100, 105, 110, 115 ]

gamma = [ 0.02,0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18]

第二次遍历得到的评估结果如下图所示

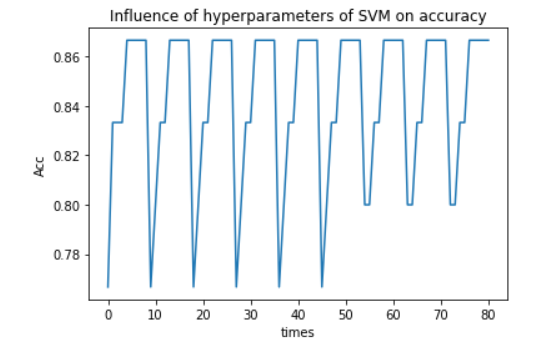


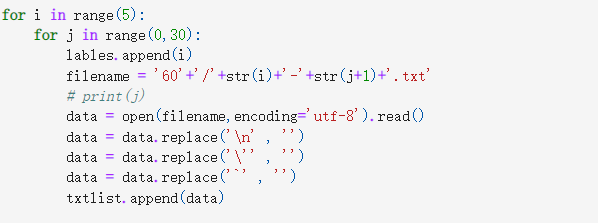
图 8 SVM第二次调参评估

根据第二次的调参结果我们发现，本次附近值的遍历结果并未提高，第一次调参后取得的最优参数仍然占据最高指标。因此，我们组最终选定 c=100 gamma = 0.1 为本次建模的SVM参数。

## 5.5使用说明

打开Jupyter Notebook,导入代码即可运行

批量导入数据集，文件名需要改成60/{i}-{j+1}.txt格式



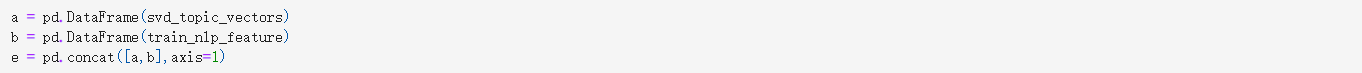
选择测试集数量为总数据集的1/5



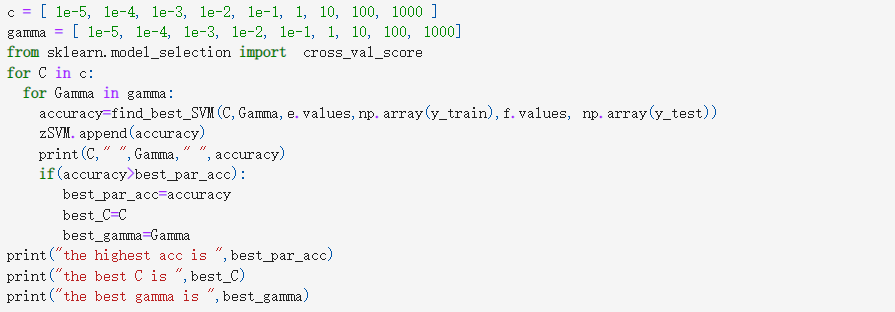
选择降维后的五十维特征



拼接特征，拼接成62维特征



选择合适的SVM超参数



数据降维并且显示出来



# 6.结果分析

对于模型组合上，我们分别对子模型采用T-SNE可视化来直观化效果。将NLP-Feature和TF-IDF得到的62维特征，导入到t-SNE降成2维可视化。得到的结果如下图所示：

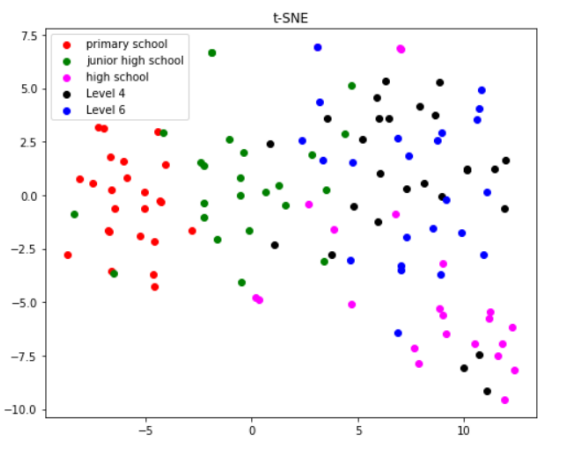


图 9 特征降维可视化散点图

由图可知，小学、初中、高中文本的聚集程度较高，进行后续分类后的准确度应会较高；而四级文本与六级文本的交叉度较高，较难进行分类。

于是，我们进一步将分类后的结果可视化绘制出柱状图

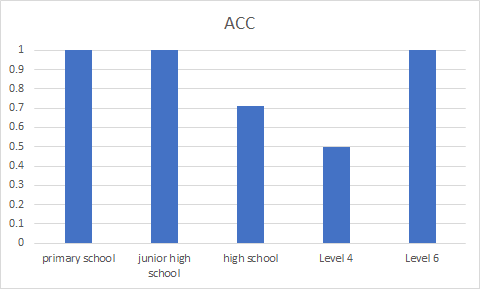


图 10 分类结果可视化柱状图

由图我们发现，四级和高中的文本分类准确度较低，其中四级文本的准确度远低于其他几项，所以我们将四级文本的分类情况做了进一步的可视化，得到下图：

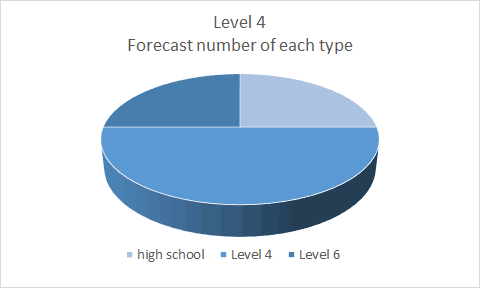


图 11四级文本分类情况饼图

我们发现，四级文本部分的测试结果有半数被归到了高中和六级文本的分类中。

经讨论分析，并询问了英语专业的教师，得知我们的数据集分类不够合理。四级作为面向在校专科生、本科生毕业要求的考试，其难度是从高中水平到大学水平的过度，易读性与高中文本、六级文本均有所交叉，从而导致模型的分类准确度有所下降。

我们需要进一步改进文本的分类，以对模型的准确性和稳定性进行更好的检验。

# 7. 收获和建议

备注：每个人单独写自己的收获和建议。例如：学到了什么？遇到了什么困难？如何解决？（黄色填充部分，正式文档需删掉）

应宇杰：

通过完成这次数据挖掘的大作业，我学会了文本分类方法和分类器的选择，通过精确性、召回率、特异性、曲线下面积、F1分数和准确度六项来评价模型是否适合这个数据集，通过二次细分参数评估得到最适合参数。通过NLP特征提取和数据降维，但是在测试集发现识别成功率低，通过SNE将数据降维到2维可视化发现四六级和高中英语相似度较大无法区分，通过增加训练数据集可以一定程度增加判断成功率。

# 项目源代码（每个部分写明是哪位同学完成）

算法寻找设计实现由蒋敏超同学实现，数据清洗部分由应宇杰完成

**from** sklearn.feature\_extraction.text **import** CountVectorizer, TfidfVectorizer, TfidfTransformer

**import** pandas **as** pd

**from** sklearn.metrics **import** accuracy\_score

**from** skfeature.function.sparse\_learning\_based **import** ls\_l21

**from** sklearn **import** svm

**from** skfeature.utility.sparse\_learning **import** construct\_label\_matrix\_pan, feature\_ranking

**import** pandas **as** pd

**from** testmetric **import** **\***

**from** lightgbm **import** LGBMClassifier

**import** numpy **as** np

**from** cls **import** randomforest, bayes, logisticregression, decisiontree, adaboost, knn

**from** cls **import** svmOpt

**from** sklearn.metrics **import** roc\_auc\_score

**import** matplotlib.pyplot **as** plt

**from** sklearn.manifold **import** TSNE *# t-sen|*

**from** sklearn.feature\_selection **import** VarianceThreshold

**from** sklearn.model\_selection **import** cross\_val\_score

**from** sklearn.ensemble **import** RandomForestRegressor

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.decomposition **import** PCA

**from** sklearn.svm **import** SVC

**from** sklearn.metrics **import** confusion\_matrix

**from** sklearn.model\_selection **import** KFold

**from** sklearn.svm **import** SVC

**from** sklearn.model\_selection **import** train\_test\_split

**import** string**,** textblob

**from** sklearn.decomposition **import** TruncatedSVD

**from** sklearn.preprocessing **import** MinMaxScaler

**import** nltk**,**re

txtlist **=** []

lables **=** []

In [3]:

**for** i **in** range(5):

**for** j **in** range(0,30):

lables**.**append(i)

filename **=** '60'**+**'/'**+**str(i)**+**'-'**+**str(j**+**1)**+**'.txt'

*# print(j)*

data **=** open(filename,encoding**=**'utf-8')**.**read()

data **=** data**.**replace('\n' , '')

data **=** data**.**replace('\'' , '')

data **=** data**.**replace('`' , '')

txtlist**.**append(data)

In [113]:

len(txtlist)

txtlist **=** []

lables **=** []

In [3]:

**for** i **in** range(5):

**for** j **in** range(0,30):

lables**.**append(i)

filename **=** '60'**+**'/'**+**str(i)**+**'-'**+**str(j**+**1)**+**'.txt'

*# print(j)*

data **=** open(filename,encoding**=**'utf-8')**.**read()

data **=** data**.**replace('\n' , '')

data **=** data**.**replace('\'' , '')

data **=** data**.**replace('`' , '')

txtlist**.**append(data)

In [113]:

len(txtlist)

X\_train,X\_test, y\_train, y\_test **=**train\_test\_split(trainDF["text\_clean"],trainDF['label'] ,test\_size**=**0.2, random\_state**=**42)

trainDF['char\_count'] **=** trainDF['text']**.**apply(len)

trainDF['word\_count'] **=** trainDF['text']**.**apply(**lambda** x: len(x**.**split()))

trainDF['word\_density'] **=** trainDF['char\_count'] **/** (trainDF['word\_count']**+**1)

trainDF['punctuation\_count'] **=** trainDF['text']**.**apply(**lambda** x: len(""**.**join(\_ **for** \_ **in** x **if** \_ **in** string**.**punctuation)))

trainDF['stopwords']**=**trainDF['text']**.**apply(**lambda** sen:len([x **for** x **in** sen**.**split() **if** x **in** lst\_stopwords])) *#停用词数目*

trainDF['sentence\_count'] **=** trainDF['text']**.**apply(**lambda** x: len(x**.**split('.')))

trainDF['avg\_sentence\_lenght'] **=** trainDF['word\_count'] **/** trainDF['sentence\_count']

*# trainDF['title\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.istitle()]))*

*# trainDF['upper\_case\_word\_count'] = trainDF['text'].apply(lambda x: len([wrd for wrd in x.split() if wrd.isupper()]))*

In [13]:

pos\_family **=** {

'noun' : ['NN','NNS','NNP','NNPS'],

'pron' : ['PRP','PRP$','WP','WP$'],

'verb' : ['VB','VBD','VBG','VBN','VBP','VBZ'],

'adj' : ['JJ','JJR','JJS'],

'adv' : ['RB','RBR','RBS','WRB']

}

*#检查和获得特定句子中的单词的词性标签数量*

**def** check\_pos\_tag(x, flag):

cnt **=** 0

**try**:

wiki **=** textblob**.**TextBlob(x)

**for** tup **in** wiki**.**tags:

ppo **=** list(tup)[1]

**if** ppo **in** pos\_family[flag]:

cnt **+=** 1

**except**:

**pass**

**return** cnt

trainDF['noun\_count'] **=** trainDF['text']**.**apply(**lambda** x: check\_pos\_tag(x, 'noun'))

trainDF['verb\_count'] **=** trainDF['text']**.**apply(**lambda** x: check\_pos\_tag(x, 'verb'))

trainDF['adj\_count'] **=** trainDF['text']**.**apply(**lambda** x: check\_pos\_tag(x, 'adj'))

trainDF['adv\_count'] **=** trainDF['text']**.**apply(**lambda** x: check\_pos\_tag(x, 'adv'))

trainDF['pron\_count'] **=** trainDF['text']**.**apply(**lambda** x: check\_pos\_tag(x, 'pron'))

In [14]:

d **=** trainDF**.**iloc[:,3:]

train\_index **=** X\_train**.**index**.**tolist()

test\_index **=** X\_test**.**index**.**tolist()

In [17]:

train\_nlp **=** d**.**iloc[train\_index]

In [18]:

test\_nlp **=** d**.**iloc[test\_index]

In [19]:

mm**=**MinMaxScaler()

train\_nlp\_feature**=**mm**.**fit\_transform(train\_nlp**.**values)

In [116]:

train\_nlp\_feature**.**shape

Out[116]:

(120, 12)

In [20]:

test\_nlp\_feature**=**mm**.**transform(test\_nlp**.**values)

clean\_txtlist **=** trainDF["text\_clean"]

In [22]:

vectorizer **=** CountVectorizer()

*# count = vectorizer.fit\_transform(clean\_txtlist)*

count **=** vectorizer**.**fit\_transform(X\_train)

In [23]:

count

Out[23]:

<120x6648 sparse matrix of type '<class 'numpy.int64'>'

with 15612 stored elements in Compressed Sparse Row format>

In [24]:

vectorizer**.**get\_feature\_names()

vectorizer**.**vocabulary\_

count**.**toarray()

ransformer **=** TfidfTransformer()

tfidf\_matrix **=** transformer**.**fit\_transform(count)

print(tfidf\_matrix**.**toarray()**.**shape)

(120, 6648)

In [26]:

test\_count **=** vectorizer**.**transform(X\_test)

In [27]:

test\_tfidf\_matrix **=** transformer**.**transform(test\_count)

In [28]:

svd **=** TruncatedSVD(n\_components**=**50, n\_iter**=**100)

In [29]:

svd\_topic\_vectors **=** svd**.**fit\_transform(tfidf\_matrix**.**toarray())

In [30]:

test\_svd\_topic\_vectors **=** svd**.**transform(test\_tfidf\_matrix**.**toarray())

In [31]:

svd\_topic\_vectors**.**shape

Out[31]:

(120, 50)

In [32]:

test\_svd\_topic\_vectors**.**shape

train\_x, valid\_x, train\_y, valid\_y **=** train\_test\_split(trainDF['text'], trainDF['label'])

*#词语级tf-idf*

tfidf\_vect **=** TfidfVectorizer(analyzer**=**'word', token\_pattern**=**r'\w{1,}', max\_features**=**5000)

tfidf\_vect**.**fit(trainDF['text'])

xtrain\_tfidf **=** tfidf\_vect**.**transform(train\_x)

xvalid\_tfidf **=** tfidf\_vect**.**transform(valid\_x)

svd **=** TruncatedSVD(n\_components**=**20, n\_iter**=**100)

svd\_xtrain\_vectors **=** svd**.**fit\_transform(xtrain\_tfidf**.**toarray())

svd\_xvalid\_vectors **=** svd**.**fit\_transform(xvalid\_tfidf**.**toarray())

*# ngram 级tf-idf*

tfidf\_vect\_ngram **=** TfidfVectorizer(analyzer**=**'word', token\_pattern**=**r'\w{1,}', ngram\_range**=**(2,3), max\_features**=**5000)

tfidf\_vect\_ngram**.**fit(trainDF['text'])

xtrain\_tfidf\_ngram **=** tfidf\_vect\_ngram**.**transform(train\_x)

xvalid\_tfidf\_ngram **=** tfidf\_vect\_ngram**.**transform(valid\_x)

svd\_ngram **=** TruncatedSVD(n\_components**=**20, n\_iter**=**100)

svd\_ngram\_xtrain\_vectors **=** svd**.**fit\_transform(xtrain\_tfidf**.**toarray())

svd\_ngram\_xvalid\_vectors **=** svd**.**fit\_transform(xvalid\_tfidf**.**toarray())

*#词性级tf-idf*

tfidf\_vect\_ngram\_chars **=** TfidfVectorizer(analyzer**=**'char', token\_pattern**=**r'\w{1,}', ngram\_range**=**(2,3), max\_features**=**5000)

tfidf\_vect\_ngram\_chars**.**fit(trainDF['text'])

xtrain\_tfidf\_ngram\_chars **=** tfidf\_vect\_ngram\_chars**.**transform(train\_x)

xvalid\_tfidf\_ngram\_chars **=** tfidf\_vect\_ngram\_chars**.**transform(valid\_x)

svd\_ngram\_chars **=** TruncatedSVD(n\_components**=**20, n\_iter**=**100)

svd\_ngram\_chars\_xtrain\_vectors **=** svd**.**fit\_transform(xtrain\_tfidf**.**toarray())

svd\_ngram\_chars\_xvalid\_vectors **=** svd**.**fit\_transform(xvalid\_tfidf**.**toarray())

a **=** pd**.**DataFrame(svd\_xtrain\_vectors)

b **=** pd**.**DataFrame(svd\_ngram\_xtrain\_vectors)

c **=** pd**.**DataFrame(svd\_ngram\_chars\_xtrain\_vectors)

d **=** pd**.**concat([a,b,c],axis**=**1)

e **=** pd**.**DataFrame(svd\_xvalid\_vectors)

f **=** pd**.**DataFrame(svd\_ngram\_xvalid\_vectors)

g **=** pd**.**DataFrame(svd\_ngram\_chars\_xvalid\_vectors)

h **=** pd**.**concat([e,f,g],axis**=**1)

# 拼接数据

a **=** pd**.**DataFrame(svd\_topic\_vectors)

b **=** pd**.**DataFrame(train\_nlp\_feature)

e **=** pd**.**concat([a,b],axis**=**1)

data **=** [e**.**values, np**.**array(y\_train), f**.**values,np**.**array(y\_test)]

In [84]:

model\_SVC **=** SVC(C**=**100,gamma**=**0.1,probability**=True**)

model\_SVC**.**fit(data[0], data[1])

indemetric **=** svmOpt(data[2], data[3], model\_SVC)

SvmOpt Classifier

In [123]:

data[3]

Out[123]:

(30,)

In [151]:

model\_SVC **=** SVC(C**=**100,gamma**=**0.1,probability**=True**)

model\_SVC**.**fit(data[0], data[1])

metric **=** pd**.**DataFrame(indemetric)

col **=** ['acc', 'auc', 'sen', 'spen', 'f1\_score','Precision']

piece **=** metric**.**loc[0, col] *# 就算出来的准确度这些值，这一行，取这些列*

In [172]:

y\_pred **=** model\_SVC**.**predict(data[2])

y\_score **=** model\_SVC**.**predict\_proba(data[2])

In [169]:

**from** sklearn.metrics **import** confusion\_matrix

In [173]:

confusion\_matrix(data[3],y\_pred)

model\_SVC **=** SVC(C**=**100,gamma**=**0.1,probability**=True**)

model\_SVC**.**fit(data[0], data[1])

indemetric **=** svmOpt(data[2], data[3], model\_SVC)

random\_inde **=** randomforest(data[0], data[1], data[2], data[3])

bayes\_inde **=** bayes(data[0], data[1], data[2], data[3])

logisticregression\_inde **=** logisticregression(data[0], data[1], data[2], data[3])

adaboost\_inde **=** adaboost(data[0], data[1], data[2], data[3])

knn\_inde **=** knn(data[0], data[1], data[2], data[3])

*# lightgbm\_inde = lightgbm(data[0], data[1], data[2], data[3])*

decisiontree\_inde **=** decisiontree(data[0], data[1], data[2], data[3])

metric **=** pd**.**DataFrame(indemetric)

random\_inde **=** pd**.**DataFrame(random\_inde)

bayes\_inde **=** pd**.**DataFrame(bayes\_inde)

logisticregression\_inde **=** pd**.**DataFrame(logisticregression\_inde)

adaboost\_inde **=** pd**.**DataFrame(adaboost\_inde)

knn\_inde **=** pd**.**DataFrame(knn\_inde)

decisiontree\_inde **=** pd**.**DataFrame(decisiontree\_inde)

*# lightgbm\_inde = pd.DataFrame(lightgbm\_inde,index=[0])*

col **=** ['acc', 'auc', 'sen', 'spen', 'f1\_score','Precision']

piece **=** metric**.**loc[0, col] *# 就算出来的准确度这些值，这一行，取这些列*

random\_inde\_piece **=** random\_inde**.**loc[0, col]

bayes\_inde\_piece **=** bayes\_inde**.**loc[0, col]

logisticregression\_inde\_piece **=** logisticregression\_inde**.**loc[0, col]

adaboost\_inde\_piece **=** adaboost\_inde**.**loc[0, col]

knn\_inde\_piece **=** knn\_inde**.**loc[0, col]

decisiontree\_inde\_piece **=** decisiontree\_inde**.**loc[0, col]

*# lightgbm\_inde\_piece = lightgbm\_inde.loc[0, col]*

piece**.**name **=** 'Svm'

random\_inde\_piece**.**name **=** 'Randomforest'

bayes\_inde\_piece**.**name **=** 'Bayes'

logisticregression\_inde\_piece**.**name **=** 'logisticregression'

adaboost\_inde\_piece**.**name **=** 'Adaboost'

knn\_inde\_piece**.**name **=** 'Knn'

decisiontree\_inde\_piece**.**name **=** 'Decision\_tree'

*# lightgbm\_inde\_piece.name = "lightgbm" lightgbm\_inde\_piece*

outCome **=** pd**.**concat(

[piece, random\_inde\_piece, bayes\_inde\_piece, logisticregression\_inde\_piece, adaboost\_inde\_piece,

knn\_inde\_piece, decisiontree\_inde\_piece], axis**=**1)

zSVM **=** []

**def** find\_best\_SVM(c,gamma,traindata,trainlabel,testdata,testlabel):

model **=** SVC(C**=**c, gamma**=**gamma, probability**=True**)

model**.**fit(traindata,trainlabel)

prediction**=**model**.**predict(testdata)

acc**=**accuracy\_score(testlabel, prediction)

**return** acc

best\_par\_acc**=**0

c **=** [ 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 1000 ]

gamma **=** [ 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1, 10, 100, 1000]

**from** sklearn.model\_selection **import** cross\_val\_score

**for** C **in** c:

**for** Gamma **in** gamma:

accuracy**=**find\_best\_SVM(C,Gamma,e**.**values,np**.**array(y\_train),f**.**values, np**.**array(y\_test))

zSVM**.**append(accuracy)

print(C," ",Gamma," ",accuracy)

**if**(accuracy**>**best\_par\_acc):

best\_par\_acc**=**accuracy

best\_C**=**C

best\_gamma**=**Gamma

print("the highest acc is ",best\_par\_acc)

print("the best C is ",best\_C)

print("the best gamma is ",best\_gamma)

zSVM **=** []

**def** find\_best\_SVM(c,gamma,traindata,trainlabel,testdata,testlabel):

model **=** SVC(C**=**c, gamma**=**gamma, probability**=True**)

model**.**fit(traindata,trainlabel)

prediction**=**model**.**predict(testdata)

acc**=**accuracy\_score(testlabel, prediction)

**return** acc

best\_par\_acc**=**0

c **=** [ 75, 80, 85, 90, 95, 100, 105, 110, 115 ]

gamma **=** [ 0.02,0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18]

**from** sklearn.model\_selection **import** cross\_val\_score

**for** C **in** c:

**for** Gamma **in** gamma:

accuracy**=**find\_best\_SVM(C,Gamma,e**.**values,np**.**array(y\_train),f**.**values, np**.**array(y\_test))

zSVM**.**append(accuracy)

print(C," ",Gamma," ",accuracy)

**if**(accuracy**>**best\_par\_acc):

best\_par\_acc**=**accuracy

best\_C**=**C

best\_gamma**=**Gamma

print("the highest acc is ",best\_par\_acc)

print("the best C is ",best\_C)

print("the best gamma is ",best\_gamma)

tsne **=** TSNE(n\_components**=**2, random\_state**=**0)

X\_2d **=** tsne**.**fit\_transform( e**.**values)

plt**.**figure(figsize**=**(8, 6.5))

plt**.**title('t-SNE')

plt**.**scatter(X\_2d[a **==** 0, 0], X\_2d[a **==** 0, 1], c**=**'r', label**=**'primary school')

plt**.**scatter(X\_2d[a **==** 1, 0], X\_2d[a **==** 1, 1], c**=**'g', label**=**'junior high school')

plt**.**scatter(X\_2d[a **==** 2, 0], X\_2d[a **==** 2, 1], c**=**'magenta', label**=**'high school')

plt**.**scatter(X\_2d[a **==** 3, 0], X\_2d[a **==** 3, 1], c**=**'black', label**=**'Level 4')

plt**.**scatter(X\_2d[a **==** 4, 0], X\_2d[a **==** 4, 1], c**=**'blue', label**=**'Level 6')

plt**.**legend()

plt**.**show()